Higher Technological Institute
Computer Science Department

Computer Graphics

Dr Osama Farouk
Dr Ayman Soliman
Dr Adel Khaled

Lecture Five
Graphics Output Primitives Revision

Lecture One

Course Outlines

- Introduction to Computer Graphics .
- Overview of Graphics systems .
- Methods for producing basic picture components such as lines, circles, and polygons.
- Algorithms for performing geometric transformations such as rotation and scaling.
- Procedures for displaying views of two dimensional and three-dimensional scenes.

Textbook

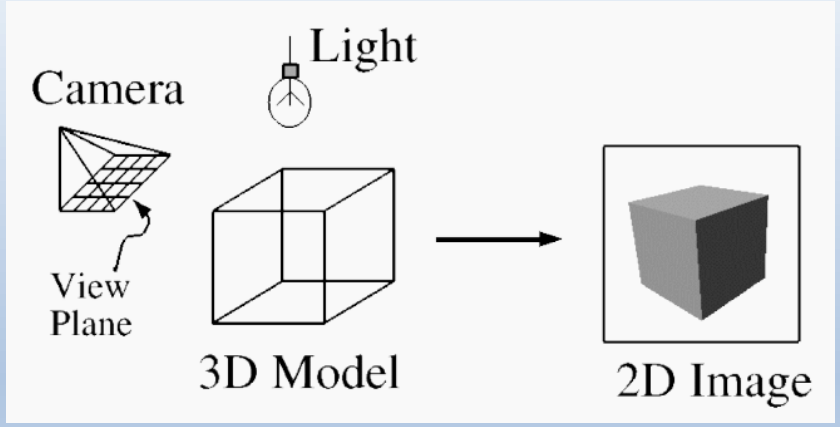
“Computer Graphics with OpenGL” , Third edition , Hearn - Baker

Introduction to Computer Graphics

What is computer graphics?

"Computer graphics is concerned with producing images and animations"

- Imaging = *representing 2D images*
- Modeling = *representing 3D objects*
- Rendering = *constructing 2D images from 3D models*
- Animation = *simulating changes over time*

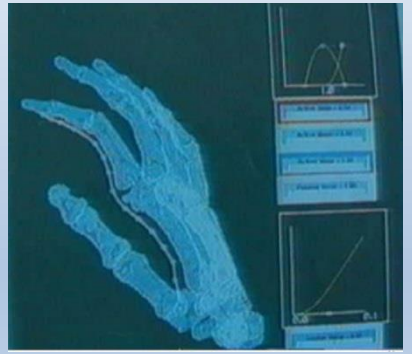


Applications

- Entertainment
- Computer-aided Design
- Scientific Visualization
- Training
- Education
- e-Commerce
- Computer Art
- Image Processing

Image Processing

- Image processing is the modification of interpretation of existing pictures.
- Some IP applications:
  - improving image quality,
  - analyzing satellite photos of the earth and telescopic recordings of galactic star distributions.
  - Medical applications: picture enhancements in tomography, simulations of surgical operations, Ultrasonic and nuclear medical scanners.



A Survey of Computer Graphics

Graphical User Interfaces (GUI)

The major components of a graphical interface are a window manager, menus, and icons.

GUI: is a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicator such as primary notation, instead of text-based user interfaces, typed command labels or text navigation

Overview of Graphics systems

We explore the basic features:

- Graphics hardware components.
  - Video display devices.
  - Input devices.
- Graphics software package.

Video Display Devices
1- Cathode Ray Tube (CRT)

- **Cathode Ray Tube (CRT)** A beam of electrons (cathode rays), emitted by an electron gun, passes through focusing and deflection systems that direct the beam toward specified positions on the phosphor-coated screen.

- The phosphor then emits a small spot of light at each position contacted by the electron beam.

- The most common method now employed for maintaining phosphor glow is to redraw the picture repeatedly by quickly directing the electron beam back over the same screen points.

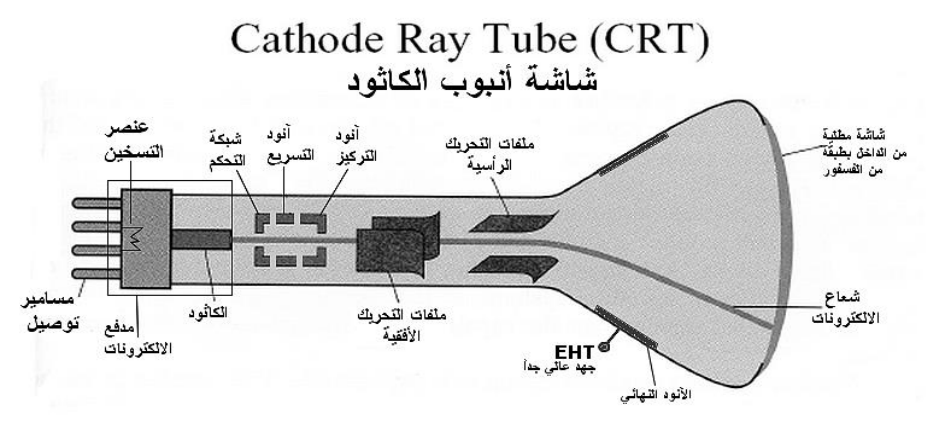
تعتمد أنبوبة الكاثود في عملها على الظاهرة الفيزيائية وهي أن مادة الفسفور تشع ضوء إذا تم قصفها بسيل من الالكترونات والتي تمتلك السرعة والجهد الكافيين لتحفيز الكترونات مادة الفسفور وإعطائها الطاقة اللازمة لكي تنطلق وتتحرر من حزمة التكافؤ إلى حزمة الطاقة الأعلى، وأثناء عملية الانطلاق إلى حزمة أعلى فإنها تطلق مجموعة من الفوتونات والتي تمثل الضوء المنبعث.

Components:

1- Electron Gun

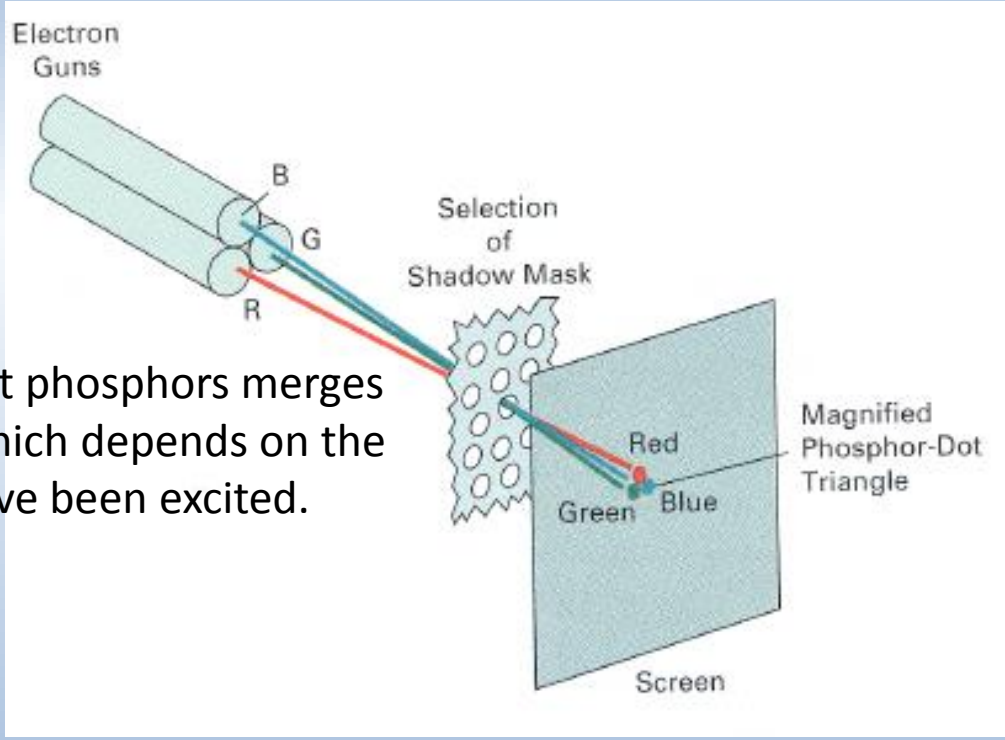
- Heating Element
- Cathode
- Control Grid
- Acceleration Anode
- Focusing Grid

2- Deflection Coils



2- Color CRT MONITOR

- A CRT monitor displays color pictures by using a combination of phosphors that emit different-colored light.



- The emitted light from the different phosphors merges to form a single perceived color, which depends on the particular set of phosphors that have been excited.

https://www.youtube.com/watch?v=cwkuCgYI91w

Flat-Panel Displays

- The term flat-panel display refers to a class of video devices that have reduced volume, weight, and power requirements compared to a CRT.
- Some additional uses for flat-panel displays are as small TV monitors, calculator screens, pocket video-game screens, laptop computer screens, armrest movie-viewing stations on airlines, advertisement boards in elevator.
- We can separate flat-panel displays into two categories: emissive displays and non-emissive displays.

Flat-Panel Displays

- The emissive displays (or emitters) are devices that convert electrical energy into light. Plasma panels, thin-film electroluminescent displays, and light-emitting diodes are examples of emissive displays.

- Non-emissive displays (or non-emitters) use optical effects to convert sunlight or light from some other source into graphics patterns. The most important example of a non-emissive flat-panel display is a liquid-crystal device LCD screen.

Input Devices

- Keyboard
- Mouse
- Joysticks
- Image scanners
- Touch panels
- Light pens
- Voice System

Graphics software

- ***Special purpose package***

  Examples of such applications include artist's painting programs

- ***General programming packages***

  Provides a library of graphics functions that can be used in a programming language such as C, C++ , open GL.

Graphics Functions

- Graphics output Primitives.
- Attributes Graphics Primitives.
- Geometric transformations.
- Viewing transformations.
- Many other operation.

OpenGL

- OpenGL is a software interface to graphics hardware.
- OpenGL is designed as hardware-independent interface to be implemented on many different hardware platforms.
- OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules.
- With OpenGL, you must build up your desired model from a small set of geometric primitive -points, lines, and polygons.

http://www.openGL.org.
https://www.opengl.org/sdk/docs/man2/

OpenGl programming Guide Book:
http://www.glprogramming.com/red/

Download OpenGL

How to get OpenGL working in Visual Studios 2013

https://www.youtube.com/watch?v=3IjxgvZq9a0

OpenGl, GLU and GLUT

- OpenGL: basic functions.
- GLU: OpenGl Utility library.
- GLUT: OpenGL Utility toolkit library.

GLU and GLUT: Handy functions for viewing and geometry.

Basic OpenGL Syntax

Function names in the OpenGL basic library

glBegin, glClear, glCopyPixel

Certain functions : for instance , a parameter name

GL_2D, GL_RGB, GL_POLYGON

Data type:

GLbyte, GLint, GLfloat, Glboolean

Related libraries

The OpenGL Utility (GLU): all GLU function names start with glu

Header files

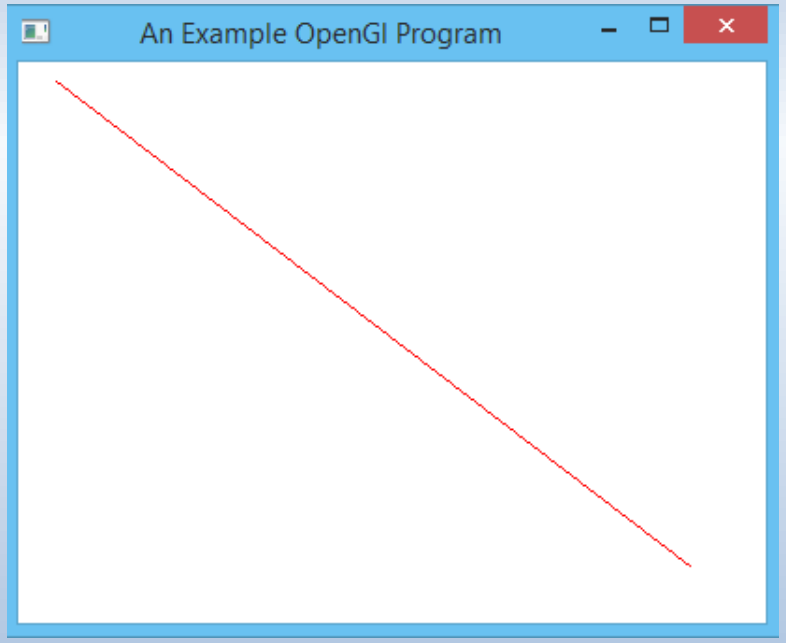
```
#include<windows.h>
#include<GL/gl.h>
#include<GL/glu.h>
```

If we used OpenGL Utility Toolkit(GLUT) we can replace the header files with

```
#include <GL/GLUT.h>
```

Example

The display windows and line segment such as following Figure :



```
#include <Windows.h>
#include <GL\glew.h>
#include <GL\freeglut.h>
#include <iostream>
#include <GL/glut.h> // (or others, depending on the system in use)
void init (void){
glClearColor (1.0, 1.0, 1.0, 0.0); // Set display-window color to white.
glMatrixMode (GL_PROJECTION); // Set projection parameters.
gluOrtho2D (0.0, 200.0, 0.0, 150.0);}
void lineSegment (void){
glClear (GL_COLOR_BUFFER_BIT); // Clear display window.
glColor3f (1.0, 0.0, 0.0); // Set line segment color to red.
glBegin (GL_LINES);
glVertex2i (180, 15); // Specify line-segment geometry.
glVertex2i (10, 145);
glEnd ( );
glFlush ( ); // Process all OpenGL routines as quickly as possible.}
void main (int argc, char** argv){
glutInit (&argc, argv); // Initialize GLUT.
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // Set display mode.
glutInitWindowPosition (50, 100); // Set top-left display-window position.
glutInitWindowSize (400, 300); // Set display-window width and height.
glutCreateWindow ("An Example OpenGL Program"); // Create display window.
init ( ); // Execute initialization procedure.
glutDisplayFunc (lineSegment); // Send graphics to display window.
glutMainLoop ( ); // Display everything and wait.
}
```

Lecture two
Graphics Output Primitives

OpenGL Point Functions

- The default color for primitives is white and the default point size is equal to the size of one screen pixel.
- The form for an OpenGL specification of a point position is

```
glBegin (GL_POINTS);
glVertex* ( );              //The coordinate values for a single position
glEnd ( );
```

- In the following example, three equally spaced points are plotted along a two-dimensional straight-line path with a slope of 2 (Figure). Coordinates are given as integer pairs.

```
glBegin (GL_POINTS);
    glVertex2i (50, 100);
    glVertex2i (75, 150);
    glVertex2i (100, 200);
glEnd ( );
```

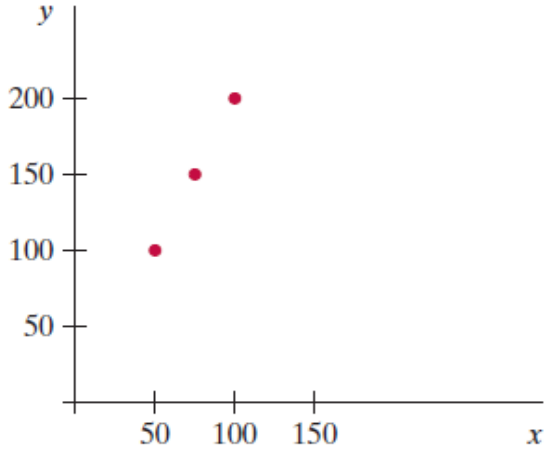


FIGURE Display of three point positions generated with `glBegin (GL_POINTS)`.

Alternatively, we could specify the coordinate values for the preceding points in arrays such as

```
int point1 [ ] = {50, 100};
int point2 [ ] = {75, 150};
int point3 [ ] = {100, 200};
```

and call the OpenGL functions for plotting the three points as

```
glBegin (GL_POINTS);
glVertex2iv (point1);
glVertex2iv (point2);
glVertex2iv (point3);
glEnd ( );
```

And here is an example of specifying two point positions in a three dimensional world reference frame. In this case, we give the coordinates as explicit floating-point values.

```
glBegin (GL_POINTS);
glVertex3f (-78.05, 909.72, 14.60);
glVertex3f (261.91, -5200.67, 188.33);
glEnd ( );
```

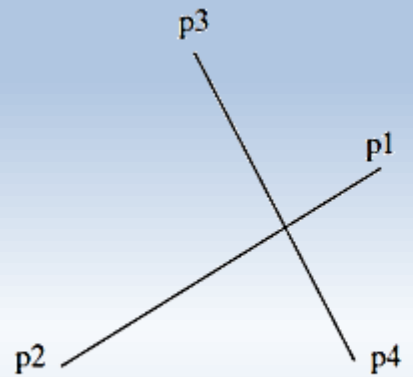
Using this class definition, we could specify two- dimensional, world-coordinate point position with the statements

```
wcPt2D pointPos;
pointPos.x = 120.75;
pointPos.y = 45.30;
glBegin (GL_POINTS);
glVertex2f (pointPos.x, pointPos.y);
glEnd ( );
```

Look to pages in textbook "Computer Graphics with open GL"
Pages(88-89)

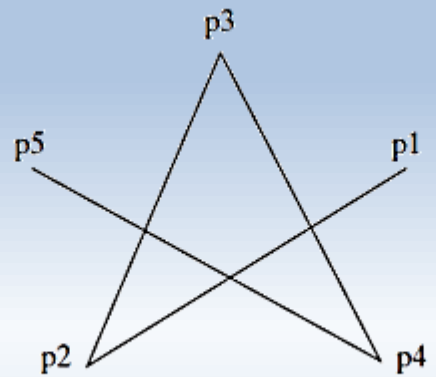
OpenGL LINE FUNCTIONS

The following code could generate the display shown in Figure.



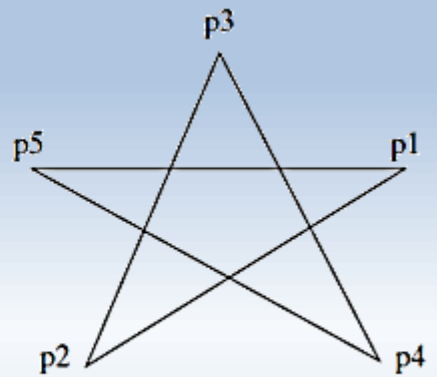
(a)

```
glBegin (GL_LINES);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd ( );
```



(b)

```
glBegin (GL_LINES_STRIP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd ( );
```



(c)

```
glBegin (GL_LINES_LOOP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd ( );
```

LINE-DRAWING ALGORITHMS

- A straight-line segment in a scene is defined by the coordinate positions for the endpoints of the segment.
- The line color is loaded into the frame buffer at the corresponding pixel coordinates.
- Stair-step effect (jaggies) produced when a line is generated as a series of pixel positions.

Line Equations

$$y = m \cdot x + b \qquad (1)$$

$$m = \frac{y_{end} - y_0}{x_{end} - x_0} \qquad (2)$$

$$b = y_0 - m \cdot x_0 \qquad (3)$$

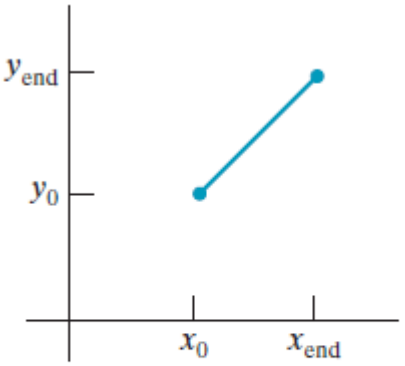


FIGURE Line path between endpoint positions (x_0, y_0) and (x_{end}, y_{end}).

For any given x interval δx along a line, we can compute the corresponding δy interval *By* from Eq. 2 as

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

$$\delta y = m \cdot \delta x \qquad (4)$$

$$\delta x = \frac{\delta y}{m} \qquad (5)$$

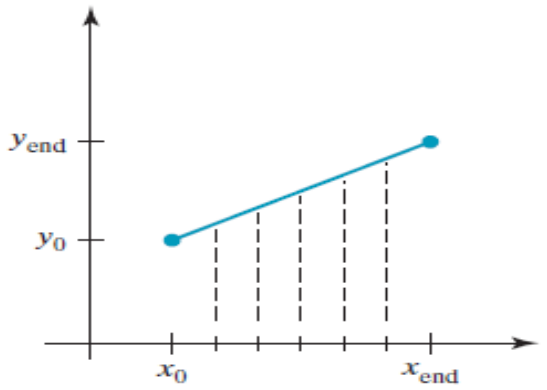


FIGURE Straight-line segment with five sampling positions along the x axis between x_0 and x_{end}.

The *Digital Differential Analyzer* (DDA)

We consider first a line with positive slope,

If the slope is less than or equal to 1, we sample at unit *x* intervals (δx = 1) and compute successive *y* values as

$$y_{k+1} = y_k + m \qquad (6)$$

Subscript *k* takes integer values starting from 0, for the first point, and increases by 1 until the final endpoint is reached. Since *m* can be any real number between 0.0 and 1.0, each calculated *y* value must be rounded to the nearest integer corresponding to a screen pixel position in the *x* column we are processing.

For lines with a positive slope greater than 1.0,we reverse the roles of *x* and *y*. That is, we sample at unit *y* intervals (δy = 1) and calculate consecutive *x* values as

$$x_{k+1} = x_k + \frac{1}{m} \qquad (7)$$

In this case, each computed *x* value is rounded to the nearest pixel position along the current *y* scan line.

Equations 6 and 7 are based on the assumption that lines are to be processed from the left endpoint to the right endpoint

If this processing is reversed, so that the starting endpoint is at the right, then either we have δx =−1 and

$$y_{k+1} = y_k - m$$

or (when the slope is greater than 1) we have δy = −1 with

$$x_{k+1} = x_k - \frac{1}{m}$$

This algorithm is summarized in the following procedure, which accepts as input two integer screen positions for the endpoints of a line segment . Horizontal and vertical differences between the endpoint positions are assigned to parameters **dx** and **dy**. The difference with the greater magnitude determines the value of parameter **steps**. Starting with pixel position (**x0**, **y0**) ,we determine the offset needed at each step to generate the next pixel position along the line path. We loop through this process **steps** times. If the magnitude of **dx** is greater than the magnitude of **dy** and **x0** is less than **xEnd**, the values for the increments in the *x* and *y* directions are 1 and *m*, respectively. If the greater change is in the *x* direction, but **x0** is greater than **xEnd**, then the decrements −1 and −*m* are used to generate each new point on the line. Otherwise, we use a unit increment (or decrement) in the *y* direction and an *x* increment (or decrement) $\frac{1}{m}$.

The *digital differential analyzer* (DDA) Algorithm: (textbook p94)

The *digital differential analyzer* (DDA) is a scan-conversion line algorithm based on calculating either δx or δy , using Eq. 4 or Eq. 5.

```
#include <stdlib.h>
#include <math.h>

inline int round (const float a)  { return int (a + 0.5); }

void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
   int dx = xEnd - x0,  dy = yEnd - y0,  steps,  k;
   float xIncrement, yIncrement, x = x0, y = y0;

   if (fabs (dx) > fabs (dy))
      steps = fabs (dx);
   else
      steps = fabs (dy);
   xIncrement = float (dx) / float (steps);
   yIncrement = float (dy) / float (steps);

   setPixel (round (x), round (y));
   for (k = 0; k < steps; k++) {
      x += xIncrement;
      y += yIncrement;
      setPixel (round (x), round (y));
   }
}
```

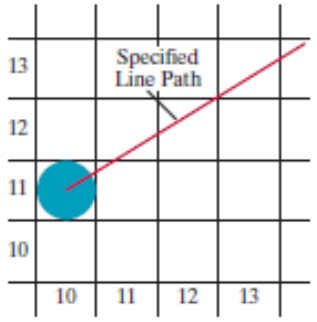


FIGURE 3-8 A section of a display screen where a straight-line segment is to be plotted, starting from the pixel at column 10 on scan line 11.

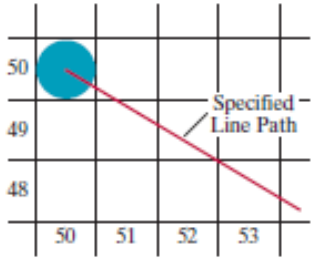


FIGURE 3-9 A section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50.

EXAMPLE 1:

Apply The *digital differential analyzer* (DDA) Algorithm to compute which pixels should be turned on to represent the line from (2,2) to (8,7)

DDA- Digital Differential Analyser

This case is for slope (m) less than 1. Slope (m) =(7-1)/(8-1) = 6/7 .

S-1: x1=1; y1=1; x2=8; y2=7.

S-2: m=(7-1)/(8-1) = 6/7 which is less than 1.

S-3: As m (6/7) is less than 1 therefore x is increased and y is calculated.

S-4 : The step will be x1=x1+1 and y1= y1+6/7

S-5 : The points genrated would be x1=1+1 and Y1=1+(5/7) => 1+0.9=>1.9=> approx 2. So X 1=2 and Y1= ~

.	X1	Y1	Pixel Plotted
p1	2	2	2,2
p2	3	2+6/7 = 2.9	3,3
p3	4	2.9 + 6/7 = 3.8	4,4
p4	5	3.8 + 6/7 = 4.7	5,5
p5	6	4.7 + 6/7 = 5.6	6,6
p6	7	5.6 + 6/7 = 7.0	7,7

The algorithm will stop here as the x value has reached 7.

EXAMPLE 2:

Apply The *digital differential analyzer* (DDA) Algorithm to compute which pixels should be turned on to represent the line from (0,0) to (4,6)

This case is for slope (m) greater than 1. Slope (m) =(6-0)/(4-0) = 6/4 .

S-1: x1=0; y1=0; x2=4; y2=6

S-2: m=(6-0)/(4-0) = 6/4 which is more than 1.

S-3: As m (6/4) is greater than 1 therefore y is increased and x is calculated.

S-4 : Now increase the value of y and calculate value of x.

- To calculate x, take line equation and find x , x2=x1+1/m
- The step will be y1=y1+1 and x1 =x1+1/(6/4) , After Simplification, Every time y1=y1+1 and x1=x1+4/6

	Y1	X1	Pixel Plotted
p0	0	0	(0,0)
p1	1	x1= (0)+4/6=0.67 =1	(1,1)
p2	2	0.67+4/6 = 1.34	(1,2)
p3	3	1.34+4/6=2.01	(2,3)
p4	4	2.01+4/6= 2.68	(3,4)
p5	5	2.68+4/6=3.35	(3,5)
p6	6	3.35+4/6=4.02	(4,6)

EXAMPLE 3:

Apply The *digital differential analyzer* (DDA) Algorithm to compute which pixels should be turned on to represent the line from (2,3) to (9,8)

S-1: x1=2, y1=3 and x2=9 , y2=8.

S-2: Calculate Slope m = (8-3)/(9-2) = 5/7, which is less than 1.

S-3: Since m is less than one that means we would increase x and calculate y.

S-4: So new x would be equal to old x plus 1 and calculate y as new y = old y +m(slope). — Easy to understand, We mean the following

x1=x1+1 and y1=y1+(5/7)

	X1	Y1	Pixel Plotted
p0	2	3	2,3
p1	3	3+5/7 => 26/7 => 26/7=>3.71	3,4
p2	4	3.71 + 5/7 = 4.42	4,4
p3	5	4.42 + 5/7 =5 .13	5,5
p4	6	5.13 + 5/7 =5.84	6,6
p5	7	5.84 + 5/7 = 6.55	7,7
p6	8	6.55+5/7=7.26	8,7
p7	9	7.26+5/7=7.97	9,8

The algorithm would stop here as we have reached the end point of the line (9,8)

Exercise :

Apply The *digital differential analyzer* (DDA) Algorithm to compute which pixels should be turned on to represent the line from (20,10) to (30,18)

dx	dy	steps	k	X Increment	Y Increment	x	y	(x,y)
						20	10	(20,10)
10	8	10	0	1	0.8	21	10.8	(21,11)
			1			22	11.6	(22,12)
			2			23	12.3	(23,12)
			3			24	13.1	(24,13)
			4			25	13.9	(25,14)
			5			26	14.7	(26,15)
			6			27	15.5	(27,15)
			7			28	16.3	(28,16)
			8			29	17.1	(29,17)
			9			30	17.9	(30,18)

Bresenham's Line Algorithm

Bresenham's Line-Drawing Algorithm for $|m| < 1.0$

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).
2. Set the color for frame-buffer position (x_0, y_0); i.e., plot the first point.
3. Calculate the constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

$$p_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k = 0$, perform the following test. If $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

$$p_{k+1} = p_k + 2\Delta y$$

   Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Perform step 4 $\Delta x - 1$ times.

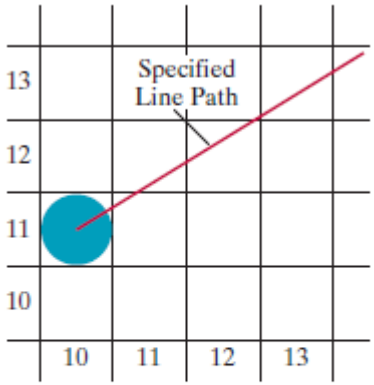


FIGURE A section of a display screen where a straight-line segment is to be plotted, starting from the pixel at column 10 on scan line 11.

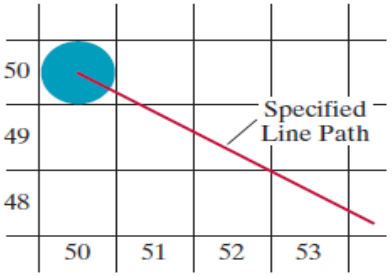


FIGURE A section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50.

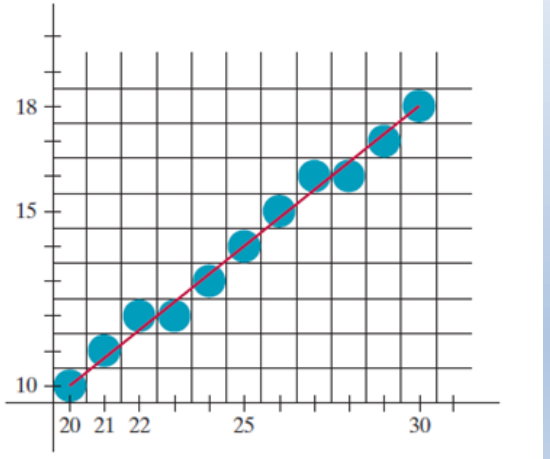
Bresenham's Line Algorithm

EXAMPLE :(textbook p136-137)

To illustrate the algorithm, we digitize the line with endpoints (20, 10) and (30, 18).

- Draw the line with endpoints (20,10) and (30, 18).
  - **$\Delta x=30-20=10$, $\Delta y=18-10=8$,**
  - **$p_0 = 2\Delta y - \Delta x=16-10=6$**
  - **$2\Delta y=16$,** and **$2\Delta y - 2\Delta x=-4$**
- Plot the initial position at (20,10), then

k	p_k	(x_{k+1}, y_{k+1})	k	p_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)	5	6	(26, 15)
1	2	(22, 12)	6	2	(27, 16)
2	−2	(23, 12)	7	−2	(28, 16)
3	14	(24, 13)	8	14	(29, 17)
4	10	(25, 14)	9	10	(30, 18)



Bresenham's Line Algorithm

An implementation of Bresenham line drawing for slopes in the range 0 < *m* < 1.0 is given in the following procedure

```
#include <stdlib.h>
#include <math.h>

/*  Bresenham line-drawing procedure for |m| < 1.0.  */
void lineBres (int x0, int y0, int xEnd, int yEnd)
{
   int dx = fabs (xEnd - x0),  dy = fabs(yEnd - y0);
   int p = 2 * dy - dx;
   int twoDy = 2 * dy,  twoDyMinusDx = 2 * (dy - dx);
   int x, y;

   /* Determine which endpoint to use as start position.  */
   if (x0 > xEnd) {
      x = xEnd;
      y = yEnd;
      xEnd = x0;
   }
   else {
      x = x0;
      y = y0;
   }
   setPixel (x, y);

   while (x < xEnd) {
      x++;
      if (p < 0)
         p += twoDy;
      else {
         y++;
         p += twoDyMinusDx;
      }
      setPixel (x, y);
   }
}
```

Lecture Three
Graphics Output Primitives

Circle drawing algorithms

Properties of Circles

A circle (Figure) is defined as the set of points that are all at a given distance *r* from a center position (x_c, y_c) .

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \quad (1)$$

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

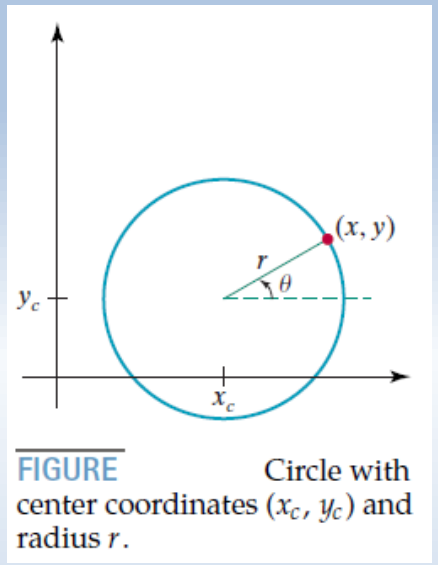


FIGURE Circle with center coordinates (x_c, y_c) and radius r.

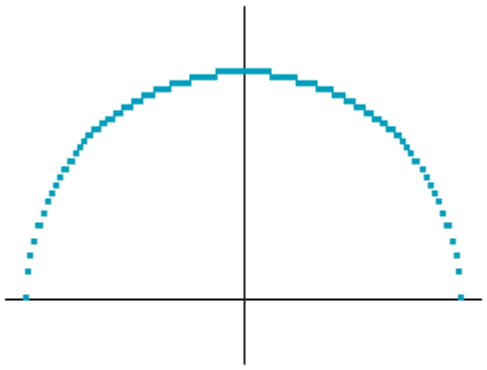


FIGURE Upper half of a circle plotted with Eq. 2 and with $(x_c, y_c) = (0, 0)$.

But this is not the best method for generating a circle.
One problem with this approach is that it involves considerable computation at each step. Moreover, the spacing between plotted pixel positions is not uniform, as demonstrated in Figure

Expressing the circle equation in parametric polar form yields the pair of equations

$$x = x_c + r\cos\theta$$
$$y = y_c + r\sin\theta \qquad (3)$$

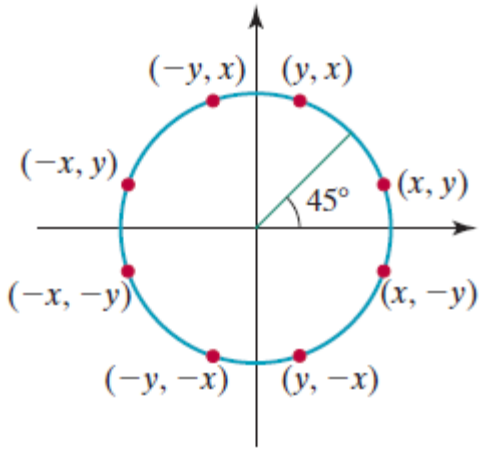


FIGURE Symmetry of a circle. Calculation of a circle point (x, y) in one octant yields the circle points shown for the other seven octants.

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 shape of the circle is similar in each quadrant. Therefore, if we determine the curve positions in the first quadrant, we can generate the circle section in the second quadrant of the *xy* plane by noting that the two circle sections are symmetric with respect to the y axis.

And circle sections in the third and fourth quadrants can be obtained from sections in the first and second quadrants by considering symmetry about the *x* axis.

Midpoint Circle Algorithm

The basic idea in this approach is to test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary.

To apply the midpoint method, we define a circle function as

$$f_{\text{circ}}(x, y) = x^2 + y^2 - r^2 \qquad (4)$$

$$f_{\text{circ}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x, y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases} \qquad (5)$$

The tests in (5) are performed for the mid positions between pixels near the circle path at each sampling step.

Assuming that we have just plotted the pixel at (x_k, y_k), we next need to determine whether the pixel at position $(x_k + 1, y_k)$ or the one at position $(x_k + 1, y_k - 1)$ is closer to the circle. Our decision parameter is the circle function (4) evaluated at the midpoint between these two pixels:

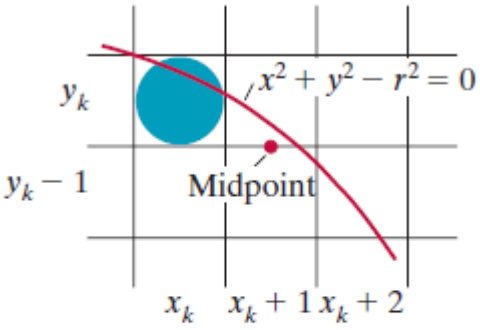


FIGURE Midpoint between candidate pixels at sampling position $x_k + 1$ along a circular path.

$$p_k = f_{\text{circ}}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \qquad (6)$$

If $p_k < 0$, this midpoint is inside the circle and the pixel on scan line y_k is closer to the circle boundary. Otherwise, the midposition is outside or on the circle boundary, and we select the pixel on scan line $y_k - 1$.

Successive decision parameters are obtained using incremental calculations.

We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$

$$p_{k+1} = f_{\text{circ}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$= [(x_k + 1) + 1]^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

or

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \qquad (7)$$

where y_{k+1} is either y_k or $y_k - 1$, depending on the sign of p_k .

Increments for obtaining p_{k+1} are either $2x_{k+1}+1$ (if p_k is negative) or $2x_{k+1}+1-2y_{k+1}$.
Evaluation of the terms $2x_{k+1}$ and $2y_{k+1}$ can also be done incrementally as

$$\begin{aligned} 2x_{k+1} &= 2x_k + 2 \\ 2y_{k+1} &= 2y_k - 2 \end{aligned} \qquad (8)$$

The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$ *from eq.(6)*:

$$\begin{aligned} p_0 &= f_{\text{circ}}\left(1, r - \frac{1}{2}\right) \\ &= 1 + \left(r - \frac{1}{2}\right)^2 - r^2 \end{aligned}$$

$$p_0 = \frac{5}{4} - r \qquad (9)$$

If the radius r is specified as an integer, we can simply round p_0 to

$$p_0 = 1 - r \qquad \text{(for } r \text{ an integer)}$$

Midpoint Circle Algorithm

Midpoint Circle Algorithm

1. Input radius r and circle center (x_c, y_c), then set the coordinates for 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 at (x_c, y_c) and plot the coordinate values:

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

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

Example: Midpoint Circle Drawing

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

$$p_0 = 1 - r = -9$$

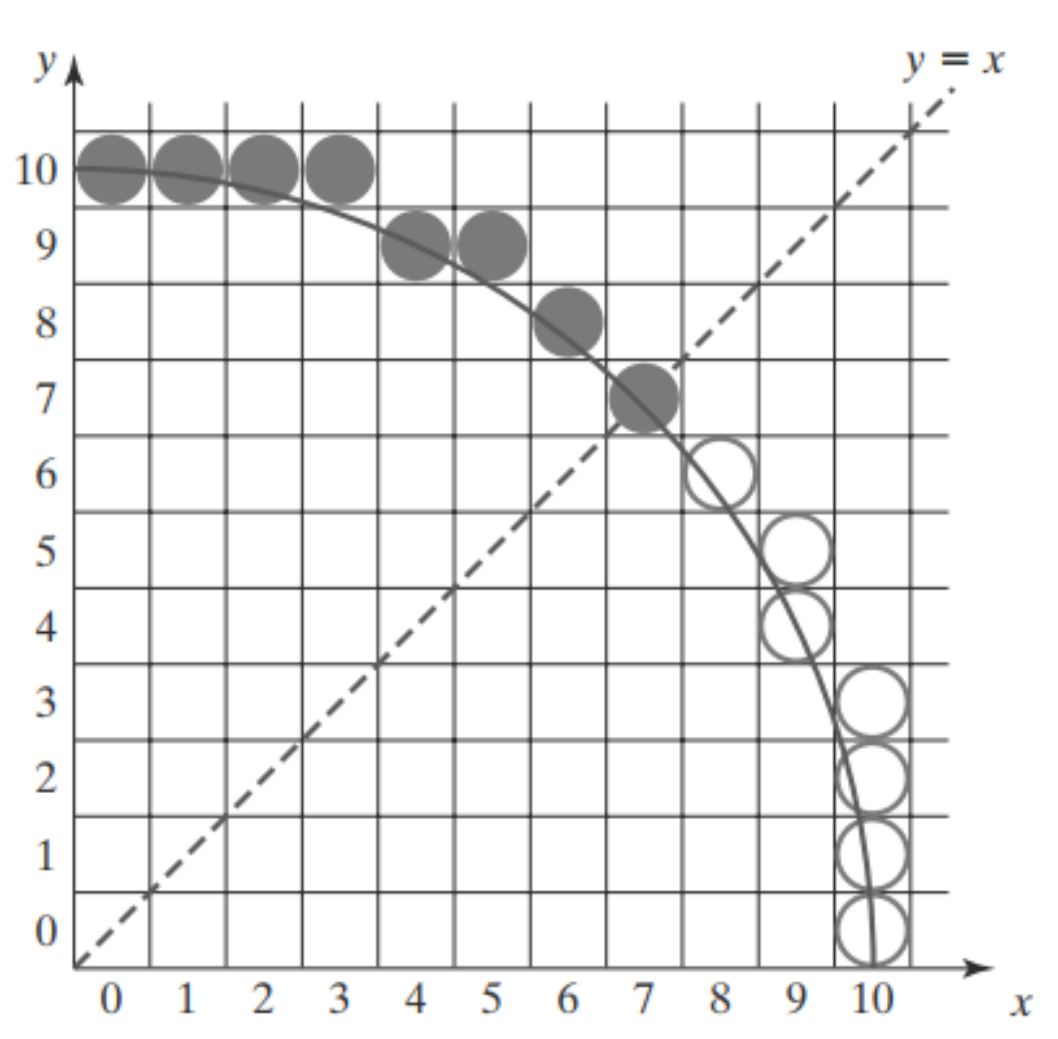
For the circle centered on the coordinate origin, the initial point is $(x_0, y_0) = (0, 10)$, and initial increment terms for calculating the decision parameters are

$$2x_0 = 0, \qquad 2y_0 = 20$$

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

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	−9	(1, 10)	2	20
1	−6	(2, 10)	4	20
2	−1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	−3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

A plot of the generated pixel positions in the first quadrant is shown



```
#include <GL/glut.h>

class screenPt
{
    private:
       GLint x, y;

    public:
       /*  Default Constructor: initializes coordinate position to (0, 0).  */
       screenPt ( )  {
           x = y = 0;
       }
       void setCoords (GLint xCoordValue, GLint yCoordValue)  {
           x = xCoordValue;
           y = yCoordValue;
       }

       GLint getx ( ) const  {
           return x;
       }

       GLint gety ( ) const  {
           return y;
       }
       void incrementx ( )  {
           x++;
       }
       void decrementy ( )  {
           y--;
       }
};
```

```
void setPixel (GLint xCoord, GLint yCoord)
{
    glBegin (GL_POINTS);
        glVertex2i (xCoord, yCoord);
    glEnd ( );
}

void circleMidpoint (GLint xc, GLint yc, GLint radius)
{
    screenPt circPt;

    GLint p = 1 - radius;          //  Initial value for midpoint parameter.

    circPt.setCoords (0, radius); //  Set coordinates for top point of circle.

    void circlePlotPoints (GLint, GLint, screenPt);
    /*  Plot the initial point in each circle quadrant.  */
    circlePlotPoints (xc, yc, circPt);
    /*  Calculate next point and plot in each octant.  */
```

```
    while (circPt.getx ( ) < circPt.gety ( )) {
        circPt.incrementx ( );
        if (p < 0)
            p += 2 * circPt.getx ( ) + 1;
        else {
            circPt.decrementy ( );
            p += 2 * (circPt.getx ( ) - circPt.gety ( )) + 1;
        }
        circlePlotPoints (xc, yc, circPt);
    }
}

void circlePlotPoints (GLint xc, GLint yc, screenPt circPt)
{
    setPixel (xc + circPt.getx ( ), yc + circPt.gety ( ));
    setPixel (xc - circPt.getx ( ), yc + circPt.gety ( ));
    setPixel (xc + circPt.getx ( ), yc - circPt.gety ( ));
    setPixel (xc - circPt.getx ( ), yc - circPt.gety ( ));
    setPixel (xc + circPt.gety ( ), yc + circPt.getx ( ));
    setPixel (xc - circPt.gety ( ), yc + circPt.getx ( ));
    setPixel (xc + circPt.gety ( ), yc - circPt.getx ( ));
    setPixel (xc - circPt.gety ( ), yc - circPt.getx ( ));
}
```

Lecture Four
Graphics Output Primitives

ELLIPSE-GENERATING ALGORITHMS

Properties of Ellipses

A precise definition of 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.

$$d_1 + d_2 = \text{constant} \qquad (1)$$

Expressing distances d_1 and d_2

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

The general ellipse equation in the form

$$A x^2 + B y^2 + C x y + D x + E y + F = 0$$

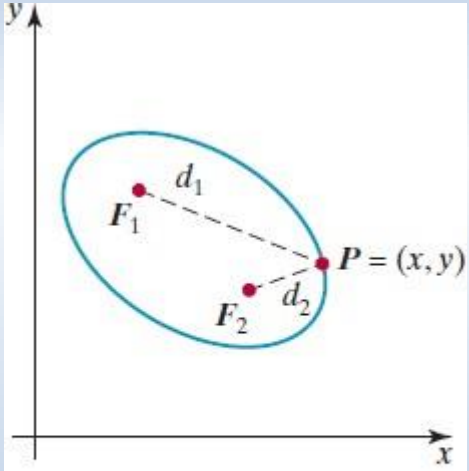


FIGURE Ellipse generated about foci F_1 and F_2.

The major axis is the straight-line segment extending from one side of the ellipse to the other through the foci. The minor axis spans the shorter dimension of the ellipse.

The equation for the ellipse

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1 \qquad (1)$$

Using polar coordinates r and θ

$$x = x_c + r_x \cos\theta$$
$$y = y_c + r_y \sin\theta \qquad (2)$$

If $r_x > r_y$, the radius of the bounding circle is $r = r_x$. Otherwise, the bounding circle has radius $r = r_y$.

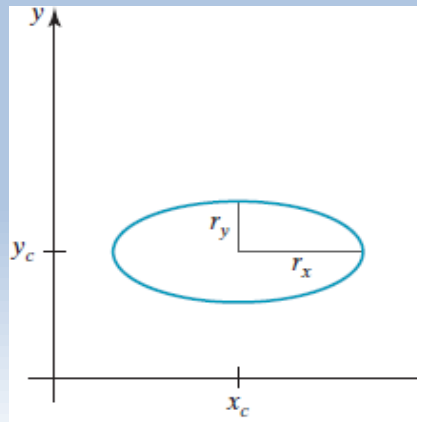


FIGURE Ellipse centered at (x_c, y_c) with semimajor axis r_x and semiminor axis r_y.

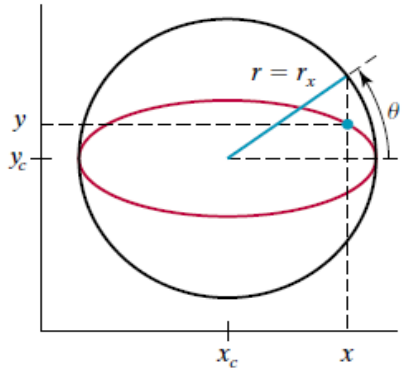
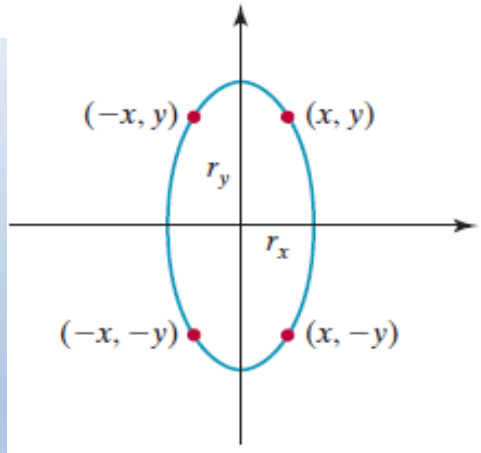


FIGURE The bounding circle and eccentric angle θ for an ellipse with $r_x > r_y$.



As with the circle algorithm, symmetry considerations can be used to reduce computations. An ellipse in standard position is symmetric between quadrants, but, unlike a circle, it is not symmetric between the two octants of a quadrant. Thus, we must calculate pixel positions along the elliptical arc throughout one quadrant, then use symmetry to obtain curve positions in the remaining three quadrants

Midpoint Ellipse Algorithm

Given parameters r_x , r_y, and (x_c , y_c), we determine curve positions (x, y) for an ellipse in standard position centered on the origin, then we shift all the points using a fixed offset so that the ellipse is centered at (x_c , y_c). If we wish also to display the ellipse in nonstandard position, we could rotate the ellipse about its center coordinates to reorient the major and minor axes in the desired directions.

The midpoint ellipse method is applied throughout the first quadrant in two parts.

Regions 1 and 2 (Figure) can be processed in various ways. We can start at position (0, r_y) and step clockwise along the elliptical path in the first quadrant, shifting from unit steps in x to unit steps in y when the slope becomes less than -1.0. Alternatively, we could start at (r_x , 0) and select points in a counterclockwise order, shifting from unit steps in y to unit steps in x when the slope becomes greater than -1.0. With parallel processors, we could calculate pixel positions in the two regions simultaneously. As an example of a sequential implementation of the midpoint algorithm, we take the start position at (0, r_y) and step along the ellipse path in clockwise order throughout the first quadrant.

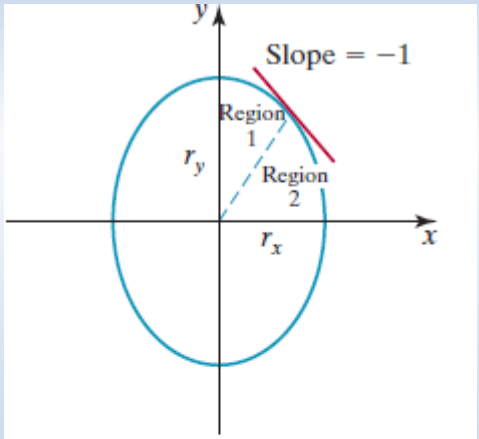


FIGURE Ellipse processing regions. Over region 1, the magnitude of the ellipse slope is less than 1.0; over region 2, the magnitude of the slope is greater than 1.0.

We define an ellipse function from Equation (2) with $(x_c , y_c) = (0, 0)$ as

$$f_{\text{ellipse}}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 \qquad (3)$$

which has the following properties:

$$f_{\text{ellipse}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the ellipse boundary} \\ = 0, & \text{if } (x, y) \text{ is on the ellipse boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the ellipse boundary} \end{cases} \qquad (4)$$

Starting at $(0, r_y)$, we take unit steps in the x direction until we reach the boundary between region 1 and region 2. Then we switch to unit steps in the y direction over the remainder of the curve in the first quadrant. At each step we need to test the value of the slope of the curve. The ellipse slope is calculated from Equation (3) as

$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y} \qquad (5)$$

At the boundary between region -1 and region 2 $dy/dx = -1.0$ and

$$2r_y^2 x = 2r_x^2 y \qquad (6)$$

Therefore, we move out of region 1 whenever

$$2r_y^2 x \geq 2r_x^2 y$$

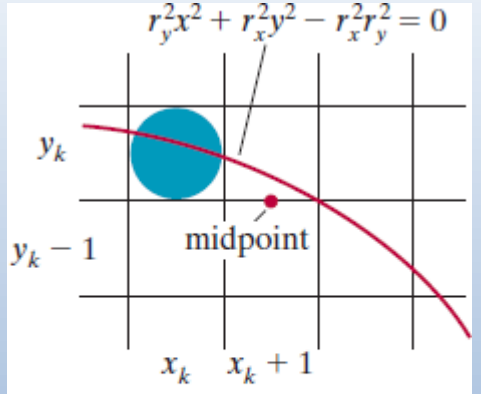


FIGURE Midpoint between candidate pixels at sampling position $x_k + 1$ along an elliptical path.

The decision parameter (that is, the ellipse function 3) at this midpoint:

$$p1_k = f_{\text{ellipse}}\left(x_k + 1, y_k - \frac{1}{2}\right)$$
$$= r_y^2(x_k + 1)^2 + r_x^2\left(y_k - \frac{1}{2}\right)^2 - r_x^2 r_y^2 \qquad (7)$$

If $p1_k < 0$, the midpoint is inside the ellipse and the pixel on scan line y_k is closer to the ellipse boundary. Otherwise, the midposition is outside or on the ellipse boundary, and we select the pixel on scan line $y_k - 1$.

At the next sampling position ($x_{k+1} + 1 = x_k + 2$), the decision parameter for region 1 is evaluated as

$$p1_{k+1} = f_{\text{ellipse}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$
$$= r_y^2[(x_k + 1) + 1]^2 + r_x^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_x^2 r_y^2 \qquad (8)$$

or

$$p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$$

where y_{k+1} is either y_k or $y_k - 1$, depending on the sign of $p1_k$.

Decision parameters are incremented by the following amounts:

$$\text{increment} = \begin{cases} 2r_y^2 x_{k+1} + r_y^2, & \text{if } p1_k < 0 \\ 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}, & \text{if } p1_k \geq 0 \end{cases} \qquad (9)$$

At the initial position $(0, r_y)$, these two terms evaluate to

$$2r_y^2 x = 0$$
$$2r_x^2 y = 2r_x^2 r_y \qquad (10)$$

In region 1, the initial value of the decision parameter is obtained by evaluating the ellipse function at the start position $(x_0, y_0) = (0, r_y)$:

$$p1_0 = f_{\text{ellipse}}\left(1, r_y - \frac{1}{2}\right)$$
$$= r_y^2 + r_x^2\left(r_y - \frac{1}{2}\right)^2 - r_x^2 r_y^2 \qquad (11)$$

or

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

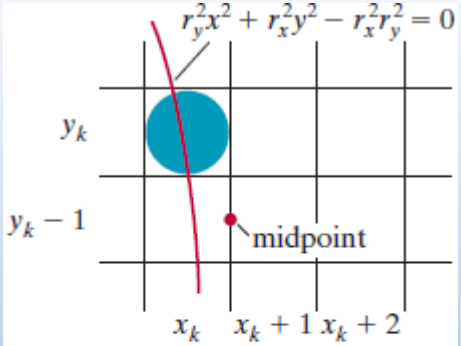


FIGURE Midpoint between candidate pixels at sampling position $y_k - 1$ along an elliptical path.

Over region 2, we sample at unit intervals in the negative *y* direction, and the midpoint is now taken between horizontal pixels at each step (Figure). For this region, the decision parameter is evaluated as

$$p2_k = f_{\text{ellipse}}\left(x_k + \frac{1}{2}, y_k - 1\right)$$
$$= r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2$$

$$p2_{k+1} = f_{\text{ellipse}}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right)$$
$$= r_y^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x^2[(y_k - 1) - 1]^2 - r_x^2 r_y^2$$

or

$$p2_{k+1} = p2_k - 2r_x^2(y_k - 1) + r_x^2 + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$$

$$p2_0 = f_{\text{ellipse}}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$
$$= r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

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

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_k position in region 1, starting at $k = 0$, perform the following test: If $p1_k < 0$, the next point along the ellipse centered on $(0, 0)$ is (x_{k+1}, y_k) and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

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

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

with

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

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

$$p2_0 = r_y^2 \left(x_0 + \frac{1}{2} \right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

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 $p2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

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

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

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 these coordinate values:

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

Example: Midpoint Ellipse Drawing (Textbook P114-115)

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

$$2r_y^2 x = 0 \qquad \text{(with increment } 2r_y^2 = 72\text{)}$$
$$2r_x^2 y = 2r_x^2 r_y \qquad \text{(with increment } -2r_x^2 = -128\text{)}$$

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

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 = -332$$

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

k	$p1_k$	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	−332	(1, 6)	72	768
1	−224	(2, 6)	144	768
2	−44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	−108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

We now move out of region 1 because $2r_y^2 x > 2r_x^2 y$.

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

$$p2_0 = f_{\text{ellipse}}\left(7 + \frac{1}{2}, 2\right) = -151$$

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

k	$P2_k$	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	−151	(8, 2)	576	256
1	233	(8, 1)	576	128
2	745	(8, 0)	—	—

A plot of the calculated positions for the ellipse within the first quadrant is shown in Figure 23.

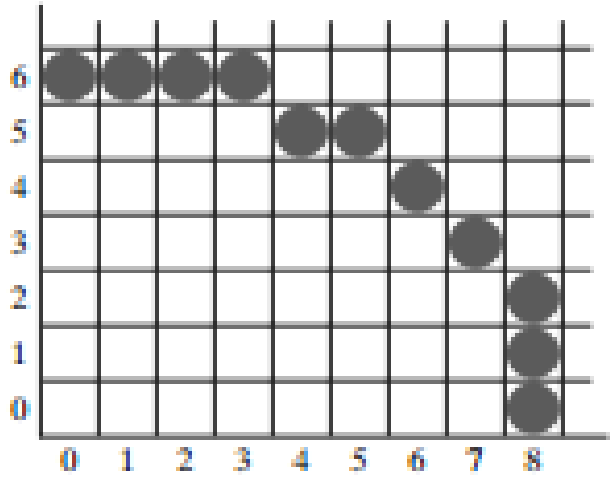


FIGURE 23
Pixel positions along an elliptical path centered on the origin with $r_x = 8$ and $r_y = 6$, using the midpoint algorithm to calculate locations within the first quadrant.

End of Lecture
Good Luck!

See you

in next lecture...