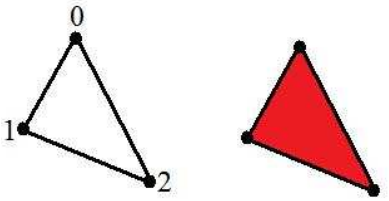
# Remplissage de l'intérieur d'un polygone ou d'une surface

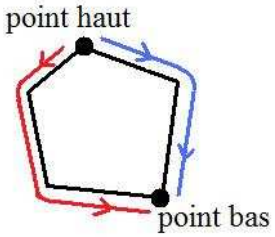
Considérons une surface fermée délimitée par des segments, comme celle d'un triangle ou d'un quadrilatère, d'un polygone convexe [1] en général. L'objectif est de colorier cette surface située à l'intérieur de la bordure polygonale. En anglais la fonction correspondante est appelée *fill poly*.

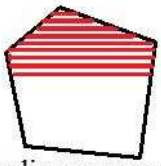



triangle et son remplissage (en rouge)

Comment faire ?

- On commence par chercher le sommet le plus haut et le sommet le plus bas de la bordure polygonale.
- Ensuite on définit deux chemins, partant tous deux du point haut pour se terminer au point bas, et parcourant les côtés du polygone, l'un par la gauche, et l'autre par la droite.
- Ces deux chemins sont parcourus de façon synchrone. Chaque fois que l'on descend d'un cran sur chacun des deux chemins, on trace la ligne horizontale entre le point du chemin gauche et le point du chemin droit. La surface est ainsi remplie grâce à ces traits horizontaux.

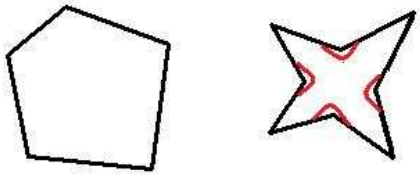


remplissage progressif
par des traits horizontaux

Dans ce contexte, la fonction qui va nous permettre d'avancer sur une ligne droite en faisant ressortir les moments où l'on descend d'un pas verticalement joue un rôle

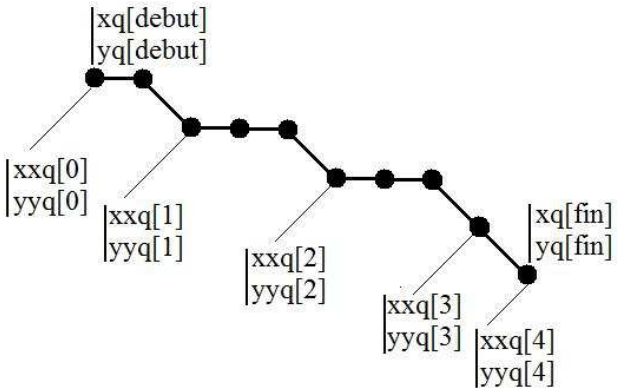
[1] Un polygone est convexe lorsque tous ses angles sont saillants (forment des pointes). Un polygone non convexe est dit simple lorsqu'il a des angles rentrants et que ses côtés ne se recoupent pas. La méthode de remplissage exposée ici ne concerne que les polygones convexes (mais on a vu comment découper en triangles un polygone simple non convexe).



A gauche un polygone convexe, à droite un polygone simple avec la présence de quatre angles rentrants.

décisif. Nous avons vu précédemment comment tracer une ligne droite, par une succession de pas horizontaux et diagonaux dans le cas d'une pente faible, ou de pas verticaux et diagonaux pour une pente forte. Il s'agissait de la fonction *ligne*(*x*0, *y*0, *x*1, *y*1, *c*). C'est ce programme que nous allons reprendre et aménager, pour aboutir à la nouvelle fonction *line*(*debut*, *fin*, *c*).

Les sommets du polygone sont numérotés 0, 1, 2, … et leurs coordonnées enregistrées dans les tableaux *xq*[ ] et *yq*[] placés en global. Dans la fonction *line*(*debut*, *fin*, *c*), *debut* et *fin* désignent les numéros des points entre lesquels on va tracer la ligne, et *c* désigne la couleur de la ligne, qui sera aussi celle du remplissage. A son tour la fonction se charge de dessiner tous les points de la ligne, et elle enregistre certains de ces points, chaque fois qu'il y a une descente verticale d'un cran. Ces points d'ordonnées croissantes (on est dans le repère écran dirigé verticalement vers le bas) sont placés dans les tableaux *xxq*[] et *yyq*[] eux aussi déclarés en global. Sur le dessin ci-contre, on a le cas d'une droite de pente faible, avec les points concernés.




On en déduit le programme de la fonction *line*(), après avoir déclaré en global les tableaux *xq*, *yq* (que l'on remplit), ainsi que *xxq* et *yyq* , avec la variable *kq* mise à 0.

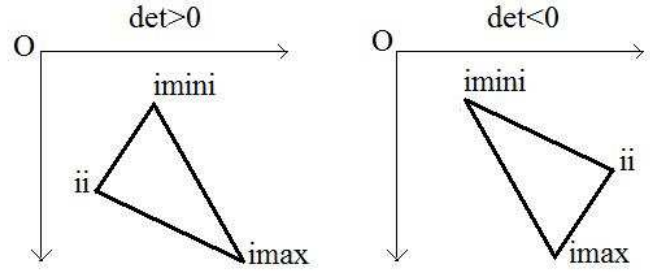
```
void line(int debut,int fin, Uint32 c)
{   int dx,dy,residu,absdx,absdy,pasx,pasy,i,xxx;
    dx=xq[fin]-xq[debut];  dy=yq[fin]-yq[debut];   residu=0;
    xxq[kq]=xq[debut];  yyq[kq]=yq[debut];  putpixel(xxq[kq],yyq[kq],c);
    if (dx>0) pasx=1;  else pasx=-1;   if (dy>0) pasy=1;   else pasy=-1;
    absdx=abs(dx);   absdy=abs(dy);
    if (dx==0) for (i=0;i<absdy;i++)
       {  kq++;  yyq[kq]=yyq[kq-1]+pasy;  xxq[kq]=xxq[kq-1];  putpixel(xxq[kq],yyq[kq],c);}
    else if (dy==0) for (i=0;i<absdx;i++)  {   putpixel(xxq[kq]+pasx*i,yyq[kq],c);  }
    else if (absdx==absdy)
       for (i=0;i<absdx;i++)
       {  kq++;xxq[kq]=xxq[kq-1]+pasx;yyq[kq]=yyq[kq-1]+pasy;
          putpixel(xxq[kq],yyq[kq],c);     }
    else if (absdx>absdy)    /* pente faible */
    {  xxx=xxq[kq];
       for (i=0;i<absdx;i++)
       {  xxx+=pasx;  residu+=absdy;
          if (residu<absdx) putpixel(xxx,yyq[kq],c);  else if (residu >= absdx)
          {   residu -=absdx;
              kq++;  yyq[kq]=yyq[kq-1]+pasy; xxq[kq]=xxx; putpixel(xxq[kq],yyq[kq],c);}
       }
    }
    else for (i=0;i<absdy;i++)   /* pente forte */
       {  kq++; yyq[kq]=yyq[kq-1]+pasy; xxq[kq]=xxq[kq-1];
          residu +=absdx;
          if (residu>=absdy)
```

```
            {   residu -= absdy;   xxq[kq] =xxq[kq-1]+pasx; putpixel(xxq[kq],yyq[kq],c);}
            else putpixel(xxq[kq],yyq[kq],c);
        }
}
```

Passons maintenant au remplissage du polygone, en commençant par le cas le plus simple, celui du triangle.

## Coloriage de l'intérieur d'un triangle

On se donne les trois sommets 0, 1, 2 du triangle, avec leurs coordonnées respectives. Puis on cherche le sommet le plus haut, celui avec l'ordonnée minimale, dont le numéro correspondant est noté *imini*. A son tour, le sommet le plus bas a son numéro placé dans *imax*. Le troisième point a son numéro placé dans *ii*. Deux cas sont alors possibles : le point *ii* est à gauche, ou bien à droite, du côté joignant les points imini et *imax*. Pour distinguer ces deux cas, on calcule le déterminant (noté *det*) correspondant aux vecteurs ***imini imax***, celui-ci étant pris en premier, et ***imini ii*** pris en second. Si ce déterminant est positif, le point *ii* se trouve à gauche du segment *imini imax*, et sinon il est à droite (voir à ce sujet le chapitre sur l'orientation et les angles).




Prenons par exemple le cas où *det*<0. Pour le cheminement de gauche, on utilise la fonction *line*() permettant d'aller des points *imini* à *imax* en enregistrant les points en descente d'un cran à chaque fois dans les tableaux *xxq*[] et *yyq*[]. Les indices correspondants à ces points sont mis dans *kq*, avec *kq* partant de 0 pour se terminer à *kfin2*. Puis les contenus de ces deux tableaux sont transférés dans les tableaux *xgauche*[] et *ygauche*[]. On prend ensuite le cheminement de droite, d'abord des points *imini* à *ii*, avec la fonction *line*() associée, puis de *ii* à *imax* avec sa fonction *line*(). L'indice variable *kq* démarre à 0 pour atteindre *kfin* en arrivant au point *ii*, puis il continue d'augmenter pour atteindre la valeur *kfin2* en arrivant au point *imax*. Il y a autant de points dans le chemin de gauche que dans celui de droite. Une fois les tableaux des chemins gauche et droite remplis (soit *xgauche*[] *ygauche*[] et *xxq*[] *yyq*[]), il suffit de les parcourir en même temps pour tracer les lignes horizontales joignant les points de gauche aux points de droite, en utilisant la couleur *c*. D'où le programme :

```
void remplirtriangle(Uint32 c)
{
   int mini,max,imini,imax,ii,kk,dx,dy,i,kfin,kfin2,;  int vx,vy,vvx,vvy;
   float det;
   mini=1000;  max=-1;
   for (i=0;i<3;i++)
   {  if (yq[i]<mini)  {   mini=yq[i];  imini=i;  }
      if (yq[i]>max)  {   max=yq[i];   imax=i;  }  /* calcul de imini et imax */
```

```
    }
    kk=0;
    for (i=0;i<3;i++) if (i!=imini && i!=imax) ii=i;  /* obtention de ii */
    dx=xq[imax] - xq[imini];  dy=yq[imax]-yq[imini];
    vx=xq[ii]-xq[imini];  vy=yq[ii]-yq[imini];  det=dx*vy-dy*vx;
    if (det<0)   /* déterminant négatif, ii est à droite */
    {
        kq=0;  line(imini,imax,c);   kfin2=kq;
        for (i=0;i<=kfin2;i++) {  xgauche[i]=xxq[i];  ygauche[i]=yyq[i];  }
        kq=0;  line(imini,ii,c);   kfin=kq;
        kq=kfin;   line(ii,imax,c);  kfin2=kq;
        for (i=0;i<=kfin2;i++) {  lignehori(xgauche[i],ygauche[i],xxq[i],c);  }
    }
     else   /* déterminant positif ou nul, ii est à gauche */
    {
        kq=0;  line(imini,ii,c);   kfin=kq;
        kq=kfin;  line(ii,imax,c);  kfin2=kq;
        for (i=0;i<=kfin2;i++) {  xgauche[i]=xxq[i];  ygauche[i]=yyq[i];  }
        kq=0;  line(imini,imax,c);  kfin2=kq;
       for (i=0;i<=kfin2;i++) {   lignehori(xgauche[i],ygauche[i],xxq[i],c);   }
    }
}

void lignehori(int xd,int yd, int xf,Uint32 c)
{  int xxx;  for (xxx=xd+1;xxx<xf; xxx++) putpixel(xxx,yd,c);  }
```

Rappelons que ce programme ne fonctionne que si les tableaux *xq*[], *yq*[], *xxq*[], *yyq*[], *xgauche*[], *ygauche*[], ainsi que la variable *kq*, sont préalablement déclarés en global.

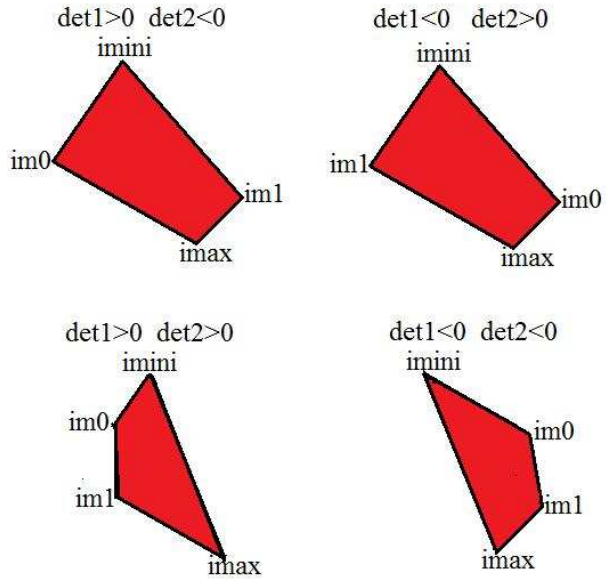
## Remplissage d'un quadrilatère convexe

Au départ on se donne un quadrilatère convexe, dont les sommets numérotés de 0 à 3 sont lus en tournant dans le sens inverse des aiguilles d'une montre (si l'on fait cela c'est surtout pour pouvoir facilement dessiner le quadrilatère avec ses quatre côtés pris en succession, mais ce n'est pas notre problème ici). Les coordonnées des sommets sont placées dans un tableau des abscisses *xq*[4] et un tableau des ordonnées *yq*[4]. Comme précédemment, il convient de trouver le point le plus haut, *imini*, et le point le plus bas *imax*. Mais ensuite, avec les deux autres points, numérotés *im*[0] et *im*[1], quatre cas sont possibles [2] , comme indiqué sur la figure ci-dessous. On peut avoir les points *im*[0] et *im*[1] de part et d'autre de la ligne *imini imax*, ce qui donne deux cas selon que *im*[0] est à gauche ou à droite. On peut aussi avoir les cas où les points *im*[0] et *im*[1] sont tous deux à gauche de la ligne *imini imax*, ou tous deux à droite. Dans ces deux derniers cas, il faut aussi déterminer si le point *im*[0] est au-dessus ou au-dessous de *im*[1]. On procède à un échange éventuel de façon que ce soit *im*[0] qui soit le point le plus haut des deux.

En calculant les deux déterminants *det1* et *det2*, leur signe indique si *im*[0] et *im*[1] sont à gauche ou à droite, par exemple *det1*>0 signifie que *im*[0] est à gauche de *imini imax* et *det2*>0 indique que *im*[1] est à gauche. On connaît ainsi dans chaque cas le

[2] Si le quadrilatère n'était pas convexe, ce serait plus compliqué.

chemin de gauche et le chemin de droite menant de *imini* à *imax*. Ce qui permet de faire le programme de la fonction *remplirquadri*(*c*), comme on l'avait fait pour le triangle.




```
#include <SDL/SDL.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
void pause(void);     /*  fonctions déjà connues, à ajouter */
void putpixel(int xe, int ye, Uint32 couleur);
Uint32 getpixel(int xe, int ye);
void line(int debut,int fin, Uint32 c);
void lignehori(int xd,int yd, int xf,Uint32 c);
void remplirquadri(Uint32 c);  /* la nouvelle fonction */
SDL_Surface * ecran; Uint32 rouge,blanc;
int xq[4],yq[4], kq, xxq[10000],yyq[10000],xgauche[10000],ygauche[10000];

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

   Se donner les quatre points 0, 1, 2, 3 sommets du quadrilatère convexe (avec leurs
    coordonnées dans xq[] et yq[]

    SDL_FillRect(ecran,0,blanc);
    remplirquadri(rouge );
    SDL_Flip(ecran); pause();   return 0;
}

void remplirquadri(Uint32 c)
{  int mini,max,imini,imax,im[2],kk,dx,dy,i,kfin,kfin2,kfin3;
    int vx,vy,vvx,vvy,aux; float det1,det2;
    mini=1000; max=-1;
    for(i=0;i<4;i++)
```

```
    { if (yq[i]<mini) {mini=yq[i]; imini=i;}  if (yq[i]>max) {max=yq[i]; imax=i;} }
  kk=0;      for(i=0;i<4;i++) if (i!=imini && i!=imax) im[kk++]=i;
  dx=xq[imax] - xq[imini]; dy=yq[imax]-yq[imini];
  vx=xq[im[0]]-xq[imini]; vy=yq[im[0]]-yq[imini];   det1=dx*vy-dy*vx;
  vvx=xq[im[1]]-xq[imini]; vvy=yq[im[1]]-yq[imini];  det2=dx*vvy-dy*vvx;
  if( det1*det2<=0)
      {if (det2<=0)   /* on a alors det1>=0  */
          { kq=0; line(imini,im[0],c);kfin=kq;kq=kfin;line(im[0],imax,c);kfin2=kq;
            for(i=0;i<=kfin2;i++) {  xgauche[i]=xxq[i];ygauche[i]=yyq[i]; }
            kq=0; line(imini,im[1],c);kfin=kq;kq=kfin;line(im[1],imax,c);kfin2=kq;
            for(i=0;i<=kfin2;i++) {  lignehori(xgauche[i],ygauche[i],xxq[i],c);}
          }
       else if (det1<=0)
          { kq=0; line(imini,im[1],c);kfin=kq;kq=kfin;line(im[1],imax,c);kfin2=kq;
            for(i=0;i<=kfin2;i++) { xgauche[i]=xxq[i];ygauche[i]=yyq[i]; }
            kq=0; line(imini,im[0],c);kfin=kq;kq=kfin;line(im[0],imax,c);kfin2=kq;
            for(i=0;i<=kfin2;i++) { lignehori(xgauche[i],ygauche[i],xxq[i],c);  }
          }
      }
   else if ( det1<0 && det2<0)
      {
        if (yq[im[0]]>yq[im[1]]) {aux=xq[im[0]]; xq[im[0]]=xq[im[1]]; xq[im[1]]=aux;
                                  aux=yq[im[0]]; yq[im[0]]=yq[im[1]]; yq[im[1]]=aux; }
        kq=0; line(imini,imax,c);kfin3=kq;
        for(i=0;i<=kfin3;i++) {  xgauche[i]=xxq[i];ygauche[i]=yyq[i];  }
        kq=0; line(imini,im[0],c);kfin=kq;
        kq=kfin;line(im[0],im[1],c);kfin2=kq;
        kq=kfin2;  line(im[1],imax,c);kfin3=kq;
        for(i=0;i<=kfin3;i++) { lignehori(xgauche[i],ygauche[i],xxq[i],c); }
      }
   else if ( det1>0 && det2>0)
      {
        if (yq[im[0]]>yq[im[1]]) {aux=xq[im[0]]; xq[im[0]]=xq[im[1]]; xq[im[1]]=aux;
                                  aux=yq[im[0]]; yq[im[0]]=yq[im[1]]; yq[im[1]]=aux;  }
         kq=0; line(imini,im[0],c);kfin=kq;
         kq=kfin;line(im[0],im[1],c);kfin2=kq;
         kq=kfin2;  line(im[1],imax,c);kfin3=kq;
        for(i=0;i<=kfin3;i++)  {  xgauche[i]=xxq[i];ygauche[i]=yyq[i];  }
         kq=0; line(imini,imax,c);kfin3=kq;
         for(i=0;i<=kfin3;i++) {  lignehori(xgauche[i],ygauche[i],xxq[i],c);  }
      }
}
```

L’utilité de ce programme apparaîtra notamment lorsqu’il s’agira de colorier les facettes d’un objet en trois dimensions. Son défaut est de donner lieu à une erreur fatale au cas où deux des quatre sommets du quadrilatère seraient confondus. Mais cela peut facilement être corrigé, en cas de besoin. Ce que nous avons fait pour les triangles et les quadrilatères convexes pourrait être généralisé à des polygones quelconques. Mais nous en resterons là, considérant que ce que nous avons fait suffit à répondre à la grande majorité des problèmes posés.

## Remplissage d'une surface par écoulement

Nous allons pour finir donner une autre fonction classique, *flood fill* –remplissage par écoulement-, qui permet de subvenir aux cas les plus généraux, où la bordure de la forme à remplir n'est plus seulement un polygone, mais une courbe.

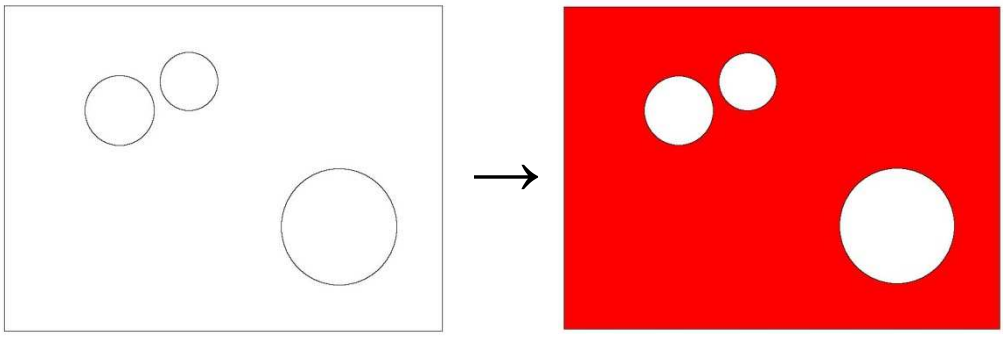
Au départ, on a une courbe fermée, délimitant une surface, cette courbe étant coloriée avec la couleur *cb*. On se donne aussi un point source(*x, y*) à l'intérieur strict de la surface. L'objectif est de colorier la surface avec une couleur donnée *cr*. Sous sa forme rudimentaire, la programmation de cette fonction consiste à partir du point source en le coloriant avec la couleur *cr*, puis à se rappeler sur ses quatre voisins dans le quadrillage de l'écran, sous réserve que ces points ne soient pas déjà coloriés, ni avec la couleur de remplissage, ni avec la couleur de la bordure. On assiste alors une sorte d'écoulement, d'épanchement autour de la source, celui-ci s'arrêtant lorsqu'il atteint la bordure de la forme. Une telle méthode récursive a l'avantage d'une extrême simplicité, mais le fait de se rappeler quatre fois risque de provoquer la saturation de la pile où sont emmagasinés les résultats en attente. Si on l'applique à une surface trop grande, on assistera inexorablement à un blocage définitif.

```
void floodfill( int x,int y, Uint32 cr,Uint32 cb)
{
   if (getpixel(x,y) !=cb && getpixel(x,y) !=cr)
   {   putpixel(x,y,cr);
       floodfill(x+1,y,cr,cb); floodfill(x-1,y,cr,cb); floodfill(x,y+1,cr,cb); floodfill(x,y-1,cr,cb);
   }
}
```

Nous allons donc donner une deuxième version moins gourmande en récursivité.

A partir du germe (*x, y*) supposé situé dans la surface à colorier, on prend son voisin de gauche (*x* - 1,*y*) et à partir de là on trace la ligne horizontale jusqu'à ce qu'on atteigne la bordure, le dernier point colorié étant noté (*xg, y*). Puis on fait de même vers la droite, à partir du point (*x* + 1, *y*) jusqu'au point touchant la frontière, noté (*xd, y*). Puis pour chaque point de cette ligne horizontale , entre *xg* et *xd* pour les abscisses, on rappelle la fonction *floodfill*() sur son voisin supérieur et son voisin inférieur. Comme il suffit d'un seul rappel pour remplir une ligne horizontale, les rappels sont peu nombreux et l'on peut ainsi remplir un rectangle remplissant l'écran 800 x 600 sans blocage.

Un exemple et le programme :



```
#include <SDL/SDL.h>
#include <math.h>
#define deuxpi 6.28318
#define N 5
#define rayon 20
void pause(void);
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 ligne(int x0,int y0, int x1,int y1, Uint32 c);
void rectangle(int x1,int y1, int x2, int y2, Uint32 c);
void floodfill( int x,int y, Uint32 cr,Uint32 cb);
void floodfill2( int x,int y, Uint32 cr,Uint32 cb);
SDL_Surface * ecran; Uint32 blanc,noir,rouge;

int main(int argc, char ** argv)
{   int x,y;
    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);
    rouge=SDL_MapRGB(ecran->format,255,0,0);
    SDL_FillRect(ecran,0,blanc);

    rectangle(0,0, 799,599,noir);
    cercle(  220,200,60,noir);    cercle( 340,150,50,noir);
    cercle( 600,400,100,noir);SDL_Flip(ecran);pause();
    x=150; y=150;
    floodfill2( x,y, rouge,noir);


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

void floodfill2( 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 ) {floodfill2(xx,y-1,cr,cb);}
      if (y<599 ) {floodfill2(xx,y+1,cr,cb);}

    }
    }
}
```