# Dessin de points, lignes et cercles

## 1- Les deux fonctions essentielles : putpixel() et getpixel()

Nous avons déjà rencontré la fonction putpixel(). La deuxième, getpixel(), s'en déduit aisément. Rappelons leur définition :

- Fonction *void putpixel*(*int xe, int ye, Uint32 couleur*) : elle colorie le pixel en position écran *xe , ye* avec la couleur *couleur*
- Fonction *Uint32 getpixel*(*int xe, int ye*): elle ramène la couleur du pixel en *xe, ye*.

D'où leur programmation :

```
void putpixel(int xe, int ye, Uint32 couleur)
{ Uint32 * numerocase;
numerocase= (Uint32 *)(ecran->pixels)+xe+ye*ecran->w;
*numerocase=couleur;
}

Uint32 getpixel(int xe, int ye)
{ Uint32 * numerocase;
  numerocase= (Uint32 *)(ecran->pixels)+xe+ye*ecran->w;
  return (*numerocase);
}
```

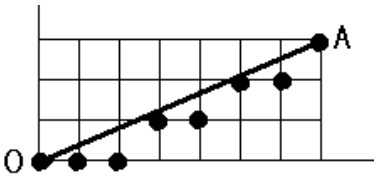
## 2- Tracé d'une ligne

### 2-1) Cas particulier d'un segment en pente douce issu de l'origine

On se place dans un repère orthonormé *Oxy*. Notre objectif est de tracer un segment issu de l'origine *O*. Plus précisément, prenons le cas d'un segment [*OA*] avec *A* de coordonnées entières positives *dx* et *dy* dans le repère orthonormé d'origine *O*, et supposons que sa pente est inférieure ou égale à 1, soit $dy \leq dx$. Rappelons que la pente d'une droite est le rapport de la variation verticale par la variation horizontale correspondante lorsque l'on circule sur la droite. Par exemple, une pente de 10% (c'est-à-dire 10 / 100 = 0,1) signifie que si l'on avance horizontalement de 100 m on monte verticalement de 10 m, ou encore que si l'on avance horizontalement d'un mètre, on monte de 0,1 m (quand on avance d'une unité, on monte d'une valeur égale à la pente). En prenant une pente inférieure ou égale à 1, cela veut dire que la droite fait un angle compris entre 0° et 45° avec l'axe horizontal, en montant lorsque l'on va vers la droite.

Pour tracer cette droite sur la grille de l'écran de l'ordinateur, où tous les pixels ont des coordonnées entières, on doit allumer des pixels qui en général ne sont pas exactement sur la droite puisque celle-ci n'a que peu de points à coordonnées entières sur elle. On va en fait construire ce que l'on appelle un chemin rasant par en-dessous. Ce chemin, comme on va le voir, est à base de pas horizontaux (1, 0) ou de pas diagonaux (1, 1). Voici un exemple avec *dx* = 7 et *dy* = 3, la pente étant 3 / 7.

Quand on prend les valeurs entières successives de *x*, de 0 à *dx* = 7, les points correspondants sur la droite ont pour ordonnée 0/7, 3/7, 6/7, 9/7, 12/7, 15/7, 18/7, 21/7=3, toutes de la forme *k dy* / *dx*, avec les numérateurs augmentant de *dy* = 3 à chaque fois. Il s'agit d'approcher ces points par des points à ordonnées entières situés au plus près d'eux et au-dessous. On va remplacer les ordonnées exactes sous forme de fraction (*k dy*) / *dx* par le quotient euclidien de *k dy* par *dx* : par exemple l'ordonnée 9 / 7 est remplacée par l'ordonnée 1. A cause de la pente inférieure ou égale à 1, chaque fois que *x* augmente de 1, l'ordonnée entière *y* (le quotient euclidien) reste soit fixe, soit augmente de 1.




La droite parfaite est approchée par la succession des points représentés par des ronds.

| Fractions : | 0/7 | 3/7 | 6/7 | 9/7 | 12/7 | 15/7 | 18/7 | 21/7 |
|---|---|---|---|---|---|---|---|---|
| Quotient euclidien *y* : | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 3 |
| Augmentation de *y* : | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| Reste euclidien : | 0 | 3 | 6 | 2 | 5 | 1 | 4 | 0 |

Pour tracer la «droite», on va utiliser la suite des restes euclidiens. Quand le numérateur *k dy* de la fraction augmente de *dy*=3, le reste augmente de 3 aussi, le quotient restant fixe, mais il peut devenir trop grand en dépassant *dx*=7, dans ce cas on doit rectifier l'erreur de la division en augmentant le quotient de 1 et en diminuant le reste de *dx*=7. A chaque fois que l'on fait cette rectification, *y* augmente de 1. Ainsi la droite parfaite est remplacée par un cheminement à base de pas horizontaux → ou diagonaux ↗ . Remarquons que si la pente n'est pas comprise entre 0 et 1, on ne peut pas utiliser ce même type de pas. On verra cela plus tard.

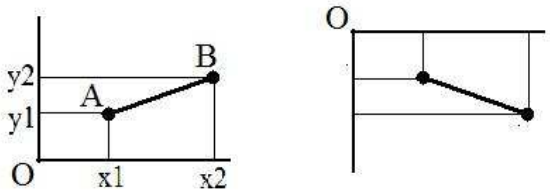
D'où le programme :

```
On se donne dx et dy entiers positifs avec dy≤ dx
x=0 ; y=0 ; reste=0 ;  dessiner ce point x,y
for(i=1 ; i <= dx ; i++)
  {x++ ; reste+=dy ; if (reste > =dx) {reste - = dx ; y ++ ;}  dessiner le point x,y}
```

Il s'agit tout simplement de l'algorithme de la division euclidienne, comme on l'apprend à l'école primaire. Quand on fait une erreur en obtenant un reste trop grand, on la rectifie en augmentant le quotient.

### 2-2) Première généralisation

On va maintenant tracer un segment joignant un point *A* (*x*1, *y*1) à un point *B*(*x*2, *y*2), toujours avec une pente entre 0 et 1. On ne part plus de l'origine *O* du repère, ce qui entraîne une légère modification du programme précédent.

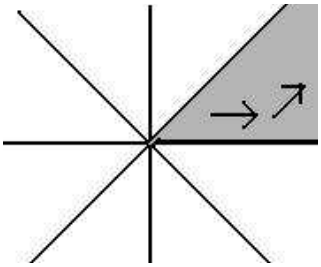



*A gauche* le segment *AB, à droite* ce que l'on voit sur l'écran

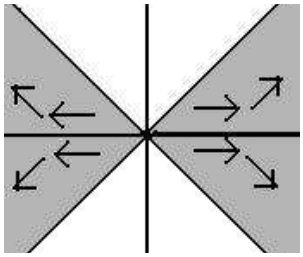
*On se donne* x1, y1, et x2, y2
dx=x2 – x1 ; dy = y2 – y1 ; residu = 0 ; *dessiner le point* (x1, y1)
for(i=0 ; i<dx ; i++)
{ x+=1 ; residu +=dy ; if (residu >=dx) {residu -=dx ; y+=1 ;}
*dessiner le point courant* (x,y)
}

Si l'on dessine les points en faisant *putpixel*(*x*,*y*) on obtiendra une figure à l'envers (voir dessin ci-dessus) puisque l'origine de la zone écran est en haut à gauche. Si l'on veut remettre les choses à l'endroit, faire *putpixel*(*x*, 599 – *y*), au cas où la hauteur d'écran est de 600.

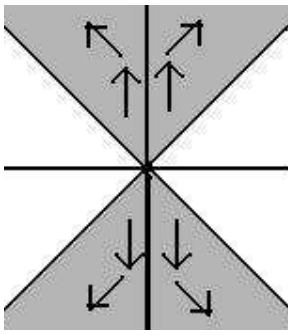
**2-3) Cas général**



A partir du point de départ *A*, on distingue huit zones avec huit frontières rectilignes. Nous avons étudié une zone, celle où la pente est entre 0 et 1, avec $dx > 0$ et $dy > 0$, d'où *x* qui avançait de 1 en 1, et *y* qui augmentait de temps à autre de +1, ce que nous écrirons maintenant *pasx*=+1, et *pasy* = +1. Mais dans d'autres zones, ces pas sont soit + 1 soit -1.



Dans les quatre zones indiquées ci-contre où la droite fait un angle faible avec l'horizontale, avec $|dx| > |dy|$, on a deux cas : lorsque *dx* est positif, on prend *pasx* = +1,et pour $dx < 0$, *pasx* = -1. De même avec *dy*.



Il reste les quatre dernières zones où la pente est forte, celles où $|dx| < |dy|$, Il suffit d'intervertir abscisse et ordonnée par rapport au cas précédent.

Enfin les huit frontières rectilignes sont traitées séparément.

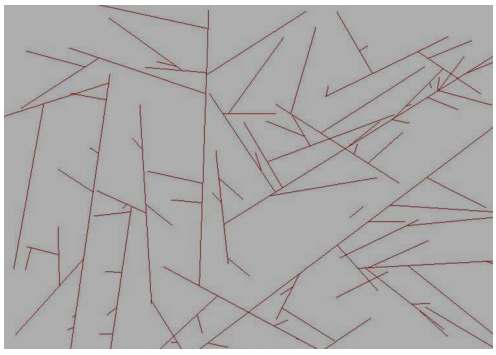
**2-4) Programme du tracé**

La fonction *ligne*(*int x0,int y0, int x1, int y1, Uint32 c*) est chargée de tracer un segment d'extrémités (*x0,y0*) et (*x1,y1*) comme si c'était un rayon lumineux, en allant du point (*x0,y0*) au point (*x1,y1*), avec la couleur *c*.

On trouvera ci-dessous le programme correspondant. Il est intégré dans un exemple où des « droites » rouges sont tracées au hasard sur l'écran (ici la fonction *getpixel* ne sert à rien)

```
#include <SDL/SDL.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

void pause(void);
void putpixel(int xe, int ye, Uint32 couleur);
void ligne(int x0,int y0, int x1,int y1, Uint32 c);
SDL_Surface * ecran;

int main(int argc, char ** argv)
{
   Uint32 rouge, blanc; int i;
   srand(time(NULL));

   SDL_Init(SDL_INIT_VIDEO);
   ecran=SDL_SetVideoMode(800,600,32, SDL_HWSURFACE);
   rouge=SDL_MapRGB(ecran->format,255,0,0);
   blanc=SDL_MapRGB(ecran->format,255,255,255);
   SDL_FillRect(ecran,0,blanc);

   for(i=0;i<1000;i++)
   ligne(rand()%800, rand()%600, rand()%800, rand()%600, rouge);

   SDL_Flip(ecran);
   pause();
   return 0;
}

void pause(void)
{
   SDL_Event evenement;
   do
   SDL_WaitEvent(&evenement);
   while(evenement.type != SDL_QUIT && evenement.type != SDL_KEYDOWN);
}

void ligne(int x0,int y0, int x1,int y1, Uint32 c)
{
int dx,dy,x,y,residu,absdx,absdy,pasx,pasy,i;
dx=x1-x0; dy=y1-y0;
residu=0;   /* il s'agit d'une division euclidienne, avec quotient et reste */
x=x0;y=y0;  putpixel(x,y,c);

if (dx>0) pasx=1;else pasx=-1; if (dy>0) pasy=1; else pasy=-1;
```

```
absdx=abs(dx);absdy=abs(dy);

if (dx==0) for(i=0; i<absdy; i++) { y+=pasy;  putpixel(x,y,c); }
            /* segment vertical, vers le haut ou vers le bas */
else if (dy==0) for(i=0; i<absdx; i++){ x+=pasx;   putpixel(x,y,c);  }
                /* segment horizontal, vers la droite ou vers la gauche */
else if (absdx==absdy) for(i=0; i<absdx; i++)
                        {x+=pasx; y+=pasy;   putpixel(x,y,c); }
        /* segment diagonal dans les 4 cas possibles */

else if (absdx>absdy)   /* pente douce */
for(i=0; i<absdx; i++)
            { x+=pasx; residu+=absdy;
             if (residu >= absdx) {residu -=absdx; y+=pasy;}
             putpixel(x,y,c);
            }
else for(i=0; i<absdy; i++)   /* pente forte */
           { y+=pasy; residu +=absdx;
             if (residu>=absdy)   {residu -= absdy;x +=pasx;}
             putpixel(x,y,c);
           }
}

void putpixel(int xe, int ye, Uint32 couleur)
{ Uint32 * numerocase;
numerocase= (Uint32 *)(ecran->pixels)+xe+ye*ecran->w;
*numerocase=couleur;
}
```

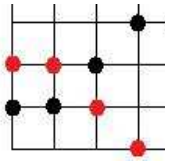


**2-5) Remarques**

- A cause des *putpixel*(*x, y, c*), le programme dessine les droites sens dessus dessous, avec l'origine du repère situé dans le coin en haut à gauche de l'écran. Si l'on veut éviter ce problème, on procède à un changement de repère, par exemple en prenant l'origine du nouveau repère au centre de l'écran, avec *xo* = 400 et *yo* = 300, on fait *xe = xo + x*, et *y = yo –y*, puis *putpixel*(*xe, ye, c*).

- Si l'on rajoute des *if* (*getpixel(x,y)== rouge) break ;* dans le programme avant les *putpixel*(), on obtient des lignes qui seront bloquées dès qu'elles rencontrent une autre ligne. Le programme *ligne*() peut ainsi être aménagé pour avoir des « droites » à tête chercheuse, réagissant en fonction de l'environnement.

- En fait on s'aperçoit sur le dessin ci-dessus que parfois certaines droites ne sont pas bloquées par d'autres. Cela tient à la présence des pas diagonaux utilisés pour tracer les droites. Deux droites sécantes peuvent avoir leur représentation sur l'écran qui ne se coupent pas, dans le cas où elles se traversent entre deux segments diagonaux :



Pour éviter ce défaut, il convient de remplacer chaque trait diagonal par un trait horizontal et un trait vertical, ce qui rajoute un point. Dans ce cas, deux droites sécantes restent sécantes sur l'écran.

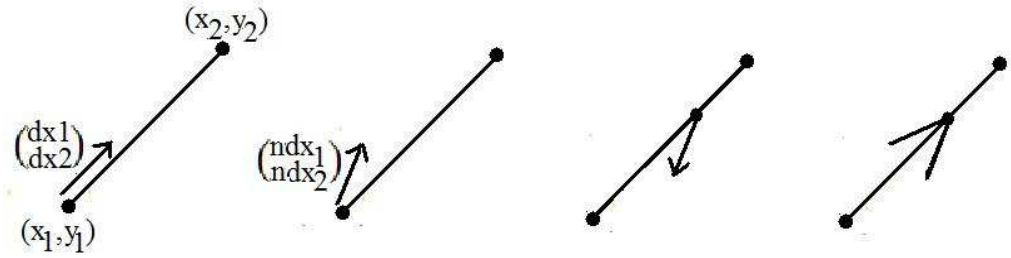
- Que l'on fasse ou non la rectification précédente en cas de besoin, il reste que deux droites sécantes en un point ont leur représentation sur l'écran qui peuvent se couper en plusieurs points, et d'autant plus que leurs pentes sont proches. Ces imperfections peuvent fausser les calculs. Dans de nombreux cas, on est amené à travailler avec des nombres flottants pour faire les calculs, et le dessin est fait ensuite sur l'écran en entiers, sans que cela se répercute sur les calculs. Par exemple, si l'on a besoin du point d'intersection de deux droites, on résout d'abord le système des deux équations des droites, en utilisant les « flottants », et on trace ensuite le point obtenu sur l'écran.

Imaginons par exemple que l'on veuille dessiner le mouvement d'une boule sur un billard rectangulaire, avec ses rebonds successifs sur les bordures. On devra calculer à chaque fois le point d'intersection entre la droite décrite par la boule et la bordure concernée. Si l'on utilisait une droite en marches d'escalier à tête chercheuse, se bloquant dès sa rencontre avec une bordure, il se produirait de petites erreurs à l'intersection, et au bout de quelques rebonds, le dessin deviendrait complètement faux.

On est souvent ramené à la stratégie classique : le programme fait les calculs en flottants, avec des coordonnées de points de l'ordre de quelques unités, puis on fait un zoom pour dessiner les résultats obtenus sur l'écran. Rappelons les formules de passage :
$xe = xo + zoom*x$ ; $ye = yo - zoom*y$.

### 2-6) Comment dessiner une flèche, pour représenter un segment orienté entre deux points, ou un vecteur

On veut placer une flèche sur un segment joignant deux points dont les coordonnées écran sont $(x_1, y_1)$ et $(x_2, y_2)$. Pour cela on prend le vecteur correspondant de coordonnées $dx = x_2 - x_1$ et $dy = y_2 - y_1$. Puis on le divise par sa longueur $d$, le calcul étant fait en flottants, d'où un vecteur de




longueur 1, et on le multiplie par 10. Le vecteur obtenu, de coordonnées $dx_1$= 10.**dx*/*d*, $dy_1$=10.**dy*/*d*, aura une longueur de 10 pixels quelle que soit la longueur du segment initial. Ce vecteur est ensuite tourné d'un certain *angle*, par exemple π/6, en appliquant les formules de la rotation, ce qui donne le vecteur de coordonnées $ndx_1$, $ndy_1$. Enfin on retourne ce vecteur, en prenant $-ndx_1$, $-ndy_1$ et on lui donne comme origine un point ($xf_1$, $yf_1$) situé par exemple aux deux-tiers du segment initial. L'extrémité du vecteur sera alors le point de coordonnées $xf_2= xf_1 - ndx_1$ $yf_2= yf_1 - ndy_1$, et il reste à tracer le segment joignant les deux points. On fait de même avec –*angle* au lieu de *angle*, et l'on aura finalement une petite flèche dessinée sur notre segment initial.

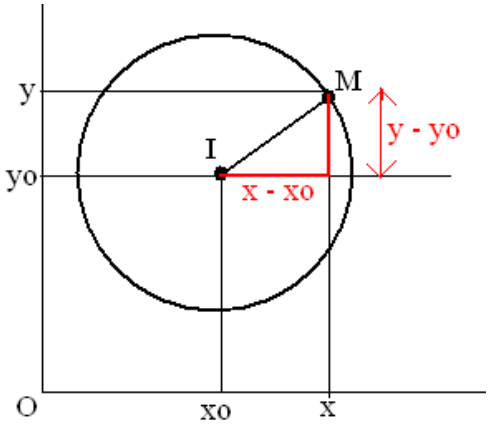
```
void fleche(int x1, int y1, int x2, int y2)
{
int dx,dy;
float xf1,yf1,xf2,yf2,d,dx1,dy1,ndx1,ndy1,ndx2,ndy2,angle=M_PI/6.;
line(x1,y1,x2,y2);
dx=x2-x1; dy=y2-y1;       d=sqrt(dx*dx+dy*dy);
if (d!=0.)      /* si le vecteur n'est pas nul */
{ dx1=10.*(float)dx/d; dy1=10.*(float)dy/d;
  ndx1=dx1*cos(angle)-dy1*sin(angle);
  ndy1=dx1*sin(angle)+dy1*cos(angle);
  xf1=0.3*x1+0.7*x2; yf1=0.3*y1+0.7*y2;     xf2=xf1-ndx1; yf2=yf1-ndy1;
  ligne(xf1,yf1,xf2,yf2);
  ndx2=dx1*cos(-angle)-dy1*sin(-angle);
  ndy2=dx1*sin(-angle)+dy1*cos(-angle);
  xf2=xf1-ndx2; yf2=yf1-ndy2;    ligne(xf1,yf1,xf2,yf2);
}
else    /* si le vecteur est nul, on dessine un cercle du point vers lui-même */
    {cercle(x1+10,y1,10); ligne(x1+20,y1,x1+23,y1-6);
     ligne(x1+20,y1,x1+15,y1-5);
    }
}
```

Ces dessins avec flèches sont essentiels dès que l'on veut montrer le cheminement d'un mobile sur un graphe.

## 3- Tracé d'un cercle

### 3-1) Equation d'un cercle

Un cercle est caractérisé par son centre (un point) et son rayon (un nombre positif ou nul). Plaçons-nous dans un repère orthonormé *Oxy*. On se donne le centre *I* du cercle par ses coordonnées *xo*, *yo*, ainsi que son rayon *R*. Un point *M* de coordonnées *x*, *y* est sur le cercle si et seulement si $IM = R$, ou encore $IM^2 = R^2$, ce qui se traduit par :



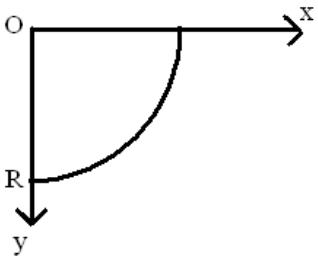

$(x - xo)^2 + (y - yo)^2 = R^2$

grâce au théorème de Pythagore.

On vient d'obtenir l'équation du cercle : un point $M(x, y)$ est sur le cercle si et seulement si ses coordonnées vérifient l'équation précédente.

Posons $F = (x - xo)^2 + (y - yo)^2 - R^2$, où $F$ est fonction de $x$ et de $y$. L'équation du cercle devient $F = 0$. Un point $M$ est à l'extérieur du cercle si et seulement si $F > 0$, et il est à l'intérieur si et seulement si $F < 0$. Il s'agit maintenant de tracer le cercle sur l'écran d'ordinateur.

### 3-2) Tracé d'un quart de cercle de centre *O* et de rayon *R* avec *R* entier



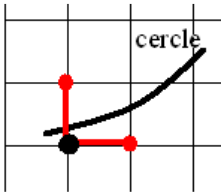

Plaçons-nous dans le repère de l'écran, d'origine $O$ situé dans le coin en haut à gauche. On se donne un nombre entier $R$ (par exemple $R = 100$), et l'on veut tracer sur l'écran le quart de cercle de centre $O$ et de rayon $R$, dont l'équation est $F = 0$, soit ici $x^2 + y^2 - R^2 = 0$. Pour cela on ne peut qu'utiliser le quadrillage des pixels de l'écran. Le cercle va être approché par des points du quadrillage qui vont former des marches d'escalier, avec des pas d'une unité soit vers la droite soit vers le haut. On part du point $A$ de coordonnées 0, $R$ qui est exactement sur le cercle, donc avec $F = 0$. Puis on va cheminer en faisant un pas vers la droite ou un pas vers le haut, en choisissant à chaque fois le point parmi les deux possibles qui soit le plus près du cercle, c'est-à-dire celui pour lequel $F$ en valeur absolue est le plus proche de 0.

A chaque étape du processus, on se trouve en un point $x$, $y$ du quadrillage, avec une certaine valeur de $F$ (qui est rarement nulle, le point étant en général à l'extérieur ou à l'intérieur). Il s'agit de déterminer le point suivant. Si l'on avance d'un pas horizontal, ce qui donne le point $x + 1$, $y$, la nouvelle valeur de $F$ est

$F1 = (x + 1)^2 + y^2 - R^2 = x^2 + y^2 - R^2 + 2x + 1 = F + 2x + 1.$
Autrement dit, $F$ augmente de $2x + 1$.

Si l'on avance d'un pas vertical, la nouvelle valeur de $F$ est

$F2 = x^2 + (y - 1)^2 - R^2 = x^2 + y^2 - R^2 - 2y + 1 = F - 2y + 1.$
Ainsi $F$ augmente de $-2y + 1$.



Entre les deux valeurs de $F1$ et $F2$ on choisit la plus petite en valeur absolue, et l'on prend le nouveau point correspondant. On avance ainsi point par point en oscillant autour du cercle au plus près, jusqu'à l'arrivée au point final d'ordonnée $y$ nulle.

On en déduit le programme de tracé de ce quart de cercle.

```
x=0 ; y=R ; F=0 ;  /* conditions initiales */
while (y >=0)
   { dessiner le point x,y
     F1=F + 2x+1 ;  F2 = F – 2y +1 ;
     if (abs(F1)<abs(F2))  {x+=1 ; F=F1;}
     else { y-=1; F=F2;}
   }
```

### 3-3) Tracé d'un cercle quelconque

Généralisons ce qui précède à un cercle de centre ($xo$, $yo$) et de rayon $R$ sur l'écran. Son équation $F = 0$ s'écrit $(x – xo)^2 + (y – yo)^2 – R^2 = 0$. On va tracer le même quart de cercle que précédemment, avec quelques petites modifications dans les calculs. Le point bas $A$ a maintenant comme coordonnées $xo+R$, $yo$. Les valeurs de $F1$ et $F2$ sont :

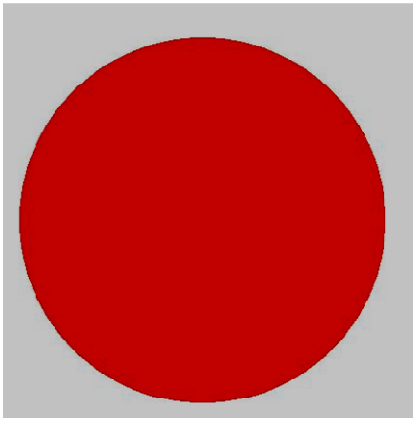
$F1 = (x + 1 – xo)^2 + (y – yo)^2 – R^2 = (x – xo)^2 + (y – yo)^2 – R^2 + 2(x – xo) + 1 = F + 2x + 1.$
$F$ augmente de $2(x – xo) + 1$.

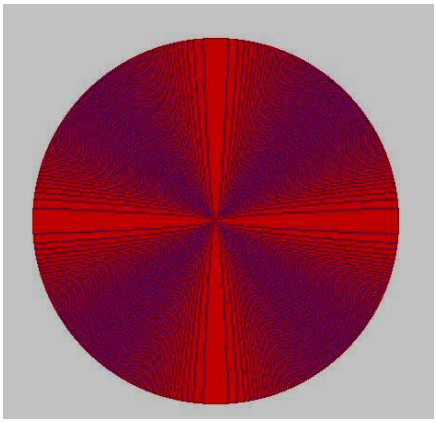
$F2 = (x – xo)^2 + (y – 1 – yo)^2 – R^2 = (x – xo)^2 + (y – yo)^2 – R^2 – 2(y – yo) + 1$
$= F – 2(y – yo) + 1.$ $F$ augmente de $– 2(y – yo) + 1$.

Chaque fois que l'on obtient un point du quart de cercle, on dessine aussi par symétries les trois points situés sur les autres quarts de cercle. Le symétrique du point $M$ ($x$, $y$) par rapport au diamètre horizontal du cercle a pour coordonnées $x'$, $y'$ telles que $x = x'$, et $(y + y') / 2 = yo$, car le milieu du segment de jonction a pour ordonnée $yo$, soit $y' = 2\ yo – y$. De même par la symétrie d'axe le diamètre vertical, le symétrique de $M$ a pour coordonnées ($2xo – x$, $y$). A son tour, le dernier symétrique a pour coordonnées ($2xo – x$, $2\ yo – y$). On en déduit le programme, avec la fonction *cercle*($xo$, $yo$, $R$, *couleur*). Nous avons pris comme dimensions d'écran 800 et 600. On a fait en sorte que les points du cercle ne débordent pas de l'écran, et qu'ils soient tous tracés une fois et une seule.

```
void cercle( int xo, int yo, int R, Uint32 couleur)
 {
    int x, y, F, F1, F2,newx,newy;
    x=xo; y=yo+R; F=0;
    if (x<800 && x>=0 && y>=0 && y<600) putpixel(x,y,couleur);
    if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600) putpixel (x,2*yo-y, couleur);

    while( y>yo)
      {
        F1=F+2*(x-xo)+1; F2=F-2*(y-yo)+1;
        if ( abs(F1)<abs(F2))  { x+=1; F=F1;}
        else {y-=1; F=F2;}
        if (x<800 && x>=0 && y>=0 && y<600) putpixel(x,y,couleur);
         newx=2*xo-x ; newy=2*yo-y ;
        if (x<800 && x>=0 && newy>=0 && newy<600) putpixel(x, newy,couleur);
```

```
            if (newx<800 && newx>=0 && y>=0 && y<600) putpixel( newx,y,couleur);
            if (newx<800 && newx>=0 && newy>=0 && newy<600) putpixel(newx,
            newy, couleur);
        }
    if (xo+R<800 && xo+R>=0) putpixel(xo+R,yo,couleur);
    if (xo-R<800 && xo-R>=0) putpixel(xo-R,yo, couleur);
}
```





*A gauche*, 200 cercles rouges de même centre et dont le rayon va de 0 à 199 en augmentant de 1 à chaque fois, ce qui donne un disque rouge. A droite les mêmes cercles, mais on colorie en bleu les pixels touchés par plus d'un cercle. On note qu'il n'y a aucun trou.

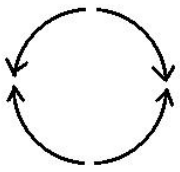
**3-4) Avantages de cette méthode de tracé**

- On a utilisé un algorithme différentiel : au lieu de calculer chaque fois *F* qui contient des termes au carré, on travaille sur la variation de *F*, qui ne fait intervenir que des termes du premier degré, avec seulement une multiplication par 2 et l'addition de 1, ce qui est très rapide.

- Lorsque l'on trace des cercles grossissants comme sur la figure ci-dessus, il ne se produit aucun trou, ce qui n'est pas le cas dans d'autres algorithmes (par exemple si l'on utilise les diagonales (comme pour la droite) au lieu de faire seulement des marches d'escalier).

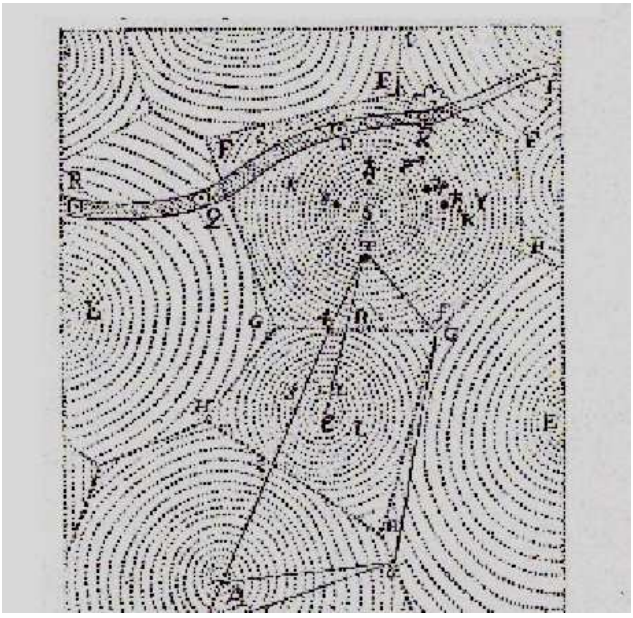
**3-5) Désavantage de la méthode**

Elle trace le cercle en quatre morceaux. Il n'y a pas continuité dans le tracé. Notamment si l'on veut tracer des arcs de cercle, il faudra s'y prendre autrement.



### 3-6) Exercice d'application : les cellules de Descartes

Considérons un ensemble de *N* points ou sites, situés dans un plan. Chacun de ces sites est entouré d'une surface dont tous les points sont plus près de ce site que des autres sites. Une telle surfaces entourant chaque site est appelée cellule de Voronoï. Ces cellules ont une forme polygonale, en général fermée sauf en bordure, et elles pavent le plan. Dans l'exemple le plus simple, avec deux sites *A* et *B* seulement, c'est la médiatrice de [*AB*] [1]qui forme la frontière entre les deux cellules occupant chacune un demi-plan. Dans le cas général, avec *N* sites, les traits bordant la cellule d'un site *A* sont aussi des morceaux de médiatrice entre *A* et quelques sites voisins. A l'image de certains phénomènes naturels, ces cellules peuvent être vues comme une sorte d'espace vital entourant équitablement chaque site.



Le nom de Voronoï, mathématicien russe des années 1900, est associé à ce type de cellules. Mais bien avant lui, vers 1640, R. Descartes donnait une façon de les construire lorsqu'il représentait la fragmentation du cosmos dans son livre *Le monde de René Descartes, ou Traité de la lumière*, comme indiqué sur son dessin ci-contre. C'est cette méthode que nous allons utiliser, à savoir la méthode des cercles grossissants.

*Algorithme*

On commence par prendre *N* points au hasard sur l'écran, de coordonnées (*x*[*i*], *y*[*i*]). Pour les voir, on dessine autour de chacun d'eux quelques cercles de rayon croissant. Les sites sont ainsi représentés par de petits disques tous de même rayon *rdebut*, avec une couleur identique (noire dans le programme, sur fond d'écran blanc) pour qu'on les voie jusqu'à la fin du processus. Puis à chaque étape de temps, on dessine autour de chacun des sites un cercle dont le rayon augmente de 1, et avec une couleur *color*[*i*] associée au site concerné de façon à les différencier. Le dessin de ces cercles successifs donne des disques qui entourent chaque site, tous de même rayon *r*[*i*] à chaque étape. Au cours du grossissement, quand un cercle associé à un site rencontre un disque entourant un autre site, on ne colorie plus sa partie qui empiète sur l'autre disque. C'est ainsi qu'apparaissent les bordures séparant les cellules voisines, à savoir les médiatrices à égale distance entre les sites.

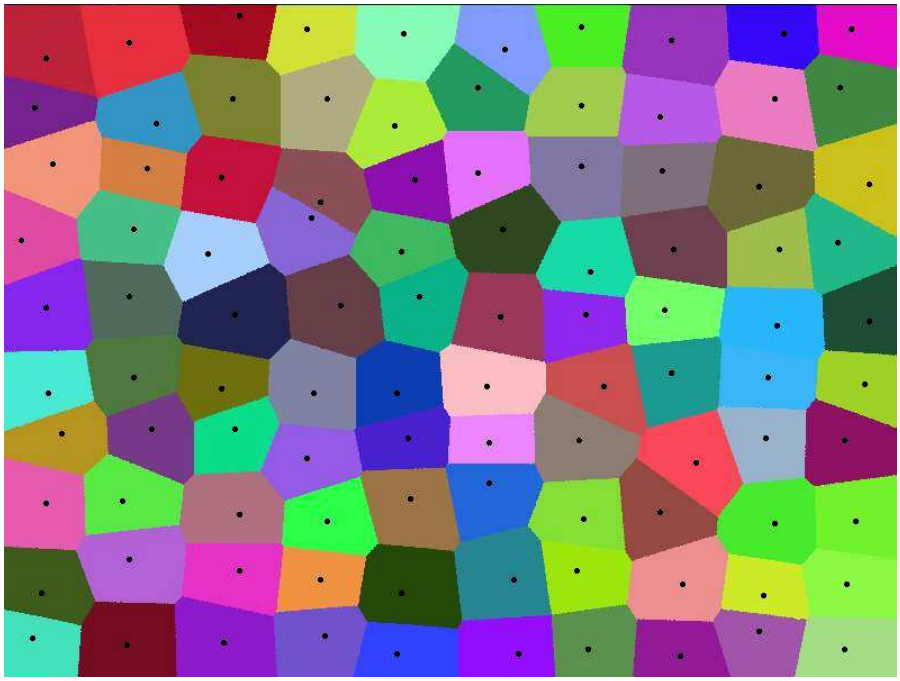
La fonction *cercle*() précédemment faite est utilisée pour tracer les petits disques des sites au départ avec le rayon *rdebut*. Pour les cercles grossissants qui vont devenir les cellules de Descartes, on utilise une nouvelle fonction *cercle* (*i*) où *i* est le numéro du site concerné. Dans cette fonction, on teste si le cercle a des points qui sont dessinés ou non sur l'écran. Si aucun point n'est dessiné, cela signifie que la cellule du site *i* est totalement dessinée, et l'on fait passer une

[1] La médiatrice d'un segment [*AB*] est la droite passant par le milieu de [*AB*] et perpendiculaire à [*AB*]. Elle a comme propriété caractéristique d'être formée par tous les points *M* situés à égale distance de *A* et de *B*, soit *MA* = *MB*.

variable *fini*[*i*] de 0 à 1. Quand *fini*[*i*] est à 1, le programme principal n'appelle plus la fonction cercle(*i*) pour les valeurs de *i* concernées. D'autre part, à chaque étape du processus, on calcule la somme des *fini*[*i*] que l'on place dans la variable *fin*. Lorsque *fin* atteint la valeur *N*, toutes les cellules sont construites, et le programme s'arrête.

```
se donner N         /* déclarations */
int xc[N], yc[N],rdebut=2, fin, fini[N],r[N],a; Uint32 color[N];
SDL_Surface * ecran; Uint32 rouge,blanc, noir;

void pause(void);    /* fonctions à  utiliser */
void putpixel(int xe, int ye, Uint32 couleur);
Uint32 getpixel(int xe, int ye);
void cercle( int xo, int yo, int R, Uint32 couleur) ;
void cercle2( int i);

int main(int argc, char ** argv) /* programme principal */
{   int i;
    SDL_Init(SDL_INIT_VIDEO);
    ecran=SDL_SetVideoMode(800,600,32, SDL_HWSURFACE|SDL_DOUBLEBUF);
    blanc=SDL_MapRGB(ecran->format,255,255,255); noir=SDL_MapRGB(ecran->format,0,0,0);
    SDL_FillRect(ecran,0,blanc);

    se donner xc[i], yc[i] et color[i] pour chaque site i
    for(i=0;i<N;i++) for(j=0; j<=rdebut; j++)    cercle(xc[i],yc[i],j,noir);
    for(i=0;i<N; i++) {fini[i]=0; r[i]=rdebut;}
    fin=0;
    while(fin!=N)  {fin=0;
                        for(i=0;i<N;i++) { fin+=fini[i];
                                              if (fini[i]==0) {r[i]++; cercle2(i);}
                                          }
                        }
    SDL_Flip(ecran); pause();  return 0;
}

void cercle2( int i)
 {int xo,yo,x, y, F, F1, F2,newx,newy,flag;
   xo=xc[i]; yo=yc[i]; flag=0;     x=xo; y=yo+r[i]; F=0;
   if (x<800 && x>=0 && y>=0 && y<600 && getpixel(x,y)==blanc)
      {putpixel(x,y,color[i]); flag=1;}
   if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600 && getpixel(x,2*yo-y)==blanc)
      {putpixel(x,2*yo-y, color[i]); flag=1;}
   while( y>=yo)
     { F1=F+2*(x-xo)+1; F2=F-2*(y-yo)+1;
       if ( abs(F1)<abs(F2))  { x+=1; F=F1;}   else {y-=1; F=F2;}
       if ( x<800 && x>=0 && y>=0 && y<600 && getpixel(x,y)==blanc)
         {putpixel(x,y,color[i]); flag=1;}
       newx=2*xo-x ; newy=2*yo-y ;
       if ( x<800 && x>=0 && newy>=0 && newy<600  && getpixel(x,newy)==blanc)
         {putpixel(x,newy,color[i]); flag=1;}
       if ( newx<800 && newx>=0 && y>=0 && y<600 && getpixel(newx,y)==blanc)
         {putpixel(newx,y,color[i]); flag=1;}
       if ( newx<800 && newx>=0 && newy>=0 && newy<600 && getpixel(newx,newy==blanc)
         {putpixel(newx,newy,color[i]);flag=1;}
     }
```

```
  if (flag==0) fini[i]=1;
 }
```



## 4) Quelques autres fonctions utiles

### 4-1) Tracé d'une droite ayant une certaine épaisseur

Il suffit de reprendre la fonction dessinant une flèche (*paragraphe* 2-6). On fait maintenant en sorte que les deux traits formant la flèche fassent un angle de 90° avec la droite (ou plutôt le segment). On obtient ainsi un trait perpendiculaire à la droite, et ayant une certaine largeur (*width*) de part et d'autre de la droite. Puis on fait bouger ce trait d'une extrémité à l'autre du segment. Ce balayage crée un segment ayant une certaine épaisseur. Et cette épaisseur est la même quelle que soit la direction de la droite.

```
void linewithwidth(int x1, int y1, int x2, int y2, int width,Uint32 c)
{
int dx,dy;
float k,xf1,yf1,xf2,yf2,d,dx1,dy1,ndx1,ndy1,ndx2,ndy2,angle=M_PI/2.;
ligne(x1,y1,x2,y2,c);
dx=x2-x1; dy=y2-y1;       d=sqrt(dx*dx+dy*dy);
if (d!=0.)      /* si le vecteur n'est pas nul */
{ dx1=(float)width*(float)dx/d; dy1=(float)width*(float)dy/d;
  ndx1=dx1*cos(angle)-dy1*sin(angle);
  ndy1=dx1*sin(angle)+dy1*cos(angle);
  ndx2=dx1*cos(-angle)-dy1*sin(-angle);
  ndy2=dx1*sin(-angle)+dy1*cos(-angle);
  for(k=0;k<=1.;k+=0.1/d)
  {
```

```
   xf1=(1.-k)*x1+k*x2; yf1=(1.-k)*y1+k*y2;
   xf2=xf1-ndx1; yf2=yf1-ndy1;  ligne(xf1,yf1,xf2,yf2,c);
   xf2=xf1-ndx2; yf2=yf1-ndy2;    ligne(xf1,yf1,xf2,yf2,c);
   }
}
}
```

### 4-2) Tracé d'une droite en marche d'escalier

Nous avons vu comment tracer un segment, grâce à la fonction *ligne*() vue au *paragraphe* 2-4. Cette méthode utilisait deux types de traits, dont l'un se faisait suivant une diagonale. On peut remplacer ce trait diagonal par deux traits en marche d'escalier, en rajoutant un point supplémentaire. La droite est alors construite grâce à des traits verticaux ou horizontaux. Pour ce faire, il suffit de modifier légèrement la fonction *ligne*(). Cette variante est fortement recommandée lorsque l'on utilise la fonction *floodfill* (voir *chapitre* 6), pour éviter les débordements.

```
void lignemarchesescalier(int x0,int y0, int x1,int y1, Uint32 c)
{
int dx,dy,x,y,residu,absdx,absdy,pasx,pasy,i;
dx=x1-x0; dy=y1-y0;  residu=0;       x=x0;y=y0;
 if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c);
if (dx>0) pasx=1;else pasx=-1; if (dy>0) pasy=1; else pasy=-1;
absdx=abs(dx);absdy=abs(dy);
if (dx==0) for(i=0;i<absdy;i++) { y+=pasy;
                                    if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c); }
else if(dy==0) for(i=0;i<absdx;i++){ x+=pasx;
                                      if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c);  }
else if (absdx==absdy)
  for(i=0;i<absdx;i++) {x+=pasx; if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c);
                         y+=pasy;
                         if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c);
                       }
else if (absdx>absdy)
   for(i=0;i<absdx;i++)
                       { x+=pasx; if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c);
                         residu+=absdy;
                         if(residu >= absdx) { residu -=absdx; y+=pasy;
                                               if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c);
                                               }
                       }
else for(i=0;i<absdy;i++)
        {y+=pasy; if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c);
         residu +=absdx;
         if (residu>=absdy)   {residu -= absdy;x +=pasx;
                               if (x>=0 && x<800 && y>=0 && y<600) putpixel(x,y,c);
                                }
        }
}
```