Output Primitives

Dr. S.M. Malaek
Assistant: M. Younesi
Output Primitives

- **Output Primitives**: Basic geometric structures (points, straight line segment, circles and other conic sections, quadric surfaces, spline curve and surfaces, polygon color areas, and character strings)

- These picture components are often defined in a continuous space.
Output Primitives

- In order to draw the primitive objects, one has to first **scan convert** the object.

- **Scan convert**: Refers to the operation of finding out the location of pixels to the intensified and then setting the values of corresponding bits, in the graphic memory, to the desired intensity code.
Output Primitives

- Each pixel on the display surface has a finite size depending on the screen resolution and hence a pixel cannot represent a single mathematical point.
Scan Converting A Point
Scan Converting A Point

- A mathematical point \((x, y)\) needs to be scan converted to a pixel at location \((x', y')\).
Scan Converting A Point

- $x' = \text{Round}(x)$ and $y' = \text{Round}(y)$
- All points that satisfy: 
  
  $x' \leq x < x' + 1$
  
  $y' \leq y < y' + 1$

are mapped to pixel $(x', y')$
Scan Converting A Point

- $x' = \text{Round}(x + 0.5)$ and $y' = \text{Round}(y + 0.5)$
- All points that satisfy:

\[
\begin{align*}
x' - 0.5 & \leq x < x' + 0.5 \\
y' - 0.5 & \leq y < y' + 0.5
\end{align*}
\]

are mapped to pixel $(x', y')$
Scan Converting A Line
Scan Converting A Line

- The Cartesian slope-intercept equation for a straight line is:

$$y = m \cdot x + b$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - m \cdot x_1$$

$$\Delta y = m \Delta x$$

$$\Delta x = \frac{\Delta y}{m}$$
Scan Converting A Line

- These equations form the basis for determining deflection voltage in analog devices.

\[ \Delta y = m \Delta x \quad |m| < 1 \]

\[ \Delta x = \frac{\Delta y}{m} \quad |m| > 1 \]
Scan Converting A Line

- On raster system, lines are plotted with pixels, and **step size** (horizontal & vertical direction) are constrained by **pixel separation**.
Scan Converting A Line

- We must *sample* a line at discrete positions and determine the nearest pixel to the line at each sampled position.
Digital Differential Analyzer (DDA Algorithm)
DDA Algorithm

- Algorithm is an incremental scan conversion method.
- Based on calculating either $\Delta x$ or $\Delta y$
- If $|m| < 1$, $(\Delta x = 1)$

$$\Delta y = m\Delta x$$
$$y_{k+1} = y_k + m$$
DDA Algorithm

- **If** \(|m|>1, (\Delta y = 1)\)
  \[
  \Delta x = \frac{\Delta y}{m} \\
  x_{K+1} = x_k + \frac{1}{m}
  \]

- **If** \((\Delta x = -1)\)
  \[
  \Delta y = m\Delta x \\
  y_{K+1} = y_k - m
  \]
Bresenham’s Line Algorithm
Bresenham’s Line Algorithm

- A highly efficient **incremental** method for scan converting lines.
- Using only **incremental integer** calculation.

By **testing** the sign of an integer parameter, whose value is proportional to the difference between the separation of two pixel positions from the actual line path.
Bresenham’s Line Algorithm

- A highly efficient \textit{incremental} method for scan converting lines.
- Using only \textit{incremental integer} calculation.

By \textit{testing} the sign of an integer parameter, whose value is proportional to the difference between the separation of two pixel positions from the actual line path.
Bresenham’s Line Algorithm

\[ y = mx + b \]
Bresenham’s Line Algorithm
Bresenham’s Line Algorithm

\[ \begin{align*}
    y_{k+1} & \quad \downarrow \\
    y & \quad \downarrow \\
    y_k & \quad \downarrow \\
    x_{k+1} & \quad \downarrow \\
\end{align*} \]
Bresenham’s Line Algorithm

\[ d_1 = y - y_k = m(x_k + 1) + b - y_k \]

\[ d_2 = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b \]

\[ d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1 \]
Bresenham’s Line Algorithm

\[ P_k = \Delta X (d_1 - d_2) = 2 \Delta Y \cdot x_k - 2 \Delta X \cdot y_k + c \]

\[ c = \text{constante} = 2 \Delta Y + \Delta X (2b -1) \]
Let's get rid of multiplications

\[ P_{k+1} = 2 \Delta Y \cdot x_{k+1} - 2 \Delta X \cdot y + c \]

\[ P_{k+1} - P_k = 2 \Delta Y (x_{k+1} - x_k) - 2 \Delta X (y_{k+1} - y_k) \]

(get rid of the constance)

\[ P_{k+1} = P_k + 2 \Delta Y - 2 \Delta X (y_{k+1} - y_k) \]

\[ P_{k+1} = P_k + 2\Delta Y \]

or

\[ = P_k + 2(\Delta Y - \Delta X) \]

with \((y_{k+1} - y_k) = 0 \text{ or } 1 \) depending on \(P_k\) sign
Bresenham’s Line Algorithm

\[ P_0 = \Delta X (d_1 - d_2) = \Delta X [2m(x_0 + 1) - 2y_0 + 2b - 1] = \Delta X [2(mx_0 + b - y_0) + 2m - 1] \]

\[ P_0 = 2\Delta Y - \Delta X \]

\[ m = \frac{\Delta y}{\Delta x} \]
**Bresenham’s Line Algorithm**

- **Example:** Digitize the line with endpoint (20,10) and (30,18)

  \( \Delta X = 10 \quad \Delta Y = 8 \)

  \( P_0 = 2 \Delta Y - \Delta X = 6 \)

  \( 2 \Delta Y = 16 \)

  \( 2(\Delta Y - \Delta X) = 4 \)

  \[
  P_{k+1} = P_k + 2\Delta Y
  \]

  or

  \[
  = P_k + 2(\Delta Y - \Delta X)
  \]
Circle Generation Algorithms
Circle Generation Algorithms

- The equation of a circle:
  \[(x - x_0)^2 + (y - y_0)^2 = r^2\]

We could solve for \(y\) in terms of \(x\):
\[y = y_0 \pm \sqrt{r^2 - (x - x_0)^2}\]
Circle Generation Algorithms

- The spacing between plotted pixel positions is not uniform.
Circle Generation Algorithms

- Computation can be reduced by considering the symmetry of circles

8-Way symmetry
Midpoint Circle Algorithm
Circle Generation Algorithms

- As in the line algorithm, we sample at unit intervals and determine the closet pixel position to the circle path at each step.
Circle Generation Algorithms

- Points are generated from 90° to 45°, moves will be made only in the +x and −y direction.
- positive x direction over this octant and use a decision parameter
Circle Generation Algorithms

- We define a circle function:

\[ f_{\text{circle}}(x, y) = x^2 + y^2 - r^2 \]

\[
\begin{cases} 
  < 0 & \quad \text{when } f(x,y) < 0 \\
  = 0 & \quad \text{when } f(x,y) = 0 \\
  > 0 & \quad \text{when } f(x,y) > 0 
\end{cases}
\]
Circle Generation Algorithms

- **Midpoint**
  
  $$(x_i + 1, y_i - \frac{1}{2})$$

- Consider the coordinates of the point halfway between pixel T and pixel S.
Circle Generation Algorithms

- We use it to define a decision parameter

\[ p_i = f(x_i + 1, y - \frac{1}{2}) = (x_i + 1)^2 + (y_i - \frac{1}{2})^2 - r^2 \]
Circle Generation Algorithms

- If $p_i < 0$ is negative, the midpoint is inside the pixel, and we choose pixel T.
- If $p_i \geq 0$ we choose pixel S.
Circle Generation Algorithms

- Parameter for the next step is:
  \[ p_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - \frac{1}{2}) - r^2 \]

- since
  \[ x_{i+1} = x_i + 1 \]
Circle Generation Algorithms

- If T is chosen \( (p_i < 0) \) we have:
  \[ y_{i+1} = y_i \]

- If pixel S is chosen \( (p_i \geq 0) \) we have:
  \[ y_{i+1} = y_i - 1 \]
Circle Generation Algorithms

In terms of \((x_i, y_i)\)

\[
p_{i+1} = \begin{cases} 
  p_i + 2(x_i + 1) + 1 & \text{if } p_i < 0 \\
  p_i + 2(x_i + 1) + 1 - 2(y_i - 1) & \text{if } p_i \geq 0 
\end{cases}
\]

In terms of \((x_{i+1}, y_{i+1})\)

\[
p_{i+1} = \begin{cases} 
  p_i + 2x_{i+1} + 3 & \text{if } p_i < 0 \\
  p_i + 2(x_{i+1} - y_{i+1}) + 5 & \text{if } p_i \geq 0 
\end{cases}
\]
Circle Generation Algorithms

Initial value for the decision parameter using the original function of (0,r)

\[ p_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - \frac{1}{2})^2 - r^2 \]

\[ p_0 = (0 + 1)^2 + (r - \frac{1}{2})^2 - r^2 = \frac{5}{4} - r \]

When r is an integer we can simply set

\[ p_0 = 1 - r \]
Circle Generation Algorithms

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

   $$p_0 = \frac{5}{4} - r$$

3. At each $x_k$ position, starting at $k = 0$, perform the following test: If $p_k < 0$, the next point along the circle centered on $(0, 0)$ is $(x_{k+1}, y_k)$ and

   $$p_{k+1} = p_k + 2x_{k+1} + 1$$

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

   $$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

   where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.

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:

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

6. Repeat steps 3 through 5 until $x \geq y$.  


Circle Generation Algorithms

```c
int x = 0;
int y = radius; int p = 1 - radius;
circlePoints(xCenter, yCenter, x, y, pix);
while (x < y) {
    x++;
    if (p < 0) {
        p += 2*x+1;
    } else {
        y--;
        p += 2*(x-y+1);
    }
circlePoints(xCenter, yCenter, x, y, pix);
}
```
Circle Generation Algorithms

- **Example:** A circle radius \( r = 10 \)
- \( x = 0 \) to \( x = y \)
- \( P_0 = 1 - r = -9 \)

\[
p_{i+1} = \begin{cases} 
  p_i + 2x_{i+1} + 1 & \text{if } p_i < 0 \\
  p_i + 2(x_{i+1} - y_{i+1}) + 1 & \text{if } p_i \geq 0 
\end{cases}
\]

\[
P_0 = 1 - r.
\]
Exercises
1. Scan Converting Arcs And Sectors?
Character Generation
Character Generation

Letters, numbers, and other characters can be displayed in a variety of size and styles.
Character Generation

- **Typeface:** The overall design style for a set of characters is called typeface: *Zar, nazanin, Titr.*

- **Font:** Referred to a set of cast metal character forms in a particular size and forma: *10 point Zar.*
Character Generation

- Two different representation are used for storing computer fonts:

1. **Bitmap font** (or **bitmapped font**)

2. **Outline font**
**Bitmap font**

- **Bitmap font (or bitmapped font):** A simple method for representing the character shapes in a particular typeface is to use rectangular grid pattern.
**Bitmap font**

- The character grid only need to be mapped to a frame buffer position.
- Bitmap fonts required more space, because each variation (size and format) must be stored in a font cash.
Outline Font

- Graphic primitives such as lines and arcs are used to define the outline of each character.
- Require less storage since variation does not require a distinct font cash.
Outline Font

- We can produce boldface, italic, or different size by manipulating the curve definition for the character outlines.

- It does take more time to process the outline fonts, because they must be scanned and converted into frame buffer.