Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Computer Aided Geometric Design 20 (2003) 79–99

COMPUTER AIDED GEOMETRIC DESIGN

www.elsevier.com/locate/cagd

Point inversion and projection for NURBS curve and surface: Control polygon approach

Ying Liang Ma *, W.T. Hewitt

Manchester Visualization Centre, University of Manchester, M13 9PL, UK

Received 12 May 2002; received in revised form 10 February 2003; accepted 19 February 2003

Abstract

This paper presents an accurate and efficient method to solve both point projection and point inversion for NURBS curves and surfaces. We first subdivide the NURBS curve or surface into a set of Bézier subcurves or patches. Based on the relationship between the test point and the control polygon of Bézier curve or the control point net of the Bézier patch, we extract candidate Bézier subcurves or Bézier patches and then find the approximate candidate points. Finally, by comparing the distances between the test point and candidate points, we are able to find the closest point. We improve its accuracy by using the Newton–Raphson method.

Keywords: Point projection; Point inversion; NURBS curve; NURBS surface

1. Introduction

Projecting a test point to a NURBS curve or surface finds the closest point on the curve or surface and point inversion finds the corresponding parameters (u) for the curve or (u, v) for the surface for this point. Both **point projection** and **point inversion** are common problems for interactively selecting a NURBS curve or surface and curve or surface fitting. When the user clicks somewhere near the curve or surface, we can find out which curve or surface has been selected based on the minimum distance between the nearest point on the curve or surface and the test point. In curve and surface fitting we need to calculate the tolerance, which is the minimum distance from the interpolate point to the desired curve or surface.

Much of the early work in this area comes from the robotics and computational geometry communities (Johnson and Cohen, 1998). Their work has often focus on the polygonal model for finding the

* Corresponding author.
E-mail address: ma@psy.gla.ac.uk (Y.L. Ma).

0167-8396/03/$ – see front matter
doi:10.1016/S0167-8396(03)00021-9

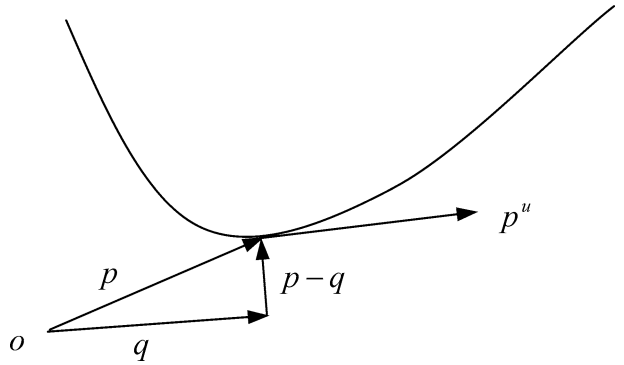


Fig. 1. Minimum distance between a point and a curve.

minimum distance between two geometric objects. Chin (1983) and Edelsbrunner (1982) both describe the $O(\log N)$ algorithms for finding the minimum distance between two convex polygons. However, all these algorithms involve presumably large number of tests on polygons and the result for the minimum distance is not precisely accurate enough for computer graphics and computer aided design communities. On the other hand, parametric model provide more accurate solutions.

In parametric model, Mortensen (1985) gave a numerical approach this problem. For the calculation of the minimum distance between the test point and the curve, he needed to find a vector $(p-q)$ from the point p to the curve $q(u)$ that is perpendicular to the tangent vector p^u at p. Fig. 1 shows the vector geometry. Mathematically, he expressed the required conditions as

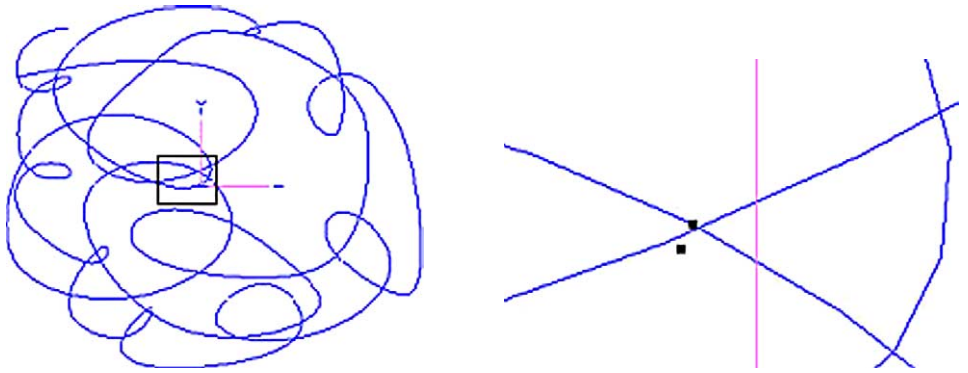
$$d_{\min} = |p-q| \quad \text{when } (p-q)\times n = 0.$$

He chose to use the Newton–Raphson method to the find the roots of the polynomial $(p-q)\times n = 0$. However, he needed to find a good initial value for the Newton–Raphson method. Similar to the curve, finding the minimum distance between a point and a surface is finding the minimum distance between a point and a plane. Let the point be denoted by q and the plane by kn, n is the unit normal vector. Let p denote the point on the plane closest to q. Then we have $d_{\min} = |p-q|$, where p must satisfy $(p-q)\times n = 0$. Finally, he also needed the Newton–Raphson method to find the roots. Limaiem (1995) has presented another approach to find the minimum distance to convex parametric curves and surfaces. Lin (1995) provided the approach for finding the minimum distance for concave surfaces. Both approaches use Newton–Raphson method to find the roots for some distance equations. Therefore, both finding the minimum distance between a point and a curve and between a point and a surface need a good initial value for achieving the convergence result.

However this initial value is hard to obtain due to the complexity of the NURBS curve or surface's shape (Piegl and Tiller, 1995). Furthermore, Newton–Raphson method may give the wrong answer when projecting a point near the intersection point of the NURBS curve (Fig. 2) and Newton–Raphson method fails quite often especially for the points near the boundaries of the surface (Piegl and Tiller, 2001).

Piegl and Tiller (2001) recently provide another method to solve the point projection problem for NURBS surface. The algorithm consists three steps:

- Decompose the NURBS surface into quadrilaterals.
- Project the test point onto the closest quadrilateral.



(a) (b)

Fig. 2. (a) Cubic NURBS curve (73 control points). (b) Wrong point projection.

- Recover parameters from the closest quadrilateral.

Obviously, decomposing the NURBS surface into quadrilaterals and finding the closest quadrilateral are very expensive.

The purpose of this article is to present a novel method for point projection and point inversion which provides a good initial value for the Newton–Raphson method to reduce the computation of the algorithm and increase the stability of the recursive function for the Newton–Raphson method.

2. Outline of algorithm

2.1. Algorithm for NURBS curve

Our approach consists of three stages: *subdivision of the NURBS curve*, *control polygon detection* and *the relationship between the test point and the Bézier subcurve*. In the first stage, we decompose the NURBS curve into its piecewise Bézier form (see Section 3). In Section 4, we give the algorithm to find the 2D/3D simple and convex control polygon of the Bézier subcurve. In Section 5, we analyse the relationship between the test point and control polygon and in Section 6 we give the overall algorithm.

2.2. Algorithm for NURBS surface

Similar to the algorithm for NURBS curve, first, we decompose the surface into a set of Bézier patches. Then we check every control point net of the Bézier patch is a valid control point net or not, which will be discussed further later. After obtaining the valid control point net, we analyse the relationship between the test point and the Bézier patch and discard the Bézier patch which the closest point lies on one of the four boundary curves (Fig. 3). Finally, we project the test point to the candidate Bézier patches to obtain the candidate points and select the closest one as the result. The accuracy of the closest point can be improved by using the Newton–Raphson method.

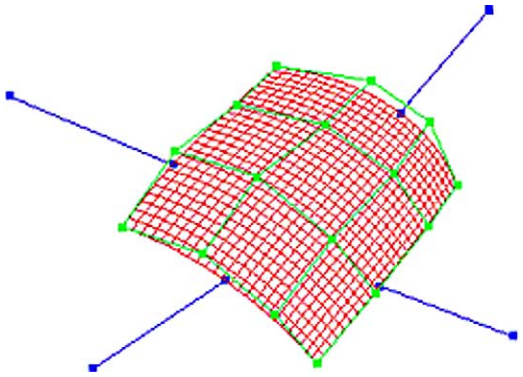


Fig. 3. Discarded Bézier patch.

3. Subdivision of the NURBS curve and surface

The objective is to obtain a set of Bézier subcurves from the original NURBS curve and also obtain a set of Bézier patches from the original NURBS surface.

3.1. Definition of NURBS curve and surface

A pth-degree NURBS curves is defined by

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i}. \tag{1}$$

Where $\{P_i\}$ are the control points for NURBS curves, $\{u\}$ is the *knot vector* from a to b, the $\{w_i\}$ are the *weights*, and $\{N_{i,p}(u)\}$ are the pth B-spline basis functions defined on the *knot vector*.

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\}.$$

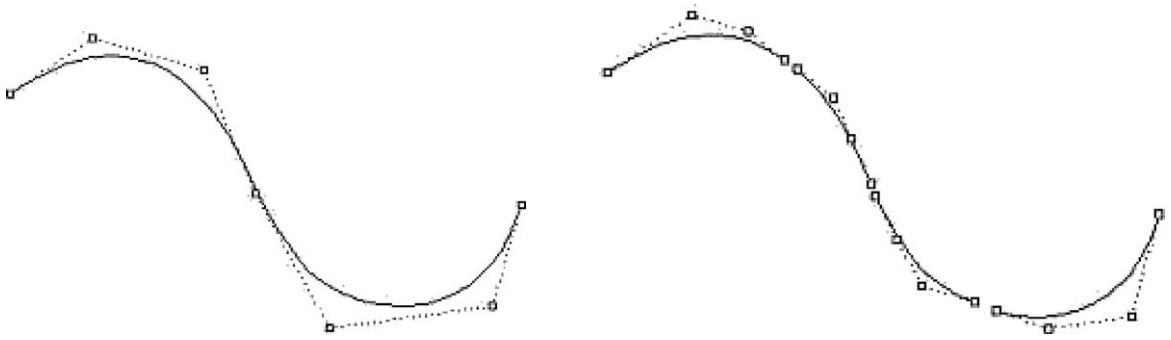
A NURBS surface of degree p in u direction and degree q in v direction is defined by:

$$S(u, v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j} P_{i,j}}{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j}}. \tag{2}$$

Where the $\{P_{i,j}\}$ is the control net, $0 \leqslant u, v \leqslant 1$, the $\{w_{i,j}\}$ are the *weights*, and $N_{i,p}(u)$ and $N_{j,q}(u)$ are non-rational B-spline basis functions defined on the knot vectors:

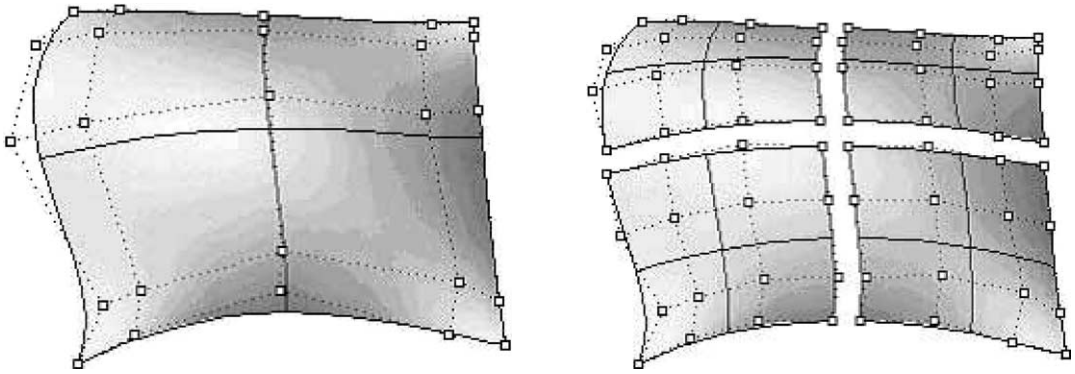
$$U = \{\underbrace{0, \ldots, 0}_{p+1}, u_{p+1}, \ldots, u_{r-p-1}, \underbrace{1, \ldots, 1}_{p+1}\},$$
$$V = \{\underbrace{0, \ldots, 0}_{q+1}, v_{q+1}, \ldots, v_{s-p-1}, \underbrace{1, \ldots, 1}_{q+1}\}.$$

Where $r = n + p + 1$ and $s = m + q + 1$.



(a) (b)

Fig. 4. (a) The original cubic NURBS curve. (b) 4 Bézier subcurves after decomposition.



(a) (b)

Fig. 5. (a) The original NURBS surface. (b) 4 Bézier patches after decomposition.

3.2. Splitting the NURBS curve and surface

Boehm (1980) and Cohen and others (1980) (the Oslo algorithm) have described the original subdivision techniques using knot insertion. A simple of proof of the Oslo algorithm is given by Prautzsch (1984) and a more efficient way of using the Oslo algorithm is given by Lyche and Morken (1986).

Given a NURBS curve of degree p, defined by Eq. (1), and an interior knot value u_i $(p+1 \leqslant i \leqslant m-p-1)$ inserting u_i with multiplicity p splits the curve into two parts.

Given NURBS surface of degree p and q, defined by Eq. (2), and two interior knot values u_i $(p+1 \leqslant i \leqslant r-p-1)$ and v_j $(q+1 \leqslant j \leqslant s-q-1)$, inserting u_i with multiplicity p and v_j with multiplicity q subdivides the surface into four NURBS sub-surfaces.

3.3. Decomposing the NURBS curve and surface into its piecewise Bézier form

For a given NURBS curve of degree p, the Bézier subcurves are obtained by inserting each interior knot $(u_{p+1}, \ldots, u_{m-p-1})$ until it has multiplicity p.

For a given NURBS surface of degree p and q, defined by Eq. (2), the Bézier patches are obtained by inserting each interior knot in U until it has multiplicity p and then inserting each interior knot in V until it has multiplicity q.

4. Control polygon and control point net detection

For NURBS curve, before we can establish the relationship between the test point and the Bézier subcurve, we need to classify the control polygons to simplify the problem. As we know, polygons can be divided into *simple* and *non-simple* polygons. Simple means no crossing edge. As shown in Fig. 6, polygon (5) is a non-simple polygon. Furthermore, simple polygons can be grouped as *convex* and *concave* polygons. Polygons (1), (2), (3) are convex and (4) is concave. Now, we define the *valid* control polygon as the simple and convex polygon in 2D. In order to generalize the definition to 3D, we give a fast way to check whether the control polygon is valid or not both in 2D and 3D.

For NURBS surface, we only analyse the relationship between the test point and the valid control point net. After the definition of the valid control polygon, we can specify the valid control point net, as every polygon both in U and V parameter direction is valid.

4.1. Fast valid control polygon detection algorithm

We give a fast way to detect whether the control polygon is simple and convex, or not, by checking the *dot product* results of two vectors. As shown in Fig. 7(a), for given a control polygon of Bézier curve of degree p ($p > 2$), we can detect the sign of dot product $R = \overrightarrow{V_1P_i} \bullet \overrightarrow{V_2P_n}$ ($i < (n/2)$) or $R = \overrightarrow{V_1P_i} \bullet \overrightarrow{V_2P_0}$ ($i \geqslant (n/2)$) to determine whether vertex P_i is in the "convex" direction or not. If the result of dot product is greater than zero, then this vertex is in "concave" direction. Note that edge P_0P_n is the chord of Bézier subcurve and vertices $P_0, P_1, \ldots, P_{n-1}, P_n$ are the control points. V_1 and V_2 are the points projecting from the vertex P_i and one of the endpoints: P_n ($i < (n/2)$) or P_0 ($i \geqslant (n/2)$) to line segment $P_{i-1}P_{i+1}$, respectively. If the dot product R is negative, we move to the next vertex and repeat this procedure until we find one is positive or finish detecting all vertices except P_0 and P_n. If all results are negative then the control polygon is a simple and convex one.

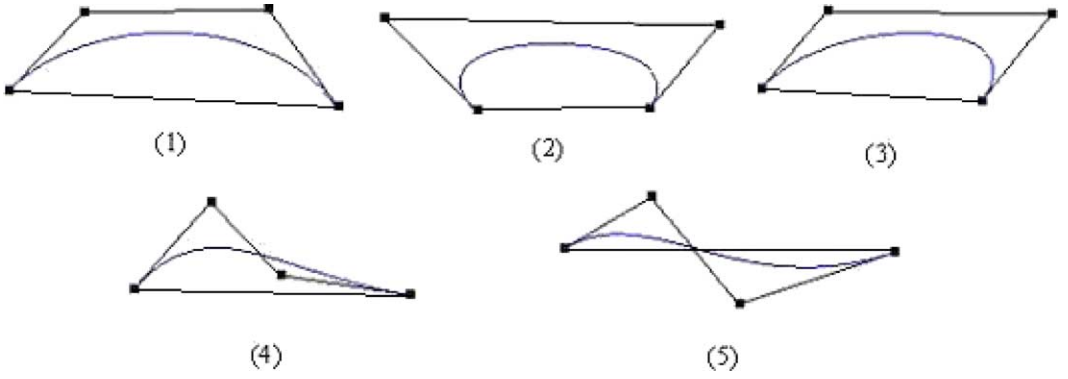


Fig. 6. Type of control polygons of 2D cubic Bézier subcurve.

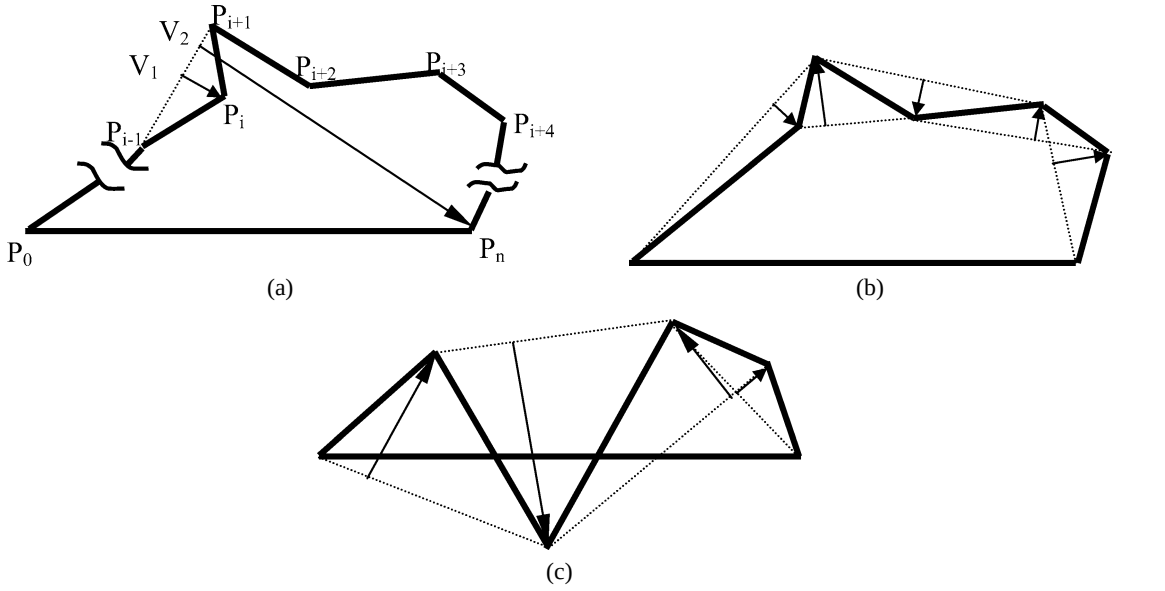


Fig. 7. (a) Dot product of two vectors. (b) Vertex direction for concave polygon. (c) Vertex direction for non-simple polygon.

This algorithm also works for detecting non-simple polygons. As shown in Fig. 7(c), although this algorithm does not tell you the vertex direction properly, it does tell you it is not a convex polygon no matter if it is a concave or non-simple one. This algorithm can generalise to 3D. We will call the 3D simple and convex polygon which satisfies the dot product conditions as previously discussed a 3D valid control polygon.

In summary, the simple and convex polygon (valid polygon) detection algorithm is shown as follows:

Algorithm 1. *Is_Valid_Polygon* {Detecting the simple and convex polygon}
Input: The control polygon P of Bézier subcurve. n is the highest index in control polygon.
Output: the result of detection.
begin
for $i = 1$ to $i < n$ **by** $i{+}{+}$ **do**
begin
compute projection vector $\overrightarrow{V_1P_i}$;
if $i < (n/2)$ **then**
compute projection vector $\overrightarrow{V_1P_n}$;
$R = \overrightarrow{V_1P_i} \bullet \overrightarrow{V_1P_n}$; {$R$ is the result of dot product}
else
compute projection vector $\overrightarrow{V_1P_0}$;
$R = \overrightarrow{V_1P_i} \bullet \overrightarrow{V_1P_0}$; {$R$ is the result of dot product}
end if
if $R > 0$ **then return** FALSE; {The polygon is not a simple or convex one}
end if
end {End of the loop for}

return TRUE; {It is a simple and convex polygon}
End of Algorithm 1.

Finally, we define the control point net whose polygons both in U and V parameter direction are valid as the valid control point net. The detection algorithm is given as follows:

Algorithm 2. *Is_Valid_CP_Net* {Detecting the valid control point net}
Input: The control point net of Bézier patch. n is the highest index of control points in U direction. m is the highest index of control points in V direction.
Output: the result of detection.
begin
for $i = 0$ to $i < m$ **by** $i{+}{+}$ **do**
{Detect every control polygon in U direction}
begin
generate a control polygon P;
if (*Is_Valid_Polygon*(P) **return** FALSE) **then**
return FALSE; {control point net is not valid}
end if
end {End of loop for}
for $i = 0$ to $i < n$ **by** $i{+}{+}$ **do**
{Detect every control polygon in V direction}
begin
generate a control polygon P;
if (*Is_Valid_Polygon*(P) **return** FALSE) **then**
return FALSE; {control point net is not valid}
end if
end {End of loop for}
return TRUE;
End of Algorithm 2.

5. The relationship between the test point and Bézier subcurve or Bézier patch

For curve, after decomposition, we obtain a set of Bézier subcurves. When detecting the valid control polygon, if we do not get a desired control polygon, we continue to subdivide the Bézier subcurve until we get a valid control polygon or the control polygon is flat enough. For a valid polygon, we analyse the relationship between the test point and the valid control polygon of the Bézier subcurve.

For surface, after decomposition, we obtain a set of Bézier patches. When detecting the valid control point net, if we do not get a desired control point net, we continue to subdivide the Bézier patch until we get a valid one or the control point net is flat enough.

For a valid control point net, we analyse the relationship between the test point and the valid control point net of the Bézier patch. As shown in Fig. 8, for given a valid control point net, we can extract valid control polygons both in U direction and V direction. Therefore, we can analyse the relationship between each valid control polygon and the test point to generalize the relationship between the valid

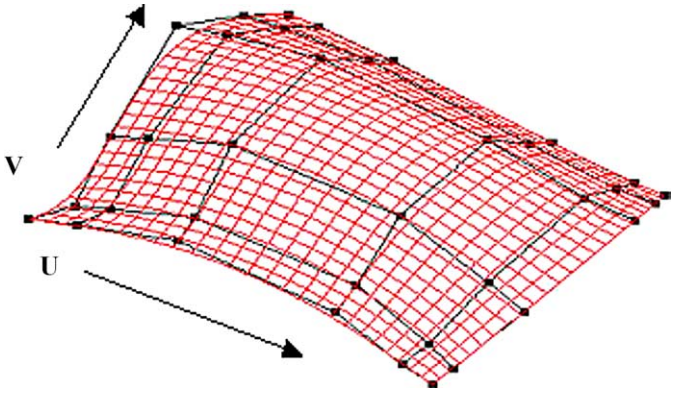


Fig. 8. Control point net.

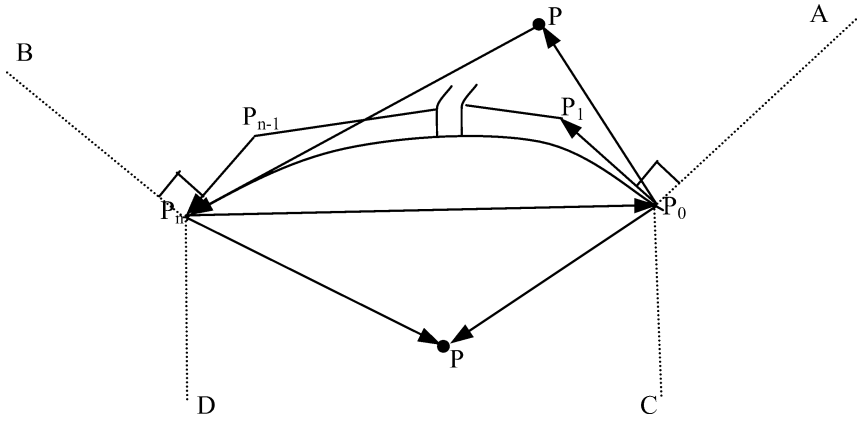


Fig. 9. Conditions for 2D Bézier curve.

control point net and the test point. We have the conclusion of the relationship between the valid control polygon (Bézier curve) and the test point as follows:

Lemma 1. *As shown in Fig.* 9*, suppose that a* n $(n > 1)$ *degree* 2D *Bézier curve has a valid control polygon* $P_0, \ldots, P_n$ *and a test point* P*. We have four dot products* $R_1 = \overrightarrow{P_0P} \bullet \overrightarrow{P_0P_1}$*,* $R_2 = \overrightarrow{PP_n} \bullet \overrightarrow{P_{n-1}P_n}$*,* $R_3 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_nP}$ *and* $R_4 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_0P}$*. If* $R_1 < 0$ *or* $R_2 < 0$ *and* $R_3 * R_4 > 0$*, then the nearest point must be one of endpoints* $(P_0$ *or* $P_n)$*.*

Proof. It is obvious that the dot product R_1 and R_2 will be positive if the test point is within the area AP_0P_nB and the multiplication $R_3 * R_4$ will be negative if the test point is within the area CP_0P_nD.

According to the formula (Eq. (3)) of the derivative of a Bézier curve, we can obtain the formulas for the end derivatives (Eq. (4)).

$$C'(u) = n\sum_{i=0}^{n-1} B_{i,n-1}(u)(P_{i+1} - P_i), \tag{3}$$

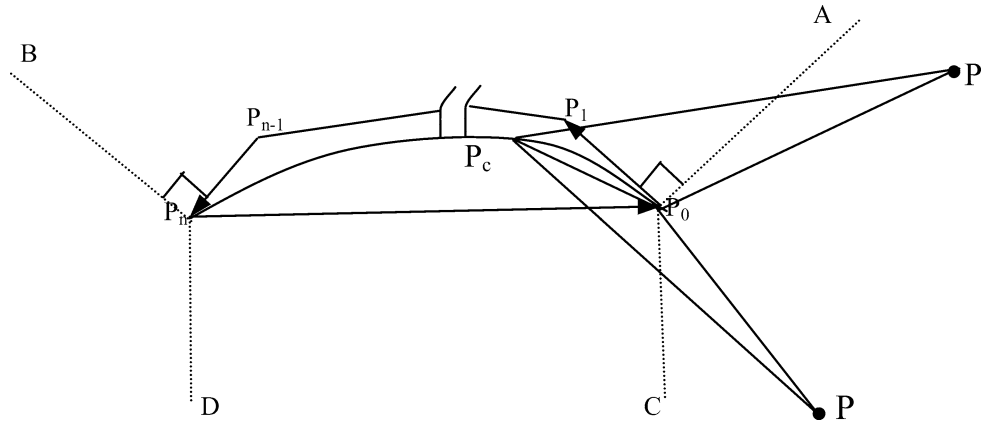


Fig. 10.

$$\begin{cases} C'(0) = n(P_1 - P_0), \\ C'(1) = n(P_n - P_{n-1}), \end{cases} \tag{4}$$

where $B_{i,n-1}(u)$ is the Bézier basis function and $P_0, \ldots, P_n$ is the control points.

For an n degree Bézier curve. Notice that, in Fig. 10, $C'(0)$ is the vector $\overrightarrow{P_0P_1}$ and $C'(1)$ is the vector $\overrightarrow{P_{n-1}P_n}$. We assume that P_c is any point on the Bézier subcurve. If the control polygon is simple and convex, according to the strong convex hull property, P_c must be inside the control polygon. Therefore, $\angle PP_0P_c$ must be larger than 90° if P is outside of area AP_0P_nB and area CP_0P_nD. Furthermore, according to the property of the triangle, $\angle PP_0P_c$ is larger than $\angle P_0P_cP$, therefore, $|PP_c|$ is larger than $|PP_0|$. It is obvious that P_0 is the nearest point of P.

We can extend this rule to 3D valid control polygon of Bézier curve. Although in 3D space the valid control polygon sometimes does not lie on the same plane, we can set the control polygon plane as the plane constructed from the first two edges of the control polygon. As shown in Fig. 11, we also have $|P_c'P'| > |P_0P'|$ (P' and P_c' are the projection points from P and P_c to control polygon plane respectively and P_c is a any point on the Bézier curve.)

Also

$$|P_cP| = \sqrt{|P_c'P'|^2 + (|PP'| + |P_cP_c'|)^2} \quad \text{(refer to Fig. 12) } |P_0P| = \sqrt{|P_0P'|^2 + |PP'|^2}.$$

Therefore $|P_cP| > |P_0P|$.

Theorem 1. *Suppose that a n ($n > 1$) degree* 2D/3D *Bézier subcurve has a valid control polygon $P_0, \ldots, P_n$ and a test point P is in the* 3D *space. Define four dot products $R_1 = \overrightarrow{P_0P} \bullet \overrightarrow{P_0P_1}$, $R_2 = \overrightarrow{PP_n} \bullet \overrightarrow{P_{n-1}P_n}$, $R_3 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_nP}$ and $R_4 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_0P}$. If $R_1 < 0$ or $R_2 < 0$ and $R_3 * R_4 > 0$, then the nearest point* must *be one of endpoints (P_0 or P_n).*

For curve, we analyse the relationship between the test point and a valid control polygon. We do four dot products, which are described in Theorem 1. If this case does not satisfy the conditions in Theorem 1, the nearest point must be one of the endpoints. Further more we can discard this Bézier subcurve. In summary, we give the algorithm as follows:

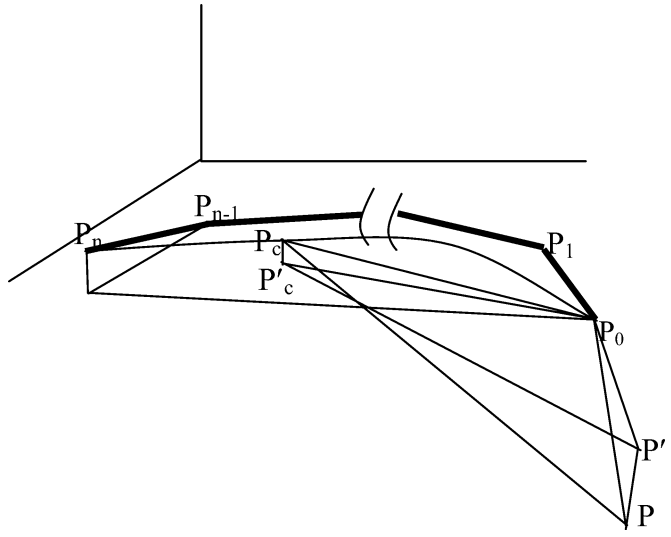


Fig. 11.

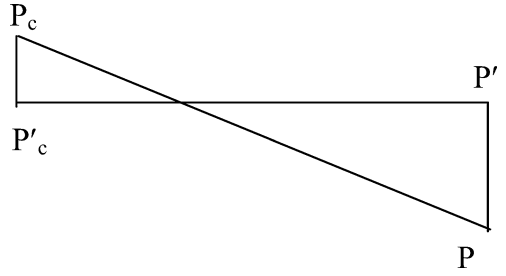


Fig. 12.

Algorithm 3. *Point_Nearest_Bézier_Curve* {Detecting whether the nearest point is one of endpoints or not according to the result of four dot products}

Input: The control polygon P of Bézier curve. n is the highest index in control polygon.

Output: the result of detection.

begin

$R_1 = \overrightarrow{P_0P} \bullet \overrightarrow{P_0P_1}$, $R_2 = \overrightarrow{PP_n} \bullet \overrightarrow{P_{n-1}P_n}$

$R_3 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_nP}$, $R_4 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_0P}$

if $R_1 < 0$ or $R_2 < 0$ and $R_3 * R_4 > 0$

then return FALSE;

else return TRUE;

end if

End of Algorithm 3.

Then, for surface, we consider the relationship between the test point and a valid control point net. We also do four dot products with every polygon in U direction and V direction. If every polygon in U direction or every polygon in V direction does not satisfy the conditions of dot product describing in

Theorem 1, the nearest point must be on one of the four boundary curves. Furthermore we can discard this Bézier patch.

In summary, we give the algorithm as follows:

Algorithm 4. *Point_Nearest_Bézier_Patch* {Detecting whether the nearest point is the point on the boundary curves}

Input: The control point net of Bézier patch. n is the highest index of control points in U direction. m is the highest index of control points in V direction.

Output: the result of detection.

```
begin
  Flag ← FALSE;
  for i = 0 to i < m by i ++ do
      {Detect every control polygon in U direction}
    begin
      generate a control polygon P;
      if (Point_Nearest_Bézier_Curve(P) return TRUE) then
        Flag ← TRUE;
        break;
      end if
    end {End of loop for}
  if Flag == FALSE then
    return FALSE; {the nearest point is the point on the boundary curves}
  end if
  Flag ← FALSE;
  for i = 0 to i < n by i ++ do
      {Detect every control polygon in V direction}
    begin
      generate a control polygon P;
      if (Point_Nearest_Bézier_Curve(P) return TRUE) then
        Flag ← TRUE;
        break;
        end if
      end {End of loop for}
    if Flag == FALSE then
      return FALSE; {the nearest point is the point on the boundary curves}
    end  if
    return TRUE; {the nearest point is the point on the Bézier patch}
End of Algorithm 4.
```

6. Find the closest point on the NURBS curve

By extracting the Bézier subcurves as candidates, we subdivide these candidate subcurves recursively until the control polygon is flat enough or reaching recursion limit. By “flat enough”, we mean that

the control polygon is close enough to a straight line so that we can approximate the nearest point (candidate point) in that region by calculating a projection vector from the test point to the straight line. The algorithm of finding the candidate points from the NURBS curve is given as follows.

Algorithm 5. Find the candidate points for projecting a point to a NURBS curve.
Nearest_Candidate_Points_Curve(P, n, U, m, *Valid*, *pt*)

```
        P: array point of control points.
        n: The highest index in the control points.
        U: array pointer of knot vector.
        m: The highest index in the knot vector.
        Valid: flag variable for valid polygon.
                (TRUE is valid. FALSE is invalid)
        pt: 2D/3D test point.
Begin
  if(Curve is a Bézier curve)
    if (Valid) then
      if (Point_Nearest_Bézier_Curve return FALSE) then
        return no candidate point was found;
      else
        if(control polygon is flat enough or recursive limit reached) then
          return the candidate point on the Bézier subcurve;
        end if
      end if
    else{Valid is FALSE}
      if(Is_Valid_Polygon return TRUE) then
        Valid = TRUE;
      end if
    end if
    Subdivide the curve at midpoint of knot vector;
    Nearest_Candidate_Points_Curve(left half);
    Nearest_Candidate_Points_Curve(right half);
end
End of Algorithm 5.
```

We start the recursive function with *Valid* set to FALSE. After the termination of this function, we select the nearest point from candidate points and improve its accuracy by applying the Newton–Raphson method. Section 10 gives some examples of projecting a set of points to the NURBS curve both in 2D and 3D.

7. Find the closest point on the NURBS surface

By extracting the Bézier patches as candidates, we subdivide these candidate patches recursively until the control point net is flat enough or reaching recursion limit. By “flat enough”, we mean that the control

point net is close enough to a plane so that we can approximate the nearest point (candidate point) in that region by projecting the test point to the approximate plane. The algorithm of finding the candidate points from the NURBS surface is given as follows:

Algorithm 6. Find the candidate points for projecting a point to a NURBS surface.

```
Nearest_Candidate_Points_Surface(pSur, Dir, Valid, pt)
        pSur: The pointer of NURBS surface object.
        Dir: The direction for splitting. (TRUE for U direction and FALSE for V direction.)
        U: array pointer of knot vector.
        Valid: flag variable for valid control point net.
                (TRUE is valid. FALSE is invalid)
        pt: 3D test point.
Begin
   if(Surface is a Bézier patch)
      if (Valid is TRUE) then
         if (Point_Nearest_Bézier_Patch return FALSE) then
            return no candidate point was found;
         else
            if(control point net is flat enough or recursive limit reached) then
            return the candidate point on the Bézier patch;
            end if
         end if
      else{Valid is FALSE}
            if(Is_Valid_CP_Net return TRUE) then
               Valid = TRUE;
            end if
         end if
      if(Dir is TRUE)
         Split the surface at midpoint of U knot vector;
      else
         Split the surface at midpoint of V knot vector;
      end if
      Nearest_Candidate_Points_Surface(pSur1, Dir, Valid, pt);
      {pSur1 is one half surface in the U or V direction.}
      Nearest_Candidate_Points_Surface(pSur2, Dir, Valid, pt);
      {pSur2 is the other half surface in the U or V direction}
end
End of Algorithm 6.
```

8. Boundary conditions of NURBS surface

After the termination of the recursive function which is described in last section, we select the closest point from the candidate points as well as the points projecting from the test point to the boundary curves.

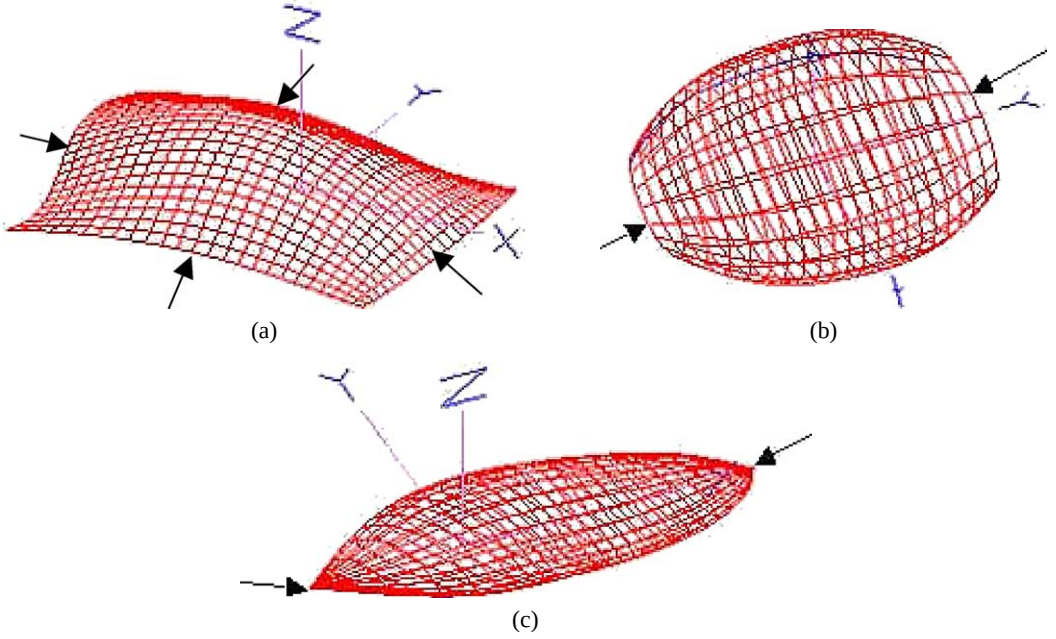


Fig. 13. Boundary curves of the NURBS surface. (a) Unclosed surface with four boundary curves. (b) Closed surface with two boundary curves. (c) Closed surface with two boundary points.

If a NURBS surface is not closed both in the U direction and V direction, it has four boundary curves. On the other hand, if the surface is closed in u direction or v direction, it has two boundary curves. In some cases, it has only two "boundary points" as shown in Fig. 13. For an unclosed NURBS surface, we have four projection points which are projected from the test point to the four boundary curves. For a closed NURBS surface, we also have two projection points. From the projection points and candidate points, we can find the closest one as the solution for the point projection for the NURBS surface.

9. The Newton–Raphson method for NURBS surface

Newton–Raphson method is used to improve the accuracy of the closest point. Piegl and Tiller (1995) used Newton–Raphson method to minimize the distance between the test point and the whole NURBS surface. However after some tests, we find Newton–Raphson method occasionally still gives the wrong answer even with a quite good initial value when applying Newton–Raphson method on the whole NURBS surface. Therefore, instead of applying on the whole NURBS surface, we apply Newton–Raphson method on the quadrilateral which is a "flat enough" Bézier patch.

10. Examples, comparison and results

For illustrative purposes we present three pictures of points projected to a 2D cubic NURBS curve (Fig. 15), a 2D NURBS curve of degree 5 (Fig. 16) and a 3D cubic NURBS curve (Fig. 17) respectively.

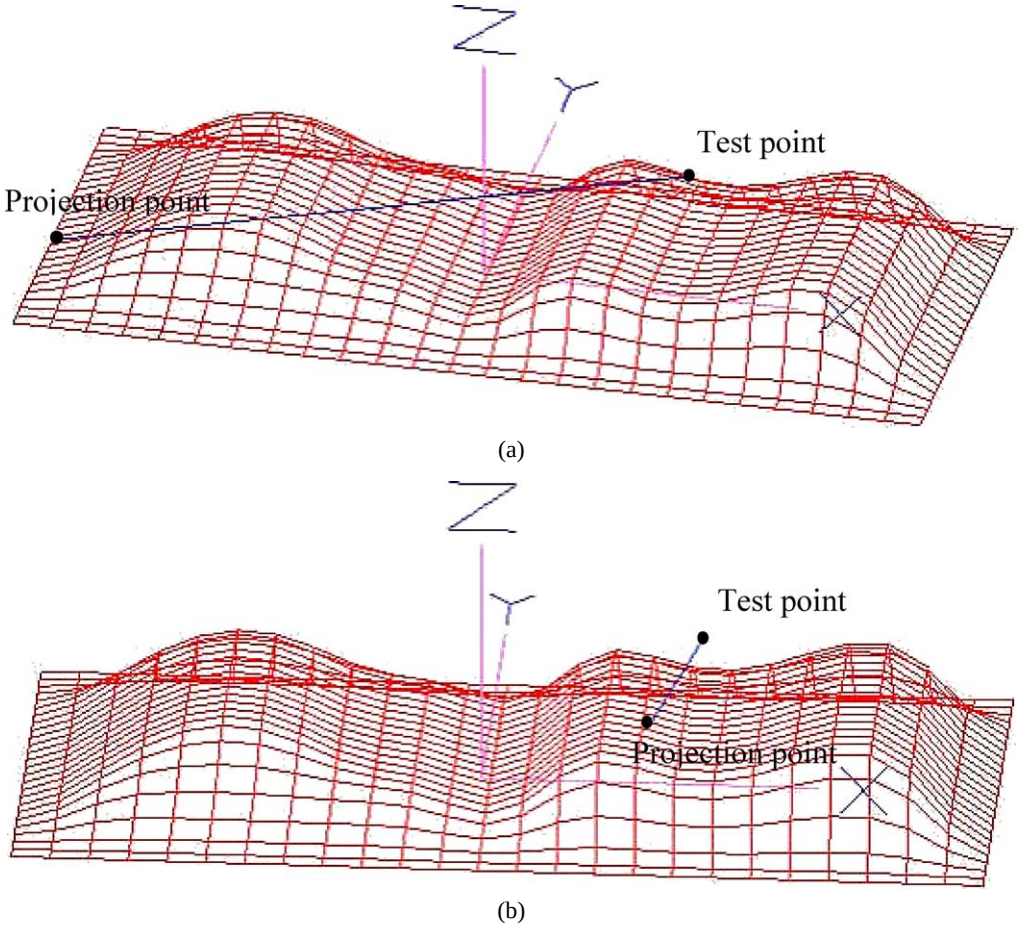


Fig. 14. (a) Wrong point projection (applying Newton–Raphson method to the whole NURBS surface). (b) Right point projection (applying Newton–Raphson method to the Bézier patch).

Finally, we show the result of projecting points to the complex curve (Fig. 18). We also compared our method with Piegl's method (Piegl and Tiller, 1995) in both efficiency and accuracy. Piegl's method is implemented in NLib (A commercial NURBS software library). All our examples were implemented with Visual $C++$ 6.0 with Windows98 in the same compatible personal computer with Pentium II 400 CPU chip and 64 M memory. Table 1 lists iteration times, the CPU time, candidate points and the accuracy. All data are the average of projecting 10 equally spaced points to the NURBS curve. The accuracy evaluated as the zero-cosine, which is the dot-product between the closest point's unit normal and the unit vector from the closest point to the test point. From the result, it can be noted that our method becomes more efficient when points are projected to a more complex curve.

We also present some examples of points projected to the NURBS surface. Fig. 19 gives the result of projecting a set of points from a straight line to the NURBS surface. Figs. 20 and 21 present the results of projecting two isocurves which are created and offset from the NURBS surface in U direction and V direction respectively. Fig. 22 gives the result of projecting a set of points from a vertical line to the NURBS surface. Finally, Fig. 23 presents the result of projecting a set of points to a closed NURBS

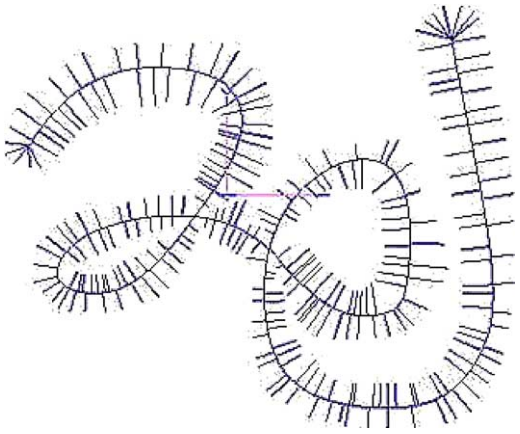


Fig. 15. Point projection for a 2D cubic NURBS curve (degree = 3, 26 control points).

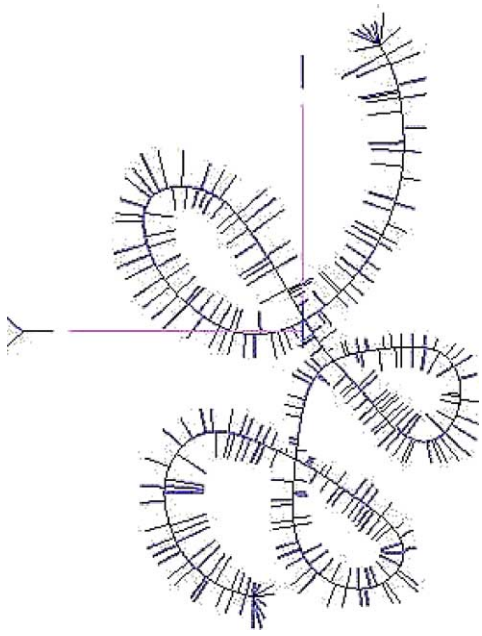


Fig. 16. Point projection for a 2D NURBS curve (degree = 5, 40 control points).

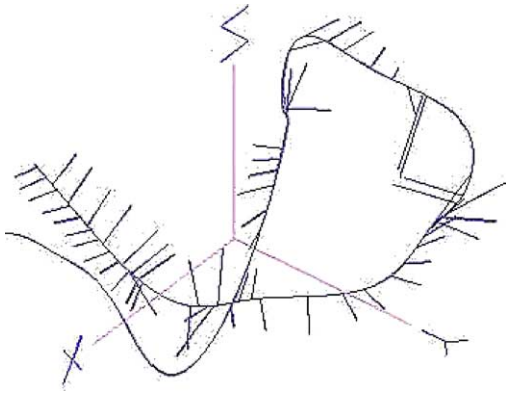


Fig. 17. Point projection for a 3D cubic NURBS curve (degree = 3, 20 control points).

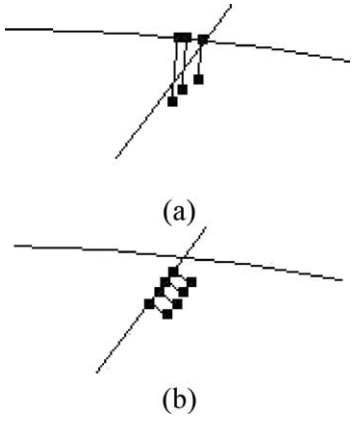


Fig. 18. (a) Wrong point projection using Nlib function (1000 iteration times, Piegl's method). (b) Correct project using our method (the same curve as shown in Fig. 2).

surface. From Table 2, it can be noted that there is a significant difference in efficiency between our method and Piegl's method, since they decompose the surface into quadrilaterals and we decompose the surface into the form of Bézier patches.

Table 1
The results of point projection for NURBS curve

Curve index	CPU time (ms)		Iteration time		Candidate points		Accuracy	
	Our method	Piegl's method	Our method	Piegl's method	Our method	Piegl's method	Our method	Piegl's method
Fig. 15	3.003	3.252	59	100	5	1	3.62e–09	5.13e–09
Fig. 16	4.558	5.701	77	200	5	1	1.51e–010	40.3e–010
Fig. 17	2.359	2.892	55	80	3	1	1.01e–08	1.07e–08
Fig. 18 (Fig. 2)	12.779	22.781	239	1000	19	3	1.31e–010	44.3e–010

Table 2
The results of point projection for NURBS surface

Surface index	CPU time (ms)		Iteration time		Candidate points		Accuracy	
	Our method	Piegl's method	Our method	Piegl's method	Our method	Piegl's method	Our method	Piegl's method
Fig. 19	34.03	89.25	327	631	1	1	3.62e–08	71.3e–08
Fig. 20	37.66	93.32	351	738	2	1	7.31e–09	57.3e–09
Fig. 23	32.65	65.37	311	621	1	1	2.34e–08	34.5e–08

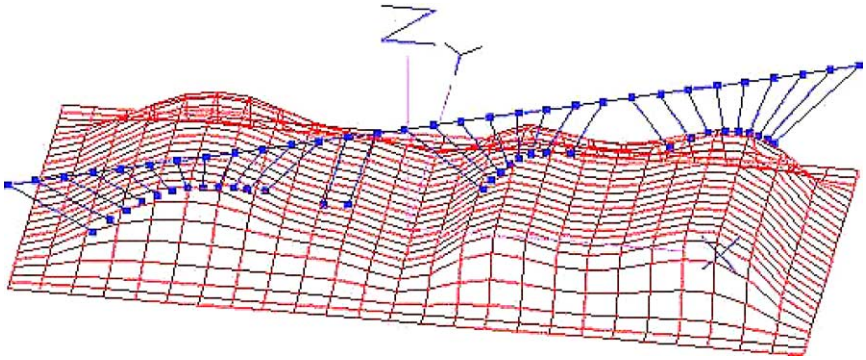


Fig. 19. Straight line projection.

11. Point inversion

Point inversion problem is very similar to point projection. The only difference is that the candidate Bézier subcurves or patches are extracted based upon the strong convex hull property (Piegl, 1991). Then, for curve, we apply Algorithm 5 to these candidate Bézier subcurves to extract the closest point on the NURBS curve and its corresponding curve parameter values. For surface, we apply Algorithm 6 to these candidate Bézier patches to extract the closest point on the NURBS surface and its corresponding surface parameter values.

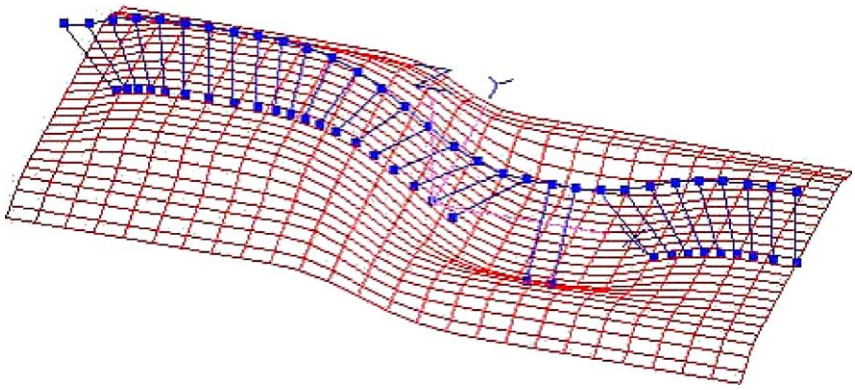


Fig. 20. Isocurve projection (U direction).

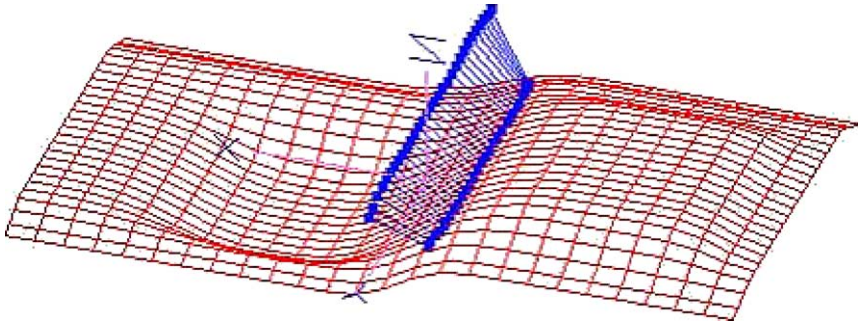


Fig. 21. Isocurve projection (V direction).

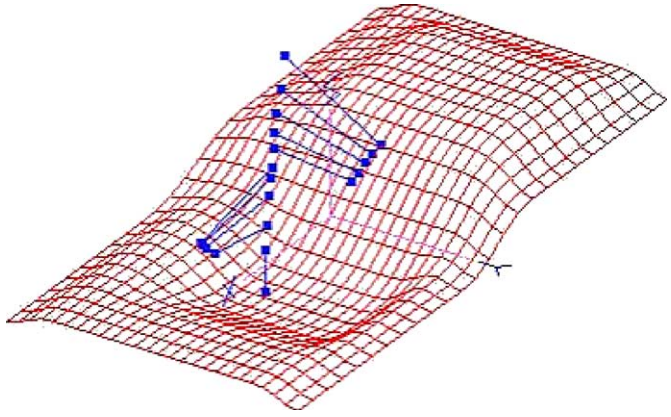


Fig. 22. Vertical line projection.

12. Conclusion

In this paper we have presented a novel method to solve both point projection and point inversion problems. This method achieves better results both in accuracy and efficiency, especially in dealing with

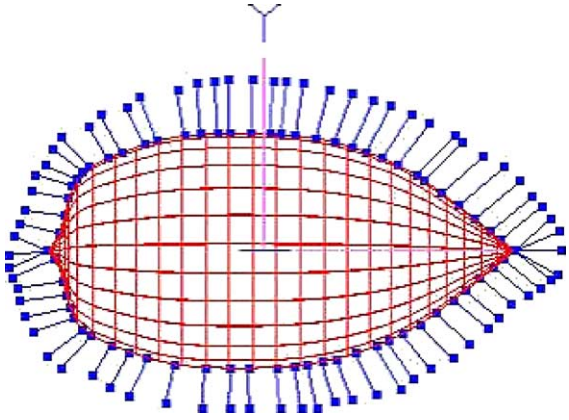


Fig. 23. Point projection for closed NURBS surface.

complex NURBS curves. Furthermore, this method extracts all candidate points for point projection so that it avoids selecting the wrong initial value for the Newton–Raphson method in the self-intersecting curve case. It also provides a good initial value to achieve reliable convergence for the Newton–Raphson method.

On the other hand, for a NURBS surface, this method dramatically decrease the computation of the algorithm compared with the method (Piegl and Tiller, 2001), which decomposes the NURBS into a set of quadrilaterals. We also apply the Newton–Raphson method on the Bézier patch instead of the whole NURBS surface, which improve the stability of the algorithm.

Acknowledgements

The authors wish to thank Tiller W. for a grant towards the cost of the NLib library. Thanks are also due to all the members of MVC for their help at various stage of this work, particularly to Robert Haines for proof reading this paper. We are also grateful to the Department of Computer Science for their financial support.

References

Boehm, W., 1980. Inserting new knots into B-spline curves. Computer-Aided Design 12, 199–201.

Cohen, E., Lyche, T., Riesebfeld, R., 1980. Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics. Comput. Graph. Image Process. 14.

Johnson, D.E., Cohen, E., 1998. A framework for efficient minimum distance computation. In: Proc. IEEE Intl. Conf. Robotics & Automation, Leuven, Belgium, pp. 3678–3684.

Chin, F., Wang, C.A., 1983. Optimal algorithms for the intersection and the minimum distance problems between planar polygons. IEEE Trans. Comput. C-32 (12), 1203–1207.

Edelsbrunner, H., 1982. On computing the extreme distances between two convex polygons. TU Graz, Tech. Rep. F96.

Limaiem, A., Trochu, F., 1995. Geometric algorithms for the intersection of curves and surfaces. Comput. Graph. 19 (3), 391–403.

Lin, M., Manocha, D., 1995. Fast interference detection between geometric models. The Visual Computer, 541–561.

Lyche, T., Morken, K., 1986. Making the Oslo algorithm more efficient. SIAM J. Numer. Anal. 23, 663–675.
Mortenson, M.E., 1985. In: Geometric Modeling. Wiley, New York, pp. 305–317.
Prautzsch, H., 1984. A short proof of the Oslo algorithm. Computer Aided Geometric Design 1.
Piegl, L., 1991. On NURBS: A survey. IEEE Comput. Graph. Appl. 11 (1), 55–71.
Piegl, L., Tiller, W., 1995. In: The NURBS Book. Springer-Verlag, pp. 229–234.
Piegl, L., Tiller, W., 2001. Parametrization for surface fitting in reverse engineering. Computer-Aided Design 33, 593–603.