Output Primitives

Ellipse Drawing

Ellipses

. An ellipses is an elongated circle and can be drawn with modified circle drawing algorithm.

. An ellipse has set of fixed points (foci) that will have a constant total of distance from all the points on the boundary.

. An ellipse has 2 axes

. Major: straight line running through the two foci and the long axis.

. Minor: straight line running through the centre that bisects the major axis – the short axis.

.Ellipse symmetry

. Ellipses are symmetrical between the four quadrants.

. Ellipses drawing algorithm only need to calculate the pixels for one quadrant.

.The general ellipse algorithm in Cartesian co-ordinates is

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

.As for circles, algorithms that use Cartesian or polar co ordinates are computationally expensive.

Mid-Point Ellipses Algorithm

. A modification of the mid-point circle algorithm

Incremental integer calculations.

. Calculations are easier if:

1. **the circle is centered on the origin (0,0)**

. points are placed in the correct position by adding x_c to the x co-ordinates and y_c to the y co-ordinates. ie performing a translation

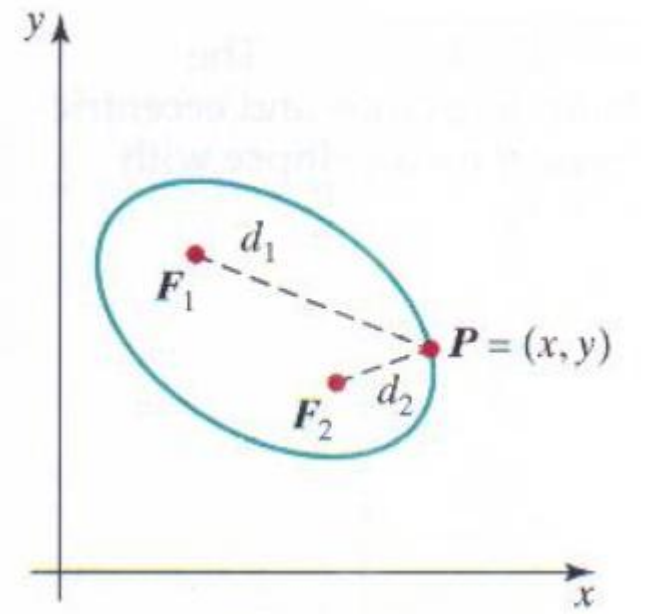
2. **the major and minor axes are aligned to be parallel with x and y axes.**

. ellipses are drawn at the correct angle by rotating the points to their correct position.

General equation

- General equation of an ellipse:

$$d_1 + d_2 = constant$$

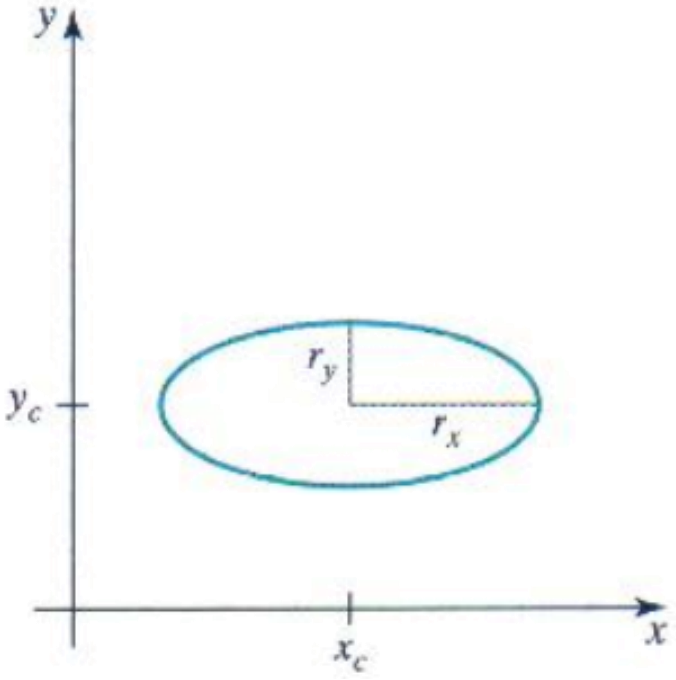


$$\sqrt{(x-x_1)^2+(y-y_1)^2}+\sqrt{(x-x_2)^2+(y-y_2)^2}=\text{constant}$$

"Standard" equation

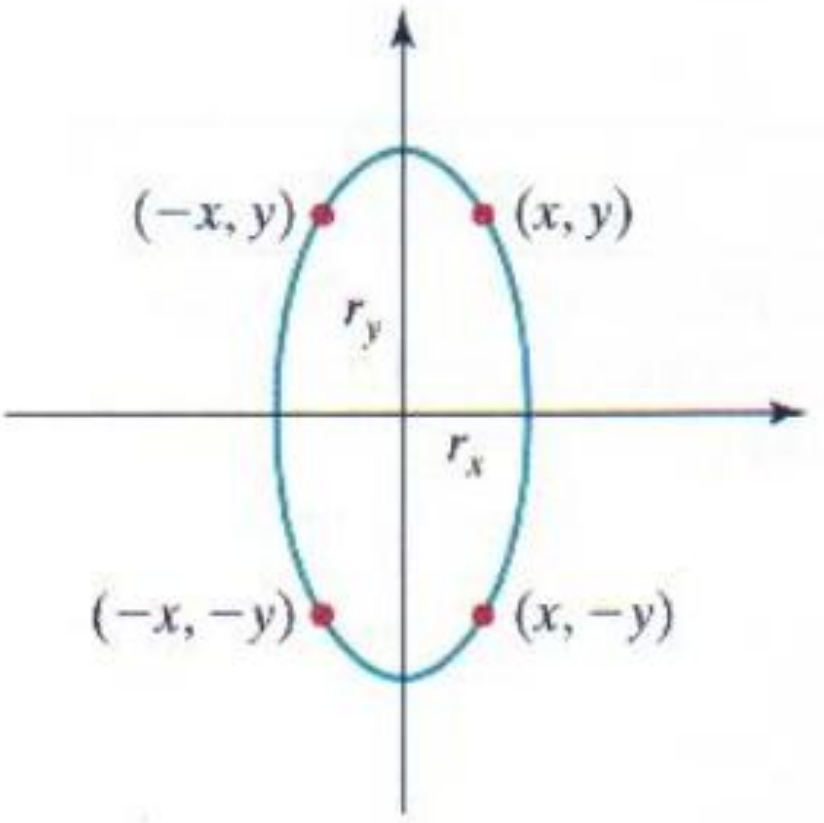
- However, we will only consider 'standard' ellipse:

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



Symmetry of an ellipse

- An ellipse only has a 2-way symmetry
- Calculation of a point (x,y) in one quadrant yields the ellipse points shown for the other three quadrants



Equation of an ellipse revisited

- Consider an ellipse centered at the origin:

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

- What is the discriminator function?

$$f_e(x,y)=r_y{}^2x^2+r_x{}^2y^2-r_x{}^2r_y{}^2$$

- …

Equation of an ellipse revisited

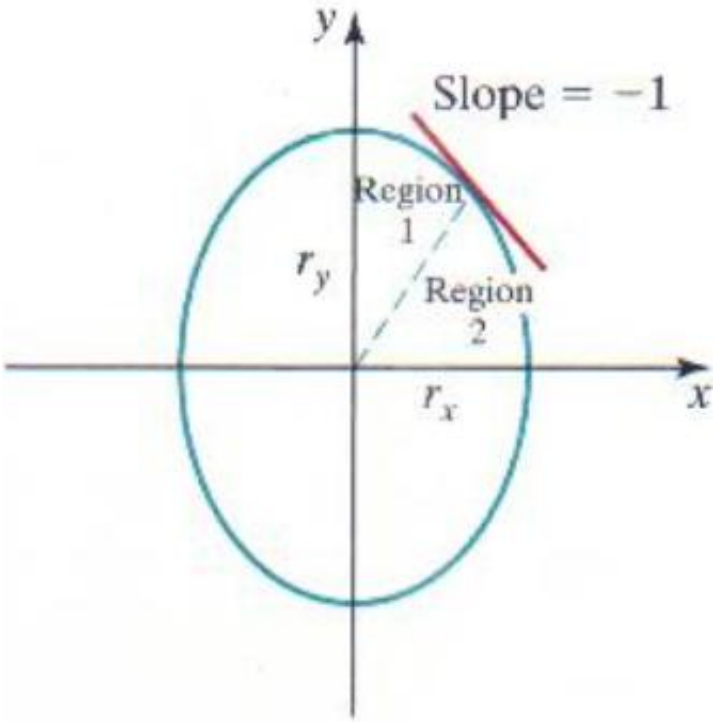
…and its properties:

$f_e(x,y) < 0$ for a point inside the ellipse
$f_e(x,y) > 0$ for a point outside the ellipse
$f_e(x,y) = 0$ for a point on the ellipse

- Need to divide the quadrant into two regions and process the pixel in each region separately
  - In region 1 the ellipse increases faster along the x axis than y axis.
  - In region 2 the ellipse increases faster along the y axis than x axis.

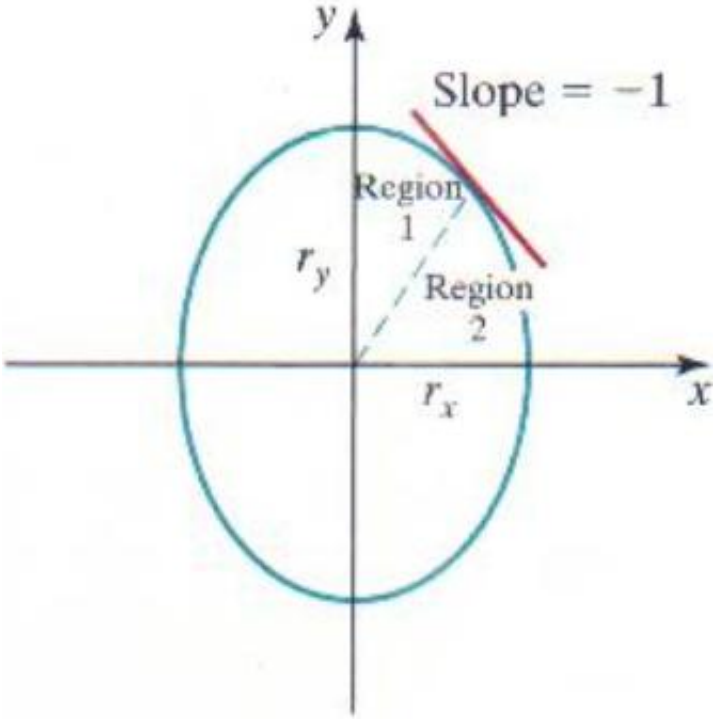
Midpoint Ellipse Algorithm

- Ellipse is different from circle.
- Similar approach with circle, different is sampling direction.
- Ellipse slope = dy/dx = - [$2r_y^2x$ / $2r_x^2y$]
- At the boundary between region 1 and region 2, dy/dx = -1 and $2r_y^2x = 2r_x^2y$
- Therefore we move out of region 1 whenever $2r_y^2x >= 2r_x^2y$



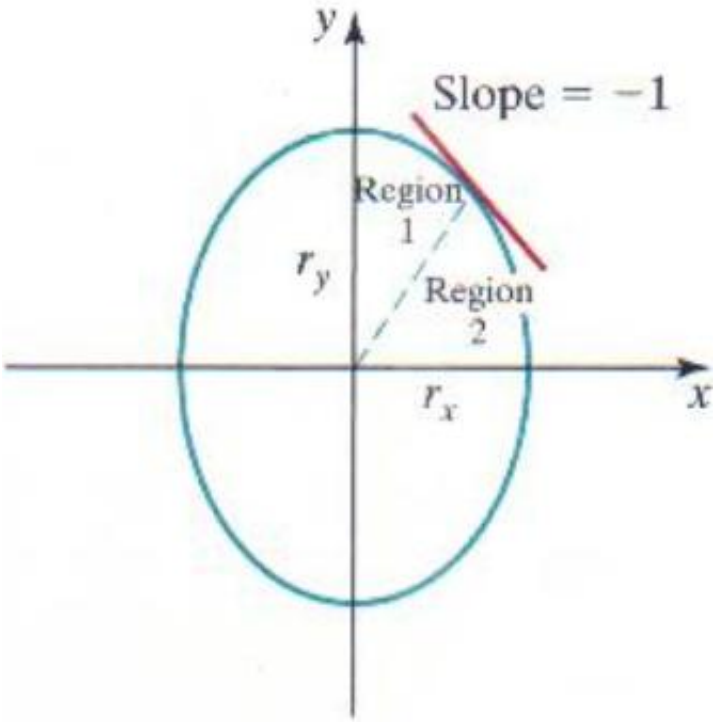
Midpoint Ellipse Algorithm

- Region 1:
  - Sampling is at *x* direction
  - Choose between (x_k+1, y_k), or (x_k+1, y_k-1)
  - Midpoint: $(x_k+1, y_k-0.5)$



Midpoint Ellipse Algorithm

- Region 2:
  - Sampling is at *y* direction
  - Choose between (x_k, y_k-1), or (x_k+1, y_k-1)
  - Midpoint: $(x_k+0.5, y_k-1)$



Decision parameters

- Region 1: $p1_k = f_e(x_k + 1, y_k - \frac{1}{2})$

–ve	• midpoint is inside • choose pixel (x_k+1, y_k)
+ve	• midpoint is outside • choose pixel (x_k+1, y_k-1)

Decision parameters

- Region 2: $p2_k = f_e(x_k + ½, y_k - 1)$

–ve	• midpoint is inside • choose pixel (x_k+1, y_k-1)
+ve	• midpoint is outside • choose pixel (x_k, y_k-1)

Midpoint Ellipse Algorithm

1. Input r_x, r_y and ellipse center (x_c, y_c). First point on the similar ellipse centered at the origin is (0, r_y).

$$(x_0, y_0) = (0, r_y)$$

2. Initial value for decision parameter at **region 1**:

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

Midpoint Ellipse Algorithm

3. At each x_k in region 1, starting from k = 0, test $p1_k$:

   If $p1_k < 0$, next point (x_k+1, y_k) and

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

   else, next point (x_k+1, y_k-1) and

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

Midpoint Ellipse Algorithm

4. Determine symmetry points in the other 3 octants.

5. Get the actual point for ellipse centered at (x_c, y_c) that is $(x + x_c, y + y_c)$.

6. Repeat step 3 - 6 until $2r_y^2 x \geq 2r_x^2 y$.

7. Initial value for decision parameter in **region 2**:

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

Midpoint Ellipse Algorithm

7. At each y_k in region 2, starting from k = 0, test $p2_k$:

   If $p2_k > 0$, next point is (x_k, y_k-1) and

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

   else, next point is (x_k+1, y_k-1) and

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

Midpoint Ellipse Algorithm

9. Determine symmetry points in the other 3 octants.

10. Get the actual point for ellipse centered at (x_c, y_c) that is $(x + x_c, y + y_c)$.

11. Repeat step 8 - 10 until $y < 0$.

Example: $r_x = 8$ and $r_y = 6$

- $2r_y^2x = 0$ (with increment $2r_y^2 = 72$)
- $2r_x^2y = 2r_x^2 r_y$ (with increment $-2r_x^2 = -128$)
- For region 1: $(x_0, y_o) = (0,6)$
- $P1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 = -332$

$2r_y^2x > 2r_x^2y$

Initial coord for region 2 is (7, 3)

$P2_0 = f(7 + 1/2, 2) = -151$

K	$P1_k$	(x_{k+1}, y_{k+1})	$2r_y^2x_{k+1}$	$2r_x^2xy_{k+1}$
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**

The codes

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

 /*  The following procedure accepts values for an ellipse
  *  center position and its semimajor and semiminor axes, then
  *  calculates ellipse positions using the midpoint algorithm.
  */
 void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry)
 {
    int Rx2 = Rx * Rx;
    int Ry2 = Ry * Ry;
    int twoRx2 = 2 * Rx2;
    int twoRy2 = 2 * Ry2;
    int p;
    int x = 0;
    int y = Ry;
    int px = 0;
    int py = twoRx2 * y;
    void ellipsePlotPoints (int, int, int, int);
```

The codes

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

 /*  The following procedure accepts values for an ellipse
  *  center position and its semimajor and semiminor axes, then
  *  calculates ellipse positions using the midpoint algorithm.
  */
 void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry)
 {
    int Rx2 = Rx * Rx;
    int Ry2 = Ry * Ry;
    int twoRx2 = 2 * Rx2;
    int twoRy2 = 2 * Ry2;
    int p;
    int x = 0;
    int y = Ry;
    int px = 0;
    int py = twoRx2 * y;
    void ellipsePlotPoints (int, int, int, int);

    /* Plot the initial point in each quadrant. */
    ellipsePlotPoints (xCenter, yCenter, x, y);
```

The codes

```
/* Region 1 */
p = round (Ry2 - (Rx2 * Ry) + (0.25 * Rx2));
while (px < py) {
   x++;
   px += twoRy2;
   if (p < 0)
      p += Ry2 + px;
   else {
      y--;
      py -= twoRx2;
      p += Ry2 + px - py;
   }
   ellipsePlotPoints (xCenter, yCenter, x, y);
}
```

The codes

```
   /* Region 2 */
   p = round (Ry2 * (x+0.5) * (x+0.5) + Rx2 * (y-1) * (y-1) - Rx2 * Ry2);
   while (y > 0) {
      y--;
      py -= twoRx2;
      if (p > 0)
         p += Rx2 - py;
      else {
         x++;
         px += twoRy2;
         p += Rx2 - py + px;
      }
      ellipsePlotPoints (xCenter, yCenter, x, y);
   }
}

void ellipsePlotPoints (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);
}
```