Lecture 4

Circle Generating Algorithms

Since the circle is a frequently used component in pictures and graphs, a procedure for generating either full circles or circular arcs is included in most graphics packages. More generally, a single procedure can be provided to display either circular or elliptical curves.

A circle is defined as the set of points that are all at a given distance \( r \) from a center position \((x_c, y_c)\) (Fig. 4-1). This distance relationship is expressed by the Pythagorean Theorem in Cartesian coordinates as:

\[
(x - x_c)^2 + (y - y_c)^2 = r^2
\]

We could use this equation to calculate the position of points on a circle circumference by stepping along the x axis in unit steps from \( x_c - r \) to \( x_c + r \) and calculating the corresponding y values at each position as:

\[
y = y_c \pm \sqrt{r^2 - (x - x_c)^2}
\]

![Figure 4-1: a circle](image)

But this is not the best method for generating a circle because:

- It involves considerable computation at each step.
- The spacing between plotted pixel positions is not uniform (gap problem).

The gap problem can be solved by:
- Interchanging x and y (stepping through y values and calculating x values) whenever the absolute value of the slope of the circle is greater than 1. (Increases the computation and processing required by the algorithm).

\[ x = x_c \pm \sqrt{r^2 - (y - y_c)^2} \]

- Calculate points along the circular boundary using polar coordinates r and. Expressing the circle equation in parametric polar form yields the pair of equations:

\[ x = x_c + r \cos \theta, \quad y = y_c + r \sin \theta \quad (3) \]

If the center of the circle is the origin (0, 0)

\[ x^2 + y^2 = R^2 \]
\[ x = R \sin \theta \]
\[ y = R \cos \theta \]

The equation of a circle

When a display is generated with these equations using a fixed angular step size, a circle is plotted with equally spaced points along the circumference. The step size chosen for \( \theta \) depends on the application and the display device. Larger angular separations along the circumference can be connected with straight line segments to approximate the circular path. For a more continuous boundary on a raster display, we can set the step size at \( l/r \). This plots pixel positions that are approximately one unit apart.

Determining pixel positions along a circle circumference using either Eq. (2), (3) still requires a good deal of computation time.

- The Cartesian equation (2) involves multiplications and square root calculations.
- Parametric equations (3) contain multiplications and trigonometric calculations.

More efficient circle algorithms are based on incremental calculation of decision parameters, as in the Bresenham’s line algorithm, which involves only simple integer operations.
Bresenham's line algorithm for raster displays is adapted to circle generation by by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.

1. **Brute force Algorithm**

It’s simple to write a piece of code (see Example 4 below) which iterates many values of $\theta$, directly calculates the $x$ and $y$ values and then plots points at those coordinates. Regrettably, this is extremely inefficient and also relies on the availability of sine and cosine tables.

```java
public void bruteCircle(int x0, int y0, int radius) {
    for(int theta=0; theta<360; theta++) {
        int x = round(radius * cos(theta));
        int y = round(radius * sin(theta));
        Plot(x,y)
    }
} //bruteCircle
```

2. **Brute force (Cheating with 8 arcs) or Eight-way symmetry Algorithm**

We can improve the process of the previous section by taking greater advantage of the symmetry in a circle. Consider first a circle centered at the origin that is $(0, 0)$. If the point $(x, y)$ is on the circle the new can trivially compute seven other points on the circle as in Fig. Eight symmetrical points on a circle

We need to compute only one 45-degree segment to determine the circle completely. For a circle centered at the origin $(0, 0)$, the eight symmetrical points can be displayed with procedure circlepoints(). This procedure can be easily generalized to the case of circles with arbitrary...
centers. Suppose the point \((x_{\text{center}}, y_{\text{center}})\) is the center of the circle. Then the above function can be modified as

```java
Public void bruteCircleWithCheating (int x0, int y0, int radius)
{
    for(int theta=0; theta<45; theta++)
    {
        int x = round(radius * cos(theta));
        int y = round(radius * sin(theta));
        circlepoints (x0, y0, x,y)
    }
}
//bruteCircle
```

The use of this symmetry means that values for \(x\) and \(y\) only need to be calculated for the octant where \(0 < x \leq y\) (the arc indicated above). Start with \(x=\) radius and \(y=0\). Iterate until \(y=x\). As you calculate and plot more points, 8 arcs “grow” and eventually meet to form the circle.

8 arcs a-growing
3. DDA ALGORITHM - An incremental algorithm

A slightly better (quicker) way of calculating x and y values is the “digital differential analyzer” algorithm which doesn’t require sine and cosine tables but instead is based upon calculating the gradient of a circle at a point and using that to approximate the position of the next point. Our starting point is the equation of a circle (of radius R) centered on the origin:

\[ x^2 + y^2 = R^2 \]
\[ y^2 = R^2 - x^2 \]
\[ y = \sqrt{R^2 - x^2} \]

By differentiating this with respect to x, we find that the gradient (m) at any point on the circle is

\[ m = \frac{dy}{dx} = \frac{-2x}{2\sqrt{R^2 - x^2}} = \frac{-2x}{2y} = \frac{-x}{y} \]

Assume that the values of \( x_i, y_i \) for one point of the circle are known (given). From them we would ideally like to calculate the values of x and y at Point T (the true values for x and y at the next point round the circle). In fact, we can calculate the x and y values at Point A quickly and by making sure than the distance between successive points (\( \varepsilon \)) is small, then Point A will lie close to the true circle.

\[ \Delta y = A_{y_{i+1}} - y_i \]
\[ \Delta x = A_{x_{i+1}} - x_i \]

For small increments (\( \sum \)) points T and A are close. i.e.
\[ \frac{\Delta y}{\Delta x} \approx \frac{A_{y_{i+1}} - y_i}{A_{x_{i+1}} - x_i} \approx -\frac{\varepsilon x_i}{\varepsilon y_i} \]

The value for \( \varepsilon \) must be selected to produce a change in \( x \) and \( y \) of around 1 pixel at a maximum. In practice this can be achieved by selecting \( \varepsilon \) such that:

\[ \varepsilon = \frac{1}{2^n} \text{ where } 2^{n-1} \leq R < 2^n \]

The DDA can be combined with our `eightPlot()` method to reduce still further the number of calculations required.

```java
public void DDACircle(int x0, int y0, int radius)
{
    float x = radius;
    float y = 0;
    int P = 1;
    for(int i=1; radius>P; i++)
    { P *= 2; }
    float E = 1/(float)P;
    while (y <= x)
    {
        x = x + (E*y);
        y = y - (E*x);
        Circlepoints (x0, y0, x, y);
    }
} //incrementalCircle
```

**Another solution**

To write an algorithm to generate a circle of the form \((x - x_c)^2 + (y - y_c)^2 = r^2\) by the help of digital differential analyzer where \((x_c, y_c)\) is the center of the circle and \( r \) is the radius.

1. START
2. Get the values of the center \((x_c, y_c)\) and radius \(r\) of the circle.
3. Find the polar co-ordinates by
4. \( x = x_c + r \cos \theta \)
5. \( y = y_c + r \sin \theta \)
6. Plot the points \((x, y)\) corresponding to the values of \( \theta \), where \( \theta \) lies between 0 and 360.
7. STOP
4. MIDPOINT CIRCLE ALGORITHM (Brenham’s Circle Algorithm)

Like Bresenham’s line algorithm, this approach is based on limiting the choice of the next pixel to be plotted to two alternatives, then creating and testing a decision variable to determine which alternative is actually plotted.

Whereas Bresenham’s line algorithm calculated the “errors” $\Delta y_1$ and $\Delta y_0$ based on the straight line equation (Equation 2), here we use the equation of a circle - Equation 14

$$D(S_i) = [(x_{i-1} + 1)^2 + (y_{i-1})^2] - R^2$$

$$D(T_i) = [(x_{i-1} + 1)^2 + (y_{i-1})^2] - R^2$$

$D(S_i)$ and $D(T_i)$ represent the differences between the squared distance between the centre of the circle and the middle of pixels $S$ and $T$. Whichever is smallest corresponds to the pixel that should be plotted.
To combine the two calculations into one “decision variable” is simply a matter of subtracting the two expressions:
\[ d_i = |D(S_i)| - |D(T_i)| \]

We can then make the decision based upon whether \( d_i \) is positive or negative:
\[
\begin{align*}
 \text{if } d_i > 0 & \text{ then} \\
 & \text{plot pixel } T \\
\text{else} & \\
 & \text{plot pixel } S
\end{align*}
\]

Actually, because the decision is based purely upon the sign of \( d_i \) (and not its value), Equation 23 can be simplified to
\[ d_i = D(S_i) + D(T_i) \]

After some re-arrangement (which is left as a exercise for the reader) the following can be obtained:
\[ d_i = 3 - 2R \]
If pixel \( S \) is chosen (based upon the sign of \( d_i \) in Equation 25 then
\[ d_{i+1} = d_i + 4x_i + 6 \]
or if pixel \( T \) is chosen
\[ d_{i+1} = d_i + 4(x_i+1 - y_i+1) + 10 \]

**Midpoint Circle Algorithm**

1. Input radius \( r \) and circle center \((x_c, y_c)\), and obtain the first point on the circumference of a circle centered on the origin as \((x_0, y_0) = (0, r)\)
2. Calculate the initial value of the decision parameter as
\[ p_0 = \left( \frac{5}{4} \right) - r \]
3. At each \( x_k \) position, starting at \( k = 0 \), perform the following test: If \( p_0 < 0 \) the next point along the circle centered on \((0,0)\) is \((x_{k+1}, y_k)\) and
\[ p_{k+1} = p_k + 2(x_{k+1}) + 1 \]
Otherwise, the next point along the circle is \((x_{k+1}, y_{k-1})\) and
\[ p_{k+1} = p_k + 2(x_k + 1) + 1 - 2(y_k + 1) \]
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position \((x, y)\) onto the circular path centered on \((x_c, y_c)\) and plot the coordinate values: \( x = x + x_c, y = y + y_c \)
6. Repeat steps 3 through 5 until \( x \geq y \).
#include "device.h"
void circleMidpoint (int xCenter, int yCenter, int radius)
{
    int x = 0;
    int y = radius;
    int p = 1 - radius;
    void circlePlotPoints (int, int, int, int);
    /* Plot first set of points */
circlePlotPoints(xcenter, ycenter, x, y);
    while (x < y) {
        x++;
        if (P < 0) { p += 2 * x + 1; }
        else { y--; p += 2 * (x - y) + 1; }
    }
    void circlePlotPoints (int xCenter, int yCenter, int x, int y)
    {
        Setpixel (xCenter + x, yCenter + y);
        setpixel (xCenter - x, yCenter + y);
        setpixel (xCenter + x, yCenter - y);
        setpixel (xCenter - x, yCenter - y);
        setpixel (xCenter + y, yCenter + x);
        setpixel (xCenter - y, yCenter + x);
        setpixel (xCenter + y, yCenter - x);
        setpixel (xCenter - y, yCenter - x);
    }

Example:
Given a circle radius \( r = 10 \), we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from \( x = 0 \) to \( x = y \).

The initial value of the decision parameter is

\[ p_0 = 1 - r = -9 \]

For the circle centered on the coordinate origin, the initial point is \((x_0, y_0) = (0, 10)\), and initial increment terms for calculating the decision parameters are:

\[ 2x_0 = 0, 2y_0 = 20 \]

Successive decision parameter values and positions along the circle path are calculated using the midpoint method as:

<table>
<thead>
<tr>
<th>( k )</th>
<th>( p_k )</th>
<th>( (x_{k-1}, y_{k-1}) )</th>
<th>( 2x_{k+1} )</th>
<th>( 2y_{k+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-9</td>
<td>(1,10)</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>(2,10)</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>(3,10)</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>(4,9)</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>-3</td>
<td>(5,9)</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>(6,8)</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>(7,7)</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>
Example:
Given a circle radius \( r = 10 \), we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from \( x = 0 \) to \( x = y \). The initial value of the decision parameter is

\[
p_0 = 1 - r = -9
\]

For the circle centered on the coordinate origin, the initial point is \((x_0, y_0) = (0, 10)\), and initial increment terms for calculating the decision parameters are

\[
2x_0 = 0, \quad 2y_0 = 20
\]

Successive midpoint decision parameter values and the corresponding coordinate positions along the circle path are listed in the following table.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( p_k )</th>
<th>((x_{k+1}, y_{k+1}))</th>
<th>(2x_{k+1})</th>
<th>(2y_{k+1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-9</td>
<td>((1, 10))</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>((2, 10))</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>((3, 10))</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>((4, 9))</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>-3</td>
<td>((5, 9))</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>((6, 8))</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>((7, 7))</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>
Ellipse Generating Algorithms

An ellipse is as a modified circle whose radius varies from a maximum value in one direction to a minimum value in the perpendicular direction. The straight-line segments through the interior of the ellipse in these two perpendicular directions are referred to as the **major** and **minor** axes of the ellipse.

An ellipse can be given in terms of the distances from any point on the ellipse to two fixed positions, called the foci of the ellipse. The sum of these two distances is the same value for all points on the ellipse. If the distances to the two focus positions from any point \( P = (x, y) \) on the ellipse are labeled \( d_1 \) and \( d_2 \), then the general equation of an ellipse can be stated as

\[
d_1 + d_2 = \text{constant}
\]

Expressing distances \( d_1 \) and \( d_2 \) in terms of the focal coordinates \( F_1 = (x_1, y_1) \) and \( F_2 = (x_2, y_2) \), we have

\[
\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = \text{constant}
\]

By squaring this equation, isolating the remaining radical, and squaring again, we can rewrite the general ellipse equation in the form

\[
A x^2 + B y^2 + C x y + D x + E y + F = 0
\]
\[
\left( \frac{x - x_c}{r_x} \right)^2 + \left( \frac{y - y_c}{r_y} \right)^2 = 1
\]

\begin{align*}
x &= x_c + r_x \cos \theta \\
y &= y_c + r_y \sin \theta
\end{align*}

**Midpoint Ellipse Algorithm**

1. Input \( r_x, r_y, \) and ellipse center \((x_c, y_c)\), and obtain the first point on an ellipse centered on the origin as

\[
(x_0, y_0) = (0, r_y)
\]

2. Calculate the initial value of the decision parameter in region 1 as

\[
p_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2
\]
3. At each $x_k$ position in region 1, starting at $k = 0$, perform the following test. If $p_{1k} < 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_{k+1}, y_k)$ and

$$ p_{1k+1} = p_{1k} + 2r_x^2 x_{k+1} + r_y^2 $$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$ p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2 $$

with

$$ 2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2, \quad 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2 $$

and continue until $2r_y^2 x \geq 2r_x^2 y$.

4. Calculate the initial value of the decision parameter in region 2 as

$$ p_{20} = r_x^2 \left( x_0 + \frac{1}{2} \right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2 $$

where $(x_0, y_0)$ is the last position calculated in region 1.

5. At each $y_k$ position in region 2, starting at $k = 0$, perform the following test. If $p_{2k} > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and

$$ p_{2k+1} = p_{2k} - 2r_x^2 y_{k+1} + r_x^2 $$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$ p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2 $$

using the same incremental calculations for $x$ and $y$ as in region 1. Continue until $y = 0$.

6. For both regions, determine symmetry points in the other three quadrants.

7. Move each calculated pixel position $(x, y)$ onto the elliptical path centered on $(x_c, y_c)$ and plot the coordinate values:

$$ x = x + x_c, \quad y = y + y_c $$
Example

Given input ellipse parameters \( r_x = 8 \) and \( r_y = 6 \), we illustrate the steps in the midpoint ellipse algorithm by determining raster positions along the ellipse path in the first quadrant. Initial values and increments for the decision parameter calculations are

\[
\begin{align*}
2r_y^2x &= 0 \quad \text{(with increment } 2r_y^2 = 72) \\
2r_x^2y &= 2r_x^2r_y \quad \text{(with increment } -2r_x^2 = -128)
\end{align*}
\]

For region 1, the initial point for the ellipse centered on the origin is \((x_0, y_0) = (0, 6)\), and the initial decision parameter value is

\[ p_{10} = r_y^2 - r_x^2r_y + \frac{1}{4}r_x^2 = -332 \]

Successive midpoint decision-parameter values and the pixel positions along the ellipse are listed in the following table.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( p_{1k} )</th>
<th>((x_{k+1}, y_{k+1}))</th>
<th>( 2r_y^2x_{k+1} )</th>
<th>( 2r_x^2y_{k+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-332</td>
<td>(1, 6)</td>
<td>72</td>
<td>768</td>
</tr>
<tr>
<td>1</td>
<td>-224</td>
<td>(2, 6)</td>
<td>144</td>
<td>768</td>
</tr>
<tr>
<td>2</td>
<td>-44</td>
<td>(3, 6)</td>
<td>216</td>
<td>768</td>
</tr>
<tr>
<td>3</td>
<td>208</td>
<td>(4, 5)</td>
<td>288</td>
<td>640</td>
</tr>
<tr>
<td>4</td>
<td>-108</td>
<td>(5, 5)</td>
<td>360</td>
<td>640</td>
</tr>
<tr>
<td>5</td>
<td>288</td>
<td>(6, 4)</td>
<td>432</td>
<td>512</td>
</tr>
<tr>
<td>6</td>
<td>244</td>
<td>(7, 3)</td>
<td>504</td>
<td>384</td>
</tr>
</tbody>
</table>

We now move out of region 1, since \( 2r_y^2x > 2r_x^2y \).

For region 2, the initial point is \((x_0, y_0) = (7, 3)\) and the initial decision parameter is

\[ p_{20} = f_{\text{ellipse}} \left( 7 + \frac{1}{2}, 2 \right) = -151 \]

The remaining positions along the ellipse path in the first quadrant are then calculated as

<table>
<thead>
<tr>
<th>( k )</th>
<th>( p_{1k} )</th>
<th>((x_{k+1}, y_{k+1}))</th>
<th>( 2r_y^2x_{k+1} )</th>
<th>( 2r_x^2y_{k+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-151</td>
<td>(8, 2)</td>
<td>576</td>
<td>256</td>
</tr>
<tr>
<td>1</td>
<td>233</td>
<td>(8, 1)</td>
<td>576</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>745</td>
<td>(8, 0)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>