Projecting points onto planar parametric curves by local biarc approximation

Hai-Chuan Song, Xin Xu, Kan-Le Shi, Jun-Hai Yong

Projecting points onto planar parametric curves by local biarc approximation

Hai-Chuan Song[a,b,c,d], Xin Xu[a,b,c,d], Kan-Le Shi[a,b,c,e], Jun-Hai Yong[a,c,d]

[a] *School of Software, Tsinghua University, Beijing 100084, P. R. China*
[b] *Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China*
[c] *Key Laboratory for Information System Security, Ministry of Education of China, Beijing 100084, P. R. China*
[d] *Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, P. R. China*
[e] *INRIA, France*

Abstract

This paper proposes a geometric iteration algorithm for computing point projection and inversion on planar parametric curves based on local biarc approximation. The iteration begins with initial estimation of the projection of the prescribed test point. For each iteration, we construct a biarc that locally approximates a segment on the original curve starting from the current projective point. Then we compute the projective point for the next iteration, as well as the parameter corresponding to it, by projecting the test point onto this biarc. The iterative process terminates when the projective point satisfies the required precision. Examples demonstrate that our algorithm converges faster and is less dependent on the choice of the initial value compared to the traditional geometric iteration algorithms based on single-point approximation.

Keywords: point project, parametric curves, biarc interpolation, local approximation

1. Introduction

Projection of a test point on a curve or surface aims to find the closest point, as well as the corresponding parameter, on the curve or surface. Specially, when the test point lies on the curve or surface, the problem of point projection becomes the problem of point inversion. This operation has been extensively used in geometric processing algorithms such as surface intersection [1], interactive object selection and shape registration [2, 3, 4]. Moreover, it is a fundamental component of the algorithms of curve and surface projection as well [5, 6]. In this paper, we address the problem of point projection and point inversion on planar parametric curves. We provide a geometric iteration algorithm, which approximates a segment on the original curve by a biarc. Compared with traditional single-point approximation algorithms [7, 8, 4], our algorithm converges faster and is less dependent on the choice of the initial value.

1.1. Related work

The problem of point projection and inversion can be translated to solving the minimum distance equation $(Q-P)\times n = 0$, where P is the prescribed test point, Q is the point closest to P on the original curve or surface and n is the normal vector of the original curve or surface at Q. In most of the early work, Newton-Raphson method, which involves the first and second order derivatives, was used to solve this minimum distance equation and get the projective point [1, 8]. Piegl and Tiller [9] gave a detailed description on this method for point projection and inversion.

In order to achieve a good initial value, which is important for Newton-Raphson method to converge reliably, subdivision methods were introduced [10, 11, 12, 13, 14, 15, 16]. The key point of this kind of algorithms is to eliminate the curve segments or the surface patches which do not contain the nearest points. Ma and Hewitt [12] divided the NURBS surface into several Bézier patches and checked the relationship between the test point and the control point nets of these Bézier patches. However their elimination criterion may fail in some cases [17]. Johnson and Cohen utilized the tangent cone to search for the portions of the surface contain the projective points [13]. A more practical exclusion criterion based on Voronoi cell test was proposed in [16]. Chen et al. [15] improved their method using the clipping circle/sphere. By replacing the clipping circle/sphere with axis-aligned lines/planes, Oh et al. [14] reduced the computing time of [15]. Based on [14], they presented an algorithm for projecting continuously moving

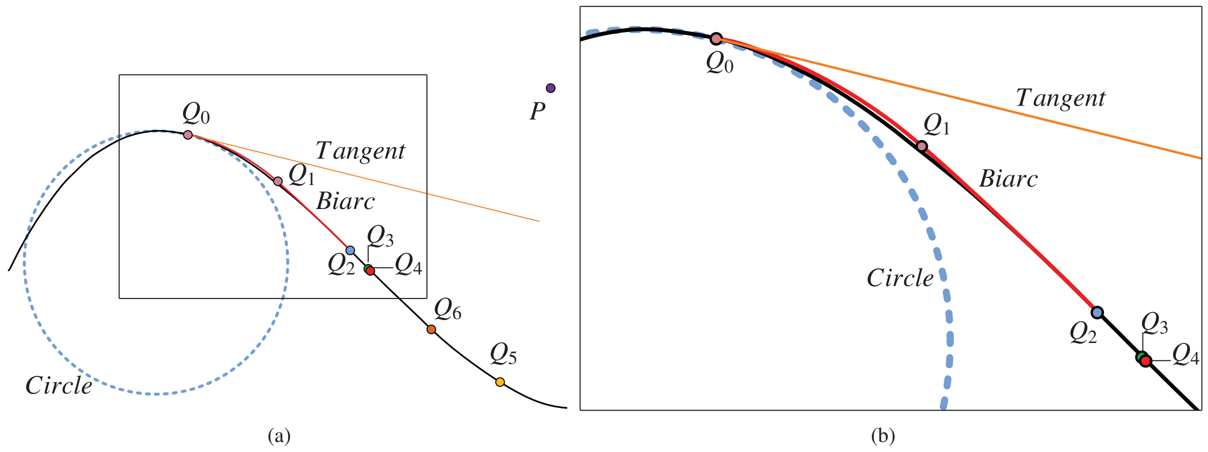

Figure 1: A comparison of the approximation precision of the first order algorithm [7, 8], the second order algorithm [4] and our algorithm: P is the test point, the black curve is the original curve, Q_0 is the initial point, orange point Q_6, yellow point Q_5, blue point Q_2 and red point Q_4 are projective points obtained by Newton-Raphson method [9], the first order algorithm [7, 8], the second order algorithm [4] and our algorithm after the first iteration, respectively. Q_3 is the exact closest point. (a) the whole view of the projection; (b) the zoom view of (a).

query points onto planar spiral curves [18]. Seong et al. [19] dealt with this problem in another way. By elevating the dimension of the problem, they transformed the point projection onto planar parametric curve into the intersection of an implicit surface and a straight line.

Besides algebraic methods (Newton-Raphson method), geometric methods were also proposed, which only involve the geometric information that is common to all possible parameterizations. Hoschek and Lasser [7], Hartmann [8] introduced the first order geometric iteration method. Hu and Wallner [4] proposed a second order geometric iteration method, in which they generated an osculating circle (a circle possessing the same curvature with the original curve at the osculating point) and projected the test point on it instead of the original curve or surface. Liu et al. [20] improved their method by replacing the circle of normal curvature with a second order osculating torus patch to the surface.

Note that the algebraic method and geometric methods generally converge (if can converge) to the local minimum projective point nearest to the initial value [13]. All the subdivision methods we introduced above [10, 11, 12, 13, 14, 15, 16] can generate the initial value near to the global minimum projective point. Therefore, generally, a complete point projection include two parts: 1. An algorithm to generate initial values; 2. An iteration algorithm to compute the precise projective point. In this paper, we mainly focus on the iteration algorithm in the second part. It means that we assume the initial value has been provided by some kind of initial value generating algorithm.

1.2. *Our contributions*

The main idea of geometric methods is to locally approximate the original curve by a special curve (first order algorithm uses tangent line, and second order algorithm uses osculating circle). The next projective point is estimated by projecting the test point onto the approximation curve instead of the original curve. It is shown in the evolution of the geometric methods that higher approximation precision generally means higher convergence speed and better stability. However, during each iteration step, the traditional geometric algorithms approximate the original curve only with curve derivatives computed at a single point. So the approximation region is limited around this point, the approximation precision will reduce when moving away from this point on the original curve.

In order to improve the convergence speed and stability of the point projection and inversion, we provide a geometric iteration algorithm based on local biarc approximation. Our method approximates the corresponding segment on the original curve by a biarc rather than only one point during each iteration. Our local biarc approximation has larger approximation region and higher approximation precision compared to traditional single-point approximation. According to the experimental results in Section 4, our algorithm converges faster and is more robust than the traditional geometric algorithms.

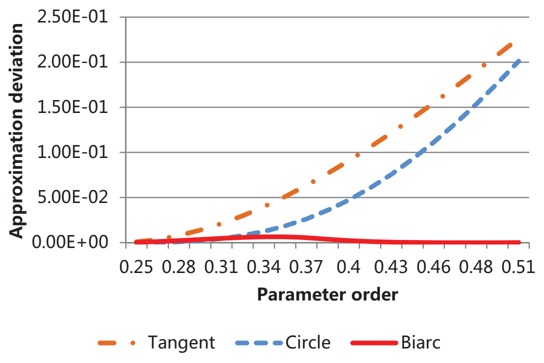

Figure 2: Approximation deviation comparison in Figure 1. Abscissa is the parameter of the moving point on the original curve. Ordinate is the distance from the moving point to approximation curve.

A brief comparison of the traditional geometric algorithms and our algorithm is shown in Figure 1. In this figure, the violet test point $P = (0.5, 0.5)$ is projected onto the black planar Bézier curve, the control points of which are $\{(-1, 0), (-0.5, 1), (0, 0), (0.5, -1), (1, 0)\}$. The parameter of the initial point Q_0 is 0.25. The orange point Q_6, yellow point Q_5, blue point Q_2 and red point Q_4 are projective points obtained by Newton-Raphson method [9], the first order algorithm [7, 8], the second order algorithm [4] and our algorithm after the first step iteration, respectively. Q_3 is the exact closest point, whose parameter is 0.51. The yellow line is the tangent line at Q_0 and is used to approximate the original curve and estimate the next projective point in the first order algorithm [7, 8]. The blue dashed curve is the osculating circle at Q_0 and is used to approximate the original curve and estimate the next projective point in the second order algorithm [4]. The red curve from Q_0 to Q_2 is the biarc used to approximate the curve segment from Q_0 to Q_2 on the original curve in our algorithm. Q_1 is the middle point of the biarc. Figure 2 compares the distances from the moving point on the original curve (from Q_0 to Q_3) to the three types of approximation curves: tangent, osculating circle, biarc. When moving away from Q_0, the approximation precisions of tangent and osculating circle drop significantly, while that of biarc seldom changes. This means compared to tangent and osculating circle, our biarc has a higher approximation precision and a larger approximation region. So the next projective point Q_4 is much closer to the exact projective point than other single-point methods.

The contributions of this paper are as follows:

1. We propose a point projection and inversion algorithm by local biarc approximation, which has a higher approximation precision and a larger approximation region compared to traditional single-point approximation algorithms.
2. We present a framework that adapts to any single-point approximation algorithm. If a single-point algorithm is integrated in our framework, the convergence speed and independence of the initial value of this single-point algorithm will be improved.

1.3. Outline of our algorithm

Given a test point P, a planar parametric curve $C(t)$ and the parameter value t_0 of the roughly estimated projective point Q_0, as illustrated in Figure 1, we need to compute the parameter of the exact projective point. Our algorithm framework can be described in summary as follows:

1. According to the initial projection parameter t_0, compute the interval width Δt using any step length strategy (we can use constant parameter increment, Newton-Raphson method [9], first order algorithm [7, 8], or second order algorithm [4]).
2. Compute the tangent vectors $C'(t_0)$ and $C'(t_0+\Delta t)$, respectively. Interpolate the boundary conditions $C(t_0)$, $C'(t_0)$ and $C(t_0+\Delta t)$, $C'(t_0+\Delta t)$ with a biarc $BA(s)$ (the red curve in Figure 1), which is used to approximate the curve segment from $C(t_0)$ to $C(t_0+\Delta t)$ on the original curve $C(t)$.
3. Project the test point P onto the biarc $BA(s)$ and compute a new estimated parameter of the projective point (Q_5 in Figure 1) on the original curve $C(t)$.
4. Use the new parameter as the initial value t_0 and repeat steps 1-3 until the corresponding projective point satisfies the precision requirement.

The rest paper is organized as follows. Section 2 presents the method for local curve approximation by biarc interpolation. In Section 3, the methods of point projection onto the biarc and parameter inversion to the original curve are described. The experimental results including the evaluation of performance data are given in Section 4. Finally, Section 5 concludes the paper.

2. Local biarc approximation of a curve segment

We approximate a curve segment on the original curve $C(t)$ by a biarc according to the initial projection parameter t_0, which consists of the following steps:

1. Compute the interval width Δt.
2. Compute the interpolation boundary conditions $C(t_0)$, $C'(t_0)$ and $C(t_0 + \Delta t)$, $C'(t_0 + \Delta t)$.

3. Interpolate the boundary conditions with a biarc.

We present the three steps in details in the following subsections, respectively.

2.1. *Compute the interval width*

The interval width Δt determines which curve segment on $C(t)$ is to be approximated. We select any step length strategy from following ones:

1. User-defined constant parametric increment, and
$$\Delta t = const\_t. \tag{1}$$

2. Newton-Raphson method [9], and
$$\Delta t = \frac{C'(t_0) \cdot (C(t_0) - P)}{C''(t_0) \cdot (C(t_0) - P) + |C'(t_0)|^2}. \tag{2}$$

3. First order algorithm [7, 8], and
$$\Delta t = \frac{C'(t_0) \cdot (Q - C(t_0))}{C'(t_0) \cdot C'(t_0)}, \tag{3}$$
where Q is the projective point of P on the tangent line at $C(t_0)$.

4. Second order algorithm [4], and
$$\Delta t = \frac{(Q - C(t_0)) \times C''(t_0)}{\kappa \|C'(t_0)\|^3}, \tag{4}$$
where Q is the projective point of P on the osculating circle at $C(t_0)$, κ is the curvature of $C(t_0)$, and we have $\kappa = (C'(t_0) \times C''(t_0))/\|C'(t_0)\|^3$.

Note that, except Strategy 1, all the other strategies based on single-point approximation can be used to compute point projection and inversion independently.

Our algorithm therefore provides a framework that adapts to any single-point approximation algorithm (use it to compute Δt). Moreover, according to our experiments, the integration converges faster and is less dependent on the choice of the initial value compared to the integrated original single-point algorithm alone.

According to our experience, considering the convergence speed and the stability, priority of the four strategies is $4 > 2 > 3 > 1$, where '>' means better (converge faster and less independent on the initial value) than. After we derive Δt, the interval is determined by $[t_0, t_0 + \Delta t]$.

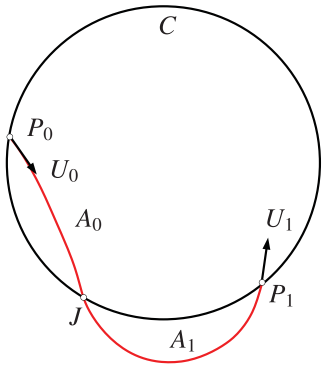

Figure 3: A biarc (red) and the joint circle (black) [21].

2.2. *Approximate the curve segment by biarc*

In this part, we approximate the corresponding curve segment $C(t_0) \sim C(t_0 + \Delta t)$ on the original curve by a biarc interpolation on the G^1 boundary data $C(t_0)$, $C'(t_0)$ and $C(t_0 + \Delta t)$, $C'(t_0 + \Delta t)$, where $C'(t)$ is the first order derivative of $C(t)$. Before that, we review the problem of biarc interpolation of G^1 boundary data. The definition of biarc interpolation given in [21] is as follows:

Definition 1. The two circular arcs A_0, A_1 are said to form a biarc interpolating given oriented G^1 data, represented by end points P_0, P_1 and unit tangent vectors U_0, U_1 (see Figure 3) if and only if the two circular arcs share one common end point J called joint and satisfy the following properties:

1. The arc A_0 has the end points P_0 and J, and U_0 is tangent to A_0 with orientation corresponding to a parametrization of A_0 from P_0 to J.
2. The arc A_1 has the end points J and P_1 and U_1 is tangent to A_1 with orientation corresponding to a parametrization of A_1 from J to P_1.
3. The two arcs have a common unit tangent vector at J, with orientation corresponding to a parametrization of A_0 from P_0 to J and of A_1 from J to P_1.

The biarc interpolation plays an important part in tool path generation in CNC (Computerized Numerical Control) [22, 23], and approximation of curves [23] and discrete data [24]. Constrained interpolation with biarc is also widely studied [21, 25].

The locus of all possible joint J is a circle (C in Figure 3) passing through P_0 and P_1 [21]. Various biarc interpolation schemes are distinguished by the choice of the joint J. Among the most important ones are the "equal chord" biarc and the "parallel tangent" biarc. The former one is constructed so that the two segments

P_0J and JP_1 have equal arc lengths on C, while the latter one ensures that the tangent at point J is parallel to segment P_0P_1. In this paper we choose the "equal chord" biarc as the interpolation tools, owning to its simplicity in implementation and better approximation precision according to our experiments.

Given boundary data $P_0 = C(t_0)$, $D_0 = C'(t_0)$ and $P_1 = C(t_0 + \Delta t)$, $D_1 = C'(t_0 + \Delta t)$, we generate the biarc $BA(s)$ in following steps:

1. Find the center of the joint circle C, by intersecting the bisectors of P_0P_1 and of $(P_0 + U_0)(P_1 + U_1)$, where U_0 and U_1 are the unit vectors of D_0 and D_1, respectively.
2. Find J by intersecting the bisector of the P_0P_1 and the minor arc on C bounded by P_0 and P_1.
3. Construct the unique arcs A_0, A_1 satisfying properties 1, 2 of Definition 1.
4. Generate a piecewise parametric biarc $BA(s)$, where $s \in [0, 1]$. It follows that $BA(s) = A_0$ when $s \in [0, s_J]$, and $BA(s) = A_1$ when $s \in [s_J, 1]$, where $s_J = ArcLength(A_0)/(ArcLength(A_0) + ArcLength(A_1))$.

An example is shown in Figure 1, where we use the second order algorithm to generate the approximation interval. In this figure, the biarc is much more closer to the original curve $C(t)$ compared to the approximation geometries of the first and the second order algorithms. It means that its approximation precision is higher than the first and the second order algorithms.

The reasons why we choose biarc as our approximation curve are as follows:

1. The construction of the biarc is simple, and can be done in constant time.
2. The interpolating biarc is G^1 osculating with the original curve. The approximation order of biarc is 3 [21]. We therefore can obtain a good local approximation of the corresponding curve segment.
3. The point projection on biarc can be computed by simply projecting the test point onto the two circles which A_0 and A_1 are embed in, respectively.

3. Point projection on the biarc and parameter inversion

3.1. Point projection on the biarc

As shown in Figure 4, the point projection on biarc $BA(s)$ can be computed by simply projecting the test point onto the two circles which A_0 and A_1 are embed in, respectively.

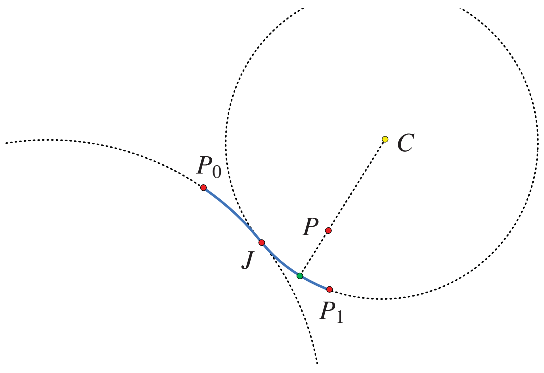

Figure 4: Point projection on the biarc.

There are four projective points on the two circles. For the projective points we obtained, we only choose one valid projective point by the following method:

1. If there is only one projective point in the parametric domain $[0, 1]$, this point is chosen as the valid projective point.
2. If there are more than one projective point in the parametric domain $[0, 1]$, we choose the projective point, which is closest to the test point P, from the projective points in the parametric domain $[0, 1]$ as the valid projective point.
3. If there is no projective point in the parametric domain $[0, 1]$, we choose the projective point, whose parameter is closest to the boundary of the parametric domain $[0, 1]$, as the valid projective point.

The parameter of the valid projective point is recorded in variable *param_biarc*.

3.2. Estimate the next projective point

After we derived the projective parameter *param_biarc* on the biarc, we refine Δt with it. Recall that the curve segment $BA(0) \sim BA(1)$ on the biarc is corresponding with the curve segment $C(t_0) \sim C(t_0 + \Delta t)$ on the original curve. So Δt can be refined by $\Delta t = param\_biarc \times \Delta t$ (note that, for extreme high-order Bézier curves, this linear parameter interpolation may be unstable). Then the parameter of the next projective point is estimated by $t_0 = t_0 + \Delta t$, which will be used for the next iteration.

We apply the convergence criteria provided by Piegl and Tiller [9], which are

$$|(t_{i+1} - t_1)C'(t_i)| \le \varepsilon_1. \tag{5}$$

$$|C(t_i) - P| \le \varepsilon_1. \tag{6}$$

$$\frac{|C'(t_i) \cdot (C(t_i) - P)|}{|C'(t_i)||C(t_i) - P|} \le \varepsilon_2. \tag{7}$$

Table 1: Convergence comparisons for Example 1.

$P_1 = (381, 252)$, $t_0 = 0.75$							
Step	1	2	3	4	5	6	CPU time (ms)
Δt_{NRA}	2.01×10^{-02}	-4.38×10^{-04}	-2.81×10^{-07}	0.00	t=0.769514		1.15×10^{-02}
Δt_{FOA}	3.24×10^{-02}	-2.38×10^{-02}	1.88×10^{-02}	-1.43×10^{-02}	1.13×10^{-02}	-8.75×10^{-02}	1.54×10^{-02}
Δt_{SOA}	2.07×10^{-02}	-1.21×10^{-03}	-4.13×10^{-06}	0.00	t=0.769514		2.46×10^{-03}
Δt_{NRIBA}	1.95×10^{-02}	3.02×10^{-06}	0.00	t=0.769514			2.11×10^{-03}
Δt_{FOIBA}	1.97×10^{-02}	-1.89×10^{-04}	0.00	t=0.769514			2.21×10^{-03}
Δt_{SOIBA}	1.95×10^{-02}	6.59×10^{-06}	0.00	t=0.769514			2.96×10^{-03}
$P_2 = (332, 200)$, $t_0 = 0.5$							
Step	1	2	3	4	5	6	CPU time (ms)
Δt_{NRA}	1.53×10^{-01}	-2.87×10^{-02}	-2.29×10^{-03}	-2.05×10^{-05}	0.00	t=0.6223419	1.65×10^{-02}
Δt_{FOA}	8.53×10^{-02}	2.86×10^{-02}	6.88×10^{-03}	1.29×10^{-03}	2.25×10^{-04}	3.86×10^{-05}	3.46×10^{-02}
Δt_{SOA}	1.21×10^{-01}	1.24×10^{-03}	-3.76×10^{-06}	0.00	t=0.6223419		2.98×10^{-03}
Δt_{NRIBA}	1.19×10^{-01}	3.84×10^{-03}	3.08×10^{-07}	0.00	t=0.6223419		2.41×10^{-03}
Δt_{FOIBA}	1.27×10^{-01}	-4.33×10^{-03}	-4.81×10^{-06}	0.00	t=0.6223419		2.89×10^{-03}
Δt_{SOIBA}	1.23×10^{-01}	-2.02×10^{-04}	0.00	t=0.6223419			2.96×10^{-03}

t_i is the parameter obtained at the ith iteration, and $\varepsilon_1, \varepsilon_2$ are two zero tolerances of Euclidean distance and cosine. The iteration is halted if any of the three conditions above is satisfied.

4. Examples and comparisons

We present five examples for point projection, and make comparisons with NRA (Newton-Raphson method [9]), FOA (first order algorithm [7, 8]), SOA (second order algorithm [4]), NRIBA (Newton-Raphson-integrated biarc algorithm: our algorithm whose Δt is generated by Newton-Raphson algorithm [9] as introduced in subsection 2.1), FOIBA (first-order-integrated biarc algorithm: our algorithm whose Δt is generated by first order algorithm [7, 8] as introduced in subsection 2.1), SOIBA (second-order-integrated biarc algorithm: our algorithm whose Δt is generated by second order algorithm [4] as introduced in subsection 2.1). All the experiments are implemented with Intel Core i5 CPU 3.0 GHz, 8G Memory. In all of our experiments $\varepsilon_1, \varepsilon_2$ are both the convergence tolerances introduced in subsection 3.2.

There are three main criteria to evaluate point projection iteration methods.

1. *Correctness*. If the distance between the computed projective point and the exact closest point satisfies a given precision, it is treated as a correct solution.
2. *Speed of convergence*. We measure the convergence speed by two kinds of experimental data: the number of iterations and the CPU time. We record the average and the worst numbers of iterations in each computation.
3. *Independence on the initial value*. The initial value has a significant impact on the correctness of the Newton-like iteration algorithms. Although there were lots of methods finding initial value (as described in subsection 1.1), no method can claim that its initial value is good enough to make the iteration always converge. So if the iteration method is less dependent on the initial value, the method will be more robust.

   In the following examples, all the initial values are set by hand (except Example 1 which uses the same initial value in [4] to compare with its second order algorithm) in order to test the performance of different iteration methods. In practice, we recommend to estimate the initial value with the method introduced in [15].

Example 1. We first test Example 2 of [4] (see Figure 5), where two test points $P_1 = (381, 252)$ and $P_2 = (332, 200)$ are projected onto a cubic B-spline curve $C(t)$ with initial parameter 0.75 and 0.5, respectively. In this example, we set $\varepsilon_1 = \varepsilon_2 = 10^{-6}$, which is the same with [4].

In Table 1, we compare the convergence of the six algorithms. If any method can converge within 10 iterations, we will provide the convergence parameter in this table. As shown in this table, our integrated algorithms (NRIBA, FOIBA and SOIBA) converge with

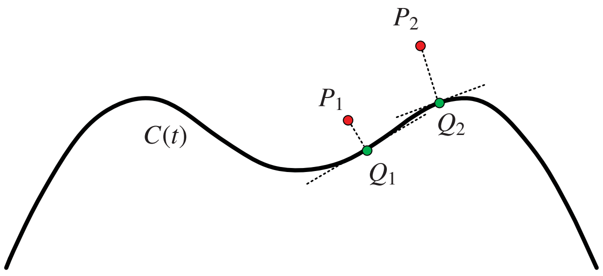

Figure 5: Illustration of Example 1: Q_1 and Q_2 are the exact projective points of $P_1 = (381, 252)$ and $P_2 = (332, 200)$. The control points and knot vector of $C(t)$ are {(100, 100), (140, 196), (200, 240), (260, 164), (340, 164), (400, 240), (460, 196), (500, 100)} and (0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1), respectively.

Table 2: Comparisons of the number of iterations for Example 1, where $P_1 = (381, 252)$.

Methods	Worst	Best	Average
NRA	12	3	7.92
FOA	68	48	55.08
SOA	8	4	5.69
NRIBA	6	2	3.98
FOIBA	6	3	4.55
SOIBA	5	2	3.84

less iterations than the corresponding single-point algorithms (NRA, FOA and SOA). The processing times of SOA, NRIBA, FOIBA and SOIBA are comparable, and are less than those of NRA and FOA.

In Table 2, we compare the robustness of the six algorithms with respect to the choice of the initial parameter t_0. We use P_1 and $C(t)$ in Example 1, and choose 101 different t_0 from $0, 0.01, 0.02, ..., 1$. Table 2 shows the number of iterations to converge correctly for the six algorithms. As shown in this table, our integrated algorithms averagely converge with less iterations than the corresponding single-point algorithms.

In Table 3, we compare the convergence of the six algorithms under different tolerances. As shown in this table, when we decrease the tolerance, the numbers of iterations of FOA, SOA, FOIBA increase; however NRA, NRIBA and SOIBA remain. NRA performs better than FOA and SOA, because the initial value is not too far from the exact projective point. However, NRIBA converges much faster than NRA. FOIBA converges much faster than FOA. In the third tolerance case, the number of iteration of FOA jumped significantly from 66 to 91. Influenced by FOA, the number of iteration of FOIBA jumped from 5 to 18, which is still relatively acceptable. SOIBA converges faster than SOA, and the gap is getting larger as we decrease the tolerance.

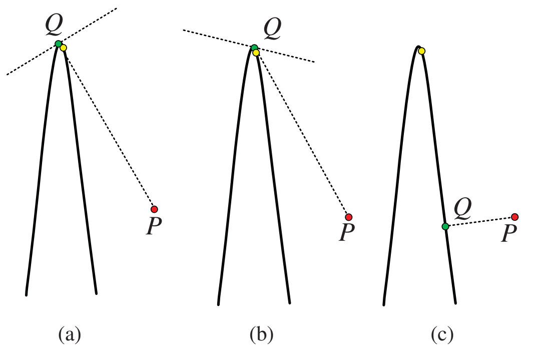

Figure 6: Illustration of Example 2. The yellow point is the initial projection. The control points of the Bézier curve are $\{(0, 0), (110, 1000), (90, 1000), (200, 0)\}$: (a) projection result Q of NRA and its tangent; (b) projection result Q of SOA and its tangent; (c) projection result Q of SOIBA.

Example 2. We project point $P = (381, 252)$ onto a cubic Bézier curve, and we set $t_0 = 0.53$ (see Figure 6). In this example, we set $\varepsilon_1 = \varepsilon_2 = 10^{-6}$. Table 4 shows the iteration steps of the six algorithms.

NRA converges to the local projective point $C(t = 0.487)$ in 4 steps, which is not the global nearest projective point (see Figure 6 (a)). However NRIBA converges to the correct global nearest projective point $C(t = 0.916)$ in 5 steps. It means that our integrated algorithm is less independent on the initial value, and is more likely to converge to the global nearest projective point.

FOA and FOIBA both converge to the correct projective point $C(t = 0.916)$ in 5 steps. As shown in Table 4, the first parametric increment of FOA is 2.49. This makes the parameter equal to 3.02 and run out of the parametric domain of the curve ($0 \sim 1$), which is unacceptable in practice. In our implementation, we patch FOA by drawing the parameter back to the nearest parametric domain boundary 1, and the iteration continues. However, FOIBA always iterate within the parametric domain of the curve, and converges to the correct projective point $C(t = 0.916)$ in 5 steps. It means that, our integrated algorithm is more stable.

SOA converges to the wrong projective point $C(t = 0.513)$ (neither the closest point nor the orthogonal projective point) in 6 steps, which is still close to the initial value (see Figure 6 (b)). However SOIBA converges to the correct projective point $C(t = 0.916)$ in 5 steps (see Figure 6 (c)). This is because $(Q - C(t_0))$ is nearly parallel with $C''(t_0)$ in Equation (4), leading to $\Delta t \approx 0$, and the iteration can hardly move away from $C(t_0)$ for SOA

Table 3: Comparisons of number of iterations / CPU time (ms) under different tolerances for Example 1, where $P_1 = (381, 252)$, $t_0 = 0.75$.

Tolerance ($\varepsilon_1 = \varepsilon_2$)	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
Δt_{NRA}	$5 / 1.15\times10^{-02}$	$5 / 1.17\times10^{-02}$	$5 / 1.11\times10^{-02}$	$5 / 1.21\times10^{-02}$	$5 / 1.22\times10^{-02}$
Δt_{FOA}	$57 / 1.54\times10^{-02}$	$66 / 1.95\times10^{-02}$	$91 / 2.96\times10^{-02}$	$101 / 3.43\times10^{-02}$	$112 / 3.64\times10^{-02}$
Δt_{SOA}	$5 / 2.46\times10^{-03}$	$5 / 2.97\times10^{-03}$	$11 / 6.48\times10^{-03}$	$11 / 6.51\times10^{-03}$	$11 / 6.50\times10^{-03}$
Δt_{NRIBA}	$4 / 2.11\times10^{-03}$	$4 / 2.14\times10^{-03}$	$4 / 2.12\times10^{-03}$	$4 / 2.21\times10^{-03}$	$4 / 2.23\times10^{-03}$
Δt_{FOIBA}	$4 / 2.21\times10^{-03}$	$5 / 2.35\times10^{-03}$	$18 / 1.50\times10^{-02}$	$18 / 1.51\times10^{-02}$	$18 / 1.50\times10^{-02}$
Δt_{SOIBA}	$4 / 2.96\times10^{-03}$	$4 / 3.06\times10^{-03}$	$4 / 4.06\times10^{-03}$	$4 / 4.07\times10^{-03}$	$4 / 4.06\times10^{-03}$

Table 4: Convergence comparisons for Example 2, where $P = (381, 252)$.

Step	1	2	3	4	5	6	7
Δt_{NRA}	-4.32×10^{-02}	3.98×10^{-04}	-4.98×10^{-08}	0.00	t=0.4872014		
Δt_{FOA}	2.49	-7.64×10^{-02}	-7.05×10^{-03}	-6.12×10^{-05}	0.00	t=0.9164463	
Δt_{SOA}	-3.18×10^{-02}	1.30×10^{-02}	1.36×10^{-03}	4.71×10^{-05}	-1.08×10^{-07}	0.00	t=0.5126524
Δt_{NRIBA}	8.78×10^{-02}	2.14×10^{-01}	8.44×10^{-02}	-9.45×10^{-05}	0.00	t=0.9164463	
Δt_{FOIBA}	3.11×10^{-01}	7.53×10^{-02}	8.64×10^{-04}	-8.98×10^{-07}	0.00	t=0.9164463	
Δt_{SOIBA}	1.16×10^{-01}	1.85×10^{-01}	8.64×10^{-02}	-3.90×10^{-04}	0.00	t=0.9164463	

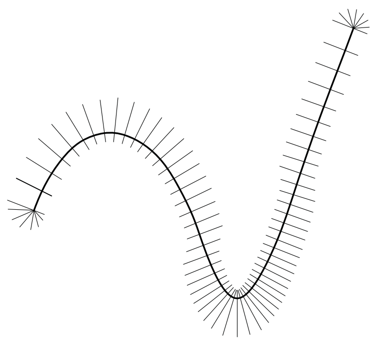

Figure 7: Illustration of Example 3: point projections on a smooth curve.

Table 5: Statistic data for Example 3.

Methods	Correct solutions	Worst iterations	Average iterations	CPU time (ms)
NRA	118	35	4.38	1.47
FOA	91	179	18.10	2.27
SOA	131	7	4.25	1.29
NRIBA	131	11	4.03	0.44
FOIBA	127	37	5.27	1.38
SOIBA	134	5	3.23	0.37

(see the first steps of SOA in Table 4). This case always occurs at the special point whose curvature is relatively much bigger than its neighboring region, leading to a very small osculating circle, the approximation region of which is small. With the help of our local biarc approximation in SOIBA, the approximation region is enlarged, and the iteration can "jump" away from the special point (see the first two steps of SOIBA in Table 4), and converges to the correct projective point even with the "bad" initial value. In this example, all the single-point algorithms fail to some extent, while our integrated algorithms all converge to the correct projective point.

Example 3. We project 134 points on a spurious offset of a smooth curve onto the curve itself (see Figure 7), using the six algorithms, respectively. In this example, we set $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, and the average initial value error is 2.27×10^{-02}. The statistic data of the projection is shown in Table 5. Note that the average iterations only record the correct projections, and the CPU time records all the projections (including both correct and incorrect projections).

Experimental results show that, our SOIBA finds all correct solutions, while the successful ratio of SOA is 97.8%, even if there is no special points mentioned in Example 2 (because all initial values are not too far from the exact projective points and there is no sharp features). The successful ratios of other algorithms are NRA: 88.1%, FOA: 67.9%, NRIBA: 97.8%, FOIBA: 94.8%, respectively. The average number of iterations

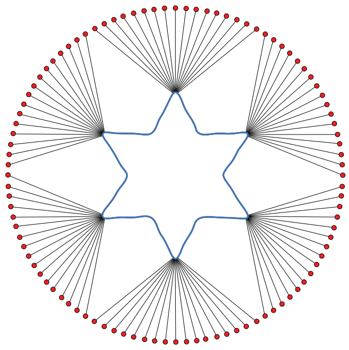
Figure 8: Illustration of Example 4: point projections on a star with sharp features.

Table 6: Statistic data for Example 4.

Methods	Correct solutions	Worst iterations	Average iterations	CPU time (ms)
NRA	62	8	6.77	5.35
FOA	0	-	-	-
SOA	67	176	19.11	7.66
NRIBA	100	5	4.7	0.24
FOIBA	79	43	5.98	4.85
SOIBA	100	5	4.04	0.43

Table 7: Statistic data for Example 5.

Methods	Correct solutions	Worst iterations	Average iterations	CPU time (ms)
NRA	237	8	4.77	3.03
FOA	75	167	11.23	5.38
SOA	256	9	4.72	0.782
NRIBA	251	6	4.38	1.41
FOIBA	226	30	5.98	0.79
SOIBA	266	6	3.63	0.72

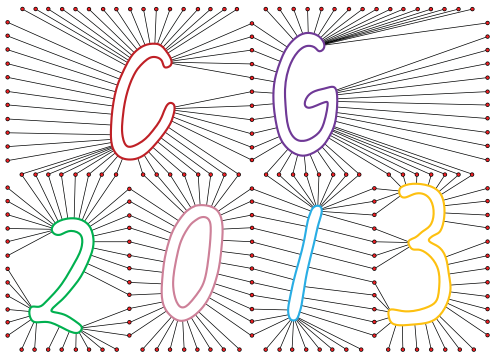
Figure 9: Illustration of Example 5: point projections on font characters.

and the CPU time of our integrated algorithms are also less than the corresponding single-point algorithms.

Example 4. We project 100 points on a bounding circle (loose bounding) onto a hexagonal star (designed by hand, not an exact hexagonal star), which has sharp features (corresponding to the special points mentioned in Example 2) on its six angles (see Figure 8), using the six algorithms, respectively. In this example, we set $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, and the average initial value error is 5.48×10^{-03}. The statistic data of the projection is shown in Table 6. Note that the average iterations only record the correct projections, and the CPU time records all the projections (including both correct and incorrect projections).

Experimental results show that, our SOIBA and NRIBA find all correct solutions, FOA fails in all projection, while the successful ration of FOIBA is 79.0%. The successful ratios of other algorithms are NRA: 62.0%, SOA: 67.0%, respectively. The average number of iterations and the CPU time of our integrated algorithms are also less than the corresponding single-point algorithms.

Example 5. We project 266 points on the bounding box (loose bounding) of several characters (represented by B-spline curves) onto these characters (see Figure 9), using the six algorithms, respectively. In this example, we set $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, and the average initial value error is 7.01×10^{-02}. The statistic data of the projection is shown in Table 7. Note that the average iterations only record the correct projections, and the CPU time records all the projections (including both correct and incorrect projections).

Experimental results show that, our SOIBA finds all correct solutions, while the successful ratio of SOA is 96.2%, even if there are no special points mentioned in Example 2 (because all initial values are not too far from the exact projective points and there is no sharp features). The successful ratios of other algorithms are NRA: 89.1%, FOA: 28.2%, NRIBA: 94.7%, FOIBA: 85.0%, respectively. The average number of iterations and the CPU time of our integrated algorithms are also less than the corresponding single-point algorithms.

5. Conclusion

We present a geometric iteration algorithm to compute the projection and inversion of a point onto planar parametric curves. Our algorithm uses biarcs to approximate the curve locally, and this local approximation achieves both higher precision and larger fitting region as compared to the single-point approximation using tangents or osculating circles [7, 8, 4]. Given the same initial value, the next projective point estimated by our algorithm is remarkably closer to the exact projective point than traditional geometric iteration algorithms based on single-point approximation. As a result, our algorithm converges faster and is less dependent on the initial value than them. Moreover, our algorithm provides a framework that adapts to any single-point approximation algorithm. The integration converges faster and is less dependent on the choice of the initial value compared to the integrated single-point algorithm alone.

Our algorithm can be easily extended to point projection and inversion on 3D parametric curves, by replacing our 2D biarc interpolation method introduced in subsection 2.2 with a 3D biarc interpolation method [26]. In future work, the local segment approximation method will be extended to support point projection and inversion on parametric surfaces, where the biarc will be replaced by some special 3D surface patch (for example, the biarc approximation surface [27]).

Acknowledgements

The research was supported by Chinese 973 Program(2010CB328001) and Chinese 863 Program(2012AA040902). The second author was supported by the NSFC(61035002, 61272235). The third author was supported by the NSFC(61063029, 61173077). The fourth author was supported by the NSFC(91315302).

References

[1] A. Limaiem, F. Trochu, Geometric algorithms for the intersection of curves and surfaces, Computers & graphics 19 (3) (1995) 391–403.

[2] P. J. Besl, N. D. McKay, Method for registration of 3-d shapes, in: Robotics-DL tentative, International Society for Optics and Photonics, 1992, pp. 586–606.

[3] H. Pottmann, S. Leopoldseder, M. Hofer, Registration without ICP, Computer Vision and Image Understanding 95 (1) (2004) 54–71.

[4] S.-M. Hu, J. Wallner, A second order algorithm for orthogonal projection onto curves and surfaces, Computer Aided Geometric Design 22 (3) (2005) 251–260.

[5] H.-C. Song, J.-H. Yong, Y.-J. Yang, X.-M. Liu, Algorithm for orthogonal projection of parametric curves onto B-spline surfaces, Computer-Aided Design 43 (4) (2011) 381–393.

[6] J. Pegna, F.-E. Wolter, Surface curve design by orthogonal projection of space curves onto free-form surfaces, Journal of Mechanical Design 118 (1) (1996) 45–52.

[7] J. Hoschek, D. Lasser, Fundamentals of Computer Aided Geometric Design, A.K. Peters, 1993.

[8] E. Hartmann, On the curvature of curves and surfaces defined by normalforms, Computer Aided Geometric Design 16 (5) (1999) 355–376.

[9] L. A. Piegl, W. Tiller, The NURBS Book, second ed, Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[10] L. A. Piegl, W. Tiller, Parameterization for surface fitting in reverse engineering, Computer Aided Design 33 (8) (2001) 593–603.

[11] D. E. Johnson, E. Cohen, A framework for efficient minimum distance computation, in: Proceedings - IEEE International Conference on Robotics and Automatio, Vol. 4, 1998, pp. 3678–3684.

[12] Y. L. Ma, W. Hewitt, Point inversion and projection for NURBS curve and surface: Control polygon approach, Computer Aided Geometric Design 20 (2) (2003) 79–99.

[13] D. E. Johnson, E. Cohen, Distance extrema for spline models using tangent cones, in: Proceedings of Graphics Interface 2005, 2005, pp. 169–175.

[14] Y.-T. Oh, Y.-J. Kim, J. Lee, M.-S. Kim, G. Elber, Efficient point projection to freeform curves and surfaces, in: Advances in Geometric Modeling and Processing, Springer, 2010, pp. 192–205.

[15] X.-D. Chen, J.-H. Yong, G. Wang, J.-C. Paul, G. Xu, Computing the minimum distance between a point and a NURBS curve, Computer-Aided Design 40 (10) (2008) 1051–1054.

[16] I. Selimovic, Improved algorithms for the projection of points on NURBS curves and surfaces, Computer Aided Geometric Design 23 (5) (2006) 439–445.

[17] X.-D. Chen, H. Su, J.-H. Yong, J.-C. Paul, J.-G. Sun, A counterexample on point inversion and projection for NURBS curve, Computer Aided Geometric Design 24 (5) (2007) 302.

[18] Y.-T. Oh, Y.-J. Kim, J. Lee, M.-S. Kim, G. Elber, Continuous point projection to planar freeform curves using spiral curves, The Visual Computer 28 (1) (2012) 111–123.

[19] J.-K. Seong, D. E. Johnson, G. Elber, E. Cohen, Critical point analysis using domain lifting for fast geometry queries, Computer-Aided Design 42 (7) (2010) 613–624.

[20] X.-M. Liu, L. Yang, J.-H. Yong, H.-J. Gu, J.-G. Sun, A torus patch approximation approach for point projection on surfaces, Computer Aided Geometric Design 26 (5) (2009) 593–598.

[21] Z. Šír, R. Feichtinger, B. Jüttler, Approximating curves and their offsets using biarcs and pythagorean hodograph quintics, Computer-Aided Design 38 (6) (2006) 608–618.

[22] J.-H. Yong, X. Chen, J.-C. Paul, An example on approximation by fat arcs and fat biarcs, Computer-Aided Design 38 (5) (2006) 515–517.

[23] J.-H. Yong, S.-M. Hu, J.-G. Sun, Bisection algorithms for approximating quadratic Bézier curves by G^1 arc splines, Computer-Aided Design 32 (4) (2000) 253–260.

[24] J.-H. Yong, S.-M. Hu, J.-G. Sun, A note on approximation of discrete data by G^1 arc splines, Computer-Aided Design 31 (14) (1999) 911–915.

[25] X.-Z. Liu, J.-H. Yong, G.-Q. Zheng, J.-G. Sun, Constrained interpolation with biarcs, Journal of Computer Aided Design & Computer Graphics 19 (1) (2007) 1–7.

[26] X. Song, M. Aigner, F. Chen, B. Jüttler, Circular spline fitting using an evolution process, Journal of computational and applied mathematics 231 (1) (2009) 423–433.

[27] Y.-J. Tseng, Y.-D. Chen, Three dimensional biarc approximation of freeform surfaces for machining tool path generation, International Journal of Production Research 38 (4) (2000) 739–763.

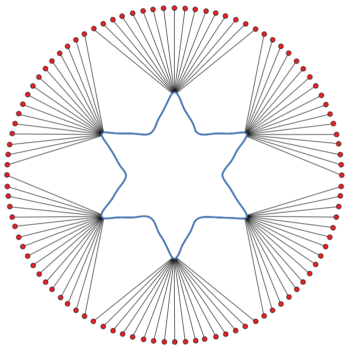

Figure 8: Illustration of Example 4: point projections on a star with sharp features.

Table 6: Statistic data for Example 4.

Methods	Correct solutions	Worst iterations	Average iterations	CPU time (ms)
NRA	62	8	6.77	5.35
FOA	0	-	-	-
SOA	67	176	19.11	7.66
NRIBA	100	5	4.7	0.24
FOIBA	79	43	5.98	4.85
SOIBA	100	5	4.04	0.43

Table 7: Statistic data for Example 5.

Methods	Correct solutions	Worst iterations	Average iterations	CPU time (ms)
NRA	237	8	4.77	3.03
FOA	75	167	11.23	5.38
SOA	256	9	4.72	0.782
NRIBA	251	6	4.38	1.41
FOIBA	226	30	5.98	0.79
SOIBA	266	6	3.63	0.72

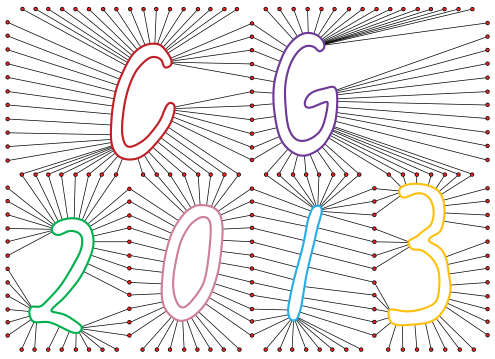

Figure 9: Illustration of Example 5: point projections on font characters.

and the CPU time of our integrated algorithms are also less than the corresponding single-point algorithms.

Example 4. We project 100 points on a bounding circle (loose bounding) onto a hexagonal star (designed by hand, not an exact hexagonal star), which has sharp features (corresponding to the special points mentioned in Example 2) on its six angles (see Figure 8), using the six algorithms, respectively. In this example, we set $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, and the average initial value error is 5.48×10^{-03}. The statistic data of the projection is shown in Table 6. Note that the average iterations only record the correct projections, and the CPU time records all the projections (including both correct and incorrect projections).

Experimental results show that, our SOIBA and NRIBA find all correct solutions, FOA fails in all projection, while the successful ration of FOIBA is 79.0%. The successful ratios of other algorithms are NRA: 62.0%, SOA: 67.0%, respectively. The average number of iterations and the CPU time of our integrated algorithms are also less than the corresponding single-point algorithms.

Example 5. We project 266 points on the bounding box (loose bounding) of several characters (represented by B-spline curves) onto these characters (see Figure 9), using the six algorithms, respectively. In this example, we set $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, and the average initial value error is 7.01×10^{-02}. The statistic data of the projection is shown in Table 7. Note that the average iterations only record the correct projections, and the CPU time records all the projections (including both correct and incorrect projections).

Experimental results show that, our SOIBA finds all correct solutions, while the successful ratio of SOA is 96.2%, even if there are no special points mentioned in Example 2 (because all initial values are not too far from the exact projective points and there is no sharp features). The successful ratios of other algorithms are NRA: 89.1%, FOA: 28.2%, NRIBA: 94.7%, FOIBA: 85.0%, respectively. The average number of iterations and the CPU time of our integrated algorithms are also less than the corresponding single-point algorithms.

5. Conclusion

We present a geometric iteration algorithm to compute the projection and inversion of a point onto planar parametric curves. Our algorithm uses biarcs to approximate the curve locally, and this local approximation achieves both higher precision and larger fitting region as compared to the single-point approximation using tangents or osculating circles [7, 8, 4]. Given the same initial value, the next projective point estimated by our algorithm is remarkably closer to the exact projective point than traditional geometric iteration algorithms based on single-point approximation. As a result, our algorithm converges faster and is less dependent on the initial value than them. Moreover, our algorithm provides a framework that adapts to any single-point approximation algorithm. The integration converges faster and is less dependent on the choice of the initial value compared to the integrated single-point algorithm alone.

Our algorithm can be easily extended to point projection and inversion on 3D parametric curves, by replacing our 2D biarc interpolation method introduced in subsection 2.2 with a 3D biarc interpolation method [26]. In future work, the local segment approximation method will be extended to support point projection and inversion on parametric surfaces, where the biarc will be replaced by some special 3D surface patch (for example, the biarc approximation surface [27]).

Acknowledgements

The research was supported by Chinese 973 Program(2010CB328001) and Chinese 863 Program(2012AA040902). The second author was supported by the NSFC(61035002, 61272235). The third author was supported by the NSFC(61063029, 61173077). The fourth author was supported by the NSFC(91315302).

References

[1] A. Limaiem, F. Trochu, Geometric algorithms for the intersection of curves and surfaces, Computers & graphics 19 (3) (1995) 391–403.

[2] P. J. Besl, N. D. McKay, Method for registration of 3-d shapes, in: Robotics-DL tentative, International Society for Optics and Photonics, 1992, pp. 586–606.

[3] H. Pottmann, S. Leopoldseder, M. Hofer, Registration without ICP, Computer Vision and Image Understanding 95 (1) (2004) 54–71.

[4] S.-M. Hu, J. Wallner, A second order algorithm for orthogonal projection onto curves and surfaces, Computer Aided Geometric Design 22 (3) (2005) 251–260.

[5] H.-C. Song, J.-H. Yong, Y.-J. Yang, X.-M. Liu, Algorithm for orthogonal projection of parametric curves onto B-spline surfaces, Computer-Aided Design 43 (4) (2011) 381–393.

[6] J. Pegna, F.-E. Wolter, Surface curve design by orthogonal projection of space curves onto free-form surfaces, Journal of Mechanical Design 118 (1) (1996) 45–52.

[7] J. Hoschek, D. Lasser, Fundamentals of Computer Aided Geometric Design, A.K. Peters, 1993.

[8] E. Hartmann, On the curvature of curves and surfaces defined by normalforms, Computer Aided Geometric Design 16 (5) (1999) 355–376.

[9] L. A. Piegl, W. Tiller, The NURBS Book, second ed, Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[10] L. A. Piegl, W. Tiller, Parameterization for surface fitting in reverse engineering, Computer Aided Design 33 (8) (2001) 593–603.

[11] D. E. Johnson, E. Cohen, A framework for efficient minimum distance computation, in: Proceedings - IEEE International Conference on Robotics and Automatio, Vol. 4, 1998, pp. 3678–3684.

[12] Y. L. Ma, W. Hewitt, Point inversion and projection for NURBS curve and surface: Control polygon approach, Computer Aided Geometric Design 20 (2) (2003) 79–99.

[13] D. E. Johnson, E. Cohen, Distance extrema for spline models using tangent cones, in: Proceedings of Graphics Interface 2005, 2005, pp. 169–175.

[14] Y.-T. Oh, Y.-J. Kim, J. Lee, M.-S. Kim, G. Elber, Efficient point projection to freeform curves and surfaces, in: Advances in Geometric Modeling and Processing, Springer, 2010, pp. 192–205.

[15] X.-D. Chen, J.-H. Yong, G. Wang, J.-C. Paul, G. Xu, Computing the minimum distance between a point and a NURBS curve, Computer-Aided Design 40 (10) (2008) 1051–1054.

[16] I. Selimovic, Improved algorithms for the projection of points on NURBS curves and surfaces, Computer Aided Geometric Design 23 (5) (2006) 439–445.

[17] X.-D. Chen, H. Su, J.-H. Yong, J.-C. Paul, J.-G. Sun, A counterexample on point inversion and projection for NURBS curve, Computer Aided Geometric Design 24 (5) (2007) 302.

[18] Y.-T. Oh, Y.-J. Kim, J. Lee, M.-S. Kim, G. Elber, Continuous point projection to planar freeform curves using spiral curves, The Visual Computer 28 (1) (2012) 111–123.

[19] J.-K. Seong, D. E. Johnson, G. Elber, E. Cohen, Critical point analysis using domain lifting for fast geometry queries, Computer-Aided Design 42 (7) (2010) 613–624.

[20] X.-M. Liu, L. Yang, J.-H. Yong, H.-J. Gu, J.-G. Sun, A torus patch approximation approach for point projection on surfaces, Computer Aided Geometric Design 26 (5) (2009) 593–598.

[21] Z. Šír, R. Feichtinger, B. Jüttler, Approximating curves and their offsets using biarcs and pythagorean hodograph quintics, Computer-Aided Design 38 (6) (2006) 608–618.

[22] J.-H. Yong, X. Chen, J.-C. Paul, An example on approximation by fat arcs and fat biarcs, Computer-Aided Design 38 (5) (2006) 515–517.

[23] J.-H. Yong, S.-M. Hu, J.-G. Sun, Bisection algorithms for approximating quadratic Bézier curves by G^1 arc splines, Computer-Aided Design 32 (4) (2000) 253–260.

[24] J.-H. Yong, S.-M. Hu, J.-G. Sun, A note on approximation of discrete data by G^1 arc splines, Computer-Aided Design 31 (14) (1999) 911–915.

[25] X.-Z. Liu, J.-H. Yong, G.-Q. Zheng, J.-G. Sun, Constrained interpolation with biarcs, Journal of Computer Aided Design & Computer Graphics 19 (1) (2007) 1–7.

[26] X. Song, M. Aigner, F. Chen, B. Jüttler, Circular spline fitting using an evolution process, Journal of computational and applied mathematics 231 (1) (2009) 423–433.

[27] Y.-J. Tseng, Y.-D. Chen, Three dimensional biarc approximation of freeform surfaces for machining tool path generation, International Journal of Production Research 38 (4) (2000) 739–763.