Infographie

François Malgouyres
Département d'informatique de l'Université Paris 13
http://www.math.univ-paris13.fr/∼malgouy
malgouy@math.univ-paris13.fr

Contents

Chapter 1

Images 2D

1.1 Le dessin sous JAVA

Pour dessiner dans une applet JAVA, nous utiliserons un objet *Graphics2d* dans lequel nous avons placé un *Panel*. Dans ce *panel* nous pouvons afficher un point de coordonnées entières (x, y) (à l'aide d'une méthode *dessinePoint(x,y,dessin)*, où *dessin* est une instance de *Graphics2d*). Nous nous restreindrons de plus à une fenêtre de taille 700 par 700. Le système de coordonnées de JAVA fait ainsi correspondre à un point $(x, y) \in \{0, ..., 699\}^2$ le point représenté sur la Figure 1.1. Notez que l'axe des ordonnées se trouve par conséquent orienté vers le bas.

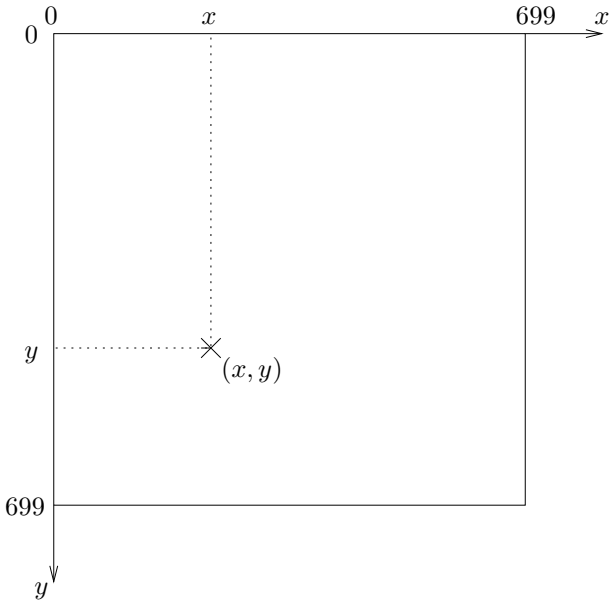

Figure 1.1: Système de coordonnées dans une applet JAVA.

1.2 Dessiner une droite

Dans cette section nous expliquons deux méthodes pour tracer un segment de droite. Ce dernier est donné par ses deux extrémités: $A = (x_a, y_a)$ et $B = (x_b, y_b)$ deux points de coordonnées entières (x_a, y_a, x_b, y_b sont dans $\mathbb{N}$). On suppose de plus que ces deux points sont à l'intérieur de la fenêtre dans laquelle nous pouvons dessiner.

La première difficulté pour tracer le segment de droite entre A et B est que l'on ne peut afficher à l'écran que des points de coordonnées entières et que les points du segment $[A, B]$ ne le sont que rarement. Ceci nous conduit à discrétiser le segment de droite (cf Figure 1.2).

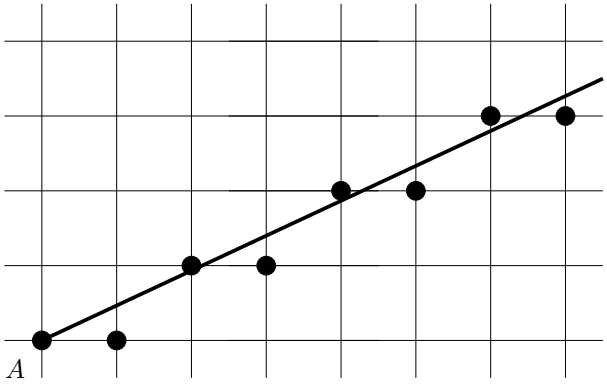

Figure 1.2: Discrétisation d'un segment de droite.

Pour simplifier la présentation, nous supposerons durant toute la section que A est à gauche par rapport à B ($x_a < x_b$) et que le module de la pente ($|p| = \left|\frac{y_b - y_a}{x_b - x_a}\right|$) est plus petit que 1. Les autres cas se déduisent simplement de ce dernier. Nous indiquerons, lorsque cela est nécessaire, les changements à effectuer pour tracer un segment quelconque.

1.2.1 Algorithme naïf

Ici, nous allons caractériser la droite joignant A à B par l'équation

$$y = p * (x - x_a) + y_a \,,$$

avec $p = \frac{y_b - y_a}{x_b - x_a}$.

L'algorithme naïf pour tracer le segment $[A, B]$ consiste à faire parcourir à un indice x toutes les valeurs entières entre x_a et x_b et pour chacune de ces valeurs à arrondir le y correspondant à l'entier le plus proche. Ceci se traduit en JAVA par l'algorithme suivant:

```
 /* Dessin d'une droite par l'algorithme naif */
public void dessineLigneNaif(int xa,int ya,int xb,int yb,Graphics2D dessin)
    {int x,y;
    float yflt;
    float pente;

    pente=(float) (yb-ya)/(float)(xb-xa);
    yflt=ya+0.5F;
    for(x=xa;x<=xb;x++)
        {y=(int) Math.floor(yflt);
         dessinePoint(x,y,dessin);
         yflt=yflt+pente;
        }
    }
```

On remarque ici l'utilisation d'une méthode *dessinePoint* qui affiche le point de coordonnées (x, y) sur le graphique *dessin*. On la suppose déjà définie. De plus, on utilise la méthode *floor* de la classe *Math* qui retourne la partie entière du flottant *yflt*.

Pour généraliser cet algorithme au cas d'un segment quelconque, on peut tout d'abord intervertir les points A et B de façon à avoir $x_a \leq x_b$. Notez que le cas d'un segment vertical ($x_a = x_b$) ne pose aucune difficulté. Le cas d'une pente entre -1 et 0 se traite de la façon décrite ci-dessus. Enfin, dans le cas d'une

pente de module plus grand que 1 ($|p| > 1$), on parcourra l'axe des ordonnées pour arrondir la valeur de l'abscisse.

Cette méthode, même si elle marche, peut être améliorée. En effet, pour chaque point calculé, on a besoin d'une addition entre flottants et d'un arrondi, ce qui ralentit le tracé du segment. La partie suivante présente l'algorithme de Foley qui évitera ces deux opérations.

Exercice 1 *Écrire l'algorithme naïf pour tracer un segment dans le cas général.*

Exercice 2 *L'algorithme ci-dessus peut-être généralisé à une courbe régulière $y = f(x)$. Utilisez cette généralisation pour écrire un algorithme traçant le graphe d'une parabole ($y = a*x^2 + b*x + c$, avec $a > 0$, b et c dans $\mathbb{R}$) entre deux abscisses $x_0 = \frac{-2-b}{2a}$ et $x_1 = \frac{2-b}{2a}$. (Vous distinguerez les cas $|2*a*x+b| > 1$ et $|2*a*x+b| \leq 1$.)*

1.2.2 Algorithme de Foley

Pour la présentation de l'algorithme de Foley, nous supposerons que la pente est positive et plus petite que 1 ($0 \leq p = \frac{y_b - y_a}{x_b - x_a} \leq 1$). Cet algorithme est basé sur l'idée que pour tracer une telle droite, on va parcourir l'axe des abscisses et que l'ordonnée est, soit identique à celle du point précédent, soit augmentée de 1. Ceci n'est évidemment pas vrai pour d'autres valeurs de la pente et celles-ci feront l'objet d'un traitement analogue et approprié.

Si l'on note (x_p, y_p) le point courant, on note $E = (x_p + 1, y_p)$ (E pour est) et $SE = (x_p + 1, y_p + 1)$ (SE pour sud-est), on a donc $(x_{p+1}, y_{p+1}) = E$ ou SE (voir figure 1.3). Afin de savoir quelle est la bonne solution (E ou SE), on va simplement déterminer de quel coté de la droite se trouve le point $M_p = (x_p + 1, y_p + \frac{1}{2})$. En effet, nous verrons que cela peut-être fait rapidement. On choisira ainsi:

- Si le point M_p est au dessus de la droite, $(x_{p+1}, y_{p+1}) = SE$
- Si le point M_p est au dessous de la droite, $(x_{p+1}, y_{p+1}) = E$.

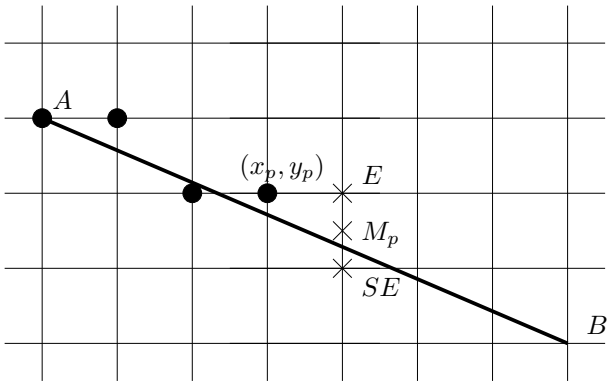

Figure 1.3: La méthode de Foley.

Pour déterminer la position du point M_p par rapport à la droite, on regarde en M_p le signe de la fonction

$$f(x, y) = dy * (x - x_a) - dx * (y - y_a),$$

avec $dx = x_b - x_a$ et $dy = y_b - y_a$. (Rappel: l'équation cartésienne de la droite (A, B) est: $f(x, y) = 0$. De plus, étant donné le système de coordonnées utilisé par JAVA, si $f(x, y) < 0$: (x, y) est sous la droite; et si $f(x, y) > 0$: (x, y) est au dessus de la droite.) En fait, afin de n'avoir à manipuler que des entiers, on va regarder le signe

$$d_p = 2\, f(M_p).$$

Afin de gagner du temps, le calcul de d_p est fait de façon incrémental. Pour cela, on distingue deux cas:

- **Si** $(\mathbf{x_{p+1}}, \mathbf{y_{p+1}}) = (\mathbf{x_p} + \mathbf{1}, \mathbf{y_p}) = \mathbf{E}$**:**

$$\begin{aligned} d_{p+1} &= 2\,f(x_{p+1}+1, y_{p+1}+\frac{1}{2}) \\ &= 2\,[dy*(x_p+2-x_a) - dx*(y_p+\frac{1}{2}-y_a)] \\ &= 2\,[dy*(x_p+1-x_a) - dx*(y_p+\frac{1}{2}-y_a)] + 2\,dy \\ &= d_p + 2\,dy \end{aligned}$$

- **Si** $(\mathbf{x_{p+1}}, \mathbf{y_{p+1}}) = (\mathbf{x_p} + \mathbf{1}, \mathbf{y_p} + \mathbf{1}) = \mathbf{SE}$**:**

$$\begin{aligned} d_{p+1} &= 2\,f(x_{p+1}+1, y_{p+1}+\frac{1}{2}) \\ &= 2\,[dy*(x_p+2-x_a) - dx*(y_p+1+\frac{1}{2}-y_a)] \\ &= 2\,[dy*(x_p+1-x_a) - dx*(y_p+\frac{1}{2}-y_a)] + 2\,(dy-dx) \\ &= d_p + 2\,(dy-dx) \end{aligned}$$

On a bien sûr initialement

$$\begin{aligned} d_0 &= 2\,f(x_a+1, y_a+\frac{1}{2}) \\ &= 2\,[dy*(x_a+1-x_a) - dx*(y_a+\frac{1}{2}-y_a)] \\ &= 2\,dy - dx. \end{aligned}$$

Ceci nous conduit à l'algorithme suivant

```
 /* Dessin d'une droite par l'algorithme de Foley */
public void dessineLigneFoley(int xa,int ya,int xb,int yb,Graphics2D dessin)
    {int x,y;
    int d;
    int dx=xb-xa,dy=yb-ya;
    int dvarE,dvarSE;

    d=2*dy-dx; // valeur de dp pour p=0
    dvarE=2*dy; // incrementation de dp dans le cas E
    dvarSE=2*(dy-dx); // incrementation de dp dans le cas SE

    dessinePoint(xa,ya,dessin);

    for(x=xa+1,y=ya;x<=xb;x++)
        {if(d<=0)
           d=d+dvarE;
         else
           {y=y+1;
            d=d+dvarSE;}
         dessinePoint(x,y,dessin);
        }
    }
```

Il faut remarquer ici que $dvarE \geq 0$ (on l'ajoute a d lorsque celui-ci est négatif) et que $dvarSE \leq 0$ (on l'ajoute a d lorsque celui-ci est positif). Ainsi, on cherche toujours à ramener d vers 0. Ce sera le cas pour toutes les valeurs de la pente du segment $[A, B]$.

Les cas où la pente n'est pas entre 0 et 1 demandent de faire un raisonnement similaire à celui que nous venons de faire. Il faut à chaque fois:

- Déterminer l'axe que l'on parcoure et les deux points possibles pour accueillir (x_{p+1}, y_{p+1}) (parmi N, NE, E, SE et S).
- Déterminer la position du point où l'on calculera $2\,f$ (en fait le milieu des deux points trouvés à la première étape).
- Calculer d_0 et les deux variations possibles pour passer de d_p à d_{p+1}

Exercice 3 *Écrire l'algorithme de Foley pour une droite quelconque.*

Exercice 4

1. *Utilisez toutes les symétries du cercle pour pouvoir déduire son tracé complet du calcul des coordonnées de points situés sur un arc de longueur minimale.*
2. *Adaptez l'algorithme de Foley pour le traçage de l'arc d'un cercle dont l'angle au centre est compris entre* 0 *et* $\frac{\pi}{4}$.
3. *Écrire un algorithme rapide pour tracer un cercle de centre et rayon donnés.*

1.3 Remplir un polygone

1.3.1 Définition de l'intérieur

Remplir un polygone revient à déterminer les points intérieurs à ce polygone. Pour cela, on utilise une définition utilisée depuis des siècles en Mathématiques:

On dit qu'un point est intérieur à un polygone si toutes les demi-droites partant de ce point intersectent le polygone un nombre impair de fois (points tangents non compris)

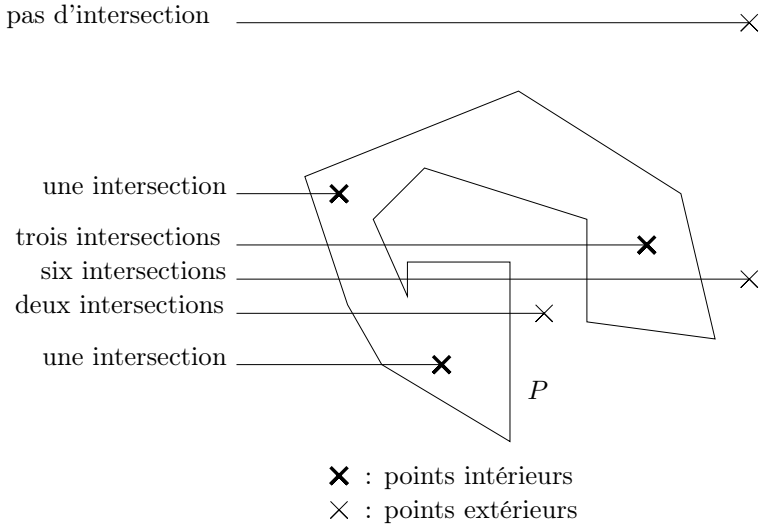

Figure 1.4: Les points intérieurs sont caractérisés par le nombre d'intersections d'une demi-droite avec le polygone.

Cette définition est illustrée sur la Figure 1.4.

Il est à noter (autre propriété mathématique) que la parité du nombre d'intersections ne dépend pas de la demi-droite choisie. Ainsi, nous prendrons dans la suite toujours la demi-droite horizontale allant vers la gauche (comme sur la Figure 1.4). De plus, cette notion d'intérieur ne correspond pas forcément à notre intuition lorsque le polygone à remplir s'auto-intersecte (voir Figure 1.5). Malgré cela, elle est pratiquement toujours utilisée en infographie.

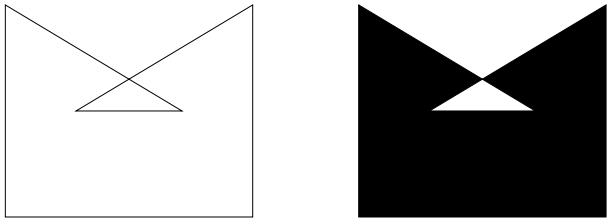

Figure 1.5: Droite: polygone de départ qui s'auto-intersecte. Gauche: On a noirci l'intérieur du polygone de droite.

Cette notion d'intérieur est bien définie pour les points loin des arêtes du polygone P. Cependant, elle reste ambiguë sur les arêtes. En effet, si l'on veut éviter que l'intérieur d'un polygone ne déborde sur un polygone adjacent, il faut décider du sort des points de coordonnées entières des arêtes car ils peuvent appartenir à deux polygones différents. Il faut adopter une règle qui décidera à l'intérieur de quel polygone sont les points ambigus (cf Figure 1.6).

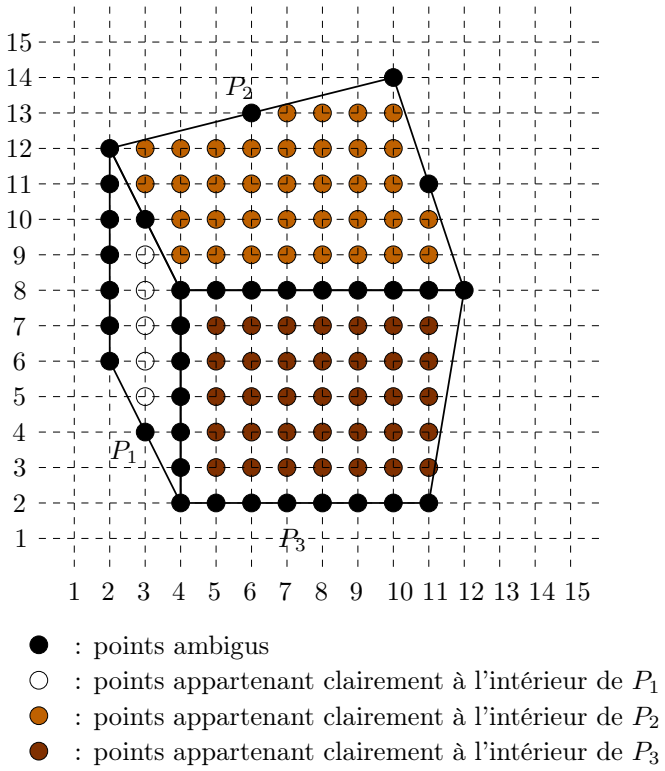

Figure 1.6: Points dont l'appartenance à l'un des polygones est ambiguë.

Nous adopterons donc une première convention: un point de coordonnées entières et situé sur une arête est à l'intérieur du polygone s'il s'agit d'une arête située à gauche du polygone. De même, afin de lever les ambiguïtés pour les arêtes horizontales, nous considérerons que l'arête est à l'intérieur du polygone si elle est située en dessous de celui-ci. Ceci nous conduit dans le cas des polygones de la Figure 1.6 au remplissage représenté sur la figure 1.7.

1.3.2 Principe du remplissage

Nous allons supposer dans la suite un remplissage **ligne après ligne** du polygone. Dans un tel cas, le remplissage du polygone consiste pour une ligne $y = y_0$ à:

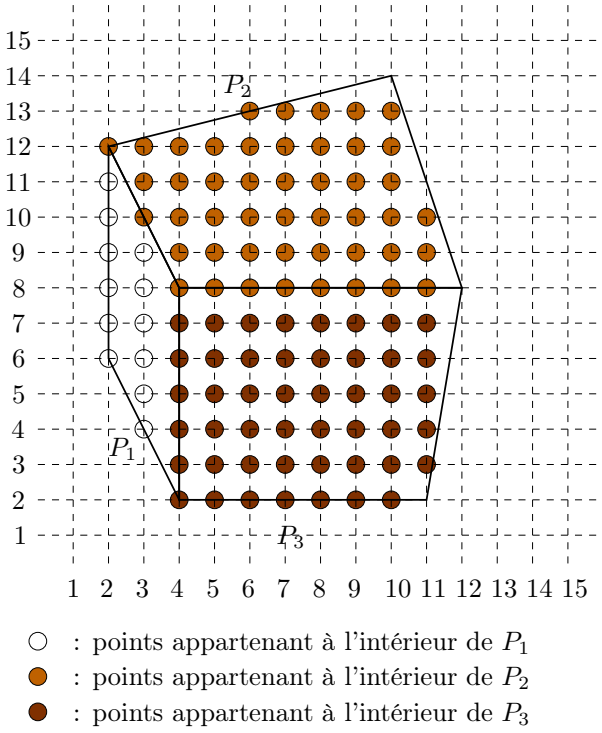

Figure 1.7: Remplissage des polygones après avoir levé les ambiguïtés.

- Déterminer toutes les abscisses (x_0, x_1, x_2,...) des intersections d'une ligne ($y = y_0$) avec le polygone (cf Figure 1.8). En pratique, on détermine les abscisses des points d'intersections dans un tableau *frontiere* que l'on construit pour chaque ligne (cf section suivante).

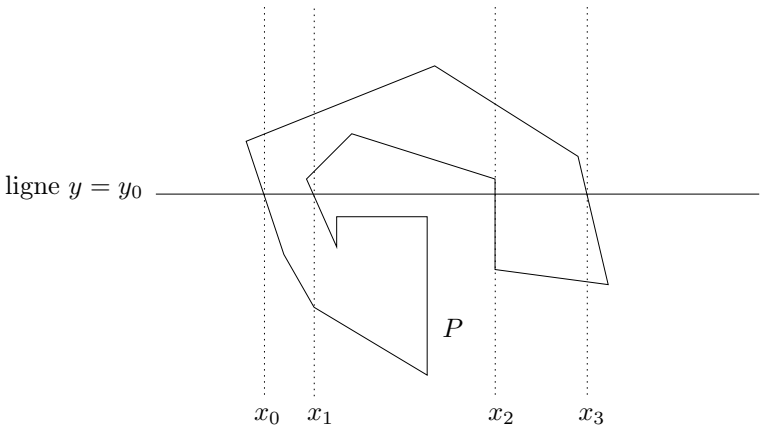

Figure 1.8: Abscisses des intersections de la ligne $y = y_0$ avec le polygone P.

- Enfin, on affiche les points entre les abscisses x_i et x_{i+1} pour i pair. Par exemple, les demi-droites horizontales issues des points d'abscisses comprises entre x_0 et x_1 et allant vers la gauche n'intersectent le polygone qu'en (x_0, y_0) (une intersection!). De même, les demi-droites horizontales issues des points d'abscisses comprises entre x_1 et x_2 et allant vers la gauche intersectent le polygone en (x_0, y_0) et (x_1, y_0) (deux intersections!). Etc...

Le point délicat dans le remplissage d'un polygone est en fait la construction de la *frontiere*. Ceci fait l'objet de la section suivante.

1.3.3 La frontière du polygone

Pour créer la frontière du polygone que l'on cherche à remplir, nous allons devoir faire face à plusieurs difficultés. La principale difficulté découle du traitement des points du polygone tangents à une droite horizontale (extrémas ou arêtes horizontales). En effet, si l'on compte ces points comme une intersection, on obtient le résultat présenté sur la Figure 1.9. Il est alors évident qu'il serait une erreur d'afficher entre x_0 et x_1.

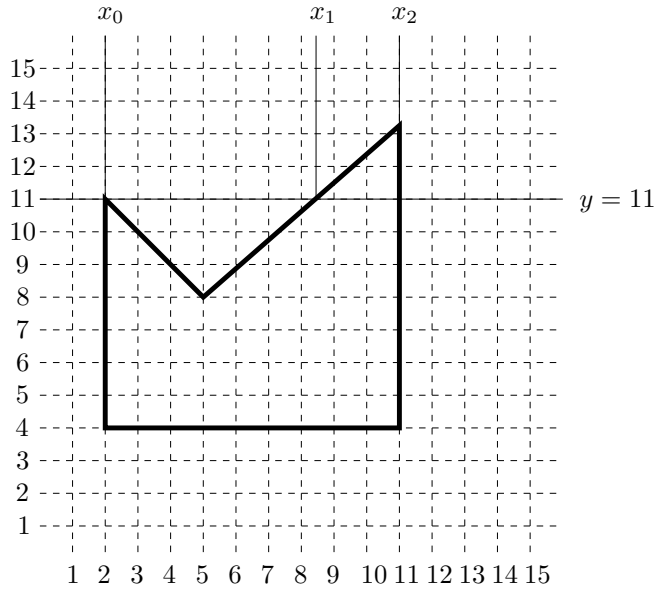

Figure 1.9: Erreur si l'on compte le point tangent un nombre impair de fois (ici une fois).

De même, dans le cas d'une arête horizontale, on obtient une erreur similaire lorsque la longueur de cette arête contient un nombre impair de points d'abscisses entières (voir Figure 1.10). Il est de même évident ici que l'on veut éviter d'afficher les points entre $(x_2, 11)$ et $(x_3, 11)$.

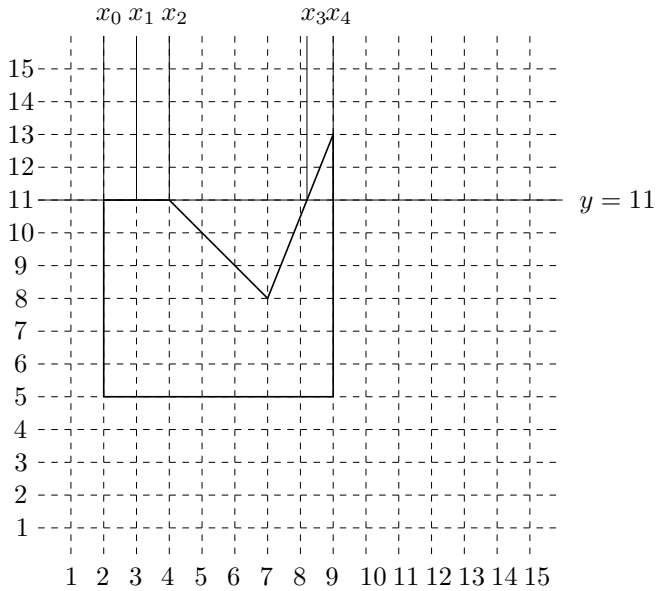

Figure 1.10: Erreur si l'on compte les points de l'arête horizontale une fois.

Afin d'éviter ces deux situations, nous adopterons les techniques suivantes:

- **Pour les points tangents:** Remarquez d'abord que cela ne peut se produire qu'à un sommet du polygone et donc à l'extrémité de deux arêtes. Remarquez ensuite que pour contourner ce problème, il suffit de compter cette extrémité un nombre pair de fois (en pratique 0 ou 2). Remarquez enfin que si l'on adoptait une technique consistant à compter tous les sommets un nombre pair de fois, on aurait des erreurs dans le cas présenté sur la figure 1.11. En effet, on veut afficher les points entre x_1 et x_2. Il y a plusieurs possibilités pour résoudre ce problème. Nous adopterons la plus simple

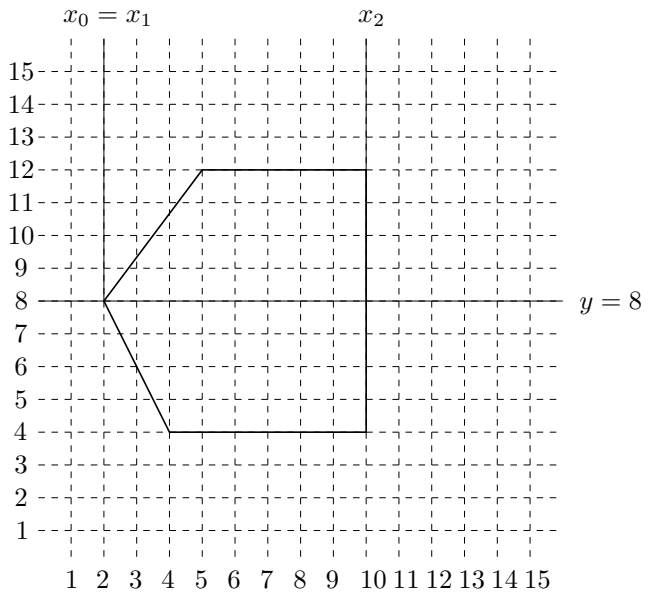

Figure 1.11: Erreur si l'on compte deux fois tous les sommets (on veut afficher les points entre x_1 et x_2).

(du point de vue de l'implémentation): lorsque la droite $y = y_0$ rencontre une arête du polygone, nous comptons une intersection si et seulement si celle-ci ne correspond pas à l'extrémité supérieure de l'arête. Ainsi les maximums locaux seront comptés zéro fois, les sommets qui ne sont pas des extrémas seront comptés une fois et les minimums locaux seront comptés deux fois (une fois par arête).

- **Pour les arêtes horizontales:** Nous choisissons simplement de ne pas les traiter. Ainsi, elles seront affichées en fonction des autres arêtes qui arrivent à leurs sommets. Par exemple, sur la figure 1.12, les arêtes $[AB]$, $[CD]$ et $[KL]$ sont affichées mais les arêtes $[EF]$, $[GH]$ et $[IJ]$ ne le sont pas. En fait, les arêtes situées en dessous du polygone sont affichées mais pas celles situées au dessus.

1.3.4 La règle d'arrondi

Pour l'arrondi, il faut veiller à respecter la convention sur les arêtes gauches et droites. Ainsi, après avoir réordonné nos intersections, on sait qu'une intersection est sur un arête gauche, si elle est d'indice pair, et sur une arête droite, si elle est d'indice pair. On arrondit donc les éléments de frontière

- au dessus, s'ils ont un indice pair (arêtes gauches)
- au dessous, s'ils ont un indice impair (arêtes droites)

Pour le cas particulier ou l'abscisse contenue dans frontière est entière, pour être fidèle à notre convention, il suffit de retrancher 1 aux éléments, de frontière, entiers qui sont d'indices impairs.

1.3.5 L'algorithme de remplissage

L'algorithme de remplissage se schématise comme suit:

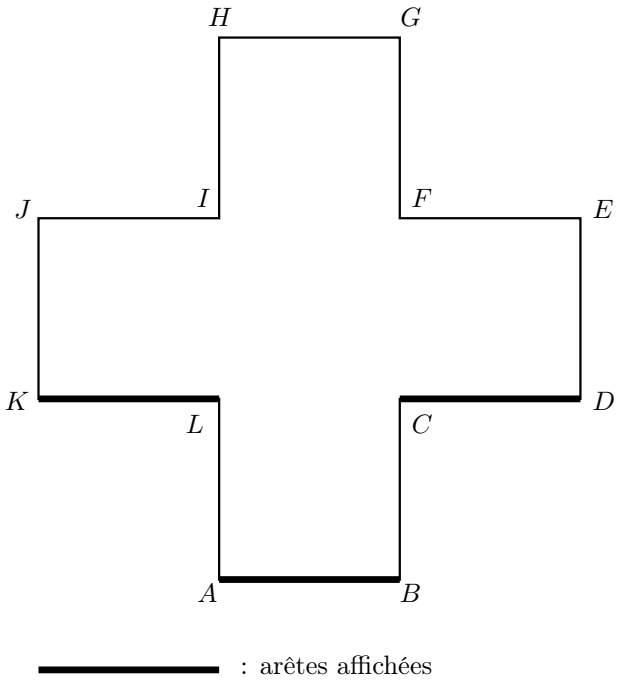

Figure 1.12: Les arêtes $[AB]$, $[CD]$ et $[KL]$ sont affichés. Les arêtes $[EF]$, $[GH]$ et $[IJ]$ ne sont pas affichés.

- Pour chaque valeur de y:
  - Pour chaque arête: si l'arête n'est pas horizontale et intersecte la ligne y en un point autre que le sommet le plus haut de l'arête: on détermine l'intersection et on la mémorise.
  - On classe les intersections par ordre croissant.
  - On arrondit les abscisses des intersections.
  - Pour chaque intersection d'indice pair: on affiche les points entre cette intersection et la suivante.

Exercice 5 *1. Concevez un algorithme qui détermine les points d'intersections d'une droite $y = y_0$ avec les sommets d'un polygone et qui mémorise le sommet:*

- 0 *fois s'il s'agit d'un maximum,*
- 2 *fois s'il s'agit d'un minimum,*
- 1 *fois sinon.*

2. Le compléter afin qu'il détermine toutes les intersections de la droite $y = y_0$ avec le polygone.

Exercice 6 *Écrire (en JAVA) une méthode effectuant l'algorithme donné en cours.*

Chapter 2

Images 3D

En infographie 3D, on veut représenter des objets. Ceux-ci sont donc des volumes. Dans le cas de volumes opaques, il suffit pour les dessiner de représenter leur partie visible qui est donc une surface. Ici, nous proposerons d'approximer une surface par un ensemble de polygones (des mailles). Ceci revient mathématiquement à adopter une représentation paramétrique de la surface. Notez qu'il existe d'autres façons de modéliser une surface. Notamment, la représentation implicite (la surface est l'ensemble des points où s'annule une fonction allant de $\mathbb{R}^3$ dans $\mathbb{R}$) se développe aujourd'hui considérablement.

2.1 Notions de maillage

Comme nous l'avons déjà dit, nous allons représenter une surface par un ensemble de polygones adjacents. Ceci nous amène donc à manipuler des **polyèdres**. Nous caractériserons un polyèdre par une suite de points P_0, P_1, ..., P_{n-1}: les sommets de polyèdre, et, une suite de mailles M_0, M_1, ..., M_{m-1}. Ces mailles seront simplement la donnée d'une suite ordonnée de sommets (en pratique repérés par leur indice) qui définissent la maille. Nous présentons dans la suite quelques exemples de formes simples. Ces exemples sont toujours d'orientation et de position fixes. Nous verrons plus tard comment implémenter des rotations et translations qui nous permettrons de modifier ces paramètres.

2.1.1 Le cube

Considérons le cube représenté sur la Figure 2.1. Nous le représenterons donc par:

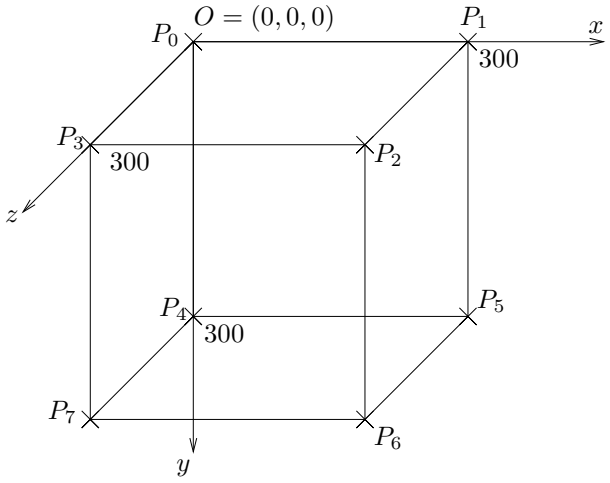

Figure 2.1: Cube de côté 300.

- ses sommets: $P_0 = (0,0,0)$, $P_1 = (300,0,0)$, $P_2 = (300,0,300)$, $P_3 = (0,0,300)$, $P_4 = (0,300,0)$, $P_5 = (300,300,0)$, $P_6 = (300,300,300)$ et $P_7 = (0,300,300)$.
- ses mailles: M_0 est le polygone défini par P_0, P_1, P_2 et P_3 (dans l'ordre) et est donc mémorisé sous la forme du quadruplet $M_0 = (0,1,2,3)$. De même, les autres faces du cube sont mémorisées sous la forme $M_1 = (0,1,5,4)$, $M_2 = (1,2,6,5)$, $M_3 = (2,3,7,6)$, $M_4 = (3,0,4,7)$ et $M_5 = (4,5,6,7)$.

Il reste maintenant à vérifier que les mailles sont bien des polygones. Il faut en effet pour cela que les points constituant la maille soient coplanaires. Ceci ne pose pas de problème dans le cas du cube car chaque maille est contenue dans un plan "simple" (ex: M_0 est dans le plan d'équation $y = 0$, M_1 est dans le plan d'équation $z = 0$,...).

2.1.2 Le cylindre

Nous définirons un cylindre par un rayon $r > 0$ et une hauteur $h > 0$. Ceci nous oblige donc à fixer l'orientation du cylindre et sa position. Comme nous l'avons déjà dit, ces paramètres pourront être modifiés ultérieurement. Ainsi, nous considérerons des cylindres centrés en $O = (0,0,0)$ et ayant pour axe de rotation l'axe des ordonnées (Oy) (voir Figure 2.2).

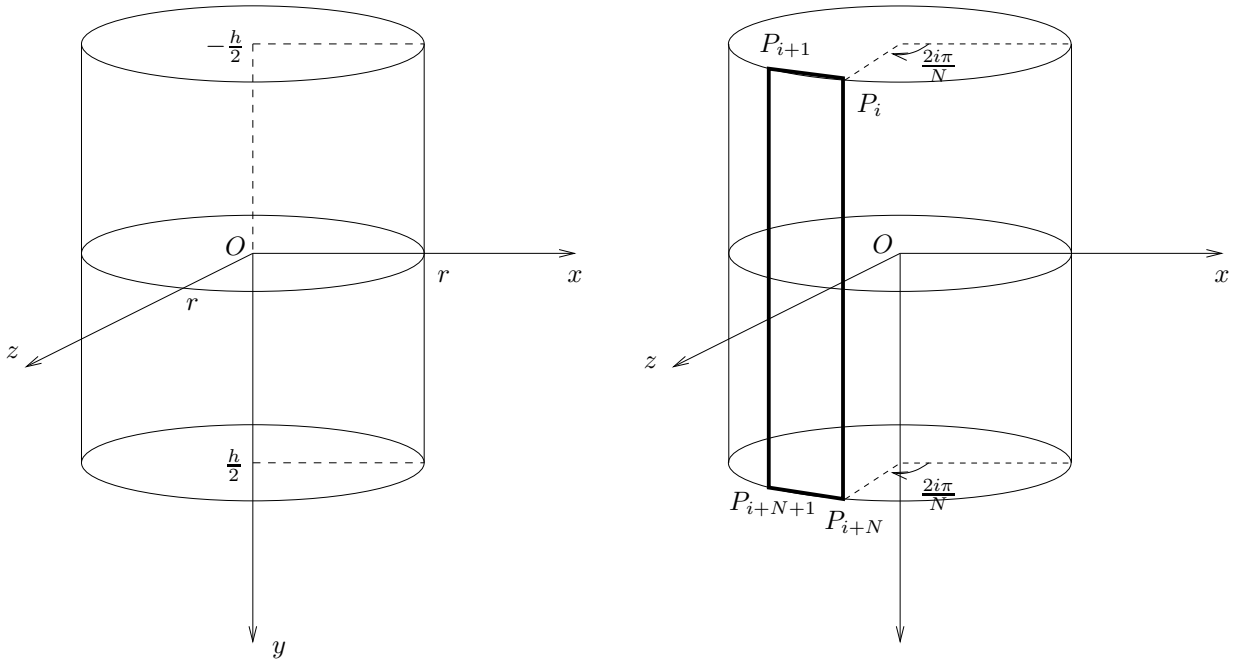

Figure 2.2: Gauche: Le cylindre de rayon r et de hauteur h (l'axe (Oy) et le centre du cylindre sont convenus). Droite: maille M_{i+2} pour $i \in \{0,1,...,N-2\}$.

Ce cylindre peut être décrit comme l'union de la face du haut, de la face du bas et du corps du cylindre. Ceux-ci s'expriment sous la forme analytique suivante:

- la face du haut:
$$\{(x,y,z),\ y = -\frac{h}{2} \text{ et } x^2 + z^2 \leq r^2\}$$
- la face du bas:
$$\{(x,y,z),\ y = \frac{h}{2} \text{ et } x^2 + z^2 \leq r^2\}$$
- le corps du cylindre:
$$\{(x,y,z),\ -\frac{h}{2} \leq y \leq \frac{h}{2} \text{ et } x^2 + z^2 = r^2\}$$

Afin de "mailler" ce cylindre, nous prendrons plusieurs mailles rectangulaires pour le corps du cylindre, une maille pour la face du haut et une maille pour la face du bas. Plus précisément, pour un maillage à $2N$ sommets, nous prendrons:

- ses sommets: $P_i = (r\cos(\frac{2i\pi}{N}), -\frac{h}{2}, r\sin(\frac{2i\pi}{N}))$ pour $i = 0,\ 1,\ ...,\ N-1$ (qui donnent les points de la face du haut) et $P_{i+N} = (r\cos(\frac{2i\pi}{N}), \frac{h}{2}, r\sin(\frac{2i\pi}{N}))$ pour $i = 0,\ 1,\ ...,\ N-1$ (qui donnent les points de la face du bas).
- ses mailles:
  - pour le corps du cylindre (voir Figure 2.2): $M_i = (i, i+1, i+N+1, i+N)$ pour $i = 0, 1, ..., N-2$ et pour la dernière maille $M_{N-1} = (N-1, 0, N, 2N-1)$.
  - pour la face du haut: $M_N = (0, 1, ..., N-1)$.
  - pour la face du bas: $M_{N+1} = (N, N+1, ..., 2N-1)$.

Dans le cas du cylindre, la vérification du fait que les mailles soient plates est un peu plus délicate. Pour les mailles représentant la face du haut et la face du bas, il va de soi qu'elles sont respectivement dans le plan $y = -\frac{h}{2}$ et le plan $y = \frac{h}{2}$. Pour les mailles du corps du cylindre, nous allons montrer que $\overrightarrow{P_iP_{i+1}} = \overrightarrow{P_{i+N}P_{i+1+N}}$. Ceci suffit en effet à garantir que P_i, P_{i+1}, P_{i+N} et P_{i+1+N} sont coplanaires. En effet, trois points (par exemple P_i, P_{i+N} et P_{i+1+N}) sont toujours coplanaires. Il reste à vérifier que le quatrième point (ici P_{i+1}) est bien dans le plan défini par P_i, P_{i+N} et P_{i+1+N}. Pour cela, il suffit qu'un vecteur partant d'un point du plan (par exemple P_i) et aboutissant en P_{i+1} soit lui même dans le plan en question et s'exprime donc sous une forme

$$\overrightarrow{P_iP_{i+1}} = \alpha_1\overrightarrow{P_{i+N}P_{i+1+N}} + \alpha_2\overrightarrow{P_{i+N}P_i}$$

(il va de soi ici que $\overrightarrow{P_{i+N}P_{i+1+N}}$ et $\overrightarrow{P_{i+N}P_i}$ "génèrent" notre plan).

Ainsi, si $\overrightarrow{P_iP_{i+1}} = \overrightarrow{P_{i+N}P_{i+1+N}}$, nous pouvons conclure que la maille est bien plate. Or, on a bien

$$\begin{aligned}
\overrightarrow{P_iP_{i+1}} &= \left(r\left(\cos(\frac{2(i+1)\pi}{N}) - \cos(\frac{2i\pi}{N})\right), 0, r\left(\sin(\frac{2(i+1)\pi}{N}) - \sin(\frac{2i\pi}{N})\right)\right) \\
&= \left(r\left(\cos(\frac{2(i+1+N)\pi}{N}) - \cos(\frac{2(i+N)\pi}{N})\right), 0, r\left(\sin(\frac{2(i+1+N)\pi}{N}) - \sin(\frac{2(i+N)\pi}{N})\right)\right) \\
&= \overrightarrow{P_{i+N}P_{i+1+N}}.
\end{aligned}$$

2.1.3 La sphère

Nous rappellerons tout d'abord qu'une sphère est définie par son centre et son rayon. Là encore nous supposerons dans un premier temps que le centre de la sphère est situé en $O = (0, 0, 0)$ et effectuerons éventuellement une translation si nous voulons la déplacer. Ainsi, pour un rayon $r > 0$ donné, la sphère est l'ensemble des points (x, y, z) tels que

$$x^2 + y^2 + z^2 = r^2.$$

Afin de repérer les points sur la sphère nous rappelons qu'un point (x, y, z) de la sphère peut être défini par un système de coordonnées cylindriques (θ, φ, r), avec $\theta \in [0, 2\pi]$, $\varphi \in [0, \pi]$ et $r > 0$. De plus, on utilisera la correspondance suivante

$$(x, y, z) = (r\sin(\varphi)\cos(\theta), r\cos(\varphi), r\sin(\varphi)\sin(\theta)) \tag{2.1}$$

Ce système de coordonnées est illustré sur la partie gauche de la Figure 2.3.

Nous avons de plus besoin de définir les méridiens et parallèles (aux intersections desquels se situeront les sommets de notre maillage) de la sphère.

- Nous appellerons **parallèle d'angle $\boldsymbol{\varphi_0}$** les point de la sphère pour lesquels la deuxième coordonnée polaire ($\boldsymbol{\varphi}$) est constante et égale à $\boldsymbol{\varphi_0}$ (c'est un cercle).

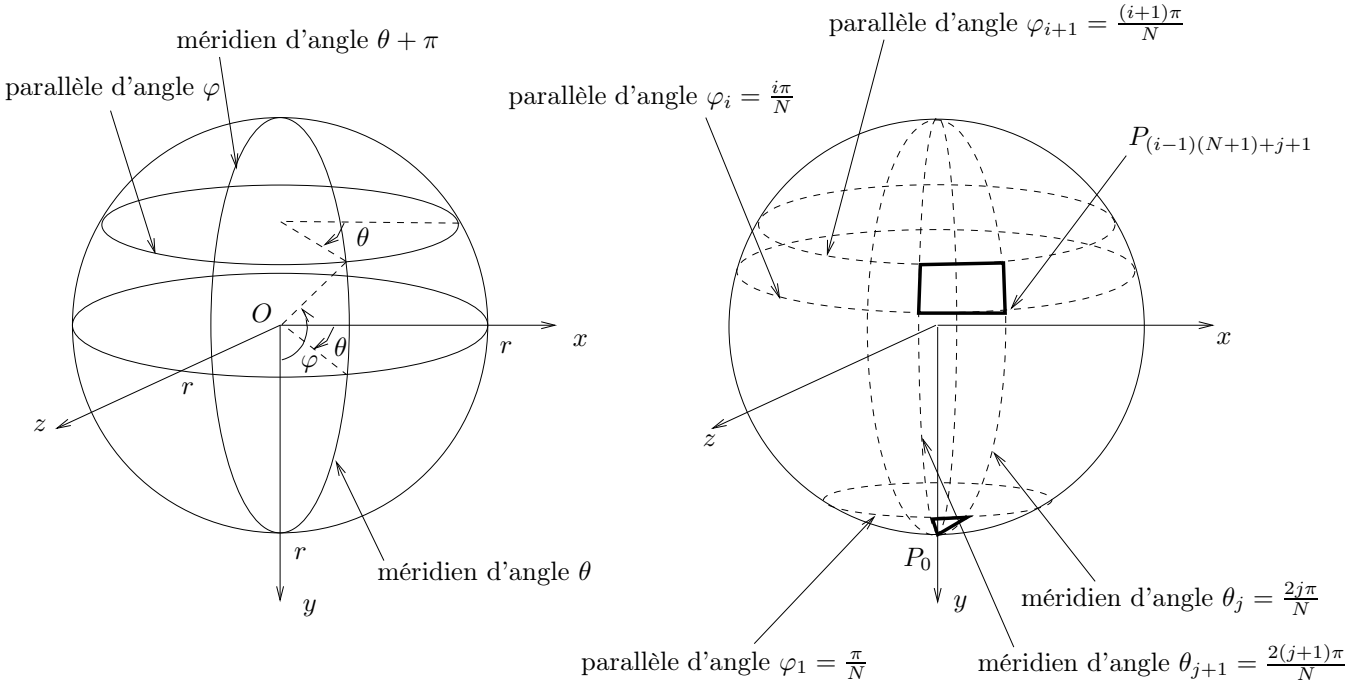

Figure 2.3: Gauche: Sphère de rayon r. Droite: maille triangulaire M_j et maille à quatre sommets M_{iN+j+N}.

- Nous appellerons **méridien d'angle $\boldsymbol{\theta_0}$** les points de la sphère pour lesquels l'angle de la première coordonnée polaire ($\boldsymbol{\theta}$) est constante et égale à $\boldsymbol{\theta_0}$ (c'est un demi-cercle).

Ainsi, nous utiliserons pour mailler la sphère avec une précision $N \in \mathbb{N}$:

- comme sommets:
  - le pôle sud: $P_0 = (0, r, 0)$ (il correspond à $\varphi = 0$).
  - les points intermédiaires: $P_{(i-1)(N+1)+j+1}$ est le point d'intersection de la parallèle d'angle $\varphi_i = \frac{i\pi}{N}$ et du méridien d'angle $\theta_j = \frac{2j\pi}{N}$ pour $i = 1, 2, ..., N-1$ et $j = 0, 1, ..., N$. Soit le point de coordonnées cartésiennes

$$P_{(i-1)(N+1)+j+1} = \left(r\sin(\frac{i\pi}{N})\cos(\frac{2j\pi}{N}), r\cos(\frac{i\pi}{N}), r\sin(\frac{i\pi}{N})\sin(\frac{2j\pi}{N}) \right)$$

    Notez ici que l'on a choisi de répéter le point $(\varphi_i, 0, r)$ de chaque parallèle afin de faciliter la création des mailles.
  - le pôle nord: $P_{N^2} = (0, -r, 0)$ (il correspond à $\varphi = \pi$).

- comme mailles:
  - Les mailles contenant le pôle sud (voir Figure 2.3): Elles sont triangulaires et lient le pôle sud aux deux points de coordonnées polaires (θ_j, φ_1, r) et $(\theta_{j+1}, \varphi_1, r)$, pour $j = 0, 1, ..., N-1$. Ainsi, on a

$$M_j = (0, j+1, j+2)$$

    pour $j = 0, ..., N-1$. On voit ici l'intérêt d'avoir pris $P_{N+1} = P_1$. Ceci nous évite d'avoir à faire un cas particulier de la maille $N-1$.
  - Les mailles intermédiaires: il s'agit de polygones à quatre sommets. Ces sommets ont respectivement les coordonnées polaires suivantes: (θ_j, φ_i, r), $(\theta_{j+1}, \varphi_i, r)$, $(\theta_{j+1}, \varphi_{i+1}, r)$ et $(\theta_j, \varphi_{i+1}, r)$

(voir Figure 2.3) pour $i = 1, ..., N-2$ et $j = 0, 1, ..., N-1$. Ceci nous conduit aux mailles suivantes:

$$M_{(i-1)N+j+N} = ((i-1)(N+1)+j+1,\ (i-1)(N+1)+j+2,\\ i(N+1)+j+2,\ i(N+1)+j+1)\,,$$

pour $i = 1, ..., N-2$ et $j = 0, 1, ..., N-1$.

– Les mailles contenant le pôle nord: Elles sont triangulaires et lient le pôle nord aux deux points de coordonnées polaires $(\theta_j, \varphi_{N-1}, r)$ et $(\theta_{j+1}, \varphi_{N-1}, r)$, pour $j = 0, 1, ..., N-1$. Ainsi, on a

$$M_{j+N^2-N} = (N^2, (N-2)(N+1)+j+1, (N-2)(N+1)+j+2)\,,$$

pour $j = 0, 1, ..., N-1$.

En ce qui concerne les mailles contenant l'un des pôles, elles n'ont que trois sommets et sont donc plates. Pour les mailles intermédiaires, ce sont des mailles à quatre sommets. Nous pouvons donc faire une vérification analogue à celle que nous avons faite pour les mailles du corps du cylindre. Afin de simplifier les notations, nous allons nous intéresser à une "maille type": une maille reliant les points de coordonnées polaires $P_1 = (\theta_1, \varphi_1, r)$, $P_2 = (\theta_2, \varphi_1, r)$, $P_3 = (\theta_2, \varphi_2, r)$ et $P_4 = (\theta_1, \varphi_2, r)$, pour $(\theta_1, \theta_2) \in [0, 2\pi]^2$ et $(\varphi_1, \varphi_2) \in [0, \pi]^2$. Un simple calcul nous donne alors (voir (2.1))

$$\begin{aligned}\overrightarrow{P_1P_2} &= r\Big(\sin(\varphi_1)\big(\cos(\theta_2)-\cos(\theta_1)\big), 0, \sin(\varphi_1)\big(\sin(\theta_2)-\sin(\theta_1)\big)\Big)\\ &= r\sin(\varphi_1)\Big(\cos(\theta_2)-\cos(\theta_1), 0, \sin(\theta_2)-\sin(\theta_1)\Big)\end{aligned}$$

et de même

$$\overrightarrow{P_4P_3} = r\sin(\varphi_2)\Big(\cos(\theta_2)-\cos(\theta_1), 0, \sin(\theta_2)-\sin(\theta_1)\Big)\,.$$

Ainsi, si $\sin(\varphi_2) \neq 0$, on a $\overrightarrow{P_1P_2} = \alpha\overrightarrow{P_4P_3}$ avec $\alpha = \frac{\sin(\varphi_1)}{\sin(\varphi_2)}$. De plus, si $\sin(\varphi_2) = 0$, P_3 et P_4 sont un seul et même point, on n'a donc que trois points qui sont alors coplanaires.

2.1.4 Translations et rotations d'un maillage

Afin de modifier la position et l'orientation d'un objet représenté, nous pouvons simplement lui appliquer des transformations simples: une translation et une rotation.

La translation

Afin de translater un maillage, il suffit de translater les sommets du maillage (les mailles restent identiques). Ainsi, le maillage translaté, d'un vecteur $(t_x, t_y, t_z) \in \mathbb{R}^3$, d'un maillage constitué des sommets $P_0 = (x_0, y_0, z_0)$, $P_1 = (x_1, y_1, z_1)$, ..., $P_{n-1} = (x_{n-1}, y_{n-1}, z_{n-1})$, et des mailles M_0, M_1, ..., M_{m-1}, est donné par:

- les sommets: $P'_0 = (x_0 + t_x, y_0 + t_y, z_0 + t_z)$, $P'_1 = (x_1 + t_x, y_1 + t_y, z_1 + t_z)$, ..., $P'_{n-1} = (x_{n-1} + t_x, y_{n-1} + t_y, z_{n-1} + t_z)$.
- les mailles: M_0, M_1, ..., M_{m-1}.

Il faut vérifier que les mailles ainsi constituées sont bien formées de points coplanaires. Ceci est évident dans le cas de la translation puisque pour $(i, j) \in \{0, 1, ..., n-1\}^2$

$$\overrightarrow{P'_iP'_j} = \overrightarrow{P_iP_j}.$$

Ainsi, si pour une maille $M_i = (k^i_0, k^i_1, ..., k^i_{m_i})$, pour $i \in \{0, 1, ..., m-1\}$, tous les vecteurs $\overrightarrow{P_{k^i_j}P_{k^i_l}}$, pour $(j, l) \in \{0, 1, ..., m_i\}^2$, appartiennent à un même plan, il en va de même pour les vecteurs $\overrightarrow{P'_{k^i_j}P'_{k^i_l}}$.

La rotation

Pour la rotation d'un maillage, là encore, il suffit d'appliquer la rotation aux sommets du maillage et de conserver les mailles de l'ancien maillage. La rotation que nous appliquerons aux sommets est définie par un axe (une droite $\mathcal{D}$) et un angle θ (voir Figure 2.4). Nous la noterons $R_{\mathcal{D},\theta}$.

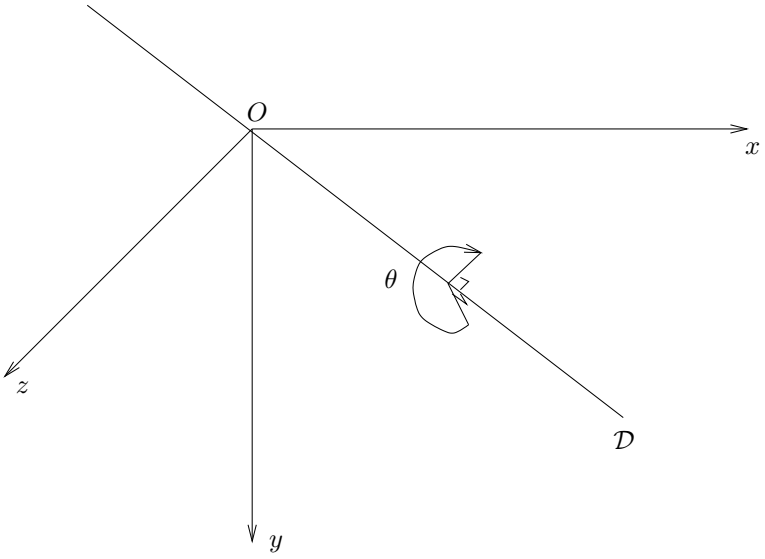

Figure 2.4: Rotation d'angle θ autour de l'axe $\mathcal{D}$.

Pour simplifier, nous supposons ici que l'orientation de notre maillage est déterminée avant sa position (que l'on ajustera avec une translation). Nous pouvons donc nous contenter des rotations autour d'axes passant par l'origine $O = (0, 0, 0)$.

Pour simplifier la saisie des différents paramètres, nous n'implémenterons que les rotations obtenues en composant des rotations autour des axes (Ox), (Oy), (Oz). Plus précisément, ces rotations sont définies par trois angles θ, φ, η dans $\mathbb{R}$ et sont de la forme

$$R_{(Oz),\eta} \circ R_{(Oy),\varphi} \circ R_{(Ox),\theta}\,.$$

Notez que l'on peut à partir de ces trois simples rotations obtenir une rotation quelconque. En effet, soit une rotation $R_{\mathcal{D},\theta}$, d'axe $\mathcal{D}$ (passant par l'origine et un point de coordonnées polaires (η, φ, r)), la rotation $R_{(Oy),-\eta}$ permet de ramener le point (η, φ, r) dans le plan (O, x, y), puis $R_{(Oz),\frac{\pi}{2}-\varphi}$ permet de ramener le point ainsi obtenu sur la droite (Ox). Il ne reste plus qu'à appliquer $R_{(Ox),\theta}$ et à revenir sur le bon axe. Ainsi, on a

$$R_{\mathcal{D},\theta} = R_{(Oy),\eta} \circ R_{(Oz),\varphi-\frac{\pi}{2}} \circ R_{(Ox),\theta} \circ R_{(Oz),\frac{\pi}{2}-\varphi} \circ R_{(Oy),-\eta}$$

Considérons donc une rotation $R_{(Ox),\theta}$ autour de l'axe (Ox) et d'angle θ. Étant donnée la définition de l'angle θ donnée sur la Figure 2.5, il est facile de voir[1] que les coordonnées de $R_{(Ox),\theta}(X)$ pour $X = (x, y, z)$ sont

$$R_{(Ox),\theta}(X) = \big(x, z\sin(\theta) + y\cos(\theta), z\cos(\theta) - y\sin(\theta)\big)\,.$$

De même, on a pour une rotation autour de l'axe (Oy) et d'angle φ

$$R_{(Oy),\varphi}(X) = \big(x\cos(\varphi) - z\sin(\varphi), y, x\sin(\varphi) + z\cos(\varphi)\big)\,.$$

[1] Il est clair que la première coordonnée x reste inchangée et que l'on applique une rotation d'angle θ au point 2D (z, y) (ici on inverse z et y pour avoir l'orientation correcte du plan 2D). Celui-ci est de coordonnées (on utilise la correspondance entre points du plan et nombres complexes)

$$(z + iy)e^{i\theta} = z\cos(\theta) - y\sin(\theta) + i\big(y\cos(\theta) + z\sin(\theta)\big).$$

Ceci nous conduit au résultat énoncé.

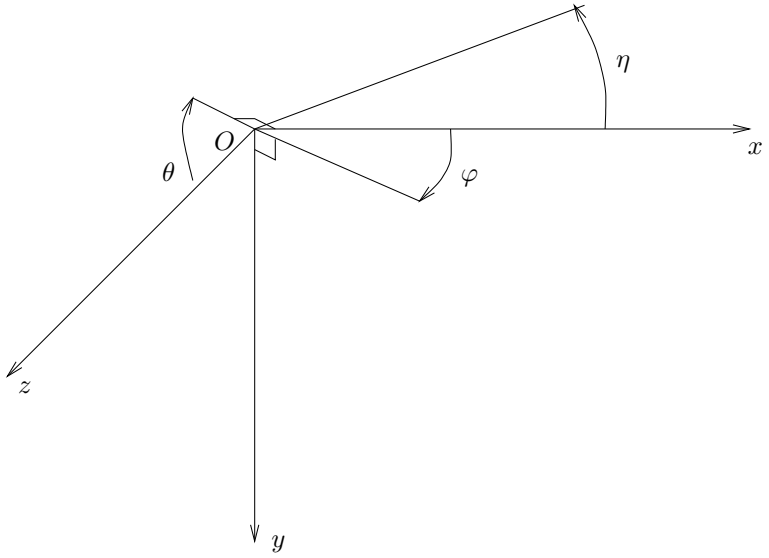

Figure 2.5: Rotation d'angle $\theta < 0$ autour de l'axe (Ox), d'angle $\varphi > 0$ autour de l'axe (Oy) et d'angle $\eta > 0$ autour de l'axe (Oz).

De même, on a pour une rotation autour de l'axe (Oz) et d'angle η

$$R_{(Oz),\eta}(X) = \big(x\cos(\eta) + y\sin(\eta), -x\sin(\eta) + y\cos(\eta), z\big)\,.$$

Le maillage tourné d'un maillage constitué des sommets P_0, P_1, ..., P_{n-1}, et des mailles M_0, M_1, ..., M_{m-1} est donné par:

- les sommets: $P'_i = R_{(Oz),\eta} \circ R_{(Oy),\varphi} \circ R_{(Ox),\theta}(P_i)$, pour $i \in \{0, 1, ..., n-1\}$.
- les mailles: M_0, M_1, ..., M_{m-1}.

Là encore, il faudrait vérifier que la rotation d'un maillage est bien un maillage (c'est à dire: que les mailles restent plates). Ceci est laissé en exercice.

Transformation générale d'un maillage

Ici, nous ne nous sommes intéressés qu'à des transformations simples (plus précisément *affines*) d'un maillage. Pour de telles transformations, des points coplanaires restent coplanaires après la transformation. Il n'en va évidement pas de même pour des transformations plus générales (par exemple: dans le cas de films, si on veut représenter une boule qui s'enfonce dans un oreiller, ou la surface d'un océan agité par la houle). Dans de tels cas, on privilégiera un maillage constitué de mailles triangulaires, car trois points sont toujours coplanaires.

2.1.5 Commentaires

On réalise sur ces exemples simples qu'il faut être très attentif lors de la définition des points et des mailles (notamment lors de l'implémentation des algorithmes). Ceci dit, une fois le maillage créé, il sera extrêmement simple d'appliquer un traitement à notre surface en parcourant les mailles une à une. Cette représentation a, de plus, l'avantage d'être relativement compacte, puisque les sommets ne sont stockés qu'une seule fois en mémoire (nous verrons par la suite que l'on a parfois besoin de les stocker deux fois pour obtenir un bon rendu de la lumière), et les mailles ne sont qu'une suite (souvent courte) d'entiers.

En fait, le défaut majeure de cette représentation est qu'il est extrêmement difficile de modifier la connexité des objets que l'on visualise. Ceci se produit si l'on veut animer un objet (film) et le séparer en plusieurs morceaux (par exemple montrer une assiette qui se casse) ou réunir plusieurs objets pour

n'en former qu'un seul (exemple: des gouttes d'eau qui tombent et forment une flaque). Ceci est la raison majeur du développement actuel des représentations implicites des surfaces en infographie. En effet, pour ce mode de représentation, modifier la connexité d'un objet est naturel. Le défaut de cette représentation étant son manque de compacité.

Exercice 7 *Montrer que la rotation autour d'un axe $\mathcal{D}$ d'angle θ d'un maillage est bien un maillage.*

Exercice 8 *Implémenter une méthode JAVA pour une classe maillage permettant d'effectuer la rotation d'un maillage obtenue en composant $R_{(Oz),\eta} \circ R_{(Oy),\varphi} \circ R_{(Ox),\theta}$. On supposera que la classe Maillage possède une variable d'instance de classe pointsDuMaillage contenant les points du maillage sous la forme d'un objet Vector dont les éléments sont des objets Point3D (la classe Point3D est supposée déjà créée et possède des méthodes analogues à celle de la classe Point).*

2.2 Projections du maillage sur le plan

Une fois que nous avons créé un maillage, pour l'afficher, il faut projeter les mailles du maillage sur le plan. Ceci nous permettra, en projetant chaque maille (pour cela, il suffit de projeter ses sommets) d'obtenir des polygones dans le plan que nous pourrons afficher. Dans un deuxième temps, nous pourrons, grâce à cette projection, déterminer quels sont les points du plan à colorier dans l'algorithme du z-buffer (voir Section 2.3).

Tout d'abord, nous devons définir le plan de projection. Dans les deux projections que nous verrons par la suite, le plan de projection sera la plan d'équation

$$z = z_0 \,.$$

En pratique, on prendra $z_0 = 600$. Notez que la valeur 600 est arbitraire mais influera sur la taille des objets que nous pourrons correctement visualiser avec la deuxième projection que nous verrons.

2.2.1 Une projection simple

Il s'agit de la projection orthogonale au plan d'équation $z = z_0$. Ainsi la projection d'un point (x, y, z) de $\mathbb{R}^3$ est simplement le point de coordonnées $(x_p, y_p) \in \mathbb{R}^2$ avec

$$x_p = x \text{ et } y_p = y \,.$$

la projection décrite dans la section suivante.

2.2.2 Une projection pour un meilleur rendu

Cette projection correspond au modèle de la caméra pinhole ("trou d'une épingle" en anglais) qui place l'oeil de l'observateur en un point O_b tel que le plan de projection se situe entre O_b et l'objet observé. Ainsi, la projection sur le plan est la "trace", sur le plan de projection, d'un rayon (de lumière dans le cas d'une caméra) allant de l'objet à l'oeil (voir Figure 2.6).

Dans notre cas, nous prendrons simplement $Ob = (x_0, y_0, z_0 + d)$, où (x_0, y_0) est au centre de la fenêtre que l'on affiche. Une valeur de $d = 400$ permet d'avoir un rendu correct pour visualiser des objets d'une taille de l'ordre de "quelques centaines" situés près de l'origine.

Nous avons maintenant fixé tous les paramètres nécessaires pour pouvoir déterminer les coordonnées (x_p, y_p, z_p) d'un point projeté. Ainsi, nous savons que O_b, X_p (le point projeté) et X (le point à projeter) sont alignés. Nous avons donc

$$\overrightarrow{O_bX_p} = \lambda \overrightarrow{O_bX} \,,$$

pour un $\lambda \in \mathbb{R}$ (en pratique on a même $0 < \lambda \leq 1$).

De plus, le théorème de Thalès nous garantit que

$$\frac{|O_bX_p|}{|O_bX|} = \frac{|O_bC'|}{|O_bX'|}$$

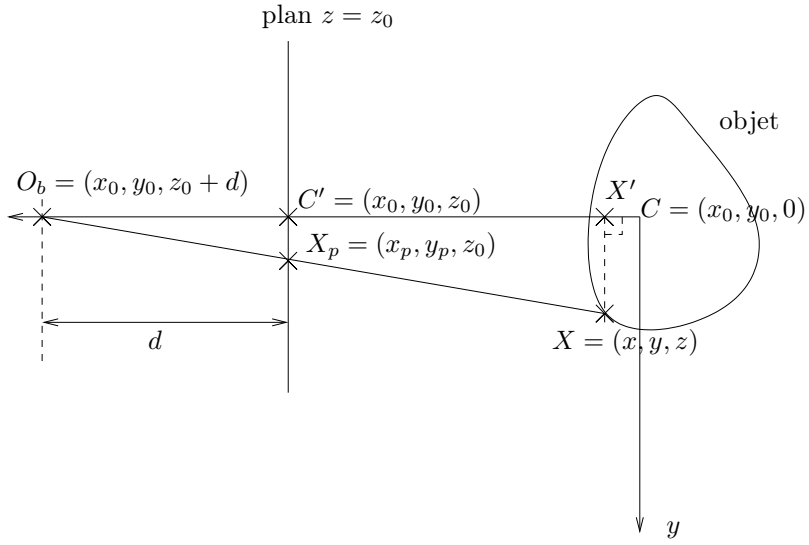

Figure 2.6: Projection sur le modèle de la caméra "pinhole".

avec $X' = (x_0, y_0, z)$ (en effet $(XX') \parallel (X_pC')$). Comme nous savons par ailleurs que la troisième coordonnée de X_p est égale à z_0, on a

$$\lambda = \frac{|O_bX_p|}{|O_bX|} = \frac{|O_bC'|}{|O_bX'|} = \frac{d}{z_0 + d - z}\,.$$

On a donc enfin $X_p = O_b + \frac{d}{z_0+d-z}(X - O_b)$ et donc

$$\left\{ \begin{array}{rcl} x_p & = & \frac{d}{z_0+d-z}(x - x_0) + x_0 \\ y_p & = & \frac{d}{z_0+d-z}(y - y_0) + y_0 \\ z_p & = & z_0 + d + \frac{d}{z_0+d-z}(z - z_0 - d) = z_0 \end{array} \right.$$

Notez que le fait que $z_p = z_o$ était attendu et nous confirme que nos calculs doivent être corrects. Notez de plus que du fait du dénominateur, $z_0 + d - z$, nous ne pouvons pas tracer un point situé sur le plan $z = z_0 + d$ (ceci se voit aussi géométriquement sur la Figure 2.6 car dans un tel cas la droite (O_bX) n'intersecte pas le plan $z = z_0$). Enfin, les objets situés dans le dos de l'observateur seront représentés à l'envers ($z_0 + d - z$ est négatif). Ces comportements sont bien sûr des cas limites et, en pratique, nous ne nous intéressons qu'au cas $z < z_0$.

Exercice 9 *Écrire une méthode JAVA retournant le polygone 2D (donné sous la forme d'un Vector) résultant de la projection "pinhole" d'un polygone de l'espace 3D (donné sous la forme d'un Vector).*

2.3 Élimination des parties cachées: Algorithme du z-buffer

2.3.1 Pré-requis pour l'algorithme du z-buffer

Nous aurons besoin pour éliminer les mailles cachées d'effectuer l'opération inverse de la projection décrite à la section précédente. Plus précisément, nous voulons, pour un point $X = (X_x, X_y, z_0)$ dont on sait qu'il est le projeté d'un point X^i d'une maille M_i, trouver les coordonnées précises de X^i.

Pour cela, nous allons d'abord déterminer l'équation du plan de la maille M_i puis déterminer les coordonnées du point d'intersection de ce plan avec la droite de projection (la droite $x = X_x$ et $y = X_y$ dans le cas de la projection simple; et la droite (O_bX) dans le cas de la projection "pinhole").

Déterminer l'équation du plan contenant une maille

Soit un polygone de l'espace 3D (en pratique une maille M_i) défini par ses sommets $(P_0, P_1, ..., P_n)$. On veut déterminer les coefficients a, b, c, d de l'équation de plan $ax+by+cz+d = 0$ tels que ce plan contienne le polygone.

Pour cela, nous allons utiliser deux propriétés mathématiques: la première est que pour un plan défini par une équation $ax + by + cz + d = 0$, le vecteur (a, b, c) est orthogonal au plan; la deuxième est que, pour deux vecteurs $\overrightarrow{V} = (V_x, V_y, V_z)$ et $\overrightarrow{W} = (W_x, W_y, W_z)$ dans le plan (par exemple: $\overrightarrow{P_0P_1}$ et $\overrightarrow{P_0P_2}$), leur produit vectoriel

$$\overrightarrow{V} \wedge \overrightarrow{W} = \begin{pmatrix} V_yW_z - V_zW_y \\ V_zW_x - V_xW_z \\ V_xW_y - V_yW_x \end{pmatrix}$$

est orthogonal au plan défini par $\overrightarrow{V}$ et $\overrightarrow{W}$ (sous réserve que $\overrightarrow{V}$ et $\overrightarrow{0}W$ ne soient pas parallèles).

Ainsi, nous pouvons, pour déterminer a, b, c, simplement calculer

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \overrightarrow{P_0P_1} \wedge \overrightarrow{P_0P_2}$$

Si $\overrightarrow{P_0P_1} \wedge \overrightarrow{P_0P_2} = 0$, alors $\overrightarrow{P_0P_1}$ et $\overrightarrow{P_0P_2}$ sont parallèles et on essaiera successivement le calcul de $\overrightarrow{P_0P_1} \wedge \overrightarrow{P_0P_3}$, $\overrightarrow{P_0P_1} \wedge \overrightarrow{P_0P_4}$, ..., $\overrightarrow{P_0P_1} \wedge \overrightarrow{P_0P_n}$.

Ici , on aurait pu être plus précis et chercher des points tels que $|\overrightarrow{P_iP_j} \wedge \overrightarrow{P_kP_l}|$ soit la plus grande possible. Cependant, dans le cas des maillages que nous avons décrit en Section 2.1, la procédure que nous venons de décrire est, en pratique, suffisante.

Il nous reste encore à déterminer d. Pour cela , nous allons utiliser le fait que, par exemple, $P_0 = ((P_0)_x, (P_0)_y, (P_0)_z)$ est dans le plan. Ainsi, on a simplement

$$d = -(a(P_0)_x + b(P_0)_y + c(P_0)_z)\,.$$

Déterminer l'intersection d'une droite et d'un plan

Nous cherchons maintenant à déterminer les coordonnées du point d'intersection d'un plan d'équation $ax + by + cz + d = 0$ et d'une droite définie par deux points $A = (A_x, A_y, A_z)$ et $B = (B_x, B_y, B_z)$. Pour l'algorithme du z-buffer, on prendra dans le cas de la projection simple: $A = X = (X_x, X_y, z_0)$ et $B = (X_x, X_y, z_0 + 1)$ (voir ci-dessus); la projection "pinhole", $A = X$ et $B = O_b = (x_0, y_0, z_0 + d)$.

Soit $M_t = tA + (1 - t)B$, pour $t \in \mathbb{R}$, un point de la droite. Ce point est aussi dans le plan si et seulement si

$$a(tA_x + (1 - t)B_x) + b(tA_y + (1 - t)B_y) + c(tA_z + (1 - t)B_z) + d = 0$$

On a donc pour le point d'intersection

$$t = -\frac{d + aB_x + bB_y + cB_z}{a(A_x - B_x) + b(A_y - B_y) + c(A_z - B_z)}$$

On peut ensuite déduire les coordonnées du point d'intersection en remplaçant cette valeur de t dans la définition de M_t.

Il y a évidemment un problème lorsque $a(A_x - B_x) + b(A_y - B_y) + c(A_z - B_z) = 0$. Ceci correspond au cas où $\overrightarrow{AB}$ est parallèle au plan $ax + by + cz + d = 0$. Dans ce cas, pour l'algorithme du z-buffer, nous choisirons simplement de faire comme si la maille M_i ne se projetait pas sur X (voir notations plus haut).

2.3.2 L'algorithme du z-buffer

Le problème que nous cherchons maintenant à résoudre est celui de l'élimination des parties cachées. En effet, lors de l'affichage d'un maillage, dans le cas d'un objet opaque, un certain nombre de mailles sont cachées du plan de projection et ne doivent donc pas être affichées.

Notons p la projection sur le plan d'équation $z = z_0$. Si l'on note $X = (x, y, z_0)$ un point du plan de projection, soit X n'est pas le projeté d'un point d'une des mailles (auquel cas on l'affiche avec la couleur du fond, par exemple le noir); soit il existe i_0, i_1, ..., i_k tels que $X = p(X^{i_j})$ où $X^{i_j} = (x^{i_j}, y^{i_j}, z^{i_j})$ est un point de la maille M_{i_j} et $j \in \{0, 1, ..., k\}$. La maille vue depuis X est alors la maille $M_{i_{jmin}}$ telle que

$$|X - X^{i_j}|$$

soit minimale. Les autres points X^{i_j} étant cachés par $X^{i_{jmin}}$.

Ici, afin d'éviter le calcul des $|X - X^{i_j}|$, nous remarquons que l'on peut simplement chercher l'indice permettant de minimiser

$$|z^{i_j} - z_0|\,.$$

Ceci se déduit simplement du théorème de Thalès (voir Figure 2.7).

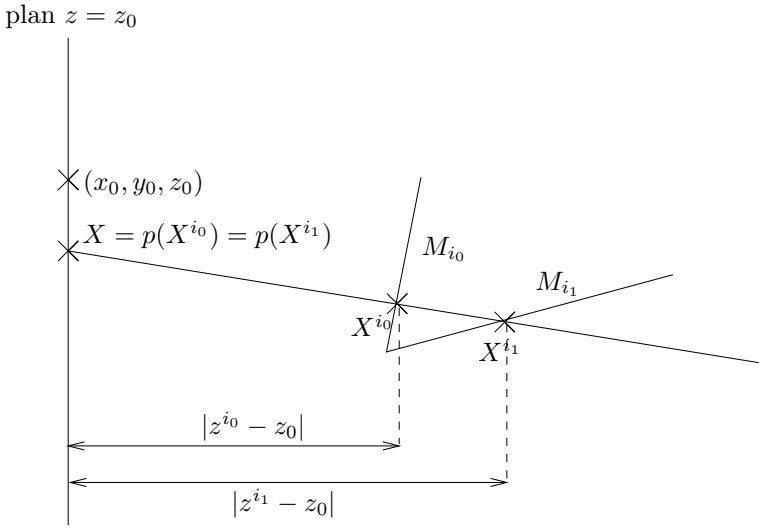

Figure 2.7: La maille M_{i_0} est visible et elle cache la maille M_{i_1} (représentation 2D).

Nous appellerons $|z^{i_j} - z_0|$ l'altitude du point X^{i_j}.

Afin de n'afficher que les mailles visibles, nous sommes donc amenés à construire un tableau qui à chaque pixel du plan de projection fait correspondre l'altitude de la maille la plus proche (d'où le nom de z-buffer). Nous mémoriserons de plus l'indice de la maille la plus proche pour effectuer l'affichage de cette dernière (par exemple dans un tableau nommé *mailleVisible*).

Ceci nous conduit à l'algorithme suivant:

- On initialise un tableau *altitude*, de la taille de notre plan de projection, à une valeur très grande (ex: 999999).
- On initialise un tableau *mailleVisible* , de la taille de notre plan de projection, à -1 (*mailleVisible[x][y]==-1* signifiera par la suite qu'aucune maille ne se projette en (x, y) et que ce point a donc la couleur du fond, exemple noir).
- On parcourt les mailles de notre maillage et pour chaque maille M_k:
  - On détermine l'équation du plan de la maille M_k (voir Section 2.3.1)
  - On projette la maille M_k sur le plan de projection (voir Section 2.2). Ceci nous donne un polygone dans le plan: *polygone2D*
  - On parcourt l'intérieur du polygone *polygone2D* (pour cela on adaptera l'algorithme de remplissage de polygone vu à la section 1.3) ainsi créé et pour chaque point (x, y) de l'intérieur de *polygone2D*, on:

- détermine l'altitude *alt* du point qui se projette en (x, y) (pour cela on calcule l'intersection de la droite de projection avec le plan contenant la maille, voir Section 2.3.1).
- Si *alt* < *altitude[x][y]* alors
  - *altitude[x][y]=alt*
  - mailleVisible[x][y]=k

Cet algorithme est un peu fastidieux à écrire et nécessite d'avoir implémenté correctement la plupart des algorithmes que nous avons vus jusque là. Il est cependant un passage obligé pour aborder le "coloriage" de nos formes 3D (éclairement, texture, couleur, ...).

2.4 L'éclairement d'un objet

Une fois implémenté l'algorithme du z-buffer, nous savons pour chaque point du plan de projection quelle est la maille vue depuis ce point. Ceci va maintenant nous permettre d'attribuer une couleur à ce point. Pour ceci, nous utiliserons les informations liées à "l'orientation" de cette maille. Cette "orientation" est en effet une information fondamentale pour déterminer l'éclairement de l'objet.

Représentation des couleurs sous JAVA

JAVA utilise le système de représentation des couleurs dit *RVB* (pour Rouge, Vert, Bleu). Chacune de ces composantes peut être repérée de deux façons:

- par un entier entre 0 et 255. Ainsi, une couleur est la donnée d'un point $(col_r, col_v, col_b) \in \{0, ..., 255\}^3$,
- par un flottant entre 0 et 1. Ainsi, une couleur est la donnée d'un point $(col_r, col_v, col_b) \in [0, 1]^3$,

Si l'on ne s'intéresse qu'à des images en niveaux de gris (cas d'objets blancs éclairés par une lumière blanche), on utilisera une couleur du type $col\,(1, 1, 1)$, où $col \in \{0, ..., 255\}$ ou $col \in [0, 1]$.

Il existe une classe *Color* permettant de créer une couleur.

Pour afficher des points sur un objet *dessin* de la classe *Graphics2D* avec la couleur $(0, 130, 247)$, il faut

- créer la couleur:

  Color col= new Color(0,130,247);

- dire que l'on veut l'utiliser sur *dessin*:

  dessin.setColor(col);

Tous les points alors affichés sur *dessin* le seront avec la couleur $(0, 130, 247)$. Pour changer de couleur, il suffit de répliquer ces deux lignes en modifiant le contenu de la méthode constructeur *Color()*.

2.4.1 Modèles d'éclairement

Nous distinguerons ici trois types de luminosités:

- la lumière ambiante
- la lumière ponctuelle sur un objet à réflexion diffuse
- la lumière ponctuelle sur un objet à réflexion spéculaire (surface brillante).

Pour rendre compte d'objets et de conditions d'éclairement différentes, on peut faire la somme de ces trois types de luminosités (éventuellement avec plusieurs sources lumineuses).

Lumière ambiante

La lumière ambiante est une lumière dont l'intensité est la même dans toutes les directions et qui est donc reflétée de la même façon par tous les objets de même nature (ex: peinture mate ou brillante). Nous avons donc un indice de réflection k qui dépendra de la nature de l'objet que l'on veut représenter. Ceci nous conduit donc au modèle lumineux suivant

$$I = kI_a$$

où I_a est l'intensité de la lumière ambiante (à trois coordonnées entre 0 et 255 ou entre 0 et 1) et I est la couleur (à trois coordonnées) au point observé. Encore une fois, k peut dépendre du maillage considéré (si vous avez plusieurs objets) ou même de la maille vue depuis le point considéré (si un objet est fait de plusieurs matières).

Lumière ponctuelle sur un objet à réflexion diffuse

Dans les deux sections suivantes, nous allons considérer une source lumineuse ponctuelle. Cette source est caractérisée par un point $P_p \in \mathbb{R}^3$ et une intensité I_p, à trois coordonnées entre 0 et 255 ou entre 0 et 1.

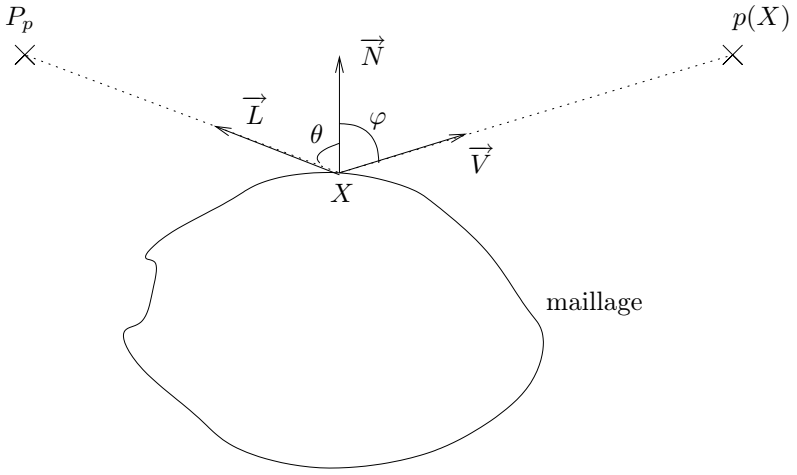

Figure 2.8: Éclairement par une source ponctuelle.

Dans le cas représenté sur la Figure 2.8, la réflexion diffuse correspond simplement à l'équation suivante:

$$I = \begin{cases} I_p k_{rd} \cos(\theta), & \text{si } \cos(\theta) > 0, \\ 0, & \text{si } \cos(\theta) \leq 0 \end{cases}$$

où I est la lumière observée, $k_{rd} \in [0, 1]$ est le coefficient de réflexion diffuse de l'objet (elle peut varier d'un maillage à l'autre) et θ est l'angle entre $\overrightarrow{N}$ et $\overrightarrow{L}$. On voit sur ce modèle qu'un point éclairé de façon zénithale est clair alors qu'un point éclairé avec une lumière rasante ($\theta \sim \frac{\pi}{2}$) est peu lumineux. Enfin, si la source lumineuse est derrière l'objet (cas où $\cos(\theta) \leq 0$), la lumière est nulle car l'objet n'est plus éclairé. Par contre, cette lumière ne dépend pas de la position de l'observateur. Autrement dit, il n'y a pas de phénomène de reflet.

Pour un point du plan de projection $p(X)$, il va nous falloir "retrouver" tous les éléments dessinés sur la Figure 2.8 afin de calculer θ. Nous le ferons comme suit:

- On détermine le point X. Pour cela on utilise l'équation de la maille et la direction de projection (voir la Section 2.3.1).
- On calcule $\overrightarrow{V} = (V_x, V_y, V_z) = \frac{\overrightarrow{Xp(X)}}{|Xp(X)|}$.
- On détermine un vecteur orthogonal au maillage en X, de norme 1, orienté vers $p(X)$. Pour cela, on peut utiliser l'équation du plan de la maille $ax + by + cz + d = 0$. Le vecteur $\overrightarrow{N} = (N_x, N_y, N_z) =$

$\frac{(a,b,c)}{\sqrt{a^2+b^2+c^2}}$ est orthogonal au maillage en X et de norme 1. Pour déterminer son orientation, on regarde simplement le signe de

$$\begin{aligned}\cos(\varphi) &= \overrightarrow{N}.\overrightarrow{V} \\ &= N_xV_x + N_yV_y + N_zV_z.\end{aligned}$$

Si $\cos(\varphi) < 0$, on remplace $\overrightarrow{N}$ par $-\overrightarrow{N}$.

- On calcule $\overrightarrow{L} = \frac{\overrightarrow{XP_p}}{|XP_p|}$.
- On calcule $\cos(\theta) = \overrightarrow{L}.\overrightarrow{N}$.

Lumière ponctuelle sur un objet à réflexion spéculaire

La réflexion spéculaire tient compte du reflet de la source lumineuse. Ainsi, la source lumineuse se reflétera surtout dans la direction symétrique, par rapport à $\overrightarrow{N}$, à la direction d'incidence de la lumière (celle de $\overrightarrow{L}$). Ceci nous conduit à définir le vecteur $\overrightarrow{R}$ de cette réflexion (voir Figure 2.9). La lumière spéculaire est alors définie par

$$I = \begin{cases} I_p k_{rs} \cos(\alpha)^n, & \text{si } \cos(\theta) > 0 \text{ et } \cos(\alpha) > 0 \\ 0, & \text{si } \cos(\theta) \leq 0 \text{ ou } \cos(\alpha) \leq 0 \end{cases}$$

pour $k_{rs} \in [0,1]$ un coefficient de réflexion spéculaire, $n \in \mathbb{N}$ et α est l'angle entre $\overrightarrow{R}$ et $\overrightarrow{V}$. Notez que plus n est grand, moins la tache lumineuse autour du reflet est grande. Des valeurs de $n = 2$, 3 ou 4 sont souvent considérées comme convenables.

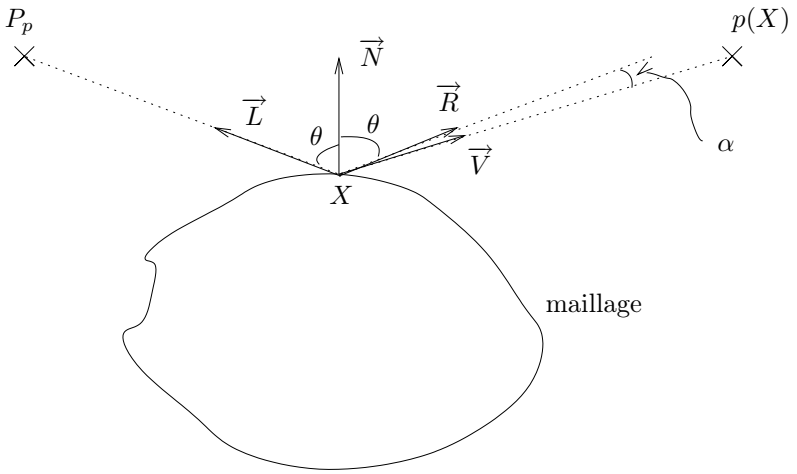

Figure 2.9: Éclairement par une source ponctuelle avec une réflexion spéculaire.

Le calcul de $\cos(\alpha)$ se fait simplement en remarquant que

$$\overrightarrow{R} + \overrightarrow{L} = 2\cos(\theta)\overrightarrow{N}.$$

On a donc

$$\begin{aligned}\cos(\alpha) = \overrightarrow{R}.\overrightarrow{V} &= \left(2\cos(\theta)\overrightarrow{N} - \overrightarrow{L}\right).\overrightarrow{V} \\ &= \left(2\left(\overrightarrow{L}.\overrightarrow{N}\right)\overrightarrow{N} - \overrightarrow{L}\right).\overrightarrow{V}\end{aligned}$$

Le calcul de $\overrightarrow{V}$, $\overrightarrow{L}$ et $\overrightarrow{N}$ se faisant comme dans le cas d'une réflexion diffuse.

Comme nous l'avons dit dans l'introduction de cette section, un éclairement réaliste est obtenu comme la somme des trois types d'éclairement que nous avons vue. Ainsi, on obtient le modèle suivant

$$I = kI_a + I_p 1_{\cos(\theta)>0} \left(k_{rd}\cos(\theta) + k_{rs}\cos(\alpha)^n 1_{\cos(\alpha)>0}\right),$$

où la fonction $1_{\cos(\theta)>0}$ est égale à 1 si $\cos(\theta) > 0$ et 0, sinon. Les paramètres sont la position de la source P_p, les intensités lumineuses I_a et I_p et les coefficients de réflexions k, k_{rd} et k_{rs}.

Une fois calculée cette intensité, on la tronque, pour la ramener entre 0 et 255, puis on l'arrondit à l'entier le plus proche; ou on la tronque pour se ramener entre 0 et 1.

2.4.2 Approximation de la normale à la surface

Principe général

Un point faible de la méthode telle que nous l'avons décrite réside dans le calcul du vecteur $\overrightarrow{N}$. En effet, si l'on prend réellement le vecteur orthogonal à la maille, on obtient un vecteur qui change brutalement en passant d'une maille à l'autre. Pour éviter ce phénomène, nous allons chercher en chaque point du maillage à calculer une approximation correcte du vrai vecteur normal à la surface initiale (avant l'approximation de la surface par le maillage). Ici, l'objectif est d'avoir une approximation des vrais vecteurs normaux qui puissent être **continue** afin d'éviter les sauts d'intensités lumineuses.

Ainsi les deux sections suivantes expliquent comment approximer les normales de façon continue. Pour ce faire, nous supposerons qu'en chaque sommet P_i de notre maillage on connaît une normale $N_i = ((N_i)_x, (N_i)_y, (N_i)_z)$. Nous allons interpoler la valeur de la normale en un point $X = (x, y, z)$ à l'intérieur d'une maille à partir des valeurs de la normale aux sommets de la maille. (Interpoler veut dire déterminer des valeurs de la normale en des points où l'on ne la connaît pas à partir de la valeur de la normale en des points où on la connaît.) Pour ce faire, nous allons interpoler chaque composante de la normale ($(N_i)_x$, $(N_i)_y$ et $(N_i)_z$) indépendamment, de manière à ce que ces composantes soient continues.

Déterminer les normales à la surface aux sommets du maillage

Pour chacun des maillages vus jusque là, nous allons redéfinir le maillage en incluant la normale. Nous mémoriserons cette normale dans une variable d'instance de la classe *Maillage*. Dans le cas du cube et du cylindre, le maillage demande à être complètement redéfini. En effet, dans ces deux cas, la normale est discontinue aux arrêtes et il nous faudra donc définir plusieurs fois un même sommet car ce sommet a plusieurs normales (cas du cube: une pour chaque face ayant le sommet; cas du cylindre: une pour le corps du cylindre et une pour les faces du haut ou du bas).

De plus, la méthode pour interpoler les normales que nous allons ensuite introduire n'est adaptée qu'à des mailles triangulaires. Nous modifions donc aussi les mailles pour qu'elles le soient.

Le cube

- Les sommets: $P_0 = (0,0,0)$, $P_1 = (300,0,0)$, $P_2 = (300,0,300)$, $P_3 = (0,0,300)$, $P_4 = (0,0,0)$, $P_5 = (300,0,0)$, $P_6 = (300,300,0)$, $P_7 = (0,300,0)$, $P_8 = (300,0,0)$, $P_9 = (300,0,300)$, $P_{10} = (300,300,300)$, $P_{11} = (300,300,0)$, $P_{12} = (300,0,300)$, $P_{13} = (0,0,300)$, $P_{14} = (0,300,300)$, $P_{15} = (300,300,300)$, $P_{16} = (0,0,300)$, $P_{17} = (0,0,0)$, $P_{18} = (0,300,0)$, $P_{19} = (0,300,300)$, $P_{20} = (0,300,0)$, $P_{21} = (300,300,0)$, $P_{22} = (300,300,300)$, $P_{23} = (0,300,300)$.

- Les normales: $N_0 = (0,-1,0)$, $N_1 = (0,-1,0)$, $N_2 = (0,-1,0)$, $N_3 = (0,-1,0)$, $N_4 = (0,0,-1)$, $N_5 = (0,0,-1)$, $N_6 = (0,0,-1)$, $N_7 = (0,0,-1)$, $N_8 = (1,0,0)$, $N_9 = (1,0,0)$, $N_{10} = (1,0,0)$, $N_{11} = (1,0,0)$, $N_{12} = (0,0,1)$, $N_{13} = (0,0,1)$, $N_{14} = (0,0,1)$, $N_{15} = (0,0,1)$, $N_{16} = (-1,0,0)$, $N_{17} = (-1,0,0)$, $N_{18} = (-1,0,0)$, $N_{19} = (-1,0,0)$, $N_{20} = (0,1,0)$, $N_{21} = (0,1,0)$, $N_{22} = (0,1,0)$, $N_{23} = (0,1,0)$.

- Les mailles: $M_0 = (0,1,2)$, $M_1 = (0,2,3)$, $M_2 = (4,5,6)$, $M_3 = (4,6,7)$ $M_4 = (8,9,10)$, $M_5 = (8,10,11)$, $M_6 = (12,13,14)$, $M_7 = (12,14,15)$, $M_8 = (16,17,18)$, $M_9 = (16,18,19)$, $M_{10} = (20,21,22)$, $M_{11} = (20,22,23)$.

Le cylindre

- Les sommets: $P_i = P_{i+2(N+1)} = (r\cos(\frac{2i\pi}{N}), -\frac{h}{2}, r\sin(\frac{2i\pi}{N}))$ pour $i = 0,\ 1,\ ...,\ N$ (qui donnent les points de la face du haut) et $P_{i+N+1} = P_{i+3(N+1)} = (r\cos(\frac{2i\pi}{N}), \frac{h}{2}, r\sin(\frac{2i\pi}{N}))$ pour $i = 0,\ 1,\ ...,\ N$ (qui donnent les points de la face du bas). Enfin, on ajoute $P_{4(N+1)} = (0, -\frac{h}{2}, 0)$ et $P_{4N+5} = (0, \frac{h}{2}, 0)$.
- Les normales: $N_i = (\cos(\frac{2i\pi}{N}), 0, \sin(\frac{2i\pi}{N}))$ pour $i = 0,\ 1,\ ...,\ N$ (qui donnent les normales au corps du cylindre des sommets de la face du haut) et $N_{i+N+1} = (\cos(\frac{2i\pi}{N}), 0, \sin(\frac{2i\pi}{N}))$ pour $i = 0,\ 1,\ ...,\ N$ (qui donnent les normales au corps du cylindre des sommets de la face du bas), $N_{i+2(N+1)} = (0, -1, 0)$ pour $i = 0,\ 1,\ ...,\ N$, $N_{i+3(N+1)} = (0, 1, 0)$ pour $i = 0,\ 1,\ ...,\ N$. On a aussi $N_{4N+4} = (0, -1, 0)$ et $N_{4N+5} = (0, 1, 0)$.
- Les mailles:
  - pour la face du haut: $M_i = (2(N+1)+i, 2(N+1)+i+1, 4N+4)$, pour $i = 0,\ 1,\ ...,\ N-1$.
  - pour la face du bas: $M_{N+i} = (3(N+1)+i, 3(N+1)+i+1, 4N+5)$, pour $i = 0,\ 1,\ ...,\ N-1$.
  - pour le corps du cylindre: $M_{2i+2N} = (i, i+1, i+N+2)$ et $M_{2i+1+2N} = (i, i+N+2, i+N+1)$ pour $i = 0, 1, ..., N-1$.

La sphère

- Les sommets: les mêmes sommets (voir page 18)
- Les normales:
  - au pôle sud: $N_0 = (0, 1, 0)$.
  - les normales intermédiaires:
$$N_{(i-1)(N+1)+j+1} = \left(\sin(\frac{i\pi}{N})\cos(\frac{2j\pi}{N}), \cos(\frac{i\pi}{N}), \sin(\frac{i\pi}{N})\sin(\frac{2j\pi}{N})\right)$$
    pour $i = 1, 2, ..., N-1$ et $j = 0, 1, ..., N$.
  - au pôle nord: $N_{N^2} = (0, -1, 0)$.
- Les mailles:
  - Les mailles contenant le pôle sud ne changent pas. On a
$$M_j = (0, j+1, j+2)$$
    pour $j = 0, ..., N-1$.
  - Les mailles intermédiaires: on a
$$M_{2(iN+j)-N} = ((i-1)(N+1)+j+1,\ (i-1)(N+1)+j+2,\ i(N+1)+j+2))$$
    et
$$M_{2(iN+j)-N+1} = ((i-1)(N+1)+j+1,\ i(N+1)+j+2,\ i(N+1)+j+1)\,,$$
    pour $i = 1, ..., N-2$ et $j = 0, 1, ..., N-1$.
  - Les mailles contenant le pôle nord: On a
$$M_{j+2N^2-3N} = \left(N^2, (N-2)(N+1)+j+1, (N-2)(N+1)+j+2\right)\,,$$
    pour $j = 0, 1, ..., N-1$.

Pour appliquer les transformations que nous avons vues (translations et rotations), il va nous falloir modifier la valeur des normales. Ceci ne pose pas de réelles difficultés. En effet:

- dans le cas de la translation la normale ne change pas.
- dans le cas d'une rotation $R_{\mathcal{D},\theta}$, autour d'une droite $\mathcal{D}$ passant par l'origine, la nouvelle normale est $N_i' = R_{\mathcal{D},\theta}(N_i)$. Si $\mathcal{D}$ ne passe pas par l'origine, on applique $R_{\mathcal{D}',\theta}$, où $\mathcal{D}'$ est une translaté de $\mathcal{D}$ passant par l'origine.

Interpolation linéaire des normales à la surface

Nous sommes ici dans la situation où on a une maille triangulaire de sommets P_1, P_2 et P_3. En chacun de ces sommets, on connaît une normale N_1, N_2 et N_3. Pour interpoler en un point quelconque X de la maille la valeur de N, on va interpoler chacune de ces composantes une à une. Afin de simplifier les notations, nous noterons dans un premier temps $f(P_1)$, $f(P_2)$ et $f(P_3)$ la valeur d'une des composantes.

Pour interpoler la valeur de f au point X, nous allons construire des fonctions affines $f_1(X)$, $f_2(X)$ et $f_3(X)$ telles que

$$f_1(P_1) = 1, f_1(P_2) = 0 \text{ et } f_1(P_3) = 0,$$
$$f_2(P_1) = 0, f_2(P_2) = 1 \text{ et } f_2(P_3) = 0,$$
$$f_3(P_1) = 0, f_3(P_2) = 0 \text{ et } f_3(P_3) = 1.$$

On pourra donc simplement poser ensuite

$$\tilde{f}(X) = f(P_1)f_1(X) + f(P_2)f_2(X) + f(P_3)f_3(X)$$

et la fonction $\tilde{f}$ ainsi créée est affine et interpole bien f.

Détaillons la construction de f_1. Notez d'abord que pour tout X sur la droite (P_2, P_3), on a

$$\overrightarrow{P_3X} \wedge \overrightarrow{P_3P_2} = (0, 0, 0).$$

Par ailleurs, si notre maillage est correct, on sait que

$$\overrightarrow{P_3P_1} \wedge \overrightarrow{P_3P_2} \neq (0, 0, 0)$$

car sinon notre maille est en fait un segment. Pour simplifier, on suppose que la première coordonnée de $\overrightarrow{P_3P_1} \wedge \overrightarrow{P_3P_2}$ est non nulle. On la note $\left(\overrightarrow{P_3P_1} \wedge \overrightarrow{P_3P_2}\right)_x \neq 0$.

La fonction

$$f_1(X) = \frac{\left(\overrightarrow{P_3X} \wedge \overrightarrow{P_3P_2}\right)_x}{\left(\overrightarrow{P_3P_1} \wedge \overrightarrow{P_3P_2}\right)_x}$$

est bien affine et vérifie $f_1(P_1) = 1$, $f_1(P_2) = 0$ et $f_1(P_3) = 0$.

De même, on pose

$$f_2(X) = \frac{\left(\overrightarrow{P_1X} \wedge \overrightarrow{P_1P_3}\right)_x}{\left(\overrightarrow{P_1P_2} \wedge \overrightarrow{P_1P_3}\right)_x}$$

et

$$f_3(X) = \frac{\left(\overrightarrow{P_2X} \wedge \overrightarrow{P_2P_1}\right)_x}{\left(\overrightarrow{P_2P_3} \wedge \overrightarrow{P_2P_1}\right)_x}$$

Notez que l'on peut se passer de vérifier que $\left(\overrightarrow{P_1P_2} \wedge \overrightarrow{P_1P_3}\right)_x \neq 0$ et $\left(\overrightarrow{P_2P_3} \wedge \overrightarrow{P_2P_1}\right)_x \neq 0$, car

$$\begin{aligned} \overrightarrow{P_3P_1} \wedge \overrightarrow{P_3P_2} &= \overrightarrow{P_1P_2} \wedge \overrightarrow{P_1P_3} \\ &= \overrightarrow{P_2P_3} \wedge \overrightarrow{P_2P_1}. \end{aligned}$$

Ceci se vérifie par un simple calcul.

En résumé, on adopte la démarche suivante:

- On calcule $\overrightarrow{P_1P_2} \wedge \overrightarrow{P_1P_3}$ (Éventuellement, composante après composante, et on s'arrête dès que l'une d'elle est non nulle.)
- On cherche la coordonnée i de $\overrightarrow{P_1P_2} \wedge \overrightarrow{P_1P_3}$ qui soit non nulle. On la mémorise dans une variable *denominateur*.

- On calcule

$$a_1 = \frac{\left(\overrightarrow{P_3X} \wedge \overrightarrow{P_3P_2}\right)_i}{denominateur},$$

$$a_2 = \frac{\left(\overrightarrow{P_1X} \wedge \overrightarrow{P_1P_3}\right)_i}{denominateur},$$

$$a_3 = \frac{\left(\overrightarrow{P_2X} \wedge \overrightarrow{P_2P_1}\right)_i}{denominateur},$$

- On calcule le vecteur normal $N = (N_x, N_y, N_z)$ en appliquant

$$N_x = a_1(N_1)_x + a_2(N_2)_x + a_3(N_3)_x,$$

$$N_y = a_1(N_1)_y + a_2(N_2)_y + a_3(N_3)_y,$$

$$N_z = a_1(N_1)_z + a_2(N_2)_z + a_3(N_3)_z.$$

- On normalise

$$N = \frac{(N_x, N_y, N_z)}{\sqrt{N_x^2 + N_y^2 + N_z^2}}$$

2.5 Le plaquage de textures

Jusqu'ici, nous n'avons représenté que des objets de couleur uniforme. Afin de représenter des objets dont la couleur change (objets texturés, dessin d'une map monde,...), nous allons "plaquer" une image 2D sur notre surface . Pour ce faire, nous avons besoin de faire correspondre à un point de notre plan de projection X, un point X' de la surface définie par le maillage (voir Section 2.3.1), puis faire correspondre à X' un point du support d'une image 2D.

Soit *image* une image 2D définie sur le support $\{0, ..., taillex\} \times \{0, ..., tailley\}$. On note $image[x][y]$ la couleur de l'image au point $(x, y) \in \{0, ..., taillex\} \times \{0, ..., tailley\}$. Nous noterons, dans cette section, $\mathcal{S}$ la surface définie par le maillage. Il nous faut donc définir une fonction

$$T : \mathcal{S} \longrightarrow [0, taillex] \times [0, tailley]$$

On notera T_x et T_y ses composantes. On appelle $T(X')$ les coordonnées de texture de $X' \in \mathcal{S}$

Nous supposerons défini sur $\mathcal{S}$ un éclairement $I(X') \in [0, 1]$. Étant donnée une projection p (la simple ou la "pinhole") la couleur finale en X, un point du plan de projection, est

$$I(X')\ image\left[Ent\left(T_x(X')\right)\right]\left[Ent\left(T_y(X')\right)\right]$$

avec $p(X') = X$. $Ent(t)$ représentant la partie entière de t.

2.5.1 Utilisation d'images sous JAVA

Pour manipuler des images sous JAVA, on va utiliser trois classes:

- *Image*: du package *java.awt.Image*. Elle permet de contenir une image. Elle contient notamment les méthodes: *int getHeight(ImageObserver img)* et *int getWidth(ImageObserver img)* qui retournent respectivement le nombre de lignes et de colonnes de l'image.

- *ColorModel*: du package *java.awt.image.ColorModel*. Elle contient le modèle (en pratique le format) de représentation de l'image. Elle contient notamment des méthodes *int getRed(int k)*, *int getGreen(int k)*, *int getBlue(int k)* qui donnent les composantes rouge, vert et bleu d'un code entier représentant une couleur.

- *PixelGrabber*: du package *java.awt.image.PixelGrabber*. Elle nous servira à interpréter l'objet *Image* qui contient l'image. Elle contient notamment une méthode constructeur *PixelGrabber(Image img, int xmin, int ymin, int xmax, int ymax, int[] pixelValue, int off, int scansize)*, une méthode *ColorModel getColorModel()* qui renvoie un modèle de couleur et une méthode *boolean grabPixels()* qui permet d'écrire le code couleur de chaque pixel dans le tableau *int[] pixelValue* (celui de la méthode constructeur).

On utilisera aussi deux méthodes de la classe *JApplet* (par héritage de la classe *Applet*): *Image getImage(URL url, String name)* qui permet de charger une image dont le nom est contenu dans *name* à l'adresse *url*, et *URL getCodeBase()* qui permet de donner l'url correspondant au répertoire où se trouve l'applet.

En résumé, on doit importer les packages nécessaires

```
import java.awt.image.*
```

Définir des variables d'instance de l'applet

```
Image notreImage;
ColorModel model;
int pixelsValue[];
int taillex,tailley;
```

Dans *init()*, charger l'image *baboon.gif* se trouvant dans le même répertoire que l'applet

```
notreImage = getImage(getCodeBase(),''baboon.gif'')
```

Enfin, on peut créer une méthode *litImage(Image img)* sur le modèle suivant:

```
public void litImage(Image img)
   {taillex=img.getWidth(this);
    tailley=img.getHeight(this);
    pixelsValue = new int[taillex*tailley];
    PixelGrabber pg = new
                PixelGrabber(img,0,0,taillex,tailley,pixelsValue,0,taillex);
    model = pg.getColorModel();
    try{pg.grabPixels();}
    catch(InterruptedException e)
      System.err.println(‘‘Erreur de lecture de l'image'');
   }
```

On peut ensuite utiliser les valeurs en chaque pixel (par exemple ici pour l'affichage direct)

```
public void afficheImage(Graphics2D dessin)
   {Color col;
    int colr,colv,colb;
    int x,y;

    for(x=0;x<taillex;x++)for(y=0;y<tailley;y++)
      {colr = model.getRed(pixelsValue[y*taillex+x]);
       colv = model.getGreen(pixelsValue[y*taillex+x]);
       colb = model.getBlue(pixelsValue[y*taillex+x]);

       col= new Color(colr,colv,colb);
       dessin.setColor(col);
       dessinePoint(x,y,dessin);
      }
   }
```

Il est évidement possible d'adopter plusieurs variantes (demander à l'utilisateur de choisir le nom de l'image à charger, avoir plusieurs images,...).

2.5.2 Correspondance entre points du maillage et points d'un rectangle

Stratégie générale

On adopte la stratégie suivante:

- Définir l'image $T(P_i)$, pour $i = 0, ..., n-1$, où les P_i sont les sommets du maillage.
- Déterminer $T(X')$ pour un point X' de la maille M_k, en interpolant linéairement les valeurs de $T(P_i)$ aux sommets P_i de la maille M_k.

En ce qui concerne la définition des $T(P_i)$, suivant le résultat souhaité, on cherchera souvent à les répartir au mieux. Par exemple, on évitera de "replier" l'image en prenant un $T(P_i)$ à l'intérieur de l'image d'une maille ne contenant pas P_i (voir Figure 2.10).

En pratique, on mémorisera les images des sommets dans un *Vector* contenant des *Point2D*. On mettra le tout dans un variable d'instance de la classe *Maillage*. Notez qu'il peut être bon de définir un booléen valant *true* si on utilise une texture et *false* sinon.

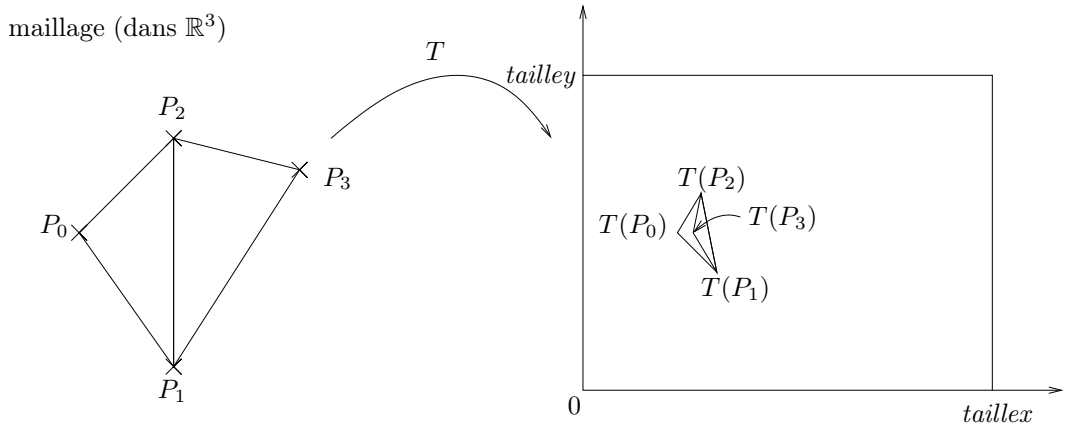

Figure 2.10: Cas de repliement de l'image de texture.

L'interpolation linéaire est bien adaptée. En effet, elle transforme un segment en un segment, et donc un triangle en un triangle. Elle est de plus continue, ce qui nous garantit que l'intérieur de la maille restera à l'intérieur du triangle image de la maille. Elle est de plus surjective (sur le triangle image), ce qui nous garantit que nous ne "manquerons" pas une partie de l'image.

On effectue cette interpolation de la même façon que dans la section 2.4.2. On effectue une interpolation linéaire de chacune des coordonnées $T_x(X')$ et $T_y(X')$. On arrondira ensuite ces valeurs à l'entier entre 0 et $taillex$ (ou $tailley$) le plus proche.

Notez que dans la plupart des cas, la surface que l'on cherche à mettre en correspondance avec le rectangle possède des propriétés mathématiques différentes de ce dernier. De ce fait, le choix de l'image peut s'avérer important. Par exemple, pour plaquer une image sur le corps d'un cylindre, il vaut mieux prendre une image "périodique" (les valeurs au bord droit de l'image sont les mêmes que celles de son bord gauche). Si ce n'est pas la cas, on obtient une brusque rupture sur le cylindre.

Le cube

Dans le cas du cube, le problème de la pèriodisation que nous venons de mentionner est insoluble. Nous proposons ici de mettre la même image sur toutes les faces du cube.

Pour ce faire, on rajoute au maillage défini page 29 une variable qui contient pour chaque sommet du maillage le point correspondant sur l'image.

- Les coordonnées de textures: $T_0 = (0, tailley)$, $T_1 = (taillex, tailley)$, $T_2 = (taillex, 0)$, $T_3 = (0, 0)$, $T_4 = (0, 0)$, $T_5 = (taillex, 0)$, $T_6 = (taillex, tailley)$, $T_7 = (0, tailley)$, $T_8 = (taillex, tailey)$, $T_9 = (0, tailley)$, $T_{10} = (0, 0)$, $T_{11} = (taillex, 0)$, $T_{12} = (taillex, tailey)$, $T_{13} = (0, tailley)$, $T_{14} = (0, 0)$,

$T_{15} = (taillex, 0)$, $T_{16} = (taillex, tailey)$, $T_{17} = (0, tailley)$, $T_{18} = (0, 0)$, $T_{19} = (taillex, 0)$, $T_{20} = (0, 0)$,$T_{21} = (taillex, 0)$, $T_{22} = (taillex, tailey)$, $T_{23} = (0, tailley)$.

Le cylindre

Nous allons ici plaquer une image sur le corps du cylindre et une image sur chaque extrémité. Les coordonnées de texture que nous donnons ici correspondent au maillage décrit en page 29. De plus, elles sont données avant l'arrondi. On note $taillemin = \min(taillex, tailley)$.

- Les coordonnées de textures:
  - $T_i = (taillex - i\frac{taillex}{N}, tailley)$, pour $i = 0,\ 1,\ ...,\ N$
  - $T_{i+N+1} = (taillex - i\frac{taillex}{N}, 0)$, pour $i = 0,\ 1,\ ...,\ N$.
  - $T_{i+2(N+1)} = T_{i+3(N+1)} = (\frac{taillex}{2} + \frac{taillemin}{2}\cos(\frac{2i\pi}{N}), \frac{tailley}{2} - \frac{taillemin}{2}\sin(\frac{2i\pi}{N}))$, pour $i = 0,\ 1,\ ...,\ N$.
  - $T_{4N+4} = (\frac{taillex}{2}, \frac{tailley}{2})$ et $T_{4N+5} = (\frac{taillex}{2}, \frac{tailley}{2})$.

Notez ici que l'on aurait pu choisir bien d'autres façons de plaquer l'image sur le maillage.

La sphère

Rappelons qu'un point de la sphère est repéré en coordonnées polaires par un point $(\theta, \varphi) \in [0, 2\pi] \times [0, \pi]$. Nous avons choisi de faire correspondre le rectangle $[0, 2\pi] \times [0, \pi]$ avec $[0, taillex] \times [0, tailley]$. Ainsi, à un point de la sphère de coordonnées polaires (θ, φ), nous ferons correspondre la coordonnée de texture $(\theta\frac{taillex}{2\pi}, \varphi\frac{tailley}{\pi})$.

Là encore, les coordonnées de texture sont données avant l'arrondi et pour le maillage décrit en page 30 (et dont les sommets sont décrits page 18).

- Les coordonnées de textures:
  - au pôle sud: $T_0 = (\frac{taillex}{2}, 0)$. (Notez que la valeur de θ, pour définir le pôle sud, n'est pas unique. Nous avons choisi de prendre $\theta = \pi$.)
  - les points intermédiaires: le sommet $P_{(i-1)(N+1)+j+1}$ est le point d'intersection de la parallèle d'angle $\varphi_i = \frac{i\pi}{N}$ et du méridien d'angle $\theta_j = \frac{2j\pi}{N}$ pour $i = 1, 2, ..., N-1$ et $j = 0, 1, ..., N$. On a donc
  $$T_{(i-1)(N+1)+j+1} = \left(j\frac{taillex}{N}, i\frac{tailley}{N}\right)$$
  - le pôle nord: $T_{N^2} = (\frac{taillex}{2}, tailley)$. (On a ici la même indéterminée que pour le pôle sud.)

Notez qu'il semble recommandable d'avoir une image dont le haut et le bas soient de couleur plus ou moins uniforme car cette partie voit sa surface se réduire énormément lorsque l'on s'approche des pôles.

Remerciements

Je voudrais remercier Rémy Malgouyres et Jacqueline Castaing pour le matériel et les idées qu'ils m'ont fournies pour faire ce cours.

Bibliographie

Ce cours est inspiré de

- Rémy Malgouyres, "Algorithmes pour la synthèse d'images et l'animation 3D", DUNOD, collection SciencesSup, 2002. ISBN 2-1000-6624-2.