Initiation à SDL : et maintenant la liberté de dessiner

Rappelez-vous où on en était resté dans notre art de programmer grâce au langage *C CodeBlocks*. On pouvait voir s'afficher des millions de chiffres ou de caractères sur l'écran. Mais interdiction de dessiner un seul point à volonté, encore moins le moindre cercle ou la moindre image. Alors remettons la main dans le cambouis, et installons *SDL*,[1] plus précisément *SDL 1*, car il existe aussi un *SDL 2* plus récent. Cela va nous permettre de faire du graphisme, c'est-à-dire de mettre des images sur notre écran, ce qui est la moindre de choses.

Pourquoi continuer à utiliser *SDL1* alors que *SDL2* existe depuis quelques années ? Parce qu'il un plus simple à utiliser, et qu'il suffit largement à nos besoins pour la grande majorité des travaux graphiques que l'on a à réaliser. Par contre, si l'on veut aller beaucoup plus loin, et notamment utiliser les fonctionnalités actuelles d'*Opengl*, si lourdes soient-elles, il conviendra de passer à *SDL2*.

1. Téléchargement de *SDL 1* sous *Windows 10*

Demandez sur Google « télécharger SDL ». Dans la liste des possibilités, choisissez *Simple DirectMedia Layer – SDL version1.2.15, http//www.libsdl.org*. Dans la liste des propositions, aller dans la rubrique *developpement librairies* (bibliothèques de développement) et cliquez sur *SDL-devel-1.2.15-mingw32.tar.gz*. Le fichier compressé apparaît. Pour le décompresser, procédez à son « extraction » grâce à un logiciel comme *winrar* ou *7zip*. Puis par copier-coller, placez le dossier obtenu *SDL-1.2.15* à l'intérieur du dossier *CodeBlocks*. Il va ainsi apparaître juste au-dessous du dossier *MinGW*. C'est fini.[2]

Il est parfois conseillé d'aller dans la rubrique *runtime librairies* (bibliothèques d'exécution), et de télécharger *SDL-1.2.15-win32-x64.zip* pour avoir à notre disposition la *dll* : *SDL.dll*. Mais je ne pense pas que ce soit utile. Le fichier *SDL.dll* se trouve déja dans *CodeBlocks/SDL-1.2.15/bin*. En plus, il est intégré dans Windows : on devrait le trouver dans *c:/windows/system32*, et si par malchance il n'est pas présent dans la multitude des *dll*, il suffit de l'ajouter.

---

[1] Encore un peu de jargon : *SDL* est une *API*, une interface de programmation (*application programming interface*), c'est-à-dire pour les profanes une bibliothèque de fonctions, graphiques et autres, qui vont se greffer sur notre machine informatique, comme un *pace maker*. Ce n'est pas la seule possibilité de bibliothèque graphique, il y a aussi *Allegro*, *DirectX*, etc. si vous préférez.

[2] Dans les versions précédentes de *SDL* et de *Windows*, j'avais procédé de façon plus complexe. Dans le fichier *SDL* où se trouvent notamment les trois répertoires : *bin*, *include*, et *lib*, j'avais pris leurs contenus et les avaits copiés dans leurs équivalents respectifs *bin*, *include* et *lib* qui se trouvent dans *CodeBlocks\MinGW*. Notamment le répertoire *SDL* (soit *SDL\include\SDL*) qui est le seul contenu de *SDL/include* comme vous pouvez le constater, va se retrouver dans *CodeBlocks\MinGW\include*, et vous aurez les fichiers *.h* du *SDL* d'origine qui se retrouvent dans *CodeBlocks\MinGW\include\SDL*. Mais pour éviter certains ennuis que j'ai eu par la suite, j'ai copié une deuxième fois tous les fichiers *.h* de *SDL\include* directement dans *CodeBlocks\MinGW\include* (ceux-là mêmes qui étaient déjà dans *CodeBlocks\MinGW\include\SDL*). Enfin j'ai pris le fichier *SDL.dll* qui est dans le *SDL\bin* d'origine et je l'ai balancé dans *c:\windows\system32*.

2. Premier programme : l'écran noir

Retournons dans notre éditeur *Code Blocks*, et demandons de créer un nouveau projet, en faisant comme d'habitude : *File → New → Project*, et l'on on tombe sur la fenêtre où se trouvait *Console application*. Mais maintenant on choisit ***SDL project*** et l'on clique dessus. On va vous demander de donner un titre à votre projet, prenez par exemple *ecrannoir*, et juste après on vous demande où se trouve *SDL*. Ecrivez alors le chemin qui mène jusqu'à *SDL-1.2.15*, en utilisant par exemple le *browse* qui se trouve là, et quand vous tomber sur *SDL-1.2.15* vous demandez d'ouvrir et le chemin sera écrit. Puis continuez jusqu'à *finish*. Votre projet *ecrannoir* se trouve dans votre espace de travail (*workspace*). Tapez sur *Sources*, puis sur le *main.cpp* qui va apparaître, et qui est déjà rempli par un programme de base. Par acquit de conscience, exécutez ce programme qui vous est gracieusement offert pour constater que ça marche ! Le logo *Code Blocks* doit apparaître sur un fond d'écran noir.

Mais il se peut que cela ne marche pas sur votre ordinateur, par manque de liens vers certains fichiers. Dans ce cas, il convient de cliquer sur *Project*, puis sur *Build Options*. Une fenêtre apparaît.

Là choisissez *linker settings*. Si rien n'apparaît dans la fenêtre *link librairies* (même quand vous cliquez sur le nom de votre programme en haut à gauche), il va falloir faire du remplissage. En utilisant *add* à trois reprises, il faut placer dans la zone vide :

mingw32 , puis *SDLmain*, puis *SDL.dll*.

Cela étant fait, allez dans *Search Directories* (situé juste après *linker settings*). Dans la fenêtre *compiler*, appuyez sur *add* et cherchez grâce au *browser* (petit carré) l'emplacement de :

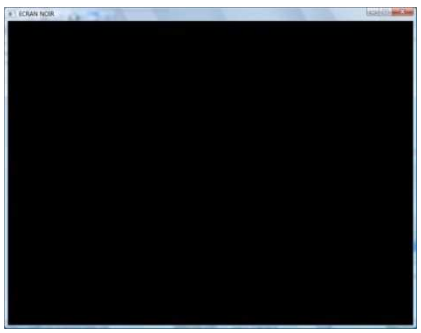
C:\Program Files\Codeblocks\MinGW\include et dites *oui*, puis *non* quand on vous demande si c'est un chemin relatif, puis *oui*. Ce dossier doit s'inscrire dans la fenêtre.

Ensuite, passez de *compiler* à *linker*, et faites de même pour que s'inscrive dans la fenêtre :

C:\Program Files\Codeblocks\MinGW\lib

C'est fini ! Revenez dans votre page *codeblocks*. Compilez et exécutez. Si cela ne marche toujours pas, ajouter la *SDL.dll* dans le dossier de votre programme. Cette *dll* se trouve dans un fichier *bin*, et elle devrait aussi être dans *c:\windows\system32* (si elle n'y est pas, mettez-la aussi).

Maintenant, le programme doit marcher, et afficher la fenêtre-écran en noir. Fermez cette fenêtre (avec la croix *X* en haut à droite) et jetez un œil sur le programme *main.cpp*. On n'y comprend rien et c'est tant mieux. On constate quand même un grand nombre de sécurités mises en place, juste pour faire peur. On va s'empresser de les supprimer ! Voilà ce que l'on peut se contenter de mettre dans notre premier programme, pour voir la fenêtre-écran apparaître, ici réduite à un rectangle noir :



```
#include <SDL/SDL.h>   /* dorénavant indispensable, pour disposer des fichiers  .h de SDL */

#define OUI 1        /* utilisé dans la fonction pause(), par souci pédagogique, car on peut s'en passer en
```

```
#define NON 0          metttant directement 0 et 1 dans la fonction pause() */
void pause(void) ;

int main (int argc , char ** argv)     /* écrire le main ainsi désormais */
{
SDL_Surface * screen ;    /* déclaration de la variable ecran, ou fenêtre, ici appelée screen */
SDL_Init(SDL_INIT_VIDEO) ;
screen= SDL_SetVideoMode(800, 600, 32, SDL_HWSURFACE) ;   /* fenêtre de 800 sur 600 */
SDL_WM_SetCaption(« ECRAN NOIR »,NULL) ;   /* facultatif : juste une fioriture  pour voir écrit
                                              ECRAN NOIR sur la bordure en haut de notre fenêtre écran*/
SDL_Flip(screen) ; /* indispensable pour voir un dessin apparaître */
pause() ; return 0 ;
}

void pause()
{  int continuer=OUI;
   SDL_Event   event; [3]
   while(continuer== OUI)
   {  SDL_WaitEvent(&event);
      if (event.type==SDL_QUIT) continuer=NON;
   }
}
```

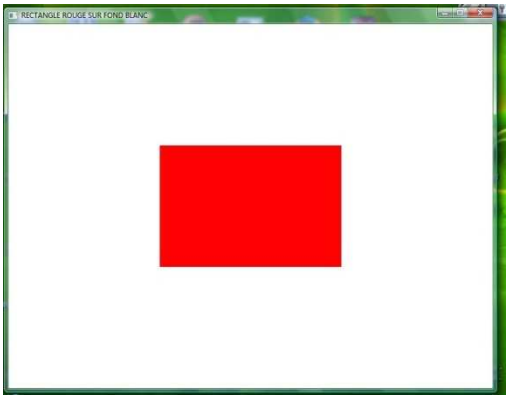
On commence par déclarer la variable *screen* (écran) comme étant l'adresse d'une surface, c'est-à-dire l'adresse de la première case que cette surface va occuper dans la mémoire de l'ordinateur.[4]

On met en conditions initiales l'installation du mode graphique, grâce à *SDL_Init*(*SDL_INIT_VIDEO*). Puis on demande la mise en place en mémoire de la fenêtre-écran, grâce à *SDL_SetVideoMode*, en indiquant que cette fenêtre va avoir une longueur de 800 pixels, une largeur de 600 pixels, et où la couleur de chaque pixel va occuper 32 bits. Cette fonction ramène l'adresse *screen*, indiquant la place où se trouve la fenêtre écran dans la mémoire, ce qui permettra d'accéder à celle-ci en cas de besoin. Comme on ne met rien dans cette zone de mémoire, on aura une fenêtre-écran noire sur notre écran d'ordinateur. Mais pas encore ! Si on ne fait rien d'autre, visiblement rien ne se passe. Encore faut-il balancer sur l'écran ce qui était jusque là dans la mémoire vidéo, d'où le *SDL_Flip*(*ecran*). Cela ne suffit toujours pas, la fenêtre-écran noire ne faisant qu'une apparition fugace avant de disparaître. Pour qu'elle reste figée sur l'écran de l'ordinateur, il convient d'ajouter une pause. D'où la fonction d'attente *pause*() où une boucle *while* ne cesse de tourner tant qu'un évènement extérieur ne se produit pas. Et le seul évènement qui est pris ici en compte est *SDL_QUIT*, qui se produira dès que nous cliquerons sur la petite croix en haut à droite de notre fenêtre écran. Alors seulement la boucle *while* s'arrête, et la fenêtre écran va finalement disparaître.

[3] Au lieu d'appeler cette variable *event*, vous pouvez l'appeler *evenement*, mais alors il faudra faire ce remplacement partout dans la suite, notamment mettre *WaitEvent(&evenement*), puis *if*(*evenement.type* …

[4] La programmation *SDL* utilise de nombreux *pointeurs*, à savoir des numéros de cases mémoires (des adresses). D'où l'apparition d'étoiles *, de & et de flèches -> (utiliser les touches – et >). Cela s'expliquera plus tard. Pour le moment se contenter de recopier !

3. Extension 1 : Plaquer une surface rectangulaire rouge sur l'écran



Mettons maintenant un peu de couleur. Dans un premier temps nous allons faire en sorte que la fenêtre-écran soit en blanc (au lieu de rester noire). Pour cela, après avoir déclaré la variable *blanc* (ou *white*) comme un entier sur 32 bits (*Uint32*), nous appelons la fonction *SDL_MapRGB* (screen->format, 255, 255, 255) où les trois 255 indiquent que les trois couleurs de base, rouge, vert et bleu, sont mises à leur valeur maximale, cette combinaison donnant alors du blanc. La fonction ramène le numéro donné à cette couleur blanche. Puis on demande de remplir le rectangle de la fenêtre-écran en blanc, grâce à *SDL_FillRect* (*screen*, *NULL*, *blanc*).

Rajoutons cela au programme précédent :

```
Uint32 blanc=SDL_MapRGB(ecran->format,255,255,255);
SDL_FillRect(screen,NULL,blanc);
```

et la fenêtre-écran est devenue blanche, après le *Flip* vers l'écran.

Allons plus loin. Nous voulons maintenant installer un rectangle rouge au centre de la fenêtre-écran blanche. Comme il s'agit toujours de rectangles, les fonctions précédentes vont encore nous servir. Mais maintenant il s'agit d'un nouveau rectangle (*newrectangle*) à l'intérieur de celui de l'écran et il convient de donner sa position. D'où l'apparition de nouvelles fonctions. Au lieu de *SDL_SetVideoMode*, comme on continue de le faire pour la fenêtre écran, on fait *SDL_CreateRGBSurface* (attention cette fonction possède huit arguments, les quatre derniers n'ayant pas d'intérêt pour nous sont mis à 0). Avec *position.x* et *position.y* on se donne les coordonnées du sommet en haut à gauche du nouveau rectangle. Tout cela est indiqué ci-dessous dans la partie mise en gras du programme. Une fois le nouveau rectangle installé, avec ses dimensions de 200 sur 100 ici, il convient de l'appliquer dans la position demandée sur la fenêtre-écran, d'où la fonction :

```
SDL_BlitSurface(newrectangle,NULL ,screen,&position)
```

Puis on balance le tout sur l'écran, grâce à *SDL_Flip(screen)* comme toujours. A la fin si l'on veut libérer la place prise en mémoire par le nouveau rectangle, on peut faire *SDL_FreeSurface*(). Voici le programme final :

```
#include <SDL/SDL.h>
#define OUI 1  /* par souci de pédagogie essentiellement */
#define NON 0
void pause();
int main (int argc, char ** argv )
{
   Uint32 rouge, blanc;
   SDL_Surface  * screen ,  * newrectangle;
   SDL_Rect   position;
   SDL_Init(SDL_INIT_VIDEO);
   screen=SDL_SetVideoMode(800, 600, 32,SDL_HWSURFACE);
   newrectangle= SDL_CreateRGBSurface(SDL_HWSURFACE,300,200,32,0,0,0,0);
   SDL_WM_SetCaption("RECTANGLE ROUGE SUR FOND BLANC",NULL); /* facultatif */
   blanc=SDL_MapRGB(screen->format,255,255,255);
   SDL_FillRect(screen,NULL,blanc);
```

```
    position.x=400-150;position.y=300-100 ;
    rouge=SDL_MapRGB(screen -> format, 255, 0, 0);
    SDL_FillRect(newrectangle,NULL,rouge);
    SDL_BlitSurface(newrectangle,NULL,screen,&position);

    SDL_Flip(screen);
    pause();    /* réutiliser la fonction déjà fabriquée dans le programme précédent */
    SDL_FreeSurface(newrectangle); /* libérer de la place, en fait sans intérêt dans ce programme */
    return 0;
}
```

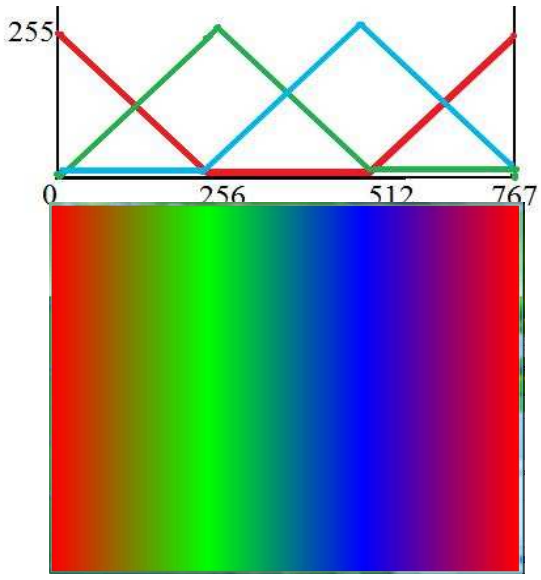
On peut ainsi plaquer (*Blit*) autant de rectangles que l'on désire sur la fenêtre-écran qui est déjà un rectangle. Ce que nous allons faire dans ce qui suit.

4. Extension 2 : Dégradé cyclique de couleurs sur l'écran

On va tracer des lignes verticales sur l'écran, chacune avec sa couleur. Une ligne verticale n'est autre qu'un rectangle de longueur 1 de largeur 600 (la hauteur de la fenêtre-écran). La longueur d'écran va être $3 \times 256 = 768$. Pour chaque valeur de *i* allant de 0 à 767, on va faire :

```
ligne[i]=SDL_CreateRGBSurface(SDL_HWSURFACE,1,600,32,0,0,0,0);
```

Et maintenant les couleurs



Comme indiqué sur les dessins ci-contre, chacune des trois couleurs *R*, *G*, *B* prend à tour de rôle la prédominance. Par exemple le rouge *R* démarre à 255 (rouge vif), les autres couleurs étant à 0, puis il diminue de un en un jusqu'à attendre la valeur 0 sur la colonne 256. Ensuite le rouge reste bloqué à 0 jusqu'à la colonne 512, puis il remonte de un en un jusqu'à 255 sur la dernière colonne 767. Les deux autres couleurs suivent la même évolution mais décalées de 256. On obtient finalement une palette cyclique de couleurs où les trois couleurs de base s'enchaînent avec des dégradés intermédiaires.

D'où cette partie de programme :

```
for(i=0;i<768;i++)
   {  if (i<256) couleur=SDL_MapRGB(screen->format,255-i,i,0);
      else if (i<512) couleur=SDL_MapRGB(screen->format,0,255-(i-256),i-256);
      else couleur=SDL_MapRGB(screen->format,i-512,0,255-(i-512));
   }
```

Dans cette même boucle, il reste à se donner la position de chaque colonne dont on connaît maintenant la couleur, puis à remplir la surface correspondante avec cette couleur, et enfin à plaquer (*blit*) cette colonne sur la fenêtre-écran. Le programme en découle :

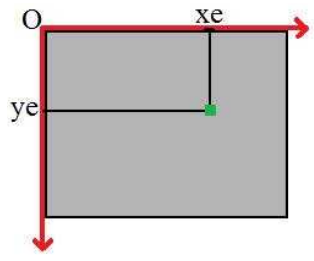
```
#include <SDL/SDL.h>
#define OUI 1
#define NON 0
void pause();
int main ( )
```

```
{  Uint32 couleur ;
   SDL_Surface *screen, *ligne[768] ;   /* ligne[] est un rectangle */
   SDL_Rect position;
   int i;

   SDL_Init(SDL_INIT_VIDEO);
   screen=SDL_SetVideoMode(768, 600, 32,SDL_HWSURFACE); /* exceptionnellement on a pris une*/
                                                        /* longueur de 768 au lieu de 800 */
   for(i=0;i<768;i++)
   ligne[i]=SDL_CreateRGBSurface(SDL_HWSURFACE,1,600,32,0,0,0,0);

   SDL_WM_SetCaption("DEGRADE SDL",NULL);

   for(i=0;i<768;i++)
   {
      position.x=i; position.y=0;
      if (i<256) couleur=SDL_MapRGB(screen->format,255-i,i,0);
      else if (i<512) couleur=SDL_MapRGB(screen->format,0,255-(i-256),i-256);
      else couleur=SDL_MapRGB(screen->format,i-512,0,255-(i-512));
      SDL_FillRect(ligne[i],NULL,couleur);
      SDL_BlitSurface(ligne[i],NULL,screen,&position);
   }

   SDL_Flip(screen);
   pause();    /* toujours cette même fonction à recopier */
   for(i=0;i<768;i++)  SDL_FreeSurface(ligne[i]);
   return 0;
}
```

On vient de voir comment notre logiciel *SDL* permet de tracer des rectangles à l'intérieur d'un rectangle d'écran. On pourrait généraliser cela au coloriage des pixels, un pixel n'étant autre qu'un rectangle de un sur un. Mais il vaut mieux s'y prendre autrement.

5. Dessiner des points

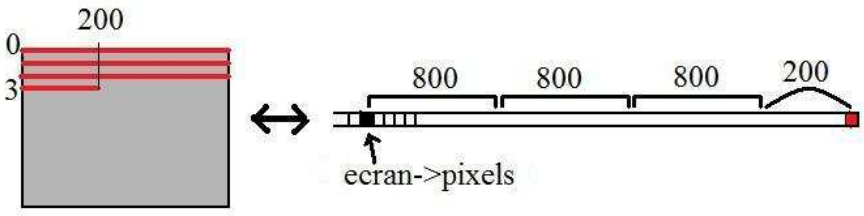
Comment attribuer une couleur donnée à un pixel de la fenêtre-écran situé à l'abscisse *xe* et l'ordonnée *ye* (rappelons que l'axe des *ye* est dirigé vers le bas) ? On va fabriquer pour cela la fonction *putpixel*(*int xe, int ye, Uint32 couleur*).



La fenêtre-écran, telle qu'on la voit, se trouve dans la mémoire de l'ordinateur, avec une certaine adresse initiale qui est *screen->pixels*, où *screen* provient de la mise en place du mode vidéo, comme on l'a déjà vu :

```
screen = SDL_SetVideoMode(800, 600, 32, SDL_SWSURFACE);
```

La mémoire ressemble à une rue très longue, bordée de cases ayant des adresses (seulement des numéros puisque la rue est unique) qui se suivent. Il s'agit de passer de la zone rectangulaire de l'écran à cette ligne de cases mémoires. Imaginons que l'on veuille donner la couleur *c* au point (200, 3). Ce point va se trouver dans la case numéro *screen->pixels* + 3 × 800 + 200, la longueur d'écran 800 étant aussi contenue dans *screen->w*. Il suffit de mettre *c* dans le contenu de cette case.




```
void putpixel(int xe, int ye, Uint32 couleur)
{ Uint32 * numerocase ;
  numerocase = (Uint32*)(screen->pixels) + xe + ye * screen->w ;
*numerocase = couleur;  /* la case numérotée par numerocase va contenir couleur. */
                        /* L'étoile * initiale indique  que l'on prend le contenu de la case dont le numéro */
                        /*  est numerocase */
}
```

6. Exemple 1 : brouillage d'écran

Nous allons dessiner une nuée de 10 000 points sur la fenêtre-écran, ces points étant placés au hasard, avec une couleur elle aussi aléatoire. Et nous allons répéter cela un millier de fois de façon à voir sur l'écran ce nuage de points en perpétuel mouvement, comme une sorte de brouillage tel qu'on le voit sur un téléviseur quand on n'arrive pas à capter une chaîne.

Pour la première fois nous utilisons une fonctionnalité déjà toute prête de *SDL* qui est le *double buffer*. Pendant qu'une image est affichée sur l'écran (grâce à *Flip*()), la suivante se préparée en arrière-plan en mémoire. D'où une succession fluide d'images, sans à-coups. On va avoir ici en succession : affichage de la nuée de points - effaçage de l'écran - nouvelle nuée - effaçage - ….

Pour cela il suffit d'annoncer au départ :

```
screen = SDL_SetVideoMode(800, 600, 32, SDL_SWSURFACE | SDL_DOUBLEBUF);
```

Le programme s'ensuit:

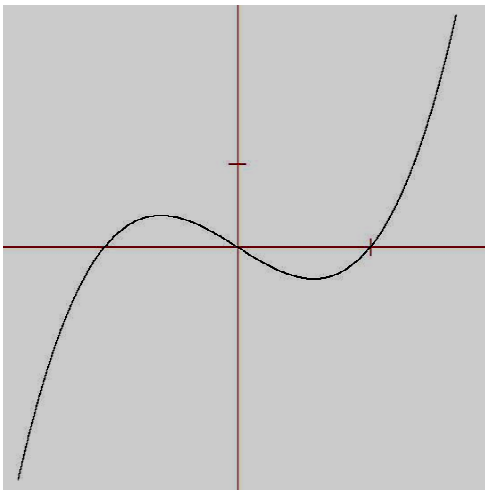
```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <SDL/SDL.h>
#define OUI 1
#define NON 0
void brouillage(void);
void putpixel(int xe, int ye, Uint32 couleur);
void pause(void);
SDL_Surface* screen;

int main (int argc, char ** argv )
{
   int i,noir;
   srand(time(NULL));
   SDL_Init(SDL_INIT_VIDEO);
   screen = SDL_SetVideoMode(800, 600, 32, SDL_SWSURFACE | SDL_DOUBLEBUF);
   SDL_WM_SetCaption("BROUILLAGE",NULL);
   noir=SDL_MapRGB(screen->format,0,0,0);
```

```
  for(i=0;i<1000;i++)
   {  brouillage();
      SDL_Flip(screen);
      SDL_FillRect(screen,NULL,noir);  /* effaçage de l'écran, en le coloriant en noir */
   }
  pause();
  return 0;
}

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

void brouillage(void)
{  int i,xe,ye;   Uint32 rouge,vert,bleu,color;
   for(i=0;i<10000;i++)
   {  xe=1+rand()%798; ye=1+rand()%598;   [5]
      rouge=rand()%256; vert= rand()%256; bleu=rand()%256;
      color=SDL_MapRGB(screen->format, rouge,vert,bleu);
      putpixel(xe,ye,color);  putpixel(xe+1,ye,color);  /* on dessine en fait 5 pixels pour faire
                                                            une petite croix  */
      putpixel(xe-1,ye,color);  putpixel(xe,ye+1,color);   putpixel(xe,ye-1,color);
   }
}
```

7. Exemple 2: Tracé d'une courbe

Maintenant que nous savons colorier des pixels sur l'écran, nous pouvons tracer des courbes point par point, à partir de leur équation sous la forme $y = f(x)$. Pour le moment, on se contente de cette méthode de tracé, même si, pour préserver la continuité du dessin, il convient de prendre un très grand nombre de points. La seule chose à faire est de passer de la zone calcul, celle de $y = f(x)$ où il s'agit de nombres à virgule de quelques unités au plus, à la zone écran où les coordonnées sont des nombres entiers pouvant atteindre des centaines d'unités, d'où la nécessité de procéder à un changement d'échelle, avec la mise en place d'un zoom.

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

SDL_Surface* screen;

void putpixel(int xe, int ye, int couleur);
void repere(void);
void courbe(void);
int xorig=400,yorig=300,zoomx=150,zoomy=100;
Uint32  rouge,blanc,noir;

int main (int argc, char* argv[])
{
   SDL_Init( SDL_INIT_VIDEO );
   screen = SDL_SetVideoMode(800, 600,
                                  32,SDL_HWSURFACE);
   rouge=SDL_MapRGB(screen->format, 255,0,0);
   blanc=SDL_MapRGB(screen->format, 255,255,255);
   noir =SDL_MapRGB(screen->format,0,0,0);
```



[5] L'abscisse *xe* va de 1 à 798. Comme on dessine en fait une petite croix, on aura des points à l'abscisse 0 et d'autres à 799, ce qui constitue les limites de la fenêtre-écran. Attention, si l'on fait par exemple *xe=1+rand()%799*, des points vont sortir de l'écran, et le programme va bloquer.

```
   SDL_FillRect(screen,0,blanc);
   repere();
   courbe();
   SDL_Flip(ecran);
   pause();   return 0;
}

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

void repere(void)
{
   int i;
   for(i=-300;i<300;i++) putpixel(xorig+i,yorig,rouge);  /* les deux axes */
   for(i=-300;i<300;i++) putpixel(xorig,yorig+i,rouge);
   for(i=-10;i<=10;i++) putpixel(xorig+zoomx,yorig+i,rouge);  /* graduations 1 */
   for(i=-10;i<=10;i++) putpixel(xorig+i,yorig-zoomy,rouge);
}

void courbe(void)
{
   float x,y; int xe,ye;
   for(x=-2.;x<2.;x+=0.001)
   {  y=x*x*x-x;          /* on a pris ici comme exemple de fonction  y = f(x) avec f(x) = x^3 – x */
      xe=xorig+(int)(zoomx*x); ye=yorig-(int)(zoomy*y);      /* passage de (x, y) à (xe, ye) */
      if (ye>20 && ye<580) putpixel(xe,ye,noir);  /* dessin dans les limites verticales de l'écran */
   }
}
```

Au point où nous en sommes arrivés, ce qui est dommage c'est de ne pas savoir tracer directement des collections de points qui finalement donnent des figures simples comme des droites ou des cercles. En attendant mieux, on peut toujours faire comme pour la courbe précédente, à savoir prendre les équations de ces courbes. Rappelons que l'équation d'une droite est de la forme $y = ax + b$, sauf si la droite est verticale, et que l'équation d'un cercle est $(x – xo)^2 + (y – yo)^2 = R^2$. Eh oui, ce qui se trouvait immédiatement disponible dans tous les langages classiques de programmation il y a trente ou quarante ans (droites, cercles, *sprites*, etc.) n'est pas présent d'office aujourd'hui, et doit être intégré par nos soins.

Heureusement de nouvelles fonctionnalités sont de nos jours disponibles, qui auparavant étaient bien plus complexes. Comme par exemple la possibilité de gérer des évènements, ce qui permet d'interagir sur ce qui se passe sur l'écran depuis l'extérieur, en cliquant sur la souris ou en appuyant sur des touches. Et surtout les langages *C* des années 2000 sont bien plus rapides que ceux des années 1980.

8. La gestion d'évènements

Jusqu'à présent le seul évènement que nous avions géré était dans la fonction *pause*() , avec *SDL_QUIT*. Le fait d'appuyer sur la croix en bordure de la fenêtre-écran, arrêtait l'exécution du programme, et en attendant cet évènement, la fenêtre restait immuablement présente. Reprenons la fonction *pause*() et ajoutons dans sa boucle *while*, juste après *if (event.type==SDL_QUIT) continuer=NON;*

```
else if (event.type== SDL_KEYDOWN && event.key.keysym.sym==SDLK_ESCAPE)
continuer = NON;
```

Maintenant le programme s'arrêtera aussi si l'on appuie sur la touche *Escape* (ou *Echap*) en haut à gauche du clavier. Mais on va faire mieux. En appuyant sur des touches, on va faire bouger un objet à volonté sur l'écran. Quel objet ? Un rectangle bien sûr, puisque *SDL* s'y prête bien.

Plus précisément nous allons prendre l'image du logo *Code Blocks* (avec ses quatre carrés accolés) dans le programme qui nous est offert chaque fois que l'on crée un nouveau projet. Cette image s'appelle *cb.bmp*, il suffit de la garder ou de la récupérer lorsque l'on fait notre projet actuel. On la charge comme on l'a fait pour la fenêtre-écran, mais en utilisant la fonction ad hoc :

```
codb = SDL_LoadBMP("cb.bmp");
```

après avoir declaré *codb* avec *SDL_Surface *codb;*. Cette surface rectangulaire possède maintenant une position (*position.x* et *position.y*) pour son coin en haut à gauche. On commence par faire en sorte que cette image *cb* se trouve au centre de l'écran, avec

```
position.x = (screen->w - codb->w) / 2; position.y = (screen->h - codb->h) / 2;
```

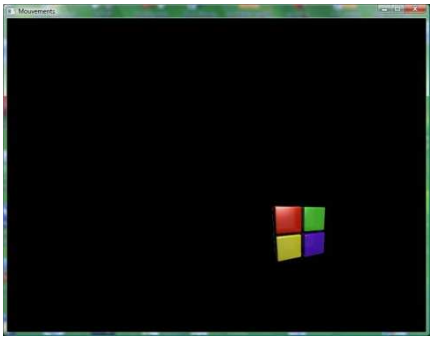
La grande nouveauté c'est que l'on va la faire bouger en appuyant sur des touches. Par exemple, en appuyant sur la touche → du clavier, l'image *cb* va se déplacer sur la droite. Pour cela, on s'inspire de notre ancienne fonction *pause*(), inutile dans le cas présent, mais dans la grande boucle *while*(*continuer== OUI*) qui constitue le corps du programme, on ne se contente plus des tests d'arrêt avec *SDL_QUIT* ou *SDL_KEYDOWN* avec *SDLK_ESCAPE*. Par exemple pour provoquer le mouvement à droite de l'image *cb* on fait :

```
if (event.type== SDL_KEYDOWN && event.key.keysym.sym==SDLK_RIGHT)
position.x +=2;
```

Cela implique que l'on mette à l'intérieur de la grande boucle *while* le placage de l'image *cb* sur la fenêtre-écran, d'abord en mémoire puis sur l'écran grâce à *Flip*(), tout cela étant réalisé par le double buffer. Le programme qui suit devrait maintenant être limpide.

```
#include <stdlib.h>
#include <SDL/SDL.h>
#define OUI 1
#define NON 0

int main ( int argc,char *argv[])
{   SDL_Surface *screen,*codb;
    SDL_Rect position;
    int continuer=OUI;
    SDL_Event event;

    SDL_Init( SDL_INIT_VIDEO );
    screen= SDL_SetVideoMode(800, 600, 32,SDL_HWSURFACE | SDL_DOUBLEBUF);
    codb = SDL_LoadBMP("cb.bmp");
    SDL_WM_SetCaption("Mouvements",NULL);  /* toujours facultatif */
    position.x = (screen->w - codb->w) / 2; position.y = (screen->h - codb->h) / 2;
    SDL_EnableKeyRepeat(10,10);    /* pour éviter le coup par coup et permettre de continuer d'avancer*/
                                   /*  quand on laisse la touche appuyée   */

    while (continuer==OUI)   /* la boucle attend des évènements extérieurs pour agir sur l'écran */
    {
       SDL_WaitEvent(&event);
       if (event.type==SDL_QUIT) continuer=NON;
       else if (event.type== SDL_KEYDOWN)
         if (event.key.keysym.sym==SDLK_ESCAPE) continuer=NON;
         else if(event.key.keysym.sym==SDLK_UP) position.y -=2;
```

```
        else if(event.key.keysym.sym==SDLK_DOWN) position.y +=2;
        else if(event.key.keysym.sym==SDLK_LEFT) position.x -=2;
        else if(event.key.keysym.sym==SDLK_RIGHT) position.x +=2;

      SDL_BlitSurface(codb, 0, screen, &position);       /* on plaque le logo CodeBlocks*/
      SDL_Flip(screen);      /* on balance le tout sur l'écran */
    }

    SDL_FreeSurface(codb);
    return 0;
}
```



et ça bouge à volonté !

Nous sommes maintenant en mesure de donner les principales fonctions graphiques permettant de tracer des points, des lignes, des cercles, etc. Si vous voulez vous contenter de les intégrer dans vos programmes sans savoir comment elles sont conçues, vous pouvez passer directement au *paragraphe 12*.

9. 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* (rappelons que l'origine du repère est en haut à gauche de la fenêtre écran).

- 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);
}
```

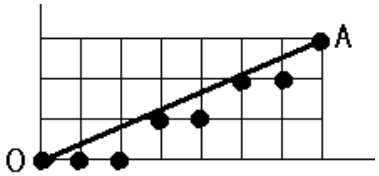
10. Tracé d'une ligne

10.1. 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 $\rightarrow$ ou diagonaux $\nearrow$. 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++)

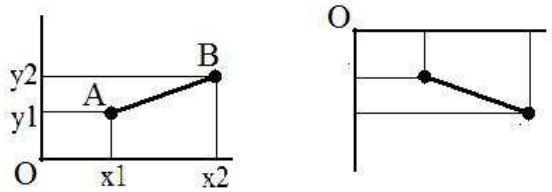
{x++ ; reste+=dy ; if (reste > =dx) {reste - = dx ; y ++ ;} *dessiner le point x,y*}

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.

10.2. 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.

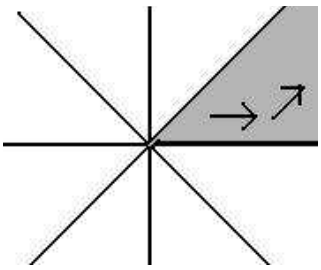
On se donne x1, y1, et x2, y2
dx=x2 – x1 ; dy = y2 – y1 ; residu = 0 ; *dessiner le point* (x1, y1)
for(i=0 ; i<dx ; i++)
{ x+=1 ; residu +=dy ; if (residu >=dx) {residu -=dx ; y+=1 ;}
dessiner le point courant (x,y)
}



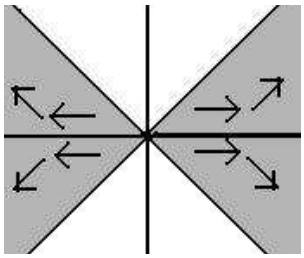
A gauche le segment *AB*, *à droite* ce que l'on voit sur l'écran

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.

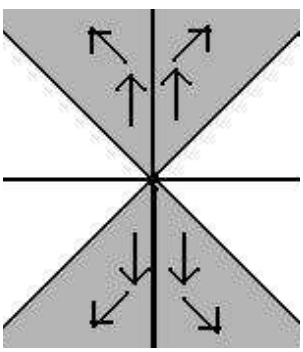
10.3. Cas général



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.

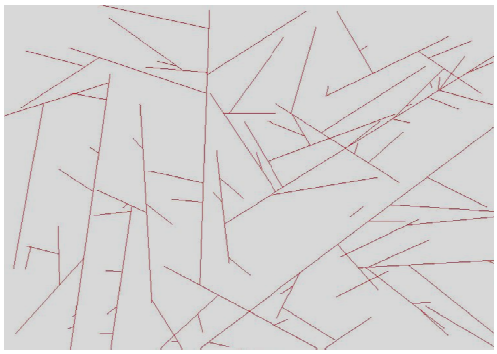
10.4. Programme du tracé

La fonction *line*(*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 line(int x0,int y0, int x1,int y1, Uint32 c);
SDL_Surface * screen;

int main(int argc, char ** argv)
{
   Uint32 rouge, blanc; int i;
   srand(time(NULL));
   SDL_Init(SDL_INIT_VIDEO);
   screen=SDL_SetVideoMode(800,600,32, SDL_HWSURFACE);
   rouge=SDL_MapRGB(screen->format,255,0,0);
   blanc=SDL_MapRGB(screen->format,255,255,255);
   SDL_FillRect(screen,0,blanc);

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

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

 /* fonction pause tant que l'on ne clique pas sur la x de la fenêtre ou qu'on n'appuie pas sur une touche */
void pause(void
{
   SDL_Event evenement;
   do
   SDL_WaitEvent(&evenement);
   while(evenement.type != SDL_QUIT && evenement.type != SDL_KEYDOWN);
}

void line(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 *)(screen->pixels)+xe+ye*ecran->w;
*numerocase=couleur;
}
```

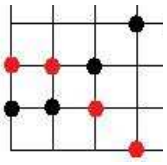


10.5. Remarques

- 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 *xorig* = 400 et *yorig* = 300, on fait *xe* = *xorig* + *x*, et *y* = *yorig* –*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 (*voir dessin ci-dessus*). Le programme *line*() 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 leurs représentations sur l'écran qui ne se coupent pas, dans le cas où elles se traversent entre deux segments diagonaux :


droites rouge et noire qui se croisent sans se toucher

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.

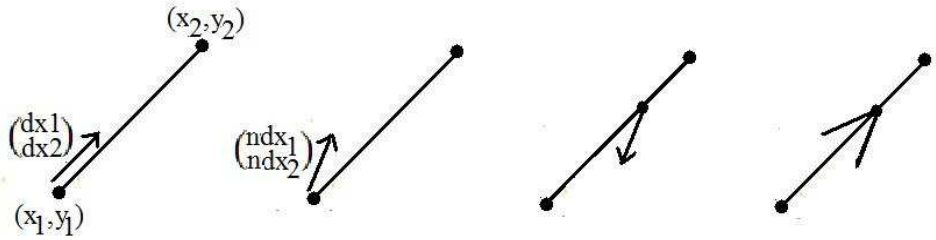
- Que l'on fasse ou non la rectification précédente en cas de besoin, il reste que deux droites sécantes en un point ont leur représentation sur l'écran qui peuvent se couper en plusieurs points, et d'autant plus que leurs pentes sont proches. Ces imperfections peuvent fausser les calculs. Dans de nombreux cas, on sera amené à travailler avec des nombres flottants pour faire les calculs de points d'intersection, et le dessin est fait ensuite sur l'écran en entiers, sans que cela ne 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. On doit constamment passer de la zone calcul à la zone 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 (*x*, *y*) de points de l'ordre de quelques unités, puis on fait un zoom pour dessiner les résultats (*xe*, *ye*) obtenus sur l'écran. Rappelons les formules de passage :

xe = *xorig* + *zoom***x* ;
ye = *yorig* – *zoom***y* ; où le point (*xorig*, *yorig*) est l'origine du repère sur l'écran.

10.6. 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 dont les coordonnées sont $dx = x_2 - x_1$ 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 (mais on peut modifier ce paramètre). 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 arrow(int x1, int y1, int x2, int y2)  /* arrow ou flèche */
{
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;     /* on peut modifier la position de la fèche */
  xf2=xf1-ndx1; yf2=yf1-ndy1;
  line(xf1,yf1,xf2,yf2);
  ndx2=dx1*cos(-angle)-dy1*sin(-angle);   ndy2=dx1*sin(-angle)+dy1*cos(-angle);
  xf2=xf1-ndx2; yf2=yf1-ndy2;    line(xf1,yf1,xf2,yf2);
}
else                              /* si le vecteur est nul, on peut dessiner un cercle du point vers lui-même, cela
                                     sera utile pour les graphes, mais en l'état actuel, on peut supprimer ce else */
    { cercle(x1+10,y1,10); line(x1+20,y1,x1+23,y1-6);
      line(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.

10.7. Tracé d'une droite ayant une certaine épaisseur

Il suffit de modifier légèrement le programme de la flèche. D'abord au lieu de prendre une flèche faisant un angle de 30° comme auparavant, on prend un angle de 90°, ce qui donne un trait perpendiculaire à la droite. Puis, au lieu de fixer ce trait comme on l'avait fait pour la flèche aux deux-tiers du segment, on le fait bouger d'une extrémité à l'autre du segment que l'on veut tracer, ce qui donne une ligne présentant une certaine épaisseur. D'où le programme :

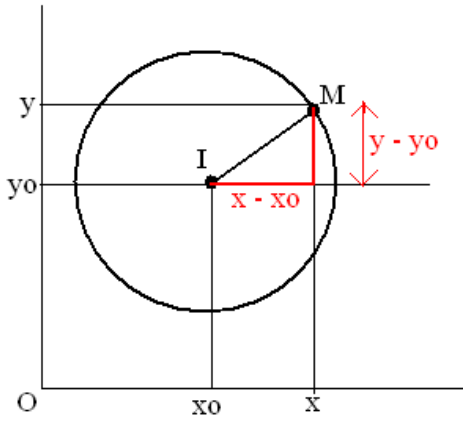
```
void droiteepaisse(int x1, int y1, int x2, int y2, int epaisseur,Uint32 c)  /* ou linewithwidth */
{
int dx,dy;
float k,xf1,yf1,xf2,yf2,d,dx1,dy1,ndx1,ndy1,ndx2,ndy2,angle=M_PI/2.;
line(x1,y1,x2,y2,c);
dx=x2-x1; dy=y2-y1;       d=sqrt(dx*dx+dy*dy);
if (d!=0.)      /* si le vecteur n'est pas nul */
  { dx1=(float)epaisseur*(float)dx/d; dy1=(float)epaisseur*(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)    /* le trait va parcourir le segment  */
     { xf1=(1.-k)*x1+k*x2; yf1=(1.-k)*y1+k*y2;
       xf2=xf1-ndx1; yf2=yf1-ndy1;  line(xf1,yf1,xf2,yf2,c);
       xf2=xf1-ndx2; yf2=yf1-ndy2;    line(xf1,yf1,xf2,yf2,c);
     }
  }
}
```

11. Tracé d'un cercle

11.1. 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 :

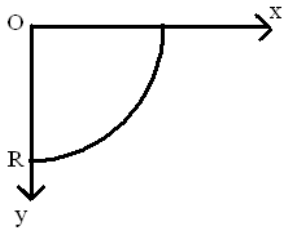
$(x - xo)^2 + (y - yo)^2 = R^2$ grâce au théorème de Pythagore.



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.

11.2. Tracé d'un quart de cercle de centre *O* et de rayon *R* avec *R* entier



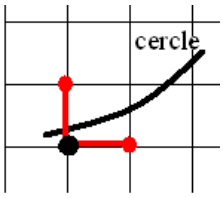
Plaçons-nous dans le repère de l'écran, d'origine *O* situé dans le coin en haut à gauche. On se donne un nombre entier *R* (par exemple $R = 100$), et l'on veut tracer sur l'écran le quart de cercle de centre *O* et de rayon *R*, dont l'équation est $F = 0$, soit ici $x^2 + y^2 - R^2 = 0$. Pour cela 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* le plus bas, 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;}
   }
```

11.3. 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, $yo + R$. Les valeurs de $F1$ et $F2$ sont :

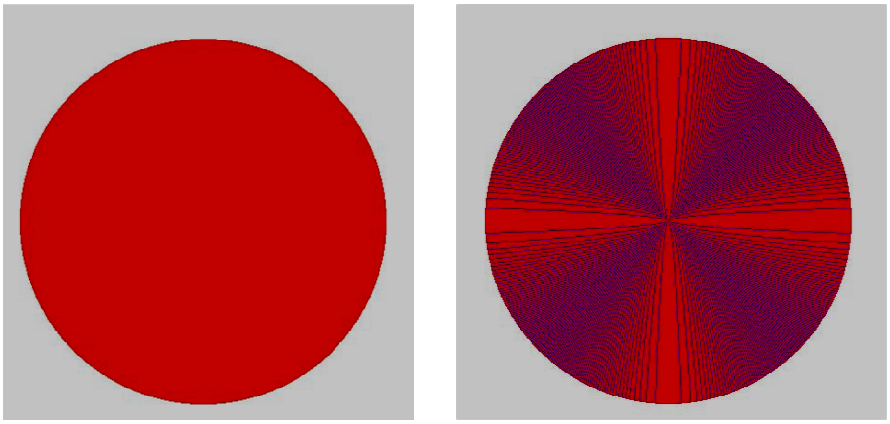
$F1 = (x + 1 – xo)^2 + (y – yo)^2 – R^2 = (x – xo)^2 + (y – yo)^2 – R^2 + 2(x – xo) + 1 = F + 2x + 1.$
F augmente de $2(x – xo) + 1$.

$F2 = (x – xo)^2 + (y – 1 – yo)^2 – R^2 = (x – xo)^2 + (y – yo)^2 – R^2 – 2(y – yo) + 1$
$= F – 2(y – yo) + 1.$ 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)   /* ou circle() */
 {
    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);
}
```



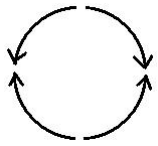
A gauche, 200 cercles rouges de même centre et dont le rayon va de 0 à 199 en augmentant de 1 à chaque fois, ce qui donne un disque rouge. A droite les mêmes cercles, mais on colorie en bleu les pixels touchés par plus d'un cercle. On note qu'il n'y a aucun trou, mais des superpositions.

11.4. Avantages et désavantages 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).

- Mais cette méthode a un désavantage : 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.

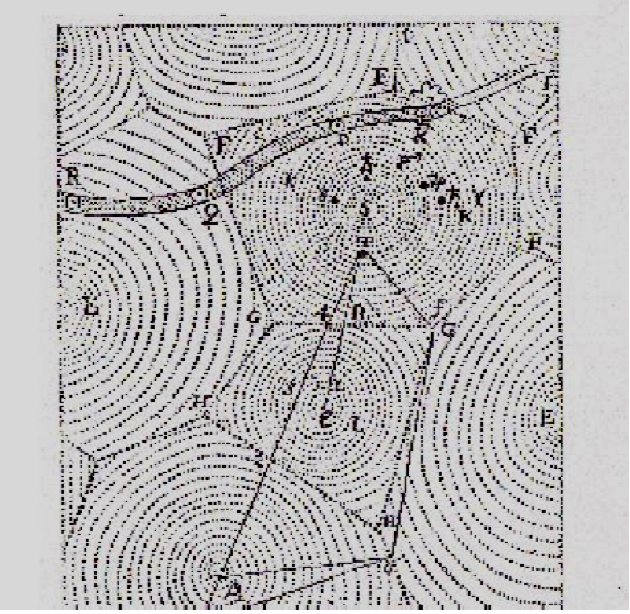


11.5. Exercice d'application : les cellules de Descartes

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 surface 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*] [6] 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-dessous. C'est cette méthode que nous allons utiliser, à savoir la méthode des cercles grossissants.



Dessin de R. Descartes

Algorithme

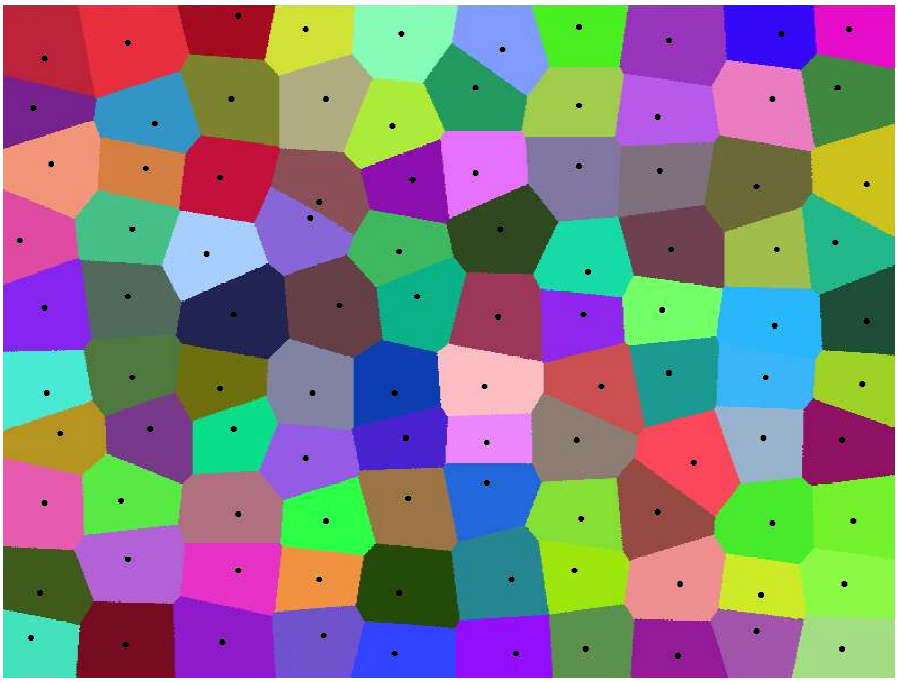
On commence par prendre *N* points au hasard sur l'écran, de coordonnées (*x*[*i*], *y*[*i*]). Pour les voir, on dessine autour de chacun d'eux quelques cercles de rayon croissant. Les sites sont ainsi représentés par de petits disques tous de même rayon *rdebut*, avec une couleur identique (noire dans 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 *cercle2* (*i*) où *i* est le numéro du site concerné. Dans cette fonction, on ne trace que les points pas encore coloriés (par d'autres cercles) et l'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 variable *fini*[*i*] de 0 à 1. Quand *fini*[*i*] est à 1, le programme

[6] 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*.

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 préliminaires à faire vous-même */
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);  /* écran sous fond blanc */

    se donner xc[i], yc[i] et color[i] pour chaque site i  (à faire vous-même)

    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;
 }
```



Cellules de Descartes autour de chacun des sites

12. Liste des fonctions graphiques en C avec SDL

Au stade où nous en sommes, nous sommes en mesure de donner les fonctions graphiques essentielles pour faire du dessin sous *SDL*. Leurs programmes sont donnés ci-dessous. Certains ont été expliqués précédemment, d'autres le sont dans le cours *Graphisme et géométrie*. Il conviendra de copier directement celles dont vous avez besoin en utilisant un convertisseur de documents *pdf*, puis de faire des copier-coller pour les placer à chaque fois dans vos propres programmes.

12.1. Ce à quoi elles servent

- *pause() : l'image est figée sur l'écran jusqu'à ce que l'on appuie sur une touche, ou sur la petite croix en haut à droite de la fenêtre.*

- *putpixel() : le pixel de coordonnées xe, ye est colorié sur l'écran avec la couleur c.*

- *getpixel() : ramène la couleur du pixel de coordonnées xe, ye.*

- *circle() : dessine un cercle de centre xo, yo, de rayon R, avec la couleur c.*

- *filldisc() : remplit le disque de centre xo, yo, de rayon R, avec la couleur c.*

- *line() : dessine une ligne entre les points (x0, y0) et (x1, y1) avec la couleur c. Deux fonctions sont disponibles. La première dessine la droite avec de petits traits horizontaux ou diagonaux. La deuxième dessine la droite avec des petits traits horizontaux et verticaux. Cette deuxième fonction doit être utilisée si l'on veut faire un floodfill dans une zone délimitée par des droites.*

- *rectangle() : dessine un rectangle dont le sommet en haut à gauche est (x1, y1) et le sommet en bas à droite (x2, y2), les côtés ayant la couleur c.*

- *arrow() : dessine un trait avec une flèche entre deux points (x1, y1, (x2, y2) avec la couleur c. Utilisée notamment pour les graphes, elle permet aussi de dessiner un petit cercle lorsque l'on va d'un point à lui-même.*

- *linewithwidth() : dessine une ligne entre deux points (x1, y1) et (x2, y2) avec la couleur c, et une certaine épaisseur width.*

- *floodfill() : remplit une surface délimitée par une courbe de bordure de couleur cb, à partir d'un point intérieur à la surface(x, y), avec une couleur de remplissage cr. Le point (x, y) joue le rôle de source et la couleur cr s'étale sur la surface par écoulement, jusqu'à la bordure de couleur cr. Si la bordure contient une ou des lignes, utiliser pour les tracer la fonction line() en marches d'escalier.*

- *arc…() : ces fonctions permettent de tracer des arcs de cercles dans certains cas particuliers mais pas dans le cas général. Il s'agit seulement des arcs de cercles occupant un quart de cadran ou un demi cadran.*

Attention : ne pas utiliser les variables de ces fonctions (style x0, xo, etc.) comme variables globales.

12.2. Leurs programmes

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

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

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

void circle( int xo, int yo, int R, Uint32 c)
 {
    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,c);
    if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600) putpixel (x,2*yo-y, c);
    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,c);
         newx=2*xo-x ; newy=2*yo-y ;
        if (x<800 && x>=0 && newy>=0 && newy<600) putpixel(x, newy,c);
        if (newx<800 && newx>=0 && y>=0 && y<600) putpixel( newx,y,c);
        if (newx<800 && newx>=0 && newy>=0 && newy<600) putpixel(newx,
        newy, c);
      }
     if (xo+R<800 && xo+R>=0) putpixel(xo+R,yo,c);
     if (xo-R<800 && xo-R>=0) putpixel(xo-R,yo, c);
 }

 void filldisc( int xo, int yo, int R, Uint32 c)
 {
    int x, y, F, F1, F2,newx,newy,xx;
    x=xo; y=yo+R; F=0;
    if (x<800 && x>=0 && y>=0 && y<600) putpixel(x,y,c);
    if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600) putpixel (x,2*yo-y, c);
    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;}
            newx=2*xo-x ; newy=2*yo-y ;
            for(xx=newx; xx<=x; xx++)if (xx<800 && xx>=0 && y>=0 && y<600 )
            putpixel(xx,y,c);
            for(xx=newx; xx<=x; xx++)if (xx<800 && xx>=0 && newy>=0 && newy<600 )
            putpixel(xx,newy,c);
          }
        if (xo+R<800 && xo+R>=0&& y>=0 && y<600) putpixel(xo+R,yo,c);
        if (xo-R<800 && xo-R>=0&& y>=0 && y<600) putpixel(xo-R,yo, c);
    }

    void line(int x0,int y0, int x1,int y1, Uint32 c)
    {
      int dx,dy,x,y,residu,absdx,absdy,stepx,stepy,i;
      dx=x1-x0; dy=y1-y0;  residu=0;       x=x0;y=y0;  putpixel(x,y,c);
      if (dx>0) stepx=1;else stepx=-1; if (dy>0) stepy=1; else stepy=-1;
      absdx=abs(dx);absdy=abs(dy);
      if (dx==0) for(i=0;i<absdy;i++) { y+=stepy;   putpixel(x,y,c); }
      else if(dy==0) for(i=0;i<absdx;i++){ x+=stepx;   putpixel(x,y,c);  }
      else if (absdx==absdy)
      for(i=0;i<absdx;i++) {x+=stepx; y+=stepy;  putpixel(x,y,c); }
      else if (absdx>absdy)
      for(i=0;i<absdx;i++)
         { x+=stepx; residu+=absdy;
           if(residu >= absdx) {residu -=absdx; y+=stepy;}
           putpixel(x,y,c);
         }
      else for(i=0;i<absdy;i++)
           {y+=stepy; residu +=absdx;
            if (residu>=absdy)   {residu -= absdy;x +=stepx;}
            putpixel(x,y,c);
           }
    }
```

/* *stair line, must be used for floodfill (ligne en marches d'escalier, à utiliser quand on fait un floodfill avec des bordures en ligne droite, ou lorsque l'on veut que deux droites sécantes se coupent bel et bien)* */

```
    void line(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);
                                 }
          }
}

void rectangle(int x1,int y1, int x2, int y2, Uint32 c)
{
   line(x1,y1,x2,y1,c);line(x1,y2,x2,y2,c);line(x1,y1,x1,y2,c);line(x2,y2,x2,y1,c);
}

void arrow(int x1, int y1, int x2, int y2, Uint32 c)
{
int dx,dy;
float xf1,yf1,xf2,yf2,d,dx1,dy1,ndx1,ndy1,ndx2,ndy2,angle=M_PI/6.;
line(x1,y1,x2,y2,c);
dx=x2-x1; dy=y2-y1;       d=sqrt(dx*dx+dy*dy);
if (d!=0.)
{ dx1=6.*(float)dx/d; dy1=6.*(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;
  line(xf1,yf1,xf2,yf2,c);
  ndx2=dx1*cos(-angle)-dy1*sin(-angle); ndy2=dx1*sin(-angle)+dy1*cos(-angle);
  xf2=xf1-ndx2; yf2=yf1-ndy2;    line(xf1,yf1,xf2,yf2,c);
}
else
    {circle(x1+10,y1,10,c); line(x1+20,y1,x1+23,y1-6,c);
     line(x1+20,y1,x1+15,y1-5,c);
    }
}

/* ajouter  #include <math.h> */
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.;
line(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;  line(xf1,yf1,xf2,yf2,c);
  xf2=xf1-ndx2; yf2=yf1-ndy2;    line(xf1,yf1,xf2,yf2,c);
  }
}
}

void floodfill( int x,int y, Uint32 cr,Uint32 cb)
{   int xg,xd,xx;
```

```
   if (getpixel(x,y) !=cb && getpixel(x,y) !=cr)
   { putpixel(x,y,cr);
     xg=x-1;
     while(xg>0 && getpixel(xg,y)!=cb)  {putpixel(xg,y,cr); xg--;}
     xd=x+1;
     while(xd<800 && getpixel(xd,y)!=cb)  {putpixel(xd,y,cr); xd++ ;}
     for(xx=xg; xx<xd;xx++)
       { if (y>1 ) {floodfill(xx,y-1,cr,cb);}
         if (y<599 ) {floodfill(xx,y+1,cr,cb);}
       }
   }
}

void arc0_90( int xo, int yo, int R, Uint32 c)  /* arc de 0° à 90°, quart de cercle dans le premier cadran */
 {
    int x, y, F, F1, F2,newx,newy;
    x=xo; y=yo+R; F=0;
    if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600) putpixel (x,2*yo-y, c);
    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;}
          newy=2*yo-y ;
         if (x<800 && x>=0 && newy>=0 && newy<600) putpixel(x, newy,c);
      }
     if (xo+R<800 && xo+R>=0) putpixel(xo+R,yo,c);
 }

 void arc270_360( int xo, int yo, int R, Uint32 c)
 {
    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,c);
    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,c);
      }
     if (xo+R<800 && xo+R>=0) putpixel(xo+R,yo,c);
 }

void arc90_180( int xo, int yo, int R, Uint32 c)
 {
    int x, y, F, F1, F2,newx,newy;
    x=xo; y=yo+R; F=0;
    if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600) putpixel (x,2*yo-y, c);
    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;}
          newx=2*xo-x ; newy=2*yo-y ;
         if (newx<800 && newx>=0 && newy>=0 && newy<600) putpixel(newx,
         newy, c);
      }
     if (xo-R<800 && xo-R>=0) putpixel(xo-R,yo, c);
 }
```

```
void arc180_270( int xo, int yo, int R, Uint32 c)
 {
    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,c);

    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;}
          newx=2*xo-x ;
         if (newx<800 && newx>=0 && y>=0 && y<600) putpixel( newx,y,c);
      }
     if (xo-R<800 && xo-R>=0) putpixel(xo-R,yo, c);
 }

void arc0_180( int xo, int yo, int R, Uint32 c)  /* demi-cercle */
 {
    int x, y, F, F1, F2,newx,newy;
    x=xo; y=yo+R; F=0;

    if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600) putpixel (x,2*yo-y, c);
    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;}
         newx=2*xo-x ; newy=2*yo-y ;
         if (x<800 && x>=0 && newy>=0 && newy<600) putpixel(x, newy,c);
         if (newx<800 && newx>=0 && newy>=0 && newy<600) putpixel(newx,
         newy, c);
      }
     if (xo+R<800 && xo+R>=0) putpixel(xo+R,yo,c);
     if (xo-R<800 && xo-R>=0) putpixel(xo-R,yo, c);
 }

 void arc180_360( int xo, int yo, int R, Uint32 c)
 {
    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,c);
    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,c);
          newx=2*xo-x ;
         if (newx<800 && newx>=0 && y>=0 && y<600) putpixel( newx,y,c);
      }
     if (xo-R<800 && xo-R>=0) putpixel(xo-R,yo, c);
 }

void arc270_90( int xo, int yo, int R, Uint32 c)
 {
    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,c);
```

```
    if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600) putpixel (x,2*yo-y, c);
    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,c);
          newy=2*yo-y ;
        if (x<800 && x>=0 && newy>=0 && newy<600) putpixel(x, newy,c);
      }
     if (xo+R<800 && xo+R>=0) putpixel(xo+R,yo,c);
  }

void arc90_270( int xo, int yo, int R, Uint32 c)
  {
    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,c);
    if (x<800 && x>=0 && 2*yo-y>=0 && 2*yo-y<600) putpixel (x,2*yo-y, c);
    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;}
         newx=2*xo-x ; newy=2*yo-y ;

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

Remarque

Pour toutes les fonctions précédentes, la fenêtre écran (*screen*) est supposée de dimension 800 sur 600 pixels, comme cela est fait dans la déclaration :

```
screen=SDL_SetVideoMode(800,600,32, SDL_HWSURFACE|SDL_DOUBLEBUF);
```

Nous avons aussi choisi le double buffer *SDL_DOUBLEBUF*, afin de rendre fluide les animations d'images en cas de besoin.

La fonction *putpixel*() n'étant pas sécurisée, il convient de maintenir le point concerné à l'intérieur strict de la fenêtre écran, sinon on aura une erreur fatale. Par contre les fonctions traçant des lignes ou des cercles sont sécurisées, les dessins pouvant déborder de l'écran.

13. Structure générale d'un programme sous C et SDL

Voici comment va se présenter un programme en C-SDL.

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

#define … éventuellement
```

```
void pause(void);
void putpixel(int xe, int ye, Uint32 c);   /* au cas où l'on utilise putpixel et getpixel */
Uint32 getpixel(int xe, int ye);

et declarations d'autres fonctions, éventuellement

SDL_Surface * screen;
Uint32 white, black, couleur[100];   déclarations des couleurs

déclaration de variables globales, éventuellement

int main(int argc, char ** argv)
{
   déclaration des variables locales

   SDL_Init(SDL_INIT_VIDEO);
   screen=SDL_SetVideoMode(800,600,32, SDL_HWSURFACE|SDL_DOUBLEBUF);

   /* exemples de couleurs avec leurs trois composantes RGB */
   couleur[0]=SDL_MapRGB(screen->format,255,255,255);   /* white */
   couleur[1]=SDL_MapRGB(screen->format,255,0,0);    /* red */
   couleur[2]=SDL_MapRGB(screen->format,0,250,0);   /* green */
   SDL_FillRect(screen,0,couleur[0]);  /* donne un fond blanc à la fenêtre */

   ….. mettre ici le programme concerné

  SDL_Flip(screen);  /* Cette fonction affiche l'image issue du programme sur l'écran. Si on ne le fait pas,
                        on ne verra rien. On sera souvent amené à l'utiliser plusieurs fois dans le programme pour
                        voir  ce qui se passe  */
  pause();return 0;
}

Placer ici les fonctions concernées, comme pause(), putpixel(), getpixel() ...
```

Conclusion : utilisez à fond les copier-coller d'un programme à un autre.