Algorithmes de base du graphisme

- ***Traçage***
- ***Découpage***
- ***Remplissage***
- ***Antialiasing :***

  Effet indésirable : les marches d'escalier ou crénelures.

  La suppression de ce phénomène est appelée **Anti-Aliasing**.

Traçage

A quoi sert le tracé de segment en informatique graphique?

Dessin en fil de fer

Remplissage de forme

Elimination des parties cachées.

Introduction

Le traçage de primitives 2D (segments, cercles, etc) est une des fonctionnalités de base du graphisme 2D qui utilisée par la plupart des applications en synthèse d'images.

Le problème qui se pose est la discrétisation (représentation discrète « pixels ») des primitives 2D.

Les aspects fondamentaux des algorithmes sont la rapidité d'exécution et la précision de l'approximation discrète réalisée.

Rappel

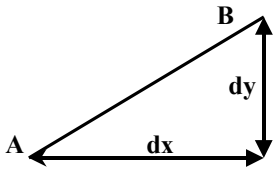
Repère

Le plan est muni d'un repère orthonormé $(o, \vec{i}, \vec{j})$, ce qui signifie que les vecteurs $\vec{i}$ *et* $\vec{j}$ sont de norme 1 et sont orthogonaux entre eux.

Chaque point M du plan est muni des coordonnées (x_m, y_m) qui déterminent sa position : $\vec{OM} = x_m \vec{i} + y_m \vec{j}$

x_m s'appelle l'abscisse du point M

y_m s'appelle l'ordonnée du point M.



Pente d'un segment

Soient A(xa, ya) et B(xb, yb) deux points distincts du plan
Par ces 2 points passe une seule droite D.

$dx = x_b - x_a$

$dy = y_b - y_a$ les différences entre les coordonnées de A et B.

m = dy/dx s'appelle la pente de la droite D.

Si dx = 0 la droite D est verticale

Si dy = 0 la droite D est horizontale.

Equation cartésienne

Considérons l'équation suivante :

$$F(M) = F(x, y) = dy*(x - x_a) - dy*(y - y_a) = 0$$

L'équation de la droite D passant par A et B.

En effet, les vecteurs AM et AB sont colinéaires : $\det\begin{vmatrix} x - x_a & x_b - x_a \\ y - y_a & y_b - y_a \end{vmatrix} = 0$

Vérification :

$$F(A)=F(x_a, y_a) = 0$$

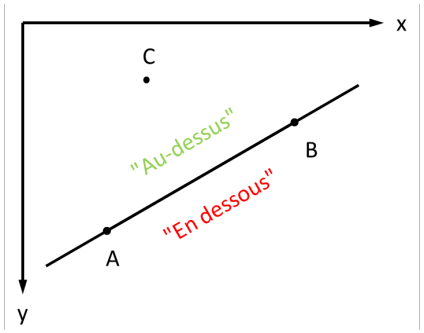
$$F(B)=F(x_b, y_b) = dy*dx - dx*dy = 0$$

Position d'un point par rapport à un segment

Soit $M(x_m, y_m)$ un point du plan tel que $M \notin D$.

Question :

M se trouve t-il à droite ou à gauche du segment [A, B] orienté de A vers B ?



Solution

Evaluation de $F(x_m, y_m)$

Si $F(x_m, y_m) < 0$, le point M se trouve à gauche du segment [A, B]

Si $F(x_m, y_m) > 0$, le point se trouve à droite du segment [A, B]

Si $F(x_m, y_m) = 0$, le point se trouve sur la droite du segment [A, B]

Augmentation de y le long d'un segment

Soit un point $M(x_m, y_m)$ de la droite D tel que $x_m = x_a + \Delta x$

Déterminer y_m ?

M vérifie l'équation cartésienne de la droite D, c'est à dire $F(M)=F(x_m, y_m)=0$

$F(x_m, y_m)=0$

$dy*(x_m - x_a) – dx*(y_m – y_a) = 0$

$dy*(x_a + \Delta x - x_a) – dx*(y_m – y_a) = 0$

→ $dy*\Delta x = dx*(y_m – y_a)$

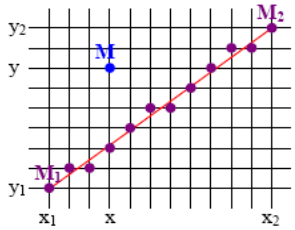
→ $y_m = y_a + (dy/ dx)*\Delta x$

→ $y_m = y_a + m*\Delta x$

m est la pente de D.

Algorithmes de tracé de segments

Cadre de l'étude



Soient M1(x1, y1) et M2(x2, y2) deux points situés sur un quadrillage régulier (un écran Bitmap : espace discret bidimensionnel).

Si on trace la droite qui relie ces deux points, alors elle est continue sans interruption.

But

Visualiser une droite de manière discrète (et non continue), et ceci à cause de la discrétisation des points situés sur l'écran.

Problématique

Comment déterminer quels points de l'espace formera cette droite ?

Conditions souhaitées

Le segment obtenu devra répondre à des Conditions :

- ✓ Tout point du segment discret est coupé par le segment continu.
- ✓ Tout point du segment discret touche au moins un autre point du segment, soit par un de ses côtés, soit par un de ses sommets.
- ✓ On trace le moins possible de points. Une conséquence est que, mis à part les deux extrémités, tout point du segment en touche exactement deux autres.
- ✓ Les tracés des segments [A, B] et [B, A] doivent être parfaitement identiques.

Simplification

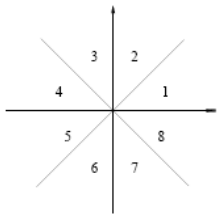


Figure 1: Les huit octants.

On désire tracer un segment de droite entre les points de coordonnées entières $A(x_a, y_a)$ et $B(x_b, y_b)$.

Pour simplifier on suppose :

xa < xb

ya <= yb

et (xb-xa) >= (yb-ya) <= 1

→ ce qui signifie que le segment est dans le premier octant.

Les autres cas se déduise par symétrie.

Remarque :

Si ces hypothèses ne sont pas vérifiées, il suffit d'opérer de la manière suivante pour se ramener aux hypothèses ci-dessus :

Si xa >= xb alors il faut intervertir xa et xb ;

Si ya > yb alors il faut intervertir ya et yb ;

et (xb-xa) < (yb-ya) alors il faut intervertir les couples (xa, xb) et (ya, yb).

Algorithme naïf (algorithme initial)

Problématique

Basé sur l'équation cartésienne de la droite D=(A, B).

m=dy/dx : pente de D avec $dx = x_b - x_a$ et $dy=y_b - y_a$.

M=(x, y) ∈ D vérifie l'équation de la droite D :

$$y = m*x + b \text{ avec } b = y_a - m*x_a$$

On fait varier l'abscisse x de notre point M de la valeur minimale x_a à la valeur maximale x_b, x étant entier, et on cherche la valeur de y entier pour que M soit le plus proche possible de la droite D(A, B).

Procédure segment(xa, xb, ya, yb, c)

Paramètre formels :

```
Xa, xb, ya, yb : Coordonnées du segment [A, B].
                 Entiers.
                 Paramètres en entré (E).
C : couleur du traçage. Entier (E).
```

Variables intermédiaires :

```
m, b : réel
X, y : entier
Début
    m ← (yb – ya) / (xb – xa)
    b ← ya – m*xa
    Pour x variant de xa à xb par pas de 1 faire
        Y ← Partie Entière (m*x +  b)
        AffichePixel(x, y, c)
    Finpour
Fin
```


Caractéristiques

- Les calculs se font avec des nombres réels : On effectue une multiplication en virgule flottante, une addition, et une opération d'arrondi.

- Des points successifs sont générés et converties en coordonnées entières avant d'être affichés à l'écran.

- Simplicité : simple à programmer.

- Gros défauts : lent à l'exécution car les calculs en virgule flottante (nombres réels) sont très lents.

➢ **Amélioration : Suppression de la multiplication.**

L' algorithme incrémental de base

Principe :

L'algorithme cherche à calculer un point à partir du point précédent.

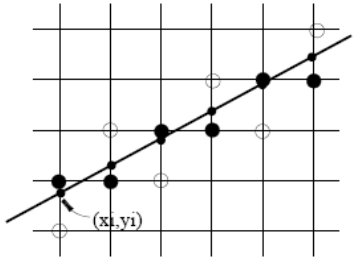


Figure : Segment dessiné avec la méthode de l'analyseur différentiel numérique.

En effet :

D'après l'équation cartésienne de la droite on calcule le point (x_{i+1}, y_{i+1}) successeur du point (x_i, y_i) de la manière suivante :

$y_{i+1} = m* x_{i+1} + b = m*(x_i + *\Delta x) + b = y_i + m*\Delta x$

si $\Delta x=1$ alors $y_{i+1} = y_i + m$

Donc, à partir de la position (x_i, y_i) on détermine la position du point suivant de manière incrémentale : $\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = y_i + m \end{cases}$

Procédure segment(xa, xb, ya, yb, c)

```
Paramètre formels :
Xa, xb, ya, yb : Coordonnées du segment [A, B]. Entiers.
                 Paramètre en entré (E).
C : couleur du traçage. Entier (E).
Variables intermédiaires :
m, b : réel
X, y : entier
Début
    m ← (yb - ya) / (xb - xa)
    y ← ya
    Pour x variant de xa à xb par pas de 1 faire
        AffichePixel(x, Partie Entière (y), c)
        Y ← y + m
    Finpour
Fin
```

Caractéristiques

Suppression de la multiplication entre flottants à chaque étape.

→ IL reste l'addition en virgule flottante et le calcul d'arrondi qui nécessite un temps de conversion non négligeable.

Amélioration : algorithme du point milieu = algorithme de Bresenham.

Algorithme du point milieu = algorithme de Bresenham(1965)

But :

Utilise des nombres entiers pour l'affichage d'une droite

Principe :

On connaît la position du point de départ, et on va raisonner par récurrence pour trouver les autres points qui formeront cette droite.

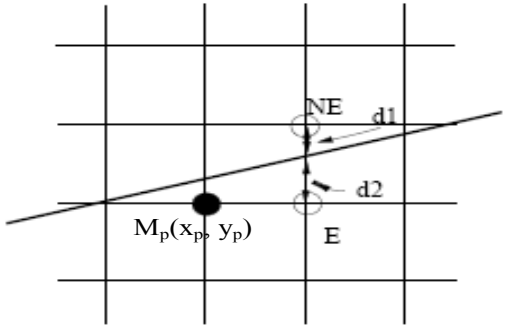


Figure : Algorithme de Bresenham

le choix se fait entre le point Est et le pointNord-est à chaque itération en fonction de l'erreur d2-d1.

Soit le Point de coordonnées $M_p(x_p, y_p)$, cherchons les coordonnées du point suivant $M_{p+1}(x_{p+1}, y_{p+1})$.

L'idée est de calculer l'erreur à chaque itération (distance du pixel à la droite) et de choisir la meilleure approximation.

On teste pour cela le signe de e=d2– d1.

$$\begin{cases} e = d2 - d1 >= 0 & NE \text{ (Nord - Est)} \\ e < 0 & E \text{ (Est)} \end{cases}$$

Nous avons deux possibilités pour calculer le pixel suivant M_{p+1} (x_{p+1}, y_{p+1}) à afficher à partir de la connaissance du pixel précédent du pixel précédent $M_p(x_p, y_p)$:

- le pixel E= (x_p+1, y_p) (Point Est)
- le pixel NE=(x_p+1, y_p+1) (Point Nord-Est)

Méthode :

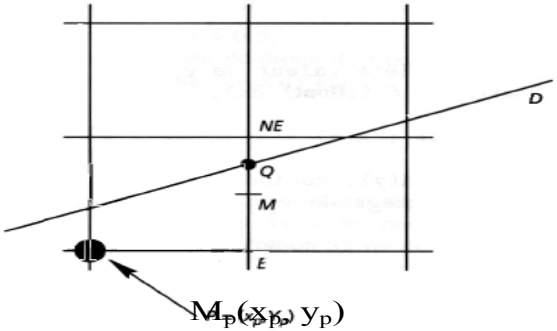


Figure 1.7 Détermination de (x_{p+1}, y_{p+1}) à partir de (x_p, y_p).

- Soit Q l'intersection de la droite D et la droite verticale $x = x_p + 1$ et soit M le point milieu du segment [E, NE].
- On cherche à déterminer si le point M se trouve au-dessus ou au-dessous du point Q.
- Si le point M se trouve au-dessus du point Q, on choisit le point E.

  Sinon on choisit le point NE.

→ Cela revient à déterminer si le point M se trouve au-dessus ou au-dessous de la droite D.

On peut le déterminer en utilisant l'équation cartésienne de D :

$F(x, y) = dy*(x - x_a) – dx*(y – y_a) = 0$

Pour savoir si le point M se trouve au-dessus ou au-dessous de la droite D, il suffit de déterminer le signe de F au point $M=(x_p+1,\ y_p+1/2)$:

Si $F(M) <= 0$, on choisit alors le point E

Si $F(M) > 0$, on choisit alors le point NE.

Remarque

Un calcul direct de F(M) est trop coûteux.

On souhaite faire un calcul incrémental.

Posons donc $\mathbf{d_p = 2*F(M)=2*F(x_p+1,\ y_p+1/2)}$

et calculons d_{p+1} en fonction de d_p.

On distingue deux cas suivant que (x_{p+1}, y_{p+1}) est égale à E ou NE.

1er cas : $E = (x_{p+1}, y_{p+1}) = (x_p+1,\ y_p)$

$d_{p+1}/2 = F(x_{p+1} + 1,\ y_{p+1} + 1/2)$

$= F(x_p+2 , y_p+ ½)$

$= dy * (x_p + 2 – x_a) –dx*(y_p+ ½ - y_a)$

or $d_p/2 = F(M)=F(x_p+1,\ y_p+1/2)$

$= dy * (x_p + 1 – x_a) –dx*(y_p+ ½ - y_a)$

Donc $d_{p+1} = d_p + 2*dy$

Posons $\delta_E=2*dy$

2éme cas : $NE = (x_{p+1}, y_{p+1}) = (x_p+1,\ y_p+1)$

$d_{p+1}/2 = F(x_{p+1} + 1,\ y_{p+1} + 1/2)$

$= F(x_p+2 , y_p+ 1 + ½)$

$= dy * (x_p + 2 – x_a) –dx*(y_p+ 3/2 - y_a)$

or $d_p/2 = F(M)=F(x_p+1,\ y_p+1/2)$

$= dy * (x_p + 1 – x_a) –dx*(y_p+ ½ - y_a)$

Donc $d_{p+1} = d_p + 2*(dy – dx)$

Posons $\delta_{NE}=2*(dy–dx)$

Donc : On détermine d_{p+1} en fonction de d_p par une addition en distinguant deux cas.

Calcul de la valeur initial d_0 :

$\mathbf{d_0 / 2} = F(x_a+1,\ y_a+1/2)$

$=dy*(x_a+1- x_a) – dx(y_a+1/2- y_a)$

$= dy – ½*dx$

donc $d_0 = 2*dy – dx$

Procédure Bresenham(xa, xb, ya, yb, c)

Paramètre formels :

```
Xa, xb, ya, yb : Coordonnées du segment [A, B]. Entiers.
                 paramètre en entré (E).
C : couleur du traçage. Entier (E).
```

Variables intermédiaires :

```
m, b : réel
X, y : entier
```

Début

```
    dy ← (yb – ya)
    dx ← (xb – xa)
    IncrE ← 2*dy
    IncrNE ← 2*(dy-dx)
    dp ← 2*dy –dx      /* Valeur initial de dp */
    y ← ya
    Pour x variant de xa à xb par pas de 1 faire
        AffichePixel(x, y, c)
        Si (dp <= 0) alors
            dp ← dp + IncrE   /* On choisit le pixel Est E */
        sinon
            y ← y + 1
            dp ← dp + IncrNE   /*On choisit le pixel Nord Est NE */
        finsi
    finpour
```

Fin

Caractéristiques

Calcul avec des nombres entiers

Affichage plus rapide → Permet d'avoir des animations de plus en plus rapide.

Généralisation au huit octants

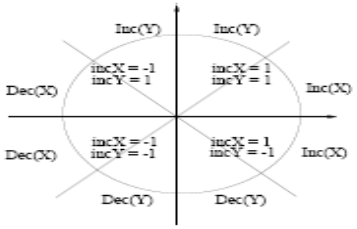


Figure 4: Généralisation de l'algorithme de Bresenham aux 8 octants.

La généralisation se fait en choisissant différents valeurs d'incrémentation pour x et y et en itérant sur x ou y en fonction de la valeur du coefficient directeur (> ou < à 1)

Supposons le segment à discrétiser A(xa, ya), B(xb, yb) avec ya <= yb

1er cas :

si xa <= xb

dx = xb – xa

dy = yb – ya

On distingue deux cas :

a) si dx >= dy

- d0 est égale à 2*dy –dx
- $\delta_E=2*dy$ **et** $\delta_{NE}=2*(dy-dx)$
- à chaque étape :

  choisir le pixel E revient à incrémenter x

  et choisir le pixel NE revient à incrémenter x et y.

b) si dx < dy

- d0 est égale à 2*dx –dy
- $\delta_E=2*dx$ **et** $\delta_{NE}=2*(dx-dy)$
- à chaque étape :

  choisir le pixel E revient à incrémenter y

  et choisir le pixel NE revient à incrémenter x et y.

2ème cas :

si xb < xa

dx = xa – xb

dy = yb – ya

On distingue deux cas :

c) si dx >= dy

- d0 est égale à 2*dy –dx
- $\delta_E=2*dy$ **et** $\delta_{NE}=2*(dy-dx)$
- à chaque étape :

  choisir le pixel E revient à décrémenter x

  et choisir le pixel NE revient à décrémenter x et incrémenter y.

d) si dx < dy

- d0 est égale à 2*dx –dy
- $\delta_E=2*dx$ **et** $\delta_{NE}=2*(dx-dy)$
- à chaque étape :

  choisir le pixel E revient à incrémenter y

  et choisir le pixel NE revient à décrémenter x incrémenter y.

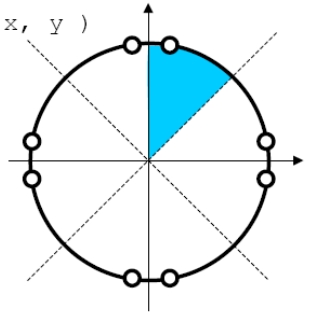
```
Procédure BresenhamGénéral(xa, xb, ya, yb, c)
Début
    dy ← (yb – ya)
    dx ← (xb – xa)
    si (dx > 0) alors
        IncrX ← 1
    Sinon
        IncrX ← -1
        dx ← -dx
    finsi
    si (dy > 0) alors
        IncrY ← 1
    Sinon
        IncrY ← -1
        dy ← -dy
    finsi
    Si dx >= dy alors
        /* Algorithme précédent */
    sinon
        /*Inverser x et y dans l'algorithme précédent */

Fin
```


Algorithme de Bresenham pour le tracé de cercles

Méthode

- Procédure incrémentale
- Découpage de l'espace en octants



Hypothèse simplifiée

- On suppose le centre o à l'origine
- Le rayon r est un entier
- On trace seulement le second octant.
- Le reste du cercle sera obtenu par symétries par rapport aux diagonales et aux axes.

Principe

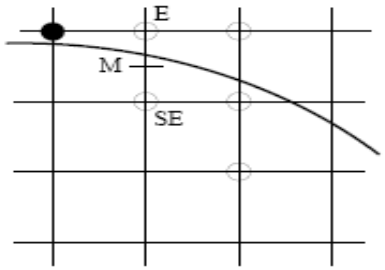
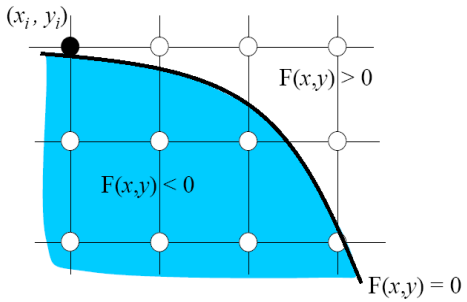
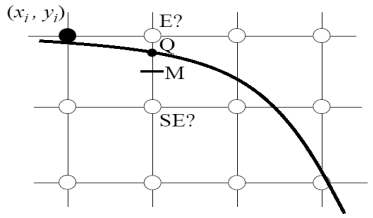


Figure 5: Tracer de cercle

- Si un pixel est allumé en position (x, y), le prochain pixel sera en position « Est » (x+1, y) soit en position « Sud Est « (x + 1, y – 1).
- L'algorithme est basé sur l'étude, pour chaque colonne, de la position du point intermédiaire entre les deux pixels superposés qui encadrent le cercle.
- L'idée de l'algorithme est de définir la position de chaque point intermédiaire de façon incrémentale, colonne après colonne.

Détermination de la formule incrémentale





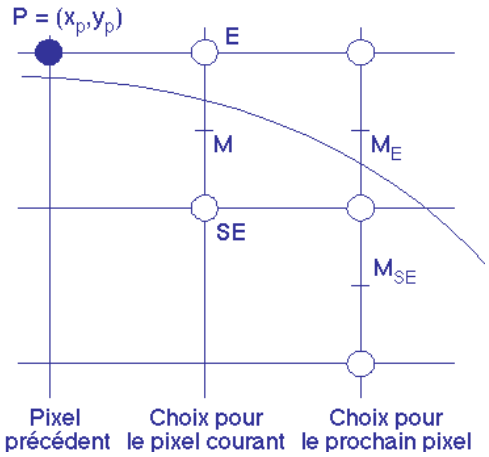
Soit un point M(x, y).

$F(x, y) = x^2 + y^2 - r^2$ est l'équation implicite d'un cercle.

Si $F(x, y) < 0$ alors M est à l'intérieur du cercle.

Si $F(x, y) = 0$ alors M est sur le cercle.

Si $F(x, y) > 0$ alors M est à l'extérieur du cercle.



Soit P(x_p, y_p) est la position du pixel allumé.

$d_p = d_{pred} = F(M) = F(x_p+1,\ y_p-1/2) = (x_p+1)^2 + (\ y_p-1/2)^2 - r^2$

Si $d_{pred} < 0$, on choisit le pixel E, et le prochain point intermédiaire est M_E

$$d_{p+1} = d_{now} = F(M_E) = F(x_p+2,\ y_p-1/2) = (x_p+2)^2 + (\ y_p-1/2)^2 - r^2$$

$$= d_p + (2*x_p + 3\ )$$

si $d_{pred} > 0$, on choisit le pixel SE, et le prochain point intermédiaire est M_{SE}

$$d_{p+1} = d_{now} = F(M_{SE}) = F(x_p+2,\ y_p-3/2) = (x_p+2)^2 + (\ y_p-3/2)^2 - r^2$$

$$= d_p + (2*x_p - 2*y_p + 5\ )$$

→ On obtient une formule récurrente en n'utilisant que des entiers.

Initialisation

La première valeur de d est obtenue à partir de la position (o, r) pour le point intermédiaire (1, r – ½)

$$d = 1^2 + (r - ½)^2 - r^2$$

$$= 5/4 - r = 1.25 - r$$

On utilise la fraction 1.25, comment peut-on l'enlever?

Si on pose $h_i = d_i - 0.25$, on peut remplacer l'initialisation par:

$h0 = 1 - r$

On devrait donc aussi modifier la comparaison par :

« Si $hi < -0.25$ alors … »

Cependant, on remarque que *hi* sera toujours un entier donc :

$hi < -0.25 \Leftrightarrow hi < 0$

On peut donc seulement changer l'initialisation et remplacer *di* par *hi* ailleurs.

PointMilieuCircle(r, c)

Paramètre formels :

```
r : rayon du cercle. Entier(E).
C : couleur du traçage. Entier (E).
```

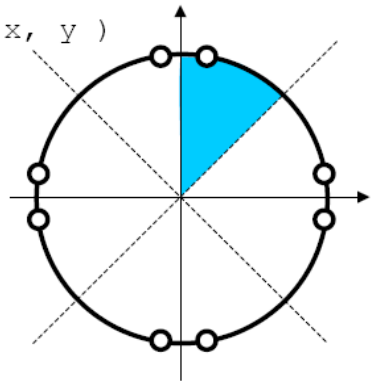
Variables intermédiaires :

```
x, y : Entiers
d : réel
```

```
Début
    x <- 0
    y ← r
    d ← 5/4 - r          // d ← 1 - r
    affichePixel(x, y, c)
    tantque (y > x ) faire    /* Octant 2 */
        si (d < 0) alors
            d ← d + 2*x + 3
        sinon
            d ← d + 2*x -2*y + 5
            y ← y -1
        finsi
        x ← x + 1
        affichePixel(x, y, c)
    fintantque
FIN
```


Remarque

La généralisation au huit octants se fait en remplaçant la procédure AffichePixel par la procédure suivante :




```
AffichePixelCircle(x, y, c)
    Début
        AffichePixel( x, y, c)
        AffichePixel(x, -y, c)
        AffichePixel(-x, y, c)
        AffichePixel(-x, -y, c)
        AffichePixel( y, x, c)
        AffichePixel(y, -x, c)
        AffichePixel(-y, x, c)
        AffichePixel(-y, -x, c)
    Fin
```