Point to Ellipse Distance: A Binary Search Approach

Zhikai Wang

Computer Science Dept. Concordia University,
No. 11, 2155 St. Marc, Montreal ,Canada, H3H2G8 Tel:(1)-514-965 9686

Abstract

We want to find the shortest distance from a point to an ellipse without solving the roots of a polynomial of order four. We use a generating set of vectors to span an ellipse without calling high-cost trigonometric functions. Then we use a binary search method to find a specific normal vector on the ellipse. If the normal vector starting from a point on the ellipse passes the outside point specified, the shortest distance is found. We give a MATLAB implementation, a short discussion and expected future researches.

Key words: Point ellipse distance, generating set, normal vector, binary search, implementation

1. Introduction

An ellipse is a convex shape. An ellipse in *orthogonal* position follows

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1, \tag{1}$$

where $x,\ y,\ a,\ b \in \mathrm{R}$ and we take $a > 0, b > 0$. If the ellipse rotates about $(0,0)$ with an angle θ(counterclockwise) and then translates itself to (x_0, y_0). The new position (x_n, y_n) can be computed by

$$\begin{cases} x_n = xcos\theta - ysin\theta + x_0, \\ y_n = xsin\theta + ycos\theta + y_0. \end{cases} \tag{2}$$

For validity of such transformations see reference [2]. We can derive the equation of an ellipse of a *generic* form as

$$c_1x^2 + c_2y^2 + c_3xy + c_4x + c_5y + c_6 = 0 \tag{3}$$

Email address: zhi_wan@encs.concordia.ca (Zhikai Wang)

from Eq.(1) and Eq.(2). While such derivation is prone to errors.

We stop at

$$\begin{aligned} & \frac{(x-x_0)^2 cos^2\theta \ + \ 2(x-x_0)(y-y_0)cos\theta sin\theta \ + \ (y-y_0)^2 sin^2\theta}{a^2} \\ + & \ \frac{(y-y_0)^2 cos^2\theta \ - \ 2(y-y_0)(x-x_0)cos\theta sin\theta \ + \ (x-x_0)^2 sin^2\theta}{b^2} \\ = & \ 1, \end{aligned} \tag{4}$$

because the equation becomes extremely long and we cannot see an easy way back from Eq.(3) to Eq.(1) and Eq.(2) by Eq.(4). While, Eq.(1) and Eq.(2) are sufficient for studying a generic ellipse.

Given a point $\mathbf{Q}$, (x_q, y_q) outside the ellipse, it is desired to find the

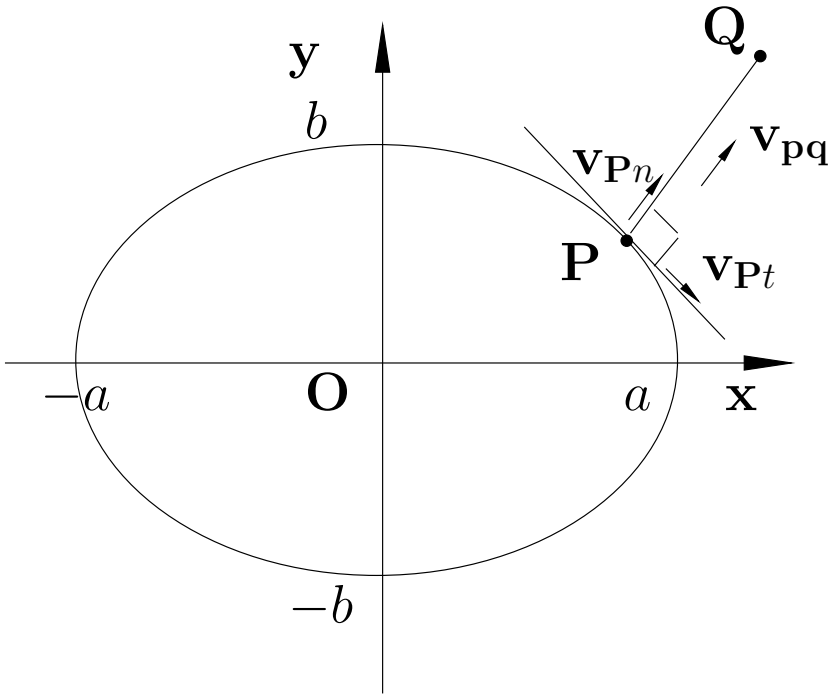

Figure 1: Shortest distance to an ellipse.

shortest distance to the ellipse. That is, we try to find a point $\mathbf{P}$ on the ellipse such that vector $\mathbf{v_{pq}}$ is orthogonal to the tangent vector $\mathbf{v}_{\mathbf{P}t}$ at $\mathbf{P}$, (x, y), or $\mathbf{v_{pq}}$ is parallel to the normal vector $\mathbf{v}_{\mathbf{P}n}$ at $\mathbf{P}$.

Now, we consider an orthogonal ellipse defined as Eq.(1) and ignore the degenerate cases. We can get

$$y' \;=\; -\frac{x}{y}\frac{b^2}{a^2}. \tag{5}$$

The tangent of $\mathbf{v_{pq}}$ is

$$\frac{y_q \;-\; y}{x_q - x}. \tag{6}$$

The tangent vector of $\mathbf{P}$ is orthogonal to the tangent of $\mathbf{v_{pq}}$, thus their product is -1. We get Eq.(7) with two unknown x and y

$$\begin{cases} -\dfrac{x}{y}\dfrac{b^2}{a^2}\cdot\dfrac{y_q \;-\; y}{x_q - x} \;=\; -1, \\ \qquad\dfrac{x^2}{a^2} \;+\; \dfrac{y^2}{b^2} \;=\; 1. \end{cases} \tag{7}$$

Further, we get the following equation with only one unkown y

$$\begin{aligned} (\frac{a^2}{b^2}-1)^2\cdot y^4 \;+\; 2(\frac{a^2}{b^2}-1)y_q\cdot y^3 \;+\; [y_q^2+\frac{a^2}{b^2}x_q^2 - b^2(\frac{a^2}{b^2}-1)^2]\cdot y^2 \\ -\; 2(\frac{a^2}{b^2}-1)b^2y_q\cdot y \;-\; b^2y_q^2 \;=\; 0. \end{aligned} \tag{8}$$

But we have to find the zeros of a polynomial of order four. Without a special mathematical package or training in numerical programming, we can't find the solution of Eq.(8) in an easy way, given the discussion of degenerate cases left behind.

Can we find a different approach of the above non-linear problem? While, the task of computational scientists is to find linear approach step by step for non-linear problems.

2. Spanning an Ellipse in 2D Space

Let $\mathbf{V}$ be a vector space over R^2, it has a *standard basis* set of $\mathbf{e}_1 \equiv (1,0)$, $\mathbf{e}_2 \equiv (0,1)$ [5]. The whole R^2 space can be spanned by a generating set that are the linear combinations of $\mathbf{e}_1$ and $\mathbf{e}_2$.

For the convenience in our computation, we use a generating set of $\mathbf{V}$ by levels. Given total levels l, we have $n+1$ generating vectors, $n = 2^{l+1}$, starting from $\mathbf{g}_0$ to $\mathbf{g}_n$. We let $\mathbf{g}_n \equiv \mathbf{g}_0$. At the first level we have five vectors

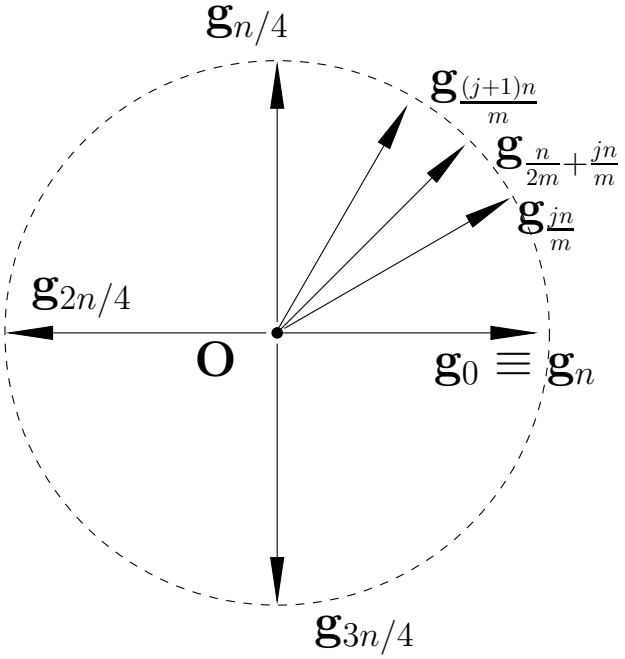

Figure 2: Generating set over R^2.

$\mathbf{g}_0 \equiv (1,0)$, $\mathbf{g}_{n/4n} \equiv (0,1)$, $\mathbf{g}_{2n/4} \equiv (-1,0)$, $\mathbf{g}_{3n/4} \equiv (0,-1)$ and $\mathbf{g}_{4n/4} \equiv \mathbf{g}_0$. From the second level on, at a given level i, we add the same number $m = 2^i$ of new generating vectors as one level upper. Each new vector is added between the two adjacent upper level vectors. The new vector is the vector sum of them then normalized. That is

$$\begin{aligned} \mathbf{g}_{\frac{n}{2m}+\frac{jn}{m}} &\equiv \mathbf{g}_{\frac{jn}{m}} \ + \ \mathbf{g}_{\frac{(j+1)n}{m}}, \\ \mathbf{g}_{\frac{n}{2m}+\frac{jn}{m}} &\equiv \mathbf{g}_{\frac{n}{2m}+\frac{jn}{m}} / |\mathbf{g}_{\frac{n}{2m}+\frac{jn}{m}}|, \end{aligned} \tag{9}$$

where $j = 0, 2, ..., m-1$. We can see the vectors are generated in a *fractal* way. Starting from $(0,0)$, we can span a 2D shape bounded by Eq.(1). For a generating vector $\mathbf{g}$, there will be a unique point on the ellipse which is the intersection of a ray from zero with tangent vector $\mathbf{g}$ with the ellipse. This intersection point is rather easy to compute. Suppose the ray along direction of $\mathbf{g}$, $\mathbf{g} \equiv (g_x, g_y)$ and $\mathbf{g}$ is normalized. We denote the intersection point $\mathbf{P} \equiv (p_x, p_y)$, t the distance from $\mathbf{P}$ to $(0,0)$. We have

$$t \ = \ \sqrt{1/(g_x^2/a^2 + g_y^2/b^2)} \tag{10}$$

and

$$\mathbf{P} \equiv (tg_x, tg_y). \tag{11}$$

For each $\mathbf{g}$ in the generating set, we can get one point. With all the points

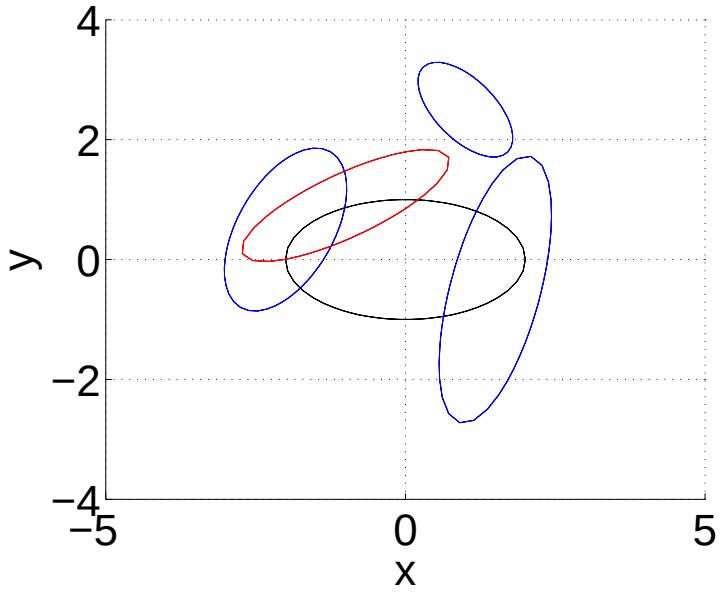

Figure 3: Some ellipses spanned by the generating set.

linked together one by one, the closed loop of the segments approximate an ellipse. In fact, this method can be extended to span any kind of convex shapes with a well-defined border equation. The key job is to make the generating set then compute the distance scale t of each vector. As $\mathbf{g}$ is directed, pointing outside from the center or zero, t is always greater than zero.

For better understanding, we attach an implementation of MATLAB script in Appendix(A). Please note in the script, MATLAB starting index is always 1. While, in this text, we utilize starting index 0. In the script almost all the computations are basic arithmetic operations of addition or multiplication, and linear transformations. While the general way to span ellipse has to call trigonometric functions frequently, whose cost is high. In our implementation, we also enable the rotation and translation transformations. It is obvious this implementation is not difficult to be re-written in C/C++, Java or other programming languages. In Figure(3), we show some

examples.

The MATLAB commands:

```
span_ellipse(2,1,0,[0 0],5,'k');
span_ellipse(1.0,0.5,−45,[1 2.5],5,'b');
span_ellipse(1.5,0.8,60,[−2 0.5],5,'b');
span_ellipse(0.75,2.3,−15,[1.5 −0.5],5,'b');
span_ellipse(1.9,0.53,25,[−1 0.9],5,'r');
```

The arguments are the ellipse's x-radius a, y-radius b (orthogonal form), the rotation angle θ in degrees, the new center coordinate (generic form), the fractal level, and the edge color.

3. Tangent Vector and Normal Vector

We take an implicit differentiation to Eq.(1), and we get

$$x' \frac{x}{a^2} \; + \; y' \frac{y}{b^2} \; = \; 0. \tag{12}$$

Proposition 3.1. *Vector* $\mathbf{v}_{\mathbf{P}t} \equiv (x', y')$ *in Eq.(12) is the tangent vector of a point* $\mathbf{P} \equiv (x, y)$ *on an orthogonal ellipse defined by Eq.(1). The vector* $\mathbf{v}_{\mathbf{P}n} \equiv (x/a^2, y/b^2)$ *is the normal vector of point* $\mathbf{P}$*. And* $\mathbf{v}_{\mathbf{P}n}$ *points outside.*

Suppose x' is a differentiation upon itself, on the ellipse where x' is defined $\mathbf{v}_{\mathbf{P}t} \equiv (1, y')$. Obviously the tangent of vector $\mathbf{v}_{\mathbf{P}t}$ is the tangent of $\mathbf{P}$. At points $(a, 0)$ and $(-a, 0)$, $x' \to 0$, we defined $\mathbf{v}_{\mathbf{P}t} \equiv (0, -1)$ or $\mathbf{v}_{\mathbf{P}t} \equiv (0, 1)$ respectively. Or we can discuss such degenerate cases separately. We can see the dot product of $\mathbf{v}_{\mathbf{P}t}$ and $\mathbf{v}_{\mathbf{P}n}$ is zero. By the definition of dot product, the vector orthogonal to the tangent vector must be the normal vector at $\mathbf{P}$, that is, its tangent computes the normal of this point. $\mathbf{v}_{\mathbf{P}n}$ points outside because its orientation is from the center pointing to the quadrant where (x, y) is located, see Figure(1).

Proposition 3.2. *In a given quadrant, a point* $\mathbf{P}$ *cut the segment of an orthogonal ellipse in this quadrant into two pieces. On the left piece of* $\mathbf{v}_{\mathbf{P}n}$*, all normal vectors point to the left direction of* $\mathbf{v}_{\mathbf{P}n}$*; on the right piece of* $\mathbf{v}_{\mathbf{P}n}$*, all normal vectors fall to the right direction of* $\mathbf{v}_{\mathbf{P}n}$*.*

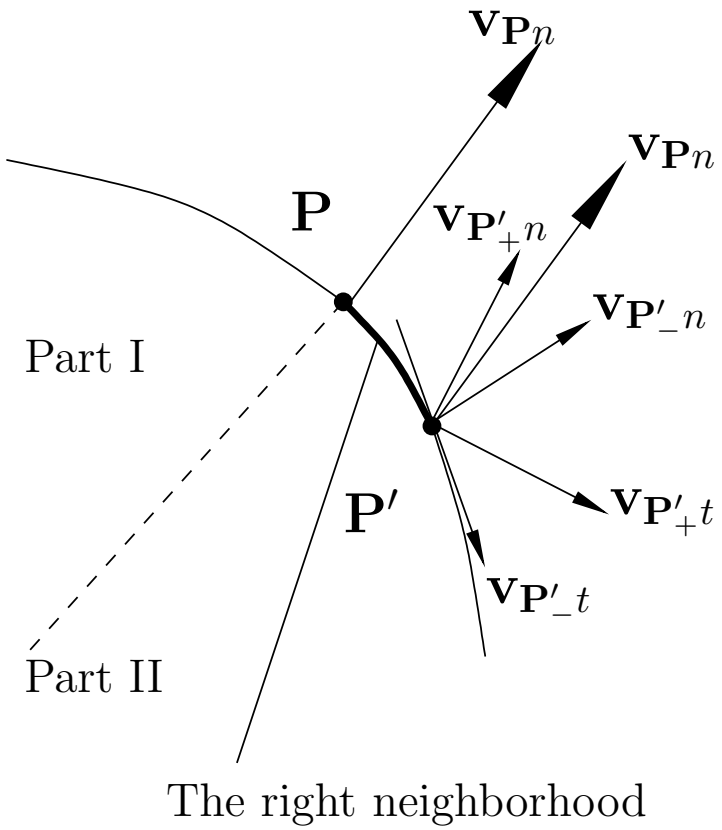

Figure 4: Within a quadrant, $\mathbf{P}$ divides the ellipse segment to two parts.

Within a quadrant(the axis excluded), the angles between the normal vectors are all less than 90 degrees. So any normal vector other than $\mathbf{P}$'s will not turn one round from one side of $\mathbf{P}$ to the other. Here the ellipse segment is sufficiently smooth. So at $\mathbf{P}$'s right side, there is a neighborhood that all normal vectors fall to the right of $\mathbf{v}_{\mathbf{P}n}$, otherwise the convex property will be destroyed. Suppose $\mathbf{P}'$ is the end of this neighborhood, see Figure(4). $\mathbf{v}_{\mathbf{P}'_+n}$ points to the left of $\mathbf{v}_{\mathbf{P}n}$ and $\mathbf{v}_{\mathbf{P}'_-n}$ points to the left of $\mathbf{v}_{\mathbf{P}n}$. Thus $\mathbf{v}_{\mathbf{P}'_+t}$ points to the left of $\mathbf{v}_{\mathbf{P}'_t}$. $\mathbf{P}'$ becomes a turning point. Such should not happen on an ellipse. The same principle applies to the other part.

Proposition 3.3. *Suppose a point* $\mathbf{Q}$ *outside the ellipse is in the same quadrant as* $\mathbf{P}$. $\mathbf{Q}$ *decides a unique* $\mathbf{P}$ *whose normal vector passes* $\mathbf{Q}$.

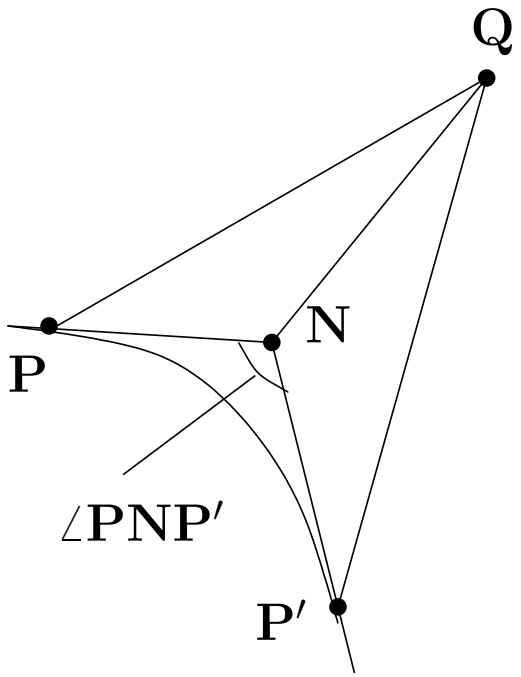

Figure 5: The normal vector passing **Q** is unique.

Suppose in the ellipse segment, there are two points **P** and **P′** and both their normal vectors pass through **Q**. Since the ellipse is convex, so the tangent lines of the two points, intersect at a point **N**. Angle $\angle\mathbf{PNP'}$, shown in Figure(5), is less than 180 degrees.

We can show that

$$\begin{array}{rl} & \angle\mathbf{PNP'} \\ = & \angle\mathbf{PQP'} + \angle\mathbf{QPN} + \angle\mathbf{QP'N} \\ > & \angle\mathbf{QPN} + \angle\mathbf{QP'N'} \\ = & 180^\circ \end{array} \tag{13}$$

It contradicts $\angle\mathbf{PNP'}$ is less than 180 degrees. So **P** is unique.

Proposition 3.4. *If we have a directed line through two points* **p** *and* **q**. *Let* $\mathbf{p} \equiv (p_x, p_y)$, $\mathbf{q} \equiv (q_x, q_y)$, *and* $\mathbf{r} \equiv (r_x, r_y)$. *We define a matrix*

$$\mathbf{D} \equiv \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}. \tag{14}$$

If $det(\mathbf{D}) > 0$ *then* $\mathbf{r}$ *lies left of the line. If* $det(\mathbf{D}) < 0$ *then* $\mathbf{r}$ *lies right of the line. If* $det(\mathbf{D}) = 0$ *then* $\mathbf{r}$ *lies on the line.*

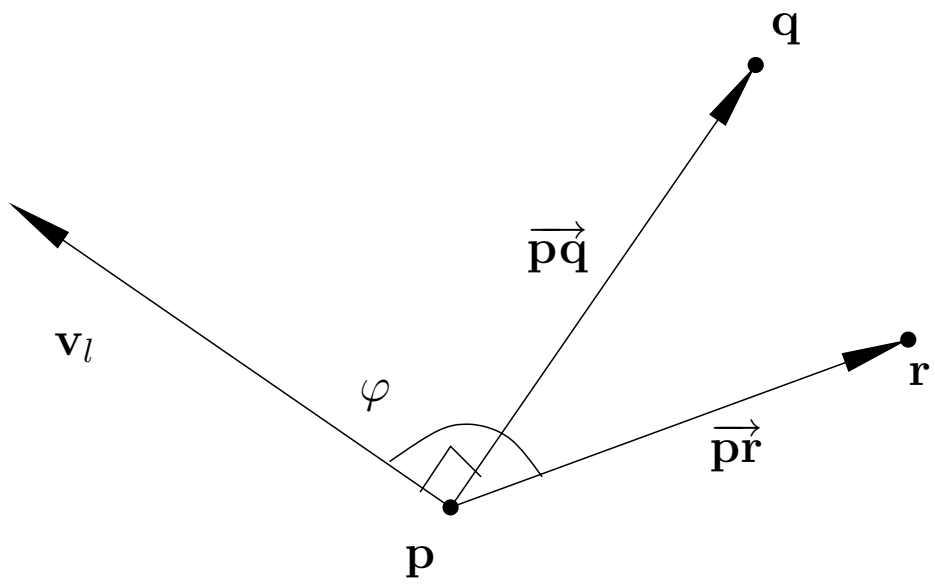

Figure 6: $< \mathbf{v}_l, \overrightarrow{\mathbf{pr}} >$ decides the sign of $cos\varphi$.

This proposition originates from a course assignment of a graduate course [6]. The following proof is given by the author.

Proof We have

$$\begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix} = \begin{vmatrix} q_x & q_y \\ r_x & r_y \end{vmatrix} - \begin{vmatrix} p_x & p_y \\ r_x & r_y \end{vmatrix} + \begin{vmatrix} p_x & p_y \\ q_x & q_y \end{vmatrix}$$

$$= q_x r_y - q_y r_x - p_x r_y + p_y r_x + p_x q_y - p_y q_x. \tag{15}$$

We also can get

$$\overrightarrow{\mathbf{pq}} \equiv (q_x - p_x, q_y - p_y), \tag{16}$$

and,

$$\overrightarrow{\mathbf{pr}} \equiv (r_x - p_x, r_y - p_y). \tag{17}$$

See Figure(6). We rotate $\overrightarrow{\mathbf{pq}}$ about $(0, 0)$ with 90° and get a left vector $\mathbf{v}_l$ of $\overrightarrow{\mathbf{pq}}$. The x component of $\mathbf{v}_l$ is

$$(q_x - p_x)cos90^\circ - (q_y - p_y)sin90^\circ. \tag{18}$$

The y component of $\mathbf{v}_l$ is

$$(q_x - p_x)sin90^\circ \ - \ (q_y - p_y)cos90^\circ. \tag{19}$$

See Eq.(2). Thus

$$\mathbf{v}_l \ \equiv \ (p_y - q_y, q_x - p_x). \tag{20}$$

The dot product

$$\begin{array}{rl} & < \mathbf{v}_l, \overrightarrow{\mathbf{pr}} > \\ = & (p_y - q_y)(r_x - p_x) \ + \ (q_x - p_x)(r_y - p_y) \\ = & p_y r_x \ - \ q_y r_x \ - \ p_x p_y \ + \ q_y p_x \ + \ q_x r_y \ - \ q_x p_y \ - \ p_x r_y \ + \ p_x p_y \\ = & q_x r_y \ - \ q_y r_x \ - \ p_x r_y \ + \ p_y r_x \ + \ p_x q_y \ - \ p_y q_x \end{array} \tag{21}$$

We have (15) $\equiv$ (21). By the definition of dot product, Eq.(21) decides the sign of $cos\varphi$ shown in Figure(6). If $< \mathbf{v}_l, \overrightarrow{\mathbf{pr}} >$ is less than 0, $\mathbf{r}$ must lie right of direct line $\mathbf{pq}$. If $< \mathbf{v}_l, \overrightarrow{\mathbf{pr}} >$ is equal to 0, $\mathbf{r}$ lies on the line $\mathbf{pq}$. If $< \mathbf{v}_l, \overrightarrow{\mathbf{pr}} >$ is greater than 0, $\mathbf{r}$ must lie left of direct line $\mathbf{pq}$. □

4. Binary Search of a Normal Vector

Now we discuss a *binary search* approach to locate a specific normal vector on the ellipse.

Given a point $\mathbf{Q}$ outside a generic ellipse, which is rotated a θ angle and translated to (x_0, y_0), we transform $\mathbf{Q}$ to the orthogonal form coordinate system of the ellipse by the following equation

$$\begin{array}{rcl} q_{nx} & = & (q_x - x_0)cos(-\theta) \ - \ (q_y - y_0)sin(-\theta), \\ q_{ny} & = & (q_y - y_0)cos(-\theta) \ + \ (q_x - x_0)sin(-\theta). \end{array} \tag{22}$$

That means we first translate $\mathbf{Q}$ into a coordinate system center at (x_0, y_0) as $(0, 0)$, then rotate the coordinate by a negative θ. The validity of such transformation is widely discussed. Further reference please see [2]. The following discussion will rely on an orthogonal ellipse model. The computational results will be transformed to the generic system by Eq.(2). There are four degenerate cases at points $(a, 0)$, $(-a, 0)$, $(0, b)$, and $(0, -b)$. How these degenerate cases are dealt please see Appendix(B). With these degenerate cases resolved, our discussion falls within a single quadrant. We take, in the R^2 vector space, two vectors $\mathbf{g}_1$ and $\mathbf{g}_2$ counterclockwise which bound a quadrant where the transformed $\mathbf{Q}$ is located. For examples, if $\mathbf{g}_1 \equiv (1, 0)$

then $\mathbf{g}_2 \equiv (0,1)$; if $\mathbf{g}_1 \equiv (0,-1)$ then $\mathbf{g}_2 \equiv (1,0)$.

At the first step, let

$$\begin{array}{rcl} \mathbf{g}_3 & \equiv & \mathbf{g}_1 + \mathbf{g}_2, \\ \mathbf{g}_3 & \equiv & \mathbf{g}_3/|\mathbf{g}_3|. \end{array} \tag{23}$$

Along the direction of g_3 we can find a point $\mathbf{P}'$ on the ellipse together with its normal vector $\mathbf{v}_{\mathbf{P}'n}$.

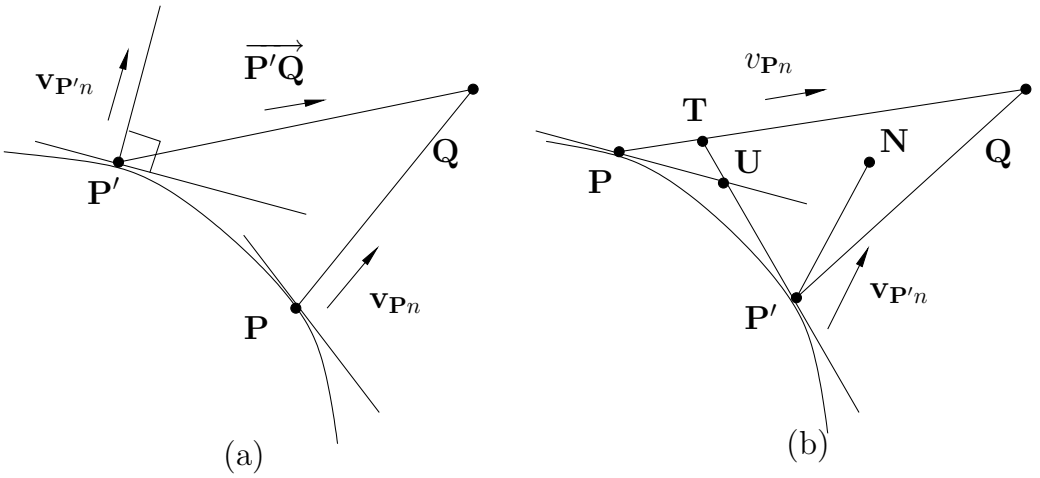

Figure 7: The angle relations of Proposition(4.1).

Proposition 4.1. *Suppose* $\mathbf{P}$ *is the nearest point to* $\mathbf{Q}$ *within a quadrant. If a point* $\mathbf{P}'$*'s normal vector* $\mathbf{v}_{\mathbf{P}'n}$ *points to the right of directed line* $\overrightarrow{\mathbf{P}'\mathbf{Q}}$ *then* $\mathbf{P}'$ *lies to the right of* $\mathbf{P}$*. If a point* $\mathbf{P}'$*'s normal vector points to the left of directed line* $\overrightarrow{\mathbf{P}'\mathbf{Q}}$ *then* $\mathbf{P}'$ *lies to the left of* $\mathbf{P}$*.*

In Figure(7.a), we show that $\mathbf{v}_{\mathbf{P}'n}$ points to the left of $\overrightarrow{\mathbf{P}'\mathbf{Q}}$ and $\mathbf{P}'$ is on the left of $\mathbf{P}$. In Figure(7.b), we assume that $\mathbf{v}_{\mathbf{P}'n}$ points to the left of $\overrightarrow{\mathbf{P}'\mathbf{Q}}$, but $\mathbf{P}'$ is on the right of $\mathbf{P}$. Let $\mathbf{P}'\mathbf{N}$ be the line which $\mathbf{v}_{\mathbf{P}'n}$ is on. Since the ellipse is convex, the two tangent line of $\mathbf{P}$ and $\mathbf{P}'$ will intersect at a point $\mathbf{U}$. And line $\mathbf{U}\mathbf{P}'$ intersects line $\mathbf{P}\mathbf{Q}$ at point T. We have the angle relation

$$\angle\mathbf{QPU} \;=\; \angle\mathbf{QTU} \;-\; \angle\mathbf{TUP}. \tag{24}$$

In $\triangle\mathbf{QP}'\mathbf{T}$, $\angle\mathbf{QTU}$ is less than $90°$, because $\angle\mathbf{QTP}' = \angle\mathbf{QTU}$ and $\angle\mathbf{QP}'\mathbf{T} > \angle\mathbf{NP}'\mathbf{T} = 90°$. It contradicts $\angle\mathbf{QPU} = 90°$. The same principle applies if

$\mathbf{v}_{\mathbf{P}'n}$ points to the left of directed line $\overrightarrow{\mathbf{P}'\mathbf{Q}}$.

Thus we know, given a $\mathbf{P}'$ calculated we will know the direction of $\mathbf{P}$. Next, we have two cases

Case I If $\mathbf{v}_{\mathbf{P}'n}$ points to the left of $\overrightarrow{\mathbf{P}'\mathbf{Q}}$, then let $\mathbf{g}_2 = \mathbf{g}_3$;

Case II If $\mathbf{v}_{\mathbf{P}'n}$ points to the right of $\overrightarrow{\mathbf{P}'\mathbf{Q}}$, then let $\mathbf{g}_1 = \mathbf{g}_3$.

Again we compute $\mathbf{g}_3$ by Eq.(23). A new $\mathbf{P}'$ is acquired by new $\mathbf{g}_3$. The

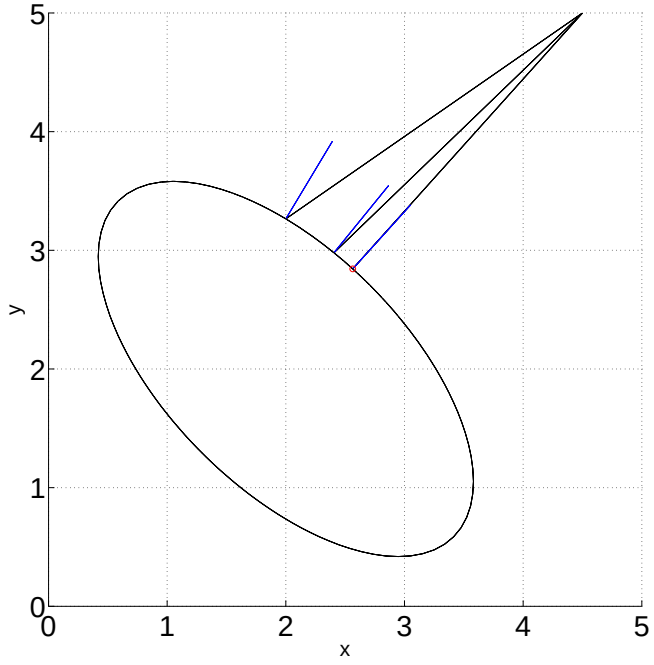

Figure 8: A computational result plotted by MATLAB.

geometrical meaning is that if $\mathbf{P}'$ is to the right of $\mathbf{P}$, we try to move $\mathbf{P}'$ to the left of $\mathbf{P}$; if $\mathbf{P}'$ is to the left of $\mathbf{P}$, we try to move $\mathbf{P}'$ to the right of $\mathbf{P}$. Each time, our searching space is halved. The search will end till $\mathbf{P}'$ and $\mathbf{P}$ overlap. Thus the complexity of this algorithm is $\mathbf{O}(log\ n)$, n is the maximum discretization number of the vector space over R^2.

We attach a MATLAB script implementation in Appendix(B). Figure(8) shows a computational result. We call the script by MATLAB command line.

$\|det\ \mathbf{D}\|_{max}$	$n = 1/\|det\ \mathbf{D}\|_{max}$	$log(n)$	iteration times
1e-1	1e+1	1	4
1e-2	1e+2	2	6
1e-3	1e+3	3	10
1e-4	1e+4	4	14

Table 1: $|det\ \mathbf{D}|_{max}$ *vs.* iteration times

```
r= generic_dist(2,1,−45,[2 2],[4.5 5], 1e−2)
r = 2.5637    2.8436
    2.8982    2.8982
    0.6711    0.7414
```

r returns the coordinate of point $\mathbf{P}$, the distance between $\mathbf{P}$ and $\mathbf{Q}$, and the normal vector at $\mathbf{P}$. The semantics of this command is that, given an orthogonal ellipse $\frac{x^2}{a^2} + \frac{b^2}{b^2} = 1$, where $a = 2,\ b = 1$, it is rotated by $-45°$ then translated to point $(2, 2)$. $\mathbf{Q}$ is point $(4.5, 5)$. The last argument is the tolerance of the matrix determinant in Proposition(3.4). At each step, we show line $\mathbf{P'Q}$ and the normal vector at $\mathbf{P'}$ (the short blue segment). The computed $\mathbf{P}$ is emphasized by a red mark. We also give another example in Figure(9) which takes more iteration steps. Table(1) list a the accuracy tolerance *vs.* the iteration times of the example in Figure(9). The readers can experiment more with this script. It can be re-distributed freely for non-profit academic and educational purposes.

```
r = generic_dist (3.75,1.25,25,[−2 3],[−4 7], 1e−3)
r =−2.6134    4.0928
    3.2210    3.2210
   −0.4302    0.9027
```

Due to the accuracy setting for convenience of programming, we didn't test smaller value of $|det\ \mathbf{D}|_{max}$. n got by $1/|det\ \mathbf{D}|_{max}$ may not be a good measurement. But we can see the iteration times generally follows our expectation of $\mathbf{O}(logn)$. More accurate searching complexity experiments can be left for further research.

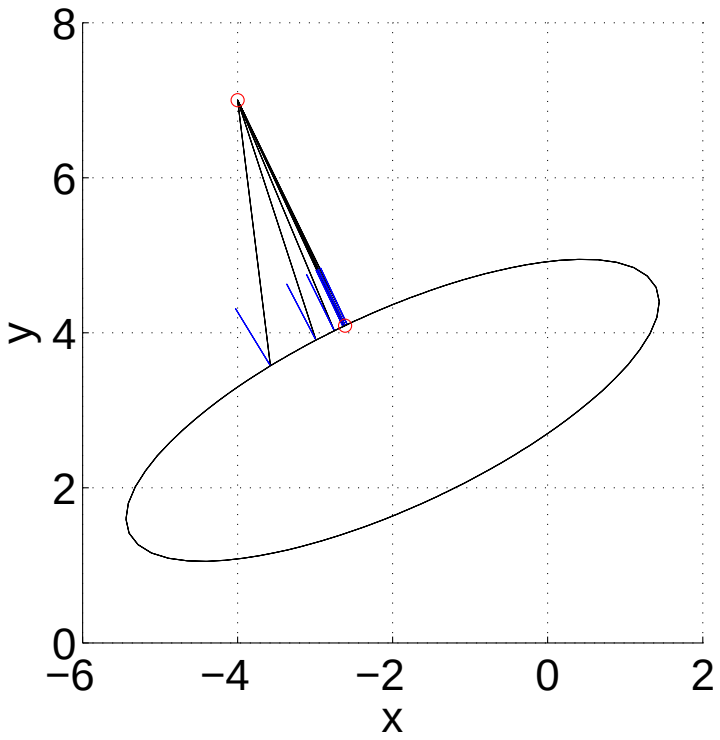

Figure 9: An example taking more iteration steps.

5. Discussion and Further Researches

The author's research interests are scientific computation, computer graphics and graphical user interface design. They don't immediately fall within the area of computational geometry. So the vision of this text may be limited. Some of the arguments may be incomplete. But the author tries to make them as strong as possible. Some propositions unless with a strong mathematical proof are only given the according explanations.

Upon the end of this text, a thought immediately flashes to the mind is that if we can also apply this method to find the shortest distance to an ellipsoid in 3D space. We may also use the generating set to span tubular surface which is usually utilized in computer graphics. The further study may also be extended to convex hull with a dictionary of discrete normal vectors or a normal vector interpolation function. In this text, point $\mathbf{P}$ is obviously a fixed point in a non-linear system. May we locate $\mathbf{P}$ at a faster

converging rate. We expect more works to be done.

A. A MATLAB Script Spanning an Ellipse

```
%% Spanning an ellipse
%% This script can be re-distributed freely,
%% but ONLY for non-profit academic or educational purposes,
%% given this file header kept.
%% Author: Zhikai Wang
%% Computer Science Dept,
%% Concordia Univ, QC, Canada
%% zhi_wan@encs.concordia.ca
function h=span_ellipse(a,b,rot_angle,p0,fractal_level,e_color)
% input: a,b x,y-radius of the ellipse (orthogonal)
% It is rotated about (0,0) with rot_angle,counterclockwise
% then translated to point p0.
% fractal_level: level of the generating set
% e_color: edge color of the ellipse
% no return values.
level = fractal_level;
n = power(2,level+1);
g(0/4*n+1,:) = [1,0];
g(1/4*n+1,:) = [0,1];
g(2/4*n+1,:) = [-1,0];
g(3/4*n+1,:) = [0,-1];
g(4/4*n+1,:) = [1,0];

% get all the vectors
for i=2:level
    m = power(2,i);
    step = n/m;
    start = step /2 + 1;
    for j = 1:m
        g(start+(j-1)*step,:)...
            = g(start+(j-1)*step - step/2,:)...
            + g(start+(j-1)*step + step/2,:);
        g(start+(j-1)*step,:) ...
            = g(start+(j-1)*step,:)...
            /norm(g(start+(j-1)*step,:));
    end
end

% decide the spanning scale of each vector;
```

```
% compute the orthogonal ellipse save
% in an array p of 2D points ;
s = size(g);
for i=1:s(1)
    gv = g(i,:);
    g1 =  gv(1);
    g2 =  gv(2);
    t = sqrt(1/( g1*g1/(a*a) + g2*g2/(b*b) ));
    p(i,:) = [t*g1 t*g2];
end

% rotate the ellipse
angle = rot_angle/180*pi;
if angle ~= 0
    ca = cos(angle);
    sa = sin(angle);
    s = size(g);
    for i=1:s(1)
        pv = p(i,:);
        p1 =  pv(1);
        p2 =  pv(2);
        p1p = p1*ca-p2*sa;
        p2p = p1*sa+p2*ca;
        p(i,:) = [p1p p2p];
    end
end

% translate the ellipse to new position p0
if norm(p0) ~= 0
    s = size(g);
    for i=1:s(1)
        pv = p(i,:);
        p1 =  pv(1);
        p2 =  pv(2);
        p1p = p1 + p0(1);
        p2p = p2 + p0(2);
        p(i,:) = [p1p p2p];
    end
end

% draw the ellipse (link the points as a loop)
s = size(p);
for i = 1: s(1)-1
    p1 = p(i,:);
    p2 = p(i+1,:);
```

```
    h =  patch( [p1(1) p2(1)],[p1(2) p2(2)],'k');
    set(h,'EdgeColor',e_color);
end
```

B. Find a Specific Normal Vector

```
%% Find shortest distance to an Ellipse
%% This script can be re-distributed freely,
%% but ONLY for non-profit academic or educational purposes,
%% given this file header kept.
%% Author: Zhikai Wang
%% Computer Science Dept,
%% Concordia Univ, QC, Canada
%% zhi_wan@encs.concordia.ca
function r=generic_dist(a,b,rot_angle,p0,Q0,tolerance)
% input: a,b x,y-radius of the ellipse (orthogonal).
% It is rotated about (0,0) with rot_angle,counterclockwise
% then translated to point p0.
% Q0: a point outside the ellipse.
% tolerance: the max absolute value of det D

% return values:
% r(1,:) return point P, Q0P is the shortest
% r(2,:) the distance |Q0P|
% r(3,:) the normal vector at P,pointing outside

angle = rot_angle/180*pi;
cosine_theta = cos(angle);
sine_theta = sin(angle);

% transform Q0 into the orthogonal coordinate system
Q = [0 0];
Q(1) = (Q0(1) -p0(1)) * cosine_theta...
    + (Q0(2) -p0(2)) * sine_theta;
Q(2) = (Q0(2) -p0(2)) * cosine_theta...
    - (Q0(1) -p0(1)) * sine_theta;

% use V to tell which quadrant Q is at
V = Q -[0 0];
if  norm(V) == 0
    s = 'ill condition, Q is at zero'
    r = -1;
```

```
37      return;
38  end
39
40  % If Q is inside the ellipse or on the ellipse
41  V = V/norm(V);
42  gv = V;
43  g1 =  gv(1);
44  g2 =  gv(2);
45  t = sqrt(1/( g1*g1/(a*a) + g2*g2/(b*b) ));
46  p = [t*g1 t*g2];
47  if  norm(p) ≥ norm(Q)
48      s = 'ill condition, Point Q inside or on the ellipse'
49      r = -1;
50      return;
51  end
52
53  % four degenerate cases; but two solutions
54  if V(1) == 0
55      r(1,:) = [0 b*V(2)];
56      r(1,:) = transform2d(r(1,:),cosine_theta,sine_theta,p0);
57      dist = abs(Q0(2))-b;
58      r(2,:) = [dist dist];
59      r(3,:) = [0 V(2)/abs(V(2))];
60      r(3,:) = transform2d(r(3,:),cosine_theta,sine_theta,p0);
61  end
62  if V(2) == 0
63      r(1,:) = [a*V(1) 0];
64      r(1,:) = transform2d(r(1,:),cosine_theta,sine_theta,p0);
65      dist = abs(Q0(1))-a;
66      r(2,:) = [dist  dist];
67      r(3,:) =[V(1)/abs(V(1)) 0];
68      r(3,:) = transform2d(r(3,:),cosine_theta,sine_theta,p0);
69  end
70
71  if V(2) == 0 || V(1) == 0
72
73      span_ellipse(a,b,rot_angle,p0,6,'k');
74      hold on
75      px =[Q0(1) r(1,1)];
76      py =[Q0(2) r(1,2)];
77      plot(px,py,'ro');
78
79      % draw PQ
80      h = patch( [r(1,1) Q0(1)],[r(1,2) Q0(2)],'k');
81      set(h,'EdgeColor','k');
```

```
    % draw the normal vector
    h = patch( [r(1,1) r(1,1)+dist/4.0*r(3,1)],...
        [r(1,2) r(1,2)+dist/4.0*r(3,2)],'k');
    set(h,'EdgeColor','b');

    daspect([1 1 1]);
    return;
end
% tell which quadrant Q is in;
% set the bounding generating vecotrs
if V(1) > 0
    if V(2) > 0
        g(1,:) = [1 0];
        g(2,:) = [0 1];
        g(3,:) = g(1,:) +g(2,:);
        g(3,:) = g(3,:)/norm(g(3,:));
    else
        g(1,:) = [0 -1];
        g(2,:) = [1 0];
        g(3,:) = g(1,:) +g(2,:);
        g(3,:) = g(3,:)/norm(g(3,:));
    end
else
    if V(2) > 0
        g(1,:) = [0 1];
        g(2,:) = [-1 0];
        g(3,:) = g(1,:) +g(2,:);
        g(3,:) = g(3,:)/norm(g(3,:));
    else
        g(1,:) = [-1 0];
        g(2,:) = [0 -1];
        g(3,:) = g(1,:) +g(2,:);
        g(3,:) = g(3,:)/norm(g(3,:));
    end

end

%gv = g(3,:);
%g1 =  gv(1);
%g2 =  gv(2);
t = sqrt(1/( g(3,1)*g(3,1)/(a*a) + g(3,2)*g(3,2)/(b*b) ));
p = [t*g(3,1) t*g(3,2)];
p_norm = [p(1)/(a*a)  p(2)/(b*b)];
p_norm = p_norm/norm(p_norm);

```

```
v_pq = Q - p;
dist = norm(v_pq);
if dist== 0
    r = 'Ill conditioned, point Q lies on the ellipse.'
    return;
end
v_pq = v_pq /dist;
% draw P'Q
pd = transform2d(p,cosine_theta,sine_theta,p0);
h = patch( [pd(1) Q0(1)],[pd(2) Q0(2)],'k');
set(h,'EdgeColor','k');
% draw normal vector of P'
pd_norm = transform2d(p_norm,cosine_theta,sine_theta,[0 0]);
h = patch( [pd(1) pd(1)+dist/4.0*pd_norm(1)],...
    [pd(2) pd(2)+dist/4.0*pd_norm(2)],'k');
set(h,'EdgeColor','b');

% use det A to tell r is left/right or on directed line pQ
pr = p + dist/4.0*p_norm;
it = 1
A = [1 p(1) p(2); 1 Q(1) Q(2); 1 pr(1) pr(2)];

dta = det(A) ;

while abs(dta) > tolerance
    %if it > 20
    %    break;
    %end

    if dta >0
        % left, then we turn to right
        g(2,:) = g(3,:);
        g(3,:) = g(1,:) +g(2,:);
        g(3,:) = g(3,:)/norm(g(3,:));
    else
        g(1,:) = g(3,:);
        g(3,:) = g(1,:) +g(2,:);
        g(3,:) = g(3,:)/norm(g(3,:));
    end

    t = sqrt(1/( g(3,1)*g(3,1)/(a*a) + g(3,2)*g(3,2)/(b*b) ));
    p = [t*g(3,1) t*g(3,2)];
    p_norm = [p(1)/(a*a)  p(2)/(b*b)];
    p_norm = p_norm/norm(p_norm);

```

```
    v_pq = Q - p;
    dist = norm(v_pq);
    if dist== 0
        r = 'Ill conditioned, point Q lies on the ellipse.'
        return;
    end
    v_pq = v_pq /dist;

    pd = transform2d(p,cosine_theta,sine_theta,p0);
    h = patch( [pd(1) Q0(1)],[pd(2) Q0(2)],'k');
    set(h,'EdgeColor','k');
    % draw normal vector of P'
    pd_norm = transform2d(p_norm,cosine_theta,sine_theta,[0 0]);
    h = patch( [pd(1) pd(1)+dist/4.0*pd_norm(1)],...
        [pd(2) pd(2)+dist/4.0*pd_norm(2)],'k');
    set(h,'EdgeColor','b');

    pr = p + dist/4.0*p_norm;
    A = [1 p(1) p(2); 1 Q(1) Q(2); 1 pr(1) pr(2)];
    dta = det(A) ;
    it = it + 1
end

r(1,:) = pd;
r(2,:) = sqrt( (pd(1)-Q0(1))*(pd(1)-Q0(1))...
    + (pd(2)-Q0(2))*(pd(2)-Q0(2)) );
r(3,:) = pd_norm;

span_ellipse(a,b,rot_angle,p0,6,'k');
hold on

px =[Q0(1) r(1,1)];
py =[Q0(2) r(1,2)];
plot(px,py,'ro');
hold on

daspect([1 1 1]);
```

C. An Auxiliary Function of Appendix(B)

```
%% transform2d.m: rotation theta then translate to p0
%% This script can be re-distributed freely,
```

```
%% but ONLY for non-profit academic or educational purposes,
%% given this file header kept.
%% Author: Zhikai Wang
%% Computer Science Dept,
%% Concordia Univ, QC, Canada
%% zhi_wan@encs.concordia.ca
function r=transform2d(inPoint,cosine_theta,sine_theta,p0)

x = inPoint(1);
y = inPoint(2);

outPoint_x = x*cosine_theta - y*sine_theta + p0(1);
outPoint_y = y*cosine_theta + x*sine_theta + p0(2);

r = [outPoint_x outPoint_y];
```

References

[1] Ioannis Z. Emiris, George M. Tzoumas, Algebraic study of the Apollonius circle of three ellipses, http://cgi.di.uoa.gr/∼geotz/papers/EmiTzo-EWCG05p-ApollEll.pdf, Last access Feb 28, 2009.

[2] Donald Hearn, M. Pauline Baker, Computer Graphics with OpenGL 3rd edition, Prentice Hall, 2003.

[3] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Ellipse, Last access Feb 28, 2009.

[4] MATLAB, the Language of Technical Computing, http://www.mathworks.com/products/matlab/, Last access Feb 28, 2009.

[5] Stephen H. Friedberg, Arnold J. Insel, Lawrence E. Spence, Linear Algebra Fourth Edition, Prentice, New Delhi, 2007.

[6] Thomas Fevens, Graduate course COMP 6711, Computational Geometry, Concordia University, Canada, 2007.

[7] Mark de Berg, marc van Kreveld, Mark Overmars, Otfried Schwarzkopf, Computational geometry, Second Revised Edition, Springer, 2000.