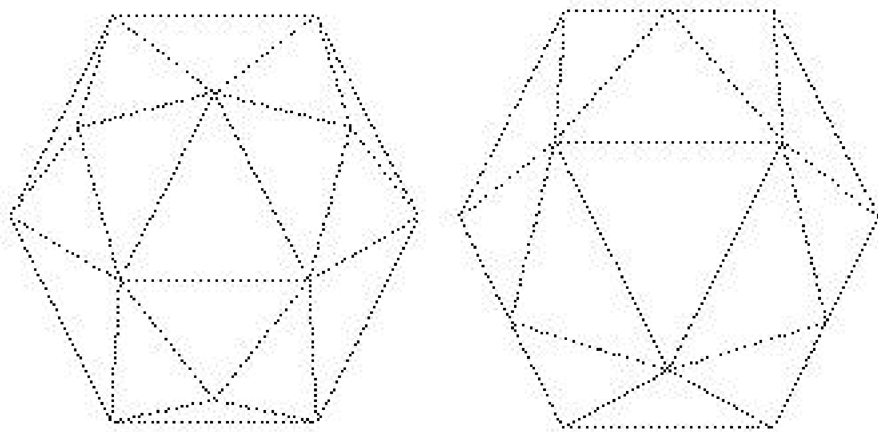
Synthèse d'images

(7) Faces visibles, faces cachées



Plan de l'exposé :

1 - Préambule
2 - Test de visibilité
3 - Z-Buffer
4 - Balayage
5 - Lignes cachées
6 - Comparaison

v5 P. CUBAUD <cubaud@cnam.fr> Dept. informatique CNAM, Paris

1. Préambule

Un petit programme de tracé :

```
#include <stdio.h>
#include <math.h>

/* structure pour decrire la scene */
#define MAXFACES 500
double OB[MAXFACES][3][3];
double O2[MAXFACES][3][2];
int NB;/* nombre reel de facettes  */


/* position de l'observateur et distance de l'ecran
*/
#define R 10
#define theta 45
#define phi 30
#define D 5

/* zone de dessin en coord. ecran */
#define xa 3
#define ya 7.35
#define xb 18
#define yb 22.25

/* variables diverses */
int i,j;
double
sthe,cthe,sphi,cphi,x,y,z,xx,yy,zz,minx,maxx,miny,max
y;
double px,py,pz,qx,qy,qz,nn,xo,yo,zo;
```

```
main()
{
 /*--- coord. cart. de l'observateur ---*/
 sthe=sin(theta*3.1416/180);
 cthe=cos(theta*3.1416/180);
 sphi=sin(phi*3.1416/180);
 cphi=cos(phi*3.1416/180);
 xo= R*cthe*cphi;
 yo= R*sthe*cphi;
 zo= R*sphi;

 /*--- lecture des coordonnees ---*/
 NB=0;i=0;
 while (scanf("%lg %lg %lg\n",
     &OB[NB][i][0],&OB[NB][i][1],&OB[NB][i][2])!=EOF)
 {
  i++;
  if (i==3) {i=0;NB++;}
 }


 /****** CODE A INSERER ICI POUR LA SUITE ********/

 /*--- proj. perspective (cf cours chap 6) ---*/
 for (i=0;i<NB;i++)
 {
  for (j=0;j<3;j++)
  {
   x=OB[i][j][0];y=OB[i][j][1];z=OB[i][j][2];
   xx= -x*sthe+y*cthe;
   yy= -x*cthe*sphi-y*sthe*sphi+z*cthe;
   zz= -x*cthe*cphi-y*sthe*cphi-z*sphi+R;
   O2[i][j][0]= D*xx/zz;
   O2[i][j][1]= D*yy/zz;
  }
 }

 /*--- pour eviter le clipping :
       calcul de la fenetre maximale ---*/
 minx=O2[0][0][0];maxx=minx;
```

```
 miny=O2[0][0][1];maxy=miny;
 for (i=0;i<NB;i++)
 {
  for (j=0;j<3;j++)
  {
   if (O2[i][j][0]<minx) {minx=O2[i][j][0];}
   if (O2[i][j][0]>maxx) {maxx=O2[i][j][0];}
   if (O2[i][j][1]<miny) {miny=O2[i][j][1];}
   if (O2[i][j][1]>maxy) {maxy=O2[i][j][1];}
  }
 }

 /*--- calcul des coordonnees ecran */
 for (i=0;i<NB;i++)
 {
  for (j=0;j<3;j++)
  {
   O2[i][j][0]=xa + (xb-xa)*(O2[i][j][0]-minx)/(maxx-
minx);
   O2[i][j][1]=ya + (yb-ya)*(O2[i][j][1]-miny)/(maxy-
miny);
  }
 }

 /*--- generation des ordres de trace en Postcript */
 printf("newpath\n");
 for (i=0;i<NB;i++)
 {
  printf("%.5f %.5f moveto\n",
         O2[i][0][0],O2[i][0][1]);
  printf("%.5f %.5f lineto\n",
         O2[i][1][0],O2[i][1][1]);
  printf("%.5f %.5f lineto\n",
         O2[i][2][0],O2[i][2][1]);
  printf("%.5f %.5f lineto\n",
         O2[i][0][0],O2[i][0][1]);
 }
 printf("stroke\n");
}
```

- **Données transmises en entrée par sphere.c**

```
1.00000 0.00000 0.00000
0.00000 0.00000 1.00000
0.00000 1.00000 0.00000
0.00000 1.00000 0.00000
0.00000 0.00000 1.00000
-1.00000 0.00000 0.00000
-1.00000 0.00000 0.00000

.... etc ...
```

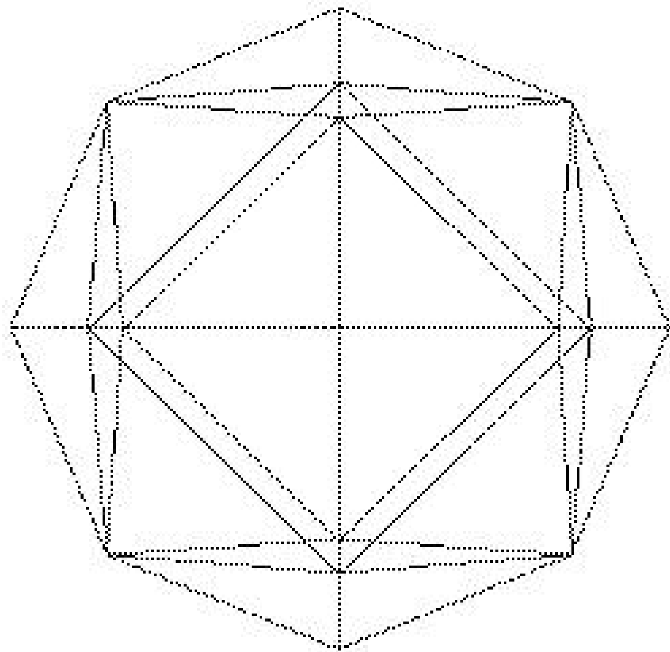
- **Résultats du programme**

```
/centimetre {72 mul 2.54 div} def
1 centimetre 1 centimetre scale
0.014 setlinewidth
newpath
10.50000 14.80000 moveto
10.50000 22.25000 lineto
18.00000 14.80000 lineto
10.50000 14.80000 lineto
18.00000 14.80000 moveto
10.50000 22.25000 lineto
10.50000 14.80000 lineto
18.00000 14.80000 lineto
.... etc ...
10.50000 7.35000 lineto
10.50000 14.80000 lineto
3.00000 14.80000 moveto
10.50000 14.80000 lineto
10.50000 7.35000 lineto
3.00000 14.80000 lineto
stroke
```

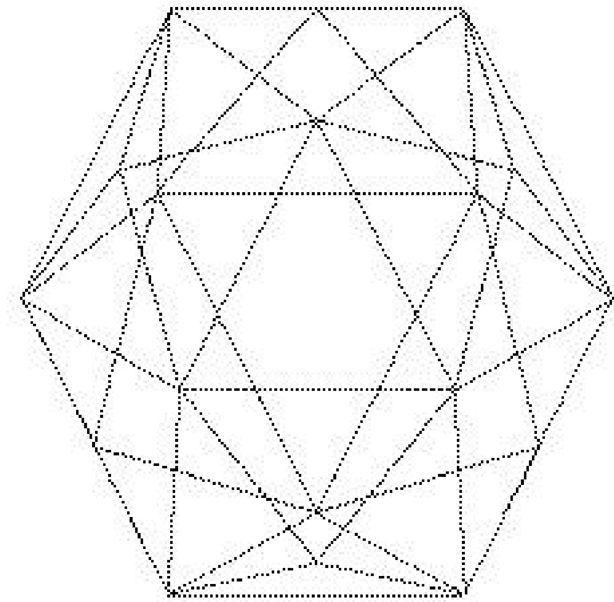
Résultats

- Visualisation avec GhostScript, Acrobat ou impression directe
- Avec le programme sphère (niveau=2)

Observateur en R = 10, D = 5, = 0°, = 0°



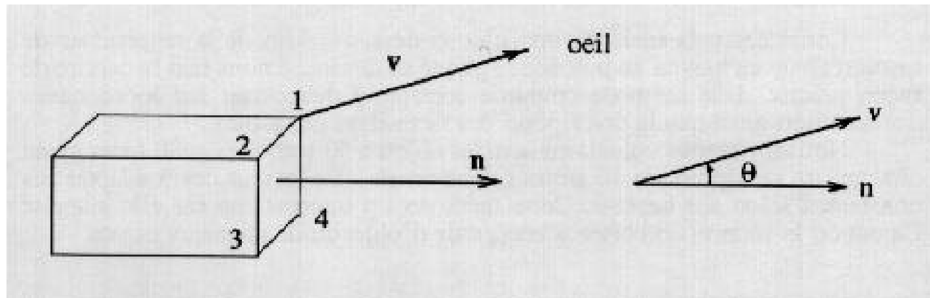
Observateur en R = 10, D = 5, = 45°, = 30°



=> ce n'est pas très lisible…

2. Test de visibilité

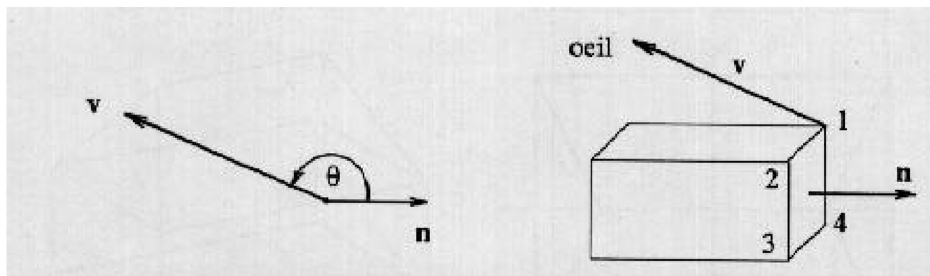
- Face visible



[DONY] p. 323

l'angle est aigu

- Face invisible



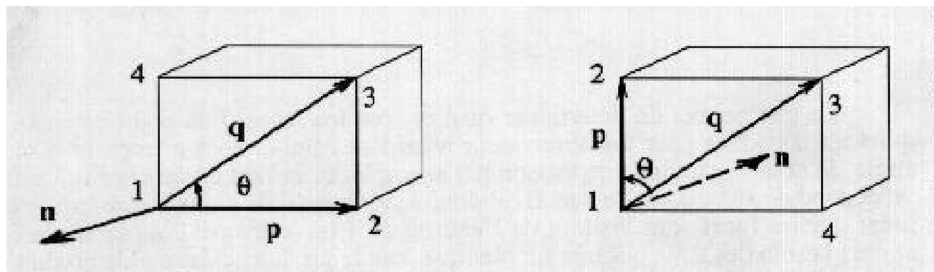
l'angle est obtu

- Test de visibilité

  - calculer le vecteur normal N à la face
  - déterminer le vecteur de vision V pour cette face
  - déterminer l'angle
  - s'il est obtu, la face est invisible

- A utiliser systématiquement avant tout algorithme d'élimination des surfaces cachées

- Calcul de la normale par le produit vectoriel

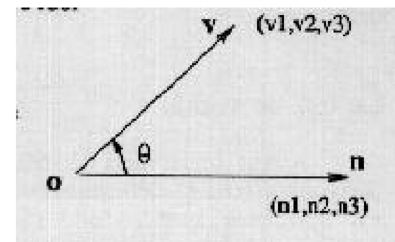


Si la numérotation des sommets des faces suit la règle "main droite", le produit vectoriel de P et Q défini la normale de la face vers <u>l'extérieur</u> de l'objet

$$\vec{N} = \vec{P} \circ \vec{Q} \qquad N = P.Q.\sin\theta$$

$$x_N = y_P.z_Q - z_P.y_Q$$
$$y_N = z_P.x_Q - x_P.z_Q$$
$$z_N = x_P.y_Q - y_P.x_Q$$

- Calcul de par le signe du produit scalaire



$$\vec{N}.\vec{V} = N.V.\cos\theta$$
$$= x_N.x_V + y_N.y_V + z_N.z_V$$

θ obtu $\quad \cos\theta < 0 \quad \vec{N}.\vec{V} < 0$

- Propriété d'un polyèdre convexe isolé :

Une face est soit entièrement visible, soit entièrement invisible

=> le test de visibilité est suffisant pour éliminer ses surfaces cachées

Modification de l'algorithme

- On ajoute des structures de données

```
double NO[MAXFACES][3];     /* les normales */
double VI[MAXFACES];        /* test de visibilite */
```

- Calcul des normales et calcul de la visibilité

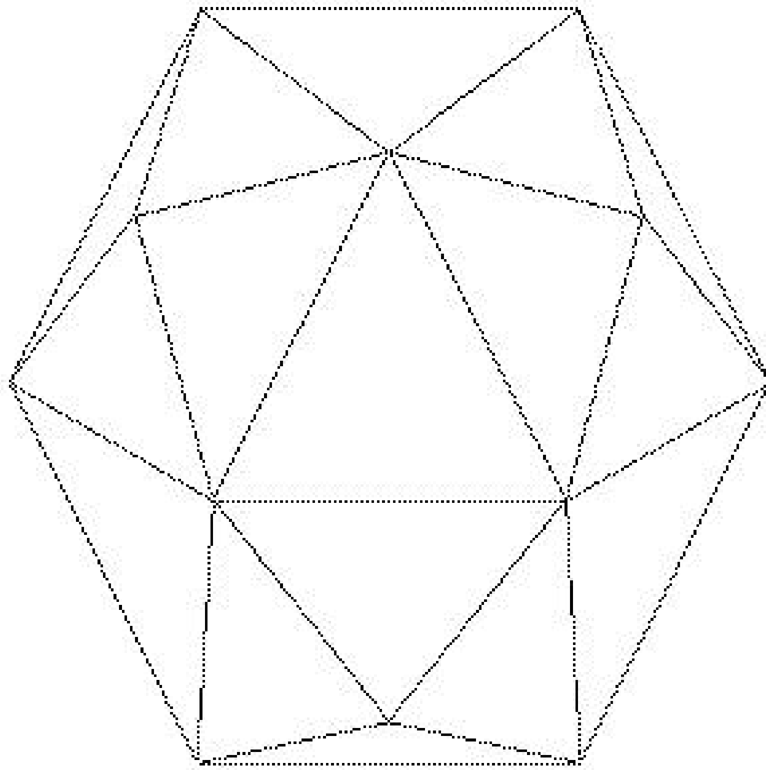
```
  for (i=0;i<NB;i++)
 {
  px=OB[i][1][0]-OB[i][0][0];
  py=OB[i][1][1]-OB[i][0][1];
  pz=OB[i][1][2]-OB[i][0][2];
  qx=OB[i][2][0]-OB[i][0][0];
  qy=OB[i][2][1]-OB[i][0][1];
  qz=OB[i][2][2]-OB[i][0][2];

  NO[i][0]=py*qz-qy*pz;
  NO[i][1]=pz*qx-qz*px;
  NO[i][2]=px*qy-qx*py;

  nn=sqrt(NO[i][0]*NO[i][0]
          +NO[i][1]*NO[i][1]
          +NO[i][2]*NO[i][2]);

  NO[i][0]/=nn;NO[i][1]/=nn;NO[i][2]/=nn;
 }
```

```
 for (i=0;i<NB;i++)
 {
  px=xo-OB[i][0][0];
  py=yo-OB[i][0][1];
  pz=zo-OB[i][0][2];
  nn=sqrt(px*px+py*py+pz*pz);
  px/=nn;py/=nn;pz/=nn;
  VI[i]=px*NO[i][0]+py*NO[i][1]+pz*NO[i][2];
 }
```

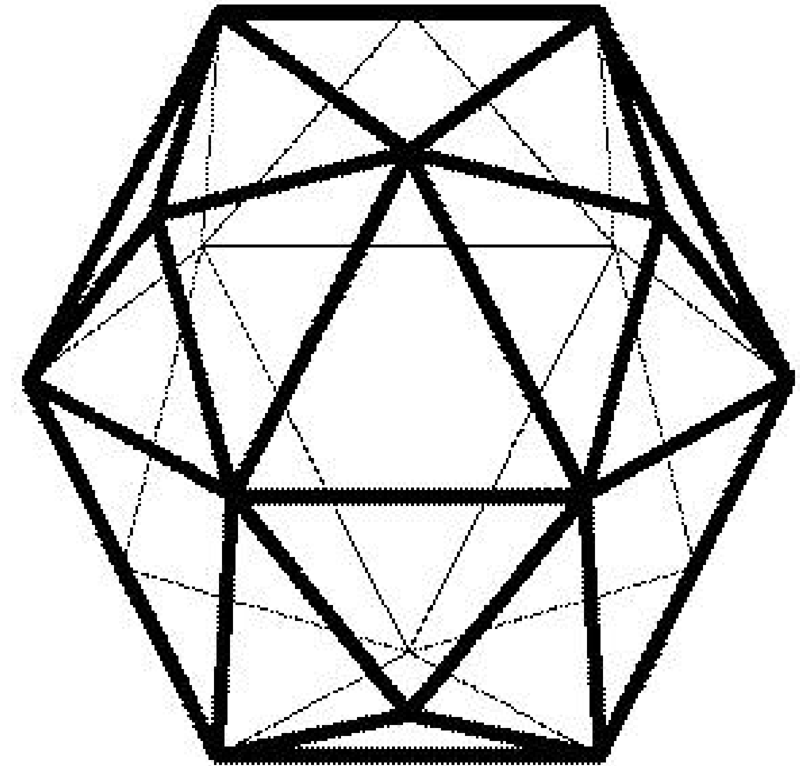
- On ne dessine que les faces visibles

```
 printf("newpath\n");
 for (i=0;i<NB;i++)
 {
  if  (VI[i]>0) {
  printf("%.5f %.5f moveto\n", O2[i][0][0],
O2[i][0][1]);
  printf("%.5f %.5f lineto\n", O2[i][1][0],
O2[i][1][1]);
  printf("%.5f %.5f lineto\n", O2[i][2][0],
O2[i][2][1]);
  printf("%.5f %.5f lineto\n", O2[i][0][0],
O2[i][0][1]);
   }
 }
 printf("stroke\n");
```

Résultats

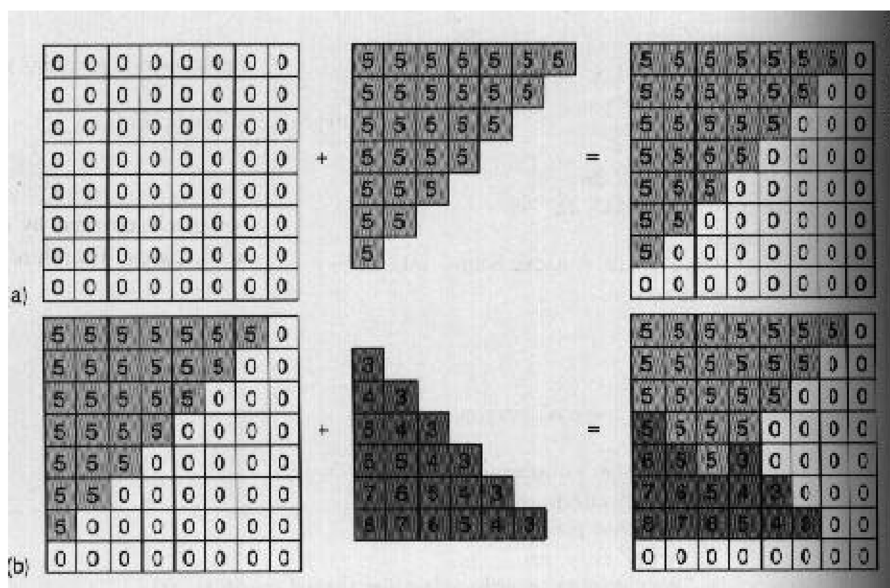


Variante : on dessine autrement les faces invisibles :



3. Algorithme du Z-buffer (Catmull, 1974)

Suppose l'existence d'une matrice (Z-buffer) associant à chaque pixel de l'ecran, la profondeur (coordonnée Z) de la portion de facette concernée.



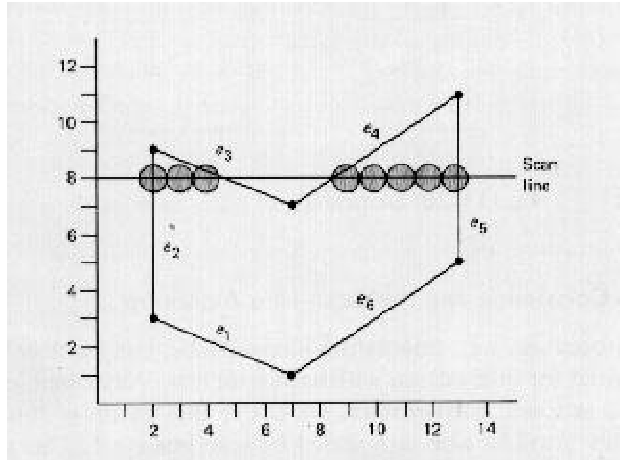
[FOLEY] p. 670

- Pseudo-code :

```
pour chaque face F faire
    pour chaque pixel (i,j) de F faire
        pz = valeur de Z pour la face F en (i,j)
        si pz < Zbuffer[i,j] alors
            Zbuffer[i,j] = pz
            pixel (i,j) = couleur de la face F
        fin si
    fin pour
fin pour
```

souvent : face = polygone (triangle)

Balayage pour les polygones



[FOLEY83] p. 457

- On ne tient pas compte des arêtes horizontales

- Attention aux "pointes" (sommets extremums)

- Peut-être spécialisé si polygones = triangles

- Principe :

pour chaque ligne de balayage (coord. y) faire

Chercher les intersections de la ligne avec chaque arête du polygone

Trier les intersections par x croissant

pour chaque pixel compris entre 2 intersections

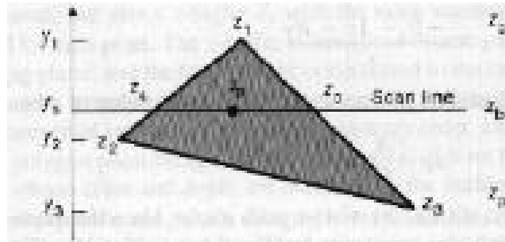
Comparaison avec Zbuffer

Coloriage

fin pour

fin pour

- Interpolation des profondeurs



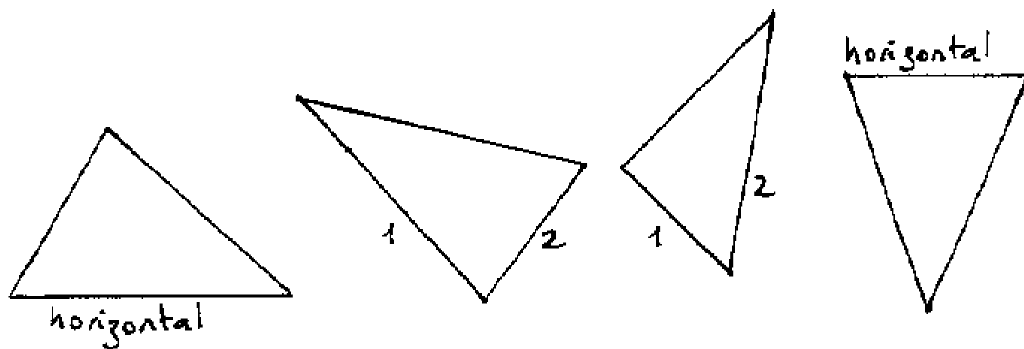
$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_S}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_S}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$

- Balayage pour un triangle :

On a 4 possibilités :



cas #2 :

Y = plus petit Y parmi les sommets

Xa = Xb = coord. x du sommet correspondant

= pente arête #2

tant que Y <= Ymax faire
    Xa = Xa + pente arête #1
    Xb = Xb +
    BALAYAGE ENTRE Xa et Xb
    si Xb = coord. x du 3ème sommet alors
        = pente arête #3
    fin si
    Y = Y + 1
fin tant que

• Algorithme général

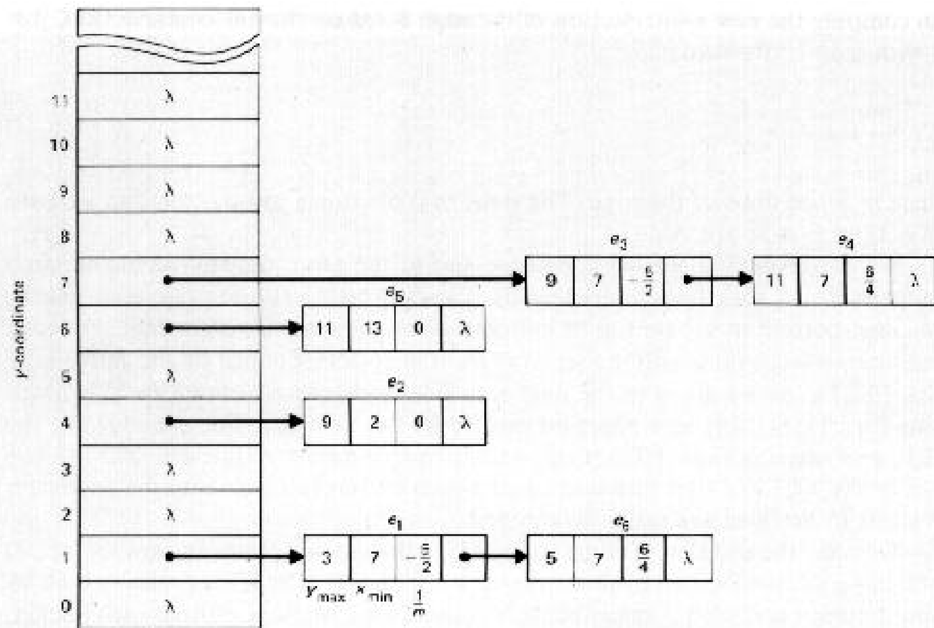
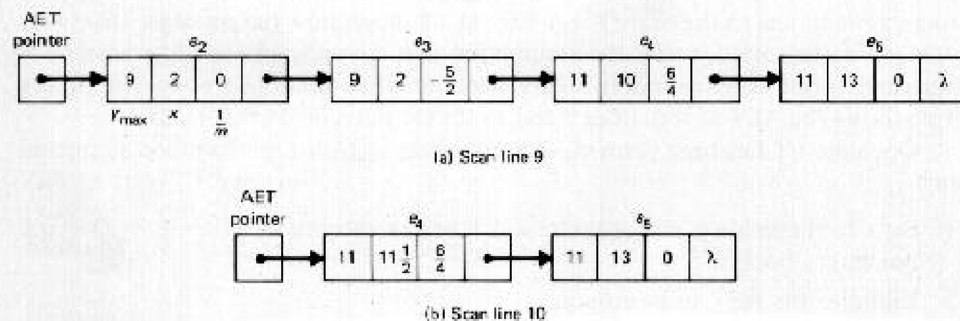


Fig. 11.27 Bucket-sorted edge table for Fig. 11.25, with edges e_2 and e_5 shortened.



(a) Scan line 9

(b) Scan line 10

[FOLEY ed. 1983 p. 460]

BET : bucket-sorted edge table
AET : active edge table

y := plus petite coord. en Y dans la BET

initialiser l'AET à vide

tant que BET et AET non vides faire

1) insérer dans l'AET par x croissant, les éléments de BET[y]

2) balayer les pixels entre les paires successives de l'AET (comparaison avec le Zbuffer + coloriage…)

3) retirer de l'AET les éléments dont y = ymax

4) pour chaque élément i de l'AET faire
AET[i].x = AET[i].x + AET[i].pente

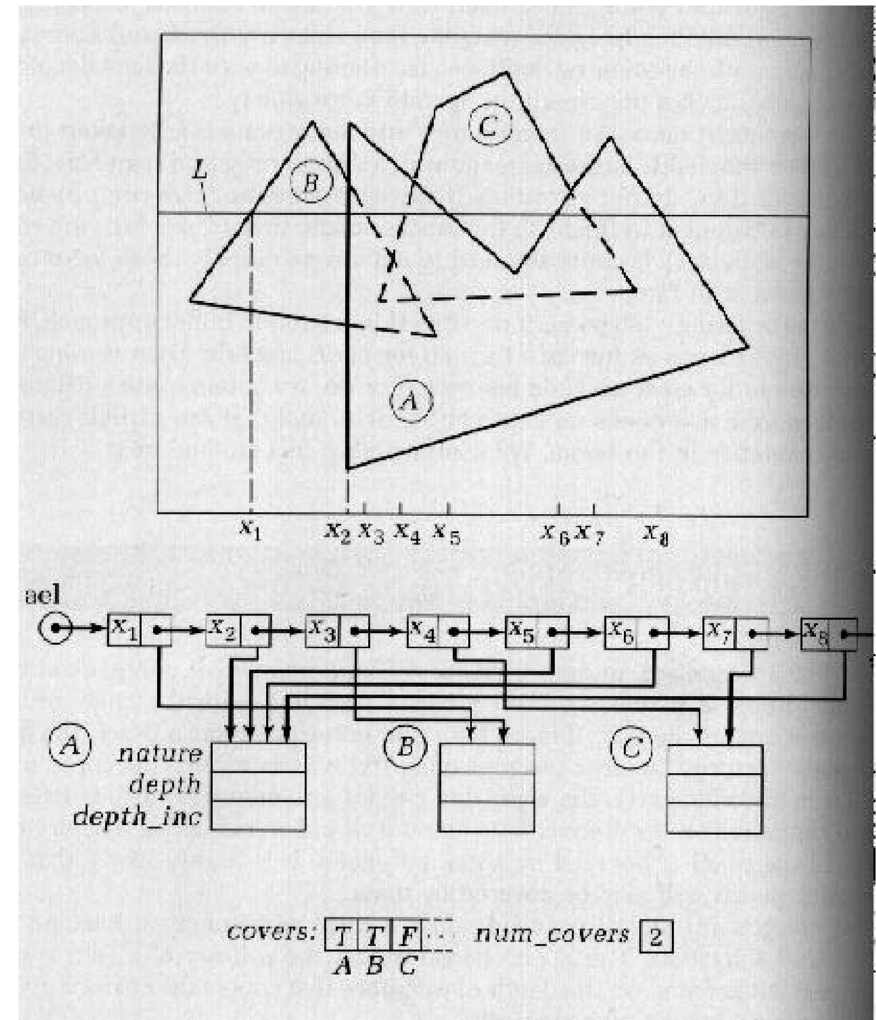
5) trier l'AET par x croissant

6) y = y + 1

fin tant que

4. Algorithme par balayage

- On balaie ("scan") l'écran ligne par ligne
- Pour chaque ligne, parcourt de la liste de tous les polygones
- On exploite la "cohérence" d'une arête à l'autre pour le calcul d'intersection
- Le Z-Buffer devient inutile



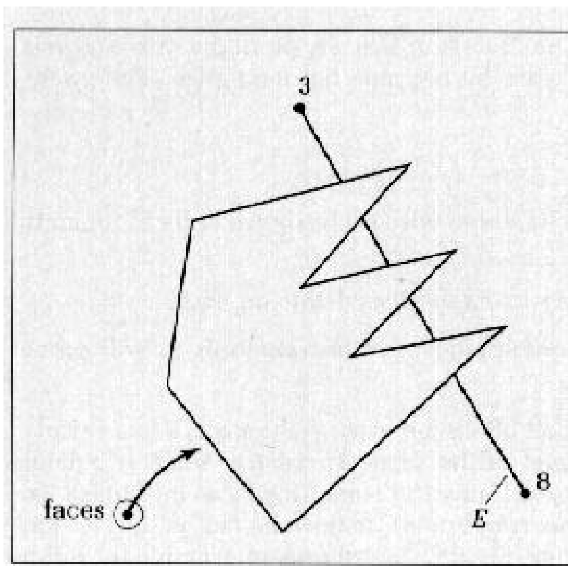
[HILL] p. 598

5. Elimination des lignes cachées

- Les algorithmes précédents ne sont utilisables pour des tables traçantes

  => il faut un algorithme spécial pour le tracé "vectoriel" des arêtes

- Principe :[HILL] p. 610




```
mettre en pile toutes les arêtes de la scène

tant que pile non vide faire

    A =  sommet de pile; A est visible

    pour chaque face F faire
        comparer(F,A)
        si A survivant alors
            empiler tous ses restes
        sinon A est invisible
    fin pour

    si A visible alors dessiner A

fin tant que
```

Comparaison face/arête

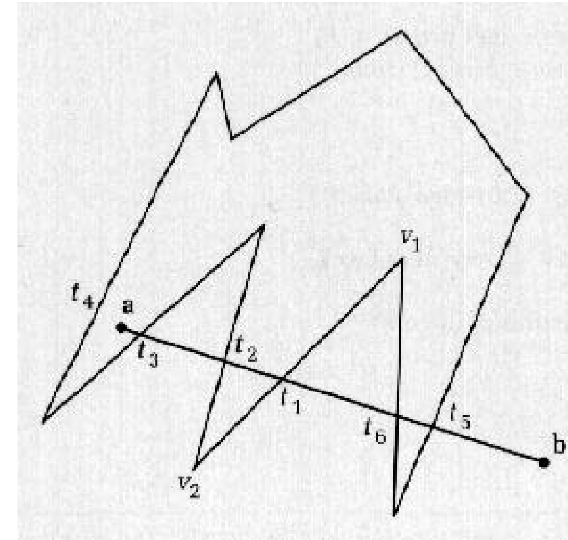
Plusieurs étapes :

1) comparaison des "extents" : si pas d'intersection, A est survivante dans son intégralité.

2) sinon, si les deux sommets de A sont devant F, alors A est survivante. Si les deux sommets sont derrière, aller en 4.

3) sinon, calcul de l'intersection : si A "plonge" dans F, alors la partie "devant" est seule survivante. Pour la partie "derrière", aller en 4

4) Calculer les intersections de A (ou derrière de A) avec chaque arête de F. Trier les portions de A restantes.



[HILL] p. 612

6. Comparaison

- Test de visibilité : à appliquer en amont de chaque algorithme

- Algorithmes par balayage :

  Les calculs sont fait en "précision pixel" => risque de problèmes du fait de la discrétisation (aliasing)

- Algorithme d'élimination des lignes cachées :

  Les calculs sont fait en "précision objet" => meilleur

  Mais remplissage des polygones impossible

- Complexité des algorithmes :

En temps :

  Z-Buffer : temps indépendant du nombre N de polygones car si N croît, la taille des polygones diminue

  Autres algorithmes : temps proportionnel à N^2 du fait des tris et des opérations supplémentaires.

En mémoire :

  Z-Buffer très gourmand, les autres beaucoup moins.