22.Растерно и векторно представяне на графични данни. Растеризация на отсечка, окръжност и елипса

Компютърната графика е част от ИТ, която изучава методите и средствата свързани със създаването, обработка, съхраняване и разпространение на графични изображения.

Предимства и недостатъци на растерното и векторното представяне

Според начина на построение на изображението компютърната графика се дели на два основни вида – векторна и растерна.

1. Растерна графика

Изображението се състои от точки /пиксели/, еднакви по размер и различни по цвят. Благодарение на особеностите на нашето зрение и микроскопичните размери на пикселите, отделните точки се сливат в цялостно изображение.

Този метод на работа определя предназначението на повечето графични редактори като средства за обработка, а не толкова за създаване на графични изображения.

За всяка точка от растерното изображение се съхранява информация във файла, който го съдържа. Това е информация за нейното местоположение в растера /решетката/ и цвета, който тя притежава. Тази информация увеличава обема на файла, в които се съхранява изображението. Оттук идва и **основният проблем** с растерните изображения, а това са големите по обем файлове.

Върху големината на файла оказват влияние и други два фактора – разделителна способност на изображението и броят на цветовете, които се съдържат в него.

Друг **съществен недостатък** на растерните изображения е невъзможността те да се увеличават за да се разгледат детайлите. При увеличаване размера на изображението се увеличава и размерът на точките, от които то се състои. Този ефект на изкривяване или понижаване качеството на изображението се нарича пикселизация.

2. Векторна графика

Векторната графика е метод за представяне на компютърни изображения, при което те се описват с помощта на математически формули, функции, вектори и др. подходящи оператори. Тя се представя във вид на формула, а не като съвкупност от точки, както е при растерната графика. От съчетанието на няколко линии се получава формата на даден обект в изображението. При визуализиране на даден обект върху екрана

програмата първо изчислява по формулата неговия вид, а след това го представя. Работи се със самите обекти, а не с линиите, поради което графиката е известна още като обектноориентирана графика.Не по-малко значима е и точката, която се представя с две числа (x,y), определящи местоположението ѝ спрямо началото на координатната система.
Векторната графика повече се използва за създаване на изображения и по-малко за тяхната обработка. Всеки обект във векторната графика има определени свойства. При линиите това са форма, дебелина, цвят и вид (плътна, пунктирана и др.) и т.н. Широко приложение във векторната графика са намерили кривите на Безие.
Предимства - при векторната графика не възникват проблеми, които са свързани с увеличаване на изображението, защото процесът увеличаване е автоматично свързан с преизчисляване на формулите на кривите, от които е изградено изображението и следователно новополученото изображение е с нови параметри. Други **предимства** са малък обем на изходният файл и възможност за прилагане на неограничен брой деформации и трансформации - ротация, транслация, преобразуване и др. Като основен **недостатък** може да се спомене голямата трудност и неефективност при пресъздаване на фотореалистични модели.

Дефиниция на пиксел

Пикселът (на английски: *Pixel*) е най-малкият елемент, който изгражда дадено цифрово изображение. Всъщност това е много малка точка с променливи цвят и яркост. Изображението се изгражда от множество пиксели, подредени в правоъгълна решетка с определена разделителна способност. Поради малкия размер на отделният пиксел човешкото око не вижда "точки", а възприема цялото изображение. Съществуват различни модели за представяне на цвета на пикселите, като най-често това става чрез три цветови компонента - червено, зелено и синьо. Отново малкият размер спомага цветовите компоненти да се възприемат от човешкото око като един общ цвят.

Разрешаваща(разделителна) способност и големина на пиксела

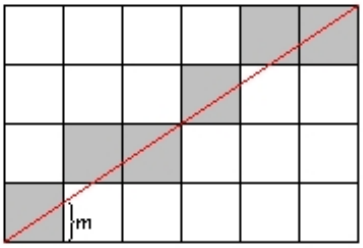
Разделителната способност е броят точки (пиксели), които могат да се изобразят в хоризонтален или вертикална посока.

Големината на пиксела се определя от броя на точките на инч и се измерва в dpi /dots per inch/.

Алгоритми на Брезенхам за растеризация на отсечка, окръжност - основна идея (компенсация на грешката от растеризация)

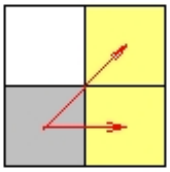
Често се налага да се изчертават на екрана основни геометрични обекти като отсечки, части от окръжност/елипса. Тъй като мониторът е правоъгълна таблица от пиксели, като всеки пиксел свети в определен цвят, естествено възниква проблемът за изчертаване на отсечка/окръжност със зададени координати върху тази правоъгълна таблица. Т.е трябва да определим кои точно пиксели трябва да светят. Проблемът не е сложен за решаване, но бързодействието е от изключително голяма важност.

1. Растеризация на отсечка



Ако си представим отсечката като безкрайно тънка линия, която минава през 2 точки от екрана, то трябва да 'светнем' квадратчетата (пикселите) през които тази отсечка минава. За удобство ще приемем, че отсечката върви от югозапад (долу-ляво) до североизток (горе-дясно) и ъгълът, който тя сключва със $\vec{Ox}$, е по-малък от 45 градуса (т.е. тангенсът на ъгъла между правата и $\vec{Ox}$ е <= 1/2). Всички останали случай са еквивалентни на този с подходящо ротиране на координатната система.

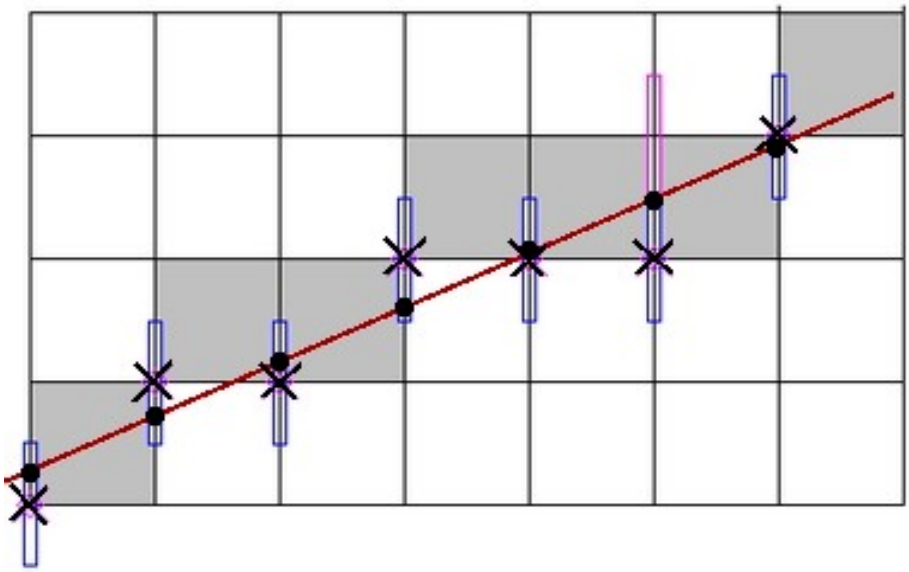
С тези ограничения задачата ни става наистина много лесна - ако си представим, че в момента сме в даден пиксел и се чудим в кой следващ пиксел да отидем имаме само 2 възможности - надясно или по диагонал нагоре и надясно:



На всяка стъпка от алгоритъма, трябва да знаем *текущото квадратче*, което сме светнали както и *грешката*, която показва колко далеч е минала линията от квадратчето.

За център на квадратчето ще приемем долния му ляв ъгъл. Ще разглеждаме също пресечните точки на линията (отсечката) с вертикалните линии на решетката. Спрямо тези точно пресечни точки (отбелязани с точка

на фигурата) избираме най-близката точка от решетката по тази вертикална линия (т.е най близкия долен ляв ъгъл на квадрат със същия x) - на фигурата е отбелязано с кръст. След като изберем долните леви ъгли просто осветваме съответните им квадратчета (т.е квадратчето, което стои горе вдясно спрямо точката).



Имаме дадени 2та края на отсечката - нека това са $(x_1, y_1), (x_2, y_2)$. Тогава $\Delta x = x2 - x1, \Delta y = y2 - y1$. Тъй като линията има наклон по малък от 45 градуса, то $\Delta x \geq \Delta y$.. Ако отбележим наклона с α, то $\alpha = \frac{\Delta y}{\Delta x} = m$. Това е първата много важна характеристика на линията (за единица дължина по x, колко пиксела нагоре (по y) се изкачва линията).

Алгоритъм на Брезенхам:

- Във всеки момент се знае кой е текущият пиксел, в който се намираме - x_c, y_c както и стойността на текущата грешка - e, т.е колко далече минава истинската линия от квадратчето. Ще приемем, че координатите на квадратчето са координатите на **долния му десен** ъгъл и ще се интересуваме повече от самото ъгълче, отколкото от целия квадрат.
- От всеки пиксел можем да се движим или надясно (т.е увеличаваме x) или по диагонал (т.е увеличаваме x и y). Това зависи от грешката - тя се изчислява като към старата грешка прибавим m. Т.е $e_{new} = e_{old} + m$. Тъй като грешката е разстоянието от долния десен ъгъл на пиксела до пресечната точка на истинската отсечка с вертикалната права, минаваща над този долен десен ъгъл, то за да получим следващата грешка прибавяме точно разстоянието, което отсечката ще "изкачи" за един пиксел. Тъй като искаме точката (долния десен ъгъл) да бъде най-близо до пресечната точка на отсечката с вертикалната права, то ако грешката е повече от $\frac{1}{2}$, избираме горната точка, иначе – долната.

Моментни стойности	x_c, y_c, e
Пресмятаме новата грешка	$e \mathrel{+}= m$
Ако грешката e > 0.5,то движение нагоре	if $(e > \frac{1}{2})$ $\{{+}{+}y_c;\ e \mathrel{-}= 1;\}$
Движение напред (винаги)	${+}{+}x_c$

Ако тангенсът на ъгъла между правата и $\vec{Ox}$ е по-голям от 0.5, то просто разменяме местата на **x** и **y** в алгоритъма.

Тъй като използването на floating point аритметика е по-бавно и води до натрупване на грешка в сравнение с използването на цели числа, то се прави преобразуване на грешката, така че да се използват само цели числа. Като начало сменяме проверката за движение нагоре(*if(e > ½)*) да бъде сравнение между цели числа. За тази цел началната грешка я правим да бъде –0.5. Това променя само условието за движение нагоре (т.е. то става от *if(e > ½) {...}* на *if(e > 0) {...}*)Тъй като $e_0 = -0.5$, то $e_1 = -0.5 + (\Delta y / \Delta x)$ => $2.e.\Delta x = 2.\Delta y - \Delta x$. Т.е. все едно умножаваме където има дроби, в които участва *e* по $2.\Delta x$

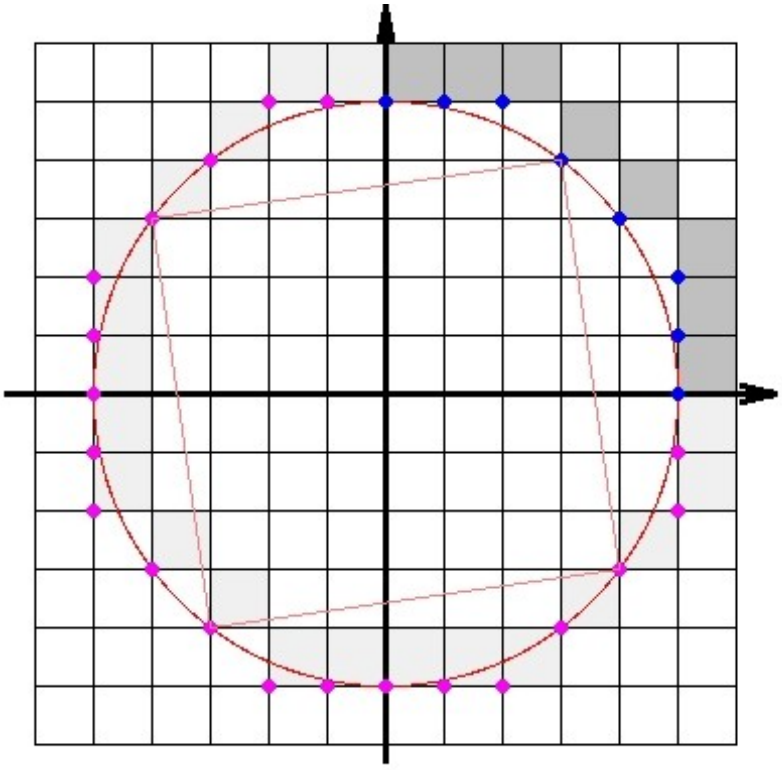
```
void DrawBresenham(int x0, int y0, int x1, int y1)
{
	bool steep = abs(x1-x0) < abs(y1-y0);
	if(x0==x1)
	{
		if(y0>y1)
			swap(y0,y1);
		for(int y=y0; y<=y1; y++)
			putpixel(x0,y);
	}
	else
	{
		if(steep) {swap(x0,y0);swap(x1,y1);}
		if(x0>x1) {swap(x0,x1);swap(y0,y1)};

		int dx = x1-x0;
		int dy = abs(y1-y0);
		int ystep = (y1>y0) ? 1 : -1;
```

```
        int y = y0;
        int e = 2*dy-dx;

        for(int x = x0; x <= x1; x++)
        {
            if(steep) putpixel(y,x);
            else putpixel(x,y);

            if( e > 0 ){
                y += ystep;
                e = e - 2*dx;
            }
            e += 2*dy;
        }
    }
}
```

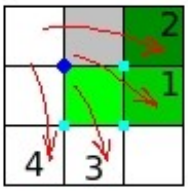
2. Растеризация на окръжност

Сега ще разгледаме подробно алгоритъма за растеризиране на окръжност. За удобство ще приемем за изчертаваната окръжност:

- центъра и е в $(0,0)$ (в общия случай трябва да транслираме)
- има целочислен радиус
- изчертаваме само частта, която е разположена в първи квадрант (останалите са симетрични)



Както при алгоритъма за отсечка, ще приемем, че пиксела се отъждествява с **долния си ляв** ъгъл. Също така по текущ пиксел в който се намираме (x, y), ще видим как да стигнем до следващия. В случая на окръжност, ако изчертаваме само първи квадрант имаме следните 3 случая: надясно (по хоризонтала), надолу и надясно (по диагонал), на долу (по вертикала). Също така има 4 случая на това от къде минава реално окръжността (т.е. между кои пиксели).



Ще разгледаме случай 1 и 2, останалите два са им симетрични. Основното, което ще използваме, е разстояние от точка до окръжност, което се свежда до разстояние между 2 точки (самата точка и центъра на окръжността). Нека с функцията $k(x, y)$ означим това разстояние. Тогава:

$k(x, y) = x^2 + y^2 - R^2$

$k(x, y) > 0$ точката е извън окръжността
$k(x, y) = 0$ точката е върху окръжността
$k(x, y) < 0$ точката е вътрешна за окръжността

Първо трябва да разберем дали сме в първите два случая (1 и 2) или във вторите два случая (3 и 4). За тази цел просто ще определим разположението на точката по диагонала (с координати $(x+1, y-1)$) спрямо окръжността - т.е дали тя е външна или вътрешна. Да положим на Δ разстоянието от диагоналната точка до центъра на окръжността:

$$\Delta = k(x+1, y-1) = (x+1)^2 + (y-1)^2 - R^2$$

Очевидно:

$\Delta < 0$	точката по диагонала е вътрешна за окръжността	намираме се в случай 1 или случай 2
$\Delta = 0$	точката по диагонала е върху окръжността	новата точка е по диагонал - няма нужда от разглеждане на случаи
$\Delta > 0$	точката по диагонала е извън окръжността	намираме се в случай 3 или случай 4

Да разгледаме подробно **случай 1** (окръжността минава между точките с координати $(x+1, y)$ и $(x+1, y-1)$ (т.е между дясната и диагоналната спрямо текущата). За да разберем коя от $2^{\text{те}}$ точки трябва да светнем, трябва да проверим до коя точка модула от разстоянието до окръжността е минимално - т.е да изберем по-близката до окръжността точка. С други думи трябва да сравним:

$$|k(x+1, y-1)| \lessgtr |k(x+1, y)| \iff \delta = |k(x+1, y)| - |k(x+1, y-1)| \lessgtr 0$$

Понеже сме в първи случай, знаем че точката $(x+1, y)$ е извън окръжността, а точката $(x+1, y-1)$ е в окръжността, следователно знаем знаците на изразите в модулите. Да махнем модула и да разпишем:

(4)

$$\begin{aligned}
\delta &= |\underbrace{k(x+1, y)}_{>0}| - |\underbrace{k(x+1, y-1)}_{<0}| \\
&= k(x+1, y) - (-k(x+1, y-1)) \\
&= (x+1)^2 + y^2 - R^2 + (x+1)^2 + (y-1)^2 - R^2 \\
&= (x+1)^2 + y^2 \underbrace{-2y+1} - R^2 + (x+1)^2 + (y-1)^2 - R^2 \underbrace{+2y-1} \\
&= 2\Big[(x+1)^2 + (y-1)^2 - R^2\Big] + 2y - 1 \\
&= 2\Delta + 2y - 1 \\
&= 2(\Delta + y) - 1
\end{aligned}$$

Сега да видим какво става според знака на δ:

$\delta > 0$ разстоянието до окръжността на горната точка е по-голямо	избираме диагоналната $(x+1, y-1)$
$\delta \leq 0$ разстоянието до окръжността на долната точка е по-голямо (или са равни)	избираме хоризонталната (т.е дясната) $(x+1, y)$

С това случай 1 е завършен. Да разгледаме **случай 2**: По принцип може да направим абсолютно същите разсъждения като в предния случай - това обаче не е необходимо. В случай 2 и диагоналната и дясната точка са в окръжността и очевидно дясната е по-близо, т.е в този случай избираме дясната. Сега ще проверим следното: **когато сме в случай 2,** $\delta \leq 0$, т.е стойността, която сметнахме за случай 1 ще важи и тук (т.е няма нужда даже да проверяваме дали сме в този случай - просто разписваме δ все едно сме в случай 1 и използваме резултата). Във втори случай сме когато дясната точка е в окръжността:

$$\begin{aligned}
k(x+1, y) &\leq 0 \\
&\iff \\
(x+1)^2 + y^2 - R^2 &\leq 0 \\
(x+1)^2 + y^2 - 2y + 1 - R^2 + 2y - 1 &\leq 0 \\
(x+1)^2 + (y-1)^2 - R^2 + 2y - 1 &\leq 0 \\
\Delta + 2y - 1 &\leq 0 \\
2(\Delta + y) - 1 &\leq \Delta < 0 \\
\delta &< 0
\end{aligned}$$

Т.е получихме, че когато сме във 2^{ри} случай, то сметнатото δ от първи случай е винаги отрицателно. Но това означава че ако просто бяхме *забравили* за 2^{ри} случай и приемем че има само първи, ако получим $\delta < 0$ пак ще вземем хоризонталната дясна точка - т.е ще направим правилен избор. Ето защо **1^{ви} и 2^{ри} случай се изчерпват с разглеждането на таблицата, показана в 1^{ви} случай**.

Както казахме 3^{ти} и 4^{ти} случай са напълно аналогични на разгледаните до сега. Ще запишем в таблица всички случай за улеснение:

$\Delta < 0$		
$\delta = 2(\Delta + y) - 1$		
$\delta > 0$	$(x+1, y-1)$	$\Delta' = \Delta + 2(x-y) + 1$
$\delta \leq 0$	$(x+1, y)$	$\Delta' = \Delta - 2y + 1$
$\Delta > 0$		
$\delta = 2(\Delta - x) - 1$		
$\delta \leq 0$	$(x+1, y-1)$	$\Delta' = \Delta + 2(x-y) + 1$
$\delta > 0$	$(x, y-1)$	$\Delta' = \Delta + 2x + 1$
$\Delta = 0$		
$\delta \leq 0$	$(x+1, y-1)$	$\Delta' = \Delta + 2(x-y) + 1$

Δ' е просто новата стойност на Δ. Така си спестявате малко умножения.

```
void Circle( int R )
{
	int x = 0;
	int y = R;
	int D = 2 * (1 - R);
	while( y > 0 )
	{
		putpixel( x, y );
		if( D < 0 )
		{
			int d = 2 * ( D + y ) - 1;
			if( d <= 0 ) mh(x, y, D);   // moves horizonatal
			else md(x, y, D);   // moves diagonal
		}
		else if( D > 0 )
		{
			int d = 2 * ( D - x ) + 1;
			if( d <= 0 ) md(x, y, D);   // moves diagonal
			else mv(x, y, D);   // moves vertical
		}
```

```
            else
            {
                md(x, y, D);   // moves diagonal
            }
        }
}

void mh( int& x, int& y, int& dd )
{
        x++;
        dd += 1 + 2 *x;
}

void mv( int& x, int& y, int& dd )
{
        y--;
        dd += 1 - 2 * y;
}

void md( int& x, int& y, int& dd )
{
        x++;
        y--;
        dd += 2 + 2 * (x - y);
}
```

Алгоритъм на Ван Акен за средната точка (за растеризация на елипса)

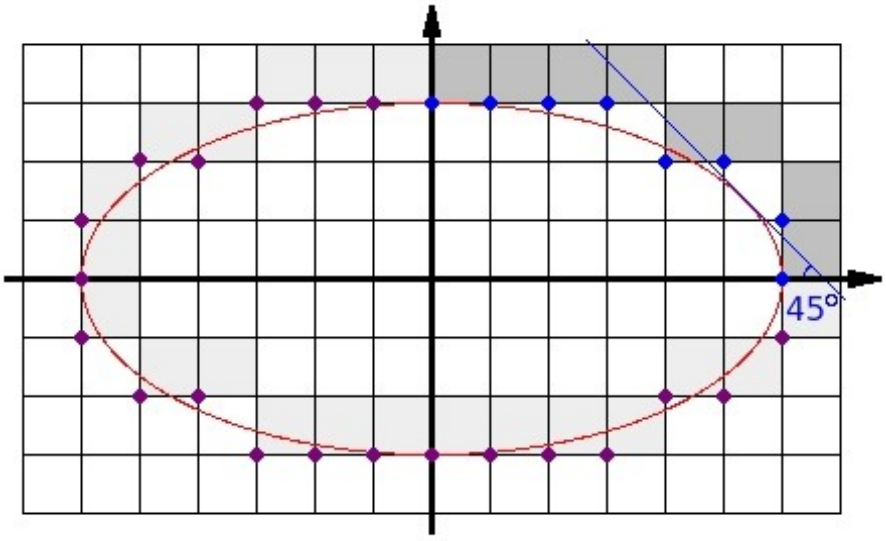
Остана да разлгедаме и алгоритъма на Ван Акен за растеризация на елипса. Ще приемем че:

- елипсата има център $(0,0)$ и осите и са успоредни на координатните (т.е така би изглеждала след канонизация)
- дължините на диаметрите й са реални числа
- ще начертаем само частта и в първи квадрант

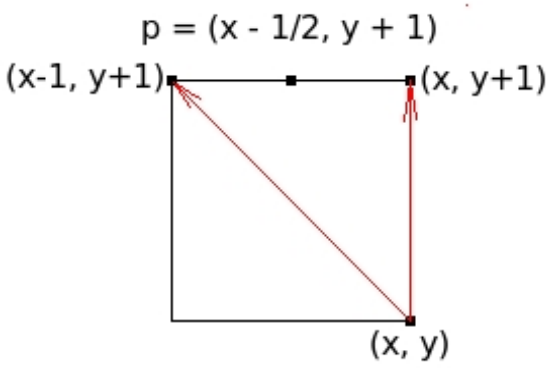
(6)

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad a, b > 0$$

Сега да погледнем какво трябва да се получи:



Ще чертаем само в първи квадрант отдолу нагоре. При тези условия ако в даден момент се намираме в клетка (x, y) имаме 3 възможни случая - нагоре, наляво и по диагонал (горе-ляво). В началото елипсата ще се движи по-скоро нагоре (т.е следващият пиксел ще е или над текущия или по диагонал), а след един точно определен момент (този, в който допирателната към елипсата сключва ъгъл 135 (= 180 - 45) градуса с Ox) - наляво. Преди и след този специален момент, алгоритъмът върви по подобен начин. Първо ще разгледаме ъгъл на допирателната по-малък от 135 градуса:



Ако в момента се намираме в (x, y), то следващата точка е или $(x, y+1)$ (нагоре) или $(x-1, y+1)$ (по диагонал). За да решим коя от 2^те ще изберем, ще проверим положението на средната точка $(x - \frac{1}{2}, y + 1)$ спрямо елипсата.

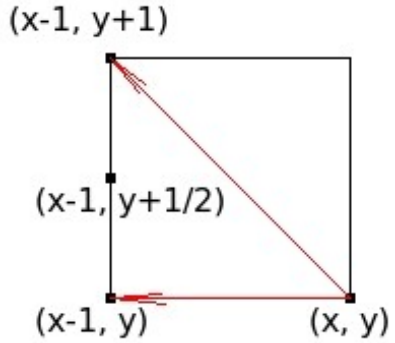
Просто ще заместим в уравнението на елипсата:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \iff f(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

$$f(x - \frac{1}{2}, y + 1) \lessgtr 0$$

$f(x-\frac{1}{2}, y+1) \leq 0$	средната точка е вътре или по контура	движим се към $(x, y+1)$
$f(x-\frac{1}{2}, y+1) > 0$	средната точка е вън	движим се към $(x-1, y+1)$

След като тангенсът на допирателната стане по-малък от -1 (в началото е $-\infty$ при ъгъл $\frac{\pi}{2}$ (защото е вертикална), накрая е 0 при ъгъл π (защото е хоризонтална)), трябва да разгледаме новата средна точка $(x-1, y+\frac{1}{2})$:



$f(x-1, y+\frac{1}{2}) \leq 0$	средната точка е вътре или по контура	движим се към $(x-1, y+1)$
$f(x-1, y+\frac{1}{2}) > 0$	средната точка е вън	движим се към $(x-1, y)$

Единствено остава да изясним преминаването от случай 1 към случай 2. Производната спрямо x на функцията $f(x,y)$ е точно:

$$\frac{dy}{dx} = -\frac{b^2x}{a^2y} \lessgtr -1$$

Заместваме със x, y и получаваме производната. После естествено я сравняваме с -1

```
void VanAken( float a, float b )
{
	int x, y;
	float d;
	x = (int)( a + 0.5 ); y = 0;
	while( b * b * (x - 0.5) > a * a * (y + 1) )
	{
		putpixel( x, y );
		d = f( x- 0.5, y + 1, a, b );
		if( d < 0 ) y++;
		else { x--; y++; }
	}
	while( x >= 0 )
	{
		putpixel( x, y );
		d = f( x - 1, y + 0.5, a, b );
		if( d < 0 ) { x--; y++; }
		else x--;
	}
}

float f( float x, float y, float a, float b )
{
	return b*b*x*x + a*a*y*y - a*a*b*b;
}
```