

Projecting points onto planar parametric curves by local biarc approximation

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

► **To cite this version:**

Hai-Chuan Song, Xin Xu, Kan-Le Shi, Jun-Hai Yong. Projecting points onto planar parametric curves by local biarc approximation. Computers and Graphics, Elsevier, 2014. <hal-00920672>

HAL Id: hal-00920672

<https://hal.inria.fr/hal-00920672>

Submitted on 19 Dec 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

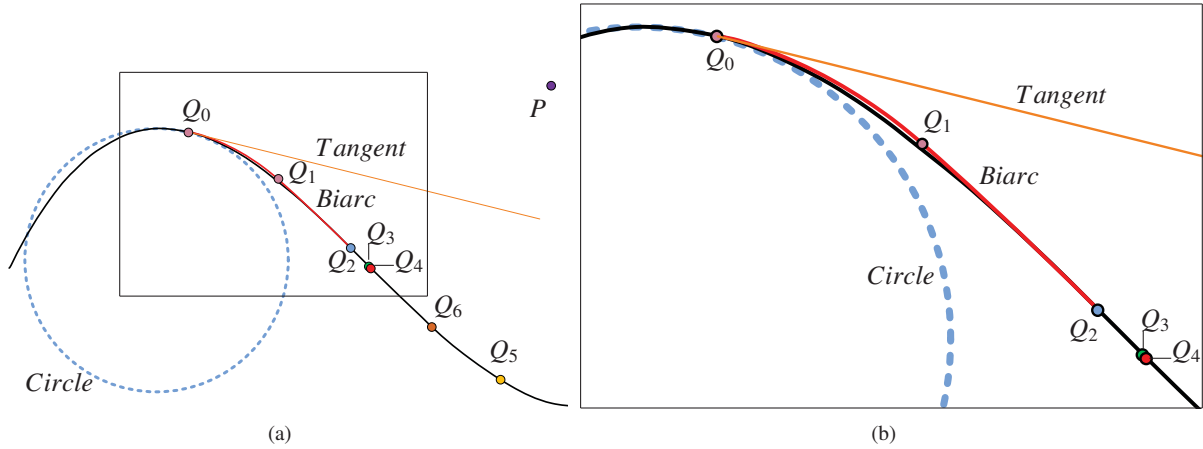


Figure 1: A comparison of the approximation precision of the first order algorithm [7, 8], the second order algorithm [4] and our algorithm: P is the test point, the black curve is the original curve, Q_0 is the initial point, orange point Q_6 , yellow point Q_5 , blue point Q_2 and red point Q_4 are 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).

51 query points onto planar spiral curves [18]. Seong et al.
 52 [19] dealt with this problem in another way. By elevat-
 53 ing the dimension of the problem, they transformed the
 54 point projection onto planar parametric curve into the
 55 intersection of an implicit surface and a straight line.

56 Besides algebraic methods (Newton-Raphson
 57 method), geometric methods were also proposed,
 58 which only involve the geometric information that is
 59 common to all possible parameterizations. Hoschek
 60 and Lasser [7], Hartmann [8] introduced the first
 61 order geometric iteration method. Hu and Wallner [4]
 62 proposed a second order geometric iteration method,
 63 in which they generated an osculating circle (a circle
 64 possessing the same curvature with the original curve
 65 at the osculating point) and projected the test point on
 66 it instead of the original curve or surface. Liu et al.
 67 [20] improved their method by replacing the circle of
 68 normal curvature with a second order osculating torus
 69 patch to the surface.

70 Note that the algebraic method and geometric meth-
 71 ods generally converge (if can converge) to the local
 72 minimum projective point nearest to the initial value
 73 [13]. All the subdivision methods we introduced above
 74 [10, 11, 12, 13, 14, 15, 16] can generate the initial value
 75 near to the global minimum projective point. Therefore,
 76 generally, a complete point projection include two parts:
 77 1. An algorithm to generate initial values; 2. An iteration
 78 algorithm to compute the precise projective point.
 79 In this paper, we mainly focus on the iteration algorithm
 80 in the second part. It means that we assume the initial

81 value has been provided by some kind of initial value
 82 generating algorithm.

83 1.2. Our contributions

84 The main idea of geometric methods is to locally ap-
 85 proximate the original curve by a special curve (first or-
 86 der algorithm uses tangent line, and second order algo-
 87 rithm uses osculating circle). The next projective point
 88 is estimated by projecting the test point onto the ap-
 89 proximation curve instead of the original curve. It is
 90 shown in the evolution of the geometric methods that
 91 higher approximation precision generally means higher
 92 convergence speed and better stability. However, during
 93 each iteration step, the traditional geometric algorithms
 94 approximate the original curve only with curve deriva-
 95 tives computed at a single point. So the approximation
 96 region is limited around this point, the approximation
 97 precision will reduce when moving away from this point
 98 on the original curve.

99 In order to improve the convergence speed and sta-
 100 bility of the point projection and inversion, we provide
 101 a geometric iteration algorithm based on local biarc ap-
 102 proximation. Our method approximates the correspond-
 103 ing segment on the original curve by a biarc rather than
 104 only one point during each iteration. Our local biarc ap-
 105 proximation has larger approximation region and higher
 106 approximation precision compared to traditional single-
 107 point approximation. According to the experimental re-
 108 sults in Section 4, our algorithm converges faster and is
 109 more robust than the traditional geometric algorithms.

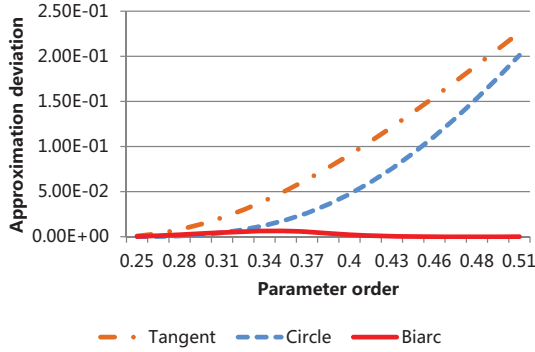


Figure 2: Approximation deviation comparison in Figure 1. Abscissa is the parameter of the moving point on the original curve. Ordinate is the distance from the moving point to approximation curve.

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 ap-

proximation 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 $C(t_0 + \Delta t)$, $C'(t_0 + \Delta t)$.

192 3. Interpolate the boundary conditions with a biarc.

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

195 2.1. Compute the interval width

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

1. User-defined constant parametric increment, and

$$\Delta t = \text{const} \cdot t. \quad (1)$$

2. Newton-Raphson method [9], and

$$\Delta t = \frac{C'(t_0) \cdot (C(t_0) - P)}{C''(t_0) \cdot (C(t_0) - P) + |C'(t_0)|^2}. \quad (2)$$

3. First order algorithm [7, 8], and

$$\Delta t = \frac{C'(t_0) \cdot (Q - C(t_0))}{C'(t_0) \cdot C'(t_0)}, \quad (3)$$

199 where Q is the projective point of P on the tangent
200 line at $C(t_0)$.

4. Second order algorithm [4], and

$$\Delta t = \frac{(Q - C(t_0)) \times C''(t_0)}{\kappa \|C'(t_0)\|^3}, \quad (4)$$

201 where Q is the projective point of P on the osculat-
202 ing circle at $C(t_0)$, κ is the curvature of $C(t_0)$, and
203 we have $\kappa = (C'(t_0) \times C''(t_0)) / \|C'(t_0)\|^3$.

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

207 Our algorithm therefore provides a framework that
208 adapts to any single-point approximation algorithm (use
209 it to compute Δt). Moreover, according to our exper-
210 iments, the integration converges faster and is less de-
211 pendent on the choice of the initial value compared to
212 the integrated original single-point algorithm alone.

213 According to our experience, considering the conver-
214 gence speed and the stability, priority of the four strate-
215 gies is $4 > 2 > 3 > 1$, where ‘>’ means better (con-
216 verge faster and less independent on the initial value)
217 than. After we derive Δt , the interval is determined by
218 $[t_0, t_0 + \Delta t]$.

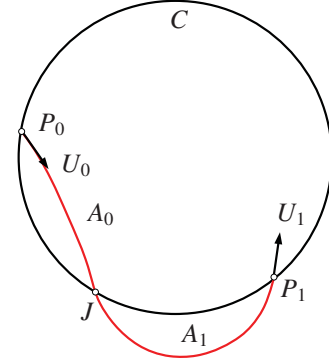


Figure 3: A biarc (red) and the joint circle (black) [21].

219 2.2. Approximate the curve segment by biarc

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

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

- 233 1. The arc A_0 has the end points P_0 and J , and U_0 is
234 tangent to A_0 with orientation corresponding to a
235 parametrization of A_0 from P_0 to J .
- 236 2. The arc A_1 has the end points J and P_1 and U_1 is
237 tangent to A_1 with orientation corresponding to a
238 parametrization of A_1 from J to P_1 .
- 239 3. The two arcs have a common unit tangent vector at
240 J , with orientation corresponding to a parametriza-
241 tion of A_0 from P_0 to J and of A_1 from J to P_1 .

242 The biarc interpolation plays an important part in tool
243 path generation in CNC (Computerized Numerical Con-
244 trol) [22, 23], and approximation of curves [23] and dis-
245 crete data [24]. Constrained interpolation with biarc is
246 also widely studied [21, 25].

247 The locus of all possible joint J is a circle (C in Fig-
248 ure 3) passing through P_0 and P_1 [21]. Various biarc
249 interpolation schemes are distinguished by the choice
250 of the joint J . Among the most important ones are
251 the ‘‘equal chord’’ biarc and the ‘‘parallel tangent’’ biarc.
252 The former one is constructed so that the two segments

253 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, owing to its

254 simplicity in implementation and better approximation precision according to our experiments.
 259 Given boundary data $P_0 = C(t_0)$, $D_0 = C'(t_0)$ and
 260 $P_1 = C(t_0 + \Delta t)$, $D_1 = C'(t_0 + \Delta t)$, we generate the biarc
 261 $BA(s)$ in following steps:

- 262 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.
- 263 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 .
- 264 3. Construct the unique arcs A_0, A_1 satisfying properties 1, 2 of Definition 1.
- 265 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 $s_J = \text{ArcLength}(A_0)/(\text{ArcLength}(A_0) + \text{ArcLength}(A_1))$.

275 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.
 278 It means that its approximation precision is higher than the first and the second order algorithms.

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

- 284 1. The construction of the biarc is simple, and can be done in constant time.
- 285 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.
- 286 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.

293 3. Point projection on the biarc and parameter inversion

295 3.1. Point projection on the biarc

296 As shown in Figure 4, the point projection on biarc
 297 $BA(s)$ can be computed by simply projecting the test
 298 point onto the two circles which A_0 and A_1 are embed
 299 in, respectively.

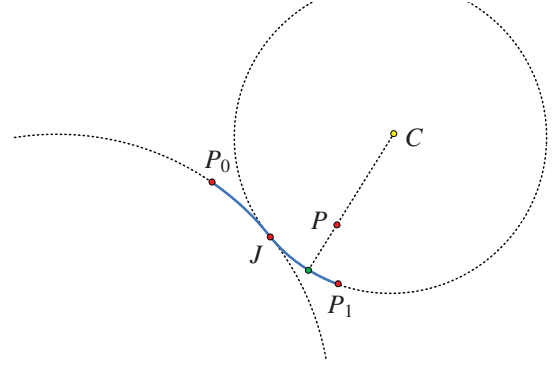


Figure 4: Point projection on the biarc.

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

- 303 1. If there is only one projective point in the parametric domain $[0, 1]$, this point is chosen as the valid projective point.
- 304 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.
- 305 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.

315 The parameter of the valid projective point is recorded
 316 in variable *param.biarc*.

317 3.2. Estimate the next projective point

318 After we derived the projective parameter
 319 *param.biarc* on the biarc, we refine Δt with it.
 320 Recall that the curve segment $BA(0) \sim BA(1)$ on
 321 the biarc is corresponding with the curve segment
 322 $C(t_0) \sim C(t_0 + \Delta t)$ on the original curve. So Δt can
 323 be refined by $\Delta t = \text{param.biarc} \times \Delta t$ (note that, for
 324 extreme high-order Bézier curves, this linear parameter
 325 interpolation may be unstable). Then the parameter of
 326 the next projective point is estimated by $t_0 = t_0 + \Delta t$,
 327 which will be used for the next iteration.

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

$$(t_{i+1} - t_i)C'(t_i) \leq \varepsilon_1. \quad (5)$$

$$|C(t_i) - P| \leq \varepsilon_1. \quad (6)$$

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

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}

328 t_i is the parameter obtained at the i th iteration, and $\varepsilon_1, \varepsilon_2$ 356
329 are two zero tolerances of Euclidean distance and cosine. 357
330 The iteration is halted if any of the three conditions 358
331 above is satisfied.

332 4. Examples and comparisons

333 We present five examples for point projection, 363
334 and make comparisons with NRA (Newton-Raphson 364
335 method [9]), FOA (first order algorithm [7, 8]), 365
336 SOA (second order algorithm [4]), NRIBA (Newton- 366
337 Raphson-integrated biarc algorithm: our algorithm 367
338 whose Δt is generated by Newton-Raphson algorithm 368
339 [9] as introduced in subsection 2.1), FOIBA (first-order- 369
340 integrated biarc algorithm: our algorithm whose Δt is 370
341 generated by first order algorithm [7, 8] as introduced in 371
342 subsection 2.1), SOIBA (second-order-integrated biarc 372
343 algorithm: our algorithm whose Δt is generated by sec- 373
344 ond order algorithm [4] as introduced in subsection 2.1). 374
345 All the experiments are implemented with Intel Core i5
346 CPU 3.0 GHz, 8G Memory. In all of our experiments
347 $\varepsilon_1, \varepsilon_2$ are both the convergence tolerances introduced in
348 subsection 3.2.

349 There are three main criteria to evaluate point projec-
350 tion iteration methods.

- 351 1. *Correctness.* If the distance between the computed
352 projective point and the exact closest point satisfies
353 a given precision, it is treated as a correct solution.
- 354 2. *Speed of convergence.* We measure the conver-
355 gence 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.

- 359 3. *Independence on the initial value.* The initial value
360 has a significant impact on the correctness of the
361 Newton-like iteration algorithms. Although there
362 were lots of methods finding initial value (as de-
363 scribed in subsection 1.1), no method can claim
364 that its initial value is good enough to make the it-
365 eration always converge. So if the iteration method
366 is less dependent on the initial value, the method
367 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 or-
der algorithm) in order to test the performance of
different iteration methods. In practice, we recom-
mend to estimate the initial value with the method
introduced in [15].

375 **Example 1.** We first test Example 2 of [4] (see
376 Figure 5), where two test points $P_1 = (381, 252)$ and
377 $P_2 = (332, 200)$ are projected onto a cubic B-spline
378 curve $C(t)$ with initial parameter 0.75 and 0.5, respec-
379 tively. In this example, we set $\varepsilon_1 = \varepsilon_2 = 10^{-6}$, which is
380 the same with [4].

381 In Table 1, we compare the convergence of the six
382 algorithms. If any method can converge within 10 iter-
383 ations, we will provide the convergence parameter in
384 this table. As shown in this table, our integrated al-
385 gorithms (NRIBA, FOIBA and SOIBA) converge with

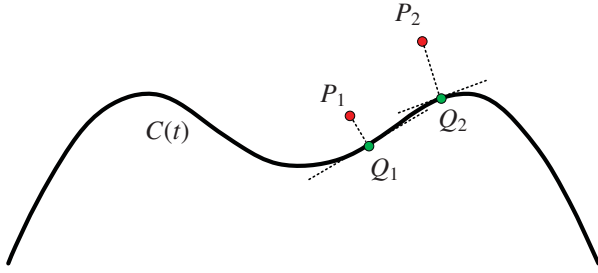


Figure 5: Illustration of Example 1: Q_1 and Q_2 are the exact projective points of $P_1 = (381, 252)$ and $P_2 = (332, 200)$. The control points and knot vector of $C(t)$ are $\{(100, 100), (140, 196), (200, 240), (260, 164), (340, 164), (400, 240), (460, 196), (500, 100)\}$ and $(0, 0, 0, 0, 0.2, 0.4,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

386 less iterations than the corresponding single-point algo-
 387 rithms (NRA, FOA and SOA). The processing times of
 388 SOA, NRIBA, FOIBA and SOIBA are comparable, and
 389 are less than those of NRA and FOA.

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

398 In Table 3, we compare the convergence of the six algo-
 399 rithms under different tolerances. As shown in this ta-
 400 ble, when we decrease the tolerance, the numbers of it-
 401 erations of FOA, SOA, FOIBA increase; however NRA,
 402 NRIBA and SOIBA remain. NRA performs better than
 403 FOA and SOA, because the initial value is not too far
 404 from the exact projective point. However, NRIBA con-
 405 verges much faster than NRA. FOIBA converges much
 406 faster than FOA. In the third tolerance case, the number
 407 of iteration of FOA jumped significantly from 66 to 91.
 408 Influenced by FOA, the number of iteration of FOIBA
 409 jumped from 5 to 18, which is still relatively acceptable.
 410 SOIBA converges faster than SOA, and the gap is get-

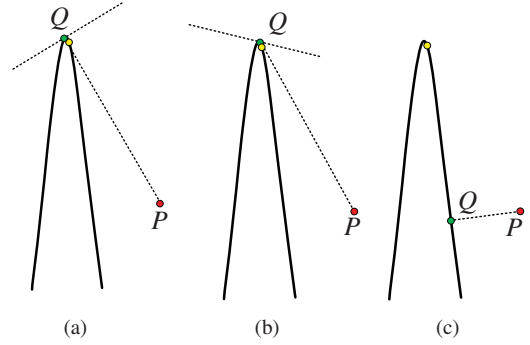


Figure 6: Illustration of Example 2. The yellow point is the initial projection. The control points of the Bézier curve are $\{(0, 0), (110, 1000), (90, 1000), (200, 0)\}$: (a) projection result Q of NRA and its tangent; (b) projection result Q of SOA and its tangent; (c) projection result Q of SOIBA.

411 ting larger as we decrease the tolerance.

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

416 NRA converges to the local projective point $C(t =$
 417 $0.487)$ in 4 steps, which is not the global nearest pro-
 418 jective point (see Figure 6 (a)). However NRIBA con-
 419 verges to the correct global nearest projective point
 420 $C(t = 0.916)$ in 5 steps. It means that our integrated
 421 algorithm is less independent on the initial value, and is
 422 more likely to converge to the global nearest projective
 423 point.

424 FOA and FOIBA both converge to the correct pro-
 425 jective point $C(t = 0.916)$ in 5 steps. As shown in
 426 Table 4, the first parametric increment of FOA is 2.49.
 427 This makes the parameter equal to 3.02 and run out of
 428 the parametric domain of the curve ($0 \sim 1$), which is un-
 429 acceptable in practice. In our implementation, we patch
 430 FOA by drawing the parameter back to the nearest para-
 431 metric domain boundary 1, and the iteration continues.
 432 However, FOIBA always iterate within the parametric
 433 domain of the curve, and converges to the correct pro-
 434 jective point $C(t = 0.916)$ in 5 steps. It means that, our
 435 integrated algorithm is more stable.

436 SOA converges to the wrong projective point $C(t =$
 437 $0.513)$ (neither the closest point nor the orthogonal pro-
 438 jective point) in 6 steps, which is still close to the initial
 439 value (see Figure 6 (b)). However SOIBA converges to
 440 the correct projective point $C(t = 0.916)$ in 5 steps (see
 441 Figure 6 (c)). This is because $(Q - C(t_0))$ is nearly par-
 442 allel with $C''(t_0)$ in Equation (4), leading to $\Delta t \approx 0$, and
 443 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 / 1.15×10^{-02}	5 / 1.17×10^{-02}	5 / 1.11×10^{-02}	5 / 1.21×10^{-02}	5 / 1.22×10^{-02}
Δt_{FOA}	57 / 1.54×10^{-02}	66 / 1.95×10^{-02}	91 / 2.96×10^{-02}	101 / 3.43×10^{-02}	112 / 3.64×10^{-02}
Δt_{SOA}	5 / 2.46×10^{-03}	5 / 2.97×10^{-03}	11 / 6.48×10^{-03}	11 / 6.51×10^{-03}	11 / 6.50×10^{-03}
Δt_{NRIBA}	4 / 2.11×10^{-03}	4 / 2.14×10^{-03}	4 / 2.12×10^{-03}	4 / 2.21×10^{-03}	4 / 2.23×10^{-03}
Δt_{FOIBA}	4 / 2.21×10^{-03}	5 / 2.35×10^{-03}	18 / 1.50×10^{-02}	18 / 1.51×10^{-02}	18 / 1.50×10^{-02}
Δt_{SOIBA}	4 / 2.96×10^{-03}	4 / 3.06×10^{-03}	4 / 4.06×10^{-03}	4 / 4.07×10^{-03}	4 / 4.06×10^{-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	

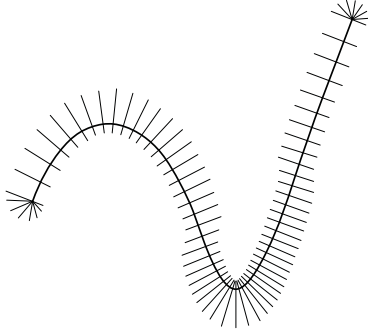


Figure 7: Illustration of Example 3: point projections on a smooth curve.

444 (see the first steps of SOA in Table 4). This case al-
 445 ways occurs at the special point whose curvature is rel-
 446 atively much bigger than its neighboring region, leading
 447 to a very small osculating circle, the approximation re-
 448 gion of which is small. With the help of our local biarc
 449 approximation in SOIBA, the approximation region is
 450 enlarged, and the iteration can “jump” away from the
 451 special point (see the first two steps of SOIBA in Ta-
 452 ble 4), and converges to the correct projective point even
 453 with the “bad” initial value. In this example, all the
 454 single-point algorithms fail to some extent, while our
 455 integrated algorithms all converge to the correct projec-
 456 tive point.

457 **Example 3.** We project 134 points on a spurious off-

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

458 set of a smooth curve onto the curve itself (see Figure 7),
 459 using the six algorithms, respectively. In this example,
 460 we set $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, and the average initial value er-
 461 ror is 2.27×10^{-02} . The statistic data of the projection is
 462 shown in Table 5. Note that the average iterations only
 463 record the correct projections, and the CPU time records
 464 all the projections (including both correct and incorrect
 465 projections).

466 Experimental results show that, our SOIBA finds all
 467 correct solutions, while the successful ratio of SOA is
 468 97.8%, even if there is no special points mentioned in
 469 Example 2 (because all initial values are not too far
 470 from the exact projective points and there is no sharp
 471 features). The successful ratios of other algorithms are
 472 NRA: 88.1%, FOA: 67.9%, NRIBA: 97.8%, FOIBA:
 473 94.8%, respectively. The average number of iterations

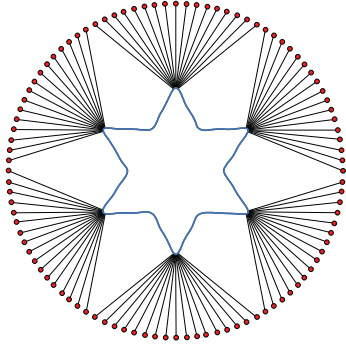


Figure 8: Illustration of Example 4: point projections on a star with sharp features.

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

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

476 **Example 4.** We project 100 points on a bounding
477 circle (loose bounding) onto a hexagonal star (designed
478 by hand, not an exact hexagonal star), which has sharp
479 features (corresponding to the special points mentioned
480 in Example 2) on its six angles (see Figure 8), using
481 the six algorithms, respectively. In this example, we set
482 $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, and the average initial value error
483 is 5.48×10^{-03} . The statistic data of the projection is
484 shown in Table 6. Note that the average iterations only
485 record the correct projections, and the CPU time records
486 all the projections (including both correct and incorrect
487 projections).

488 Experimental results show that, our SOIBA and
489 NRIBA find all correct solutions, FOA fails in all pro-
490 jection, while the successful ration of FOIBA is 79.0%.
491 The successful ratios of other algorithms are NRA:
492 62.0%, SOA: 67.0%, respectively. The average number
493 of iterations and the CPU time of our integrated algo-
494 rithms are also less than the corresponding single-point
495 algorithms.

496 **Example 5.** We project 266 points on the bounding

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

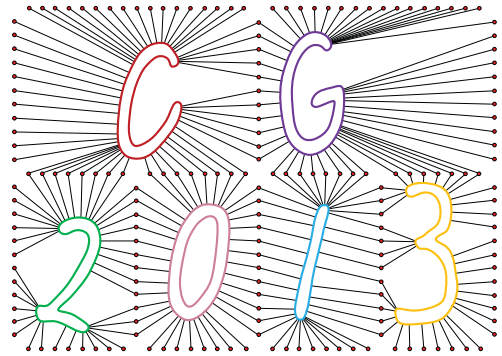


Figure 9: Illustration of Example 5: point projections on font characters.

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

506 Experimental results show that, our SOIBA finds all
507 correct solutions, while the successful ratio of SOA is
508 96.2%, even if there are no special points mentioned
509 in Example 2 (because all initial values are not too far
510 from the exact projective points and there is no sharp
511 features). The successful ratios of other algorithms are
512 NRA: 89.1%, FOA: 28.2%, NRIBA: 94.7%, FOIBA:
513 85.0%, respectively. The average number of iterations
514 and the CPU time of our integrated algorithms are also
515 less than the corresponding single-point algorithms.

516 5. Conclusion

517 We present a geometric iteration algorithm to com-
518 pute the projection and inversion of a point onto pla-

nar 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 Program(2010CB328001) and Chinese 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. Limaïem, 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. Leopoldseider, 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.

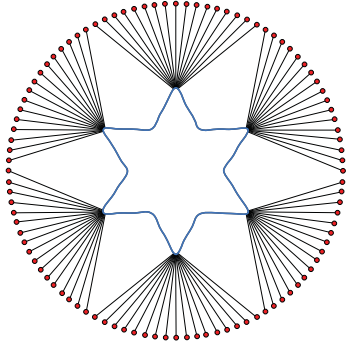


Figure 8: Illustration of Example 4: point projections on a star with sharp features.

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

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

476 **Example 4.** We project 100 points on a bounding
477 circle (loose bounding) onto a hexagonal star (designed
478 by hand, not an exact hexagonal star), which has sharp
479 features (corresponding to the special points mentioned
480 in Example 2) on its six angles (see Figure 8), using
481 the six algorithms, respectively. In this example, we set
482 $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, and the average initial value error
483 is 5.48×10^{-03} . The statistic data of the projection is
484 shown in Table 6. Note that the average iterations only
485 record the correct projections, and the CPU time records
486 all the projections (including both correct and incorrect
487 projections).

488 Experimental results show that, our SOIBA and
489 NRIBA find all correct solutions, FOA fails in all pro-
490 jection, while the successful ration of FOIBA is 79.0%.
491 The successful ratios of other algorithms are NRA:
492 62.0%, SOA: 67.0%, respectively. The average number
493 of iterations and the CPU time of our integrated algo-
494 rithms are also less than the corresponding single-point
495 algorithms.

496 **Example 5.** We project 266 points on the bounding

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

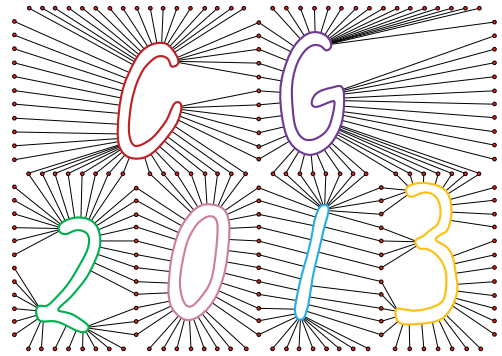


Figure 9: Illustration of Example 5: point projections on font characters.

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

506 Experimental results show that, our SOIBA finds all
507 correct solutions, while the successful ratio of SOA is
508 96.2%, even if there are no special points mentioned
509 in Example 2 (because all initial values are not too far
510 from the exact projective points and there is no sharp
511 features). The successful ratios of other algorithms are
512 NRA: 89.1%, FOA: 28.2%, NRIBA: 94.7%, FOIBA:
513 85.0%, respectively. The average number of iterations
514 and the CPU time of our integrated algorithms are also
515 less than the corresponding single-point algorithms.

516 5. Conclusion

517 We present a geometric iteration algorithm to com-
518 pute the projection and inversion of a point onto pla-

nar 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 Program(2010CB328001) and Chinese 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. Limaïem, 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. Leopoldseider, 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.