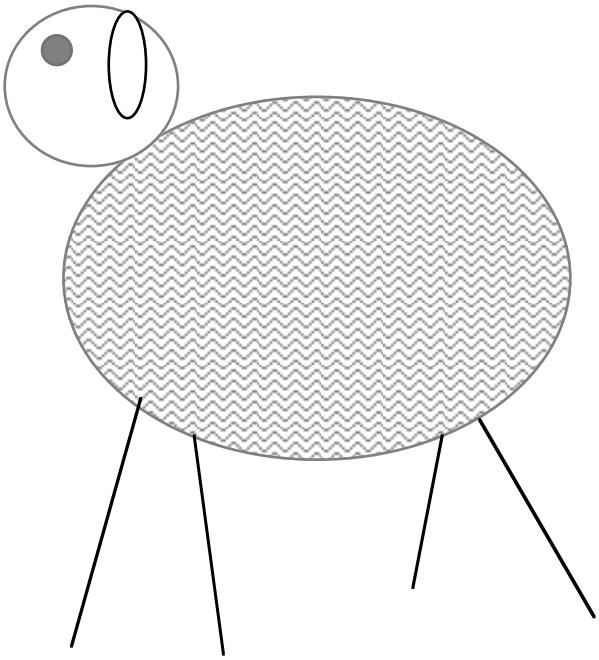
ED 1. Comment dessiner un mouton

L'objectif de l'ED est d'élaborer les algorithmes de tracé de primitives élémentaires, telles que celles nécessaires pour le dessin ci-dessous :



Pour produire et visualiser l'image, les programmes à écrire produiront un fichier à la norme PGM ("Portable GrayMap"), à partir d'un tableau à deux dimensions de pixels toujours présent en mémoire. On ne se préoccupera pas de vérifier si les primitives de tracé "sortent" du tableau de pixels à remplir.

On peut partir du canevas de programme ci-dessous (en Ada) :

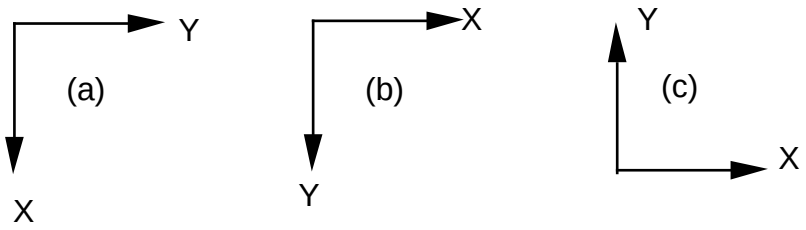
```
with ada.text_io, ada.integer_text_io;
use  ada.text_io, ada.integer_text_io;
procedure ED1 is

  -- structure pour le graymap ou se font les dessins

  BLANC : constant integer := 255;
  NOIR  : constant integer := 0;
  GRIS  : constant integer := 127;

  H : constant integer :=400;
  L : constant integer :=400;

  Image : array(1..H,1..L) of integer :=
              (others=>(others => BLANC));
```

```
  ---  procedures de dessin

  procedure Point(x,y:integer;p:integer) is
  begin
          ---    crire !
  end Point;

  procedure Droite(x1,y1,x2,y2:integer;p:integer) is
  begin
          ---    crire !
  end Droite;

  procedure Ellipse(x1,y1,x2,y2:integer;p:integer) is
  begin
          ---    crire !
  end Droite;

  -- procedures d'affichages

  procedure AfficheImageASCII is
  begin
   put("P2");new_line;
   put(L);put(H);new_line;
   put("255");new_line;
   for i in 1..H loop
    for j in 1..L loop
     put(Image(i,j),4);
    end loop;
    new_line;
   end loop;
   new_line;
  end AfficheImageASCII;

  procedure AfficheImageBIN is
  begin
   put("P5");new_line;
   put(L);put(H);new_line;
   put("255");new_line;
   for i in 1..H loop
    for j in 1..L loop
     put(character'val(Image(i,j)));
    end loop;
   end loop;
  end AfficheImageBIN;

  -- programme principal
  -- trac  du mouton

begin
  Droite(57,160,32,250, NOIR);
  Droite(77,170,90,255, NOIR);
  Droite(172,170,160,227, NOIR);
```

```
  Droite(185,162,227,237, NOIR);
  SEllipse(27, 42, 220, 180, 2,GRIS);
  SEllipse(5, 10, 72, 67, 2,GRIS);
  SEllipse(45, 12, 60, 51, 2,GRIS);
  SEllipse(20, 20, 30, 30, 2,NOIR);
  AfficheImageBIN;
end ED1;
```

Exercice 1 : Garder ses repères

Une image décrite au format PGM suppose le repère (a). Les coordonnées pour les primitives de tracé du mouton ont été extraites de Mac Draw, qui utilise le repère (b). Enfin, on utilise traditionnellement en mathématique un repère de type (c).



Comment convertir des coordonnées d'un pixel du repère (b) vers le repère (c) ?

Comment convertir des coordonnées du repère (c) vers le repère (a) ?

En déduire le code de la procédure Point, et la manière dont Point est appelé dans Droite et Ellipse.

Exercice 2 : Tracé de segment de droite

On souhaite écrire la procédure Droite en tenant compte des cas triviaux : ligne horizontale, verticale, en diagonale. On se place dans un repère de type (c).

Combien de cas de segments faut-il pouvoir gérer au total ?

Ecrire complètement la procédure

L'algorithme précédent a deux défauts majeurs, dont l'un rédhibitoire pour l'informatique à l'ancienne.

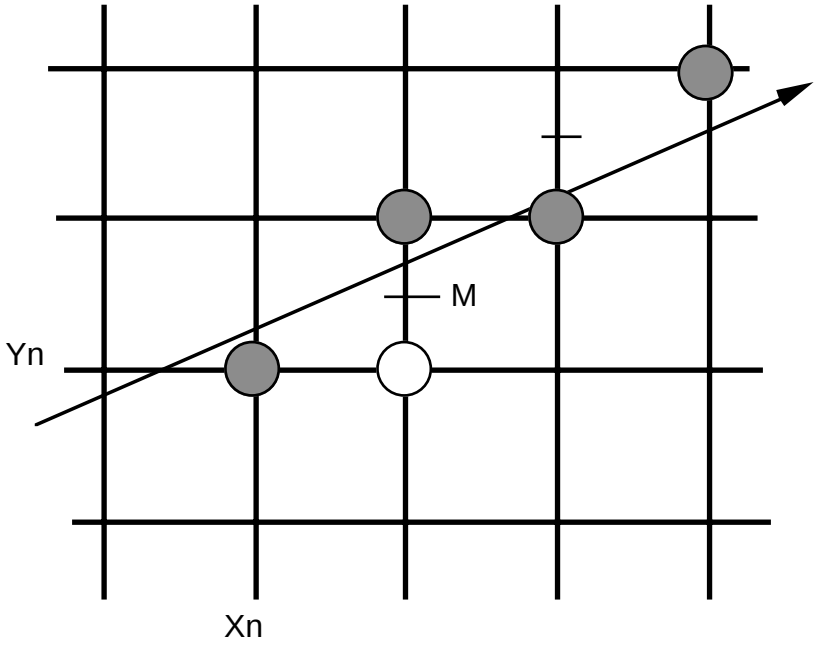
Préciser ces inconvénients

Exercice 3 : Algorithme du point milieu (Bresenham, 1965)

Il existe un algorithme ne nécessitant que des nombres entiers, en utilisant une méthode incrémentale de calcul des points.

Pour l'établir, on se limitera ici au cas d'une droite (D) de pente m<1, tracée "de bas en haut". On est toujours dans un repère de type (c).

Si à la n-ième étape, on a sélectionné le pixel <X(n),Y(n)>. On cherche ensuite le "meilleur" <X(n+1), Y(n+1)>. On aura soit <X(n)+1, Y(n)>, soit <X(n)+1, Y(n)+1> :



Le choix dépend finalement la position de la droite réelle par rapport au point milieu M. On peut connaitre la position de M par rapport à (D) : il suffit de calculer le signe de l'équation de la droite (D) en M :

(D) : F(x, y) = y - m*x - b = 0 avec m = dy/dx = (Y1-Y0)/(X1-X0) et b = Y0 - m*X0

Sous une forme plus commode : F(x,y) = dy*(x-X0)-dx*(y-Y0)

En M, on a F(XM,YM) = F(X(n)+1,Y(n)+1/2) que l'on notera d(n), dont on cherche à connaître le signe.

Exprimer d(n+1) en fonction de d(n). Calculer d(0)

En déduire l'algorithme de tracé pour le cas particulier de segment du schéma.

(et modifier ensuite le programme général de l'ex. 1)

Exercice 4 : Super-ellipses

Les super-ellipses sont une généralisation de l'ellipse, définies par la forme implicite :

(SE) : (x/a)^n + (y/b)^n = 1

en notant a, le demi-grand axe (parallèle à l'axe Ox) et b, le demi-petit axe (parallèle à l'axe 0y) et en supposant le centre de (SE) à l'origine. Pour produire un tracé de la super-ellipse, on préfère employer une représentation explicite (ou paramétrée) :

x(t) = a*cos(t)^(2/n) avec $-\pi \leq t \leq +\pi$

y(y) = b*sin(t)^(2/n)

Montrer que l'on retrouve l'équation de l'ellipse avec n=2.

Si a = b = r, on parle de supercercle de rayon r

Donner l'allure de quelques supercercles, avec n = 1/2 n = 1 n = 5

Donner un algorithme de tracé de super-ellipses

Donner un algorithme plus rapide pour le cas des ellipses

Exercice 5 : Remplissage avec motifs

On souhaite maintenant colorier l'intérieur de nos super-ellipses. Il existe un algorithme classique simple ("flood-fill alg.") de remplissage de zones dans l'image qui suppose que :

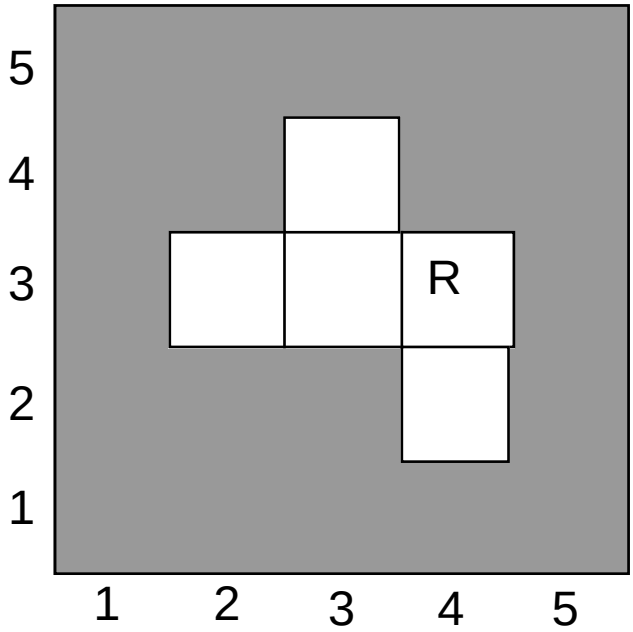
- un pixel au moins à l'intérieur de la région est connu. On l'appelera la racine ("seed")

- les pixels de la région à colorier sont tous d'une certaine couleur au début (COUL_INT);

- la frontière de la région est fermée et définie par des pixels d'une autre couleur (COUL_FRONT) que ceux de l'intérieur.

Donner un algorithme récursif pour le remplissage

Soit la figure suivante à colorier ([HILL] p. 438) :

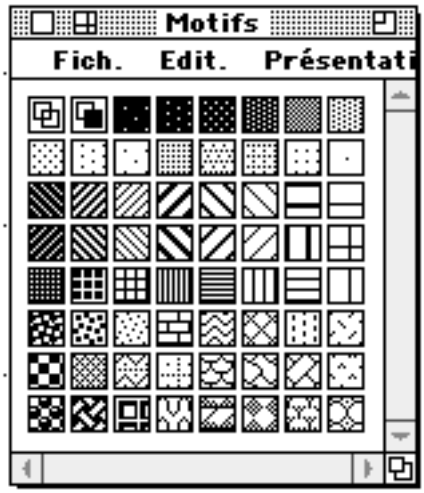


Exécuter l'algorithme sur l'exemple. Que faut-il en penser ?

Indiquer une manière plus efficace de procéder

RQUE : Il existe des algorihmes beaucoup plus efficaces pour des régions correspondant à des polygones convexes de sommets connus (cf ED4)

Pour remplir la région avec autre chose qu'une couleur identique, on peut utiliser une forme répétitive ("tile_pattern"), comme celles proposées dans MacDraw :



Modifier l'algorithme de remplissage pour qu'il en tienne compte

ED 2. PostScript-eries

Le langage PostScript a été rendu public par la société Adobe en 1985 (pour la version 1, on en est maintenant à la version 3). Il s'agit d'un langage spécialisé dans la description d'images à imprimer permettant de faire abstraction du dispositif physique chargé de l'impression : un même programme PostScript peut ainsi être utilisé par une imprimante laser grand public et une photocomposeuse, par ex. PostScript n'est pas un simple langage de description, mais bien un langage impératif à part entière, offrant des structures de contrôle (tests, boucles, procédures, récursion) et des E/S élémentaires. On dispose également de types : entiers, réels, booléens, chaînes de caractères et de structures de données : tableaux, fichiers, dictionnaires. On remarquera peut-être l'influence du langage Forth (notation postfixée, notion de pile d'opération, assimilation code - données)

Les programmes PostScript sont en principe produits par les logiciels de traitement de texte et de dessin, mais on peut très bien en écrire soi-même. Un programme PostScript est un fichier textuel encodé en ASCII (donc qui ne continet aucun caractère accentué). Ce fichier est en général transmis à une imprimante laser qui est en fait un ordinateur spécialisé exécutant un programme qui interprète le PostScript reçu pour imprimer du papier. Il est également possible de faire lire le fichier PostScript à des programmes qui affichent le résultat à l'écran :

- avec Acrobat Distiller d'Adobe (conversion en PDF puis visualisation avec Acrobat Reader)

- avec Ghostscript d'Aladin. Freeware à récupérer sur `http://www.aladin.com`

Il n'est pas question ici de résumer toutes les fonctions de PostScript. Nous nous limiterons aux fonctions nécessaires pour l'ED.

Pour en savoir plus : plein d'info sur le web et les news. Pour lire du papier :

• Adode Systems Inc. *Manuel de référence du langage PostScript* 2ème édition. Addison-Wesley 1992 (téléchargeable sur `http://www.adobe.com`)

• Adobe Systems Inc. *PostScript par l'exemple. Outils pour l'édition électronique.* InterEditions 1987

• Bernard-Paul Eminet *Le livre de PostScript* Editions P.S.I. 1987

• P. Blanc *PostScript, l'essentiel* Eyrolles 1993

• Ph. Gille *PostScript facile* Marabout service 1989

• J.D. Fekete *Langages de description de page* (article H7018 des Techniques de l'ingénieur, en ligne depuis le CNAM).

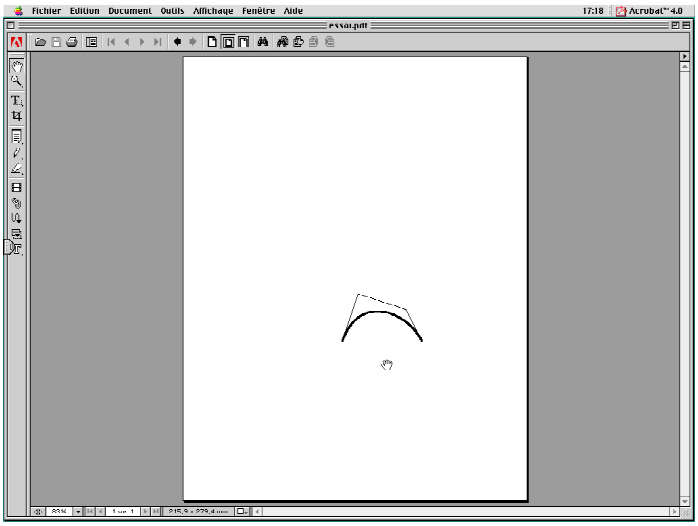
Exercice 1 : Tracé de courbes de Bézier

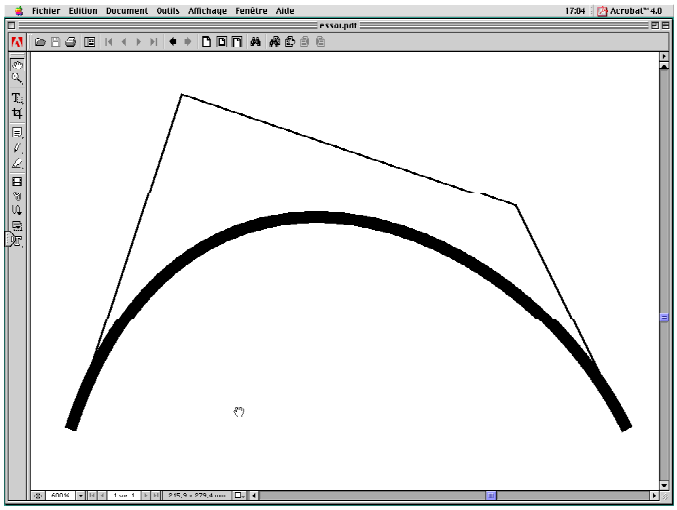
Il est très simple de demander à PostScript de dessiner des courbes de Bézier. Voilà un programme PS de base :

```
% definitions
 /centimetre {72 mul 2.54 div} def
 1 centimetre 1 centimetre scale
 0.1 setlinewidth
% je trace la courbe de Bezier
 newpath
  10 10 moveto
  11 13 14 12 15 10 curveto
 stroke
% je relie les points de controle
 0.01 setlinewidth
 newpath
```

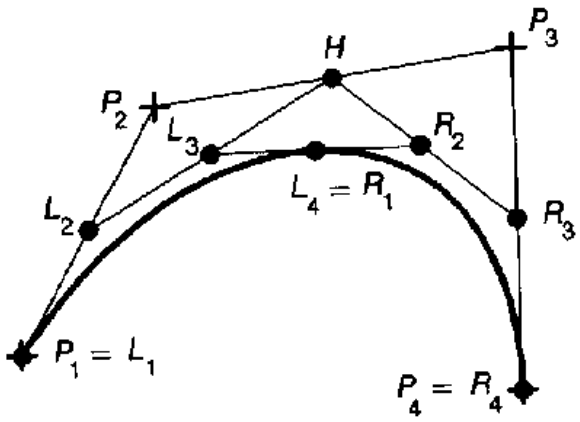
```
  10 10 moveto
  11 13 lineto
  14 12 lineto
  15 10 lineto
 stroke
% impression de la page
showpage
```

Et son résultat visualisé avec Acrobat Reader (à gauche, la vue en pleine page - à droite : zoom)





Le tracé de courbes de Bézier s'effectue de manière efficace avec l'algorithme de De Casteljau.



Soient P1, P2, P3, P4 les 4 points de contrôle de la courbe, pour obtenir le point L4 (=R1) on calcule succesivement :

(1) L1 = P1
(2) L2 = (P1+P2)/2
(3) H = (P2+P3)/2
(4) L3 = (L2+H)/2
(5) R3 = (P3+P4)/2
(6) R2 = (H+R3)/2
(7) R1 = (L3+R2)/2
(8) L4 = R1

Comment obtient-on d'autres points de la courbe ?

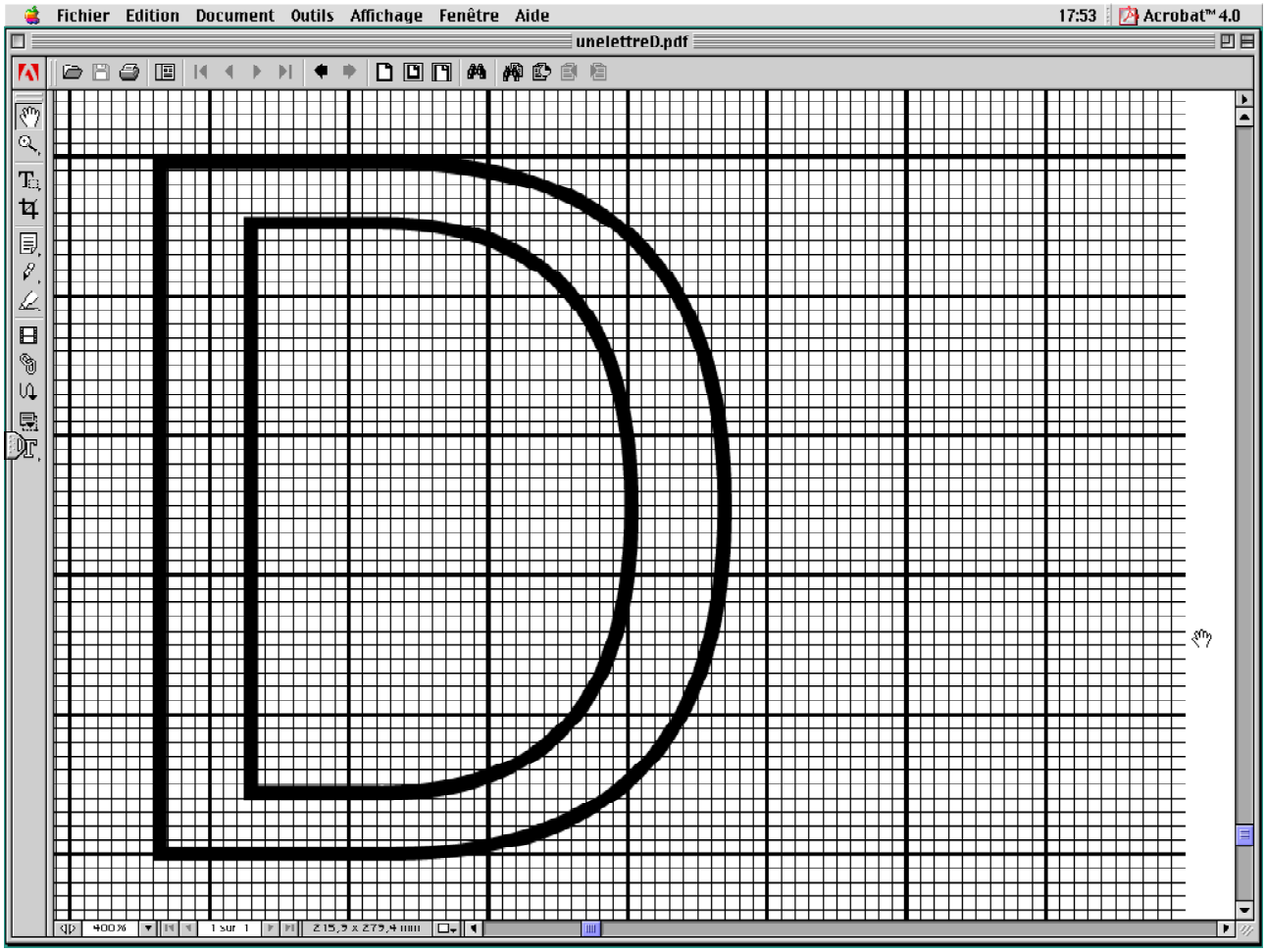
En déduire un algorithme récursif de tracé

Transformer l'algorithme en schéma itératif, avec une liste chaînée

Exercice 2 : Dessin de caractères

Les polices PostScript niveau 1 permettent de définir des contours qui contiennent des segments de courbes de Bézier. C'est le cas par exemple de la police PostScript standard Helvetica.

Retrouver approximativement le chemin de tracé du caractère suivant. On situera en particulier les segments de Bézier et leurs points de contrôle



RQUE : Il existe de nombreux opérateurs pour afficher du texte en PostScript. Par exemple :

```
/Times-Roman findfont 72 scalefont setfont
100 100 moveto
(Cnam) show
showpage
```

L'interpréteur PostScript dessine tous les caractères d'une page comme il traite les autres graphiques. Pourtant, on peut penser que cette action est assez coûteuse et certainement très redondante.

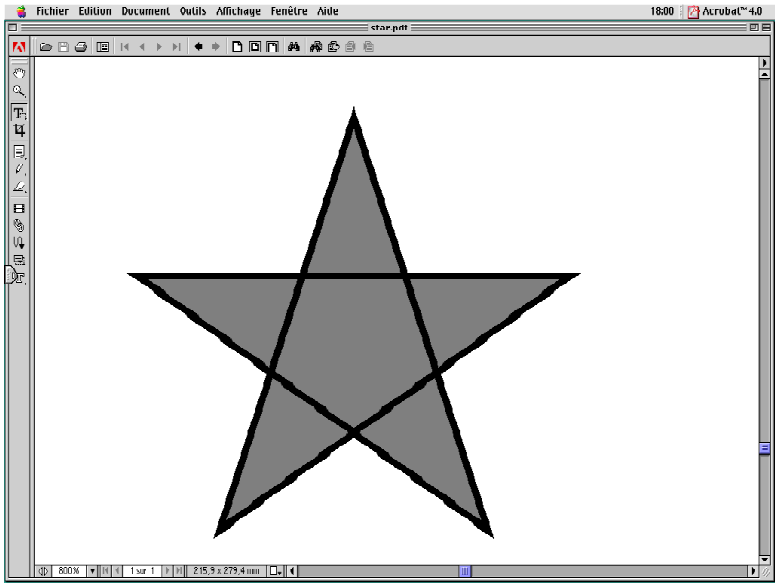
Quel mécanisme peut-on implanter dans l'interpréteur pour en tenir compte ?

Exercice 3 : Une première fractale

[d'après le Tutoriel PostScript p. 51 et p. 73]

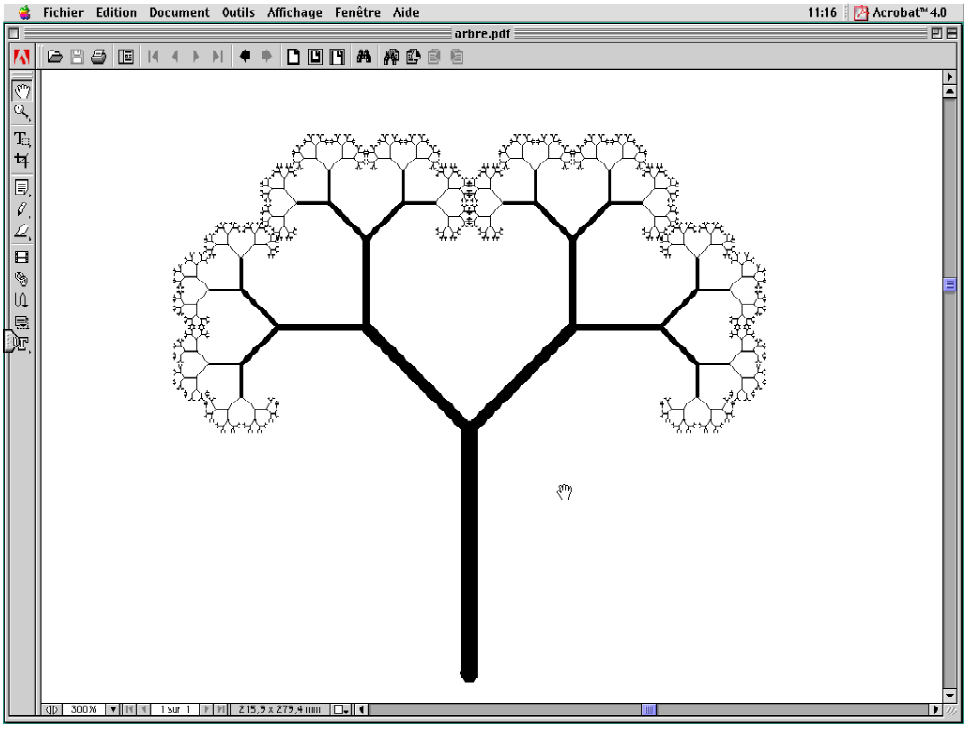
Modifier le système de coordonnées (origine du repère et/ou orientation des axes et/ou facteur d'échelle) pendant le dessin permet souvent d'en simplifier la spécification.

Le programme ci-dessous permet d'étudier les opérations les plus courantes de manipulation du système de coordonnées en cours. Son but est de dessiner une étoile à cinq branches, avec l'intérieur rempli et le bord redessiné ("outlined").




```
/starside {
 72 0 lineto
 currentpoint translate
 -144 rotate
} def

/star {
 moveto % on suppose que x et y sont en pile
 currentpoint translate
 5 { starside } repeat

 gsave                  % fill efface le chemin courant
 0.5 setgray fill       % on le sauve avant
 grestore               % on le recupere pour stroke qui suit
 stroke
} def

200 200 star
showpage
```

On souhaite maintenant écrire un programme qui produit des dessins du type suivant :



Compléter le canevas de programme ci-dessous

On pourra aussi se baser sur le programme Java vu en cours (chap. 4)

```
/Prof 0 def
/ProfMax 10 def

/Branche {
  0 144 rlineto
  currentpoint translate
  stroke
  0 0 moveto
} def

/Arbre {
   %%%%% A COMPLETER
} def

%--- programme principal
1 setlinecap 1 setlinejoin
300 400 moveto
FlecheFractale
stroke
showpage
```

Modifier le programme pour que l'angle des branches soit choisi au hasard à chaque itération

Que faire pour que le programme dessine ensuite une forêt de 50 arbres ?

Exercice 4 : Un pavage en hommage à Escher

[basé sur le programme escher.ps de Bob Wallis UUCP {sun, pyramid, cae780, apple}! weitek! wallis - ce programme est entre autre diffusé avec GhostView]

Paver le plan consiste à déposer de manière répétitive une forme de telle sorte que toute la surface du plan soit recouverte. Si on utilise comme forme uniquement un polygone régulier, on doit se limiter au triangle (équilatéral), au carré ou à l'hexagone. En revanche, on ne peut pas utiliser un pentagone (par ex.). Dans son œuvre maintenant universellement connue, Escher (1898-1972) a très souvent exploré l'univers graphique par la technique des pavages, s'inspirant en particulier des mosaïques maures.

RQUE : On trouve énormément d'informations sur les pavages (tessalation, tilings en anglais) sur le Web. Je recommande particulièrement le site "Totally Tessellated" :

`http://www.abc.lv/thinkquest/tq-entries/16661/index2.html`

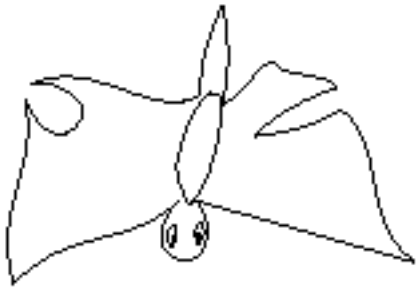
Dans son programme postscript (qui est plus complexe que notre exercice), Bob Wallis fournit la description d'une forme élémentaire : un papillon - composé essentiellement d'arcs de Bézier. Les points ont été obtenus par export d'un fichier issu d'un logiciel de dessin (Adobe Illustrator).

```
/m { moveto} def
/c {curveto} def
/CS {closepath stroke} def

/body {
     314.96 280.19 m
     383.4 261.71 445.11 243.23 513.52 224.68 c
     463.68 256.59 490.26 328.83 446.99 360.76 c
     423.71 347.32 397.08 339.7 367.07 337.9 c
     388.93 358.28 414.14 372.84 442.73 381.58 c
     426.68 398.18 394.07 389.7 387.2 403.84 c
     371.52 404.96 362.56 372.48 340.16 366.88 c
```

```
     346.88 396.01 346.88 425.12 340.16 454.24 c
     326.72 427.35 320 400.48 320 373.6 c
     270.71 352.1 221.44 411.23 168.88 384.02 c
     189.04 388.03 202.48 380.4 212.57 366.95 c
     216.72 350.85 209.23 341.46 190.1 338.79 c
     177.34 343.57 167.94 354.17 161.9 370.59 c
     176.06 305.52 132.02 274.05 152 205.6 c
     201.29 257.12 250.56 234.72 299.84 279.52 c
     288.64 266.08 284.16 252.64 286.4 239.2 c
     298.27 223.97 310.15 222.18 322.02 233.82 c
     328.62 249.28 328.51 264.74 314.96 280.19 c
     CS
} def

/eyes {
     294.8125 238.3246 m
     296.9115 238.3246 298.6132 242.7964 298.6132 248.3125 c
     298.6132 253.8286 296.9115 258.3004 294.8125 258.3004 c
     292.7135 258.3004 291.0118 253.8286 291.0118 248.3125 c
     291.0118 242.7964 292.7135 238.3246 294.8125 238.3246 c
     CS
     319.5 241.1782 m
     321.7455 241.1782 323.5659 245.4917 323.5659 250.8125 c
     323.5659 256.1333 321.7455 260.4468 319.5 260.4468 c
     317.2545 260.4468 315.4341 256.1333 315.4341 250.8125 c
     315.4341 245.4917 317.2545 241.1782 319.5 241.1782 c
     CS
     296.875 242.0939 m
     297.4608 242.0939 297.9356 243.479 297.9356 245.1875 c
     297.9356 246.896 297.4608 248.2811 296.875 248.2811 c
     296.2892 248.2811 295.8143 246.896 295.8143 245.1875 c
     295.8143 243.479 296.2892 242.0939 296.875 242.0939 c
     CS
     318.5 243.7707 m
     319.281 243.7707 319.9142 245.0766 319.9142 246.6875 c
     319.9142 248.2984 319.281 249.6043 318.5 249.6043 c
     317.719 249.6043 317.0858 248.2984 317.0858 246.6875 c
     317.0858 245.0766 317.719 243.7707 318.5 243.7707 c
     CS
} def


/thorax
{
     327.5 300 m
     316.5 283 315.5 275.5 308 277.5 c
     294 311.5 299 313.5 304 334 c
     309 354.5 315.5 362 322.5 372 c
     329.5 382 327.5 376.5 331 376 c
     334.5 375.5 339.1367 379.1109 339 369 c
     338.5 332 333.4999 324.5 330.5 311.5 c
     CS
```

```
} def

/bfly {                    % paint 1 butterfly
     body
     eyes
     thorax
} def
```

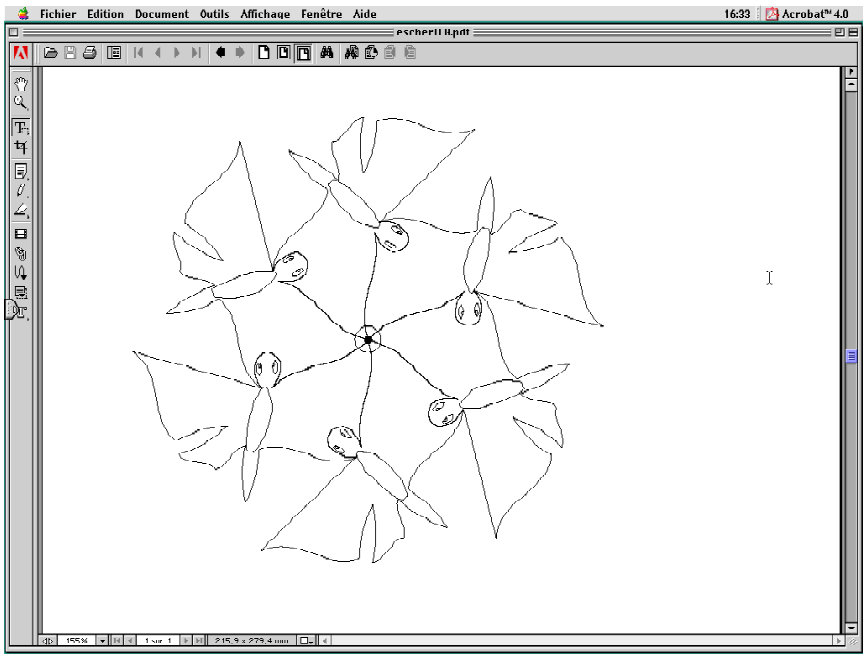
Lepapillon ainsi créé a la forme suivante :



A votre avis, comment peut-on paver le plan avec cette forme ? retrouver les coordonnées des centres de symétrie dans le code ci-dessus.

Ecrire un programme PostScript permettant de dessiner le papillon à une position quelconque de la page

Reprendre ensuite le programme pour dupliquer 6 fois le papillon par rotation autour du centre de symétrie :

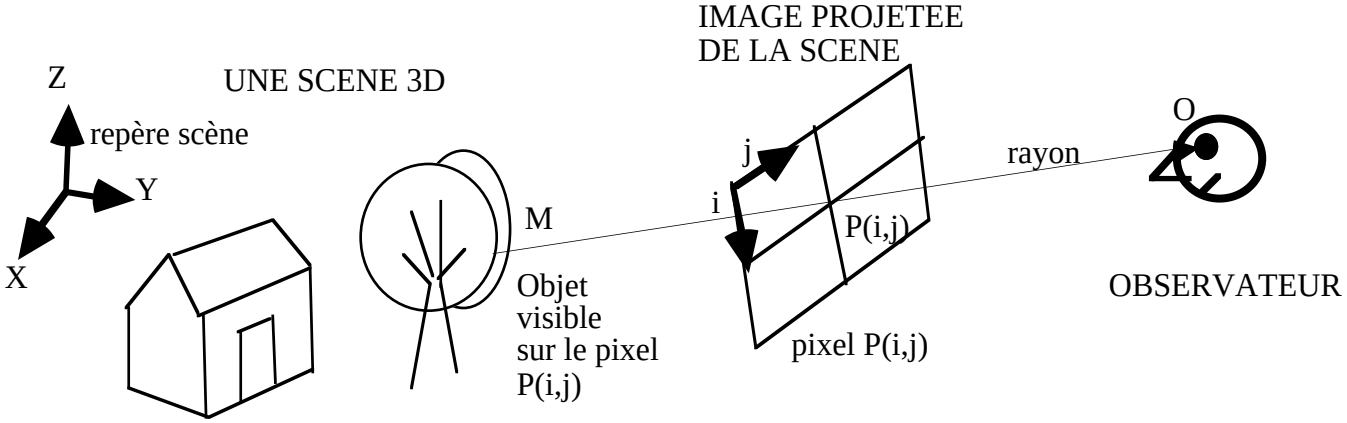


Ecrire le programme final de tesselation du plan

ED 3. Navigation

Exercice 1. La caméra synthétique

On ne considère ici qu'un repère orthonormé : les 3 axes X, Y, Z sont orthogonaux entre eux. L'origine du repère n'est pas décrite explicitement. Ce sont les objets de la scène dont on donne les coordonnées. En pratique, on choisit souvent comme origine le centre ou le socle d'un objet particulier de la scène, vers lequel regarde l'observateur. L'unité de mesure pour le repère n'a pas d'importance pour les calculs proprement dits. On choisira la plus commode selon le type de scène à dessiner (le parsec si on dessine des galaxies, ou le mètre si on dessine des maisons). Pour simplifier, on supposera ici que tous les objets sont décrits dans la même unité de mesure.



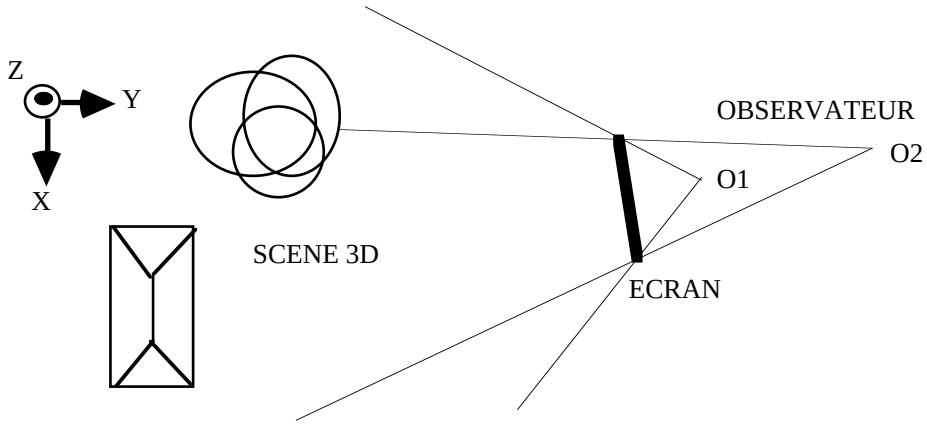
Question 1

Quel est le type du repère dessiné ci-dessus ?

Question 2

Combien de paramètres sont au total nécessaires pour définir l'orientation de la caméra ?

Il faut aussi définir la taille de l'écran (en hauteur et largeur), dans l'unité de mesure choisie pour la scène. On définit ensuite la distance entre l'observateur et l'écran. Ces trois autres paramètres définissent le volume de vue. Plus l'observateur est proche de l'écran, plus le volume de vue s'agrandit :



Question 3. Changement de repère.

On suppose connus :

(Rx, Ry, Rz) : les coordonnées du point de **R**éférence de l'écran (son centre)

(Cx, Cy, Cz) : les coordonnées du point de vue **C**hoisi dans la scène

(Hx, Hy, Hz) : le vecteur définissant le **H**aut de l'écran

D : la distance entre l'observateur et l'écran

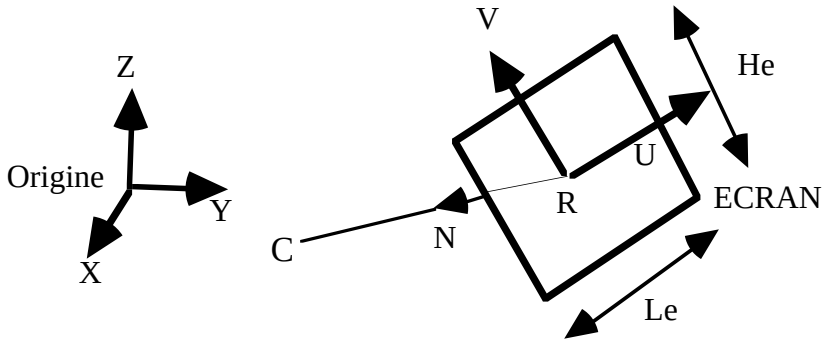
He : la hauteur de l'écran (dans l'unité de mesure du repère 3D)

Le : la largeur de l'écran (dans l'unité de mesure du repère 3D)

HAUT : la hauteur de l'écran (en pixels)

LARG : la largeur de l'écran (en pixels)

En déduire les coordonnées des vecteurs U, V et N dans le repère de la scène



Soit P, un point de la scène, de coordonnées (Px, Py, Pz) dans le repère (O, X, Y, Z).

Comment calculer ses coordonnées (P'x, P'y, P'z) dans le repère (R,U,V,N) ?

Question 4. Coordonnées projetées.

On cherche ensuite les coordonnées du point projeté sur l'écran (P*x, P*y). Celui-ci est par définition à l'intersection entre le plan de l'écran et la droite reliant P et l'observateur O.

*Calculer (P*x, P*y)*

En déduire une condition pour qu'un point soit visible sur l'écran

Question 5. Coordonnées image

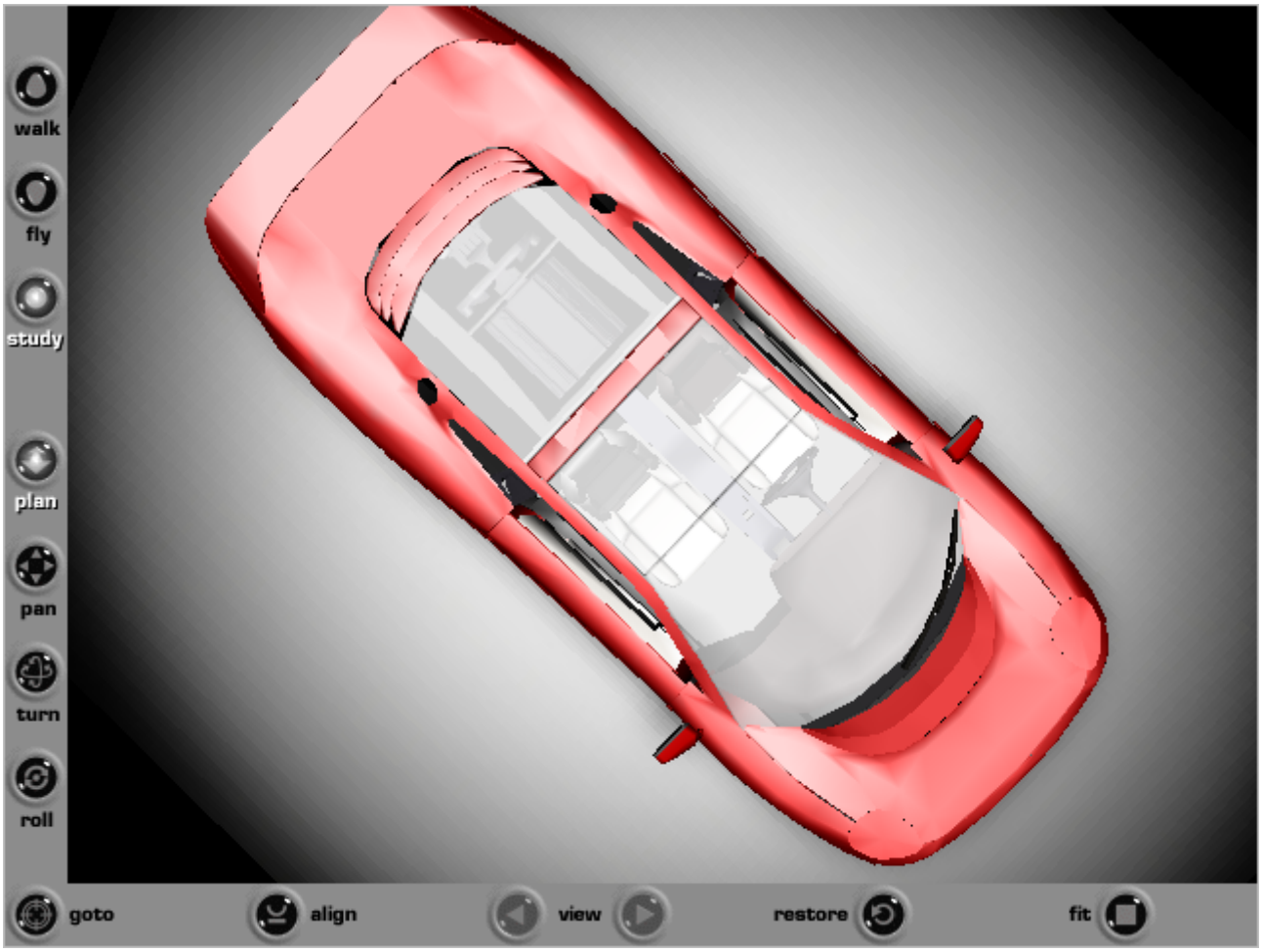
L'origine de l'image est le coin en haut à gauche. C'est le pixel (1,1). On veut donc que le pixel (1,1) ait les coordonnées (-Le/2, +He/2). De même, le pixel (LARG, HAUT) doit avoir les coord. (+Le/2, -He/2)

*En déduire les coordonnées image (i,j) pour un point (visible) quelconque (P*x, P*y)*

Exercice 2. Navigation dans un client VRML

Pour naviguer dans une scène 3D, l'utilisateur modifie la position de la caméra. Dans un logiciel client VRML, cette modification provoque "immédiatement" un nouvel affichage de la scène.

Voici par exemple, comment est visualisée une scène VRML avec Cortona (v. 1.0 pour mac) de Parallel Graphics :



Question 1

Quels sont les deux principaux modes de manipulation de la caméra ?

Question 2

Décrivez un périphérique adapté à la manipulation de la caméra synthétique

Question 3

Comment assurer les mêmes fonctions avec une souris conventionnelle ?

Question 4

A quoi correspondent les 4 variantes de Cortona ?