بسم الله الرحمن الرحیم

Computer Graphics

By
DONALD HEARN
M.PAULINE BAKER

گردآورنده و ترجمه : سید مجید خراشادیزاده

تابستان 1386

فصل اول :

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 مقیماً 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.

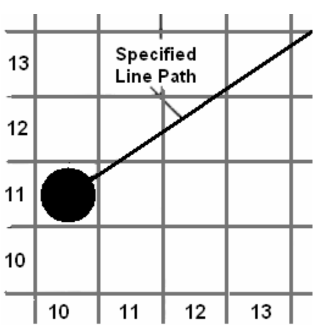
رویه زیر، نحوه اجرای این الگوریتم را نشان میدهد:

```
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);
          }
  }
```

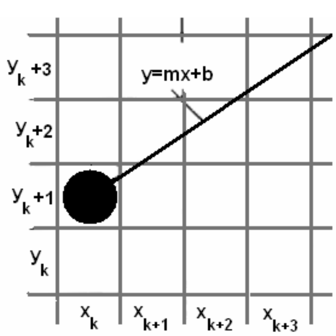
الگوریتم Bresenham :

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.

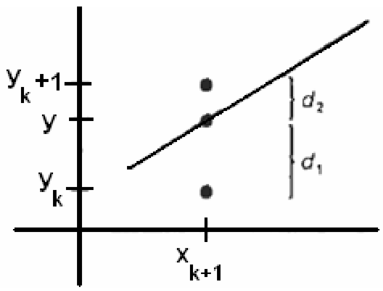


در شکل روبرو بخشی از یک خط مستقیم نشان داده شده است .

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)

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

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;
          }
```

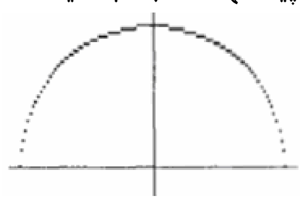
```
    int dx=abs(x2-x1);
    int dy=abs(y2-y1);
    int two_dy=(2*dy);
    int two_dy_dx=(2*(dy-dx));
    int p=((2*dy)-dx);
    int x=x1;
    int y=y1;
    putpixel(x,y,color);
    while(x<x2)
        {
          x++;
          if(p<0)
                p+=two_dy;
          else
                {
 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

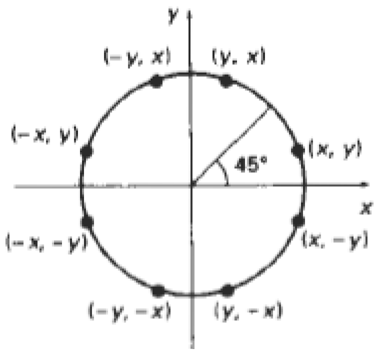
اما این روش خوبی برای تولید دایره نیست، زیرا اولاً در هر مرحله نیاز به انجام محاسبات فراوانی است ثانیاً همانطور که در شکل زیر نشان داده شده است فاصله بین پیکسلها همه جا ثابت نیست.



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^o 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:

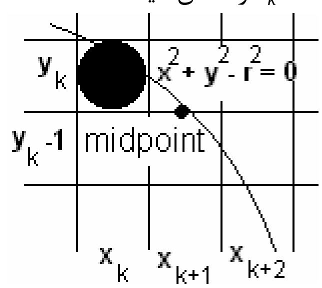
دراین روش ما در هر مرحله نزدیکترین نقطه به مسیر دایره را انتخاب میکنیم. برای یک شعاع داده شده r و یک مرکز (x_c, y_c) ابتدا دایره را به مرکز $(0,0)$ رسم میکنیم سپس نقاط محاسبه شده را با مقدار (x_c, y_c) جمع میکنیم تا دایره به نقطه اصلی منتقل شود.

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.

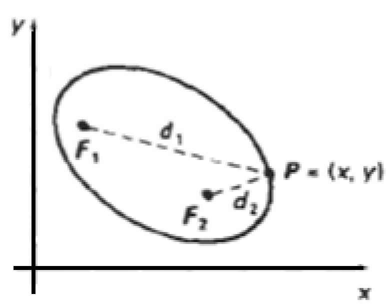
```
void midpoint_circle(const int h,const int k,const int r)
  {
    int color=getcolor( );
    int x=0;
    int y=r;
    int p=(1-r);
    do   {
           putpixel((h+x),(k+y),color);
           putpixel((h+y),(k+x),color);
           putpixel((h+y),(k-x),color);
           putpixel((h+x),(k-y),color);
           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 .



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

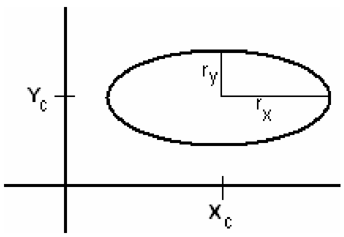
$$F_1 = (x_1, y_1), F_2 = (x_2, y_2)$$

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

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:



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

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:

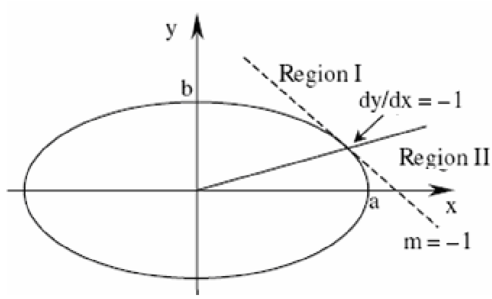
پیاده سازی روش زاویه در C++ بصورت زیر است:

```
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.

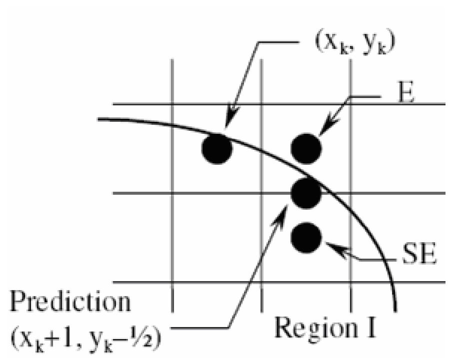


به معادله کلی بیضی توجه کنید:

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

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*:

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

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

calculate the change of ΔP_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* :

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

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

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

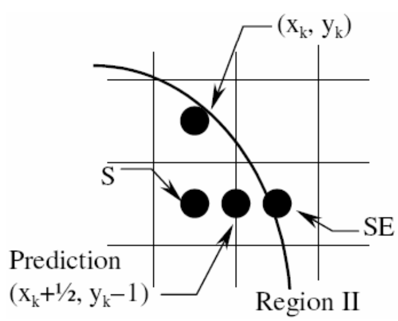
$\Delta^2 P_k^{SE} = \Delta P_{k+1}^{SE} - \Delta P_k^{SE} = 2(a^2+b^2)$

Initial Value :

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

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

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 : $x_{k+1} = x_k + 1$.

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

if $P_k > 0$, *select* S:

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

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

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:

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

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

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

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

Initial Value at region2 :

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

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);
        }
  }
```

فصل دوم :

نمایی کلی از سیستمهای گرافیکی

مباحث این فصل:

- **انواع سیستم های نمایش**
  - **سیستم های Random Scan**
  - **سیستم های Raster Scan**
- **تکنولوژی های نمایش گر ها**
  - **نمایشگر های غیر Flat**
    - **نمایشگر های CRT**
    - **نمایشگر های DVST**
  - **نمایشگرهای Flat**
    - **نمایشگرهای پلاسما**
    - **نمایشگرهای LCD**
    - **نمایشگر های EL**

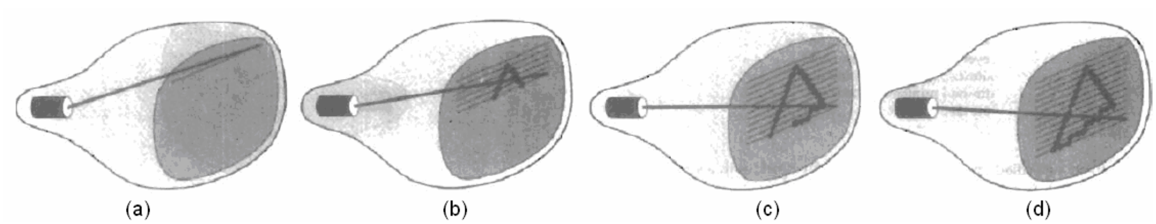
به طور مشخص، اولین و اصلی ترین دستگاه خروجی در یک سیستم گرافیکی نمایشگرهای ویدئویی هستند. عملکرد اکثر نمایشگرهای ویدئویی بر پایه طراحی CRT است، اما هم اکنون تکنولوژی های دیگری نیز وجود دارند که کاربرد فراوانی هم دارند.

دو روش در رسم تصاویر وجود دارد :

- روش ترسیم برداری .
- روش ترسیم راستر .

صفحه نمایش های Raster – Scan

اکثر مانیتورها از سیستم raster- scan استفاده می کنند. در این سیستم پرتو الکترونی کل صفحه نمایش را بصورت افقی و خط به خط از چپ به راست و از بالا به پایین طی میکند. در همان حال که پرتو صفحه نمایش را طی میکند، شدت روشنایی پرتو خاموش روشن میشود تا یک الگو از نقاط روشن ایجاد کند. این الگو در حافظه ای به نام Frame Buffer ذخیره میشود. سپس این الگو بر روی صفحه نمایش نمایش داده میشود. شکل زیر عملکرد سیستم raster – scan را نشان میدهد.

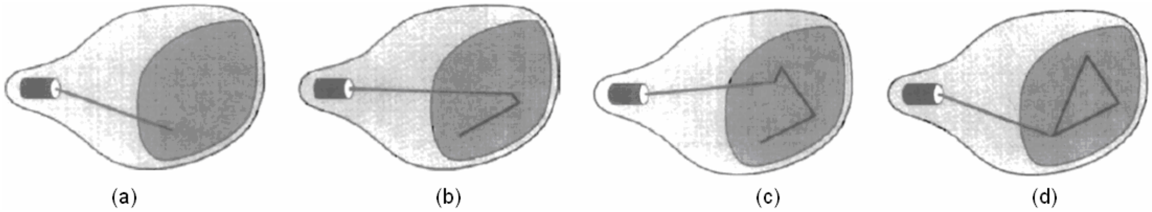


تلویزیونها و چاپگرها از این سیستم استفاده میکنند. هر نقطه روشن روی صفحه نمایش را یک پیکسل میگویند. روشنایی هر پیکسل به سیستم گرافیکی بستگی دارد. در یک سیستم سیاه سفید، تنها از یک بیت برای روشنایی پیکسل استفاده میشود. بیت 1 به معنای روشن بودن و بیت 0 به معنای خاموش بودن پیکسل است. در سیستم های گرافیک بالا معمولا از 24 بیت به بالا استفاده میشود. همچنین تازه سازی با سرعتی حدود 60 تا 80 فرم در ثانیه انجام میشود.

صفحه نمایش های Random – scan :

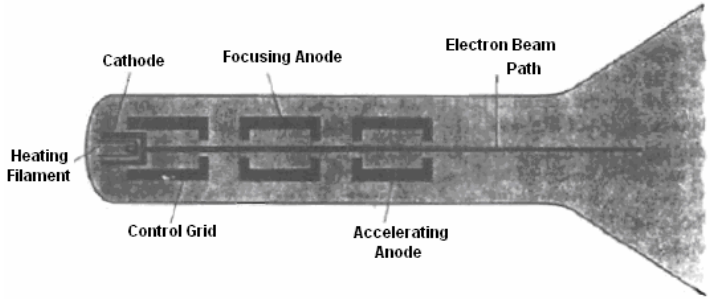
در این صفحه نمایش ها، پرتو الکترونی تمام صفحه نمایش را طی نمیکند. بلکه پرتو تنها به آن قسمت از صفحه که قرار است تصویر در آن رسم شود هدایت میشود.

به این صفحه نمایش ها، صفحه نمایش های برداری Vector scan نیز گفته میشود. قلم رسام از این سیستم استفاده میکند. سرعت تازه سازی بستگی به تعداد خطوطی دارد که قرار است رسم شوند.



نمایشگرهای CRT[1]:

شکل زیر نحوه عملکرد یک **CRT** را نشان میدهد.



یک پرتو الکترونی توسط تفنگ الکترونی منتشر میشود. سپس از سیستم کانونی[2] و سیستم انحراف[3] که پرتو را به یک نقطه معین روی صفحه فسفر هدایت میکند میگذرد.

فسفر در هر کجا که پرتو با آن برخورد کرده باشد یک نقطه کوچک را نورانی خواهد کرد. از آنجاکه پرتو تابیده شده به فسفر به سرعت محو میشود ، متدهایی لازم است تا تصویر را روی صفحه نمایش نگه دارند. یکی از این متدها ارسال پیاپی پرتو الکترون به همان نقطه قبلی است. به این عمل تازه سازی میگویند.

حال قسمتهای مختلف یک نمایشگر **CRT** و وظایف آنها را تک تک تشریح میکنیم:

Heating filament : به کمک این گرم کننده تفنگ الکترونی گرم خواهد شد تا بتواند تولید پرتوهای الکترونی را تسریع بخشد.

Electronic Gun : تولید پرتوهای الکترونی با فرکانس خاصی را بر عهده دارد که این فرکانسمعمولاً به دقت مانیتور وابسته است.

Control Grid : شدت پرتو الکترونی توسط تغییر سطح ولتاژ این قسمت کنترل میشود. یک ولتاژ منفی قوی در **Control Grid** استفاده میشود تا مانع عبور الکترونها از حفره انتهایی **control grid** شود. یک ولتاژ پایین به راحتی میتواند تعداد الکترونهای عبوری را کاهش دهد. از آنجا که میزان نور ساتع شده از صفحه فسفر به تعداد الکترونهای برخوردی به صفحه فسفر بستگی دارد، لذا میتوانیم میزان روشنایی صفحه نمایش را با کنترل سطح ولتاژ **control grid** تنظیم کرد. همهچنین میتوان با اعمال یک ولتاژ مثبت سرعت عبور الکترون را افزایش داد.

Focusing System : این سیستم پرتو نور را مجبور میکند تا در یک نقطه روی صفحه فسفر همگرا شود. عمل متمرکز کردن توسط میدانهای مغناطیسی و الکتریکی انجام میشود.

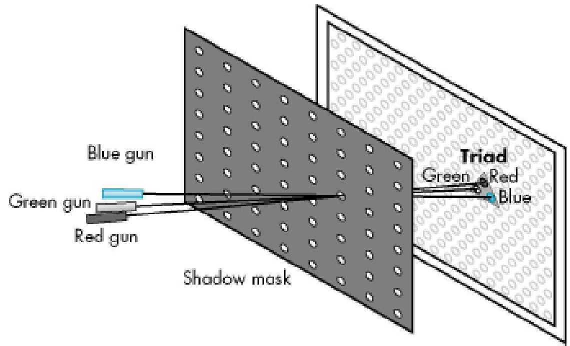
مانیتورهای CRT رنگی:

این مانیتورها تصاویر رنگی را با استفاده از صفحات فسفری که رنگهای مختلف را از خود ساتع میکنند، نشان میدهند. با ترکیب این رنگها میتوان رنگهای متعددی را بدست آورد.

یکی از روشهای تولید مانیتورهای CRT رنگی، استفاده از صفحات Shadow mask است.

---

[1] Cathode Ray Tube
[2] Focusing System
[3] Deflection System



روش Shadow mask بیشتر روی سیستم های raster – scan استفاده میشود. این گونه مانیتور دارای سه صفحه فسفر اند. یکی نور قرمز، دیگری نور آبی و سومی نور سبز را از خود منتشر میکند. همچنین در این روش از سه تفنگ الکترونی استفاده میشود. سه پرتو الکترونی تولید شده بصورت گروهی متمرکز و منحرف میشوند و پس از عبور از صفحه Shadow mask بصورت یک نقطه رنگی کوچک روی صفحه نمایش ظاهر میشوند.

تکنولوژ ِی DVST[4]:

این تکنولوژی مانند تکنولوژی CRT است. با این تفاوت که نیاز به عمل تازه سازی نخواهیم داشت. در این تکنولوژی تصاویر در صفحه به عنوان Charge Distribution ذخیره میشوند . معادل با هر پیکسل یک خازن داریم. در پشت صفحه اصلی صفحه ای است که مانند خازن عمل میکند و قدرت ذخیره سازی ولتاژ را خواهد داشت.

مزیت این تکنولوژی : با توجه به اینکه در این روش نیاز به عمل تازه سازی نیست نیاز به پهنای باند بالا ندارد و تصاویر با دقت بالا میتوانند بدون لرزش روی صفحه نمایش نشان داده شوند.

عیب این تکنولوژی : برای تغییر دادن بخشی از تصویر باید الگوی تصویر دوباره در خازن ذخیره شود که این عمل مدت زمان زیادی طول میکشد.

صفحه نمایشگرهای Flat Panel:

با اینکه اکثر مانیتورهایی که امروزه استفاده میشوند، مانیتورهای CRT هستند اما تکنولوژی های دیگری نیز پدید آمده اند که به زودی جای مانیتورهای CRT را خواهند گرفت.

واژه نمایشگرهای Flat Panel به رده ای از نمایشگرها مربوط میشود که در مقایسه با مانیتورهای CRT دارای حجم، وزن و توان مصرفی کمتری هستند. مهمترین خصیصه این نمایشگرها باریک بودن آنهاست به گونه ای که حتی میتوان آنها را به دیوار آویزان کرد. این مانیتورها بیشتر در ماشین حسابها ، لپتاب ها و ... استفاده میشوند.

نمایشگرهای flat panel را میتوان به دو دسته تقسیم کرد. نمایشگرهای ساتع کننده[5] و غیر ساتع کننده[6].

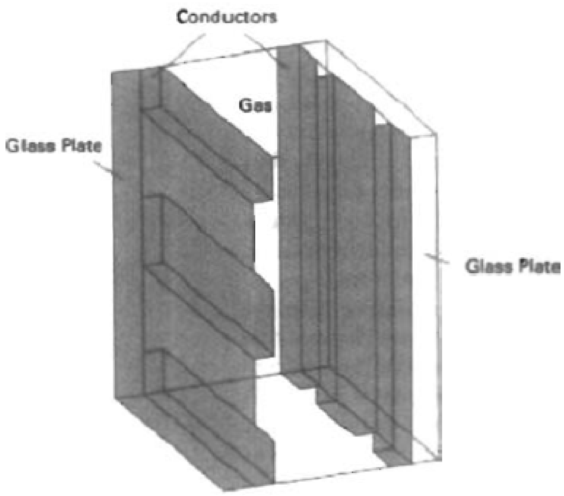
مثالی از نمایشگرهای ساتع کننده عبارتند از : صفحه نمایشگرهای پلاسما و نمایشگر های دیودی. در مقابل صفحه نمایشگرهای کریستال مایع نمونه ای از نمایشگرهای غیرساتع کننده اند.

---

[4] Direct View Storage Tube
[5] Emissive Display
[6] Non-Emissive Display

تکنولوژی پلاسما[7] :

صفحه نمایشگرهای پلاسما که به آنها gas-discharge display نیز گفته میشود بصورت زیر ساخته میشوند. ابتدا محوطه بین دو صفحه شیشه ای با مخلوطی از گازها که معمولا ً حاوی نئون است پر میشود.
یک سری از نوارهای هادی عمودی روی یکی از صفحات قرار میگیرد. روی صفحه دیگر نوارهای هادی افقی قرار میگیرد.
شکل زیر ساختمان این نوع نمایشگر را نشان میدهد.



اعمال یک ولتاژ گرمایشی به هر جفت از هدایت کننده های افقی و عمودی موجب میشود گاز موجود در تقاطع دو نوار هادی به ذرات پلاسمای تابان از یونها و الکترونها تبدیل شود. الگوی تصویر در Refresh Buffer ذخیره میشود و سپس ولتاژ گرمایشی در هر ثانیه 60 بار اعمال میشود.

مزیت ها : زاویه دید بزرگ ، روشنایی نسبتا ً خوب ، مناسب برای تصاویر بزرگ
معایب : هزینه بالا ، روشنایی کمتر از CRT ، پیکسل های بزرگ در حدود 1 میلیمتر

تکنولوژی کریستال مایع:

اینگونه صفحه نمایشها بیشتر در دستگاه های کوچک مانند ماشین حسابها ، کامپیوترهای لپتاپ و … استفاده میشوند. کلمه Liquid Crystal به این حقیقت برمیگردد که این نمایشگرها حاوی مولکولهای بلورینی هستند که مانند مولکولهای مایع معلقند. این نمایشگر ها دارای 6 لایه هستند. اولین لایه کریستال خواهد بود و آخرین لایه Reflector که نور را برگشت میدهد. خاصیت مولکول های بلورین در این است که در جهت قطبی شدن یا پلاریته شدن نور مرتب میشوند . هنگامی که نور از اولین صفحه عبور میکند به طور عمودی پلاریته میشود. هنگامی که از کریستال عبور میکنند 90 درجه چرخش و به صورت افقی در خواهند آمد. کریستالها اگر در میدان الکتریکی قرار گیرند، در این جهت مرتب خواهند شد و نوری که از آنها عبور میکند میتواند تغییری در جهت پلاریته شدن ایجاد نکند. اگر جهت پلاریته شدن عوض نشود نقطه مربوطه تاریک دیده خواهد شد. در واقع نور در صفحه بعدی جذب میگردد، اما اگر جهت پلاریته شود تغییر کند، نور به آخرین صفحه برخورد خواهد کرد و برگشت داده میشود که در این حالت نقطه روشن دیده خواهد شد. در مانیتور LCD نیز نیاز به عمل refresh داریم، زیرا جهت عوض شدن پلاریته ها به صورت موقت میباشد. در بعضی از مانیتور های LCD برای هر نقطه یک ترانزیستور وجود دارد تا بتوان تغییر کریستال را سریعتر انجام داد.

تکنولوژی EL :

این تکنولوژی ترکیبی از پلاسما و DVST میباشد. تخت بودن آن از پلاسما و Refresh کردن آن از DVST .

[7] Plasma Display Panel

فصل سوم :

پر کردن و برش اشکال

مباحث این فصل:

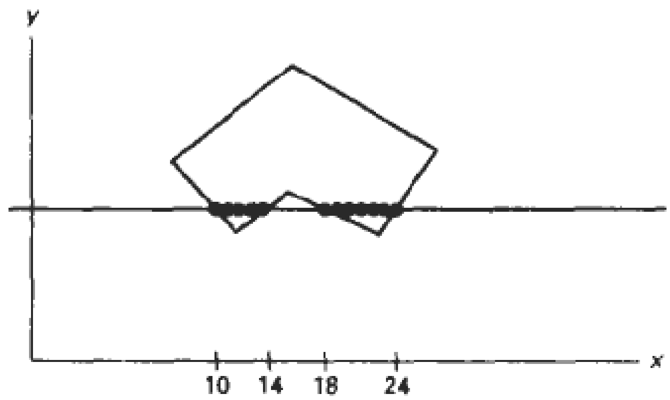
- پر کردن اشکال
- رسم اشکال با قطر بیش از یک پیکسل
  - **Replaceing pixel**
  - **Moving Pen**
  - **Filling Area Between Bound Areas**
  - **Approximately By Tick Poly Line**
- برش اشکال
  - **الگوریتم Cohen-suterland**
  - **الگوریتم Cyrus-Beck**

پر کردن اشکال:

دو رهیافت اساسی برای پرکردن نواحی در سیستم های راستر وجود دارد. روش اول بر پایه اسکن خطوط و روش دوم بر پایه معادلات ریاضی میباشد. در این بخش ما به توضیح روش اول میپردازیم.

الگوریتم پر کردن چندضلعی نامنتظم با استفاده از روش اسکن خطوط:

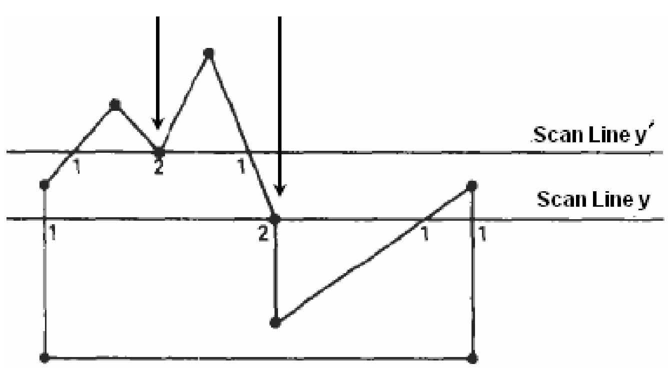
شکل زیر نحوه عملکرد این الگوریتم را برای رنگامیزی یک چندضلعی نامنتظم نشان میدهد.



برای هر خط اسکن شده که از چندضلعی عبور کرده است، الگوریتم نقاط تقاتع خط اسکن شده با چندضلعی را بدست می آورد. سپس این نقاط تقاطع از چپ به راست مرتب میشوند و در نهایت مکانهای بین هر جفت از این نقاط در میانگیر قاب با رنگ مشخصی مقداردهی میشوند.

در همین مثال قبلی چهار نقطه تقاطع توسط الگوریتم تعیین میشوند و پیکسلهای داخلی از $x=10$ تا $x=14$ و از $x=18$ تا $x=24$ در میانگیر قاب با رنگ مشخص مقداردهی میشوند.

اما دقت کنید که اگر نقاط تقاطع خط اسکن با چندضلعی، رئوس چندضلعی باشد در این صورت نیاز به یک سری دستکاری نقاط تقاطع داریم. برای مثال شکل زیر را در نظر بگیرید. در این صورت چگونه میتوان ناحیه داخل چندضلعی را تشخیص داد.



در اینجا خط اسکن y هیچ مشکلی همراه ندارد اما خط اسکن y' در نقطه 2 دارای مشکل است. برای حل مشکل میتوان بدینگونه عمل کرد: برای هر نقطه تقاطع، قبل و بعد نقطه را نگاه میکنیم. اگر به طور یکنواخت نزولی یا صعودی

بود که نقطه را به عنوان نقطه تقاطع در نظر میگیریم و در غیراین صورت نقطه را به صورت دو نقطه تقاطع در یک مکان در نظر میگیریم.
برای پیاده سازی این روش نیاز به محاسبات فراوان میباشد. ولی مزیت آن نسبت به الگوریتم معادلات ریاضی توانایی در پر کردن اشکال چندضلعی نامنتظم است.

رسم اشکال با قطر بیش از یک پیکسل:

جهت رسم اشکال با قطر با بیش از یک پیکسل چهار روش وجود دارد:

روش اول Replaceing pixel :

در این روش فرض میشود که قطر شکل یک پیکسل است و شکل رسم میشود. به کمک هر پیکسل میتوان پیکسل های اطراف آن را تعیین نمود. مشکلات این روش عبارتند از :

1) ضخامت یکسانی روی خطوط افقی، عمودی و مایل وجود ندارد
2) یکنواختی روی مرز اشیاء وجود ندارد.
3) برای اشکال غیر از خط محاسبات زیادی باید انجام شود.

روش دوم Moving Pen :

در این روش یک قلم به شکل کادر مستطیل شکل در نظر گرفته میشود که مرکز یا گوشه کادر مستطیل شکل روی یک پیکسل از خط مرزی حرکت میکند. برای رسم خط این روش بهترین نتیجه را خواهد داد.

روش سوم Filling Area Between Bound Areas :

در این روش ابتدا مرز درونی و بیرونی اشیاء تعیین شده و سپس فاصله بین این دو مرز پر خواهد شد. در این روش باید پیکسل های مرز بیرونی را با محاسبات ریاضی بدست آوریم. همچنین بدست آوردن نقاط بین مرزها نیز مشکل است.

روش چهارم Approximately By Tick Poly Line :

با استفاده از تعدادی خط پیوسته که معمولاً دارای طولی کوتاه می باشند، شکل را تقریب میزنند. این روش کارایی چندانی ندارد.

عملیات برش[8] :

به طور کلی هر رویه، که بخشی از تصویر که داخل یا خارج یک محدوده معین است را مشخص میکند، الگوریتم برش[9] نامیده میشود. ناحیه ای که عمل برش بر روی آن انجام میشود پنجره برش[10] نامیده میشود.
عملیات برش معمولاً برای انتخاب بخشی از تصویر، خوش نماسازی خطوط و یا مرزهای اشیاء، و همچنین عملیات مربوط به نقاشی و ترسیم (مانند کپی، انتقال و یا حذف ناحیه ای مشخص) کاربرد دارد.

---
[8] Clipping Operation
[9] Clipping Algorithm
[10] Clipping Window

بسته به نوع کاربرد، پنجره برش میتواند یک چندضلعی نامتقارن و یا یک منحنی بسته باشد. در اینجا تنها برش را با پنجره برش مستطیل شکل انجام خواهیم داد.
در این فصل ما برش نقطه و برش خط را بررسی میکنیم. روالهای برش خط و برش ناحیه از اجزاء استاندارد بسته های گرافیکی هستند اما همه این بسته ها دارای روال مخصوص برای برش منحنی و اشکال کانونی(مثل دایره و بیضی) نیستند و ممکن است این اشکال را بصورت خط مستقیم در نظر بگیرند و سپس با استفاده از روال های برش خط و ناحیه به هدف خود برسند.

برش نقطه (Point Clipping)

فرض کنید پنجره برش ما یک مستطیل در موقعیت استاندارد باشد. ما یک نقطه را برای نمایش ذخیره میکنیم اگر معادلات زیر بر قرار باشد:

$$\begin{cases} xw_{\min} \le x \le xw_{\max} \\ yw_{\min} \le y \le yw_{\max} \end{cases}$$

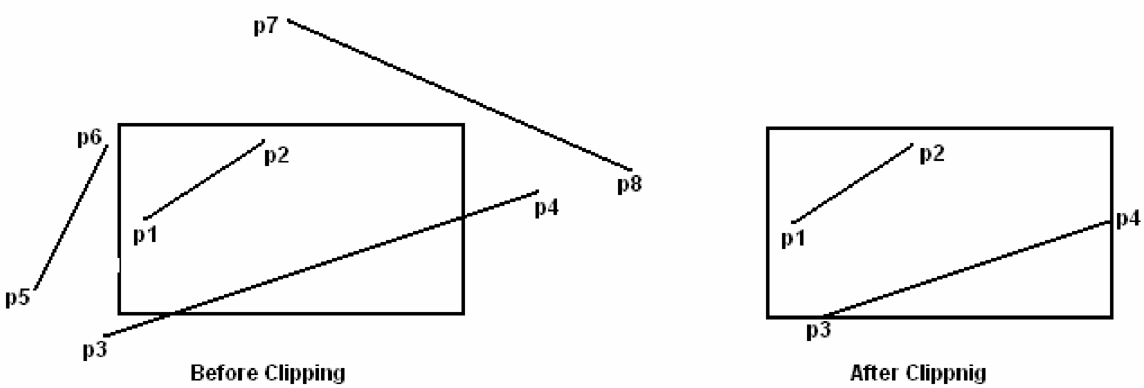
که در آن رئوس پنجره برش مستطیل شکل عبارتند از :

$$(xw_{\min}, xw_{\max}, yw_{\min}, yw_{max})$$

در صورتیکه حتی یکی از چهار معادله فوق نادرست باشد نقطه حذف خواهد شد(برای نمایش ذخیره نمیشود).
با اینکه رویه برش نقطه کمتر از رویه های برش خط و ناحیه کاربرد دارد، با این حال بعضی از کاربردها به این رویه نیازمندند. (مانند صحنه های مربوط به انفجار)

برش خط (Line Clipping)

شکل زیر رابطه بین موقعیت خطوط و موقعیت پنجره برش را برای عملیات برش نشان میدهد.

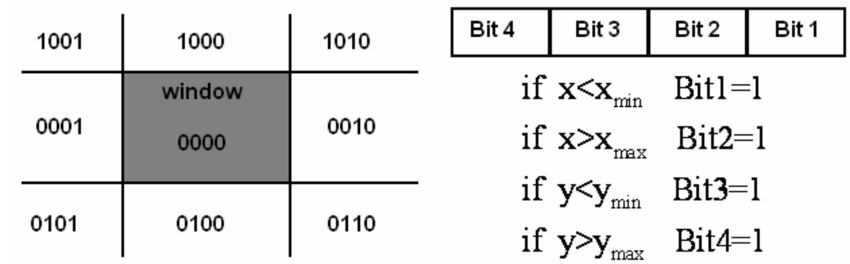


ابتدا بررسی میکنیم که آیا خط به طور کامل داخل پنجره برش قرار دارد یا نه. اگر قرار نداشت بررسی میکنیم آیا خط کاملا خارج پنجره برش قرار دارد یا نه. در نهایت اگر خط کاملا در داخل یا خارج ناحیه برش نبود با انجام یک سری معادلات تشخیص میدهیم کدام بخش از خط، داخل و کدام بخش خارج پنجره برش قرار دارد. در نهایت هدف ما بدست آوردن الگوریتمی کارامد برای حذف خطوط خارج از ناحیه با انجام کمترین محاسبات است.

الگوریتم های کارامدی برای برش خط وجود دارد که ما در اینجا مهمترین آنها را بررسی میکنیم. بعضی از این الگوریتمها مختص اشکال دو بعدی می باشند و بعضی دیگر به سادگی به اشکال سه بعدی تعمیم می یابند.

الگوریتم Cohen – Sutherland:

این الگوریتم از الگوریتمهای قدیمی است که بدلیل تعداد کم محاسبات از سرعت اجرای بالایی برخوردار است. در این روش به هر یک از دو سر انتهایی خط یک کد دودویی چهار رقمی نسبت داده میشود که این کد موقعیت خط را نسبت به مرزهای مستطیل برش مشخص میکند. ناحیه ها به صورت زیر کد گذاری میشوند.



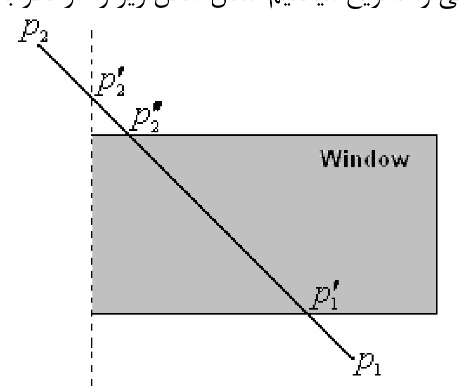
با این کد گذاری میتوان براحتی وضعیت خط را نسبت به مستطیل برش تشخیص داد.

الف) خط داخل مستطیل برش است: زمانیکه کد مربوط به انتهای هر دو سر خط برابر 0000 باشد. این خطوط بدون انجام محاسبات ریاضی پذیرفته میشوند. برای تست این شرط میتوان کد مربوط به دو انتهای خط را با هم OR کرد اگر نتیجه 0000 بود خطاما ً داخل ناحیه برش است.

ب) خط خارج ناحیه برش است : زمانیکه حداقل یک بیت 1 در یک مکان مشخص در کد هر دو انتهای خط یافت شود. این خطوط بدون انجام محاسبات ریاضی حذف میشوند. برای تست این شرط میتوان کد مربوط به دو انتهای خط را با هم AND کرد اگر نتیجه چیزی غیر از 0000 بود خطاما ً خارج ناحیه برش است.

ج) بخشی از خط داخل و بخشی از خط خارج ناحیه برش است: هر حالتی به جز حالات بالا. در این حالت برای اینکه موقعیت خط را نسبت به مستطیل تشخیص دهیم نیاز به یک سری محاسبات ریاضی داریم.

برای درک عملکرد این الگوریتم، مثالی را تشریح میکنیم. مثال شکل زیر را در نظر بگیرید:



با کدگذاری انتهای خط مشخص میشود خط به طور کامل داخل یا خارج مستطیل برش قرار ندارد و در شرایط اول و دوم صدق نمیکند، برای برش خط نیاز به یک سری محاسبات ریاضی داریم.

برای پیدا کردن نقاط تقاطع میتوان از فرمول شیب خط استفاده کرد. برای خطی با نقاط انتهایی (x_1, y_1) و (x_2, y_2) فرمول خط به صورت زیر است:

$$y = y_1 + m(x - x_1)$$

مقدار m که بصورت زیر محاسبه میشود:

$$m = (y_2 - y_1)/(x_2 - x_1)$$

هر کجا مقدار x برابر با یکی از مقادیر $xw_{\min}$ یا $xw_{\max}$ قرار گیرد، آنگاه مقدار متناظر با y بدست میآید. این شرط ها در زیر آورده شده است. برای نقاط انتهایی شرایط زیر را بررسی میکنیم.

$$x_p \geq xw_{\min} \quad \text{else} \quad P' : y = m.xw_{\min} + b$$
$$x_p \leq xw_{\max} \quad \text{else} \quad P' : y = m.xw_{\max} + b$$
$$y_p \geq yw_{\min} \quad \text{else} \quad P' : yw_{\min} = m.x + b$$
$$y_p \leq yw_{\max} \quad \text{else} \quad P' : yw_{\max} = m.x + b$$

ابتدا p_1 را با چهار ضلع مستطیل برش مقایسه میکنیم. نتیجه میگیریم که p_1 در پایین مستطیل قرار دارد. بنابراین p_1' را که نقطه تقاطع خط با مستطیل برش است را پیدا میکنیم. این کار برای نقطه p_2 نیز انجام میشود.مزیت این الگوریتم محاسبات کم و عیب آن کار کردن تنها با ناحیه برش مستطیل شکل است.

الگوریتم Cyrus – Beck :

این الگوریتم یکی از الگوریتم های سریع برای برش خط است که بر پایه معادلات پارامتری خط و بوسیله Cyrus و Beck توسعه یافته است.

معادله خط در دستگاه دکارتی به سه صورت نوشته میشود:

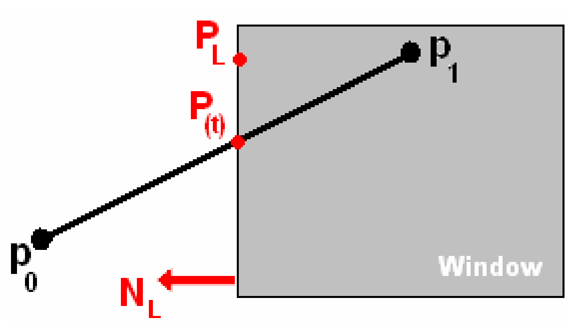
الف) معادله خط بر اساس شیب و عرض از مبدأ آن: $y = mx + b$

ب) معادله ضمنی خط: $Ax + By + C = 0$

ج) معادله پارامتری خط: در این فرم خط بر اساس دو نقطه ابتدایی و انتهایی آن نوشته میشود. $P(t) = P_0 + (P_1 - P_0)t$ که در آن P خود یک بردار به صورت $[x, y]$ است. بنابراین میتوان معادله خط را به صورت دو معادله زیر هم نوشت:

$$\begin{cases} x = x_0 + (x_1 - x_0)t \\ y = y_0 + (y_1 - y_0)t \end{cases}$$

برای درک این الگوریتم مراحل زیر را گام به گام انجام میدهیم:



1) معادله پارامتری خط را مینویسیم.
2) برای هر لبه (ضلع) مستطیل برش دو پارامتر زیر را بدست میآوریم:
   a. بردار عمود بر لبه N_L که میتواند یکی از مقادیر زیر را اختیار کند:
   $(1,0),(0,1),(-1,0),(0,-1)$
   b. یک نقطه بر روی لبه P_L

3) با توجه به اینکه بردار های N_L و $P(t)-P_L$ بر هم عمودند داریم :
$$N_L \bullet (P(t)-P_L)=0$$
4) مقدار $P(t)$ را در فرمول بالا جایگزین میکنیم و سپس مقدار t را از آن بصورت زیر بدست می آوریم:
$$t=\frac{N_L \bullet (P_0-P_L)}{-N_L \bullet (P_1-P_0)}$$
5) مقدار t را از رابطه بالا برای تمام نقاط مشترک خط و لبه های مستطیل برش بدست می آوریم.
6) بر اساس مقدار t داریم:
   a. تمام $t<0$ یا $t>1$ رد میشوند.
   b. نقاط اشتراکی باقیمانده را رده بندی میکنیم.
      i. نقاط مستعد برای داخل بودن(PE) $N_L\left[P_1-P_0\right]<0$
      ii. نقاط مستعد برای خارج بودن(PL) $N_L\left[P_1-P_0\right]>0$
7) در این مرحله کمترین مقدار PL و بیشترین مقدار PE را پیدا و مقدار tآنها را در معادله پارامتری خط جایگزین کرده و خط بین نقاط حاصل را به عنوان برشی از خط اولیه رسم میکنیم.

فصل چهارم :

تبدیلات هندسی دو بعدی
و
سه بعدی

مباحث این فصل:

- **تبدیلات اولیه**
  - **انتقال**
  - **دوران**
  - **تغییر مقیاس**
- **ماتریسهای همگن**
- **تبدیلات مرکب**
- **تبدیلات سه بعدی**
- **سایر تبدیلات**
  - **تبدیلات** Shear
  - **تبدیلات** Affine

تبدیلات هندسی عبارتند از تغییر در موقعیت یا شکل و یا اندازه اشکال و تصاویر. تبدیلات هندسی اولیه عبارتند از : انتقال، دوران و تغییر مقیاس. سایر تبدیلاتی که به اشکال اعمال میشوند عبارتند از : انعکاس و تبدیلات Shear .

تبدیلات اولیه :

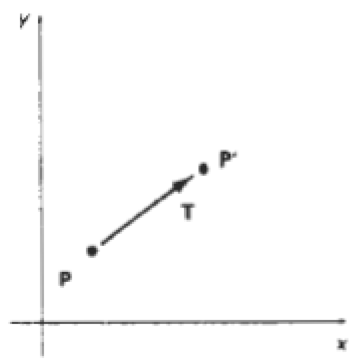
در اینجا به تبدیلات هندسی پایه میپردازیم و در بخش بعدی بیان میکنیم چگونه این تبدیلات را به صورت بیان ریاضی از ماتریسهای ساده تر پیاده سازی کنیم تا بتوان ترکیبی از این تبدیلات را محاسبه کرد.

انتقال :

انتقال یعنی حرکت دادن یک شئ در امتداد یک خط راست از نقطه ای به نقطه دیگر. برای انتقال یک نقطه در مختصات دو بعدی کافیست مختصات بردار انتقال $\left(t_x, t_y\right)$ را به مختصات آن نقطه اضافه کنیم.

$$x' = x + t_x \quad , \quad y' = y + t_y$$

شکل زیر انتقال نقطه توسط بردار انتقال نشان میدهد :



معادلات بالا را میتوان به صورت ماتریسی نیز بیان کرد:

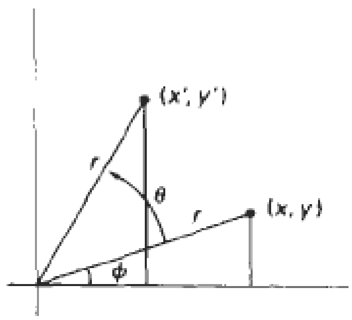
$$p = \begin{bmatrix} x \\ y \end{bmatrix} \ , \ p' = \begin{bmatrix} x' \\ y' \end{bmatrix} \ , \ t = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$p' = p + t$$

واضح است که انتقال نوعی تبدیل هندسی سخت است یعنی شکل را تغییر نمیدهد، تنها مکان هرنقطه از تصویر را توسط یک مقدار ثابت تغییر میدهد.

دوران :

دوران در مختصات دو بعدی یعنی انتقال شئ روی یک مسیر دایره ای در صفحه xy . برای انجام عمل دوران به دو پارامتر نیاز داریم: اول زاویه چرخش و دوم نقطه محوری. ابتدا معادلات دوران یک نقطه حول مبدأ مختصات را بدست می آوریم. شکل زیر دوران نقطه حول مبدأ مختصات را نشان میدهد.



$$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta$$

همچنین مختصات اولیه نقطه در مختصات قطبی برابر است با :

$$x = r\cos\phi \quad , \quad y = r\sin\phi$$

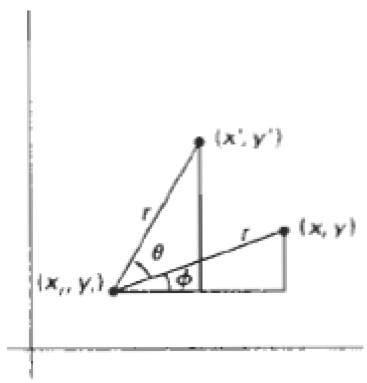
با جایگزینی معادلات بالا داریم:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

مانند قبل میتوان این معادلات را بر حسب ماتریسها بدست آورد:

$$p = \begin{bmatrix} x \\ y \end{bmatrix} \ , \ p' = \begin{bmatrix} x' \\ y' \end{bmatrix} \ , \ \mathrm{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$
$$p' = R.p$$

برای دوران حول نقطه غیرمبدأ از فرمول های زیر استفاده میکنیم:



$$x' = x_1 + (x-x_1)\cos\theta - (y-y_1)\sin\theta$$
$$y' = y_1 + (x-x_1)\sin\theta + (y-y_1)\cos\theta$$

نکته 1 : جرخش در جهت عکس حرکت عقربه های ساعت است .

نکته 2 : واضح است که دوران یک تبدیل سخت است..

تغییر مقیاس:

تغییر مقیاس تبدیلی است که اندازه تصویر را تغییر میدهد. برای تغییر اندازه از ضریب مقیاس استفاده میشود.

$$x' = x \cdot s_x \quad , \quad y' = y \cdot s_y$$

که در آن s_x تغییر اندازه روی محور x و s_y تغییر اندازه روی محور y میباشد. این معادلات را میتوان به صورت معادلات زیر نوشت:

$$p = \begin{bmatrix} x \\ y \end{bmatrix} \ , \ p' = \begin{bmatrix} x' \\ y' \end{bmatrix} \ , \ \mathrm{S} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \qquad \Rightarrow p' = S.p$$

میتوان هر مقدار مثبتی را به ضریب مقیاس نسبت داد. مقادیر کمتر از یک موجب کوچک شدن و مقادیر بیشتر از یک موجب بزرگتر شدن جسم می شود. اگر به هر دو مؤلفه ی ضریب مقیاس، مقدار یک داده شود، در این صورت شکل تغییر اندازه نخواهد داد.

نمایش ماتریسی و مختصات همگن:

در برنامه های گرافیکی معمولاً از ترکیبی از تبدیلات متوالی استفاده میشود. دراین بخش ما فرمولهای ماتریسی بخش قبل را دوباره بازنویسی میکنیم تا فرایند چند تبدیل متوالی را راحتتر انجام دهیم. در بخش پیش دیدیم که هر یک از تبدیلات اولیه را میتوان به صورت زیر بیان کرد :

$$P' = M_1.P + M_2$$

که در آن P مختصات نقطه اولیه، P' مختصات نقطه نهایی، M_1 یک ماتریس دو در دو که میتواند یک ماتریس ضریب مقیاس و یا یک ماتریس دوران باشد و در نهایت M_2 برابر با یک ماتریس یک در دو است که یک ماتریس انتقال است.
اگر به جای استفاده از ماتریس دو در دو از یک ماتریس سه در سه استفاده کنیم، میتوانیم تمام تبدیلات را بصورت ضرب بیان کنیم. برای این منظور به جای استفاده از مختصات دوبعدی (x, y) از مختصات سه بعدی $(x, y, 1)$ استفاده میکنیم. استفاده از مختصات سه بعدی این امکان را میدهد تا فرمولهای تبدیلات را دوباره و تنها بر اساس ضرب بیان کنیم. حال دوباره فرمولهای تبدیلات هندسی را بر اساس ضرب ماتریسهای سه در سه بیان میکنیم:

انتقال :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad , \quad P' = T(t_x, t_y) \cdot P$$

دوران:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad , \quad P' = R(\theta) \cdot P$$

تغییر مقیاس:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad , \quad P' = S(s_x, s_y) \cdot P$$

تبدیلات مرکب

با استفاده از نمایش ماتریسی که در بخش قبل ارائه شد، میتوان برای هر ترکیبی از تبدیلات یک ماتریس تبدیل مرکب بنویسیم. ماتریس حاصل معمولاً ماتریس مرکب یا ماتریس الحاق می نامند. برای بدست آوردن ماتریس تبدیل مرکب، ماتریسهای تبدیل را از آخر به اول در هم ضرب میکنیم.

اگر دو تبدیل انتقال T_1 و T_2 به یک نقطه اعمال شود، ماتریس تبدیل مرکب به صورت زیر بدست می آید.

$$\begin{bmatrix}1 & 0 & t_{x2}\\ 0 & 1 & t_{y2}\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}1 & 0 & t_{x1}\\ 0 & 1 & t_{y1}\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}1 & 0 & t_{x1}+t_{x2}\\ 0 & 1 & t_{y1}+t_{y2}\\ 0 & 0 & 1\end{bmatrix}$$

به همین ترتیب برای دو دوران متوالی $R(\theta_1)$ و $R(\theta_2)$ داریم :

$$R_{total=}R(\theta_2).R(\theta_1)=R\left(\theta_1+\theta_2\right)$$

و در نهایت برای دو تغییر مقیاس متوالی با ضریب S_1 و S_2 داریم :

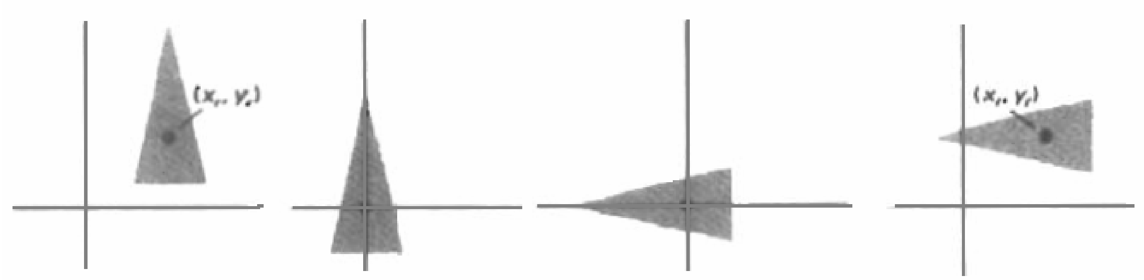
$$\begin{bmatrix}s_{x2} & 0 & 0\\ 0 & s_{y2} & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}s_{x1} & 0 & 0\\ 0 & s_{y1} & 0\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}s_{x1}.s_{x2} & 0 & 0\\ 0 & s_{y1}.s_{y2} & 0\\ 0 & 0 & 1\end{bmatrix}$$

دوران حول نقطه دلخواه :

برای دوران حول نقطه دلخواه مراحل زیر را دنبال میکنیم:

1- شکل را همراه با نقطه محوری به مبدأ انتقال میدهیم.
2- شکل را حول مبدأ مختصات دوران میدهیم .
3- شکل را به مکان اولیه خود باز میگردانیم.

این انتقالات در شکل زیر نشان داده شده است(از چپ به راست).



شئ در مکان اولیه خود و نقطه محوری | نقطه محوری همراه با شئ به مبدأ منتقل شده است | دوران حول مبدأ | انتقال نقطه محوری و شئ به مکان اولیه

بنابراین ابتدا یک انتقال در امتداد بردار $(-x_c,-y_c)$ سپس یک دوران به اندازه θ و در نهایت یک انتقال دیگر در امتداد بردار (x_c,y_c). لذا ماتریس ترکیب به صورت زیر بدست می آید.

$$\begin{bmatrix}1 & 0 & x_r\\0 & 1 & y_r\\0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}\cos\theta & -\sin\theta & 0\\\sin\theta & \cos\theta & 0\\0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}1 & 0 & -x_r\\0 & 1 & -y_r\\0 & 0 & 1\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta & x_r(1-\cos\theta)+y_r\sin\theta\\\sin\theta & \cos\theta & y_r(1-\cos\theta)-x_r\sin\theta\\0 & 0 & 1\end{bmatrix}$$

برای عمل تغییر مقیاس نیز کافیست که نقطه را به مبدأ مختصات انتقال و سپس عمل تغییر مقیاس را انجام دهیم.

تبدیلات سه بعدی :

انتقال : در مختصات سه بعدی، یک نقطه از موقعیت $P(x,y,z)$ به موقعیت $P'(x',y',z')$ توسط ماتریس زیر انتقال می یابد.

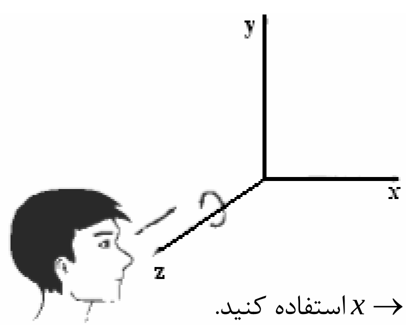
$$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & t_x\\0 & 1 & 0 & t_y\\0 & 0 & 1 & t_z\\0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\1\end{bmatrix}\Rightarrow\begin{cases}x=x+t_x\\y=y+t_y\\z=z+t_z\end{cases}$$

تغییر اندازه : ماتریس تغییر اندازه را میتوان به صورت زیر نوشت :

$$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=\begin{bmatrix}S_x & 0 & 0 & 0\\0 & S_y & 0 & 0\\0 & 0 & S_z & 0\\0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\1\end{bmatrix}\Rightarrow\begin{cases}x=x.S_x\\y=y.S_y\\z=z.S_z\end{cases}$$

دوران : برای تولید یک تبدیل دوران باید محور دوران و زاویه چرخش را مشخص کنیم. در سه بعدی دوران را میتوان حول هر خط موجود در فضا انجام داد اما در اینجا ما تنها دوران حول محورهای مختصات را بررسی می کنیم.

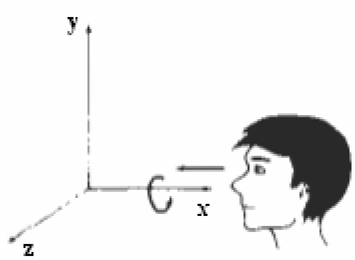
دوران حول محور z :



$$\begin{cases}x'=x\cos\theta-y\sin\theta\\y'=x\sin\theta+y\cos\theta\\z'=z\end{cases}\Rightarrow\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\\sin\theta & \cos\theta & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

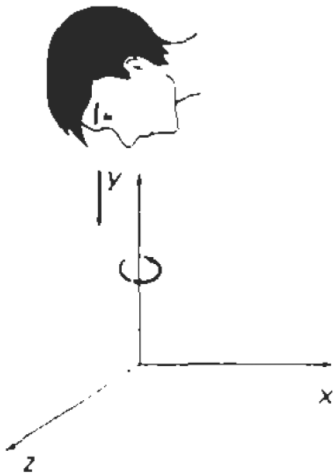
برای بدست آوردن فرمول دوران حول سایر محورها از جایگذاری $x\to y\to z\to x$ استفاده کنید.

دوران حول محور x :



$$\begin{cases}x'=x\\y'=y\cos\theta-z\sin\theta\\z'=y\sin\theta+z\cos\theta\end{cases}\Rightarrow\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & 0\\0 & \cos\theta & -\sin\theta & 0\\0 & \sin\theta & \cos\theta & 0\\0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

دوران حول محورy :

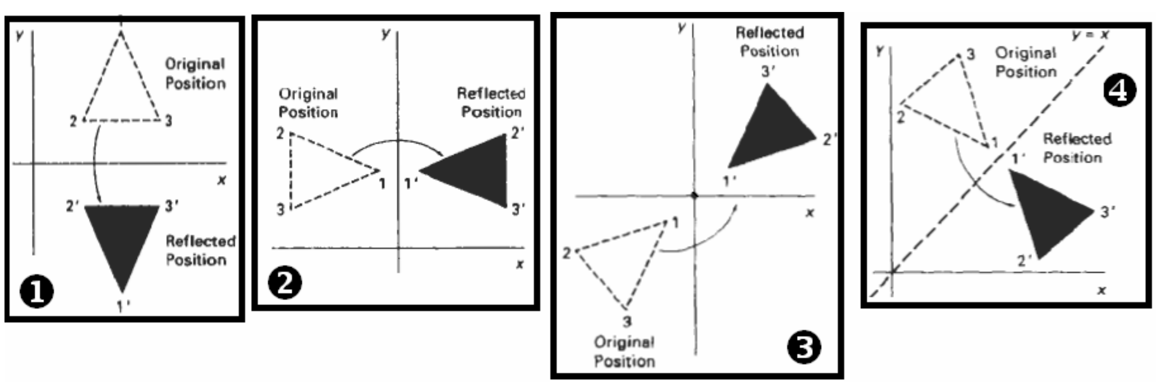


$$\begin{cases} x' = z\cos\theta - x\cos\theta \\ y' = y \\ z' = z\cos\theta - x\sin\theta \end{cases} \Rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

سایر تبدیلات :

علاوه بر تبدیلات اولیه که بیان شدند، تبدیلات مفید دیگری نیز وجود دارند که مهمترین آنها عبارتند از : بازتاب و تبدیلات Shear.

برای عمل بازتاب یک محور بازتاب نیاز داریم . در دو بعدی این محور میتواند هر خطی در صفحه مختصات باشد. ما در این جا بازتاب نسبت به محورهای اصلی و محورهای فرعی بیان میکنیم.

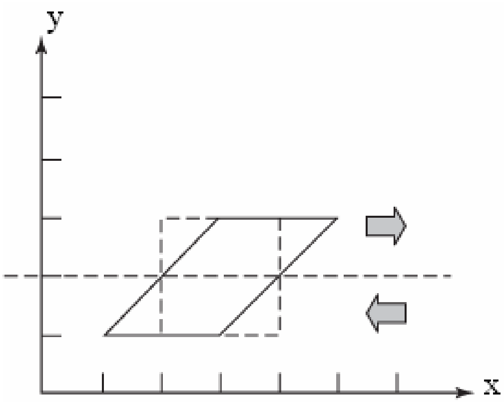


معادلات مربوط به بازتاب اشکال بالا عبارتند از :

$$❶\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad ❷\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad ❸\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad ❹\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

تبدیلات Shear :

تبدیلات Shear تصویر را در جهتهای مشخص شیفت میدهند. در واقع این تبدیلات نقطه را به نسبت فاصله از یک خط و موازی با خط حرکت میدهند. نقاط واقع بر روی خط شیفت داده نمیشود و نقاطی که در طرف مقابل اند در جهت مخالف شیفت میابند. تبدیلات Shear شئ را دگرگون میکنند اما با این حال موازی بودن خطوط را حفظ میکنند. شکل روبرو یک نمونه از این تبدیل را نشان میدهد که با مقیاس 1 و نسبت به خط $y = 2$ انجام شده است.



معمولا ً شیفت Shear در جهت محور x یا y انجام میشود .

شیفت Shear در جهت محور x : ماتریس زیر نقاط را با مقیاس sh_x و به نسبت فاصله از خط $y = y_{ref}$ و موازی با آن شیفت میدهد.

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{cases} x' = x + sh_x(y - y_{ref}) \\ y' = y \end{cases}$$

شیفت Shear در جهت محور y : ماتریس زیر نقاط را با مقیاس sh_y و به نسبت فاصله از خط $x = x_{ref}$ و موازی با آن شیفت میدهد.

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{cases} x' = x \\ y' = y + sh_y(x - x_{ref}) \end{cases}$$

تبدیلات Affine :

تبدیلاتی هستند که در آنها موازی بودن خطوط حفظ میشود ولی طول خطوط تغییر نمی یابند. معروفترین این تبدیلات عبارتند از : انتقال، دوران و تغییر اندازه. برای این سه تبدیل زاویه بین خطوط بعد از تبدیل تغییر نخواهد کرد.

سؤال : چه رابطه ای بین دوران با زاویه $\theta = n\pi$ و عمل تغییر مقیاس در حالت دوبعدی وجود دارد ؟ با استفاده از روابط جبری ارتباط مورد نظر را اثبات کنید.

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{\theta=n\pi, n=2k} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{cases} s_x = 1 \\ s_y = 1 \end{cases}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{\theta=n\pi, n=2k+1} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{cases} s_x = -1 \\ s_y = -1 \end{cases}$$

از حالت اول نتیجه میگیریم که دوران با $2k\pi$ هیچ تغییری در شکل ندارد.

در حالت دوم نمیتوان رابطه ای بین تغییر مقیاس و دوران پیدا نمود. چرا که مولفه های تغییر مقیاس منفی میشوند.

فصل پنجم :

پردازش تصویر

مباحث این فصل:

- سیستم پردازش تصویر
- فشرده سازی تصاویر
- کدگذاری تصاویر

پردازش تصویر :

بطور کلی گرافیک علمی است که در ارتباط با نحوه رسم اشکال پایه با توجه به سخت افزار مربوطه و محاسبات بهینه بحث می نماید.

اما پردازش تصویر علمی است که در ارتباط با عملیات بر روی تصاویر و همچنین تشخیص به کمک تصویر استفاده میشود.

موارد مهم در پردازش تصویر عبارتند از :

- بهینه سازی کیفیت تصاویر
- مقایسه تصاویر با یکدیگر
- برش و ترکیب تصاویر
- تشخیص به کمک تصویر

معمولاً پردازش تصویر در حالت دوبعدی و سه بعدی مورد بررسی قرار میگیرد. محاسباتی که در پردازش تصویر مورد نیازند عبارتند از محاسبات مربوط به ماتریسها و همچنین محاسبات مربوط به انتگرالها که در بحث بهینه سازی تصاویر استفاده میشود.

مهمترین اجزای یک سیستم پردازش تصویر عبارتند از :

- وسیله ای جهت انتقال تصاویر به درون کامپیوتر
- حافظه
- عملیات کنترل
- نمایشگر
- یک پردازشگر

جهت مقایسه تصاویر باید بتوان تصاویر را مدل سازی نمود. برای این کار یک مدل ریاضی را از تصویر ایجاد میکنیم. به طور کلی برای مدل سازی یک تصویر از رابطه زیر استفاده میکنیم.

$f(x,y)=i(x,y)+r(x,y)$

که در آن $f(x,y)$ و $i(x,y)$ بین صفر و بینهایت میباشند و $r(x,y)$ بین صفر و یک است.

بنابراین برای مدلسازی باید هر دو قسمت $i(x,y)$ و $r(x,y)$ مخالف صفر باشند. رابطه فوق که رابطه مدلسازی است باید تعیین کننده بحث نمونه برداری از تصویر و روشنایی مربوط به تصویر باشد.

1- تبدیل فوریه :

تبدیل فوریه گسسته : $f(\omega)=\int f(t)e^{-j\omega t}dt$

تبدیل فوریه پیوسته : $f(\omega)=\sum_{n=2}^{\infty} f(n)e^{-j\omega n}$

2- تبدیل Wavelet : مانند فوریه است با این تفاوت که بجای تابع $e^{-j\omega n}$ میتوان از توابع دیگر مثل $\cos(\omega n)$ استفاده نمود. کاربرد این تبدیل در فشرده سازی است.

3- تبدیل Sin-convolation: در این روش انتگرال به صورت دو بعدی و سه بعدی تخمین زده میشود. و اطلاعات مربوطه به صورت مستقیم در حافظه به روش ماتریسی ذخیره میشود.

بعد از مدلسازی که تابع $f(x, y)$ تولید شد، مولفه $i(x, y)$ در واقع تعیین کننده میزان نور خارج شده از منبع است و $r(x, y)$ میزان نور منعکس شده از جسم را نشان میدهد. بطور کلی تابع یک تابع پیوسته است و برای تبدیل این تابع به گسسته یا دیجیتال از مراحل نمونه برداری (Sampling) و کوانتومی کردن (Quantization) استفاده میشود.
منظور از نمونه برداری این است که با یک بازه زمانی خاص بتوان به طور تکراری مقادیری را از تابع پیوسته اندازه گیری نمود. نمونه های نباید از یکدیگر خیلی فاصله داشته باشند، زیرا در این صورت تعداد نمونه ها کم و ممکن است در کار سیستم اختلال ایجاد شود. همچنین اگر تعداد نمونه ها زیاد باشد، نیاز داریم که حجم زیادی از نمونه های را نگهداری کنیم که مشکلاتی را برای حافظه ایجاد خواهد کرد. بنابراین فرکانس نمونه گیری باید متعادل باشد.
منظور از مرحله کوانتومی کردن این است که بتوان نتایج بدست آمده از مرحله نمونه برداری را به مقادیر بالاتر گرد نمود.

تکنیک های نمونه برداری:	
روش Uniform :	روش Non-Uniform :
در این روش تمامی مختصات تابع $f(x, y)$ بصورت نقطه در نظر گرفته و آنها را در یک ماتریس $m \times n$ ذخیره میکنیم. هر چه تعداد نمونه ها بیشتر باشد، دقت نمونه برداری بیشتر خواهد بود. و بطور کلی هر چه m, n بیشتر شود کیفیت تصویر بیشتر خواهد شد.	در این روش m, n را با توجه به پیچیدگی تصویر تغییر میدهیم. یعنی m, n در هر قسمت از تصویر مقادیر متفاوتی خواهد داشت. در هر کجا کیفیت بیشتر باشد، m, n بیشتر خواهد بود.

مهمترین اهداف کدگذاری تصویر عبارتند از حجم کوچکتر و امنیت بالاتر در هنگام انتقال داده ها. بطور کلی داده دیجیتال میتواند کدگذاری شود و در واقع علاوه بر کدگذاری قابلیت فشرده سازی را نیز دارد. اما داده آنالوگ قابلیت کدگذاری و قابلیت فشرده سازی را ندارد.
BMP : فایلهای پردازش نشده با حجم بالا
GIF : تعداد رنگ ها فشرده سازی شده است.
JPG : مقدار رنگ و جزئیات فشرده سازی شده است.

در کدگذازی، تصویر به یک تصویر جدید تبدیل میشود که معمولاً کیفیت تصویر جدید را نخواهد داشت. روش های مختلف کدگذاری در جدول زیر آورده شده اند.

روشهای کدگذاری داده ها	
روش کدگذاری ماتریس	در این روش یک ماتریس، معادل تصویر مربوطه ایجاد میگردد و جابجایی بین نقاط ماتریس که در واقع نقاط تصویر هستند بر اساس الگوریتمی خاص صورت می گیرد. استفاده از این روش موجب میشود تصویر کدشده با تصویر اولیه تفاوت پیدا کند. اما تأثیری بر روی حجم داده های ذخیره شده و عمل فشرده سازی نخواهد داشت. مزیت این روش سادگی و عیب آن دقت پایین آن است.
روش PCM	در این روش تغییرات رنگ برای هر پیکسل نسبت به یک مقدار پایه ذخیره میگردد. هدف این روش فشرده سازی و عیب آن مشکل بودن یافتن مقدار پایه است.
روش هافمن	هدف این روش بهینه سازی طول کد موردنظر است. به این صورت که اگر یک

	حرف تعداد تکرار کمتری داشته باشد، طول کد اختصاص یافته به آن کمتر است.
روش Entropy	این روش مانند روش هافمن است . و لی بهتر از آن عمل مبکند.
روش Ran length	اساس این روش بر این است که پیکسلهای همسایه دارای رنگ مشابه اند. این روش نسبت به روش اول کلاسیک تر است. منظور از پیکسلهای همسایه پیکسلهایی هستند که فاصله آنها تا نقطه مورد نظر یک واحد است. در این روش به جای ذخیره سازی مختصات $f(x,y)$ میتوان فاصله بین مکانها را کهمعمولاً عددی ثابت است ذخیره نمود. عیب این روش این است که حجم داده کد شده به پیچیدگی تصویر وابسته است و ممکن است حجم را خیلی کم نکند. بطور کلی برای تصاویر ساده خوب است، اما برای تصاویر پیچیده به خوبی عمل نمیکند.
روش Two symbol	

فشرده سازی:

هدف از فشرده سازی حجم داده کمتر و همچنین محاسبات کمتر است. روش های فشرده سازی در جدول زیرآمده اند.

روشهای فشرده سازی	
فشرده سازی بر اساس مدل تصویر	این روش بر روی تابع $f(x,y)$ انجام میشود و طوری بر روی تابع عمل میکند که خروجی این تابع هنگام پیاده سازی تعداد بیت کمتری را شامل شود. درصد فشرده سازی این روشنسبتاً کم است.
فشرده سازی بر اساس تقریب	این روش از تقریب زدنمثلاً تقریب هر قسمت شکل یا تصویر با خط استفاده می نمایدمعمولاً طول هر خط به همراه نقاط ابتدایی و انتهایی ذخیره میشود.
روش Fractal :	در این روش تصویر به بخش هایی تقسیم میشود و سپس عمل فشرده سازی بر روی هر بخش انجام میشود. کیفیت فشرده سازی در این روش نسبت به حالات قبلی بهتر است.
فشرده سازی بصورت گسسته	در این روش از روی تابع اصلی عمل نمونه برداری را با سرعت کم انجام میدهیم. در نتیجه خروجی عمل نمونه برداری باعث میشود که تصویر بدست آمده دارای حجم کمتری باشد. باید دقت نمود، اگر سرعت نمونه برداری خیلی کم شود، کیفیت تصویر کاهش می یابد.
فشرده سازی با استفاده از تبدیلات	در این روش از فیلتر ها استفاده میشود. تصویری که از یک فیلتر عبور میکند از نظر ویژگیها با تصویر اولیه متفاوت است. فیلترها بر دو نوع خطی و غیرخطی میباشند.