Scan Converting an Ellipses

The ellipse is also a symmetric figure like a circle, but it has four- way symmetry rather than eight-way. There are two methods:

(1) Polynomial Method of defining an Ellipses

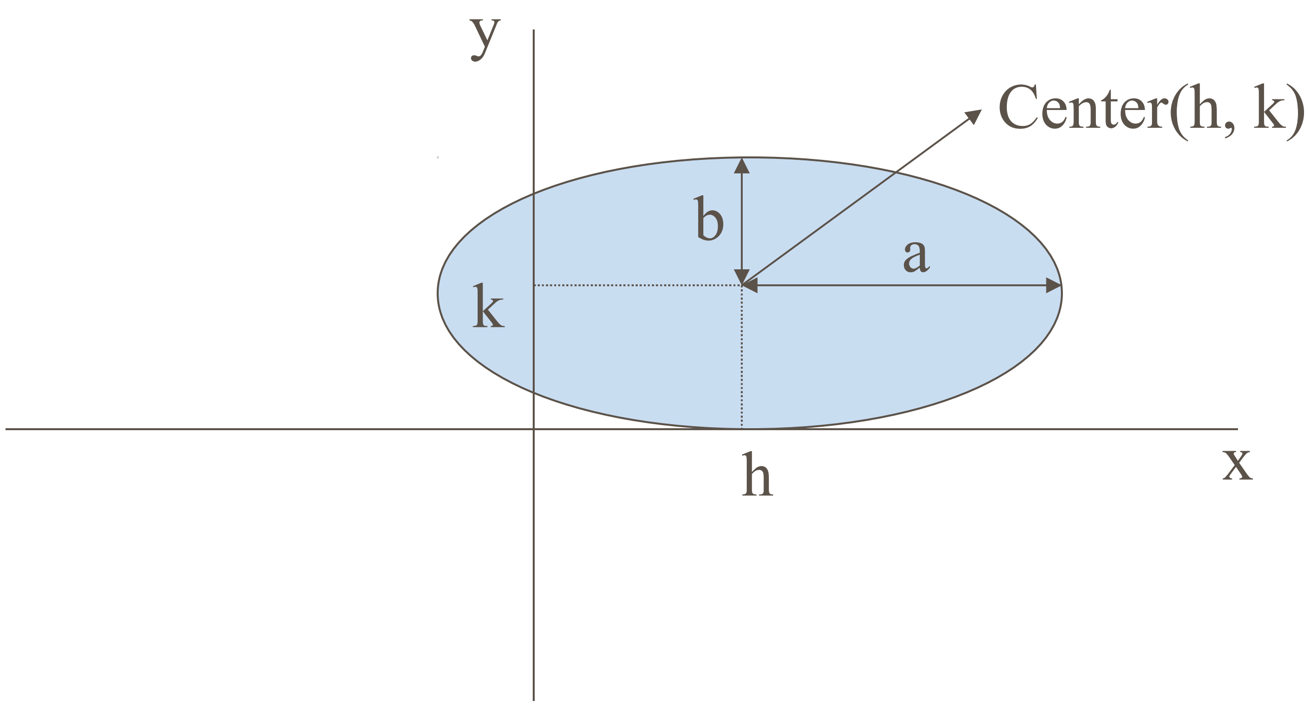
It is given by the equation:

$$\frac{(x - h)^2}{a^2} + \frac{(y - k)^2}{b^2} = 1$$

where (h, k) = ellipse center

a = length of major axis

b = length of minor axis



When the polynomial method is used to define an ellipse, the value of x is incremented from h to a. For each step of x, each value of y is found by evaluating the expression.

$$y = b\sqrt{1 - \frac{(x-h)^2}{a^2}} + k$$

This method is very inefficient, because the square of a and (x – h) must be found; then floating-point division of $(x-h)^2$ by a^2 etcs….

(2) Trigonometric method of defining an Ellipses:

The following equations define an ellipse trignometrically :

$$x = a\cos(\theta) + h \qquad \text{and} \qquad y = b\sin(\theta) + k$$

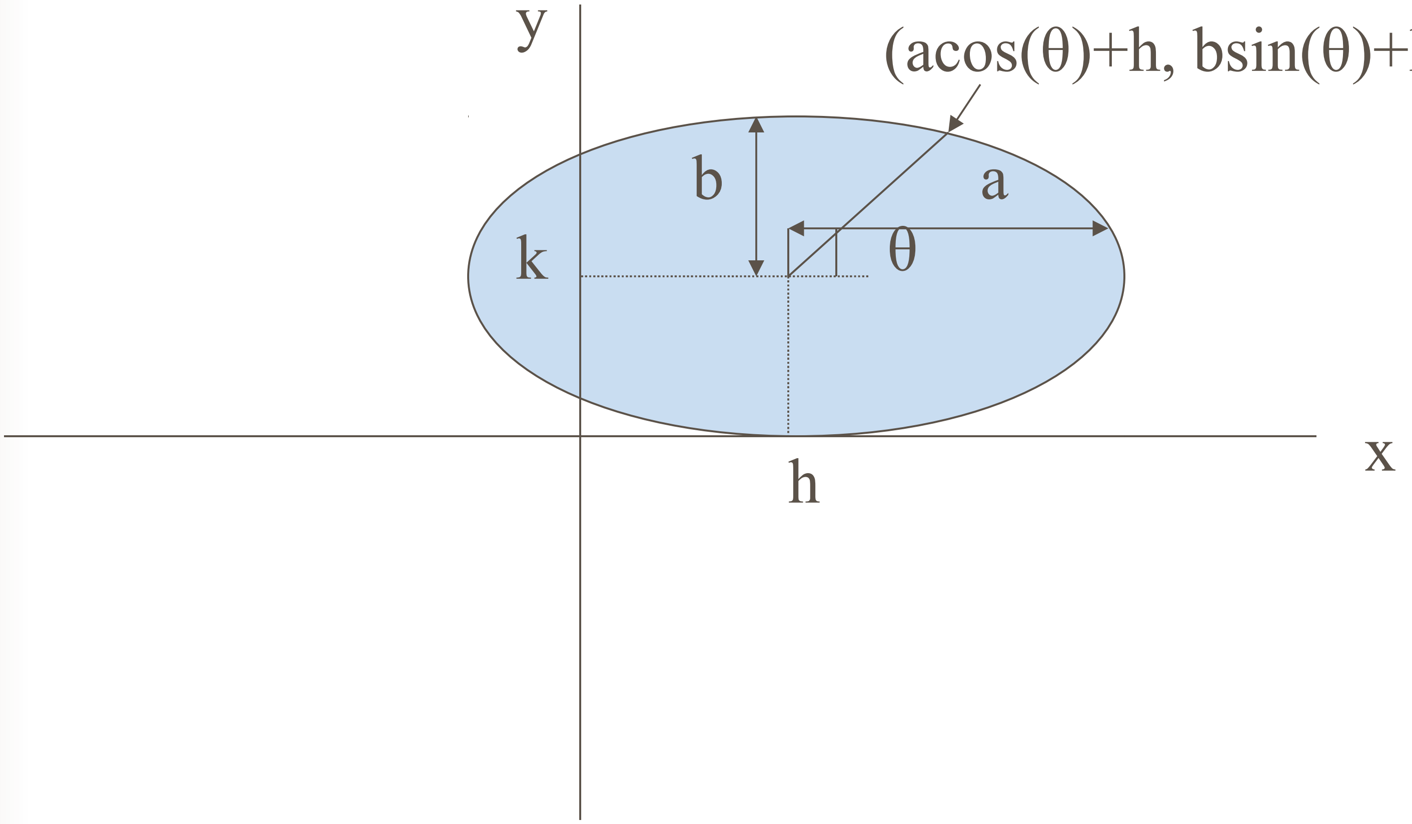
Where (x, y) = the current coordinates

a = length of major axis

b = length of minor axis

θ = current angle

The value of θ is varied from 0 to π/2 radians. The remaining points are found by symmetry.



Ellipse Axis Rotation

Since the ellipse shows four-way symmetry, it can easily be rotated 90°. The new equation is found by trading a & b, the values which describe the major & minor axes. The equation is:

$$\frac{(x - h)^2}{b^2} + \frac{(y - k)^2}{a^2} = 1$$

a = length of major axis

b = length of minor axis

In trigonometric method the equations are:

$x = b\cos(\theta) + h$ and $y = a\sin(\theta) + k$

Where (x, y) = the current coordinates

a = length of major axis

b = length of minor axis

θ = current angle

Assume that the ellipse is rotate through an angle of 90º. This rotation can be accomplished by rotating the x & y axis α degrees.

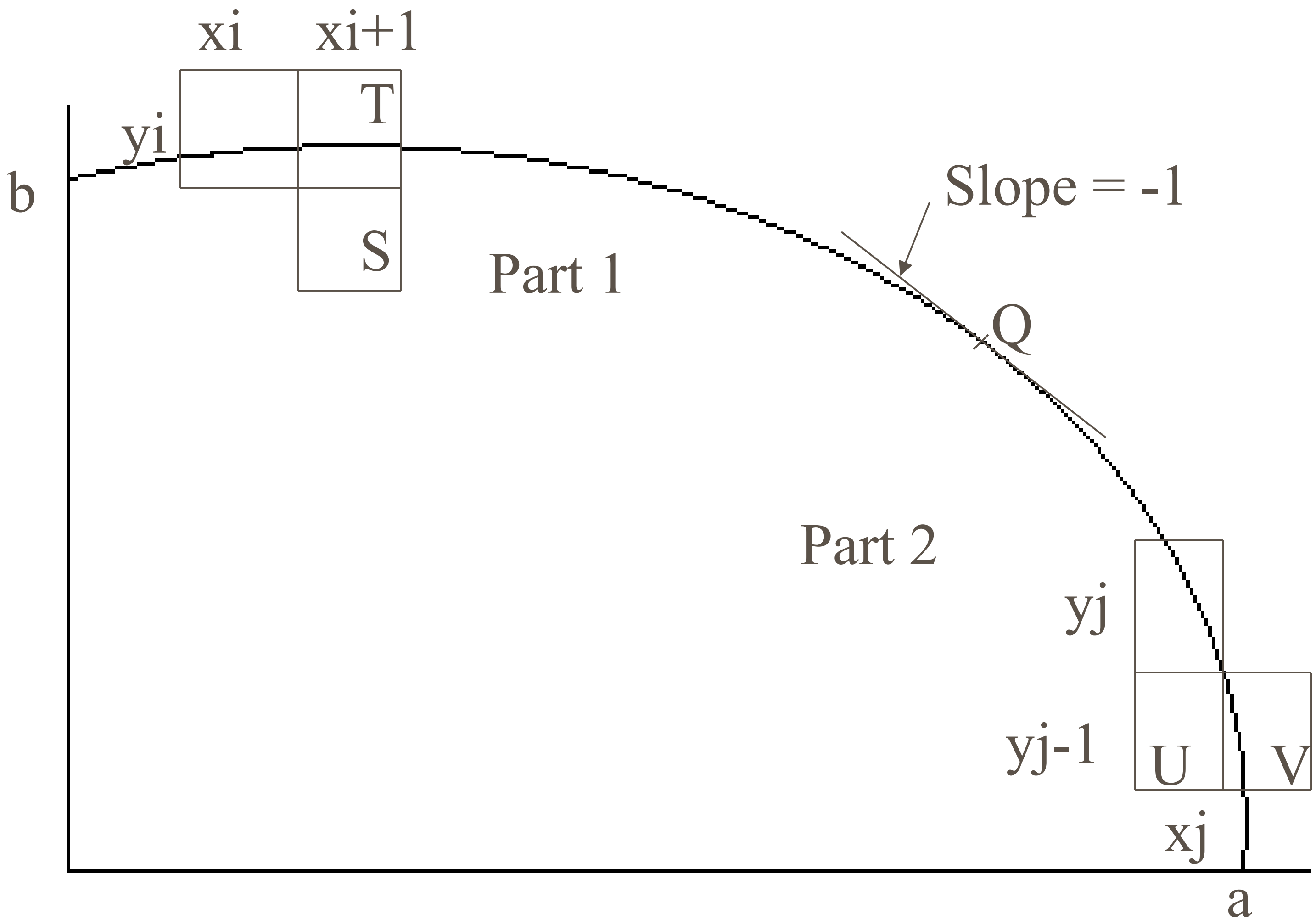
Midpoint Ellipse Algorithm

This is an incremental method for scan converting an ellipse that is centered at origin in standard position i.e with its major & minor axes parallel to coordinate system axis.

It is very similar to midpoint circle algorithm. However, b'coz of the four-way symmetry property we need to consider the entire elliptical curve in the first quadrant.

Let's first rewrite the ellipse equation and define function f that can be used to decide if the midpoint between two candidate pixel is inside or outside the ellipse:

$$f(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = \begin{cases} < 0 & (x,y) \text{ inside} \\ 0 & (x,y) \text{ on} \\ > 0 & (x,y) \text{ outside} \end{cases}$$


xi
xi+1
T
yi
b
Slope = -1
S
Part 1
Q
Part 2
yj
yj-1
U
V
xj
a

Now divide the elliptical curve from (0,b) to (a,0) into two parts at point Q where the slope of the curve is –1.

Slope of the curve is defined by $f(x,y) = 0$ is $dy/dx = - fx/fy$, where fx & fy are partial derivatives of $f(x,y)$ with respect to x & y.

We have $fx = 2b^2x$, $fy = 2a^2y$ &

$dy/dx = - 2b^2x/ 2a^2y$. Hence we can monitor the slope value during the scan-conversion process to detect Q.

Out starting point is (0,b).

Suppose that the coordinates of the last scan converted pixel upon entering step i are (xi, yi). We are to select either $T(x_{i+1}, y_i)$ or $S(x_{i+1}, y_i - 1)$ to be the next pixel. The midpoint of T & S is used to define the following decision parameter.

$$pi = \boldsymbol{f}(x_{i+1}, y_i - ½)$$

$pi = b^2(x_{i+1})^2 + a^2(y_i - ½)^2 - a^2b^2$

➔If $pi < 0$, the midpoint is inside the curve, & we choose pixel T.

➔If $pi \geq 0$, the midpoint is outside or on the curve, & we choose pixel S.

Decision parameter for the next step is:

$p_{i+1} = \boldsymbol{f}(x_{i+1} + 1, y_{i+1} - ½)$

$= b^2(x_{i+1} + 1)^2 + a^2(y_{i+1} - ½)^2 - a^2b^2$

Since $x_{i+1} = xi + 1$, we have

$$p_{i+1} - pi = b^2[(x_{i+1} + 1)^2 - x_{i+1}{}^2]$$

$$+ a^2[(y_{i+1} – ½)^2 – (yi – ½)^2]$$

$$p_{i+1} = pi + 2b^2x_{i+1} + b^2 + a^2[(y_{i+1} – ½)^2 –$$

$$(yi – ½)^2]$$

➔ If T is chosen pixel (meaning $pi < 0$), we have $y_{i+1} = yi$

➔ If S is chosen pixel (meaning $pi \geq 0$), we have $y_{i+1} = yi – 1$. Thus we can express

p_{i+1} in terms of pi and (x_{i+1}, y_{i+1}):

$$p_{i+1} = \begin{cases} pi + 2b^2x_{i+1} + b^2 & \text{if } pi < 0 \\ pi + 2b^2x_{i+1} + b^2 - 2a^2y_{i+1} & \text{if } pi > 0 \end{cases}$$

The initial value for this recursive expression can be obtained by evaluating the original definition of pi with (0,b):

$$p1 = b^2 + a^2(b - ½)^2 - a^2b^2$$

$$= b^2 - a^2b + a^2/4$$

We now move on to derive a similar formula for part 2 of the curve

Suppose pixel (xj, yj) has just been scan converted upon entering step j. The next pixel is either $U(x_j, y_j-1)$ or $V(x_j+1, y_j-1)$. The midpoint of the horizontal line connecting U & V is used to define the decision parameter.

$$qj = f(x_j + \frac{1}{2}, y_j - 1)$$

$$qj = b^2(x_j + \frac{1}{2})^2 + a^2(y_j - 1)^2 - a^2b^2$$

➔ If $qj < 0$, the midpoint is inside the curve, & we choose pixel V.

➔ If $qj \geq 0$, the midpoint is outside or on the curve, & we choose pixel U. Decision parameter for the next step is:

$$q_{j+1} = \boldsymbol{f}(x_{j+1} + ½, y_{j+1} – 1)$$

$$= b^2(x_{j+1} + ½)^2 + a^2(y_{j+1} – 1)^2 – a^2b^2$$

Since $y_{j+1} = yj - 1$, we have

$$q_{j+1} - qj = b^2[(x_{j+1} + ½)^2 – (x_j + ½)^2] +$$

$$a^2[(y_{j+1} – 1)^2 – (y_{j+1})^2]$$

$$q_{j+1} = qj + b^2 [(x_{j+1} + ½)^2 – (x_j + ½)^2]$$

$$- 2a^2y_{j+1} + a^2$$

➔ If V is chosen pixel (meaning $qj < 0$), we have $x_{j+1} = xj + 1$

➔ If U is chosen pixel (meaning $pi > 0$), we have $x_{j+1} = xj$. Thus we can express

q_{j+1} in terms of qj and (x_{j+1}, y_{j+1}):

$$q_{j+1} = \begin{cases} qj + 2b^2 x_{j+1} - 2a^2 y_{j+1} + a^2 & \text{if } qj < 0 \\ qj - 2a^2 y_{j+1} + a^2 & \text{if } qj > 0 \end{cases}$$

The initial value for this recursive expression is computed using the original definition of qj

And the coordinates (x_k, y_k) of the last pixel chosen for part 1 of the curve:

$$q1 = f(x_k + ½, y_k - 1)$$

$$= b^2(x_k + ½)^2 - a^2(y_k - 1)^2 - a^2b^2$$

Algorithm:

int x=0, y=b; (starting point)

int fx =0, fy = $2a^2b$ (Initial partial derivative)

int p = $b^2 - a^2b + a^2/4$

```
while (fx < fy) /* |slope| < 1 */
 {  setPixel(x,y)
    x++;
    fx = fx + 2b²;
if (p < 0)
        p = p + fx + b²;
else {
        y--;
        fy = fy – 2a²;
```

```
  p= p + fx + b² – fy;
  }
}
setPixel(x,y);     /* set pixel at (x_k,y_k) */
p = b²(x+0.5)² + a²(y-1)² - a²b²;
while (y > 0) {
y--;
fy = fy – 2a²;
if (p >= 0)
```

```
  p = p – fy + a²;
else {
   x++;
   fx = fx + 2b²;
   p = p + fx – fy +a²;
 }
 setPixel(x,y);
}
```