Le clipping

Michel Buffa, UNSA, UFR Sciences, 1997

1 Introduction

Jusqu'à présent, nous ne nous sommes pas tellement inquiétés de la position des objets que nous désirons visualiser. En effet, tant que nous avons utilisé les primitives graphiques de X11, aucun problème n'est intervenu, XDrawLine ou XFillPolygon fonctionnent vite et savent traiter les cas ou les lignes ou les polygones que l'on désire dessiner se trouvent entirement ou partiellement en dehors de l'écran.

Oui, mais comment faire du tramage, du lissage de Gouraud ou du texture mapping avec les fonctions X11 ? Ce n'est pas possible, et il a bien fallu se résoudre à écrire nos propres fonctions de remplissage de facettes.

Aïe ! Déjà ça n'est pas facile d'écrire une fonction de remplissage qui soit à la fois rapide et qui ne laisse pas de "trous", mais en plus il faut traiter le cas où les objets peuvent sortir de l'écran... gare aux "segmentation fault" et autres "core dumps" si on sort des limites du buffer image !

Comment faire ? Une solution évidente consiste à faire des tests avant de dessiner chacun des points du polygone. Le point courant est-il dans l'écran ? Cela revient à effectuer 4 tests pour chacun des pixels (il faut tester par rapport aux 4 côtés de l'écran)! Ce n'est pas bien malin car un polygone contient de très nombreux pixels. Et puis, nous n'avons pas envie de toucher à nos fonctions de remplissage. Nous avons eu déjà bien du mal à les écrire!

Si vous avez bien suivi mes conseils, votre fonction de remplissage ressemble à ceci :

```
- Calculer Ymax et Ymin à partir des coordonnées Y des n sommets du
  polygone
-
- Initialiser scanLineTab entre Ymin et Ymax
-
- Pour chaque côté du triangle
   - Mettre à jour scanLineTab
- FinPour
-
- Pour Y de Ymin à Ymax
   - Pour X de scanLineTab[Y].xgauche à scanLineTab[Y].xdroit
      - dessiner le pixel(X, Y)
   - FinPour
- FinPour
```

Comme vous le voyez, cette fonction prend en entrée un polygone de n côtés, parcourt chacun des côtés, met à jour le tableau scanLineTab (contenant les Min et Max des attributs de chacun des sommets interpolés le long de chaque côté), puis parcourt le tableau pour dessiner chaque pixel de chaque ligne.

Le clipping va intervenir en amont de cette fonction. Ainsi, le polygone clippé sera un polygone comme un autre pour cette fonction, qui demeurera inchangée.

Avant de poursuivre plus en avant, nous allons étudier comment fonctionne le clipping 2D pour les primitives simples.

2 Clipping d'un point

Le clipping dans le cas d'un pixel est trivial, il suffit de quelques comparaisons pour décider si les coordonnées d'un pixel sont dans les limites de l'écran:

```
DrawPoint(int x, int y, unsigned char color)
{
  if((x >= 0)&&(x < LARGEUR_ECRAN) && (y >= 0)&&(y < HAUTTEUR_ECRAN)) {
      image[y*LARGEUR_ECRAN + x] = color;
  }
}
```

3 Clipping d'une droite

On pourrait, lorsqu'on dessine une droite, appeler la fonction DrawPoint pour chaque pixel appartenant à la droite, mais cela n'est pas bien malin, car nous introduisons des tests dans une boucle. Argh! Des tetss dans une boucle, la chose à éviter coûte que coûte si l'on ne veut pas grèver les performances!

Nous allons construire une fonction qui prend en entrée les extrêmités d'un segment de droite, et qui retourne les coordonnées de la partie de la ligne qui se trouve dans l'écran. *Ce traitement est le fameux clipping*.

Clipper revient à calculer les coordonnées de point(s) d'intersection entre le segment de droite et les bords de l'écran.

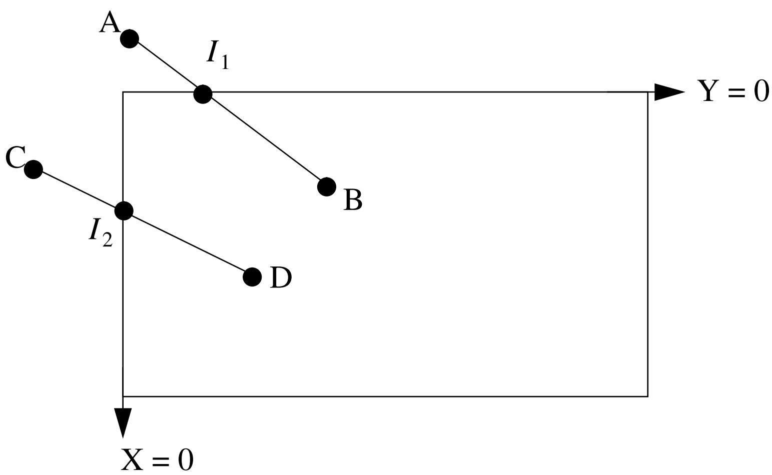


FIGURE 1. Plusieurs intersections possibles

Un des problèmes est que nous ne savons pas exactement avec lequel des 4 côtés il y a une intersection.

Par exemple, avec l'exemple de la FIGURE 1., le segment AB intersecte les bords de l'écran avec le côté $Y = 0$, alors que le segment CD coupe le côté $X = 0$.

Pour ne plus se poser ce problème, nous clipperons dans un premier temps un segment avec les bords verticaux, et dans un second temps avec les bords horizontaux.

Autre exemple :

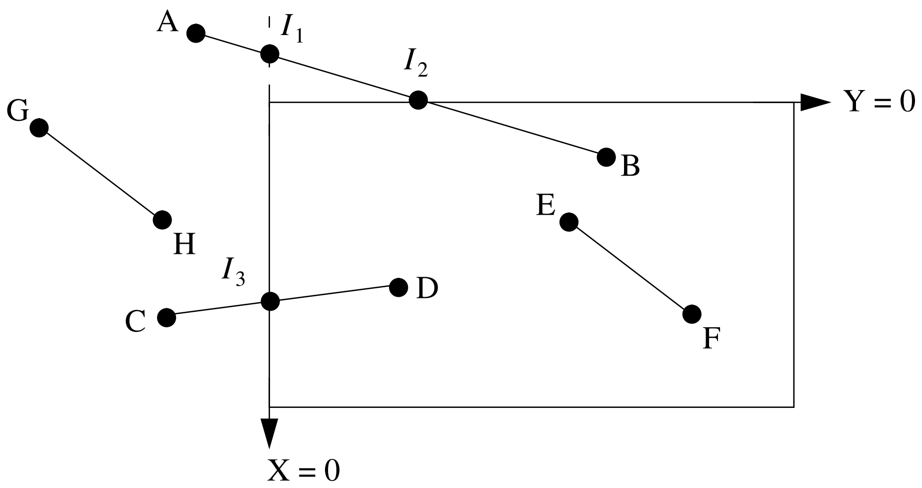


FIGURE 2. Différents cas

Dans l'exemple de la FIGURE 2., après un premier appel à la routine de clipping (bords verticaux), le segment AB devient I_1B, aprés le second appel (bords horizontaux), I_1B devient I_2B qui est sa forme finale.

On peut voir aussi que certains segments ne seront pas du tout clippés, comme le segment EF, clippés juste une fois, comme CD, ou encore clippés deux fois, comme AB. Finalement, le segment GH est entièrement clippé.

Comment calculer un point d'intersection ? Ce n'est pas bien compliqué :

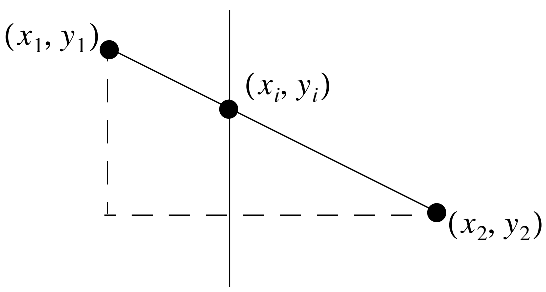


FIGURE 3. Calcul d'un point d'intersection

A partir de la FIGURE 3. on déduit l'équation suivante :

$$\left(\frac{y_2 - y_i}{x_2 - x_i} = \frac{y_2 - y_1}{x_2 - x_1}\right) \Rightarrow y_i = \frac{(y_2 - y_1) \times (x_2 - x_i)}{x_2 - x_1} \qquad \text{(EQ 1)}$$

Facile non ? Mais bon, comptons les instructions.... hmmm trois soustractions, une multiplication et argl! Une division!

Comment faire mieux ? Pourquoi ne pas essayer la dichotomie ?

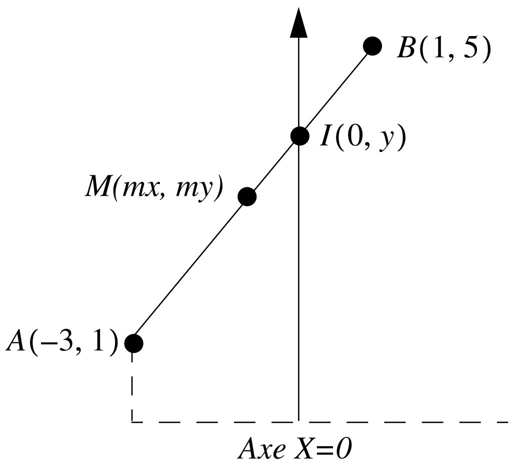


FIGURE 4. Calcul d'un point d'intersection

Supposons que nous voulons clipper la droite qui va de *A(-3,1)* vers *B(1,5)*, sur l'axe d'équation *X=0.* Nous devons trouver la coordonnée *y* du point d'intersection. Par dichotomie on commance par chercher le point *M(mx, my)* milieu de *AB* :

$$mx = (Ax + Bx)/2 = (-3+1)/2 = -1$$

$$mY = (Ay + By)/2) = (1+5)/2 = 3 \qquad \text{(EQ 2)}$$

Maintenant regardons où se trouve l'axe *X=0* par rapport à *M*. Il est encore à droite, ainsi, des deux segments A-M et M-B, l'axe X=0 intersecte M-B. Mettons A en M et recommençons !

$$mx = (-1+1)/2 = 0$$

$$my = (3+5)/2 = 4 \qquad \text{(EQ 3)}$$

Cette fois-ci le point d'intersection est sur l'axe. Ainsi la valeur recherchée pour la coordonnée y du point d'intersection est le point M de l'équation 3. Il pourrait sembler que cette méthode implique un nombre important de calculs. En effet, mais ils sont très peu couteux : juste une addition et une division par deux à chaque itération. Rappelons qu'une division par deux, dans le cas de calculs entiers, est juste un décalage de 1 bit sur la droite ! La pratique montre en effet que la méthode de clipping par dichotomie est plus rapide que les méthodes classiques.

Lorsqu'on effectue des divisions par décalage, il faut cependant faire attention ! En effet, -1 décalé de 1 bit ver la droite donne toujours -1 !!!! Ceci signifie que si l'on utilise l'algorithme décrit ci-dessus pour clipper le segment allant d'un point (-1, y1) au point (0, y2) sur l'axe X=0, il y a des chances pour que l'on boucle ! Il faudra donc penser à résoudre ce cas particulier !

```
int clippingEnX(Vertex *vertex1, Vertex *vertex2) {

   static Vertex tmp0, tmp1, tmp2; /* Pour les sauvegardes */
   static Vertex tmp3, tmp4, tmp5;
   Vertex *l, *r, *m, *t; /* Point gauche, droit, milieu, temporaire */
   Vertex *vmn, vmx;
   int whereto, swap;

   2D_clipping = 0;    // globale ici...

   if(*vertex1->ScreenVertex.x < *vertex2->ScreenVertex.x) {
       swap = 0; vmn = vertex1; vmx = vertex2;
   } else {
       swap = 1; vmn = vertex2; vmx = vertex1;
   }

   /* test pour voir si par hasard il n'y a pas de clipping */
   if((vmn->x > X_CLIPPING_MAX) || (vmx->x < X_CLIPPING_MIN))
       return(0);
   else {
       if(vmn->x <= X_CLIPPING_MIN) {
          tmp0 = *vmn; l = &tmp0; /* r, l, m, t ne pointent que sur des
                                     copies */
          tmp1 = *vmx; m = &tmp1
          r = &tmp2;

             whereto = 0;
       while(m->x != X_CLIPPING_MIN) {
             if(whereto == 1) {t = l; l = m; m = t;}
             else { t = r; r = m; m = t;}

             m->x = (l->x + r->x) >> 1;
             m->y = (l->y + r->y) >> 1;

             whereto = m->x < X_CLIPPING_MIN;
          }
          vmn->x = m->x;
          vmn->y = m->y;

          2D_clipping = swap^1;
       }
```

... Suite page suivante !

```
        if(vmx->x >= X_CLIPPING_MAX) {
           tmp3 = *vmn; l = &tmp3; /* r, l, m, t ne pointent que sur des
                                    copies */
           tmp4 = *vmx; m = &tmp4
           r = &tmp5;

           whereto = 0;
           while(m->x != X_CLIPPING_MAX) {
              if(whereto == 1) {t = l; l = m; m = t;}
              else { t = r; r = m; m = t;}

              m->x = (l->x + r->x) >> 1;
              m->y = (l->y + r->y) >> 1;

              whereto = m->x < X_CLIPPING_MAX;
           }
           vmn->x = m->x;
           vmn->y = m->y;

           2D_clipping |= swap & 1;
        }
     }
     return(1);
  }
```

La valeur de retour de cette fonction est zéro si la ligne est complètement clippée (en dehors de l'écran), sinon la fonction renvoie 1. L'utilité de la variable *2D_clipping* deviendra plus claire lorsqu'on abordera le clipping des polygones.

Remarquez que lorsqu'on clippera des cotés de polygones, dans le cas d'un lissage de gourand, ou d'un plaquage de texture, il faudra également calculer pour les points milieux successifs, non seulement les attributs *X* et *Y*, mais aussi les Intensités, les coordonées d'attachement des textures, etc...

On pourra dériver de la meme manière une fonction faisant le clipping en Y.

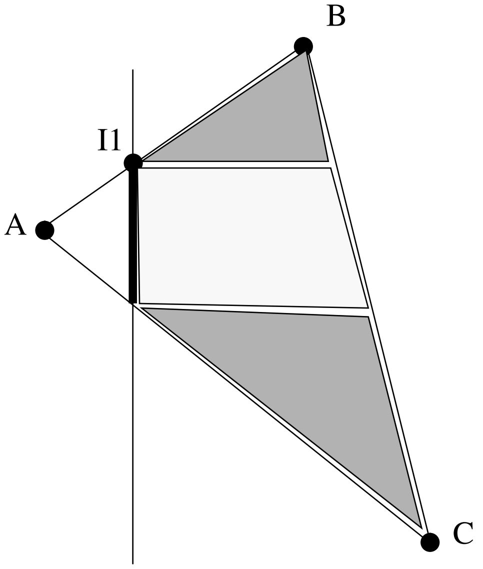
Il suffira d'insérer des appels aux fonctions de clipping juste au début des fonctions de balayage des cotés:

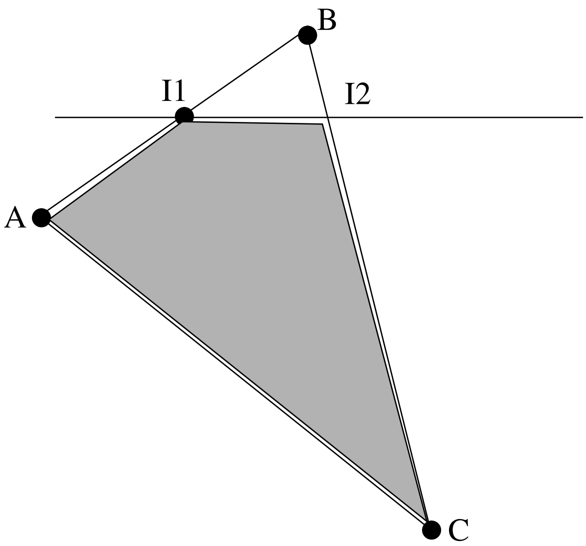
```
  ...
  if(clippingEnX(&v1, &v2)) {          /* clipping horizontal */
     if(clippingEnY(&v1, &v2)) {      /* clipping vertical */
        code de la fonction de scan des cotés standard...
        ...
```

Remarquons au passage que si une ligneest complètement clippée, on ne va pas scanner ses cotés.

4 Clipping de polygones

Souvenez-vous maintenant que pour dessiner un polygone, on commence par balayer ses cotés. En appelant les fonctions de clipping précédentes, cela ne résoud malheureusement pas complètement le problème. Il reste encore la moitié du travail à effectuer ! En effet, considéros les figures suivantes :



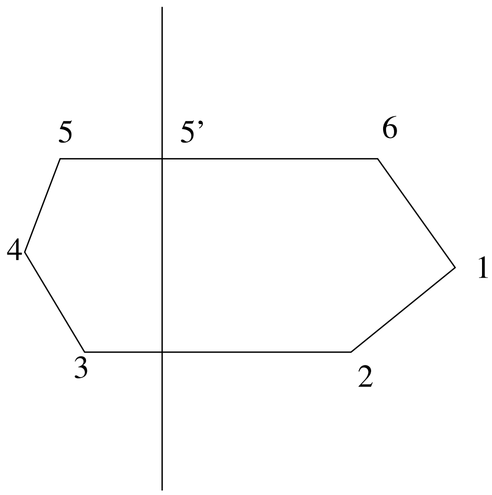


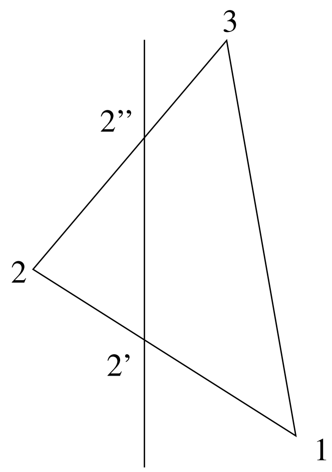
Le coté A-B contribue dans la première figure, au bord gauche du triangle. Le clipping ne pose pas de problèmes dans le cas de la figure de droite, en effet, il suffit de supprimer la partie I1-B du segment AB, et la partie I2-C du segment B-C, et de dessiner le polygone A-I1-I2-C.

En revanche, dans le cas de la figure de gauche, , il ne suffit pas simplement de supprimer le segment clippé I1-B, car on perdrait à ce moment-là la frontière gauche du polygone. Sans frontière gauche, on ne peut pas scanner le polygone. Avec juste des frontières dues au clipping horizontal, on peut quand meme scanner les cotés, ça ne pose pas de problèmes.

On observe donc qu'un polygone que l'on scanne un seul côté à la fois ne peut qu'être clippé avec les axes horizontaux.

Pour créer un polygone clippé avec les axes verticaux est un peu plus compliqué car le polygone résultant peut possèder plus de cotés que le polygone original. Considérons les situations suivantes :





Le polygone original de la figure de gauche possède 6 côtés avant le clipping le long de X=0, il en possède 5 après. Le polygone de la figure de droite possède un côté de plus après le clipping.

Il parait évident qu'il va nous falloir créer une structure de données temporaire pour travailler sur le polygone clippé.

Nous savons clipper un segment de droite, regardons comment des segments clippés forment un polygone.

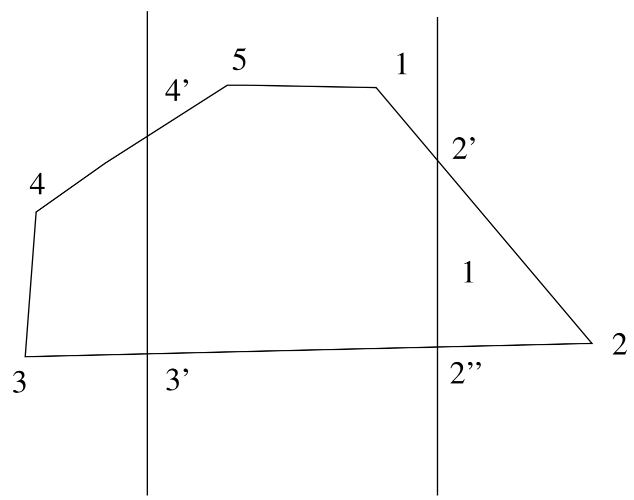
On suppose que l'on clippe contre l'axe X=0. Il apparait évident que si un côté est complètement à l'extérieur ses sommets ne seront pas présents dans le nouveau polygone. Par ailleurs, si un côté est à l'intérieur de l'écran, ses deux sommets seront présents dans le nouveau polygone. En outre, tous les points d'intersection entre un côté et l'axe seront présents également.

Une dernière considération : chaque sommet appartient à deux côtés, ainsi, lorsqu'un côté est non clippé, on pourra ajouter dans la nouvelle structure uniquement le second sommet, en supposant qu'on se sera déjà ocupé du premier lorsqu'on a traité le côté précédent.

Résumons tout cela :

1. Si un côté est non clippé, ou si le second sommet de ce côté est à l'intérieur de l'écran, on ajoute à l'intérieur de la structure uniquement le second point.
2. Lorsque le premier sommet est modifié (point d'intersection) ou lorsque les deux sommets sont modifiés, on ajoute les deux sommets à lq structure.
3. Lorsque les deux points sont à l'extérieur, on en ajoute aucun.

Il se trouve que cet algorithme n'a pas besoin d'être modifié lorsqu'on clippe avec deux axes :



- Segment 1-2 : second point clippé, cas 1, on ajoute 2'
- Segment 2-3 : les deux points sont modifiés, cas 2, on ajoute 2'' et 3'
- Segment 3-4 : les deux points sont à l'extérieur, cas 3, on ne fait rien
- Segment 4-5 : le premier point est modifié, cas 2, on ajoute 4' et 5
- Segment 5-1 : pas de clipping, cas 1, on ajoute 1

...le polygone final est 2'-2''-3'-4'-5-1, c'est bien ce que l'on cherche ! On comprend maintenant l'intérêt de la variable 2D_clipping : elle vaut 1 lorsque le premier des deux sommets a été modifié et 0 si aucun sommet ou si seul le deuxième sommet a été changé.

Donc, avec la valeur de retour de la fonction de clipping et cette variable on peut distinguer tous les cas.

```
Pour tous les côtés
Faire
  if(clipingEnX(&v1, &v2)) {
      if(2D_clipping) {
         ajouter v1 à la structure
         ajouter v2 à la structure
         nbVertex += 2;
      } else {
         ajouter v2 à la structure
         nbVertex++;
      }
  }
FinFaire

Recopier la structure dans le polygone.
```