OUTPUT PRIMITIVES
(Chapter 3 in *Computer Graphics*)

- Points and Lines
- Line-drawing Algorithms
- Antialiasing Lines
- Line Command
- Fill Areas
- Circle-generating Algorithms
- Other Curves
- Character Generation
- Instruction Sets for Display Processors
Points and Lines

- plotting points
  - illuminate a phosphor dot on a CRT
    - for a random-scan CRT, deflect and activate the electron beam
    - for a raster-scan CRT, set a bit or byte at the appropriate location in the frame buffer

- plotting lines
  - specify the endpoint coordinates of each line segment
  - fill the straight line path between the endpoints
    - analog devices (random-scan CRTs, etc.) produce excellent lines
    - digital devices (raster-scan CRTs, plasma panels, pen plotters) use pixels near the true line, resulting in the "jaggies"
Line-drawing Algorithms

- DDA (Digital Differential Analyzer) algorithms
  - simple DDA
  - symmetric DDA

- Bresenham's algorithm
  - more efficient than DDA algorithms
    - adds
    - shifts
    - compares


Simple DDA

\[
\text{procedure dda (x1, y1, x2, y2 : integer);}
\]
\[
\text{var}
\]
\[
dx, dy, steps, k : integer;
\]
\[
x\text{.increment, y\text{.increment, x, y, : real;}}
\]
begin
\[
dx := x2 - x1;
\]
\[
dy := y2 - y1;
\]
\[
\text{if abs(dx) > abs(dy) then steps := abs(dx)}
\]
\[
\text{else steps := abs(dy);}
\]
\[
x\text{.increment} := dx / steps;
\]
\[
y\text{.increment} := dy / steps;
\]
\[
x := x1 + 0.5; y := y1 + 0.5;
\]
\[
\text{set\_pixel (trunc(x), trunc(y));}
\]
\[
\text{for } k := 1 \text{ to steps do begin}
\]
\[
x := x + x\text{.increment;}
\]
\[
y := y + y\text{.increment;}
\]
\[
\text{set\_pixel (trunc(x), trunc(y));}
\]
\[
\text{end \{for k\}}
\]
end; \{dda\}

Symmetric DDA

pick a line length estimate which is some power of 2 and greater than or equal to \text{max (abs(dx), abs(dy))}
\[ y_i + 2 \]
\[ y = mx + b \]
\[ y_i + 1 \]
\[ y_i \]
\[ x_i \]
\[ x_i + 1 \]
\[ x_i + 2 \]
1. Input line end points. Store left endpoint in \((x_1, y_1)\).
   Store right endpoint in \((x_2, y_2)\).

2. The first point to be selected for display is the left endpoint \((x_1, y_1)\).

3. Calculate \(\Delta x = x_2 - x_1\), \(\Delta y = y_2 - y_1\), and \(p_1 = 2\Delta y - \Delta x\).
   If \(p_1 < 0\), the next point to be set is \((x_1 + 1, y_1)\). Otherwise, the next point is \((x_1 + 1, y_1 + 1)\).

4. Continue to increment the \(x\) coordinate by unit steps. At position \(x_{i+1}\) the coordinate to be selected, \(y_{i+1}\), is either \(y_i\) or \(y_i + 1\),
   depending on whether \(p_i < 0\) or \(p_i \geq 0\). The calculations for each parameter \(p\) depend on the last one. If \(p_i < 0\), the form for the next parameter is
   \[p_{i+1} = p_i + 2\Delta y\]
   But if \(p_i \geq 0\), the next parameter is
   \[p_{i+1} = p_i + 2(\Delta y - \Delta x)\]
   Then, if \(p_{i+1} < 0\), the next \(y\) coordinate to be selected is \(y_{i+1}\). Otherwise
   select \(y_{i+1} + 1\). (Coordinate \(y_{i+1}\) was determined to be either \(y_i\) or \(y_{i+1}\)
   by the parameter \(p_i\) in step 3.)

5. Repeat the procedures in step 4 until the \(x\) coordinate reaches \(x_2\).
procedure bres_line (x1, y1, x2, y2: integer);
  var
dx, dy, x, y, x_end, p, const1, const2 : integer;
begin
  dx := abs(x1 - x2);
dy := abs(y1 - y2);
p := 2 * dy - dx;
const1 := 2 * dy;
const2 := 2 * (dy - dx);
{determine which point to use as start, which as end}
if x1 > x2 then begin
  x := x2; y := y2;
x_end := x1
end {if x1 > x2}
else begin
  x := x1; y := y1
  x_end := x2
end; {if x1 <= x2}
set_pixel (x, y);
while x < x_end do begin
  x := x + 1;
  if p < 0 then p := p + const1
  else begin
    y := y + 1;
p := p + const2
  end; {else begin}
set_pixel (x, y);
end {while x < x_end}
end; {bres_line}
extensions for slopes > 1

- interchange the roles of x and y

extensions for negative slopes

- decrement rather than increment
loading the frame buffer

- use the set_pixel procedure
- \(\text{ADDR}(x,y) = \text{ADDR}(0,0) + y(x_{\text{max}}+1) + x\)

- simplification
  - \(\text{ADDR}(x + 1, y) = \text{ADDR} (x,y) + 1\)
  - \(\text{ADDR} (x + 1, y + 1) = \text{ADDR}(x,y) + x_{\text{max}} + 2\)
Antialiasing Lines

- aliasing: rounding to discrete integer pixel positions
- antialiasing: adjusting pixel intensities to remove the stairstep appearance
use of sampling theory

- pixels have measurable diameters
- lines have measurable thickness
- make intensity proportional to overlap
pixel phasing

- adjust pixel positions
  - 1/4 of a pixel diameter
  - 1/2 of a pixel diameter

- alter the sizes of individual pixels
Line Command

- plot
  - line segments
  - points ( = very short line segments)

- polyline (n,x,y)
  - a single point, a single line segment or a series of connected line segments
  - n = number of vertices
    x = array of n coordinate values
    y = array of n coordinate values
Line Command, continued

- plotting a single point
  \[ x[1] := 150; \]
  \[ y[1] := 100; \]
  \[ \text{polyline } (1,x,y); \]

- plotting a single line segment
  \[ x[1] := 50; \]
  \[ y[1] := 100; \]
  \[ x[2] := 250; \]
  \[ y[2] := 25; \]
  \[ \text{polyline } (2,x,y); \]

- absolute coordinates

- relative coordinates

- current position
Fill Areas

- `fill_area (n,x,y)`
- A closed polygon defined by
  - `(x[1],y[1])` to `(x[2],y[2])`
  - `(x[2],y[2])` to `(x[3],y[3])`
    ...
  - `(x[n],y[n])` to `(x[1],y[1])`

- `fill with`
  - border only
  - background
  - color
  - pattern
Circle-generating Algorithms

- DDA algorithm
- Bresenham algorithm
points on the circumference of a circle

- $(x - xc)^2 + (y - yc)^2 = r^2$

- $y = yc + \left[ r^2 - (x - xc)^2 \right]^{1/2}$
  - considerable computation
  - nonuniform spacing
alternatives

- parametric polar form
  - $x = xc + r\cos(\Theta)$
  - $y = yc + r\sin(\Theta)$

- DDA
  - $dy/dx = -x/y$
  - $x' = x + (y)(\text{epsilon})$
  - $y' = y - (x')(\text{epsilon})$
improvement

- calculate points from $x = 0$ to $x = y$
- take advantage of symmetry
Bresenham’s circle algorithm

- determine which of two pixels is nearer the true boundary of the circle

\[ x_i, x_i + 1, x_i + 2 \]

\[ y_i \]

\[ y_i - 1 \]

\[ y_i - 2 \]

\[ x^2 + y^2 = r^2 \]

- more efficient than other algorithms
  - adds
  - subtracts
  - shifts
  - compares
1. Select the first position for display as 
   \[ (x_1, y_1) = (0, r) \]

2. Calculate the first parameter as 
   \[ p_1 = 3 - 2r \]
   If \( p_1 < 0 \), the next position is \((x_1 + 1, y_1)\). Otherwise, the next position is 
   \((x_1 + 1, y_1 - 1)\).

3. Continue to increment the \( x \) coordinate by unit steps, and calculate each 
   succeeding parameter \( p \) from the preceding one. If for the previous parameter 
   we found that \( p < 0 \), then 
   \[ p_{i+1} = p_i + 4x_i + 6 \]
   Otherwise (for \( p \geq 0 \)), 
   \[ p_{i+1} = p_i + 4(x_i - y_i) + 10 \]
   Then, if \( p_{i+1} < 0 \), the next point selected is \((x_i + 2, y_{i+1})\). Otherwise, the 
   next point is \((x_i + 2, y_i + 1 - 1)\). The \( y \) coordinate is 
   \[ y_{i+1} = y_i \text{, if } p < 0 \text{ or } y_{i+1} = y_i - 1 \text{, if } p \geq 0. \]

4. Repeat the procedures in step 3 until the \( x \) and \( y \) coordinates are equal.
procedure bres_circle(x_center, y_center, radius : integer);
var
  p, x, y : integer

procedure plot_circle_points:
begin
  set_pixel (x_center + x, y_center + y);
  set_pixel (x_center - x, y_center + y);
  set_pixel (x_center + x, y_center - y);
  set_pixel (x_center - x, y_center - y);
  set_pixel (x_center + y, y_center + x);
  set_pixel (x_center - y, y_center + x);
  set_pixel (x_center + y, y_center - x);
  set_pixel (x_center - y, y_center - x);
end; {plot_circle_points}

begin {bres-circle}
x := 0;
y := radius;
p := 3 - 2 * radius;
while x < y do begin
  plot_circle_points;
  if p<0 then p := p + 4 * x + 6
  else begin
    p := p + 4 * (x - y) + 10
    y := y - 1
  end; {if p not < 0}
x := x + 1
end; {while x < y}
if x = y then plot_circle_points;
end; {bres_circle}
plotting ellipses and circles

- ellipse \((x_c, y_c, r_1, r_2)\)
- for circles, \(r_1 = r_2\)

plotting elliptical arcs and circular arcs

- ellipse \((x_c, y_c, r_1, r_2, \Theta_1, \Theta_2)\)
  - \(\Theta_1 = \) angle between x-axis and start of arc
  - \(\Theta_2 = \) angle between x-axis and end of arc
Other Curves

- procedures are similar to those for circles and ellipses
- common curves
  - sine functions
  - exponential functions
  - polynomials
  - probability distributions
  - spline functions
- symmetry considerations can improve efficiency
- points along curves can be connected using short line segments or using other curve-fitting techniques
Character Generation

- rectangular grid patterns
  - 5-by-7 to 9-by-14
  - stored in read-only memory
  - copied into the frame buffer
Output Commands

- text (x, y, string)
  - example: text (100, 450, "Winter in Rochester")

- polymarker (n, x, y)
  - example: polymarker (6, x, y)
Instruction Sets for Display Processors

- dependent on the type of device in use

- raster-scan systems
  - registers for coordinate values
    - endpoints
    - circle and ellipse parameters
    - positions for character strings and markers
  - fields
    - opcode: type of operation
    - address: register or memory location
  - instructions for loading the frame buffer
  - intensities are read from the frame buffer

- random-scan systems
  - instructions are used by the refresh process
  - load and execute the first instruction
  - increment the instruction counter
  - load and execute the next instruction
    -
    -
  - reset the instruction counter and repeat
OUTPUT PRIMITIVES

- Points and Lines
- Line-drawing Algorithms
- Antialiasing Lines
- Line Command
- Fill Areas
- Circle-generating Algorithms
- Other Curves
- Character Generation
- Instruction Sets for Display Processors