Scan-Conversion

Topics

- Scan-Converting
- a Point,
- a Line,
- a Circle,
- an Ellipse,
- Arcs and Sectors
- a Rectangle
- a Character

Scan-Converting a Point

- A mathematical **point (x, y)** where x and y are real numbers within an image area, needs to be scan-converted to a pixel at location (x', y').
- This may be done by making x' to be the **integer part of x** and y' the **integer part of y**.
- In other words,

  x' = Floor(x) and

  y' = Floor(y),
- the function Floor() returns the largest integer that is less than or equal to the argument.

- All points that satisfy

  $x' <= x <= x'+1$ and $y' <= y <= y'+1$ are mapped to pixel (x', y').

For example,

- **point p_1 (1.7, 0.8)** is represented by **pixel (1, 0)**.
- **point p_2 (2.3, 1.9)** is represented by **pixel (2, 1).**
- **point p_2 (3.7, 4.9)** is represented by **pixel (3, 4).**
- **point p_2 (7.6, 8.9)** is represented by **pixel (7, 8).**

Scan-Converting a Line

- A line in computer graphics typically refers to a **line segment**, which is a portion of a straight line that extends indefinitely in opposite directions.
- It is defined by its two endpoints and the line **equation $y = mx + b$**, where m is called **the slope** and b is **the y intercept of the line.**
- Two endpoints are described by **$P_1(x_1, y_1)$** and **$P_2(x_2, y_2)$**.
- The line equation describes the coordinates of all points the lie between the two endpoints.

- The **slope-intercept equation** is not suitable for vertical lines.
- **Horizontal, vertical,** and **diagonal ($|m|=1$) lines** can and often should, be handled as special cases without going through the following scan-conversion algorithms.
- **A line connects two points.**
- **It is a basic element in graphics.**
- **To draw a line, you need two points between which you can draw a line.**
- **In the following three algorithms, we refer the one point of line as (x_0,y_0) and the second point of line as (x_1, y_1).**

Algorithms are:

- DDA (Digital Differential Analyzer) Algorithm
- Bresenham’s Line Algorithm
- Mid-Point Algorithm

DDA (Digital Differential Analyzer) Algorithm

Digital Differential Analyzer(*DDA)* algorithm is the **simple line generation algorithm** which is explained step by step here.

Step 1: Get the input of two end points $\boldsymbol{(x_0, y_0)}$ and $\boldsymbol{(x_1, y_1)}$**.**
Step 2: Calculate the **differences** between two end points.

$dx = x_1 - x_0$
$dy = y_1 - y_0$

Step 3: Based on the calculated difference in Step 2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (dx > dy)
        Steps = absolute(dx);
else
        Steps = absolute(dy);
```

Step 4: Calculate the increment in x coordinate and y coordinate.

$X_{increment}$ = dx / (float) steps;
$Y_{increment}$ = dy / (float) steps;

Step 5: Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for(int v=0; v < Steps; v++)
{
        x = x + Xincrement;
        y = y + Yincrement;
        putpixel(x,y);
}
```

Bresenham's Line Algorithm

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

Step 1: Input the two end-points of line, storing the left end-point in $(x0, y0)$.
Step 2: Plot the point $(x0, y0)$.
Step 3: Calculate the constants dx, dy, 2dy, and $2dy - 2dx$ and get the first value for the decision parameter as −

$$p0 = 2dy - dx$$

Step 4: At each Xk along the line, starting at k = 0, perform the following test −

If $pk < 0$, the next point to plot is $(xk + 1, yk)$ and $pk{+}1 = pk + 2dy$

Otherwise, $pk{+}1 = pk + 2dy - 2dx$

Step 5: Repeat step 4 $dx - 1$ times. For m > 1, find out whether you need to increment x while incrementing y each time. After solving, the equation for decision parameter Pk will be very similar, just the x and y in the equation gets interchanged.

Bresenham's Line Algorithm

In short,
Bresenham's algorithm for scan-converting a line from $P_1(x_1, y_1)$ to $P_2(x_2, y_2)$ with $x_1' < x_2'$ and o<m<1 can be stated as follows:

```
int x = x1, y =y1;
int dx = x2 - x1, dy = y2 - y1, dT = 2(dy - dx), dS = 2dy;
int d = 2dy – dx;
setPixel(x,y);
while(x<x2)
{
        x++;
        if(d<0)
                d = d + dS;
        else
        {
                y++;
                d = d + dT;
        }
        setPixel(x, y);
}
```

Bresenham’s Line Algorithm: Description

- Here we first initialize decision variable d and set pixel P_1.
- During each iteration of the while loop, we increment x to the next horizontal position, then use the current value of d to select the bottom or top (increment y) pixel and update d, and at the end set the chosen pixel.

- As for lines that have other m values we can make use of the fact that they can be mirrored either horizontally, vertically, or diagonally into this 0^0 to 45^0 angle range.

- For example, a line from (x_1', y_1') to (x_2', y_2') with $-1 <= m < 0$ has a horizontally mirrored counterpart from $(x_1', -y_1')$ to $(x_2', -y_2')$ with $0 <= m < 1$.
- We can simply use the algorithm to scan-convert this counterpart, but negate the y coordinate at the end of each iteration to set the right pixel for the line.
- For a line whose slope is in the 45^0 to 90^0 range, we can obtain its mirrored counterpart by exchanging the x and y coordinates of its endpoints.
- We can then scan-convert this counterpart but we must exchange x and y in the call to setPixel.

Scan-Converting a Circle

- A circle is a symmetrical figure.
- Any circle-generating algorithm can take advantage of the circle's symmetry to plot eight points for each value that the algorithm calculates.
- Eight-way symmetry is used to reflecting each calculated point around each 45^0 axis.

- For example, if **point 1** were calculated with a circle algorithm, seven more points could be found by reflection.

  - The reflection is accomplished by reversing the x, y coordinates as in **point 2**,
  - reversing the x, y coordinates and reflecting about the y axis as in **point 3**,
  - reflecting about the y axis as in **point 4**,
  - switching the signs of x and y as in **point 5**,
  - reversing the x, y coordinates and reflecting about the x axis as in **point 6**,
  - reversing the x, y coordinates and reflecting about the y axis as in **point 7**, and
  - reflecting about the x axis as in **point 8**.

Scan-Converting a Circle …

To summarize,

$p_1 = (x, y)$ $\qquad p_5 = (-x, -y)$

$p_2 = (y, x)$ $\qquad p_6 = (-y, -x)$

$p_3 = (-y, x)$ $\qquad p_7 = (y, -x)$

$p_4 = (-x, y)$ $\qquad p_8 = (x, -y)$

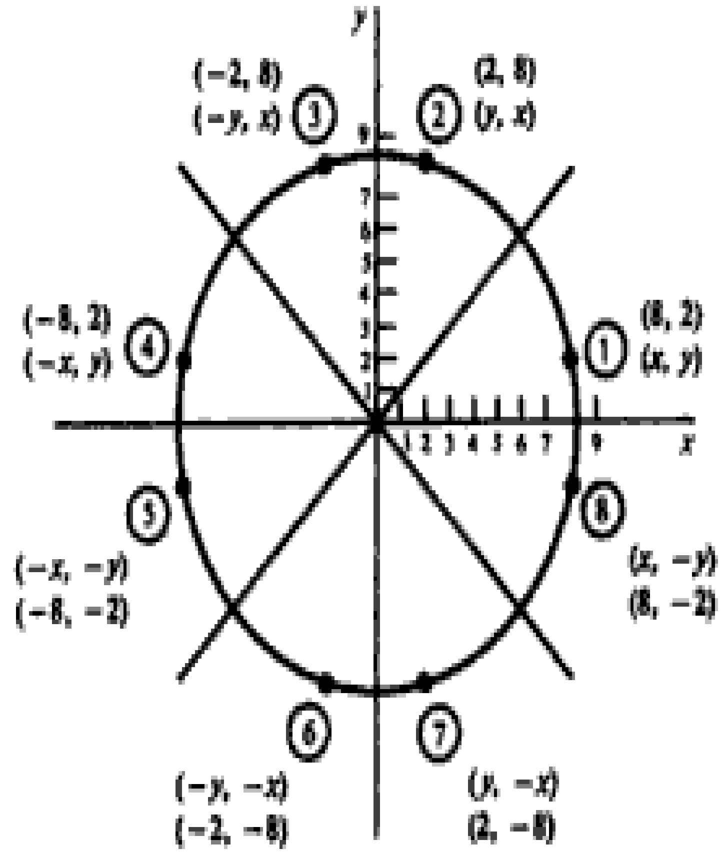


Fig. 3-4 Eight-way symmetry of a circle.

Defining a Circle

- There are **two standard methods** of mathematically defining a circle centered at the origin.
  - **Polynomial Method**
  - **Trigonometric Method**

- The first method defines a circle with the second-order polynomial equation:

$$y^2 = r^2 - x^2,$$

where x = the x coordinate,
y = the y coordinate and
r = the circle radius.

- With this method, each x coordinate in the sector, from 90^0 to 45^0, is found by stepping
  - x from 0 to $r/(\sqrt{2})$, and
  - each y coordinate is found by evaluating $\sqrt{(r^2-x^2)}$ for each step of x.

- This is a very inefficient method, however, because for each point both x and r must be squared and subtracted from each other, then the square root of the result must be found.

Defining a Circle

- The second method of defining a circle makes use of trigonometric functions:

  $x = r \cos \theta$ and

  $y = r \sin \theta$

  where θ = current angle

  r = circle radius

  x = x coordinate

  y = y coordinate

- By this method, θ is stepped from 0 to $\pi/4$, and each value of x and y is calculated.
- However, computation of the values of $\sin \theta$ and $\cos \theta$ is even more time-consuming than the calculations required by the first method.

Mathematical Problems

Solved Problems

Problem 01: Indicate which raster locations would be chosen by Bresenham's Algorithm when scan-converting a line from pixel coordinate (1, 1) to pixel coordinate (8,5).

Indicate which raster locations would be chosen by Bresenham's algorithm when scan-converting a line from pixel coordinate (1, 1) to pixel coordinate (8, 5).

SOLUTION

First, the starting values must be found. In this case

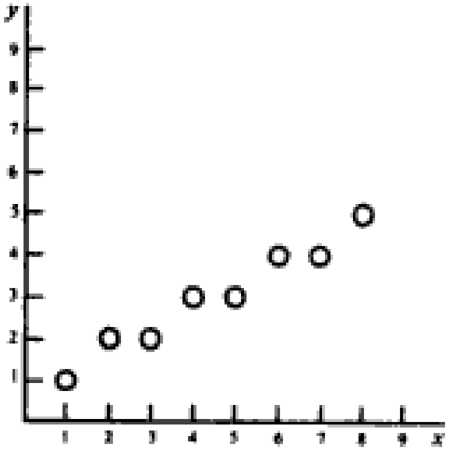
$$dx = x_2 - x_1 = 8 - 1 = 7 \qquad dy = y_2 - y_1 = 5 - 1 = 4$$

Therefore:

$$Inc_1 = 2dy = 2 \times 4 = 8$$
$$Inc_2 = 2(dy - dx) = 2 \times (4 - 7) = -6$$
$$d = Inc_1 - dx = 8 - 7 = 1$$

The following table indicates the values computed by the algorithm (see also Fig. 3-33).

d	x	y
1	1	1
$1 + Inc_2 = -5$	2	2
$-5 + Inc_1 = 3$	3	2
$3 + Inc_2 = -3$	4	3
$-3 + Inc_1 = 5$	5	3
$5 + Inc_2 = -1$	6	4
$-1 + Inc_1 = 7$	7	4
$7 + Inc_2 = 1$	8	5



Solved Problems

Problem 08: Modify the description of Bresenham's line Algorithm in the text to set all pixels from inside the loop structure.

Modify the description of Bresenham's line algorithm in the text to set all pixels from inside the loop structure.

SOLUTION 1

```
int x = x'_1, y = y'_1;
int dx = x'_2 − x'_1, dy = y'_2 − y'_1, dT = 2(dy − dx), dS = 2dy;
int d = 2dy − dx;
while (x <= x'_2) {
   setPixel(x, y);
   x++;
   if (d < 0)
      d = d + dS;

   else {
      y++;
      d = d + dT;
   }
}
```

SOLUTION 2

```
int x = x'_1 − 1, y = y'_1;
int dx = x'_2 − x'_1, dy = y'_2 − y'_1, dT = 2(dy − dx), dS = 2dy;
int d = −dx;
while (x < x'_2) {
   x++;
   if (d < 0)
      d = d + dS;
   else {
      y++;
      d = d + dT;
   }
   setPixel(x, y);
}
```

Solved Problems

Problem 09: What steps are required to generate a circle using the polynomial method.

What steps are required to generate a circle using the polynomial method?

SOLUTION

1. Set the initial variables: r = circle radius; (h, k) = coordinates of the circle center; $x = 0$; i = step size; $x_{end} = r/\sqrt{2}$.
2. Test to determine whether the entire circle has been scan-converted. If $x > x_{end}$, stop.
3. Compute the value of the y coordinate, where $y = \sqrt{r^2 - x^2}$.
4. Plot the eight points, found by symmetry with respect to the center (h, k), at the current (x, y) coordinates:

$$\begin{array}{ll}
\text{Plot}(x+h, y+k) & \text{Plot}(-x+h, -y+k) \\
\text{Plot}(y+h, x+k) & \text{Plot}(-y+h, -x+k) \\
\text{Plot}(-y+h, x+k) & \text{Plot}(y+h, -x+k) \\
\text{Plot}(-x+h, y+k) & \text{Plot}(x+h, -y+k)
\end{array}$$

5. Increment x: $x = x + i$.
6. Go to step 2.

Solved Problems

Problem 10: What steps are required to scan-convert a circle using the trigonometric method.

What steps are required to scan-convert a circle using the trigonometric method?

SOLUTION

1. Set the initial variables: r = circle radius; (h, k) = coordinates of the circle center; i = step size; $\theta_{end} = \pi/4$ radians $= 45°$; $\theta = 0$.
2. Test to determine whether the entire circle has been scan-converted. If $\theta > \theta_{end}$, stop.
3. Compute the value of the x and y coordinates:
4. Plot the eight points, found by symmetry with respect to the center (h, k), at the current (x, y) coordinates:

$$\begin{array}{ll} \text{Plot}(x+h, y+k) & \text{Plot}(-x+h, -y+k) \\ \text{Plot}(y+h, x+k) & \text{Plot}(-y+h, -x+k) \\ \text{Plot}(-y+h, x+k) & \text{Plot}(y+h, -x+k) \\ \text{Plot}(-x+h, y+k) & \text{Plot}(x+h, -y+k) \end{array}$$

5. Increment θ: $\theta = \theta + i$.
6. Go to step 2.

Solved Problems

Problem 11: What steps are required to scan-convert a circle using Bresenham's algorithm

What steps are required to scan-convert a circle using Bresenham's algorithm?

SOLUTION

1. Set the initial values of the variables: (h, k) = coordinates of circle center; $x = 0$; y = circle radius r; and $d = 3 - 2r$.
2. Test to determine whether the entire circle has been scan-converted. If $x > y$, stop.
3. Plot the eight points, found by symmetry with respect to the center (h, k), at the current (x, y) coordinates:

$$\begin{array}{ll} \text{Plot}(x+h, y+k) & \text{Plot}(-x+h, -y+k) \\ \text{Plot}(y+h, x+k) & \text{Plot}(-y+h, -x+k) \\ \text{Plot}(-y+h, x+k) & \text{Plot}(y+h, -x+k) \\ \text{Plot}(-x+h, y+k) & \text{Plot}(x+h, -y+k) \end{array}$$

4. Compute the location of the next pixel. If $d < 0$, then $d = d + 4x + 6$ and $x = x + 1$. If $d \geq 0$, then $d = d + 4(x - y) + 10$, $x = x + 1$, and $y = y - 1$.
5. Go to step 2.

Solved Problems

Problem 12:

In the derivation of Bresenham's circle algorithm we have used a decision variable $d_i = D(\mathrm{T}) + D(\mathrm{S})$ to help choose between pixels S and T. However, function D as defined in the text is not a true measure of the distance from the center of a pixel to the true circle. Show that when $d_i = 0$ the two pixels S and T are not really equally far away from the true circle.

SOLUTION

Let $d\mathrm{S}$ be the actual distance from S to the true circle and $d\mathrm{T}$ be the actual distance from T to the true circle (see Fig. 3-35). Also substitute x for $x_i + 1$ and y for y_i in the formula for d_i to make the following proof easier to read:

$$d_i = 2x^2 + y^2 + (y-1)^2 - 2r^2 = 0$$

Since $(r + d\mathrm{T})^2 = x^2 + y^2$ and $(r - d\mathrm{S})^2 = x^2 + (y-1)^2$ we have

$$2rd\mathrm{T} + d\mathrm{T}^2 = x^2 + y^2 - r^2 \qquad \text{and} \qquad -2rd\mathrm{S} + d\mathrm{S}^2 = x^2 + (y-1)^2 - r^2.$$

Hence

$$2rd\mathrm{T} + d\mathrm{T}^2 - 2rd\mathrm{S} + d\mathrm{S}^2 = 0$$

$$d\mathrm{T}(2r + d\mathrm{T}) = d\mathrm{S}(2r - d\mathrm{S})$$

Since $d\mathrm{T}/d\mathrm{S} = (2r - d\mathrm{S})/(2r + d\mathrm{T}) < 1$, we have $d\mathrm{T} < d\mathrm{S}$. This means that, when $d_i = 0$, pixel T is actually closer to the true circle than pixel S.

Solved Problems

Problem 13: Write a

Write a description of the midpoint circle algorithm in which decision parameter p is updated using x_{i+1} and y_{i+1} instead of x_i and y_i.

SOLUTION

```
int x = 0, y = r, p = 1 - r;
while (x <= y) {
   setPixel(x, y);
   x++;
   if (p < 0)
      p = p + 2x + 1;
   else {
      y--;
      p = p + 2(x - y) + 1;
   }
}
```

Solved Problems

What steps are required to generate an ellipse using the polynomial method?

SOLUTION

1. Set the initial variables: a = length of major axis; b = length of minor axis; (h, k) = coordinates of ellipse center; $x = 0$; i = step size; $x_{\text{end}} = a$.
2. Test to determine whether the entire ellipse has been scan-converted. If $x > x_{\text{end}}$, stop.
3. Compute the value of the y coordinate:

$$y = b\sqrt{1 - \frac{x^2}{a^2}}$$

4. Plot the four points, found by symmetry, at the current (x, y) coordinates:

$$\begin{array}{ll} \text{Plot}(x + h, y + k) & \text{Plot}(-x + h, -y + k) \\ \text{Plot}(-x + h, y + k) & \text{Plot}(x + h, -y + k) \end{array}$$

5. Increment x: $x = x + i$.
6. Go to step 2.

Solved Problems

What steps are required to scan-convert an ellipse using the trigonometric method?

SOLUTION

1. Set the initial variables: a = length of major axis; b = length of minor axis; (h, k) = coordinates of ellipse center; i = counter step size; $\theta_{end} = \pi/2$; $\theta = 0$.
2. Test to determine whether the entire ellipse has been scan-converted. If $\theta > \theta_{end}$, stop.
3. Compute the values of the x and y coordinates:

$$x = a\cos(\theta) \qquad y = b\sin(\theta)$$

4. Plot the four points, found by symmetry, at the current (x, y) coordinates:

$$\text{Plot}(x + h, y + k) \qquad \text{Plot}(-x + h, -y + k)$$
$$\text{Plot}(-x + h, y + k) \qquad \text{Plot}(x + h, -y + k)$$

5. Increment θ: $\theta = \theta + i$.
6. Go to step 2.

Solved Problems

What steps are required to scan-convert an arc using the trigonometric method?

SOLUTION

1. Set the initial variables: a = major axis; b = minor axis; (h, k) = coordinates of arc center; i = step size; θ = starting angle; θ_1 = ending angle.
2. Test to determine whether the entire arc has been scan-converted. If $\theta > \theta_1$, stop.
3. Compute the values of the x and y coordinates:

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

4. Plot the points at the current (x, y) coordinates: Plot(x, y).
5. Increment θ: $\theta = \theta + i$.
6. Go to step 2.

(Note: for the arc of a circle $a = b$ = circle radius r.)

Solved Problems

What steps are required to generate an arc of a circle using the polynomial method?

SOLUTION

1. Set the initial variables: r = radius; (h, k) = coordinates of arc center; x = x coordinate of start of arc; x_1 = x coordinate of end of arc; i = counter step size.
2. Test to determine whether the entire arc has been scan-converted. If $x > x_1$, stop.
3. Compute the value of the y coordinate:

$$y = \sqrt{r^2 - x^2}$$

4. Plot at the current (x, y) coordinates:

$$\text{Plot}(x + h, y + k)$$

5. Increment x: $x = x + i$.
6. Go to step 2.

Solved Problems

Problem 18:

Write a pseudo-code procedure to implement the boundary-fill algorithm in the text in its basic form, using the 4-connected definition for region pixels.

SOLUTION

```
BoundaryFill (int x, y, fill_color, boundary_color)
{
   int color;
   getPixel(x, y, color);
   if (color != boundary_color && color != fill_color) {
      setPixel(x, y, fill_color);
      BoundaryFill(x + 1, y, fill_color, boundary_color);
      BoundaryFill(x, y + 1, fill_color, boundary_color);
      BoundaryFill(x – 1, y, fill_color, boundary_color);
      BoundaryFill(x, y – 1, fill_color, boundary_color);
   }
}
```

Solved Problems

Write a pseudo-code procedure for generating the Koch curve K_n (after the one in the text for generating C_n).

SOLUTION

```
Koch-curve (float x, y, len, alpha; int n)
{
   if (n > 0) {
      len = len/3;
      Koch-curve(x, y, len, alpha, n − 1);
      x = x + len*cos(alpha);
      y = y + len*sin(alpha);
      Koch-curve(x, y, len, alpha − 60, n − 1);
      x = x + len*cos(alpha − 60);
      y = y + len*sin(alpha − 60);
      Koch-curve(x, y, len, alpha + 60, n − 1);
      x = x + len*cos(alpha + 60);
      y = y + len*sin(alpha + 60);
      Koch-curve(x, y, len, alpha, n − 1);
   } else
      line(x, y, x + len*cos(alpha), y + len*sin(alpha));
}
```

Solved Problems

Problem 20: Presume that the following statement produces a filled triangle with vertices at (x_1, y_1), (x_2, y_2), and (x_3, y_3):

triangle($x_1, y_1, x_2, y_2, x_3, y_3$)

Write a pseudo-code procedure for generating the Sierpinski gasket S_n (after the procedure in the text for generating C_n).

SOLUTION

```
S-Gasket (float x1, y1, x2, y2, x3, y3; int n)
{
   float x12, y12, x13, y13, x23, y23;
   if (n > 0) {
      x12 = (x1 + x2)/2;
      y12 = (y1 + y2)/2;
      x13 = (x1 + x3)/2;
      y13 = (y1 + y3)/2;
      x23 = (x2 + x3)/2;
      y23 = (y2 + y3)/2;
      S-Gasket(x1, y1, x12, y12, x13, y13, n - 1);
      S-Gasket(x12, y12, x2, y2, x23, y23, n - 1);
      S-Gasket(x13, y13, x23, y23, x3, y3, n - 1);
   } else
      triangle(x1, y1, x2, y2, x3, y3);
}
```

Thanks