Pacific Graphics (2014)
J. Keyser, Y. J. Kim, and P. Wonka (Editors)

Projecting points onto planar parametric curves by local biarc approximation

Hai-Chuan Song[1,2,3,4] and Kan-Le Shi[1,3,4] and Jun-Hai Yong[1,3,4] and Sen Zhang[1,2,3,4]

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

Abstract

This paper proposes a geometric iteration algorithm for computing point projection and inversion on surfaces 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 3D biarc on the original surface to locally approximate the original surface starting from the current projection point. Then we compute the projection 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 projection 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.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

1. Introduction

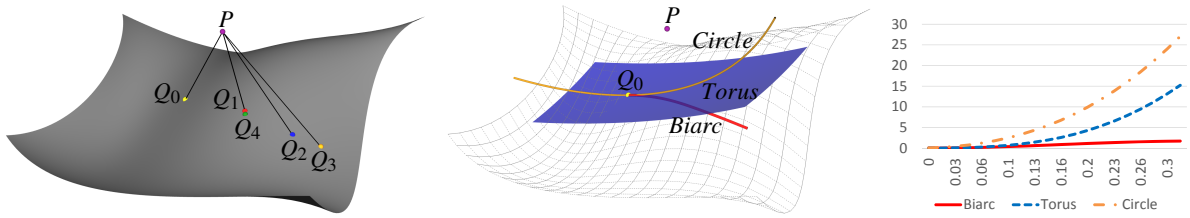
Projection of a test point on a surface aims to find the closest point, as well as the corresponding parameter, on the surface. Specially, when the test point lies on the surface, the problem of point projection becomes point inversion. This operation has been extensively used in geometric processing algorithms such as surface intersection [LT95], interactive object selection and shape registration [BM92, PLH04, HW05]. It is also a fundamental component of the algorithms of curve and surface projection as well [SYYL11, PW96].

Point projection and inversion can be translated to solving the minimum distance equation $(Q-P)\times n=0$, where P is the test point, Q is the point closest to P on the original surface and n is the normal vector of the original 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 equation [LT95, Har99]. Piegl and Tiller [PT97] 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 [PT01, JC98, MH03, JC05, OKL*10, CYW*08, Sel06]. The key point of this kind of algorithms is to eliminate the surface patches which do not contain the closest points. Ma and Hewitt [MH03] 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 [CSY*07]. Johnson and Cohen utilized the tangent cone to search for the portions of the surface contain the projection points [JC05]. A more practical exclusion criterion based on Voronoi cell test was proposed in [Sel06].

Geometric methods, which converge faster and are more robust than algebraic methods (Newton-Raphson method) [SXSY14], were also proposed. Hoschek and Lasser [HL93], Hartmann [Har99] introduced first order method. Hu and Wallner [HW05] proposed second order method, in which they generated an osculating circle (a circle possessing the same curvature with the original surface at the osculating point) and projected the test point on it instead of the original surface. Liu et al. [LYY*09] improved their

DOI: 10.2312/pgs.20141248



(a) (b) (c)

Figure 1: *A comparison of the approximation precision of Hu and Wallner's algorithm [HW05], Liu et al.'s algorithm [LYY*09] and our algorithm: (a) the projection points obtained by the three algorithms after the first step iteration. The violet test point P is projected onto the gray 3D surface. The yellow point Q_0 is the initial point. The orange point Q_3, blue point Q_2, and red point Q_1 are projection points obtained by Hu and Wallner's algorithm [HW05], Liu et al.'s algorithm [LYY*09] and our algorithm after the first step iteration, respectively. Q_4 is the exact closest point; (b) the three types of approximation geometries used in the three algorithms; (c) approximation deviation comparison. Abscissa: parametric distance away from Q_0. Ordinate: Hausdorff distance from the three types of approximation geometries to the original surface.*

method by replacing the osculating circle with osculating torus patch.

As shown in the evolution of the geometric methods, higher approximation precision generally means higher convergence speed and better stability [SXSY14]. However, in each iteration, traditional geometric algorithms approximate the original surface only at a single point. The approximation precision reduces when moving away from this point on the original surface.

In order to improve the convergence speed and stability of point projection and inversion, we provide a geometric iteration algorithm based on local biarc approximation, which approximates the corresponding region on the original surface by a biarc (a curve, satisfying given G^1 boundary data, composed of two connected arcs having the same tangent at the common end point. Detailed definition of biarc is in [ŠFJ06]) in each iteration. An example is shown in Figure 1, where the orange circle and the blue surface are the osculating circle and torus patch at Q_0 used in [HW05] and [LYY*09], the red curve is the biarc used in our algorithm. Our biarc has larger approximation region and higher approximation precision than single-point approximation (as shown in Figure 1 (c), when moving away from Q_0, the approximation precisions of osculating circle and torus drop significantly, while that of biarc changes slowly). So our next projection point Q_1 is much closer to the exact projection point than other single-point methods as shown in Figure 1. According to the experimental results in Section 4, our algorithm converges faster and is more robust than traditional geometric algorithms. This algorithm is an extension of another paper of ours [SXSY14], in which we deal with point projection and inversion on planar parametric curves.

Given a test point P, a 3D parametric surface $S(u,v)$ and the parameter value $q_0 = (u_0, v_0)$ of the roughly estimated projection point Q_0, as illustrated in Figure 1, we need to compute the parameter of the exact projection point. Our algorithm can be described in summary as follows:

1. According to initial parameter $q_0 = (u_0, v_0)$, compute the initial interval width $(\Delta u_0, \Delta v_0)$ and corresponding tangent T_0 of $S(u,v)$ at Q_0, and set $q_1 = (u_1, v_1) = (u_0 + \Delta u_0, v_0 + \Delta v_0)$, $Q_1 = S(q_1)$.
2. Compute parameter increment $(\Delta u_1, \Delta v_1)$ and corresponding tangent T_1 of $S(u,v)$ at Q_1.
3. Interpolate boundary conditions Q_0, T_0 and Q_1, T_1 with a biarc $BA(s)$ (the red curve in Figure 1 (b)), which is used to approximate $S(u,v)$ from Q_0 to Q_1.
4. Project P onto $BA(s)$ and compute a new estimated parameter $q_2 = (u_2, v_2)$ (Q_1 in Figure 1).
5. Set $q_0 = q_1$, $Q_0 = Q_1$, $T_0 = T_1$, and $q_1 = q_2$.
6. Repeat steps 2-5 until corresponding projection point $S(u_1, v_1)$ satisfies the precision requirement.

We will introduce our algorithm in detail in the following sections.

2. Local biarc approximation

With the initial projection point $Q_0 = S(u_0, v_0)$, we use first order algorithm [HL93, Har99] to compute interval width $(\Delta u_0, \Delta v_0)$ as follows:

$$Q - Q_0 = S_{u0} \cdot \Delta u_0 + S_{v0} \cdot \Delta v_0, \tag{1}$$

where Q is the projection point of P onto the tangent plane at Q_0. Then we have:

$$\Delta u_0 = \frac{({S_{v0}}^2 \cdot S_{u0} - (S_{u0} \cdot S_{v0}) \cdot S_{v0}) \cdot (Q - Q_0)}{{S_{u0}}^2 \cdot {S_{v0}}^2 - (S_{u0} \cdot S_{v0})^2}, \tag{2}$$

$$\Delta v_0 = \frac{({S_{u0}}^2 \cdot S_{v0} - (S_{u0} \cdot S_{v0}) \cdot S_{u0}) \cdot (Q - Q_0)}{{S_{u0}}^2 \cdot {S_{v0}}^2 - (S_{u0} \cdot S_{v0})^2}. \tag{3}$$

More details can be found in [HL93, Har99]. For the first iteration, we set $q_1 = (u_1, v_1) = (u_0 + \Delta u_0, v_0 + \Delta v_0)$ and

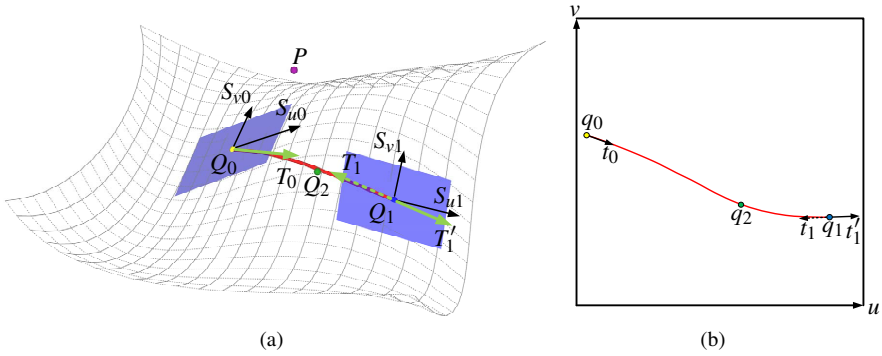


Figure 2: *Local biarc approximation:(a) the 3D biarc on $S(u,v)$, where P is the test point, Q_0 and Q_1 are the start point and end point of the biarc, S_{u0} and S_{v0} are the partial derivatives of $S(u,v)$ at Q_0 and Q_1, T_0 and T_1 are the tangents, determined by P, at Q_0 and Q_1, T_1' is the inverse of T_1, Q_2 is the next projection point estimated by our method in the first iteration; (b) the 2D biarc in the parametric domain of $S(u,v)$, where the start point q_0 and the end point q_1 are pre-image points of Q_0 and Q_1 in the parametric domain, t_0 and t_1 are the pre-image vectors of T_0 and T_1, t_1' is the inverse of t_1, q_2 is the pre-image point of Q_2 in the parametric domain estimated by our method in the first iteration.*

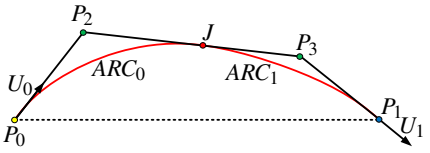


Figure 3: *A 3D biarc (red) and its tangents (black).*

$Q_1 = S(q_1)$. For other iterations, q_1 and Q_1 can be derived by last iteration, which will be introduced in the following part. We set T_0 equals to the unit vector of $(Q - Q_0)$, and t_0 equals to the unit vector of $(\Delta u_0, \Delta v_0)$ in the parametric domain of $S(u,v)$. Analogously, T_1 and t_1 can also be computed for Q_1 and q_1 using first order algorithm introduced above (See Figure 2).

Given 2D boundary data $P_0 = q_0$, $U_0 = t_0$ and $P_1 = q_1$, $U_1 = t_1$, with the method in [SXSY14], we generate 2D biarc $ba(s)$, where $s \in [0,1]$. It follows that $ba(s) = arc_0$ when $s \in [0, s_J]$, and $ba(s) = arc_1$ when $s \in [s_J, 1]$, where $s_J = ArcLength(arc_0)/(ArcLength(arc_0) + ArcLength(arc_1))$, arc_0 and arc_1 are the two arcs of $ba(s)$. More details can be found in [SXSY14].

Given 3D boundary data $P_0 = Q_0$, $U_0 = T_0$ and $P_1 = Q_1$, $U_1 = T_1$, Chui et al. [CCY08] proposed a 3D biarc interpolation method as follows. As shown in Figure 3, P_0P_2 and P_2J are tangent to ARC_0, JP_3 and P_3P_1 are tangent to ARC_1. Set $P_2 = P_0 + x \cdot U_0$, $P_3 = P_1 - y \cdot U_1$ it follows that:

$$x = \frac{(P_1 - P_0)^2 - 2 \cdot y \cdot ((P_1 - P_0) \cdot U_1))}{(2 \cdot y \cdot (1 - U_0 \cdot U_1) + 2 \cdot ((P_1 - P_0) \cdot U_0))}. \quad (4)$$

More details can be found in [CCY08]. Once y is determined, x can be computed by Equation 4. Then P_2 and P_3 are both determined. As a result, ARC_0 and ARC_1 are also determined. Therefore, we call y the shape parameter of 3D biarc. Generally, the shape of 2D biarc interpolation is better defined (because of low degree of freedom). In order to make $BA(s)$ and $ba(s)$ in similar shapes, in this paper, shape parameter y is determined by:

$$y = \frac{\|Q_0Q_1\| \cdot r_1 \cdot \tan \frac{l_1}{2 \cdot r_1}}{\|q_0q_1\|}, \quad (5)$$

where r_1 and l_1 are the radius and arc length of the second arc arc_1 of $ba(s)$, respectively. $BA(s)$ is parameterized in the same way as $ba(s)$ (based on arc length), so $BA(s)$ and $ba(s)$ share the same parameter s. Moreover, in order to obtain a well-shaped biarc interpolation, before interpolation, we preprocess the boundary data as follows:

1. if $(q_1 - q_0) \cdot t_0 < 0$, we reverse t_0 and T_0.
2. if $(q_1 - q_0) \cdot t_1 < 0$, we reverse t_1 and T_1.

An example for 3D and 2D biarc interpolation is shown in Figure 2. Note that, t_1 and T_1 are reversed to t_1' and T_1' because $(q_1 - q_0) \cdot t_1 < 0$.

3. Point projection on biarc and parameter inversion

We project P onto $BA(s)$ by simply projecting onto ARC_0 and ARC_1, respectively. Then record the parameter of the valid projection point by *param_biarc*. Recall that $BA(s)$ and $ba(s)$ share the same parameter s. So a new parameter is estimated by evaluating $ba(param\_biarc)$, and we set $q_2 = (u_2, v_2) = ba(param\_biarc)$ (q_2 in Figure 2 (b)). Then we set $q_0 = q_1$, $Q_0 = Q_1$, $T_0 = T_1$, $q_1 = q_2$, and continue to the next iteration.

We apply the convergence criteria provided by Piegl and Tiller [PT97]:

1. $\|(u_1-u_0)S_u(u_1,v_1)+(v_1-v_0)S_v(u_1,v_1)\| \le \varepsilon_1$.
2. $\|S(u_1,v_1)-P\| \le \varepsilon_1$.
3. $\dfrac{|S_u(u_1,v_1)\cdot(S(u_1,v_1)-P)|}{\|S_u(u_1,v_1)\|\cdot\|S(u_1,v_1)-P\|} \le \varepsilon_2$,
   $\dfrac{|S_v(u_1,v_1)\cdot(S(u_1,v_1)-P)|}{\|S_v(u_1,v_1)\|\cdot\|S(u_1,v_1)-P\|} \le \varepsilon_2$.

$S_u(u,v)$ and $S_v(u,v)$ are the partial derivatives of $S(u,v)$, ε_1 and ε_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 make comparisons with Hu and Wallner's algorithm [HW05] and Liu et al.'s algorithm [LYY*09] with four examples. All experiments are implemented with Intel Core i5 CPU 3.0 GHz, 8G Memory. In all experiments $\varepsilon_1 = \varepsilon_2 = 10^{-10}$, which is the same as [LYY*09].

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

1. *Correctness*. If the distance between the computed projection point and the exact projection point satisfies a given precision, it is treated as a correct solution.
2. *Speed of convergence*. We measure the convergence speed by number of iterations and CPU time.
3. *Independence on the initial value*. If a method is less dependent on initial value, this method is more robust.

Example 1. We first test Example 1 of [LYY*09], where two test points $P_1 = (120,10,100)$ (Case 1) and $P_2 = (-120,10,100)$ (Case 2) are projected onto a bi-cubic B-spline surface with initial parameter (0.9,0.6) and (0.1,0.6), respectively.

As shown in Table 1, in Case 1 [HW05] uses 6 iterations to converge, [LYY*09] uses 4 iterations, while our algorithm only uses 3 iterations. The processing time of our algorithm is 21.8% of [HW05] and 43.5% of [LYY*09]. In Case 2 [HW05] cannot converge after 10 iterations, [LYY*09] converges after 4 iterations, while our algorithm converges only after 3 iterations. The processing time of our algorithm is 29.2% of [HW05] and 50.7% of [LYY*09].

Example 2. We project point $P = (150,200,252)$ onto a bi-cubic B-spline surface with sharp features, and $q_0 = (0.2,0.6)$ (see Figure 4). Table 2 shows the iteration steps of these algorithms.

[HW05] converges after 552634 iterations. This is because the initial projection point lies in the sharp featured region, leading to a very small osculating circle used by [HW05]. The iteration oscillates, and can hardly move beyond this region. This case always occurs at the special point whose curvature is relatively much bigger than its neighboring region.

[LYY*09] fails to converge. In the second iteration the parametric increment is $(1.2,-3.0)$. This makes the parameter run out of the parametric domain of the surface ($(0\sim 1)\times(0\sim 1)$). As suggested in [LYY*09], we draw it back to the nearest parametric domain boundary (1,0). However, after a few iterations, it runs out again. [LYY*09] cannot converge after 1000000 iterations. The reason is similar to [HW05], where the sharp featured region leads to a very small torus used by [LYY*09], resulting in unstable estimations of the next projection point.

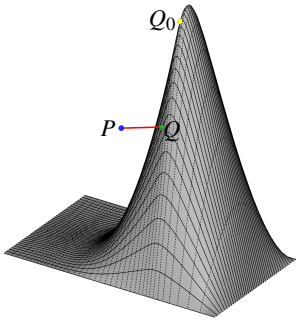


Figure 4: *Illustration of Example 2. The blue point P, the yellow point Q_0 and the green point Q are the test point, the initial projection point and the exact projection point. The control points, u knot vector and the v knot vector of the bi-cubic B-spline surface are* $\{(0,0,0),(0,90,0),(0,110,0),(0,200,0),(90,0,0),(110,110,600),(110,90,600),(90,200,0),(110,0,0),(90,110,600),(90,90,600),(110,200,0),(200,0,0),(200,90,0),(200,110,0),(200,200,0),(290,0,0),(310,110,0),(310,90,0),(290,200,0),(310,0,0),(290,110,0),(290,90,0),(310,200,0),(350,0,0),(350,90,0),(350,110,0),(350,200,0)\}$, $(0,0,0,0,0.25,0.5,0.75,1,1,1,1)$ *and* $(0,0,0,0,1,1,1,1)$.

Table 3: *Statistic data for Example 3.*

Methods	Correct solutions	Worst iterations	Average iterations	CPU time (ms)
H&W	173217	77	27.93	40986.53
Liu et al.	214931	12	4.17	10426.70
Ours	235670	7	3.13	2984.03

With the help of our local biarc approximation, approximation region is enlarged, and iterations can "jump" away from the special point and converge to the correct projection point only in 7 iterations though with "bad" initial value.

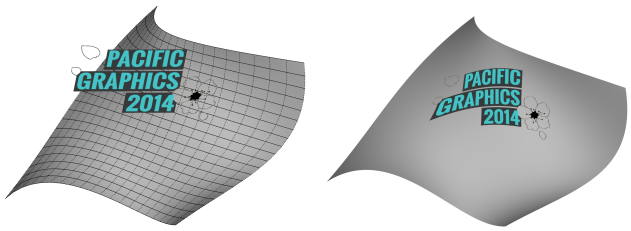
Example 3. We project 235670 points sampled from the logo of "Pacific Graphic 2014" onto a smooth surface (see Figure 5). The average initial value error is 7.26×10^{-02}. Table 3 shows that, our algorithm finds all correct solutions, while the successful ratios of [HW05] and [LYY*09] are

Table 1: *Convergence comparisons for Example 1.*

Method	Step	1	2	3	4	5	6	CPU time (ms)
Case 1								
H&W	Δu	-3.4×10^{-02}	-4.3×10^{-03}	3.8×10^{-05}	-5.1×10^{-06}	9.0×10^{-08}	-1.2×10^{-08}	0.17
	Δv	-4.8×10^{-02}	6.5×10^{-03}	2.3×10^{-04}	7.3×10^{-08}	5.4×10^{-07}	2.0×10^{-10}	
Liu et al.	Δu	-4.3×10^{-02}	3.9×10^{-03}	1.9×10^{-05}	5.1×10^{-10}			0.085
	Δv	-4.5×10^{-02}	3.8×10^{-03}	4.9×10^{-06}	1.6×10^{-10}			
Ours	Δu	-3.7×10^{-02}	-9.6×10^{-04}	-1.7×10^{-05}				0.037
	Δv	-4.7×10^{-02}	5.2×10^{-03}	1.2×10^{-05}				
Case 2								
H&W	Δu	3.1×10^{-02}	-9.4×10^{-03}	7.0×10^{-03}	-4.9×10^{-04}	7.8×10^{-04}	-1.5×10^{-07}	0.13
	Δv	2.9×10^{-02}	3.8×10^{-02}	1.4×10^{-03}	5.5×10^{-03}	5.1×10^{-06}	2.3×10^{-06}	
Liu et al.	Δu	2.4×10^{-02}	5.1×10^{-03}	2.0×10^{-04}	3.0×10^{-7}			0.075
	Δv	7.2×10^{-02}	1.6×10^{-03}	5.7×10^{-05}	8.0×10^{-8}			
Ours	Δu	3.1×10^{-02}	-1.1×10^{-03}	-1.9×10^{-05}				0.038
	Δv	5.0×10^{-02}	2.4×10^{-02}	-3.6×10^{-04}				

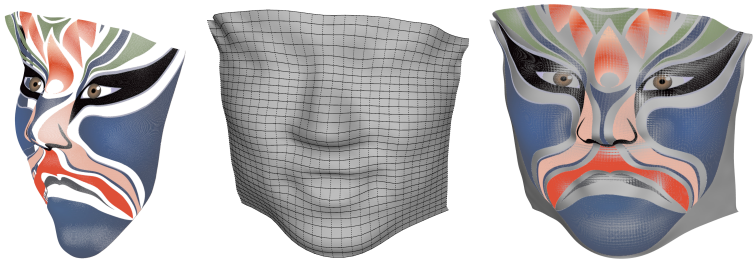
Table 2: *Convergence comparisons for Example 2.*

Method	Step	1	2	3	4	5	6	7	CPU time (ms)
H&W	Δu	0.074	0.086	−0.11	0.04	−0.13	0.23	−0.14	Fail
	Δv	0.36	−0.16	0.029	−0.053	0.11	−0.047	−0.054	
Liu et al.	Δu	−0.050	1.2	−0.25	−0.35	0.29	−0.54	−0.26	Fail
	Δv	0.20	-3.0	0.74	0.27	0.13	0.86	−0.15	
Ours	Δu	0.058	−0.0091	−0.00078	0.0026	0.0020	0.00028	0.00013	0.051
	Δv	0.16	0.030	0.019	−0.0056	−0.00092	−0.0018	−0.000093	



(a) (b)

Figure 5: *Illustration of Example 3: point projections on a smooth surface. (a) the input surface and points sampled from the logo of "Pacific Graphic 2014"; (b) the projection result.*



(a) (b) (c)

Figure 6: *Illustration of Example 4: point projections on scanned human face surface.(a) the input points sampled from a Chinese Peking Opera mask; (b) the input scanned human face surface; (c) the projection result.*

Table 4: *Statistic data for Example 4.*

Methods	Correct solutions	Worst iterations	Average iterations	CPU time (ms)
H&W	21776	989	19.65	6959.66
Liu et al.	28293	76	4.32	3215.63
Ours	35520	36	3.18	1004.21

73.5% and 91.2%, even if there is no special points mentioned in Example 2 (all initial values are not too far from the exact projection points and there is no sharp features). The average number of iterations of our algorithm is also less than the other two algorithms. The processing time of our algorithm is 7.3% of [HW05] and 28.6% of [LYY*09].

Example 4. We project 35520 points sampled from a Chinese Peking Opera mask onto a complicated scanned human face surface (see Figure 6). In this example, the average initial value error is 1.95×10^{-02}. Table 4 shows that, our algorithm finds all correct solutions, while the successful ratios of [HW05] and [LYY*09] are 61.3% and 79.7%. The average number of iterations of our algorithm is also less than the other two algorithms. The processing time of our algorithm is 14.4% of [HW05] and 31.2% of [LYY*09].

5. Conclusion

We present a geometric iteration algorithm to compute projection and inversion of points onto 3D parametric surfaces. Our algorithm uses biarcs to approximate the surface locally, which achieves both higher precision and larger fitting region as compared to single-point approximation [HW05, LYY*09]. Given the same initial value, the next projection point estimated by our algorithm is remarkably closer to the exact projection point than traditional geometric algorithms. As a result, our algorithm converges faster and is less dependent on the initial value than them.

Acknowledgements

The research was supported by Chinese 973 Program (2010CB328001) and International Science & Technology Cooperation Program of China(2013DFE13120). The second author was supported by the NSFC (61035002, 61272235). The last author was supported by the NSFC (91315302, 61173077).

References

[BM92] BESL P. J., MCKAY N. D.: Method for registration of 3-d shapes. In *Robotics-DL tentative* (1992), International Society for Optics and Photonics, pp. 586–606. 1

[CCY08] CHUI K., CHIU W., YU K.: Direct 5-axis tool-path generation from point cloud input using 3d biarc fitting. *Robotics and Computer-Integrated Manufacturing 24*, 2 (2008), 270–286. 3

[CSY*07] CHEN X.-D., SU H., YONG J.-H., PAUL J.-C., SUN J.-G.: A counterexample on point inversion and projection for NURBS curve. *Computer Aided Geometric Design 24*, 5 (2007), 302. 1

[CYW*08] CHEN X.-D., YONG J.-H., WANG G., PAUL J.-C., XU G.: Computing the minimum distance between a point and a NURBS curve. *Computer-Aided Design 40*, 10 (2008), 1051–1054. 1

[Har99] HARTMANN E.: On the curvature of curves and surfaces defined by normalforms. *Computer Aided Geometric Design 16*, 5 (1999), 355–376. 1, 2

[HL93] HOSCHEK J., LASSER D.: *Fundamentals of Computer Aided Geometric Design*. A.K. Peters, 1993. 1, 2

[HW05] HU S.-M., WALLNER J.: A second order algorithm for orthogonal projection onto curves and surfaces. *Computer Aided Geometric Design 22*, 3 (2005), 251–260. 1, 2, 4, 6

[JC98] JOHNSON D. E., COHEN E.: A framework for efficient minimum distance computation. In *Proceedings - IEEE International Conference on Robotics and Automatio* (1998), vol. 4, pp. 3678–3684. 1

[JC05] JOHNSON D. E., COHEN E.: Distance extrema for spline models using tangent cones. In *Proceedings of Graphics Interface 2005* (2005), pp. 169–175. 1

[LT95] LIMAIEM A., TROCHU F.: Geometric algorithms for the intersection of curves and surfaces. *Computers & graphics 19*, 3 (1995), 391–403. 1

[LYY*09] LIU X.-M., YANG L., YONG J.-H., GU H.-J., SUN J.-G.: A torus patch approximation approach for point projection on surfaces. *Computer Aided Geometric Design 26*, 5 (2009), 593–598. 1, 2, 4, 6

[MH03] MA Y. L., HEWITT W.: Point inversion and projection for NURBS curve and surface: Control polygon approach. *Computer Aided Geometric Design 20*, 2 (2003), 79–99. 1

[OKL*10] OH Y.-T., KIM Y.-J., LEE J., KIM M.-S., ELBER G.: Efficient point projection to freeform curves and surfaces. In *Advances in Geometric Modeling and Processing*. Springer, 2010, pp. 192–205. 1

[PLH04] POTTMANN H., LEOPOLDSEDER S., HOFER M.: Registration without ICP. *Computer Vision and Image Understanding 95*, 1 (2004), 54–71. 1

[PT97] PIEGL L. A., TILLER W.: *The NURBS Book, second ed.* Springer-Verlag, Berlin, Heidelberg, New York, 1997. 1, 3

[PT01] PIEGL L. A., TILLER W.: Parameterization for surface fitting in reverse engineering. *Computer Aided Design 33*, 8 (2001), 593–603. 1

[PW96] PEGNA J., WOLTER F.-E.: Surface curve design by orthogonal projection of space curves onto free-form surfaces. *Journal of Mechanical Design 118*, 1 (1996), 45–52. 1

[Sel06] SELIMOVIC I.: Improved algorithms for the projection of points on NURBS curves and surfaces. *Computer Aided Geometric Design 23*, 5 (2006), 439–445. 1

[ŠFJ06] ŠÍR Z., FEICHTINGER R., JÜTTLER B.: Approximating curves and their offsets using biarcs and pythagorean hodograph quintics. *Computer-Aided Design 38*, 6 (2006), 608–618. 2

[SXSY14] SONG H.-C., XU X., SHI K.-L., YONG J.-H.: Projecting points onto planar parametric curves by local biarc approximation. *Computers & Graphics 38* (2014), 183–190. 1, 2, 3

[SYYL11] SONG H.-C., YONG J.-H., YANG Y.-J., LIU X.-M.: Algorithm for orthogonal projection of parametric curves onto B-spline surfaces. *Computer-Aided Design 43*, 4 (2011), 381–393. 1