جامعة حماة
الكلية التطبيقية
قسم تقنيات الحاسوب

- المقرر: بيانيات الحاسب Computer Graphics
- المحاضرة : الرابعة

Chapter 4

خصائص المستقيم بإستخدام OpenGL

للمستقيم خصائص ثلاث: اللون، الثخانة، والنمط. يتم تحديد اللون باستخدام التابع نفسه الذي يطبق أيضًا على كافة الأوليات، في حين يتم تحديد الثخانة والنمط بواسطة توابع منفصلة.

تولّد الثخانة القياسية **standard-width** لمستقيم، كما في خوارزمية **Bresnham** **بعرض بكسل** واحد تلقائيًا. للحصول عل مستقيمات أكثر ثخانة، يتم بإظهار عدد من المستقيمات المتوازية المتلاصقة ونستخدم (**horizontal spane or vertical spane**) وفقًا لميل المستقيم.

الألوان والحساسية

الألوان:

code	البتات المخزنة			
	R (red)	G (green)	B (blue)	
0	0	0	0	
1	0	0	1	
2		1		
4	1			
5	1	1		
6	1		1	
7	255	255	255	

الحساسية:

الحساسية				
0.0	0	0	0	
0.33	1	0	1	
0.67	1	1	0	
1.0	1	1	1	

أنواع الخطوط & أوامر الرسم

يوجد 3 أنواع مختلفة للخطوط :

- متصل
- مقطع
- منقط
- نوع الخط SetLinetype(t): : - خط متقطع (1) - خط متصل (0).
- عرض الخط: gllinewidth(w)
- لتحديد اللون Setcolor(x)
- عدد الرؤوس (n). Polyline(n,x,y)
- لتحريك المؤشرالى بيكسل محددة moveto(x,y)
- لرسم خط من الموقع الحالي الى نقطة محددة lineto(x,y)

width :تعبر عن ثخانة المستقيم وهي قيمة حقيقية يتم تدويرها إلى أقرب قيمة صحيحة موجبة. يمكن تطبيق عدة أنماط :

solid line , dashed line , dotted line...

يتم ذلك بإظهار مقاطع محددة من المستقيم وترك عدد من البكسلات دون إظهار.

تلقائيًا يتم إظهار المستقيم بنمط **solid line** يمكن التحكم بعدد المستقيمات القصيرة **dashes** والنقاط **dots** والفراغات بواسطة التابع:

glLineStipple(repeatFactor, pattern);

Pattern: النقشة وتستخدم للدلالة على عدد صحيح مؤلف من **16 bits** يصف كيفية إظهار المستقيم (**1 bit=pixel on, 0 bit=pixel off**)

repeatFactor: معامل التكرار وتحدد عدة المرات التي تكرر من أجلها كل بت في النقشة.

قبل تطبيق التابع السابق يتوجب تأهيل **activate** النمط بواسطة التابع:

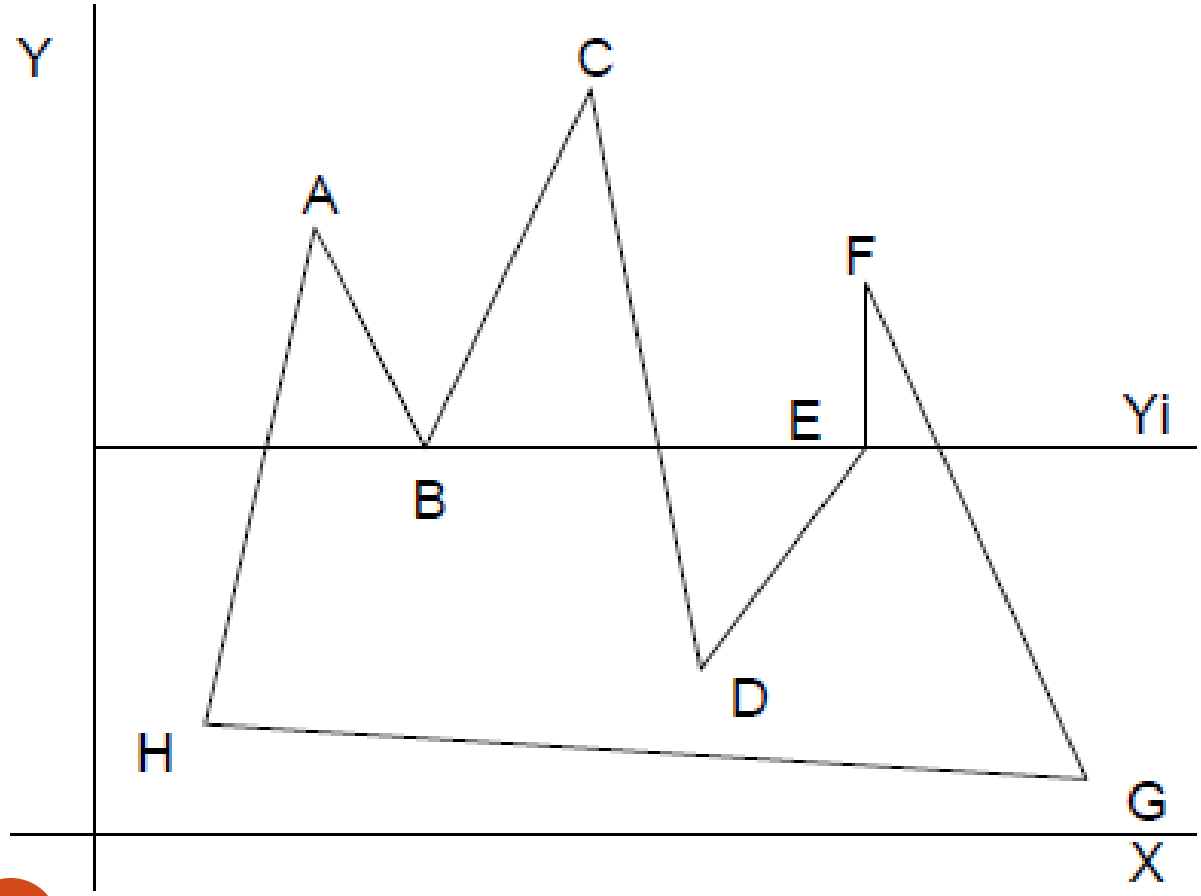
glEnable (GL_LINE_STIPPLE);

بعد الإنتهاء من الإظهار يتوجب إلغاء التأهيل **desactivate** النمط بواسطة التابع:

glDisable (GL_LINE_STIPPLE);

الخط الماسح

كون قائمة بالأضلاع التي تتقاطع مع الخط الماسح Yi مرتبة حسب خوارزمية الخط الماسح موضحا مؤشر البداية والنهاية؟



مؤشر البداية **CB**

CD

AB

AH

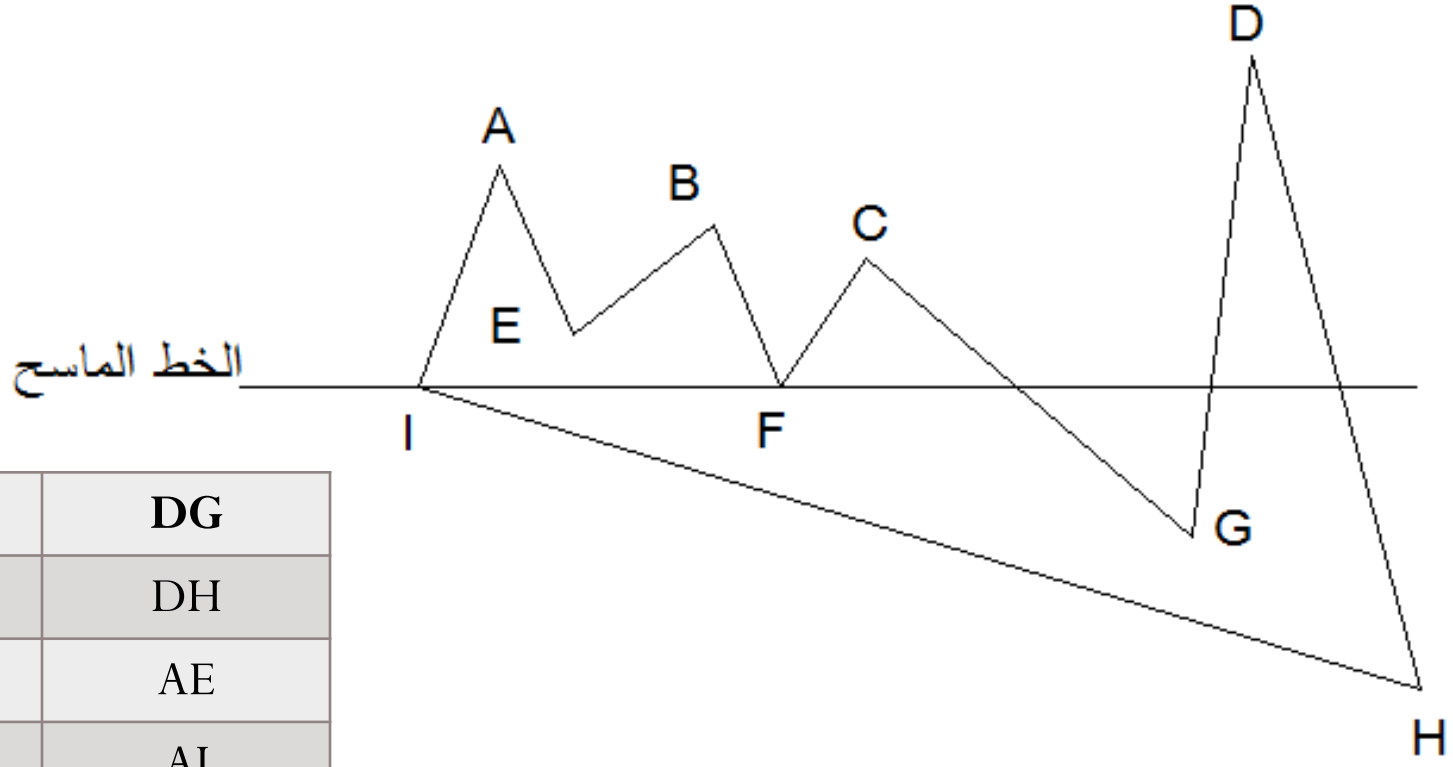
FE

FG

ED

مؤشر النهاية **HG**

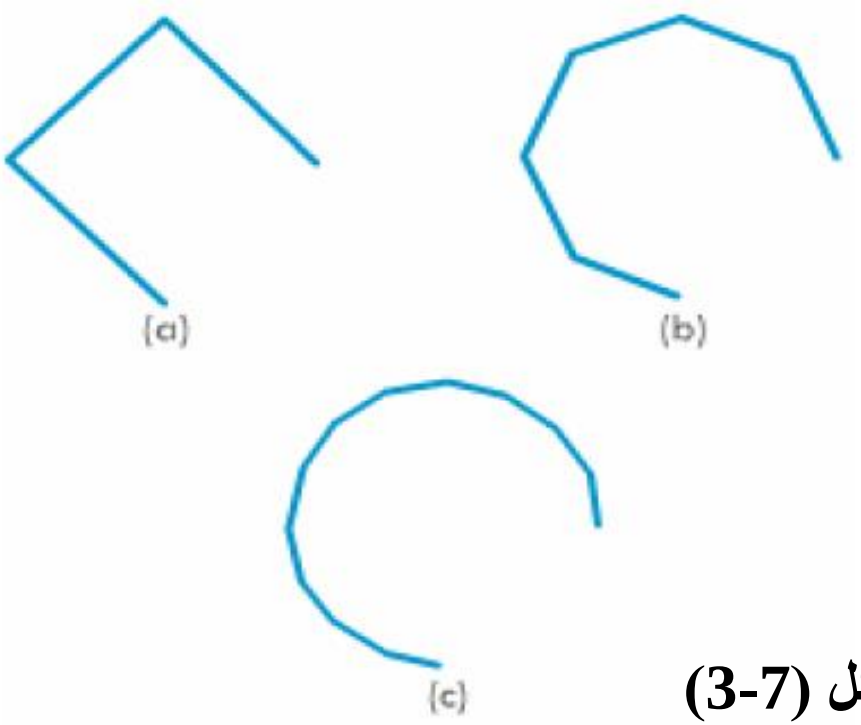
مثال: بين القائمة النشطة للخط الماسح في الشكل التالي:



DG	البداية
DH	
AE	
AI	
BE	
BF	
CF	
CG	
IH	النهاية

Circle-Generating Algorithms .4

لايتضمن OpenGL توابع توليد المنحنيات مثل القطوع والدوائر. يمكن توليد المنحني البسيط بتقريبه إلى مجموعة من المستقيمات **Polyline** بتحديد عدد من النقاط واقعة على المنحني وربطها بعدد من المستقيمات واستدعاء خوارزمية **Bresnham** من أجل كل مستقيم جزئي الشكل (3-7)



الشكل (3-7)

تعرّف الدائرة بمجموعة من النقاط تقع على مسافة r متساوية من المركز (x_c, y_c) وفقاً للمعادلة التالية :

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

يمكن إستخدام المعادلة السابقة لإظهار نقاط الدائرة من أجل x التي تأخذ قيمها بخطوات واحدية **unit steps** ضمن المجال $(x_c - r - x_c + r)$ وحساب قيم yعند كل خطوة

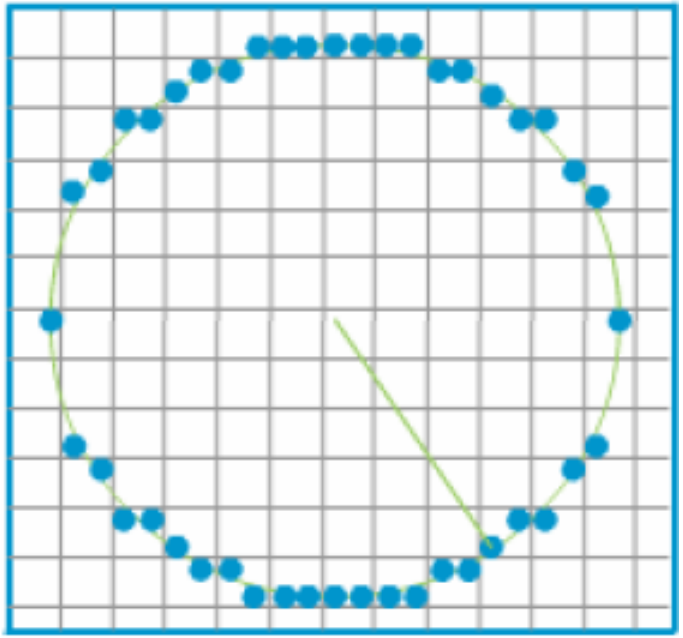
$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

يتطلب إظهار الدائرة وفقًا لهذه الطريق حسابات متعددة كما تظهر عدة فراغات بين النقاط عند الإظهار كما هو مبين في الشكل (8-3):

وفقاً للخوارزمية التالية:

```
void circleSimple(int xCenter, int yCenter,

int radius)
{
int pix = c.getRGB();
int x, y, r2;
r2 = radius * radius;
for (x = -radius; x <= radius; x++) {
y = (int) (sqrt(r2 - x*x) + 0.5);
setPixel(xCenter + x, yCenter + y);
setPixel(xCenter + x, yCenter - y);
}
}
```

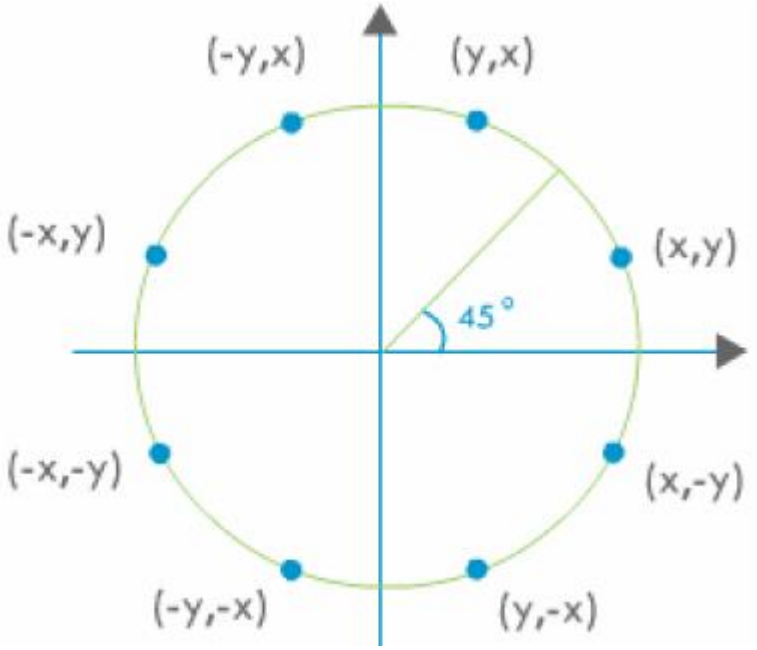


الشكل (3-8)

يمكن تحسين الخوارزمية السابقة باستخدام الإحداثيات القطبية (r , θ) والتعبير عن إحداثيات نقطة (x,y) من الدائرة:

$$x = x_c + r\cos\theta \quad , \quad y = y_c + r\sin\theta$$

مع ذلك، ورغم التحسين، تظل هذه الطريقة ذات زمن تنفيذ طويل. يمكن تسريع هذه الخوارزمية بالأخذ بعين الاعتبار تناظر نقاط الدائرة. عند تحديد نقاط المنحني في الثمن الأول للدائرة، يمكن توليد كافة الأثمان في المستو xy بالتناظر بالنسبة لمحور الإحداثيات x و y الشكل (3-9):



الشكل (3-9)

1-خوارزمية Bresnham's في توليد الدائرة

يمكن تطبيق خوارزمية **Bresnham** في رسم دائرة، نصف قطرها r ومركزها (x_c, y_c) بنفس الأسلوب الذي تمت دراسته في الفقرة السابقة، وذلك بتحديد البكسل الأقرب إلى الدائرة.

تبدأ الخوارزمية برسم النقاط، في الثمن الأول والتي تتغير قيم x فيه ضمن المجال $[0 - y]$ وقيم الميل ضمن المجال $[-1\ 0]$.

من أجل كل خطوة لـ x نستخدم الـ **Decision parameter** من أجل تحديد قيمة y الأقرب إلى الدائرة. نطبق خوارزمية **midpoint** على تابع الدائرة المعرف:

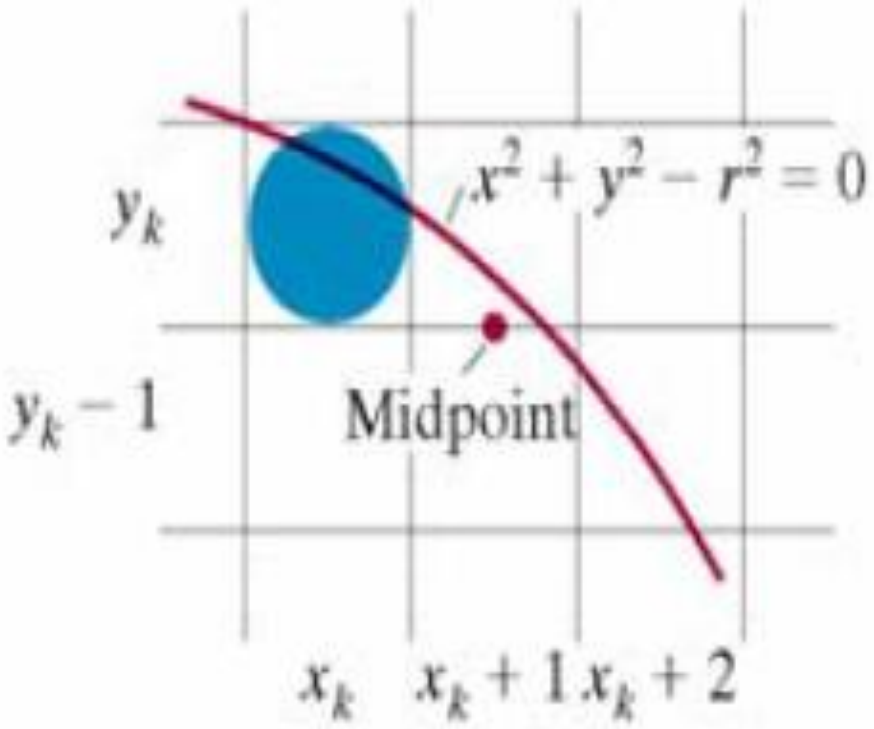
$$f_{circ}(x, y) = x^2 + y^2 - r^2$$

يتحدد موضع نقطة بالنسبة للدائرة وفق ثلاث حالات:

- النقطة داخل الدائرة يكون التابع سالباً $f_{circ}(x,y)<0$
- النقطة تقع على الدائرة وتحقق معادلته ويكون $f_{circ}(x,y)=0$.
- النقطة خارج الدائرة يكون التابع موجباً $f_{circ}(x,y)>0$

يتم تطبيق المعادلات الثلاث من أجل منتصف النقطة بين البكسلات قرب مسار الدائرة عند كل خطوة وبشكل متزايد حيث يعتبر تابع الدائرة الـ . **Decision parameter**.

يوضح الشكل (3–10) كيفية تطبيق خوارزمية **midpoint** حيث تم إظهار البكسل (x_k, y_k) ونحتاج إلى تحديد البكسل الأقرب لمسار الدائرة من بين بيكسلين مرشحين من أجل $x_k + 1$: البكسل $(x_k + 1, y_k)$ أو البكسل $(x_k + 1, y_k - 1)$ يتم تطبيق المعادلة الأخيرة من أجل منتصف النقطة بين البيكسلين:



الشكل (3-10)

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

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

إذا كان $p_k < 0$ تكون منتصف النقطة داخل الدائرة

ويكون البكسل ذو الإحداثية $(x_k + 1, y_k)$ الأقرب إلى مسار الدائرة، وإلا تكون منتصف النقطة خارج مسار الدائرة وبالتالي نختار البكسل ذو الإحداثية $(x_k + 1, y_k - 1)$.

يتم الحساب بشكل تكراري متزايد من أجل قيم x متزايدة ودراسة الـ **Decision parameter** وبالتالي نستنتج:

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

حيث y_{k+1} هي إما y_k أو y_k-1 وفقاً لإشارة p_k وبالتالي تتم الزيادة، للحصول على P_{K+1}، بمقدار $2x_{k+1}+1$ (في حال p_k سالب)، أو بمقدار $2x_{k+1}+1-2y_{k+1}$.

يتم الحصول على القيمة البدائية لـ **Decision parameter** من أجل النقطة $(x_0, y_0) = (0, r)$ حيث بعد التعويض الإحداثية $(1, r - 1/2)$ في معادلة الدائرة:

$$p_0 = \frac{5}{4} - r$$

كما يتم تقريب القيمة السابقة p_0 إلى القيمة الصحيحة بحيث تصبح:

$$p_0 = 1 - r$$

تعطى خوارزمية **midpoint** باستخدام توابع **OpenGL** على النحو التالي:

```
#include <GL/glut.h>
class scrPt {
public:
GLint x, y;
};
void setPixel (GLint x, GLint y)
```


```
#include <GL/glut.h>
class scrPt {
public:
GLint x, y;
};
void setPixel (GLint x, GLint y)
```

```
GLint p = 1-radius; //Initial value of midpoint
parameter.
circPt.x = 0; // Set coordinates for top point of circle.
circPt.y = radius;
/* Plot the initial point in each circle quadrant. */
circlePlotPoints (circCtr, circPt);
/* Calculate next points and plot in each octant. */
while (circPt.x < circPt.y) {
circPt.x++;
if (p < 0)
p += 2 * circPt.x + 1;
```

```
else {
circPt.y--;
p += 2 * (circPt.x - circPt.y) + 1;
}
circlePlotPoints (circCtr, circPt);
}
}
void circlePlotPoints (scrPt circCtr, scrPt circPt);
{
setPixel (circCtr.x + circPt.x, circCtr.y + circPt.y);
setPixel (circCtr.x - circPt.x, circCtr.y + circPt.y);
setPixel (circCtr.x + circPt.x, circCtr.y - circPt.y);
setPixel (circCtr.x - circPt.x, circCtr.y - circPt.y);
setPixel (circCtr.x + circPt.y, circCtr.y + circPt.x);
setPixel (circCtr.x - circPt.y, circCtr.y + circPt.x);
setPixel (circCtr.x + circPt.y, circCtr.y - circPt.x);
setPixel (circCtr.x - circPt.y, circCtr.y - circPt.x);
}
```