ترسیم اشکال پایه هندسی

مباحث این فصل:

- رسم نقطه
- رسم خط
  - الگوریتم DDA
  - الگوریتم برسنهام
- رسم دایره
  - الگوریتم زاویه
  - الگوریتم نقطه میانی
- رسم بیضی
  - الگوریتم زاویه
  - الگوریتم نقطه میانی

هر تصویر را میتوان به روشهای گوناگون توصیف کرد. برای مثال میتوان هر تصویر را به صورت مجموعه ای از اشیاء در نظر گرفت و سپس این اشیاء را بصورت اشکال اولیه مانند نقطه، خط ، دایره و ... مدل کرد و در نهایت این اشکال اولیه را توسط مجموعه ای از پیکسل های نورانی نمایش داد.

رسم نقطه:

برای رسم نقطه تنها باید به طریقی مختصات نقطه مورد نظر را به رویه ای مناسب برای دستگاه خروجی ببریم. در سیستم های راستر مکان متناظر با نقطه، در فریم بافر مقداردهی میشود و سپس هنگام تابیدن الکترون مکان مورد نظر روشن میشود.

معمولاً برای مقداردهی کردن فریم بافر (ترسیم نقطه) از تابع سطح پایین زیر استفاده میشود:

setpixel (x , y, color)

رسم خط:

الگوریتم DDA :

یکی از الگوریتم های ترسیم خط الگوریتم DDA است که بر پایه معادله شیب خط عمل میکند. بسته به مقدار شیب خط یکی از فرمول های زیر را استفاده خواهیم کرد.

$$\Delta y = m\Delta x \Rightarrow \begin{cases} y_{k+1} = y_k + m\Delta x \overset{\Delta x=1}{\Rightarrow} y_{k+1} = y_k + m & (1) \\ x_{k+1} = x_k + \dfrac{1}{m}\Delta y \overset{\Delta y=1}{\Rightarrow} x_{k+1} = x_k + \dfrac{1}{m} & (2) \end{cases}$$

الف) در صورتیکه شیب خط مقدار مثبت و کمتر از یک باشد در این صورت در هر مرحله مقدار Δx را یک واحد افزایش میدهیم و سپس مقدار y را از فرمول 1 محاسبه میکنیم.

ب) در صورتیکه شیب خط مقدار مثبت و بیشتر از یک باشد در این صورت در هر مرحله مقدار Δy را یک واحد افزایش میدهیم و سپس مقدار x را از فرمول 2 محاسبه میکنیم.

از آنجا که مقدار m میتواند هر عدد حقیقی باشد بنابراین نیاز به گرد کردن عدد حاصل داریم .الگوریتم DDA الگوریتم سریعی برای پیدا کردن پیکسل ها است که مستقیماً از معادله خط استفاده میکند. اما به دلیل عملیات پیاپی در گردکردن میتواند منجر به خطاهای چشمگیری در خطوط طولانی شود. همچنین عملیات گرد کردن و کار با اعداد اعشاری بسیار وقت گیر است.

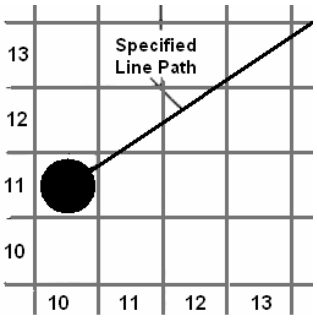
رویه زیر، نحوه اجرای این الگوریتم را نشان میدهد:

```
void dda_line(const int x_1,const int y_1,const int x_2,const int y_2)
  {
    int color=getcolor( );
    int x1=x_1;
    int y1=y_1;
    int x2=x_2;
    int y2=y_2;
    if(x_1>x_2)
        {
          x1=x_2;
```

```
            y1=y_2;
            x2=x_1;
            y2=y_1;
        }
    float dx=(x2-x1);
    float dy=(y2-y1);
    int steps=abs(dy);
    if(abs(dx)>abs(dy))
        steps=abs(dx);
    float x_inc=(dx/(float)steps);
    float y_inc=(dy/(float)steps);
    float x=x1;
    float y=y1;
    putpixel(x,y,color);
    for(int count=1;count<=steps;count++)
        {
            x+=x_inc;
            y+=y_inc;
            putpixel((int)(x+0.5),(int)(y+0.5),color);
        }
}
```

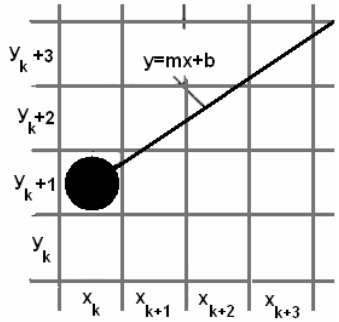
الگوریتم Bresenham :

این الگوریتم یکی از الگوریتم های سریع و مناسب برای رسم خط است که توسط برسنهام ایجاد شد. و تنها از محاسبات بر روی اعداد صحیح استفاده میکند ضمناً میتواند با کمی تغییر برای رسم دایره و سایر خطوط منحنی استفاده شود.

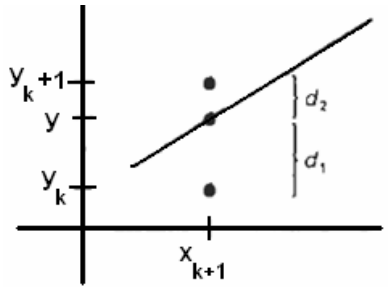


در شکل روبرو بخشی از یک خط مستقیم نشان داده شده است .

پس از رسم نقطه $(10,11)$ در مرحله بعد ما باید تصمیم بگیریم که کدام یک از نقاط $(11,11)$ و یا $(11,12)$ انتخاب شود. این سؤال توسط الگوریتم برسنهام، و به کمک علامت یک پارامتر تصمیم گیری که فاصله بین هر یک از پیکسل ها و خط واقعی را نشان میدهد، پاسخ داده میشود. تنها مشکل این الگوریتم آن است که به شیب خط مورد نظر بستگی دارد.



برای نشان دادن روش الگوریتم، ابتدا رسم خطوط با شیب مثبت و کمتر از 1 را بررسی میکنیم. ما در هر مرحله، مؤلفه x را یک واحد افزایش میدهیم، سپس مؤلفه y را محاسبه میکنیم. شکل روبرو الگوریتم را در مرحله k ام نشان میدهد. فرض کنید ما تشخیص داده ایم که پیکسل (x_k, y_k) رسم شود. در مرحله بعد باید تصمیم بگیریم که کدام پیکسل در ستون x_{k+1} رسم شود. انتخاب های ما پیکسل های (x_{k+1}, y_k) و (x_{k+1}, y_{k+1}) هستند.



ابتدا مقدار واقعی y در ستون x_{k+1} را از رابطه زیر بدست می آوریم و سپس فاصله بین هر یک از پیکسلها را تا این مقدار محاسبه میکنیم. (شکل روبرو)

$$y = m(x_k+1)+b$$
$$d_1 = y - y_k = m(x_k+1)+b-y_k$$
$$d_2 = y_k+1-y = y_k+1-m(x_k+1)+b$$
$$d_1-d_2 = 2m(x_k+1)-2y_k+2b-1 \qquad (*)$$

پارامتر تصمیم گیری p_k در هر مرحله از رابطه (*) بدست می آید. با جایگزینی $m = \Delta y / \Delta x$ در رابطه، پارامتر تصمیم گیری بصورت زیر بدست می آید.

$$p_k = \Delta x(d_1-d_2) = 2\Delta y.x_k - 2\Delta x.y_k + c$$

که در آن c برابر است با $2\Delta y + \Delta x(2b-1)$ و مقدار ثابتی است که میتوان آن را در معادلات بازگشتی نادیده گرفت. حال میتوان بیان کرد:

1- اگر p_k دارای مقدار منفی بود آنگاه پیکسل قرار گرفته در y_k به خط نزدیکتر است و پیکسل پایینی انتخاب خواهد شد.

2- اگر p_k دارای مقدار مثبت بود آنگاه پیکسل قرار گرفته در y_k+1 به خط نزدیکتر است و پیکسل بالایی انتخاب خواهد شد.

حال در مرحله بعد، در ستون x_k+1 داریم :

$$p_{k+1} = 2\Delta y.x_{k+1} - 2\Delta x.y_{k+1} + c$$

مقدار دو پارامتر را از هم کم میکنیم :

$$p_{k+1} - p_k = 2\Delta y.(x_{k+1}-x_k) - 2\Delta x.(y_{k+1}-y_k)$$

که در آن $(y_{k+1}-y_k)$ میتواند مقدار 0 یا 1 داشته باشد. این رابطه بازگشتی باید در هر مرحله محاسبه شود. مقدار اولیه رابطه بازگشتی را نیز از رابطه p_k بصورت زیر بدست می آوریم.

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

رویه زیر پیاده سازی این الگوریتم به زبان C++ را نشان میدهد:

```
void bresenham_line(const int x_1,const int y_1,const int x_2,const int y_2)
  {
    int color=getcolor( );
    int x1=x_1;
    int y1=y_1;
    int x2=x_2;
    int y2=y_2;
    if(x_1>x_2)
        {
          x1=x_2;
          y1=y_2;
          x2=x_1;
          y2=y_1;
        }
```

```
    int dx=abs(x2-x1);
    int dy=abs(y2-y1);
    int two_dy=(2*dy);
    int two_dy_dx=(2*(dy-dx));
    int p=((2*dy)-dx);
    int x=x1;
    int y=y1;
    putpixel(x,y,color);
    while(x<x2)
        {
          x++;
          if(p<0)
              p+=two_dy;
          else
              {
                y--;
                p+=two_dy_dx;
              }
          putpixel(x,y,color);
        }
  }
```

الگوریتم رسم دایره :

یک دایره به صورت مجموعه نقاطی است که از یک نقطه مرکزی (x_c, y_c) به فاصله مشخص r هستند. معادله دایره در دستگاه دکارتی به صورت $(x-x_c)^2+(y-y_c)^2=r^2$ تعریف میشود.

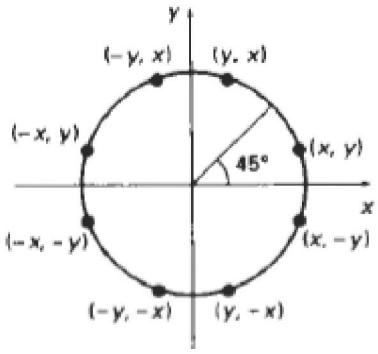
ما میتوانیم از همین معادله برای پیدا کردن نقاط پیکسلها استفاده کنیم. بدین صورت که مؤلفه x را از بازه x_c-r تا x_c+r تغییر میدهیم و مقدار y را برای هر x از رابطه $y=y_c\pm\sqrt{r^2-(x_c-x)^2}$ بدست آوریم.

اما این روش خوبی برای تولید دایره نیست، زیرا اولاً در هر مرحله نیاز به انجام محاسبات فراوانی است و ثانیاً همانطور که در شکل زیر نشان داده شده است فاصله بین پیکسلها همه جا ثابت نیست.

یک روش برای حذف فضاهای خالی عوض کردن جای مؤلفه های x, y در جایی است که شیب خط بیشتر از یک یا کمتر از 1- باشد. که این روش نیز، نیاز به محاسبات فراوان دارد . روش دیگر برای حذف فضاهای بین پیکسلی استفاده از روش زاویه ای و رسم دایره در مختصات قطبی بر پایه r و θ می باشد. معمولاً مقدار θ را در هر مرحله به مقدار $\frac{1}{r}$ افزایش میدهند تا دایره رسم شود.

$$\begin{cases} x=x_c+r\cos\theta \\ y=y_c+r\sin\theta \end{cases}$$

نکته مهم اینجاست که ما میتوانیم میزان محاسبات را با توجه به تقارن دایره کاهش دهیم . ما میتوانیم تنها $\frac{1}{8}$ دایره را رسم کرده و سپس کل دایره را با تقارن از محورهای $x, y, x=y$ بدست آوریم و تنها باید دایره را برای 45° محاسبه کنیم.



ما تنها دایره را از نقطه $x=0$ تا $x=y$ رسم کرده و سپس آنرا با تقارن دایره کامل میکنیم. اما همانطور که گفته شد معادلات دکارتی و قطبی روش خوبی برای رسم دایره نیستند. زیرا روش دکارتی نیاز به محاسبه ضرب و رادیکال و روش قطبی نیاز به محاسبات مثلثاتی دارد. روش بهتر الگوریتم MidPoint است .

ابتدا پیاده سازی الگوریتم رسم دایره با زاویه را در زبان C++ نشان میدهیم و سپس به الگوریتم نقطه میانی خواهیم پرداخت .

الگوریتم زاویه برای رسم دایره:

```
void trigonometric_circle(const int h,const int k,const int r)
  {
    int color=getcolor( );
    float x=0;
    float y=r;
    float angle=0;
    float range=M_PI_4;
    do
        {
          putpixel((int)(h+x+0.5),(int)(k+y+0.5),color);
          putpixel((int)(h+y+0.5),(int)(k+x+0.5),color);
          putpixel((int)(h+y+0.5),(int)(k-x+0.5),color);
          putpixel((int)(h+x+0.5),(int)(k-y+0.5),color);
          putpixel((int)(h-x+0.5),(int)(k-y+0.5),color);
          putpixel((int)(h-y+0.5),(int)(k-x+0.5),color);
          putpixel((int)(h-y+0.5),(int)(k+x+0.5),color);
          putpixel((int)(h-x+0.5),(int)(k+y+0.5),color);
          angle+=0.001;
          x=(r*cos(angle));
          y=(r*sin(angle));
        }
    while(angle<=range);
  }
```

Midpoint Circle Drawing Algorithm :

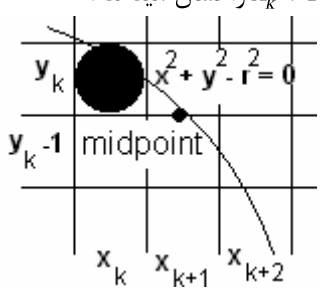
دراین روش ما در هر مرحله نزدیکترین نقطه به مسیر دایره را انتخاب میکنیم. برای یک شعاع داده شده r و یک مرکز (x_c, y_c) ابتدا دایره را به مرکز $(0,0)$ رسم میکنیم سپس نقاط محاسبه شده را با مقدار (x_c, y_c) جمع میکنیم تا دایره به نقطه اصلی منتقل شود.

دایره را از $x=0$ تا $x=y$ رسم میکنیم در این بازه شیب منحنی بین 0 تا 1- تغییر میکند بنابراین ما میتوانیم با افزایش مقدار x در هر مرحله از یک پارامتر تصمیم گیری برای تعیین اینکه کدام یک از دو پیکسل به دایره نزدیکترند، استفاده کنیم.

برای استفاده از روش نقطه میانی، تابع دایره را به صورت $f(x,y) = x^2 + y^2 - r^2$ تعریف میکنیم . در این حالت هر نقطه روی محیط دایره مقدار تابع f را برابر 0 میکند. و اگر نقطه داخل دایره باشد مقدار f منفی و در صورت خارج بودن نقطه از دایره این مقدار مثبت خواهد بود. بنابراین بطور خلاصه داریم :

$$f(x,y)\begin{cases} <0 & \text{Point is inside the Circle boundary} \\ =0 & \text{Point is on the Circle boundary} \\ >0 & \text{Point is outside the Circle boundary} \end{cases}$$

شکل زیر دو پیکسل کاندید شده در ستون x_k+1 را نشان میدهد .



فرض کنید پیکسل (x_k, y_k) را انتخاب کرده ایم، حال در مرحله بعدی باید یکی از دو پیکسل (x_k+1, y_k) یا (x_k+1, y_k-1) را انتخاب کنیم. تابع دایره برای نقطه میانی بین دو پیکسل یعنی نقطه $(x_k+1, y_k-\frac{1}{2})$ را محاسبه میکنیم و داریم :

$$p_k = f_{circle}(x_k+1, y_k-\frac{1}{2}) = (x_k+1)^2 + (y_k-\frac{1}{2})^2 - r^2$$

حال اگر p_k مثبت بود، پیکسل پایینی یا (x_k+1, y_k-1) انتخاب میشود . در غیر این صورت پیکسل بالایی یا همان (x_k+1, y_k) انتخاب می شود.

برای راحتی کار یک فرمول بازگشتی برای محاسبه p_k بدست می آوریم.

$$p_{k+1} = f_{circle}(x_k+2, y_{k+1}-\frac{1}{2}) = (x_k+2)^2 + (y_{k+1}-\frac{1}{2})^2 - r^2$$

که در آن مقدار y_{k+1} بسته به علامت p_k میتواند یکی از دو مقدار y_k یا y_k-1 باشد. با مقایسه p_k و p_{k+1} داریم :

$$p_{k+1} = \begin{cases} p_k + 2x_k + 3 & \text{if } y_{k+1} = y_k \ ;(P_k > 0) \\ p_k + 2x_k - 2y_k + 5 & \text{if } y_{k+1} = y_k - 1 \ ;(P_k < 0) \end{cases}$$

مقدار اولیه پارامتر تصمیم گیری به ازای نقطه شروع $(0,r)$ برابر است با :

$$p_0 = f_{circle}(1, r-\frac{1}{2}) = 1 + (r-\frac{1}{2})^2 - r^2 = \frac{5}{4} - r$$

رویه زیر پیاده سازی این الگوریتم در زبان C++ را نشان میدهد.

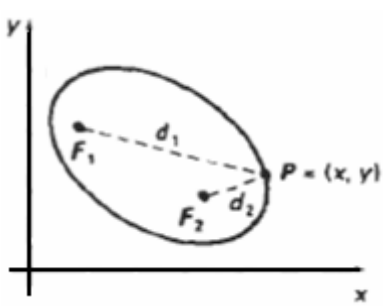
```
void midpoint_circle(const int h,const int k,const int r)
  {
    int color=getcolor( );
    int x=0;
    int y=r;
    int p=(1-r);
    do   {
            putpixel((h+x),(k+y),color);
            putpixel((h+y),(k+x),color);
            putpixel((h+y),(k-x),color);
            putpixel((h+x),(k-y),color);
            putpixel((h-x),(k-y),color);
            putpixel((h-y),(k-x),color);
            putpixel((h-y),(k+x),color);
            putpixel((h-x),(k+y),color);
            x++;
            if(p<0)
                p+=((2*x)+1);
            else {
                  y--;
                  p+=((2*(x-y))+1);}
         }
    while(x<=y);
  }
```

الگوریتم رسم بیضی :

در یک تعریف مقدماتی ، بیضی یک دایره کشیده شده است . بنابراین ، اشکال بیضیوار را می توان با اصلاح الگوریتم رسم دایره رسم کرد .

خواص بیضی :

یک بیضی بصورت مجموعه ای از نقاط تعریف می شود که مجموع فاصله آنها از دو نقطه ثابت (کانون ها) مقداری ثابت است .



$$d_1 + d_2 = \text{Constant}$$

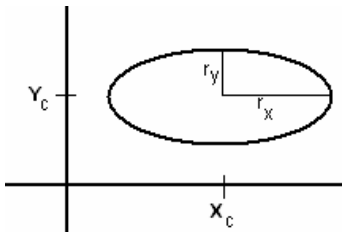
$$F_1 = (x_1, y_1), F_2 = (x_2, y_2)$$

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

معادله بیضی را با استفاده از روابط بالا به صورت زیر هم می توان نوشت:

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

که در آن ضرایب A,B.C.D.E.F با توجه به مختصات نقاط کانونی و اندازه قطر اصلی و فرعی بیضی مشخص می شود . قطر اصلی ، خط مستقیمی است که از دو کانون بیضی می گذرد و قطر دوم عمود منصف قطر اول است . یک روش برای مشخص کردن بیضی داشتن مختصات نقاط کانونی و مختصات یک نقطه روی محیط بیضی است . با این سه مختصات می توانیم مقدار ثابت معادله را محاسبه کرده و سپس ضرایب معادله را بدست آوریم . در صورتی که قطر اصلی و فرعی بیضی هم جهت محورهای مختصات باشند ، معادله بیضی بسیار ساده خواهد بود . شکل بیضی استاندارد بصورت شکل مقابل بوده و معادله آنرا میتوان بصورت زیر نوشت:



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

در مختصات قطبی نیز این معادله بصورت $\begin{cases} x = x_c + r_x \cos\theta \\ y = y_c + r_y \sin\theta \end{cases}$ نوشته میشود.

همچنین میتوان با در نظر گرفتن تقارن بیضی میزان محاسبات را کاهش داد. بر خلاف دایره که در هر یک هشتم متقارن است، یک بیضی استاندارد در هر یک چهارم متقارن خواهد بود، بنابراین باید بیضی ربع اول رسم کرده و سپس با استفاده از تقارن، بیضی را کامل کرد.

الگوریتم زاویه برای رسم بیضی:

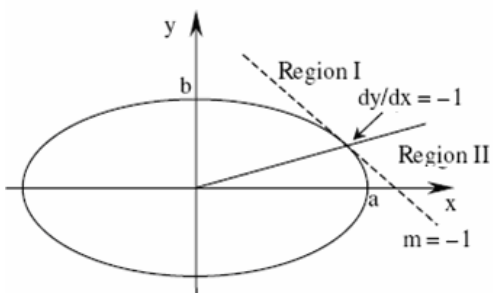
پیاده سازی روش زاویه در C++ بصورت زیر است:

```
void trigonometric_ellipse(const int h,const int k,const int rx,const int ry)
  {
    int color=getcolor( );
    float x=0;
    float y=ry;
    float angle=0;
    float range=rx;
    do
        {
          putpixel((int)(h+x+0.5),(int)(k+y+0.5),color);
          putpixel((int)(h+x+0.5),(int)(k-y+0.5),color);
          putpixel((int)(h-x+0.5),(int)(k-y+0.5),color);
          putpixel((int)(h-x+0.5),(int)(k+y+0.5),color);
          angle+=0.05;
          x=(rx*cos(angle));
          y=(ry*sin(angle));
        }
    while(angle<=range);
  }
```

الگوریتم نقطه میانی برای رسم بیضی :

این الگوریتم یک روش رسم بیضی در گرافیک کامپیوتری است. این الگوریتم تغییر یافته الگوریتم برسنهام است. مزیت این روش تغییر یافته این است که در حلقه برنامه تنها به عمل جمع نیاز است که منجر به پیاده سازی آسان و سریع آن در تمام پردازنده ها میشود.

اجازه دهید تنها به یک چهارم اول بیضی فکر کنیم. منحنی خود به دو ناحیه تقسیم میشود. در ناحیه اول، شیب منحنی بیشتر از -1 است. در حالیکه در ناحیه دوم شیب کمتر از -1 است.

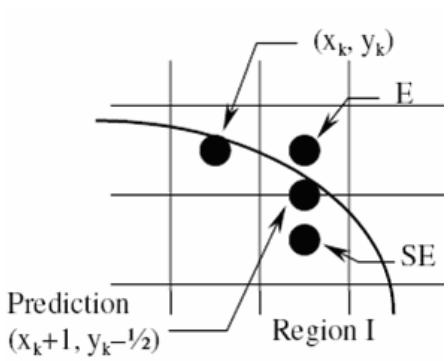


به معادله کلی بیضی توجه کنید:

$$b^2x^2 + a^2y^2 - a^2b^2 = 0$$

که در آن a شعاع افقی و b شعاع عمودی است. ما می توانیم از همین تابع برای رسم بیضی استفاده کنیم.

در ناحیه اول که رابطه $dy/dx > -1$ برقرار است :



در هر مرحله مؤلفه x یک واحد افزایش می یابد. به عبارتی داریم : $x_{k+1} = x_k + 1$

داریم $y_{k+1} = y_k$ هر گاه پیکسل E انتخاب شود و داریم $y_{k+1} = y_k - 1$ هر گاه پیکسل SE انتخاب شود. برای اینکه بتوانیم بین انتخاب دو پیکسل S و SE تصمیم بگیریم، نقطه میانی دو پیکسل کاندید را در معادله قرار میدهیم. تابع پیشگوی P_k بصورت زیر تعریف میشود:

$$P_k = f(x_k+1, y_k-\tfrac{1}{2}) = b^2(x_k+1)^2 + a^2(y_k-\tfrac{1}{2})^2 - a^2b^2 = b^2(x_k^2+2x_k+1) + a^2(y_k^2 - y_k + \tfrac{1}{4}) - a^2b^2$$

if $P_k < 0$, *select E*:

$$P_{k+1}^{\ E} = f(x_k+2, y_k-\tfrac{1}{2}) = b^2(x_k+2)^2 + a^2(y_k-\tfrac{1}{2})^2 - a^2b^2 = b^2(x_k^2+4x_k+4) + a^2(y_k^2 - y_k + \tfrac{1}{4}) - a^2b^2$$

$$\Delta P_k^{\ E} = P_{k+1}^{\ E} - P_k = b^2(2x_k+3)$$

if $P_k > 0,\ select\ SE$:

$P_{k+!}^{SE} = f(x_k+2, y_k-3/2) = b^2(x_k+2)^2 + a^2(y_k-3/2)^2 - a^2b^2 = b^2(x_k^2+4x_k+4) + a^2(y_k^2-3y_k+9/4) - a^2b^2$

$\Delta P_k^{SE} = P_{k+1}^{SE} - P_k = b^2(2x_k+3) - 2a^2(y_k-1)$

$calculate\ the\ change\ of\ \Delta P_k$:

if $E\ is\ selected$:

$\Delta P_{k+1}^{E} = b^2(2x_k+5)$

$\Delta^2 P_k^{E} = \Delta P_{k+1}^{E} - \Delta P_k^{E} = 2b^2$

$\Delta P_{k+1}^{SE} = b^2(2x_k+5) - 2a^2(y_k-1)$

$\Delta^2 P_k^{SE} = \Delta P_{k+1}^{SE} - \Delta P_k^{SE} = 2b^2$

if $SE\ is\ selected$:

$\Delta P_{k+1}^{E} = b^2(2x_k+5)$

$\Delta^2 P_k^{E} = \Delta P_{k+1}^{E} - \Delta P_k^{E} = 2b^2$

$\Delta P_{k+1}^{SE} = b^2(2x_k+5) - 2a^2(y_k-1)$

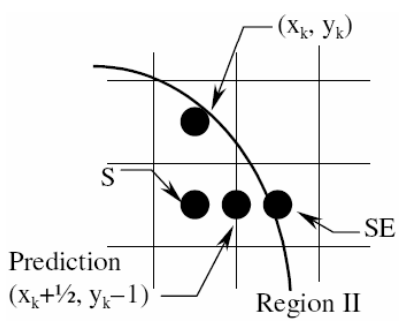
$\Delta^2 P_k^{SE} = \Delta P_{k+1}^{SE} - \Delta P_k^{SE} = 2(a^2+b^2)$

$Initial\ \ Value$:

$x_0 = 0, y_0 = b, P_0 = b^2 + 1/4\, a^2(1-4b)$

$\Delta P_0^{E} = 3b^2, \Delta P_0^{SE} = 3b^2 - 2a^2(b-1)$

در ناحیه دوم که $dy/dx < -1$ تمام محاسبات مانند ناحیه اول است، با این تفاوت که در ناحیه دوم مقدار y در هر 1 واحد کاهش می یابد.



در هر مرحله مؤلفه y یک واحد کاهش می یابد. به عبارتی داریم : $y_{k+1} = y_k - 1$

اگر S انتخاب شود داریم : $x_{k+1} = x_k$ و اگر SE انتخاب شود داریم : $x_{k+1} = x_k + 1$.

$P_k = f(x_k+1/2, y_k-1) = b^2(x_k+1/2)^2 + a^2(y_k-1)^2 - a^2b^2 = b^2(x_k^2+x_k+1/4) + a^2(y_k^2-2y_k+1) - a^2b^2$

if $P_k > 0,\ select$ S:

$P_{k+1}^{S} = f(x_k+1/2, y_k-2) = b^2(x_k+1/2)^2 + a^2(y_k-2)^2 - a^2b^2 = b^2(x_k^2+x_k+1/4) + a^2(y_k^2-4y_k+4) - a^2b^2$

$\Delta P_k^{S} = P_{k+1}^{S} - P_k = a^2(3-2y_k)$

if $P_k < 0$, *select SE*:

$P_{k+1}^{SE} = f(x_k + 3/2, y_k - 2) = b^2(x_k + 3/2)^2 + a^2(y_k - 2)^2 - a^2b^2 = b^2(x_k^2 + 3x_k + 9/4) + a^2(y_k^2 - 4y_k + 4) - a^2b^2$

$\Delta P_k^{SE} = P_{k+1}^{SE} - P_k = 2b^2(x_k + 1) + a^2(3 - 2y_k)$

calculate the change of ΔP_k :

if S *is selected* :

$\Delta P_{k+1}^{S} = a^2(5 - 2y_k)$

$\Delta^2 P_k^{S} = \Delta P_{k+1}^{S} - \Delta P_k^{S} = 2a^2$

$\Delta P_{k+1}^{SE} = 2b^2(x_k + 1) + a^2(5 - 2y_k)$

$\Delta^2 P_k^{SE} = \Delta P_{k+1}^{SE} - \Delta P_k^{SE} = 2a^2$

if *SE is selected* :

$\Delta P_{k+1}^{S} = a^2(5 - 2y_k)$

$\Delta^2 P_k^{S} = \Delta P_{k+1}^{S} - \Delta P_k^{E} = 2a^2$

$\Delta P_{k+1}^{SE} = 2b^2(2x_k + 2) - a^2(5 - 2y_k)$

$\Delta^2 P_k^{SE} = \Delta P_{k+1}^{SE} - \Delta P_k^{SE} = 2(a^2 + b^2)$

درمرز بین دو ناحیه داریم:

$$f(x, y) = 0, \quad \frac{dy}{dx} = \frac{-bx}{a^2\sqrt{1 - x^2/a^2}}.$$

$$when \quad dy/dx = -1, \quad x = \frac{a^2}{\sqrt{a^2 + b^2}} \quad and \quad y = \frac{b^2}{\sqrt{a^2 + b^2}}.$$

$$At\ region1,\ dy/dx > -1,\ x < \frac{a^2}{\sqrt{a^2 + b^2}} \quad and \quad y > \frac{b^2}{\sqrt{a^2 + b^2}},\ therefore$$

$$\Delta P_k^{SE} < b^2\left(\frac{2a^2}{\sqrt{a^2 + b^2}} + 3\right) - 2a^2\left(\frac{b^2}{\sqrt{a^2 + b^2}} - 1\right) = 2a^2 + 3b^2$$

Initial Value at region2 :

$$x_0 = \frac{a^2}{\sqrt{a^2 + b^2}} \quad and \quad y_0 = \frac{b^2}{\sqrt{a^2 + b^2}}$$

این الگوریتم در زبان C++ بصورت زیر پیاده سازی میشود.

```
void midpoint_ellipse(const int h,const int k,const int a,const int b)
  {
    float aa=(a*a);
    float bb=(b*b);
    float aa2=(aa*2);
    floatbb2=(bb*2);
    float x=0;
```

```
    float y=b;
    float fx=0;
    float fy=(aa2*b);
    float p=(int)(bb-(aa*b)+(0.25*aa)+0.5);
    putpixel((h+x),(k+y),color);
    putpixel((h+x),(k-y),color);
    putpixel((h-x),(k-y),color);
    putpixel((h-x),(k+y),color);
    while(fx<fy)
        {
          x++;
          fx+=bb2;
          if(p<0)
              p+=(fx+bb);
          else {
                y--;
                fy-=aa2;
                p+=(fx+bb-fy);
              }
          putpixel((h+x),(k+y),color);
          putpixel((h+x),(k-y),color);
          putpixel((h-x),(k-y),color);
          putpixel((h-x),(k+y),color);
        }
    p=(int)((bb*(x+0.5)*(x+0.5))+(aa*(y-1)*(y-1))-(aa*bb)+0.5);
    while(y>0)  {
          y--;
          fy-=aa2;
          if(p>=0)
              p+=(aa-fy);
          else {
                x++;
                fx+=bb2;
                p+=(fx+aa-fy);
              }
          putpixel((h+x),(k+y),color);
          putpixel((h+x),(k-y),color);
          putpixel((h-x),(k-y),color);
          putpixel((h-x),(k+y),color);
        }
  }
```