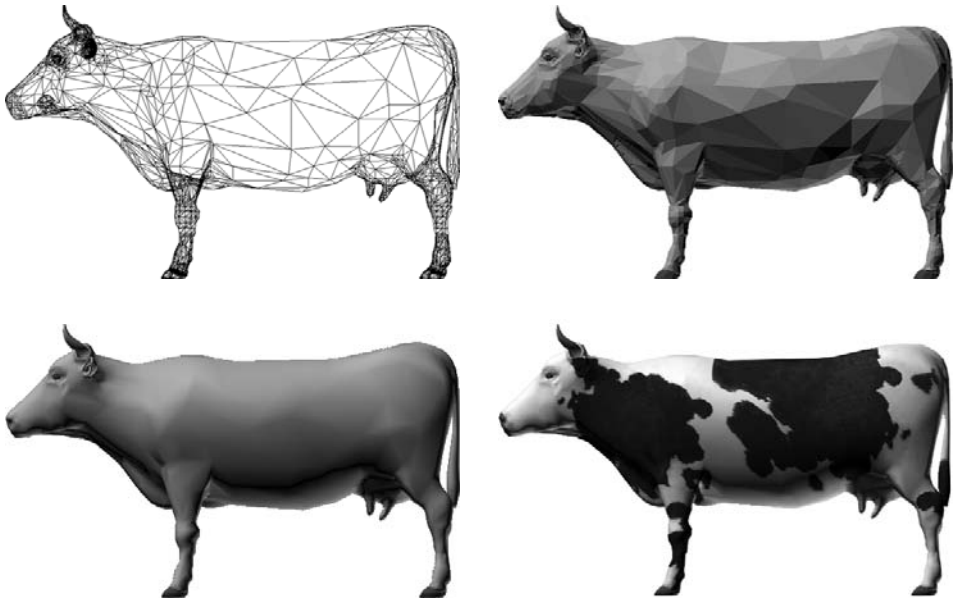
UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

Initiation à la synthèse d'images

Rendu d'objets polygonaux

Pascal Mignot



Cours de Maîtrise d'informatique

ANNÉE UNIVERSITAIRE 2004-2005

Table des matières

1 Géométrie **7**
1 Rappel de calcul vectoriel 8
1.1 Base d'un espace vectoriel 8
1.2 Somme de vecteurs 8
1.3 La norme d'un vecteur 8
1.4 Le produit scalaire 9
1.5 Le produit vectoriel 9
1.6 Changement de base 10
2 Rappel de quelques propriétés des espaces affines 10
2.1 Transformations de base 11
2.2 Changement de repère 12
3 Mise en situation géométrique 12
3.1 Observateur et repère de l'observateur 13
3.2 Mise en perspective et projection 14
3.3 Analogie avec l'appareil photographique 17
3.4 Retour à la grille de l'écran 20
3.5 Calibrage complet d'une caméra 22
4 Objets polygonaux 24
5 Exercices 28

2 Algorithmes 2D **31**
1 Le tracé de droites 31
1.1 Considérations 32
1.2 Un algorithme simple 33
1.3 L'algorithme de Bresenham (ou algorithme du point milieu) . 35
2 Remplissage 37
3 Clipping 39
4 Exercices 42

3 Algorithmes 3D **47**
1 Clipping 3D 47
2 Élimination des faces cachées 50
2.1 Algorithme du peintre 50

2.2 Algorithme du tampon de profondeur (Z-buffer) 52
3 Exercices 55

4 Rendu **59**
1 Rendu 60
1.1 Sources de lumière 60
1.2 Modèle élémentaire d'illumination 61
1.3 Ombres 66
2 Cas particulier des polygones 69
2.1 Rendu des surfaces polygonales 69
2.2 Ombrage de Gouraud 70
2.3 Ombrage de Phong 71
2.4 Problème avec ces modèles 73
2.5 Calcul des normales 76
3 Exercices 76

5 Pipeline graphique **79**
0.1 Pipeline des traitements 79

Introduction

- rendu géométrique :
  - description géométrique de la scène :
    pour tous les objets de la scène, on connaît :
    - leurs descriptions mathématiques. Par exemple :
      - pour une surface, l'équation de cette surface.
      - pour une sphère, son rayon.
    - leurs mises en situation dans la scène
      - position
      - orientation
      - échelle
  - positionnement d'un observateur.
    - position
    - la direction de son regard
  - mise en perspective : déformation des objets de la scène afin que ceux-ci paraissent petit quand ils sont loin.
  - suppression des faces cachées de la scène.
- rendu réaliste :
  - mise en place de sources de lumière, et modèle de sources lumineuses.
  - attribution de caractéristiques physiques aux objets :
    - couleur
    - propriétés réflexion
  - traitement pour réduire les artefacts
    - gouraud/phong

Chapitre 1

Bases de géométrie pour la synthèse d'images

Contenu du chapitre

1 Rappel de calcul vectoriel 8

1.1 Base d'un espace vectoriel 8

1.2 Somme de vecteurs 8

1.3 La norme d'un vecteur 8

1.4 Le produit scalaire 9

1.5 Le produit vectoriel 9

1.6 Changement de base 10

2 Rappel de quelques propriétés des espaces affines 10

2.1 Transformations de base 11

2.2 Changement de repère 12

3 Mise en situation géométrique 12

3.1 Observateur et repère de l'observateur 13

3.2 Mise en perspective et projection 14

3.3 Analogie avec l'appareil photographique 17

3.4 Retour à la grille de l'écran 20

3.5 Calibrage complet d'une caméra 22

4 Objets polygonaux 24

5 Exercices 28

1 Rappel de calcul vectoriel

On rappelle dans cette partie les définitions et les propriétés de base du calcul dans l'espace vectoriel $\mathbb{R}^3$ (cet espace ne contient **que** des vecteurs).

1.1 Base d'un espace vectoriel

Une famille de vecteurs $\{\mathbf{e}_i\}_{i=1...n}$ est une base dans un espace vectoriel E de dimension n si pour tout vecteur $\mathbf{V}$ de E, il existe un unique n-uplet $\{\lambda_i\}_{i=1...n}$ tel que :

$$\mathbf{V} = \sum_{i=1}^{n} \lambda_i.\mathbf{e}_i$$

Autrement dit, dans une base, tout vecteur s'exprime de façon unique en fonction des vecteurs de cette base. Les $\{\lambda_i\}_{i=1...n}$ sont appelés les coordonnées du vecteur $\mathbf{V}$ dans la base $\{\mathbf{e}_i\}_{i=1...n}$.

Dans le cas $\mathbb{R}^3$ qui nous intéresse, les vecteurs dits "canoniques" (ou implicites) sont les vecteurs $\mathbf{e}_1 = (1,0,0)$, $\mathbf{e}_2 = (0,1,0)$ et $\mathbf{e}_3 = (0,0,1)$. Par exemple, le vecteur $\mathbf{V} = (a,b,c)$ s'écrit sous la forme :

$$\mathbf{V} = a.\mathbf{e}_1 + b.\mathbf{e}_2 + c.\mathbf{e}_3$$

Les éléments du triplet (a,b,c) sont les coordonnées du vecteur $\mathbf{V}$ dans la base $(\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3})$.

Par la suite, sauf indication contraire, on se placera dans cette base canonique.

1.2 Somme de vecteurs

Soit $\mathbf{V_1} = (x_1, y_1, z_1)$ et $\mathbf{V_2} = (x_2, y_2, z_2)$ deux vecteurs de $\mathbb{R}^3$. Le somme de $\mathbf{V_1}$ et $\mathbf{V_2}$ est :

$$\mathbf{V_1} + \mathbf{V_2} = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$$

1.3 La norme d'un vecteur

Définition *Soit $\mathbf{V} = (x,y,z)$ un vecteur de $\mathbb{R}^3$. La norme du vecteur $\mathbf{V}$ (notée $|\mathbf{V}|$) est définie par :*

$$|\mathbf{V}| = \sqrt{x^2 + y^2 + z^2}$$

Propriétés

- *La norme représente la "longueur" du vecteur.*
- *On dit qu'un vecteur est normé (ou unitaire) si sa norme est égale à 1 (i.e. $\left|\vec{V}\right| = 1$). Si $\mathbf{V}$ est un vecteur quelconque non nul, alors $\frac{\mathbf{V}}{|\mathbf{V}|}$ est un vecteur normé.*

1.4 Le produit scalaire

Définition *Soit $\mathbf{V_1} = (x_1, y_1, z_1)$ et $\mathbf{V_2} = (x_2, y_2, z_2)$ deux vecteurs de $\mathbb{R}^3$. Le produit scalaire entre les deux vecteurs $\mathbf{V_1}$ et $\mathbf{V_2}$ est le scalaire défini par :*

$$\mathbf{V_1}.\mathbf{V_2} = x_1x_2 + y_1y_2 + z_1z_2$$

Propriétés

- *$\mathbf{V_1}$ et $\mathbf{V_2}$ sont orthogonaux si et seulement si $\mathbf{V_1}.\mathbf{V_2} = 0$.*
- *si θ est l'angle entre $\mathbf{V_1}$ et $\mathbf{V_2}$, alors on a : $\mathbf{V_1}.\mathbf{V_2} = |\mathbf{V_1}| \, . \, |\mathbf{V_2}| \cos\theta$.*
- *si les vecteurs $\mathbf{V_1}$ et $\mathbf{V_2}$ sont unitaires, alors $\mathbf{V_1}.\mathbf{V_2} = \cos\theta$. Cette propriété est très souvent utilisée car un produit scalaire est moins coûteux que l'appel de la fonction* cos*.*
- $\mathbf{V_1}.\mathbf{V_1} = |\mathbf{V_1}|^2$

1.5 Le produit vectoriel

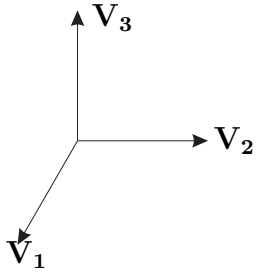
Soit $\mathbf{V_1} = (x_1, y_1, z_1)$ et $\mathbf{V_2} = (x_2, y_2, z_2)$ deux vecteurs de $\mathbb{R}^3$.

Définition *Le produit vectoriel (noté $\wedge$) entre les deux vecteurs $\mathbf{V_1}$ et $\mathbf{V_2}$ est le vecteur défini par :*

$$\mathbf{V_1} \wedge \mathbf{V_2} = \begin{pmatrix} y_1z_2 - y_2z_1 \\ z_1x_2 - z_2x_1 \\ x_1y_2 - x_2y_1 \end{pmatrix}$$

Propriétés

- *le produit vectoriel de $\mathbf{V_1}$ et $\mathbf{V_2}$ est orthogonal à $\mathbf{V_1}$ et à $\mathbf{V_2}$.*
- *Le produit vectoriel de $\mathbf{V_1}$ avec lui-même est le vecteur nul.*
- *Si $(\mathbf{V_1}, \mathbf{V_2}, \mathbf{V_3})$ forment une base orthogonale*[1]*, alors :*
  - *Si $\mathbf{V_3} = \mathbf{V_1} \wedge \mathbf{V_2}$, alors le repère est dit direct (i.e. dans le même sens que le repère canonique).*
  - *Si $\mathbf{V_3} = \mathbf{V_1} \wedge \mathbf{V_2}$, alors $\mathbf{V_1} = \mathbf{V_2} \wedge \mathbf{V_3}$ et $\mathbf{V_2} = \mathbf{V_3} \wedge \mathbf{V_1}$.*
  - *Si $\mathbf{V_3} = \mathbf{V_1} \wedge \mathbf{V_2}$, alors $\mathbf{V_2} \wedge \mathbf{V_1} = -\mathbf{V_3}$.*



Repère direct.

- *si θ est l'angle entre $\mathbf{V_1}$ et $\mathbf{V_2}$, alors on a : $|\mathbf{V_1} \wedge \mathbf{V_2}| = |\mathbf{V_1}| \, . \, |\mathbf{V_2}| \, . \, |\sin\theta|$.*

[1] *i.e.* dans une base orthogonale, les vecteurs définissant la base sont orthogonaux deux à deux. Par exemple, si $\mathbf{e}$ est une base orthogonale, on a $\mathbf{e_1}.\mathbf{e_2} = 0$, $\mathbf{e_1}.\mathbf{e_3} = 0$ et $\mathbf{e_2}.\mathbf{e_3} = 0$. La base est dite orthonormée si les vecteurs de la base sont tous de norme 1.

1.6 Changement de base

Le problème que nous nous posons maintenant est le suivant : je connais les coordonnées (a, b, c) du vecteur $\mathbf{V}$ dans la base $\mathbf{e} = (\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3})$. Quels sont alors les coordonnées de $\mathbf{V}$ dans la base $\mathbf{E} = (\mathbf{E_1}, \mathbf{E_2}, \mathbf{E_3})$?

Si on note (E_i^x, E_i^y, E_i^z) les coordonnées du vecteur $\mathbf{E_i}$ dans la base $\mathbf{e}$, alors :

$$\begin{bmatrix}\mathbf{E_1}\\ \mathbf{E_2}\\ \mathbf{E_3}\end{bmatrix} = \begin{bmatrix}E_1^x & E_1^y & E_1^z\\ E_2^x & E_2^y & E_2^z\\ E_3^x & E_3^y & E_3^z\end{bmatrix} . \begin{bmatrix}\mathbf{e_1}\\ \mathbf{e_2}\\ \mathbf{e_3}\end{bmatrix} = M. \begin{bmatrix}\mathbf{e_1}\\ \mathbf{e_2}\\ \mathbf{e_3}\end{bmatrix} \tag{1.1}$$

La matrice M représente alors la matrice de passage de la base $\mathbf{E}$ vers la base $\mathbf{e}$, et donc M^{-1} la matrice de passage inverse. En effet, le vecteur $\mathbf{V}$ s'écrit dans la base $\mathbf{e}$:

$$\mathbf{V} = \begin{bmatrix}a & b & c\end{bmatrix} . \begin{bmatrix}\mathbf{e_1}\\ \mathbf{e_2}\\ \mathbf{e_3}\end{bmatrix} \tag{1.2}$$

$$= \begin{bmatrix}a & b & c\end{bmatrix} .M^{-1}. \begin{bmatrix}\mathbf{E_1}\\ \mathbf{E_2}\\ \mathbf{E_3}\end{bmatrix} \text{ (par l'équation 1.1)} \tag{1.3}$$

Par conséquent, les coordonnées du vecteur $\mathbf{V}$ dans la base $\mathbf{E}$ sont $\begin{bmatrix}a & b & c\end{bmatrix} .M^{-1}$.

Dans le cas où $\mathbf{e}$ et $\mathbf{E}$ sont deux bases orthogonales[1], alors la matrice de passage M est une matrice dite orthogonale dont une propriété utile est que : $M^{-1} = {}^tM$. Ceci sera (presque) toujours le cas dans le cadre dans lequel nous nous plaçons.

2 Rappel de quelques propriétés des espaces affines

On se place maintenant dans l'espace affine $\mathbb{R}^3$. Intuitivement (et classiquement), c'est l'espace géométrique en 3 dimensions tel que nous avons l'habitude de le manipuler. Par défaut, un point P y est localisé par ses trois coordonnées (x, y, z). Celles-ci sont relatives à un repère $(O, \mathbf{e})$ constitué par :

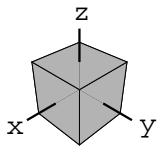
- le point d'origine O dont les coordonnées sont $(0, 0, 0)$.
- la base de vecteur canonique $\mathbf{e}$ (cf section 1.1).

La position du point P est alors obtenue par translation de vecteur (x, y, z) depuis le point O :

$$\begin{aligned} P &= O + \begin{bmatrix}x & y & z\end{bmatrix} . \begin{bmatrix}\mathbf{e_1}\\ \mathbf{e_2}\\ \mathbf{e_3}\end{bmatrix} \\ &= O + x.\mathbf{e_1} + y.\mathbf{e_2} + z.\mathbf{e_3} \end{aligned}$$

2.1 Transformations de base

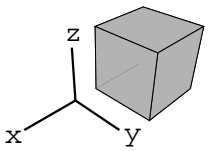
Les transformations seront présentées sur l'exemple suivant :



Pour l'écriture des transformations, on notera P le point original et P' le point transformé.

– Translation de vecteur (t_x, t_y, t_z)

$$P' = P + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$



t_x=−0.5
t_y=1
t_z=1

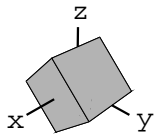
– Rotation autour d'un axe passant par O.
Si $R(\theta)$ est une matrice de rotation, alors la rotation d'un point est obtenue par :

$$P' = R(\theta).P$$

où les matrices de rotation d'angle θ suivant les 3 axes Ox, Oy, Oz notées respectivement R_x, R_y et R_z sont définies comme suit.

– Rotation d'axe Ox

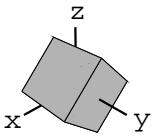
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$



θ=$\pi/4$

– Rotation d'axe Oy

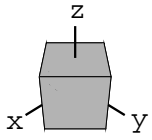
$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$



θ=$\pi/4$

– Rotation d'axe Oz

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



θ=$\pi/4$

Toute rotation dans $\mathbb{R}^3$ peut se représenter comme une composition de ces trois rotations élémentaires.

2.2 Changement de repère

Il est très important de bien comprendre les mécanismes de changement de repère car ceux-ci sont très fréquemment utilisés lors de la construction géométrique d'une scène.

Le changement du repère canonique $(O, \mathbf{e})$ à un repère quelconque $(O', \mathbf{E})$ se traite de façon similaire à celui du changement de base dans un espace vectoriel à la différence notable près qu'il faut maintenant gérer le centre de ce repère. On se place également dans le cas de changement de repères orthogonaux.

Soit P un point de coordonnées (x, y, z) dans le repère $(O, \mathbf{e})$, et de coordonnées (x', y', z') dans le repère $(O', \mathbf{E})$ où les coordonnées de $O' = (x_{O'}, y_{O'}, z_{O'})$ sont exprimées dans le repère $\mathbf{e}$. M est la matrice qui contient les coordonnées de la base $\mathbf{E}$ dans la base $\mathbf{e}$ (voir section 1.6). On a :

$$\begin{aligned}\begin{bmatrix} x & y & z \end{bmatrix} . \begin{bmatrix} \mathbf{e_1} \\ \mathbf{e_2} \\ \mathbf{e_3} \end{bmatrix} &= O' . \begin{bmatrix} \mathbf{e_1} \\ \mathbf{e_2} \\ \mathbf{e_3} \end{bmatrix} + \begin{bmatrix} x' & y' & z' \end{bmatrix} . \begin{bmatrix} \mathbf{E_1} \\ \mathbf{E_2} \\ \mathbf{E_3} \end{bmatrix} \\ &= \left(O' + \begin{bmatrix} x' & y' & z' \end{bmatrix} .M \right) . \begin{bmatrix} \mathbf{e_1} \\ \mathbf{e_2} \\ \mathbf{e_3} \end{bmatrix}\end{aligned}$$

Ainsi,

$$\begin{bmatrix} x & y & z \end{bmatrix} = O' + \begin{bmatrix} x' & y' & z' \end{bmatrix} .M$$

D'où on tire :

$$\begin{aligned}\begin{bmatrix} x' & y' & z' \end{bmatrix} &= \left(\begin{bmatrix} x & y & z \end{bmatrix} - O' \right) .{}^tM \\ &= \begin{bmatrix} x - x_{O'} & y - y_{O'} & z - z_{O'} \end{bmatrix} .{}^tM\end{aligned}$$

ou encore, après transposition,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = M. \begin{bmatrix} x - x_{O'} \\ y - y_{O'} \\ z - z_{O'} \end{bmatrix} \tag{1.4}$$

3 Mise en situation géométrique

On suppose dans cette section que nous disposons déjà d'un ensemble d'objets géométriques organisés qui représentent ce que l'on souhaite voir représentés dans la scène synthétique. La façon de construire ces objets à partir de polygones sera présenté en section 4. Le repère dans lequel ces objets sont définis est appelé le repère global.

Nous nous intéressons dans cette section aux problèmes géométriques suivants :

1. Comment définir et positionner un observateur dans la scène ?
2. Comment se place alors l'ensemble des objets de la scène vis-à-vis de l'observateur ? (changement de repère)
3. Comment faire apparaître la perspective des objets ? (mise en perspective)
4. Finalement, quelle image voit l'observateur ? (projection dans le plan de l'image)

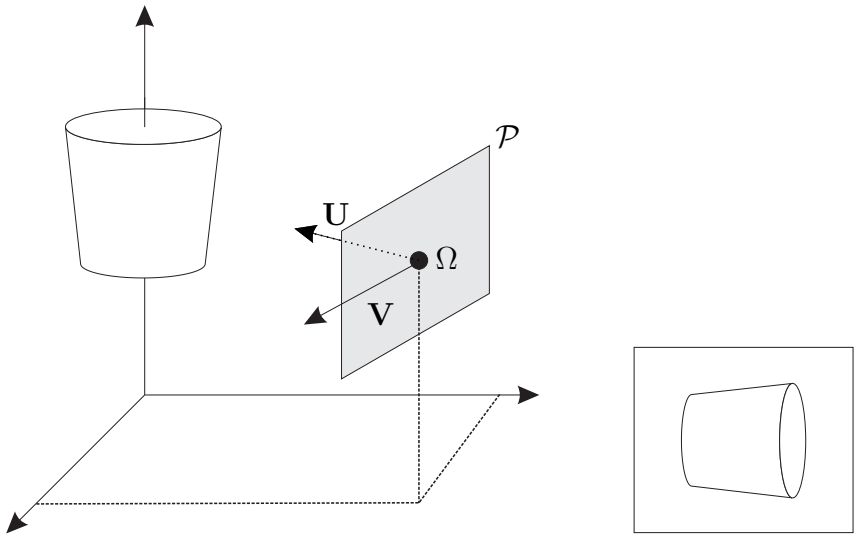
L'ensemble de ces étapes constituent la mise en situation géométrique. Nous décrivons maintenant chacune d'entre elles. Nous supposons dorénavant que tous les vecteurs manipulés sont unitaires.

A noter que le réglage des paramètres et la façon de construire le repère de l'observateur ne sont qu'indicatifs, et ne constituent pas un standard[2].

3.1 Observateur et repère de l'observateur

La position de l'observateur est un point dans le repère global. Plusieurs paramètres supplémentaires sont nécessaires pour définir complètement ce que voit l'observateur :

- Le point Ω où est positionné l'observateur.
- La direction $\mathbf{U}$ dans laquelle il regarde (direction du regard).
- La direction $\mathbf{V}$ du haut pour l'observateur (vecteur allant de ses pieds vers sa tête).



(a) Paramètres entrant dans la définition du repère de l'observateur.

(b) Ce que voit l'observateur.

FIG. 1.1: Repère de l'observateur.

Il est alors facile de construire un repère caractéristique de l'observateur à partir de ces informations. On utilise le produit vectoriel de façon à ce que le repère obtenu soit

[2]La construction présentée ici est celle utilisée sur le logiciel *RenderMan* de *Pixar*

direct :

$$\mathbf{W} = \mathbf{V} \wedge \mathbf{U} \tag{1.5}$$

L'origine du repère est placé en Ω. Une base de vecteurs adéquate pour le repère est $(\mathbf{W}, \mathbf{V}, \mathbf{U})$, car $\mathbf{U}$ doit correspondre à l'axe des z (la profondeur), $\mathbf{V}$ doit correspondre à l'axe des y (la verticale de l'écran) et $\mathbf{W}$ l'axe des x (celui qui reste : l'axe horizontal de l'écran).

En passant les objets géométriques du repère global dans le repère de l'observateur, on obtient la position et l'orientation précise de tous les objets relativement à l'observateur.

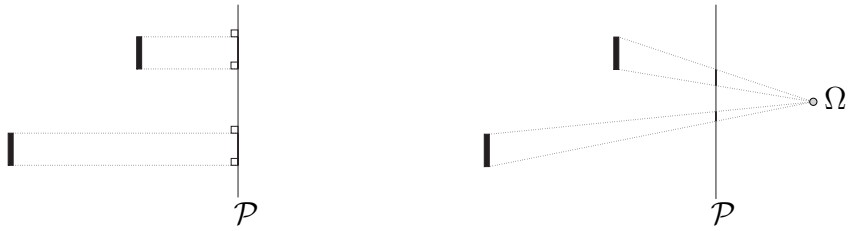
3.2 Mise en perspective et projection

L'observateur est désormais placé et nous lui avons associé le repère $(\Omega, (\mathbf{W}, \mathbf{V}, \mathbf{U}))$; mais que voit-il ?

Une première approche consiste à se dire : soit le plan définit par le point Ω et le vecteur normal $\mathbf{U}$. Ce plan (du moins une partie de celui-ci située autour de Ω) peut être considéré comme la rétine de l'observateur. Ce que l'observateur voit est la projection dans ce plan de tous les objets situés devant l'observateur.

Le vecteur normal $\mathbf{U}$ représente la direction de l'axe de "profondeur" pour l'observateur. Plus on se déplace dans cette direction, plus on s'éloigne de l'observateur. Il peut alors être tentant de considérer la projection (parallèle) consistant à supprimer la coordonnée associée à $\mathbf{U}$, pour ne garder que les deux autres (celles associées à $\mathbf{V}$ et $\mathbf{W}$). L'exemple (a) de la figure 1.2 montre que cette projection produit des résultats en désaccord avec la réalité :

- Deux objets de même taille, l'un proche ou l'autre lointain, ont la même taille.
- Deux droites partant à l'infini restent parallèles.



(a) Projection parallele. (b) Projection à un point.

FIG. 1.2: Deux différents types de projection considérés ici.

Le problème peut se résumer ainsi : il n'y a pas de déformation des objets en fonction de leurs distances à l'observateur. Nous allons donc considérer un autre type de projection (la projection à un point) qui permet d'obtenir une projection plus conforme à ce qu'observe l'oeil humain.

Le principe d'une projection à un point est simple. Elle est définie à partir d'un point Ω appelé **centre de projection** (qui nous identifieront comme la position de l'observateur) et d'un plan de projection $\mathcal{P}$ (qui pour nous sera notre écran dans lequel nous souhaitons projeter notre scène). La projection d'un point P est obtenue en prenant l'intersection de la demi-droite partant de Ω vers P avec le plan $\mathcal{P}$ (voir figure 1.2). On obtient alors un comportement similaire à l'oeil humain : les objets lointains deviennent petits, les lignes parallèle s'intersectent à l'infini (lignes de fuites).

Les équations de la transformation géométrique associée s'obtiennent facilement à partir de la figure géométrique 1.3. On considère la projection à un point définie par le point Ω et le plan de projection $\mathcal{P}$. On nomme Q le projeté du point P sur le plan $\mathcal{P}$. Ces points sont respectivement de coordonnées (x_q, y_q, z_q) et (x_p, y_p, z_p).

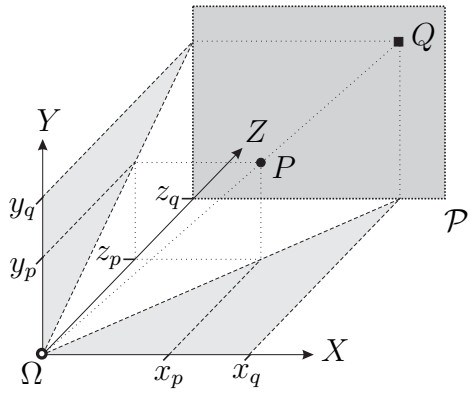


FIG. 1.3: Géométrie associée à la projection à un point.

Afin de simplifier le problème, on suppose que l'équation du plan de projection est $z = d$ (comme c'est la cas ici, z représente l'axe de profondeur de la scène -dont la direction est **U**- ; on projette sur un plan orthogonal à celui-ci). On connaît les points Ω (l'origine du repère et le centre de projection), $\mathcal{P}$ (le plan de projection), P (le point à projeter). On cherche les coordonnées du point Q.

On déduit de la figure les relations suivantes :

$$\begin{cases} \dfrac{x_q}{z_q} = \dfrac{x_p}{z_p} & \text{(voir le triangle grisé dans le plan } Oxz\text{)} \\ \dfrac{y_q}{z_q} = \dfrac{y_p}{z_p} & \text{(voir le triangle grisé dans le plan } Oyz\text{)} \\ z_q = d & \text{(par définition, projection sur le plan } \mathcal{P}\text{)} \end{cases} \Rightarrow \begin{cases} x_q = d.\dfrac{x_p}{z_p} \\ y_q = d.\dfrac{y_p}{z_p} \\ z_q = d \end{cases}$$

Cette transformation s'exprime simplement à l'aide du paramètre d qui est la distance orthogonale entre le centre de projection et le plan de projection. La projection d'un point P s'écrit donc :

$$\begin{bmatrix} x_q & y_q & z_q \end{bmatrix} = \frac{d}{z_p}. \begin{bmatrix} x_p & y_p & z_p \end{bmatrix} \tag{1.6}$$

Les coordonnées de P sur le plan de projection $z = d$ (la rétine de l'observateur) sont donc (x_q, y_q)

Nous dérivons maintenant quelques propriétés géométriques liées aux projections à un point. Considérons les droites $D_{\alpha,\beta}$ de la forme $\{x = \alpha.z,\ y = \beta.z\}$.

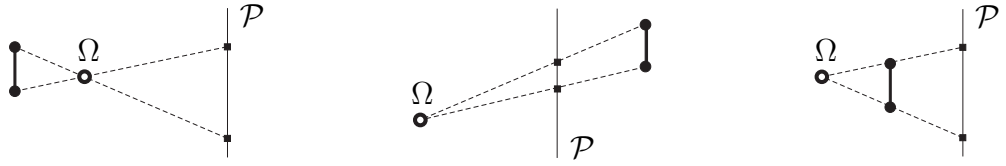
- L'intersection de toutes ces droites est le centre de projection Ω.
- L'intersection de la droite $D_{\alpha,\beta}$ et du plan $\mathcal{P}$ est le point $(\alpha d, \beta d, d)$. De plus, la projection de tout point de la droite $D_{\alpha,\beta}$ est le point $(\alpha d, \beta d, d)$.
- La projection du plan $x = \alpha.z$ (resp $y = \beta.z$) est la droite $\{x = \alpha d,\ z = d\}$ (resp. $\{y = \beta d,\ z = d\}$). Cette droite est l'intersection du plan avec le plan de projection $\mathcal{P}$.

L'ensemble des droites $D_{\alpha,\beta}$ sont appelés les droites de projection. Si on considère les 4 plans d'équation $x = \alpha_0.z$, $x = -\alpha_0.z$, $y = \beta_0.z$ et $y = -\beta_0.z$, alors toutes les droites $D_{\alpha,\beta}$ telles que $-\alpha_0 \leq \alpha \leq \alpha_0$ et $-\beta_0 \leq \beta \leq \beta_0$ sont encadrées par ces 4 plans. Comme conséquence, tout point encadré par ces plans se projette dans le carré (x, y, d) avec $x \in [-\alpha.d, \alpha.d]$ et $y \in [-\alpha.d, \alpha.d]$.

Une remarque pratique sur la projection à un point

Il est nécessaire de bien comprendre les faits suivants :

- Cette projection n'a de sens pour nous que si les points projetés sont **devant** l'observateur. Exprimé géométriquement, cela signifie que l'on ne l'applique **que** aux points dont le z dans le repère de l'observateur est strictement positif. Il est donc nécessaire d'écarter strictement tous les autres points sous peine de voir des points **derrière** l'observateur se projette **devant** lui sur le plan de projection (voir figure 1.4 (a)).
- Les déformations engendrées sont conformes avec les propriétés attendues. Plus un point est loin, plus sa projection est réduite. On notera que les objets au-delà du plan du projection rapetissent (voir figure 1.4 (b)), tandis que ceux situés entre le plan de projection et l'observateur apparaissent plus gros (figure 1.4 (c)).
- Le paramètre d calibre l'importance des déformations qu'engendre la projection à un point. Plus d est petit, et plus les déformations de perspective sont visibles. Plus d est grand, plus le résultat s'approche d'une projection parallèle.



(a) Attention, tout point placé derrière l'observateur se projette devant lui.

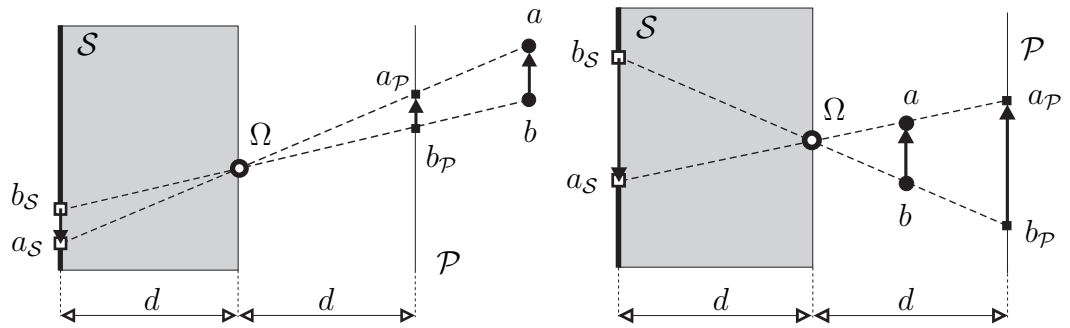
(b) Les objets placés au-delà du plan de projection rapetissent.

(c) Les objets placés entre l'observateur et le plan de projection grossissent.

FIG. 1.4: Effet de la projection à un point en fonction de la position de l'objet à projeter.

3.3 Analogie avec l'appareil photographique

"Mais cela ne marche pas comme cela dans la réalité !" pourrait dire un étudiant de maîtrise. Il peut sembler paradoxal que la "rétine" de l'observateur (*i.e.* le plan de projection) soit placée **devant** lui, et qu'il soit capable de voir des objets placés dans son oeil. Mais il s'agit seulement d'une équivalence géométrique utilisée pour simplifier l'exposé. En fait, cette projection est exactement équivalente au modèle d'appareil photographique "pinhole" (trou d'aiguille). Ce modèle est présenté à la figure 1.5 sur laquelle on montre que la projection sur le plan placé à une distance d du centre de projection donne des résultats symétriques que ce plan soit placé devant ou derrière le centre projection. On constate également qu'en plaçant le plan de projection devant l'observateur, l'image projetée n'est pas inversée (voir le sens de la flèche entre les deux points sur les plans de projection et sur l'objet).



(a) Cas de la projection d'un point situé au-delà du plan de projection.

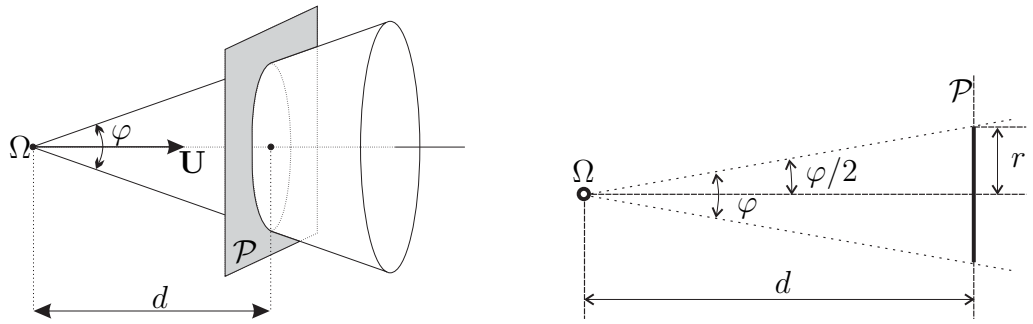
(b) Cas de la projection d'un point situé entre l'observateur et le plan de projection.

FIG. 1.5: Modèle de l'appareil photographique "trou d'aiguille". La zone grisée représente l'intérieur de l'appareil. Le centre de projection Ω est la position du trou. La portion du plan $\mathcal{S}$ en gras est la plaque photographique. La projection du point a sur le plan $\mathcal{P}$ est notée $a_\mathcal{P}$ (notation similaire pour $a_\mathcal{S}$, $b_\mathcal{P}$ et $b_\mathcal{S}$).

Afin de finir d'ancrer ce modèle géométrique dans la réalité, nous continuons par des considérations photographiques :

- Comme nous l'avons vu dans la section précédente, le point Ω est la position du trou d'aiguille ("l'objectif") par lequel passe la lumière. En photographie, on appelle ce point le *point nodal*.
- La distance d s'appelle la *distance focale*.
- Le plan de projection peut être assimilé à la pellicule (plan focal) sur laquelle l'image se forme (se projette) lorsque l'on prend une photographie.
- La droite de vecteur directeur $\mathbf{U}$ et passant par Ω est appelée l'*axe optique*. On la notera $\mathcal{D}$.
- Si on appelle Ω' le point d'intersection entre le plan de projection $\mathcal{P}$ et l'axe optique $\mathcal{D}$ alors ce point est le centre de l'image que l'on cherche à obtenir. Autrement dit, l'image recherchée est une petite zone du plan $\mathcal{P}$ autour de ce point.

– La taille d'une photo sur une pellicule dans un appareil photographique est fixe (souvent $24mm \times 36mm$). Transposé dans notre modèle, cela revient à dire que la zone du plan contenant la projection à considérer est un rectangle de position et de taille fixe. On appellera la projection sur cette partie du plan l'*image projetée*.
– On appelle cône de vision le demi-cône de centre Ω, d'axe $\mathcal{D}'$, et de plus petite ouverture φ contenant l'image projetée (voir figure 1.6 et 1.9). φ est aussi appelée *largeur du champ de vision*. On notera que le champ de vision humain n'est pas à section circulaire, pas plus que celui associé à une photographie rectangulaire.



(a) Cône de vision défini par une largeur de champ φ.

(b) Géométrie interne à l'appareil photographique.

FIG. 1.6: Les différents paramètres définissant le champ de vision de l'observateur.

Pour un utilisateur classique, la distance focale d n'a pas de sens intuitif, et surtout, elle n'est pas indicative des déformations liées à la perspective. Aussi, on lui préfère généralement la largeur φ du champ de vision (voir figure 1.6 (a)). On consultera aussi l'exercice 5 pour un exemple de calcul automatique de l'observateur.

La figure correspondant à la géométrie de la scène est présenté à la figure 1.6. On en déduit immédiatement la relation suivante :

$$\tan \frac{\varphi}{2} = \frac{r}{d}$$

où φ est la largeur de champ, d est la distance focale, et r est le "rayon" de l'image (*i.e.* le rayon du cercle le plus petit contenant le rectangle de la photographie). Comme nous le précisions dans les préliminaires, ce r est une constante. Par conséquent (voir figure 1.6), la distance focale est égale à :

$$d = \frac{r}{\tan \frac{\varphi}{2}} \tag{1.7}$$

La valeur de r peut être choisie arbitrairement (on peut prendre par exemple $r = 1$). On peut également choisir une valeur de r plus en adéquation avec le matériel utilisé

en photographie dans la réalité. En fixant l'échelle de la scène à 1 unité = $1mm$, et pour une pellicule de $24 \times 36mm$, alors la valeur de r égale à :

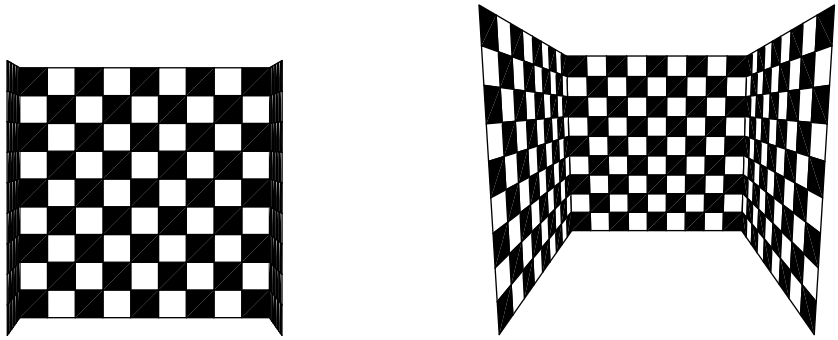
$$r = \frac{1}{2}\sqrt{24^2 + 36^2} = 21.6$$

Il est alors possible de faire la correspondance directe[3] entre les objectifs usuellement utilisés en photographie et la distance focale utilisée dans la scéne avec la formule 1.7. La table 1.1 présente les valeurs de φ associées à différents objectifs. La dernière ligne de cette table (FOV vert.) est souvent celle utilisée en synthèse car elle indique la déformation minimale de perspective observable sur les bords, tandis que la première ligne indique la déformation maximale.

dist. focale	$400mm$	$200mm$	$100mm$	$50mm$	$35mm$	$28mm$	$24mm$
FOV diag. ($r = 21.6$)	6.2°	12.3°	24.4°	46.8°	63.4°	75.4°	84.1°
FOV horiz. ($r = 36/2$)	5.2°	10.3°	20.4°	39.6°	54.4°	65.5°	73.7°
FOV vert. ($r = 24/2$)	3.4°	6.9°	13.7°	27°	37.8°	46.4°	53.1°

TAB. 1.1: Champ de vision φ (FOV=Field of Vision) associés aux différentes distances focales utilisée en photographie.

On observera sur la figure 1.7 que les petites valeurs φ offrent peu de déformations, les valeurs entre $15°$ et $40°$ donnent un aspect naturel, les valeurs au-delà de $40°$ donnent des résultats similaires aux objectifs grand-angle.



(a) Valeur de φ petite. (b) Valeur de φ grande.

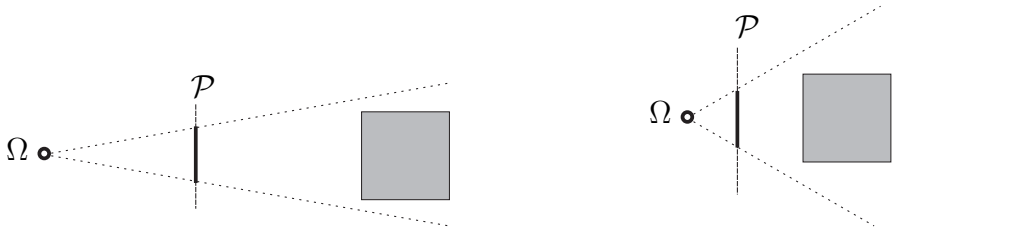
FIG. 1.7: Influence de la valeur de la largeur de champ φ sur la perspective.

Afin de mieux comprendre le rôle que joue l'ouverture de l'objectif dans un dispositif photographique (donc également pour notre observateur), nous terminons sur l'exemple de la figure 1.8 :

[3]Par définition en photographie, utiliser un objectif de $50mm$ signifie que la distance focale de cet objectif est $50mm$.

– Si on souhaite voir complètement un gros objet proche de l'observateur, il n'y a pas d'autres choix que de prendre un champ de vision large.
– Inversement, si on souhaite zoomer sur un objet qui est loin ou qui est très petit, il n'y a pas d'autre choix que de prendre un champ de vision étroit.

Par conséquent, la position de l'observateur et la largeur de champ qu'occupe l'objet qu'il veut observer déterminent complètement la perspective observée sur le film photographique (pour nous, sur notre image de synthèse). Donc, si on souhaite observer un objet avec une certaine largeur de champ, il n'y a pas d'autres solutions que de se déplacer.



(a) Plus l'objet est loin, plus φ doit être petit. (b) Plus l'objet est près, plus φ doit être grand.

FIG. 1.8: Comment régler son appareil photographique virtuel. La plaque photographique est de taille fixe (en gras sur le plan de projection).

3.4 Retour à la grille de l'écran

Dans cette partie, on s'intéresse à l'opération qui nous permet de passer de l'image projetée à une image numérique. On rappelle que :

– une image numérique est basiquement une matrice rectangulaire de valeurs.
– la taille de cette matrice correspond à la résolution de l'image. Par la suite, on notera $N \times M$ cette taille.
– à chaque valeur dans cette matrice correspond une couleur à travers une table (la Look-Up Table ou LUT). La LUT fait le lien entre un numéro d'index (le numéro de la couleur) et la couleur elle-même.

Le choix du changement de repère et de la projection à un point des sections précédentes implique que le cône de vision entier se projette sur le plan $\mathcal{P}$ dans le cercle de centre Ω' et de rayon r (voir figure 1.9).

Le champ de vision associé à une image est rectangulaire. Par conséquent, on découpe une image rectangulaire ayant les bonnes proportions dans ce cercle. Celle-ci peut soit être inscrite dans le cercle (cas FOV diagonale), soit contenir partiellement ou totalement le cercle (cas FOV verticale ou diagonale).

Pour passer maintenant d'un point de cette image rectangulaire à un point de l'image numérique, on découpe l'image rectangulaire à l'aide d'une grille régulière :

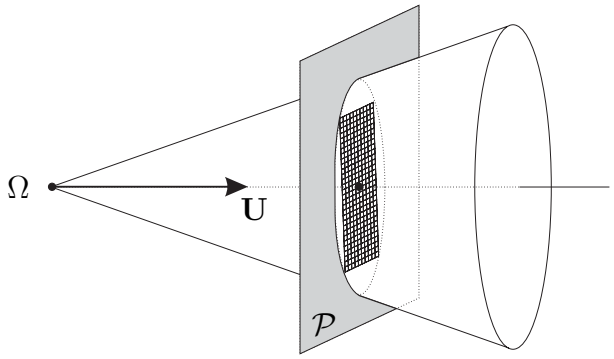


FIG. 1.9: Position de la grille de l'écran dans le repère géométrique de l'observateur.

- On choisit une grille de N carreaux horizontaux par M carreaux verticaux qui découpe l'image rectangulaire en petits carrés (sinon, l'image sera déformée).
- A chacun des carreaux, on associe ses coordonnées dans la grille (i, j) où $i \in \{0, .., N-1\}$ et $j \in \{0, .., M-1\}$. On peut ainsi créer une bijection directe entre un carreau de la grille et un point de l'image numérique de taille $N \times M$. Au point de coordonnées (i, j) de l'image numérique est associé le carreau de coordonnées (i, j) de la grille.
- A tout point dans le champ de vision dont la projection est dans le carreau (i, j) de la grille, on associe le point de coordonnées (i, j) sur l'image numérique.

En se plaçant dans le cas où le cercle est totalement inscrit dans l'image rectangulaire, la taille d'un carré de la grille est :

$$\delta = \frac{2.r}{\min\{N, M\}} \tag{1.8}$$

La transformation qui permet d'obtenir les coordonnées (i, j) du pixel sur l'image numérique correspondant au point de coordonnées (x, y, d) sur l'image projetée est donc :

$$(i,j) = \begin{cases} \left(\left\lfloor \frac{x}{\delta} + \frac{N}{2} \right\rfloor, \left\lfloor \frac{y}{\delta} + \frac{M}{2} \right\rfloor \right) & \text{si } (x,y) \in \left[-\frac{N\delta}{2}, \frac{N\delta}{2} \right] \times \left[-\frac{M\delta}{2}, \frac{M\delta}{2} \right] \\ \text{à l'extérieur de l'écran} & \text{sinon} \end{cases} \tag{1.9}$$

où $\lfloor \ \rfloor$ est l'opération d'arrondi au plus petit entier inférieur.

Afin d'accélérer les procédures d'affichage, on fait toujours[4] suivre la phase de changement de repère d'une phase dite de clipping dont le but est de déterminer la liste des objets visibles par l'observateur. Celle-ci supprime les objets à l'extérieur du cône de vision, et tronque ceux qui sont partiellement visibles. On limite ainsi le nombre d'objets à traiter à ceux (ou à leurs portions visibles) nécessaires. Les algorithmes de clipping 2D et 3D seront décrits dans la suite du cours.

[4]mais seulement si l'algorithme de rendu est de type peintre ou z-buffer.

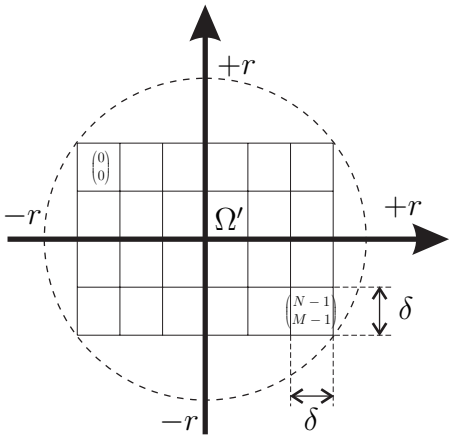


FIG. 1.10: Grille de l'écran construite à partir de la zone visible par l'observateur (le cercle).

3.5 Calibrage complet d'une caméra

Les notions que nous avons utilisées jusqu'à présent pour définir la position de l'observateur sont essentiellement géométriques. Afin de simplifier la mise en place de l'observateur et régler les paramètres géométriques, on utilise plutôt des réglages analogues à ceux d'un appareil photographique. Les réglage possibles sont les suivants :

- la position de l'observateur (point Ω)
- l'axe de l'objectif (vecteur **U**)
- la direction approximative du haut. C'est un vecteur $\mathbf{V}'$ orienté vers le haut mais pas nécessairement orthogonal à **U**. Par exemple, s'il est facile de donner la direction du plafond, il est plus difficile de donner la direction exacte orthogonale au plan du plafond.
- la largeur du champ de vision (angle φ). C'est ce paramètre qui détermine l'objectif utilisé (un grand angle pour un champ de vision large, un zoom pour un champ de vision étroit).

A partir de ces données, nous voyons maintenant comment déterminer les autres paramètres géométriques nécessaires à la construction de la base associée à l'observateur ainsi que les paramètres de la projection à un point.

Détermination de la base de associée à l'observateur

On connaît la position de l'observateur Ω, la direction de son regard **U**, et la direction approximative du haut $\mathbf{V}'$. Pour obtenir la base complète, on doit donc calculer les vecteurs **V** et **W**.

- Calcul de la direction **V** de l'axe vertical à partir de la direction approximative $\mathbf{V}'$ du haut.
  Si **U** et **V'** sont deux vecteurs unitaires, alors $\mathbf{U}.\mathbf{V}' = \cos \widehat{\mathbf{U}.\mathbf{V}'}$ comme représenté sur la figure 1.11. On remarquera aussi que les deux vecteurs **U** et **V'** définissent un plan dans lequel le vecteur **V** recherché se trouve aussi. Alors le vecteur sui-

vant :

$$\mathbf{V}' - (\mathbf{U}.\mathbf{V}')\,.\mathbf{U}$$

n'est autre (**après renormalisation**) que le vecteur recherché $\mathbf{V}$.
Erreur courante : ne pas oublier de normer les vecteurs $\mathbf{U}$ et $\mathbf{V}'$ (avant), et le vecteur $\mathbf{V}$ (après).

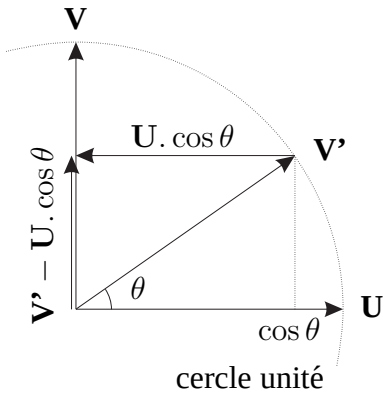


FIG. 1.11: Méthode de construction de **V** à partir de **V'**

– Calcul de la direction de l'axe horizontal $\mathbf{W}$.
Connaissant maintenant $\mathbf{V}$, on utilise directement l'équation 1.5 page 14. Le repère de l'observateur $(\Omega, (\mathbf{W}, \mathbf{V}, \mathbf{U}))$ est complètement déterminé.

Note : on inverse souvent le sens de l'axe $\mathbf{V}$ obtenu afin de tenir compte du fait que l'origine d'un écran étant en haut à gauche, le vecteur vertical pointe vers le bas, et non vers le haut. Cela se fait tout simplement en inversant le signe de la coordonnées y après transformation.

Détermination des paramètres de la projection à un point L'équation 1.7 permet de déterminer directement la valeur du paramètre d utilisé pour la projection à un point uniquement à partir de la largeur φ du champ de vision (le paramètre r étant une constante).

Chaîne de traitements complètes On dispose ainsi de l'ensemble de la chaîne de traitement permettant de projeter un objet de l'espace en 3 dimensions sur une image numérique :

1. déterminer le repère de l'observateur et les paramètres de la projection à un point comme indiqué ci-dessus.
2. se placer dans le repère de l'observateur $(\Omega, (\mathbf{W}, \mathbf{V}, \mathbf{U}))$ (voir équation 1.4).
3. effectuer la projection à un point (voir équation 1.6).
4. passer de la projection à l'image numérique (*cf* les équations 1.8 et 1.9).

A noter que l'utilisation d'une telle méthode pour projeter par exemple un cube ne permet pas de déterminer quelles sont les faces du cubes effectivement visibles par l'observateur. Cette problématique sera abordée dans le chapitre 3.

4 Objets polygonaux

Comme précisé dans l'introduction, nous ne nous intéressons dans ce cours qu'au rendu des objets définis par des polygones. Un grand nombre de techniques spécifiques et rapides existent pour leurs traitements.

Définition *Un polygone est défini à partir d'une liste de points (où sommets) coplanaires*[5] $(P_1, \ldots, P_n)$ *comme la zone délimitée par la ligne polygonale fermée reliant ses sommets dans l'ordre (i.e.* P_i *à* P_{i+1} *et* P_n *à* P_1*). On appelle arête un segment reliant deux sommets consécutifs.*

Définition *Une facette est un polygone à trois sommets.*

Les facettes sont très utilisées pour les raisons suivantes :
- 3 points sont toujours coplanaires. Donc, un polygone à trois points est toujours planaire.
- Certains algorithmes peuvent être simplifiés et accélérés dans le cas où ceux-ci ne manipulent que des facettes.
- Tout polygone peut se décomposer en une union de facettes utilisant les mêmes sommets (on ne perd rien en généralité par rapport à un polygone quelconque).

Approximation polygonale Lorsqu'un objet est représenté sous forme de polygones, cette représentation est souvent une approximation polygonale d'un objet original plus précis. Faire une approximation polygonale d'un objet 3D complexe revient à approximer celui-ci localement par des portions de plan.

Exemple 1 (Approximation polygonale d'une sphère) *On rappelle que l'équation paramétrique d'une sphère de centre* O *et de rayon* r *est :*

$$S(\theta,\phi) = \begin{cases} S_x(\theta,\phi) = r.\cos\theta.\cos\phi \\ S_y(\theta,\phi) = r.\sin\theta.\cos\phi \\ S_z(\theta,\phi) = r.\sin\phi \end{cases} \quad \textit{avec } \theta \in [0; 2\pi[, \phi \in [-\pi/2; pi/2]$$

[5] S'il est possible de définir des polygones non planaires, les algorithmes présentés dans ce cours n'y sont pas adaptés

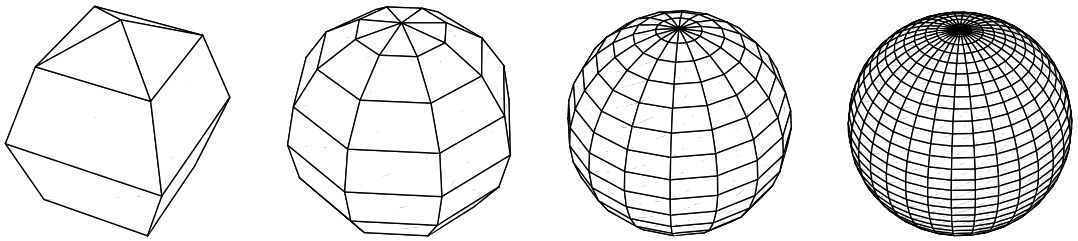
En échantillonnant régulièrement θ et ϕ de la façon suivante,

$$\begin{cases} \theta_i = i.\frac{2\pi}{N} & \text{avec } i \in \{0, \ldots, N-1\} \\ \phi_j = -\frac{\pi}{2} + j.\frac{\pi}{N} & \text{avec} j \in \{0, \ldots, N\} \end{cases}$$

et en définissant les points échantillonnés :

$$\begin{aligned} P_{i,j} &= (x_{i,j}, y_{i,j}, z_{i,j}) \\ &= (S_x(\theta_i, \phi_j), S_y(\theta_i, \phi_j), S_z(\theta_i, \phi_j)) \end{aligned}$$

et les faces $F_{i,j} = \{P_{i,j}, P_{i+1,j}, P_{i+1,j+1}, P_{i,j+1}\}$ (de chacune de ces faces, on peut construire deux facettes), on obtient une approximation polygonales de la sphère.



(a) $N = 4$ (b) $N = 8$ (c) $N = 16$ (d) $N = 32$

FIG. 1.12: Approximation polygonale d'une sphère avec $N \times N$ polygones.

La figure 1.12 présente les résultats obtenus par $N = 4$, $N = 8$, $N = 16$ et $N = 32$. Évidemment, plus N est grand, meilleure est l'approximation de l'objet original.

Règles et propriétés des objets polygonaux En règle générale, on construit les objets polygonaux en 3 dimensions de façon à ce que les propriétés suivantes soient vraies :

- Tout sommet est partagé par au moins 3 polygones.
- Toute arête est partagée par exactement deux polygones.
- Les polygones n'ont d'intersections que sur leurs arêtes.
- L'objet a un intérieur et un extérieur ; et il n'est pas possible d'aller de l'intérieur vers l'extérieur sans traverser un des polygones définissant l'objet. Ceci a pour conséquence qu'une seule des deux faces de chaque polygone est visible depuis l'extérieur (resp. l'intérieur). Ainsi, chaque polygone a une face interne et une face externe.

Il est important que ces règles soient vérifiées car elles permettent des simplifications structurelles et conceptuelles des algorithmes.

Orientation des facettes Afin de concrétiser l'intérieur et l'extérieur d'un polygone, on oriente celui-ci en donnant un sens arbitraire à l'ordre des sommets. Par exemple, si un polygone a n sommets $\{P_1, \ldots, P_n\}$,

– Lorsque je regarde la face extérieure, l'ordre des sommets est dans le sens des aiguilles d'une montre.
– Lorsque je regarde la face intérieure, l'ordre des sommets est dans le sens inverse des aiguilles d'une montre.

Ce sens peut être facilement trouvé en utilisant le produit vectoriel. Soient $\mathbf{V_1} = \overrightarrow{P_1P_2}$ et $\mathbf{V_2} = \overrightarrow{P_1P_n}$, alors le produit vectoriel $\mathbf{V_2} \wedge \mathbf{V_1}$ est orienté vers l'extérieur de la facette (voir figure 1.13).

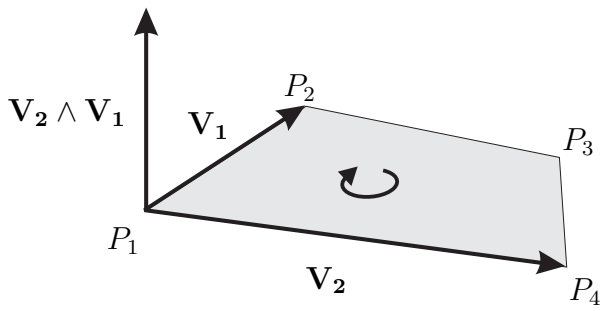


FIG. 1.13: Orientation d'un polygone

Utilisation de la normale à la facette De par les propriétés du produit vectoriel, le vecteur $\mathbf{N} = \mathbf{V_2} \wedge \mathbf{V_1}$ est normal à la facette. Le vecteur $\mathbf{N}$ a deux utilisations directes :

– Détermination si le polygone est visible
Soit $\mathbf{V}_\Omega$ la direction de l'observateur depuis le polygone (défini par exemple comme le vecteur $\overrightarrow{P\Omega}$ où P est le centre ou un point du polygone, et Ω la position de l'observateur). Le vecteur $\mathbf{N}$ étant orienté vers l'extérieur du polygone. Une facette n'est alors visible que si l'angle entre la direction de l'observation $\mathbf{V}_\Omega$ et $\mathbf{N}$ est supérieur à 90°. (voir figure 1.14). Un test très simple de cette condition revient à tester le signe du produit scalaire :

si $\mathbf{V}_\Omega.\mathbf{N} \geq 0$
alors le polygone est visible
sinon le polygone n'est pas visible

Dans le cas où le polygone n'est pas visible, la propriété sur l'existence d'un intérieur et d'un extérieur impose qu'un ou plusieurs autres polygones du même objet présenteront leurs faces visibles et occulteront complètement ce polygone. A noter que pour ce test, $\mathbf{V}_\Omega$ n'a pas besoin d'être normalisé. On ne le normalisera donc que si le polygone est potentiellement visible.
– Détermination du plan contenant le polygone
Une des propriétés du produit vectoriel implique que le vecteur $\mathbf{N}$ est normal au plan de la facette. L'équation d'un plan, dont un vecteur normal est **N** et passant

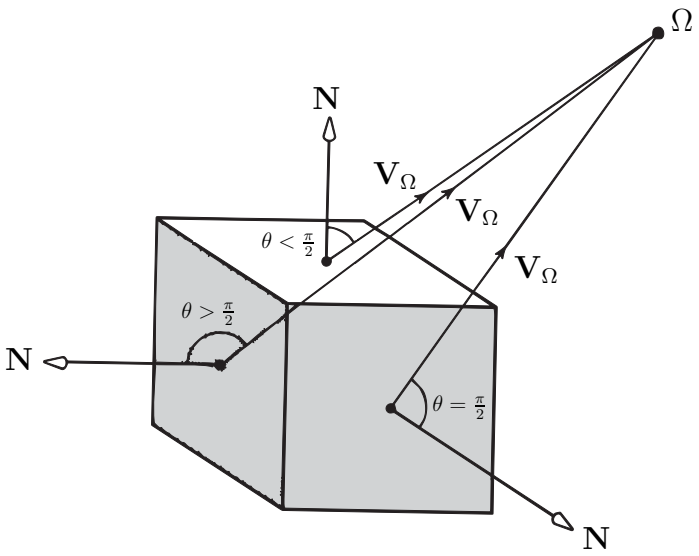


FIG. 1.14: Utilisation de la normale pour déterminer la visibilité de face. Le vecteur **N** est la normale à la face, $\mathbf{V}_\Omega$ la direction de l'observateur et θ l'angle entre **N** et $\mathbf{V}_\Omega$.

par le point P_1 est :

$$\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix} - P_1 \right) .\mathbf{N} = 0$$

Ce qui signifie qu'un point $P = (x, y, z)$ est dans ce plan si le vecteur $\overrightarrow{P_1 P}$ est orthogonal à **N**. Cette équation est, par exemple, utilisée les coordonnées de tout point du polygone.

Stockage en machine Pour stocker en mémoire un ensemble de polygones, on utilise généralement une double liste :

- Une table de tous les sommets de tous les polygones. Un sommet appartenant à plusieurs polygones n'apparaît qu'une seule fois dans cette liste.
- Une table de tous les polygones, où un polygone est stocké de la façon suivante :
  - le nombre de sommets.
  - un tableau d'entiers où chaque entier fait référence au numéro d'ordre du sommet dans la table des sommets, et l'ordre de ces entiers reflète l'ordre des sommets du polygone.

Exemple 2 (Le cube unité) *Pour le cube unité, le stockage proposé ci-dessus donne les deux tables suivantes (voir figure 1.15) :*

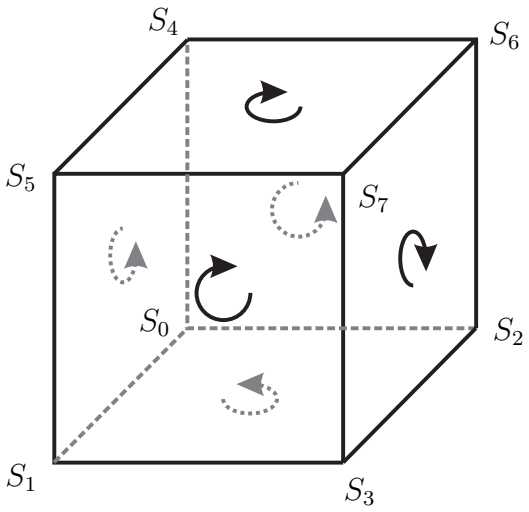


FIG. 1.15: Le cube unité : le sommet numéro i est noté S_i.

Table des sommets

num.	*coord.*
0	*(0,0,0)*
1	*(1,0,0)*
2	*(0,1,0)*
3	*(1,1,0)*
4	*(0,0,1)*
5	*(1,0,1)*
6	*(0,1,1)*
7	*(1,1,1)*

Table des polygones

nb sommets	*liste sommets*			
4	*0*	*1*	*3*	*2*
4	*4*	*6*	*7*	*5*
4	*0*	*4*	*5*	*1*
4	*2*	*3*	*7*	*6*
4	*0*	*2*	*6*	*4*
4	*1*	*5*	*7*	*3*

5 Exercices

Exercice 1 (Rappel de géométrie élémentaire) *Prouver que si $\mathbf{V}_1$ et $\mathbf{V}_2$ sont deux vecteurs unitaires, alors leurs produits scalaire et vectoriel donnent bien les résultats attendus.*

Exercice 2 (Rotation d'axe quelconque) *Soit un axe quelconque défini par un point A et un vecteur $\mathbf{V}$. Donner l'expression de la rotation d'angle θ défini autour de cet axe (indice : utiliser un changement de repère).*

Exercice 3 (Calcul numérique d'une position d'observation) *On reprend le cube unité lors de ce chapitre. On fixe le vecteur d'observation suivant :* $P = (2, 2, 4)$, $\mathbf{U} = (-1, -1, -2)$, $\mathbf{V}' = (0, 0, 1)$.

1. *Effectuer le changement de repère plaçant tous les points du cube dans le repère de l'observateur.*
2. *Représenter le résultat d'une projection orthogonale de ce cube dans le plan Oxy de l'observateur.*
3. *On choisit maintenant une largeur de champ de vision de $\varphi = 45^\circ$. Calculer la nouvelle position des points après projection.*
4. *Représenter le cube obtenu après mise en projection.*
5. *Calculer la position des points sur l'écran si celui-ci a une résolution de 512×512.*

Exercice 4 (Approximation polygonale d'un cylindre creux) *On considère le cylindre creux suivant, centré en 0, de hauteur 2, de rayon extérieur 1, de rayon intérieur 0.8, et d'axe Oz (voir figure 1.16).*

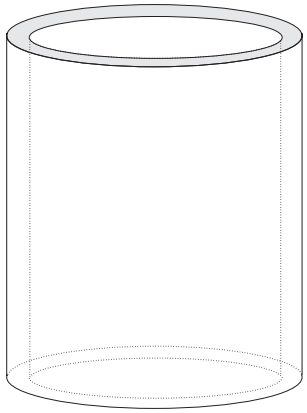


FIG. 1.16: Cylindre creux de l'exercice 4

1. *En approximant la section ronde du cylindre par un objet polygonal à 6 côtés, proposer une approximation polygonale de ce volume. On pourra suivre le cheminement suivant :*
   (a) *Rappeler l'équation paramétrique d'un cylindre.*
   (b) *Déterminer et numéroter l'ensemble des points qui vont servir de sommets pour les polygones.*
   (c) *Déterminer le nombre de polygones nécessaire.*
   (d) *Les représenter.*
2. *Après avoir choisi un sens d'orientation, orienter chaque face.*
3. *Écrire les deux tableaux nécessaires au stockage de ce polygone dans la mémoire d'un ordinateur.*

Exercice 5 (Calcul automatique d'une position d'observation) *On considère un objet inclus dans une sphère englobante de centre A et de rayon R. On veut regarder cet objet dans la direction* $\mathbf{U}$. *Soit le vecteur* $\mathbf{V}'$ *indiquant le haut de l'objet, et φ l'angle d'ouverture du champ de vision.*

- *Proposer une façon de calibrer la distance d' observation d'un objet en fonction de sa taille et de l'angle d'ouverture afin que l'objet considéré (donc la sphère) occupe juste la totalité de l'écran.*
- *Calculer la position de l'observateur en fonction de tous les paramètres proposés dans l'énoncé.*

Chapitre 2

Algorithmes de base en deux dimensions

Contenu du chapitre

1 Le tracé de droites 31
1.1 Considérations 32
1.2 Un algorithme simple 33
1.3 L'algorithme de Bresenham (ou algorithme du point milieu) 35
2 Remplissage 37
3 Clipping 39
4 Exercices 42

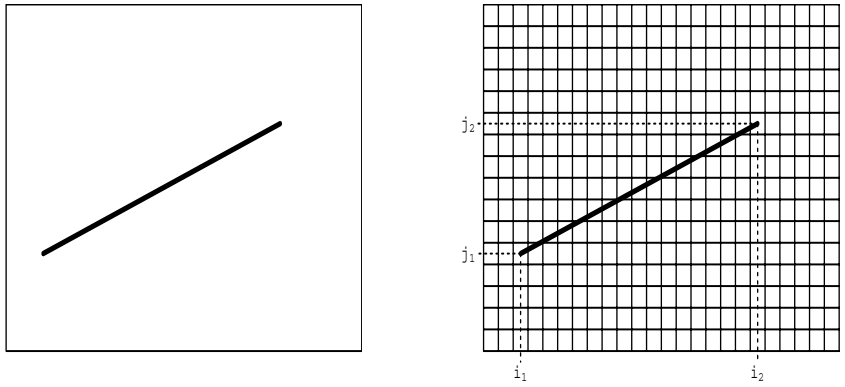
On traite des moyens d'effectuer le dessin de primitives de base sur une grille discrète (i.e. un écran d'ordinateur). Ces primitives d'affichage en deux dimensions sont celles qui sont utilisées pour l'affichage de tout objet graphique. La représentation d'un objet en 3 dimensions n'est faite qu'aprés sa projection sur le plan 2D de l'image, et se finalise donc par un ensemble de primitives en 2 dimensions. Il est donc important de disposer d'algorithmes efficaces.

1 Le tracé de droites

Les primitives que nous voulons dessiner sont définies à l'origine dans un espace géométrique ($\mathbb{R}^2$) qui est continu. On définit la grille discrète comme un couple (i, j) de $\mathbb{N}^2$, appelé pixel de coordonnées (i, j) et noté $p_{i,j}$.On peut alors définir une application[1] (surjective) qui à tout point (x, y) de $\mathbb{R}^2$ associe le point (i, j) de $\mathbb{N}^2$ tel que $(x, y) \in I_{i,j}$

[1]Cette application est tout simplement (et tout naturellement) la partie entière de chaque coordonnée !

où $I_{i,j} = [i, i+1) \times [j, j+1)$. La grille discrète peut alors se voir comme un ensemble d'intervalles sur $\mathbb{R}^2$. Nous allons voir que cette application (dont le choix semble le plus naturel) ne répond pas aux contraintes que l'on s'impose pour transformer un objet continu de $\mathbb{R}^2$ en un objet discret de $\mathbb{N}^2$.



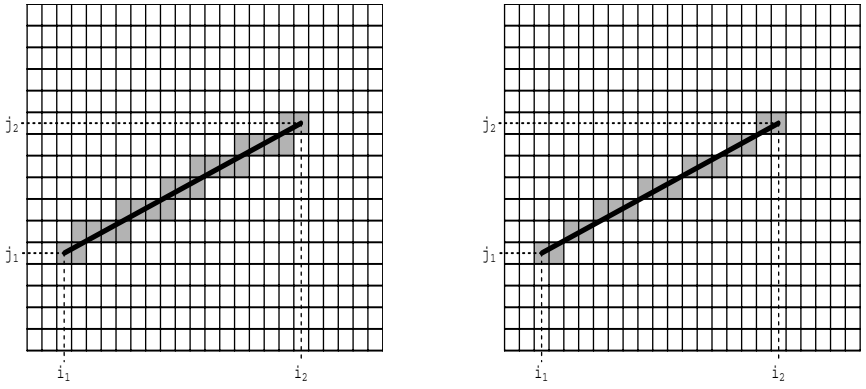
(a) Ligne dans l'espace continu $\mathbb{R}^2$

(b) Espace discret : comment représenter la ligne

FIG. 2.1: Problème de la définition et de la représentation d'une ligne (continue) dans un espace discret

1.1 Considérations

On cherche à tracer un segment entre deux points de coordonnées entières (i_0, j_0) et (i_1, j_1) sur un écran (bitmap). On cherche un algorithme rapide et efficace répondant aux spécificités suivantes (voir figure 2.2) :



(a) Tracé incorrect

(b) Tracé correct

FIG. 2.2: Approximation du tracé d'une ligne

1. Tout point (i, j) appartenant au segment discret est tel qu'il existe un point (x, y) du segment continu appartenant à $I_{i,j}$. Sur la figure, cela signifie que l'on s'autorise à choisir tous les pixels par lesquels le segment continu passe.
2. Le segment discret est "continu" dans le sens où tout point du segment possède au moins un point voisin sur la grille qui appartient au segment, et qu'il est possible d'aller de p_{i_0,j_0} en p_{i_1,j_1} en se déplaçant de voisin en voisin (connexité du segment)[2].
3. On veut tracer le moins de points possibles. En fait, on en veut exactement $n+1$ où $n = \max\{|i_0 - i_1|, |j_0 - j_1|\}$ qui est la distance (de Manhattan) entre ces deux pixels. On trace donc un seul point par ligne si $|i_0 - i_1| < |j_0 - j_1|$ et sinon un seul point par colonne.

1.2 Un algorithme simple

La méthode la plus simple est d'utiliser l'équation de la droite cartésienne ($y = ax + b$) pour $x \in \{i_0, ..., i_1\}$. En exprimant a et b en fonction de (i_0, j_0) et (i_1, j_1), on obtient $a = \frac{j_1 - j_0}{i_1 - i_0}$ et $b = j_0 - a.i_0$. On propose le premier algorithme trivial suivant valable dans le cas où $|a| \leq 1$:

Algorithme 2.1 (tracé de ligne (cas $|a| \leq 1$)) *La fonction **putpixel** affiche un pixel à l'écran aux coordonnées indiquées.*

```
int       i,j;
double    a,b;

a = (double)(j1 - j0) / (double)(i1 - i0);
b = j0 -a*i0;

for(i=i0;i<=i1;i++) {
    j = (int)( a*i + b );
    putpixel(i,j);
}
```

On notera que l'expression de la condition correspond au cas où la droite avance (en x) plus vite qu'elle ne monte ou descend (en y), ce qui nous donne l'assurance que la droite tracée est continue au sens des 8-voisins.

Cet algorithme est simple, répond aux critères fixés, et utilise des opérations telles que les conversions (entier $\longleftrightarrow$ flottant) et des opérations en virgule flottante.

[2] Les voisins du pixel central (noir) sont définis comme étant les 8 pixels grisés. On parle alors de 8-voisins. Ces appellations proviennent de la géométrie discrète et plus précisément de la morphologie mathématique qui s'intéresse à la topologie des ensembles discrets.

Les processeurs modernes utilisent largement le pipelining et le packing, si bien qu'il n'est plus vrai que le temps d'exécution de n multiplications est n fois plus long que l'exécution d'une seule multiplication. Afin d'intégrer la possibilité du pipelining, les documentations des microprocesseurs donnent désormais pour chaque instruction assembleur :

- **sa latence** L, à savoir le nombre de cycles requis pour l'exécution complète de l'instruction.
- **son débit** (inverse) D, à savoir le nombre de cycles à attendre avant d'exécuter une nouvelle fois la même instruction.

Par exemple, si on ne tient compte que de la durée d'exécution de l'addition ($L > D$), la table 2.1 donne des exemples de performance de codes en fonction de leurs possibilités d'effectuer du pipelining.

code	**temps d'exécution**
`for(i=0;i<n;i++) a[i] = b[i] + c[i];`	$L+(n-1).D$
`for(i=0;i<n;i++) a[i] = a[i] + 1;`	$L+(n-1).D$
`for(i=0;i<n;i++) sum = sum + a[i];`	$n.L$
`for(i=1;i<n-1;i++) a[i] = a[i-1] + a[i] + a[i+1];`	$(n-1).L$

TAB. 2.1: Exemple de codes supportant ou pas le pipelining.

Dans les deux derniers cas, l'addition précédente doit avoir été effectuée avant de passer à la suivante, ce qui empêche *tel quel* l'utilisation du pipelining.

Afin de donner des ordres de grandeur sur les écarts entre les opérations classiques, nous donnons à titre d'indication la table 2.2 pour les instructions assembleur d'un Pentium IV.

Type	Bits	Instruction	Latence	Débit
addition entière	8, 16, 32	`INC`	1	0.5
	8, 16, 32	`ADD`	0.5	0.5
	64P	`PADDQ`	2	1
	8P, 16P, 32P	`PADDB/W/D`	2	2
multiplication entière	8, 16, 32	`IMUL`	14	3
	16P, 32P, 64P	`PMULHW/LW/UDQ`	8	2
division entière	8, 16, 32	`DIV/IDIV`	56-70	23
addition flottante	32, 64	`FADD`	5	1
	32P, 64P	`ADDPS/PD`	4	2
multiplication flottante	32, 64	`FMUL`	7	2
	32P, 64P	`MULPS/PD`	6	2
division flottante	16/32/64	`FDIV/FDIVP/FIDIV`	23/38/43	23/38/43

TAB. 2.2: Débit et latence des operateurs classiques sur le Pentium IV d'Intel.

Pour résumer, cela signifie que, si le pipelining n'est pas possible, le calcul en entier reste plus rapide que le calcul flottant (d'un facteur 2.5). Mais, dès qu'il peut être

utilisé, il permet d'obtenir des performances en calcul sur les flottant **similaires** à celles sur les entiers.

On notera que sur un Pentium IV :
- la division constitue une exception notable pour laquelle ses caractéristiques en pipelining font qu'une division flottante est jusqu'à deux fois plus rapide qu'un division entière.
- les opérations de conversion entre type entier et flottant ont une latence comparable aux operateurs classiques (de 2 à 4), mais un débit comparable à celui d'une multiplication (8 à 16).

Le calcul sur des entiers étant à l'origine toujours plus rapide que les calculs en virgule flottante a conduit à la recherche d'un algorithme de calcul n'utilisant que des entiers : l'algorithme de *Bresenham*. Sur les processeurs modernes, cet algorithme reste largement compétitif à cause de sa simplicité et sa compacité.

1.3 L'algorithme de Bresenham (ou algorithme du point milieu)

On considère le cas où $0 < a < 1$ et $i_0 < i_1$. Alors, si $p_{i,j}$ est un pixel se déplaçant sur le segment discret partant de p_{i_0,j_0} et allant vers p_{i_1,j_1}, sa position suivante est soit $p_{i+1,j}$ (il avance horizontalement), soit $p_{i+1,j+1}$ (il avance en diagonale). L'algorithme de *Bresenham* est un algorithme partant du pixel p_{i_0,j_0}, et qui choisit itérativement le pixel suivant de façon à minimiser l'erreur avec le segment continu.

Reformulons maintenant le problème pour n'utiliser plus que des entiers. Soit $\delta_i = i_1 - i_0$ et $\delta_j = j_1 - j_0$. L'équation cartésienne $y = a.x + b$ peut s'écrire :

$$\begin{aligned} y &= \frac{\delta_j}{\delta_i}x + j_1 - \frac{\delta_j}{\delta_i}i_1 \\ &\Leftrightarrow \delta_j.x - \delta_i.y + (\delta_i.j_1 - \delta_j.i_1) = 0 \end{aligned}$$

Cette équation est une équation implicite de la forme $F(x, y) = 0$ **dont tous les coefficients sont des entiers**. Une caractéristique de cette équation est que pour tout point (x, y) : si $F(x, y) > 0$ alors le point est au-dessous de la droite et si $F(x, y) < 0$ alors le point est en-dessus de la droite.

Le test qui décide du pixel choisi utilise un point (appelé point milieu) défini par $(i + 1, j + \frac{1}{2})$ où (i, j) est le point du segment discret auquel on se trouve à cette étape de l'algorithme. Il se formule ainsi :
- si $F(i + 1, j + \frac{1}{2}) > 0$, alors on choisit le pixel $p_{i+1,j+1}$ (car le point milieu est en dessous du segment).
- si $F(i + 1, j + \frac{1}{2}) \leq 0$, alors on choisit le pixel $p_{i+1,j}$ (car le point milieu est au-dessus du segment).

L'évaluation de la valeur à tester $F(i + 1, j + \frac{1}{2})$ est réalisée de façon itérative en remarquant que :
- En (i_0, j_0), $F(i_0, j_0) = 0$.

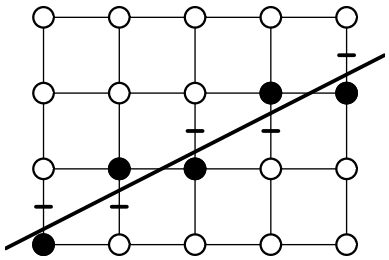


FIG. 2.3: Principe de sélection du pixel par le point milieu. L'espace représenté est l'espace continu. Les ronds correspondent aux pixels sur la grille discrète (noir=activé, blanc=éteint). Les petits traits verticaux en gras représentent la position du point milieu.

– On commence le segment en p_{i_0,j_0}, donc :

$$\begin{aligned} F(i_0+1, j_0+\frac{1}{2}) &= F(i_0,j_0) + \delta_j - \frac{1}{2}\delta_i \\ &= \delta_j - \frac{1}{2}\delta_i \end{aligned}$$

Cette valeur n'est pas entière (à cause du $\frac{1}{2}$), mais comme elle est testée par rapport à zéro, on considérera 2 fois la fonction d'erreur. Les incréments calculés ci-dessous devront également être multipliés par 2.

– Au pixel $p_{i,j}$, la valeur à tester est $F\left(i+1, j+\frac{1}{2}\right)$. On a deux cas :
  – si le pixel choisi est $p_{i+1,j+1}$, alors la valeur devient au pas suivant :

$$F\left((i+1)+1, (j+1)+\frac{1}{2}\right) = F\left(i+1, j+\frac{1}{2}\right) + \delta_j - \delta_i$$

On l'incrémente donc de $\delta_j - \delta_i$.

  – si le pixel choisi est $p_{i+1,j}$, alors la valeur devient au pas suivant :

$$F\left((i+1)+1, j+\frac{1}{2}\right) = F\left(i+1, j+\frac{1}{2}\right) + \delta_j$$

On l'incrémente donc de δ_j.

On obtient donc l'algorithme suivant :

Algorithme 2.2 (Bresenham) *L'algorithme suivant n'utilise des opérations que sur les entiers, et correspond au cas où* $0 < a < 1$.

```
int          i,j,di,dj,f;

j = j0;
di = i1 - i0;
dj = j1 - j0;
f = 2*dj - di;

for(i=i0;i<=i1;i++) {
    putpixel(i,j);
    if (f < 0) {
        j++;
        f += 2 * (dj - di);
    } else {
        f += 2 * dj;
    }
}
```

Les autres cas sont obtenus par symétrie ou miroir :

1. en faisant en sorte $i0 \leq i1$, on limite le nombre de cas à 4 dans la demi-plan $x \geq 0$.
2. si $|a| \leq 1$,
   - le cas $0 \leq a \leq 1$ est celui proposé ci-dessus.
   - dans le cas $0 > a \geq -1$, on fait un miroir en y, *i.e.* on décrémente j.
3. si $a > 1$, inverser les axes x et y.

Une accelération de l'algorithme est proposé en exercice. A noter que des algorithmes similaires existent pour les autres primitives graphiques telles que les cercles, ellipses, etc... (voir aussi la partie exercices).

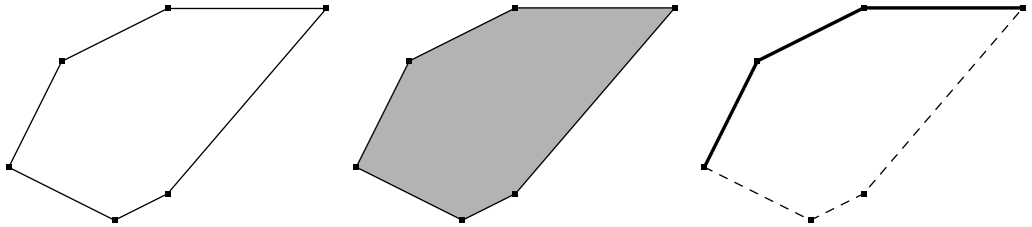
2 Remplissage

Le remplissage d'une courbe fermée discrète consiste à allumer les points de l'écran qui correspondent à la partie de l'espace discret délimitée par cette courbe (voir figure 2.4). Nous présentons le cas où la courbe est un polygone à n sommets $\{P_i\}_{i=1...n}$.

Tout d'abord, considérons le cas où le polygone est convexe[3]. L'algorithme de remplissage consiste alors à partir du point le plus à gauche et à suivre la suite des segments qui composent le polygone jusqu'au point le plus à droite (voir figure 2.4).

Algorithme 2.3 (scanline - remplissage d'un polygone convexe)

[3]Une facette est un polygone convexe



(a) Polygone convexe (b) Remplissage (c) Segments supérieur (en gras), segments inférieurs (en pointillés)

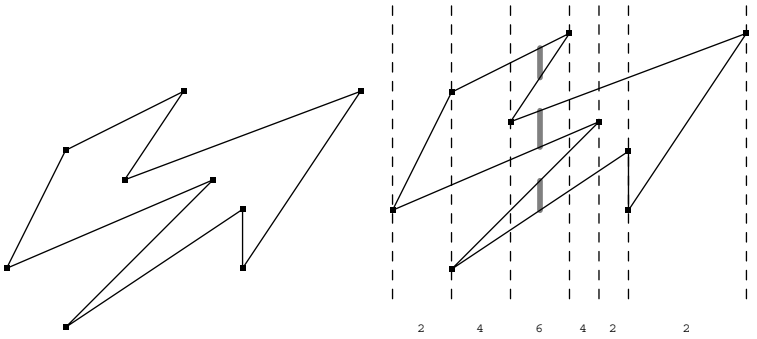
FIG. 2.4: Remplissage dans le cas d'un polygone convexe

1. ***initialisation***
   P_i = *le plus à gauche des sommets.*
   S_H *le segment montant qui part de* P_i.
   S_B *le segment descendant qui part de* P_i.
   x = *l'abscisse de* P_i.
2. ***faire***
   (a) *Mettre à jour les segments* S_H *et* S_B *(si on atteint l'extrémité droite du segment, on passe au segment suivant).*
   (b) *Calculer l'ordonnée* y_H *du point d'abscisse* x *sur le segment* S_H. *Calculer l'ordonnée* y_B *du point d'abscisse* x *sur le segment* S_B.
   (c) *Tracer le segment vertical reliant* (x, y_B) *à* (x, y_H).
   (d) *Incrémenter* x *de* 1.
3. ***tant que*** x *n'a pas atteint l'ordonnée du point le plus à droite du polygone.*

Quand le polygone n'est plus convexe, sa forme peut devenir très complexe (voir figure 2.5). Dans ce cas on ne gère plus séparément les segments inférieurs et supérieurs mais on détermine à chaque étape la liste des intersections de l'ordonnée courante avec le polygone (en comptant double l'intersection avec un sommet). En partant du bord, on trace les segments reliant les intersections paires aux intersections impaires (voir figure 2.5)

Algorithme 2.4 (scanline - remplissage d'un polygone quelconque)

1. ***initialisation***
   x =*l'ordonnée du point le plus à gauche.*
2. ***faire***
   (a) *calculer la liste des intersections avec le polygone et trier cette liste (on note* $2n_x$ *le nombre de ces intersections et* $\{(x, u_i)\}_{i=1..2n_x}$ *la liste triée)*



(a) Polygone quelconque (b) Nombre d'intersections détectées sur la ligne

FIG. 2.5: Remplissage dans le cas d'un polygone quelconque

(b) tracer les n_x segments qui relient (x, u_{2i}) à (x, u_{2i+1}) (i va de 0 à $n_x - 1$).

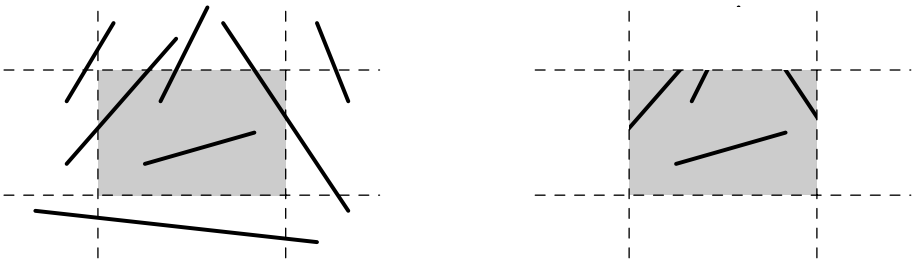
(c) Incrémenter x de 1.

3. ***tant que** x n'a pas atteint l'ordonnée du point le plus à droite du polygone.*

Une implémentation efficace de cette méthode n'effectue pas directement un calcul des intersections à chaque étape mais calcule itérativement cette liste en combinant un algorithme du type *Bresenham* et une gestion de la liste des sommets du polygone.

3 Clipping

Le clipping est l'opération destinée à restreindre un objet à sa portion présente dans une courbe fermée (voir figure 2.6). Dans cette partie, on ne s'intéresse qu'au problème du clipping d'un segment par un rectangle. Pour le résoudre, on utilise l'algorithme de *Cohen-Sutherland*.



(a) Ensemble de droites. La zone de clipping est déterminée par les droites en pointillés.

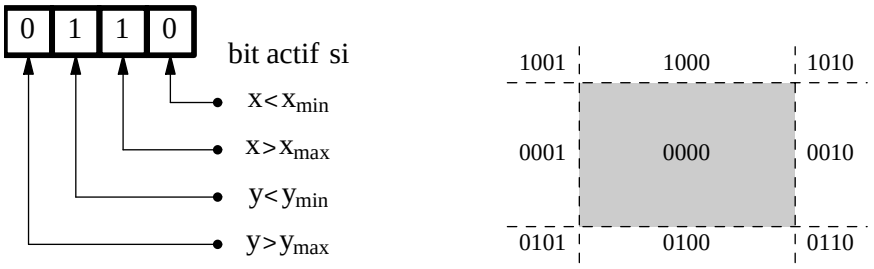
(b) Résultat

FIG. 2.6: Clipping : ce que l'on cherche à faire.

Il consiste à utiliser la position des deux points extrêmes du segment afin de déterminer rapidement s'il est possible que ce segment intersecte le rectangle. Si c'est le

cas, on tronque le segment. Puis on itère le procédé jusqu'à ce que le segment soit restreint à sa partie placée à l'intérieur du rectangle ou déclaré comme n'ayant pas d'intersection avec le rectangle.

On note $P_{\min} = (x_{\min}, y_{\min})$ et $P_{\max} = (x_{\max}, y_{\max})$, les coins extrêmes du rectangle, et P_1 et P_2 les points extrêmes du segment. La première étape est basée sur l'astuce suivante. On associe un code binaire à quatre digits à chaque extrémité du segment (notés B_1 et B_2) qui représente grossièrement la position du segment par rapport au rectangle de la façon suivante :



(a) Construction du code binaire (b) Zones associées dans le plan

FIG. 2.7: Clipping : signification du code binaire dans l'algorithme de Cohen-Sutherland

Puis, on applique les règles de décisions suivantes :

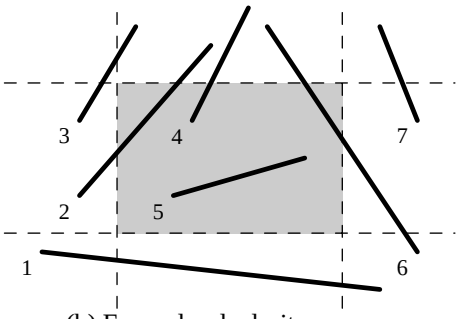
1. si $B_1 = 0$ et $B_2 = 0$ alors le segment est dans le rectangle.
2. si $B_1 = 0$ ou (exclusif) $B_2 = 0$ alors le segment intersecte le rectangle.
3. sinon ($B_1 \neq 0$ et $B_2 \neq 0$), on calcule un ET logique binaire entre B_1 et B_2.
   (a) si $B_1 \& B_2 \neq 0$ alors le segment n'intersecte pas le rectangle (car ce test revient à dire que l'un des tests $x < x_{\min}$, $x > x_{\max}$, $y < y_{\min}$ ou $y > y_{\max}$ est vrai pour les deux extrémités du segment, et donc qu'il est positionné respectivement à gauche, à droite, en bas ou en haut du rectangle).
   (b) sinon, il faut plus d'informations pour décider (voir cas 3 figure 2.8).

Dans les cas 2 et 3(b), on utilise les intersections entre le segment et les droites déterminant le rectangle (ce sont les traits pointillés de la figure 2.8). Si un bit est activé, alors le segment intersecte une de ces droites (par exemple, si le premier bit est activé, alors le segment a une intersection avec la droite $x = x_{\min}$). Cette propriété nous permet de déterminer les droites avec lesquelles le segment a une intersection. On remplace alors les points des extrémités par l'intersection entre le segment et l'une de ces droites ; puis on recommence avec le nouveau segment ainsi construit.

Exemple 3 *Sur la figure 2.8, l'algorithme s'applique comme suit. On utilise la même notation que ci-dessus, P_1 étant le point le plus proche du numéro associé au segment.*

Segment	B_1	B_2	$B_1 \& B_2$	Cas
1	0101	0110	0100	$3(a)$
2	0001	1000	0000	$3(b)$
3	0001	1000	0000	$3(b)$
4	0000	1000	0000	2
5	0000	0000	0000	1
6	0110	1000	0000	$3(b)$
7	0010	1010	0010	$3(a)$

(a) Les différents cas présents dans la figure ci-contre.



(b) Exemples de droites

FIG. 2.8: Figure associée à l'exemple 3

L'algorithme s'écrit donc de la façon suivante :

Algorithme 2.5 (Cohen-Sutherland) *l'algorithme renvoie la partie du segment incluse dans le rectangle et rien si le segment n'en fait pas partie.*

segment COHENSUTHERLAND *(rectangle [$P_{\min}, P_{\max}$], segment [P_1, P_2])*

1. *déterminer les codes binaires B_1 et B_2 de P_1 et P_2*
2. ***si*** $B_1 = 0$ *et* $B_2 = 0$ ***alors retourner*** P_1, P_2.
   sinon
   si $B_1 \& B_2 \neq 0$ ***alors retourner*** *rien.*
   sinon

   (a) ***si*** $B_1 \neq 0$ ***alors*** $P_1 =$INTERSECTION*([$P_{\min}, P_{\max}$],[P_1, P_2], B_1)*
   (b) ***si*** $B_2 \neq 0$ ***alors*** $P_2 =$INTERSECTION*([$P_{\min}, P_{\max}$],[P_2, P_1], B_2)*
   (c) ***retourner*** COHENSUTHERLAND*([$P_{\min}, P_{\max}$],[P_1, P_2])*

et la fonction Intersection par :
point INTERSECTION*(rectangle [$P_{\min}, P_{\max}$], segment [P_d, P_f], byte B,)*

1. ***si*** *le* 1^{er} *bit de* $B = 1$
   alors $P =$*intersection du segment* $[P_d, P_f]$ *avec* $x = x_{\min}$.
2. ***si*** *le* $2^{ème}$ *bit de* $B = 1$
   alors $P =$*intersection du segment* $[P_d, P_f]$ *avec* $x = x_{\max}$.
3. ***si*** $3^{ème}$ *bit de* $B = 1$
   alors $P =$*intersection du segment* $[P_d, P_f]$ *avec* $y = y_{\min}$.
4. ***si*** $4^{ème}$ *bit de* $B = 1$
   alors $P =$*intersection du segment* $[P_d, P_f]$ *avec* $y = y_{\max}$.
5. ***retourner*** P.

Cet algorithme peut se généraliser en 3D en utilisant des plans à la place de droite. Une de ses utilisations pratiques est de simplifier (après rejet trivial) la géométrie à l'extérieur du cône de vision pour ne conserver dans la chaîne de traitement que les polygones observables ou partiellement observables. La définition des plans associés au cône de vision sera détaillé dans le chapitre suivant.

4 Exercices

Exercice 6 (Algorithme de Bresenham) *Ecrire l'algorithme de Bresenham dans le cas général. On pourra utiliser la méthodologie suivante :*

- *Partitionner l'espace en fonction des paramètres de l'algorithme.*
- *Réduire le nombre de cas si possible.*
- *Écrire l'algorithme.*

Exercice 7 (Algorithme en entiers de remplissage d'une facette) *Ecrire un algorithme de remplissage d'une facette n'utilisant que des entiers. Le sens de remplissage des facettes devra être toujours le même (par exemple de gauche à droite) - ceci a pour conséquence que le sens de tracé d'un segment entre deux sommets est toujours le même. L'exercice précédent est un prérequis.*

Exercice 8 (Calcul d'intersections par une méthode paramétrique) *On rappelle que l´equation d'une droite paramétrique passant par le point P et de vecteur directeur $\mathbf{V}$ s'écrit :*

$$D(\lambda) = P + \lambda.\mathbf{V} \text{ avec } \lambda \in \mathbb{R}$$

où $D(\lambda)$ donne l'ensemble des points de la droite quand λ varie.

1. *Calculer l'intersection entre deux droites $(P_1, \mathbf{V_1})$ et $(P_2, \mathbf{V_2})$ dans $\mathbb{R}^2$.*
2. *Calculer l'intersection entre deux droites $(P_1, \mathbf{V_1})$ et $(P_2, \mathbf{V_2})$ dans $\mathbb{R}^3$. On pourra utiliser la méthode suivante :*
   (a) *Trouver un test permettant de déterminer si les deux droites appartiennent à un même plan.*
   (b) *Si les deux droites appartiennent au même plan, on utilisera la méthode suivante :*
      i. *Résoudre le problème dans le cas où $\mathbf{V_2}$ est parallèle à l'axe des z.*
      ii. *Sinon,*
         - *Construire un vecteur $\mathbf{V'_2}$ orthogonal à $\mathbf{V_2}$.*
         - *Multiplier l'équation $D_1(\lambda) = D_2(\mu)$ par $\mathbf{V'_2}$, et en déduire la valeur de λ. On distinguera les différents cas.*
3. *Calculer l'intersection entre une droite $(P_1, \mathbf{V_1})$ et un plan défini par sa normale $\mathbf{N}$ et un point P_2 y appartenant (on utilisera l'équation paramétrique de la droite*

et carthésienne du plan). A noter que cette méthode de calcul d'intersection avec une surface peut être utilisée de façon assez générale.

4. *Même question dans le cas où le plan considéré a pour équation $x = a$ (resp. $y = a$ et $z = a$ où a est une constante). On ne traitera complètement qu'un seul des cas.*
5. *Reprendre les questions précédentes en remplaçant le mot droite par segment. Un segment sera alors défini par les points de ses deux extrémités P et Q (utiliser l'équation paramètrique de la droite et la propriété que le paramètre λ doit alors vérifier certaines contraintes).*

Exercice 9 (Clipping de polygones) *Le clipping de polygone doit être traité de façon sensiblement différente du clipping de ligne vu ci-dessus. Dans ce cas, il est important que la structure du polygone (i.e. l'ordre de ses arêtes) soit conservée, et l'intèrieur du polygone peut générer de nouveaux segments (voir les exemples de la figure 2.9).*

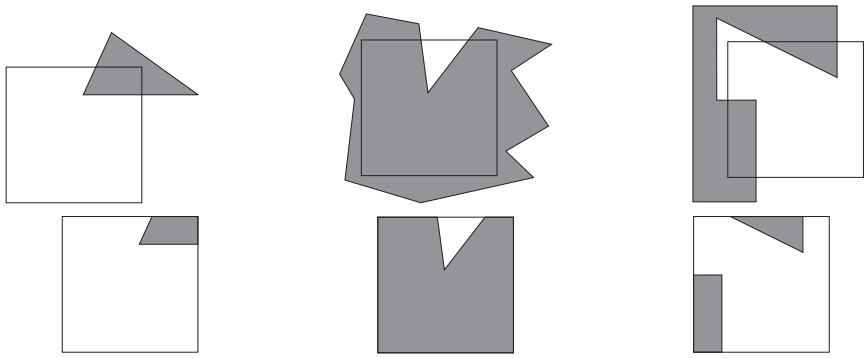


FIG. 2.9: Exemples de clipping de polygones

On se propose dans cet exercice de réécrire l'algorithme de Sutherland-Hodgman. Il est basé sur un clipping effectué bord par bord, comme le montre la figure 2.10.

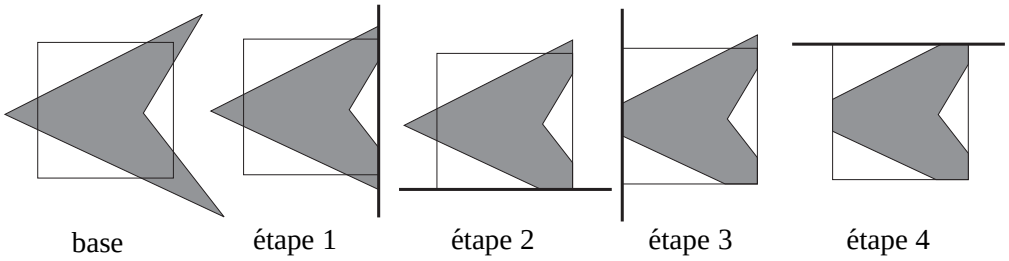
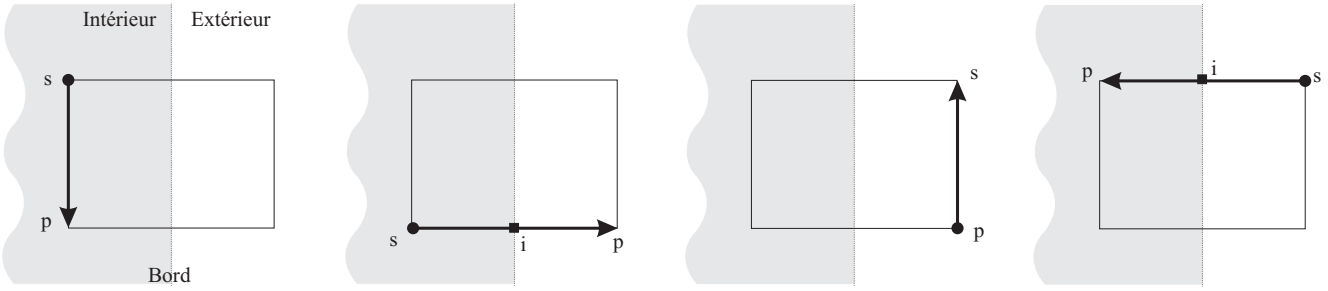


FIG. 2.10: Principe de l'algorithme de Sutherland-Hodgman

1. *Indiquer comment, pour un clipping effectué sur un seul bord, on construit un nouveau polygone basé sur ce bord (indice : considèrer les 4 cas de la figure 2.11).*



cas 1 :segment interne cas 2 :segment sortant cas 3 :segment externe cas 4 :segment entrant

FIG. 2.11: Algorithme de Sutherland-Hodgman : les quatre cas.

2. *Comment utiliser la fonction que vous avez définie à la question précédente pour effectuer un clipping complet ?*
3. *Dessiner le polygone clippé obtenu par cet algorithme si on l'applique sur le premier exemple de la figure 2.9 ? Est-ce que cela constitue un problème (par exemple pour un algorithme de remplissage de polygone) ?*

Exercice 10 *Accelération de l'algorithme de Bresenham*

On considère le tracé d'une ligne discrète entre deux points $A_1 = (i_1, j_1)$ et $A_2 = (i_2, j_2)$. On supposera que la pente du segment discret $[A_1, A_2]$ est comprise entre 0 et 1. Comme dans le cours, on nomme $F(x, y)$ la forme implicite de l'équation de la droite.

1. *Quelle est la valeur de la fonction $F(x, y)$ au point A_1 ?*
2. *Comment choisit-on le point suivant A_1 ? On donnera la valeur de la fonction $F(x, y)$ au point milieu associé à A_1.*
3. *Expliquer alors comment on met à jour la valeur de $F(x, y)$ au point milieu associé au point courant en fonction de la valeur de $F(x, y)$ au point milieu du point précédent. On traitera les deux cas.*

   On considère maintenant le segment discret $[A_2, A_1]$ (le même segment que précédemment, mais en se déplaçant de A_2 vers A_1).
4. *Quelle est la pente de ce segment ? En déduire $F(x, y)$.*
5. *Quelle est la valeur de la fonction $F(x, y)$ au point A_2 ?*
6. *Comment choisit-on le point suivant A_2 ? On donnera la valeur de la fonction $F(x, y)$ au point milieu associé à A_2. Comparer cette valeur avec celle obtenue à la question 2.*
7. *Donner dans ce cas les équations de mise à jour de la fonction $F(x, y)$ pour le point milieu dans les deux cas. Comparer les équations obtenues avec celle de la question 3.*
8. *Comment exploiter ces similitudes ?*

9. *Écrire l'algorithme de Bresenham accéléré associé. Pour simplifier, on se placera dans le cas où la pente est entre 0 et 1, et où le nombre de points entre A_1 et A_2 est pair.*

Exercice 11 *Algorithme de Bresenham dans le cas du tracé d'un cercle*

On se propose maintenant d'adapter l'algorithme de Brésenham au tracé d'un cercle centré en $(0, 0)$ et de rayon R. On ne s'intéressera qu'au tracé du quart de cercle allant du point de coordonnées $(0, -R)$ au point de coordonnées $(R, 0)$, le reste du cercle pouvant être obtenu par symétrie.

1. *Quelle est l'équation implicite $F(x, y)$ du cercle ? On rappelle qu'une équation implicite est de la forme $F(x, y) = 0$ et qu'un cercle de rayon R est l'ensemble des points situés à une distance R de son centre. On évitera la forme faisant intervenir la fonction racine carrée.*
2. *A partir de l'équation de $F(x, y)$, indiquez les zones de l'espace dans lesquelles $F(x, y) > 0$, puis celles dans lesquelles $F(x, y) < 0$.*
3. *Dans quel intervalle varie la tangente au cercle sur ce quart de cercle ? En se basant sur l'idée de l'algorithme de Bresenham, en déduire qu'il est nécessaire de décomposer encore le problème en deux.*
4. *En utilisant l'algorithme du point milieu à partir du point $(0, -R)$, donner :*
   - *La valeur d'initialisation du $F(x, y)$ au point milieu associé au point $(0, -R)$.*
   - *La façon de mettre à jour la valeur de $F(x, y)$ pour son calcul au point milieu suivant.*
5. *En déduire l'algorithme de tracé de cercle en entier par Bresenham sur ce huitième de cercle.*
6. *En déduire l'algorithme sur le quart de cercle, puis l'algorithme sur le cercle entier.*

Chapitre 3

Algorithmes en trois dimensions

Contenu du chapitre

1 Clipping 3D 47
2 Élimination des faces cachées 50
2.1 Algorithme du peintre 50
2.2 Algorithme du tampon de profondeur (Z-buffer) 52
3 Exercices 55

Nous abordons dans cette partie deux types d'algorithmes 3D :
- La base des algorithmes de clipping 3D. Le clipping est une phase préparatoire de l'affichage ou du rendu dont le but est d'éliminer le plus grand nombre de polygones possible afin de minimiser la quantité objet géométrique à manipuler lors des phases d'affichage à suivre.
- Plusieurs algorithmes d'élimination des faces cachées des polygones présents dans une scène. Ces algorithmes pourront être mixés avec les méthodes de rendu présentés au chapitre suivant.

1 Clipping 3D

En pratique, le but du clipping est de se débarrasser du plus grand nombre de polygones possibles pour ne se concentrer que sur ceux effectivement visibles par l'observateur, en restreignant éventuellement ceux qui ne sont que partiellement visible. Il s'avère que les algorithmes vus dans le cas bidimensionnel peuvent être facilement adaptés dans le cas tridimensionnel moyennant quelques astuces :
- Dans ce cas, la zone de clipping est déterminée non plus par des droites mais par un ensemble de plans. Cette zone est constituée de 5 à 6 plans :
  - Le plan $z = 0$ séparant les objets devant ou derrière l'observateur,

  - 4 plans matérialisant le cône de vision,
  - un plan (optionnel) correspondant à la limite du champs de vision au delà de laquelle les objets seront considérés comme trop loin pour pouvoir être observé. Si L est cette distance, le plan $z = L$ représente cette barrière.
- Puisque les polygones que nous considérons sont planaires, le cas 3D (intersection entre un polygone et un plan $\mathcal{P}$) peut se ramener au cas 2D (intersection entre un polygone et une droite $\mathcal{D}$) en calculant dans le plan du polygone l'intersection entre :
  - le polygone
  - la droite $\mathcal{D}$ d'intersection entre le plan du polygone et le plan $\mathcal{P}$.

  Comme le clipping doit être une opération rapide, ce calcul doit être placé dans un repère adéquat afin que les calculs à effectuer soient le plus simple possible.
- A cet effet, on remarque que la projection à un point s'écrire comme la composition de deux transformations :
  - Une transformation de **mise en perspective** déformant l'espace parallèlement au plan de projection :

$$\begin{cases} x' &= \frac{d}{z}.x \\ y' &= \frac{d}{z}.y \\ z' &= z \end{cases}$$

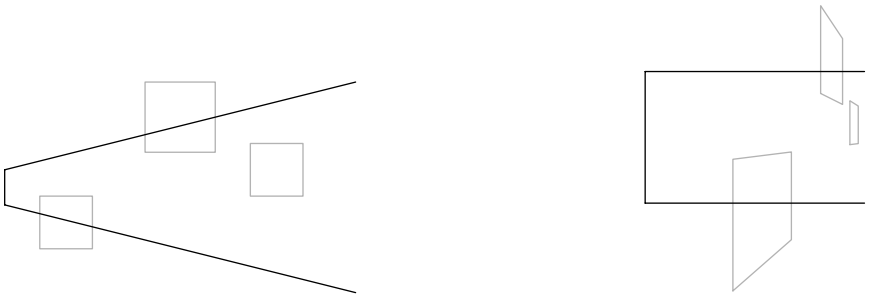
    Dans la suite de ce polycopié, les mots **mise en perspective** font références à cette opération indépendamment de l'opération de projection.
  - Une **projection** parallèle sur le plan projection $z = d$:

$$\begin{cases} x'' &= x' \\ y'' &= y' \\ z'' &= d \end{cases}$$

  Alors, si on se place dans l'espace de l'observateur **après mise en perspective**, alors le cône de vision est un parallélépipède rectangle. Les deux suites d'opérations suivantes sont alors équivalentes (voir aussi figure 3.1) :

1. Passage dans le repère de l'observateur. 2. Clipping avec le cône de vision. 3. Mise-en-perspective. 4. Projection parallèle	$\equiv$	1. Passage dans le repère de l'observateur. 2. Mise-en-perspective. 3. Clipping avec la mise en perspective du cône de vision. 4. Projection parallèle.

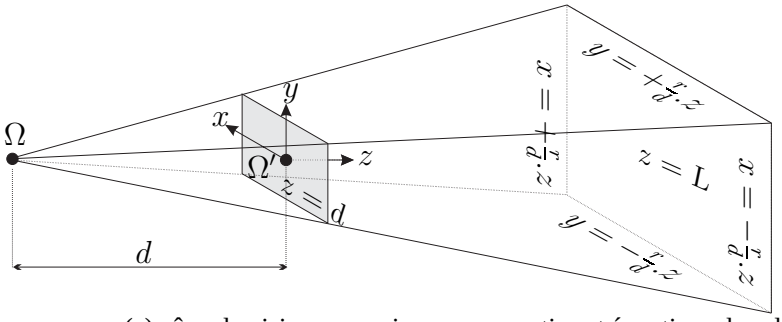
  On utilisera évidemment la deuxième méthode qui rend extrêmement simple les équations des plans définissant le cône de vision. Ces équations sont présentées sur la figure 3.2. Il devient alors simple d'adapter l'algorithme de Sutherland-Hogdman au cas du clipping 3D (cf exercice 12).



(a) Cône de vision avant mise en perspective.

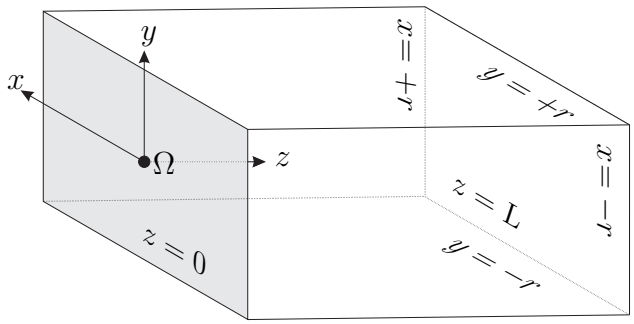
(b) Cône de vision après mise en perspective.

FIG. 3.1: Principe de l'équivalence dans le cas 2D du clipping avant ou après la mise en perspective.



$$\begin{cases} x = +\frac{r}{d}.z \\ x = -\frac{r}{d}.z \\ y = +\frac{r}{d}.z \\ y = -\frac{r}{d}.z \\ z = 0 \\ z = L \end{cases}$$

(a) cône de vision sans mise en perspective et équations des plans associés.



$$\begin{cases} x = +r \\ x = -r \\ y = +r \\ y = -r \\ z = 0 \\ z = L \end{cases}$$

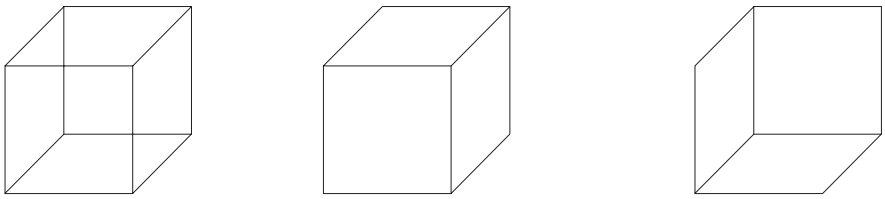
(b) cône avec mise en perspective et équations des plans associés.

FIG. 3.2: Cône de vision dans le cas 3D.

2 Élimination des faces cachées

2.1 Algorithme du peintre

L'idée de cet algorithme consiste à utiliser la technique d'un peintre pour représenter une scène : il peint tout d'abord les objets de l'arrière plan pour terminer par les objets qui sont les plus près (au premier plan).



(a) Cube en filaire (b) Faces cachées : première solution. (c) Faces cachées : deuxième solution.

FIG. 3.3: L'ambiguité des représentations en mode filaire.

Algorithmiquement parlant, on trie donc les objets en fonction de leur profondeur dans l'écran, puis on affiche ces objets en partant des objets les plus lointains jusqu'aux plus proches.

Cet algorithme travaille dans l'espace des objets (i.e. la résolution de l'écran n'entre pas en ligne de compte). En pratique, on l'applique sur des objets plats (des facettes ou des polygones dont les points sont coplanaires) afin de limiter les problèmes pouvant survenir lors du tri. Il devra donc être précédé d'une phase de facettisation des objets. Mais, il n'existe pas de relation d'ordre totale entre deux objets. Cela signifie que si A et B sont deux objets dans la scène, alors la relation "A est devant B" ne permet pas de classer de manière unique tous les objets car elle sous-entend que tous les points de A sont devant tous les points de B. Or il est facile de construire des cas où cette relation d'ordre ne sait pas trancher même dans les cas où les objets ne s'intersectent pas (voir figure 3.4). Dans des cas semblables, une solution consiste à subdiviser les objets jusqu'à ce que le problème ne se pose plus.

L'algorithme est donc le suivant :

Algorithme 3.1 (du peintre) *Affichage d'une scène avec parties cachées contenant n polygones $\{P_k\}_{k=1..n}$. On note I_{ij} l'intervalle du plan de projection correspondant au pixel (i, j), et $I(i, j)$ l'intensité de l'image à ce pixel.*

image PEINTRE*(entier n, liste_de_polygones$\{P_k\}_{k=1..n}$)*

1. *Trier les polygones $\{P_k\}$ de l'arrière vers l'avant.*
2. *Initialiser l'image de résultat I à la couleur du fond.*
3. *Pour k allant de 1 à n*

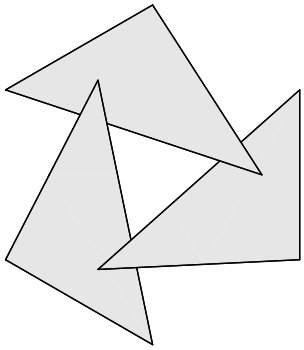


FIG. 3.4: Exemple montrant qu'il n'existe pas de relations d'ordre entre les facettes dans l'espace

   (a) *Calculer la projection P' du polygone P_k dans le plan de l'image.*
   (b) *Pour tous les intervalles I_{ij} intersectant P',*
   Affecter à $I(i,j)$ l'intensité de la partie du polygone P se projetant en $I_{ij} \cap P'$.

4. *Renvoyer I*

On remarque que si l'intensité de la partie polygone $I_{ij} \cap P'$ est calculée grâce à un rendu, alors ce rendu sera fait en chaque point de l'objet, sans savoir s'ils sont effectivement visibles ou non. Une solution consiste à dessiner les polygones en commençant par les plus proches et à ne peindre que les pixels qui n'ont pas déjà été peints. Cet algorithme s'écrit comme suit :

Algorithme 3.2 (du peintre inverse) *Affichage d'une scène avec parties cachées contenant n polygones $\{P_k\}_{k=1..n}$. On note I_{ij} l'intervalle du plan de projection correspondant au pixel (i,j), et $I(i,j)$ l'intensité de l'image à ce pixel.*

image PEINTREINVERSE*(entier n, liste_de_polygones $\{P_k\}_{k=1..n}$)*

1. *Trier les polygones $\{P_k\}$ de l'avant vers l'arrière.*
2. *Initialiser l'image de résultat I à la couleur du fond.*
3. *Initialiser le masque M des pixels déjà peints à faux : $M(i,j) =$ faux pour tout i,j.*
4. *Pour k allant de 1 à n*
   (a) *Calculer la projection P' du polygone P_k dans le plan de l'image.*
   (b) *Pour tous les intervalles I_{ij} intersectant P' et tel que $M(i,j) =$ faux,*
      i. *Affecter à $I(i,j)$ l'intensité de la partie du polygone P se projetant en $I_{ij} \cap P'$.*

ii. $M(i,j) =$ *vrai.*

5. *Renvoyer* I

Le fait qu'un tri soit utilisé pour classer les objets fait que cet algorithme ne peut pas être plus rapide que $n \log n$ où n est le nombre d'objets.

2.2 Algorithme du tampon de profondeur (Z-buffer)

L'idée de l'algorithme du Z-buffer est de ne plus du tout faire de tri pour afficher les polygones mais :

- de conserver une image de profondeur (ou Z-buffer) des pixels qui ont déjà été affichés sur l'image. Si on note Z_b cette image, $Z_b(i,j)$ contient la profondeur en Z de l'objet le plus proche qui a été affiché au pixel (i,j).
- de ne mettre à jour l'image I au pixel (i,j) que si la profondeur de l'objet se projetant au pixel (i,j) est plus proche que la valeur indiquée dans le Z-buffer $Z_b(i,j)$. Si c'est le cas, on met à jour la valeur dans le Z-buffer.

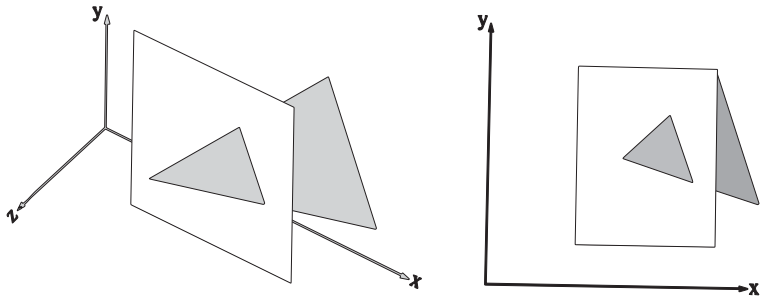
2.2.1 Algorithme de base

Sa formulation complète la suivante (voir aussi figure 3.5) :

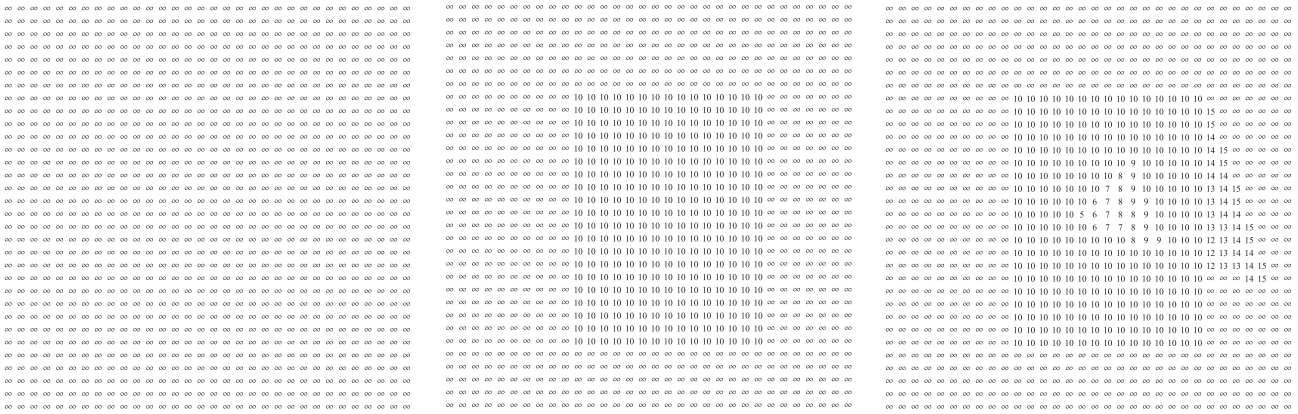
Algorithme 3.3 (Z-buffer) *Affichage d'une scène avec parties cachées contenant* n *polygones* $\{P_k\}_{k=1..n}$*. On note* I_{ij} *le carré de la grille dans le plan de projection correspondant au pixel* (i,j)*, et* $I(i,j)$ *l'intensité de la image à ce pixel.*

image ZBUFFER*(entier* n*, liste_de_polygones*$\{P_k\}_{k=1..n}$*)*

1. *Initialiser l'image de résultat* I *à la couleur du fond.*
2. *Initialiser l'image de profondeur* Z_b *à* $+\infty$ *:* $Z_b(i,j) = +\infty$ *pour tout* i*,*j*.*
3. *Pour* k *allant de* 1 *à* n
   (a) *Calculer la projection* P' *du polygone* P_k *dans le plan de l'image.*
   (b) *Pour tous les intervalles* I_{ij} *intersectant* P'*,*
      i. *Calculer la profondeur* z *de* $I_{ij} \cap P'$*.*
      ii. *Si* $z < Z_b(i,j)$ *alors*
      affecter à $I(i,j)$ *l'intensité de la partie du polygone* P *se projetant en* $I_{ij} \cap P'$*.*
      $Z_b(i,j) = z$*.*
4. *renvoyer* I*.*

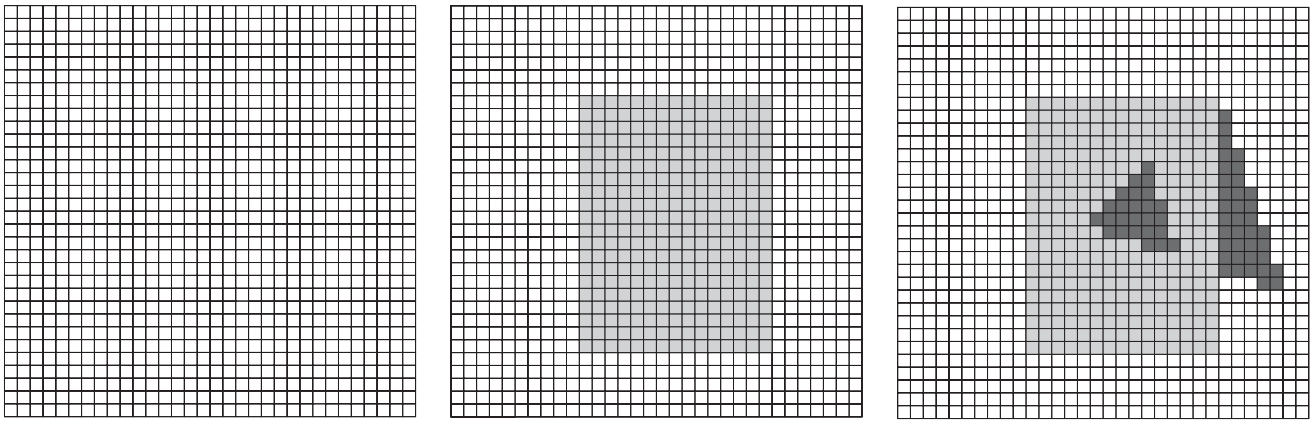


Exemple : deux polygones s'intersectant.



(1) Initialisation. (3) Ajout du rectangle. (5) Ajout du triangle.

Etape 1 : mise-à-jour de la carte de profondeur.



(1) Initialisation. (3) Ajout du rectangle. (5) Ajout du triangle.

Etape 2 : modification aux pixels où la carte de profondeur est modifiée.

FIG. 3.5: Exemple de l'algorithme du Z-buffer.

Pour l'étape $3)a)i)$, on choisira par exemple la profondeur du point du polygone le plus proche, et de ne calculer le rendu qu'en ce point.

Il faut noter au sujet de cet algorithme que :

- Il a tendance à générer des effets de crénelage (voir chapitre sur l'aliasing dans le cours du second semestre).
- Seule les transparences sans réfraction sont gérées (*i.e.* une mince couche de verre, mais pas de bouteille remplie d'eau).
- Il peut s'implémenter entièrement avec des techniques incrémentales qui permettent de n'effectuer que des opérations simples sur des entiers.
- Sa complexité est de l'ordre de n où n est le nombre d'objets.

2.2.2 Projection arrière

Dans certains cas (voir par exemple, la partie interpolation du chapitre 4), on peut avoir besoin, lorsque l'on travaille sur une facette projetée, de retrouver les informations de profondeurs en tout point de la facette. Autrement dit, on dispose de 3 sommets $S_0 = (x_0, y_0, z_0)$, $S_1 = (x_1, y_1, z_1)$, $S_2 = (x_2, y_2, z_2)$ dans l'espace géométrique de l'observateur, dont les projections dans l'espace des pixels de l'écran sont respectivement $P_0 = (i_0, j_0)$, $P_1 = (i_1, j_1)$ et $P_2 = (i_2, j_2)$. Soit P un point de la facette $\{P_0, P_1, P_2\}$, comment retrouver le point S associé dans l'espace géométrique ?

Résolution du problème dans l'espace continu

On sait que la projection dans le plan de l'écran, s'effectue avec la transformation suivante :

$$(x', y', z') = \frac{d}{z}(x, y, z)$$

On peut, par ailleurs facilement calculer l'équation du plan contenant la facette qui est de la forme (voir la partie sur les objets polygonaux dans le chapitre 1) :

$$A.x + B.y + C.z + D = 0$$

Si on connait la projection (x', y') exacte du sommet dans le plan de l'écran, l'équation du plan contenant la facette (donc A,B,C,D), les paramètres de notre caméra (donc d), et on veut retrouver la profondeur z associée à ce point. On a donc un système de 3 équations à 3 inconnues :

$$\begin{cases} x' = d.x/z \\ y' = d.y/z \\ A.x + B.y + C.z + D = 0 \end{cases}$$
$$\Rightarrow A.(x'z/d) + B.(y'z/d) + C.z + D = 0$$
$$\Rightarrow z = -\frac{D}{A.x'/d + B.y'/d + C}$$

Les valeurs de x et y sont immédiatement obtenues à partir de la valeur de z avec $x = x'z/d$ et $y = y'z/d$.

Résolution du problème en partant de l'espace des pixels

On part désormais d'un point dans l'espace des pixels de coordonnées (i, j), et on veut retrouver ses coordonnées dans l'espace géométrique.

Dans ce cas, la solution la plus simple consiste à utiliser un espace hybride construit à partir de l'espace des pixels et des coordonnées z. Autrement dit, à partir des 3 points (i_0, j_0, z_0), (i_1, j_1, z_1), (i_2, j_2, z_2), calculer l'équation du plan :

$$A'.i + B'.j + C'.z + D' = 0$$

les contenant. Une fois obtenue, la résolution est identique au cas continu.

Remarque : on notera que

- il y a une équation de plan par facette, et que cette équation dépend de la position de l'observateur.
- lorsque l'on part de l'espace des pixels, la position des somnets est ramenée au centre du pixel dans lequel il se projette. En d'autres termes, il ne s'agit plus d'exactement le même plan. Cela n'a heureusement pas d'impact direct, car un même sommet est toujours projeté de la même façon, quelque soit la facette à laquelle il appartient.

Cette méthode est fréquemment utilisée pour éviter les déformations de perspective liées à l'interpolation des profondeurs, et pour le calcul des ombres (voir chapitre 4).

3 Exercices

Exercice 12 (Algorithme de Sutherland-Hogdman en 3D (clipping en 3D)) *On cherche dans cet exercice à généraliser l'algorithme de Sutherland-Hogdman de clipping de polygone proposé dans le chapitre 2.*

1. *Proposer une méthode rapide de calcul d'intersection entre deux plans quelconques.*
2. *Même question si l'un des deux plans a pour équation $x = a$ (resp. $y = a$, $z = a$) où a est une constante.*
3. *Proposer une adaptation tridimensionnelle de l'algorithme de Sutherland-Hogdman.*
4. *Quelle est alors la chaîne de traitements nécessaires permettant d'obtenir l'ensemble des polygones visibles par l'observateur à partir de la liste brute des polygones dans le repère global ?*

Exercice 13 (Pratique du z-buffer) *On considère la figure géométrique 2D présenté en 3.6. On définit l'axe de profondeur comme étant l'axe vertical. Le but de cet exercice est d'appliquer l'algorithme du z-buffer en utilisant comme plan de projection l'axe horizontal. Les droites verticales séparent les portions d'espace visibles par chaque pixel. Les droites horizontales sont disposés par pas de 1 sur l'axe z.*

1. *Calculer le z-buffer associé à chacun des polygones 1, 2 et 3.*

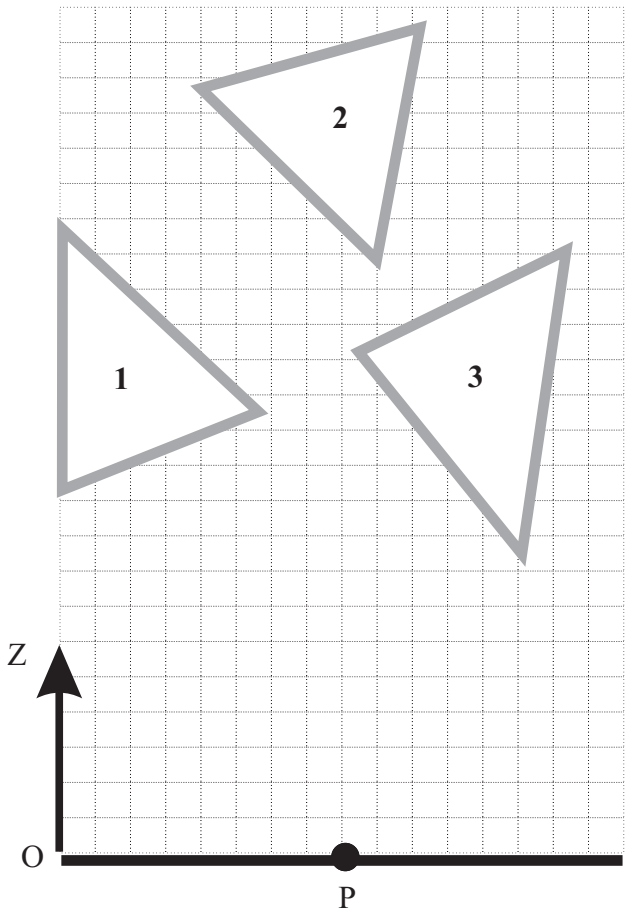


FIG. 3.6: pratique du z-buffer de l'exercice 13.

2. *Appliquer l'algorithme du z-buffer pour la figure complète. On calculera l'état du z-buffer à chaque étape :*
   - *Initialisation*
   - *Ajout du polygone* 1.
   - *Ajout du polygone* 2.
   - *Ajout du polygone* 3.
3. *Même question en inversant l'ordre des polygones.*

Exercice 14 (Algorithme de face cachée utilisant l'orientation des facettes) *Dans le premier chapitre, nous avons proposé d´orienter les facettes en indiquant que cette orientation pouvait être utilisée afin de déterminer si un polygone est visible ou non depuis la position de l'observateur.*

1. *Rappeler le principe du test permettant de déterminer la visibilité de la facette.*
2. ***Objet unique***
   (a) *Pour quels types d'objets ce principe peut-il être utilisé comme base d'algorithme de faces cachées ?*
   (b) *Écrire l'algorithme associé ?*
3. ***Objets multiples***
   (a) *Dans quels cas ce principe peut-il être utilisé pour afficher un ensemble d'objets ?*
   (b) *Proposer un algorithme (indice : utiliser des sphères englobantes et des listes de priorité d'affichage).*

Exercice 15 (Bresenham pour le calcul du z sur un polygone) *On recherche une adaptation de l'algorithme de Bresenham destinée à calculer incrémentalement la profondeur entière k en chaque point entier (i, j) de la projection de sa face sur l'écran. On rappelle que l'équation implicite du plan contenant le polygone est de la forme :*

$$a.i + b.j + c.k + d = 0$$

où a, b, c et d sont des entiers relatifs. On suppose que le remplissage des valeurs de profondeur du polygone se fait ligne par ligne par un algorithme de type scanline.

1. *Passage à la ligne suivante*
   Le passage à la ligne suivante s'effectue sur une des arêtes du polygone que nous formalisons comme suit. Soit deux points $P_0 = (i_0, j_0, k_0)$ et $P_1 = (i_1, j_1, k_1)$ les coordonnées entières tridimensionnelles de deux points. Soit $(\Delta_i(p), \Delta_j(p))_{p=1..n}$ les incréments correspondant au tracé du segment de (i_0, j_0) à (i_1, j_1) obtenu par l'algorithme de Bresenham (avec $n = \max\{|i_0 - i_1|, |j_0 - j_1|\}$). Le $l^{ième}$ point de ce segment est obtenu par :

$$(i_0 + \sum_{p=1}^{l} \Delta_i(p), j_0 + \sum_{p=1}^{l} \Delta_j(p))$$

(a) *Conditions initiales*
- *Quelle est la première valeur de z ?*
- *Quelle est la valeur initiale de la fonction d'erreur ?*

(b) *Test incrémental*
- *Quel est le test entier qu'il faut réaliser pour choisir la valeur suivante de z ?*
- *Une fois la valeur choisie, comment mettre à jour la fonction d'erreur ?*

(c) *Écrire l'algorithme correspondant.*

2. *Déplacement sur la même ligne*
Nous sommes maintenant dans le cas où nous remplissons une des lignes du polygone en allant d'un point (i_0, j_0, k_0) à un point (i_0, j_1, k_1). La valeur de la coordonnée i est fixe (le trait est horizontal) ; la valeur de la coordonnée j va de j_0 à j_1 (on supposera $j_0 < j_1$) ; la valeur de la coordonnée k est à calculer de manière incrémentale en fonction de l'équation du plan.

(a) *Conditions initiales*
- *Quelle est la première valeur de z ?*
- *Quelle est la valeur initiale de la fonction d'erreur ?*

(b) *Test incrémental*
- *Quel est le test entier qu'il faut réaliser pour choisir la valeur suivante de z ?*
- *Une fois la valeur choisie, comment mettre à jour la fonction d'erreur ?*

(c) *Écrire l'algorithme correspondant.*

Chapitre 4

Rendu élémentaire

Contenu du chapitre

1 Rendu 60
1.1 Sources de lumière 60
1.2 Modèle élémentaire d'illumination 61
1.3 Ombres 66
2 Cas particulier des polygones 69
2.1 Rendu des surfaces polygonales 69
2.2 Ombrage de Gouraud 70
2.3 Ombrage de Phong 71
2.4 Problème avec ces modèles 73
2.5 Calcul des normales 76
3 Exercices 76

Dans ce chapitre, nous présentons une méthode de rendu élémentaire pour les polygones afin de leur donner un aspect réaliste. Nous aborderons également les problèmes spécifiques liés à l'implémentation de ces méthodes sur des polygones.

Mise en garde

L'opération de **mise en perspective** change les propriétés géométriques de la scène. Par conséquent, pour l'ensemble des calculs de rendu qui sont présenté dans ce chapitre, il est impératif de les effectuer avec la géométrie de la scène avant **mise en perspective**. Ceci peut s'effectuer simplement en disposant d'une double liste de points.

1 Rendu

Afin de simplifier la présentation et de la limiter à l'ensemble des connaissances nécessaires seulement à la programmation du rendu d'ensemble de polygones, nous ne décrirons pas la physique à la base des équations présentées. Ces différents aspects seront abordés dans la deuxième partie du cours. Notamment, nous ne parlerons pas :

- Des modèles de source de lumière avancée.
- De la gestion de la couleur.
- De la gestion d'effets optiques tels que la transparence et les reflets.
- De l'habillage des objets (texture et bump-mapping).
- Du traitement de l'aliasing.

Les éléments présentés permettront d'obtenir des rendus de luminance (image en niveaux de gris), en gérant la position des sources de lumière, les objets mats, lisses et brillants, ainsi que les ombrages.

1.1 Sources de lumière

Nous ne considérons que deux type de sources de lumière :

- la lumière ambiante.
  C'est une lumière diffuse (non directionnelle) dont il n'est pas possible de déterminer la source. Tout objet de la scène est plongé dans cette lumière.
- les sources ponctuelles.
  Une source de lumière ponctuelle est une lumière dont les rayons lumineux sont émis à partir d'un seul point dans toutes les directions. Ce modèle approxime bien des sources de lumière localisées comme une ampoule électrique. Un point P d'un objet est éclairé par une source ponctuelle situé en A s'il existe une segment de droite reliant A à P sans rencontrer d'obstacle.
  Par la suite, on notera $\mathbf{L}$ le vecteur unitaire pointant dans la direction de la source de lumière ponctuelle (*i.e.* en P, la direction de A - donc le vecteur $\overrightarrow{PA}$ renormalisé).

L'une des caractéristiques principales d'une source lumineuse est son intensité (c.a.d. l'éclat avec lequel elle éclaire la scène), également appelée luminance. En général, on notera :

- I_a l'intensité lumineuse de la lumière ambiante.
- I_p l'intensité lumineuse de la source ponctuelle (s'il n'y en a qu'une) et A sa position.
- $I_p^{(i)}$ l'intensité lumineuse de la $i^{\text{ième}}$ source ponctuelle, et $A^{(i)}$ sa position.

Les valeurs que peut prendre I sont conventionnellement codées de la manière suivante :

- une intensité I égale à 0 correspond à une absence de lumière (bref, le noir).
- une intensité I égale à 255 correspond à l'intensité maximale (donc, du blanc).
- toute intensité intermédiaire correspond à un niveau de gris.

– une intensité supérieure est seuillée à 255.

Nous abordons maintenant un modèle élémentaire d'interaction entre la lumière et une surface.

1.2 Modèle élémentaire d'illumination

Cette partie a pour but la description du calcul de la quantité de lumière que reçoit un point d'une surface dans une scène.

1.2.1 Présentation

Le modèle le plus simple à mettre en oeuvre est le modèle de Phong. Ce modèle est basé sur l'hypothèse que l'intensité en un point peut se décomposer comme la somme de l'intensité ambiante, diffuse et spéculaire calculée en ce point. En un point P, l'intensité s'exprime de la façon suivante :

$$I(P) = \underbrace{k_a(P).I_a}_{\substack{\text{contribution de la} \\ \text{lumière ambiante}}} + \sum_{i=1}^{p} S_i(P, A^{(i)}).\underbrace{I_p^{(i)}.\left\{k_d(P).\cos\theta(P, A^{(i)}) + k_s(P).\left(\cos^+\alpha(P, A^{(i)})\right)^{n(P)}\right\}}_{\text{contribution de la } i^{\text{ème}} \text{ source de lumière ponctuelle}}$$

Transversalement, l'intensité totale au point est la somme des intensités en provenance des différentes sources éclairant l'objet. Les notations entre parenthèse indiquent les dépendances ; $k(P)$ indique que le paramètre k ne dépend que de la position du point P ; $\theta(P, A^{(i)})$ indique que le paramètre θ dépend à la fois de la position du point P et de la $i^{\text{ème}}$ source de lumière. Nous décrivons maintenant chacun de ces termes :

– Contribution de la lumière ambiante.
  Le paramètre $k_a(P)$ est le coefficient de réflexion ambiante de l'objet auquel appartient le point P. Sa valeur est comprise entre 0 et 1 ($0 \leq k_a(P) \leq 1$), et indique la capacité de l'objet à réfléchir la lumière ambiante. Le terme $k_a(P).I_a$ donne donc la proportion de lumière ambiante réfléchie par l'objet.

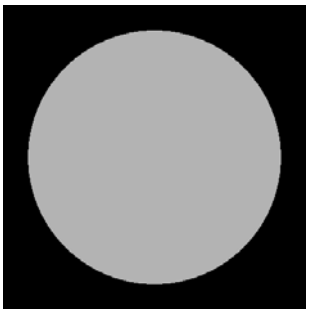


FIG. 4.1: Lumière ambiante appliquée sur une sphère

– Le paramètre $S_i(P, A^{(i)})$ de visibilité de la $i^{\text{ème}}$ source.
Ce paramètre est égal à 1 si la $i^{\text{ème}}$ source est visible depuis le point P considéré et à 0 sinon. La section suivante sur le calcul des ombre permet de connaître la valeur de ce paramètre.
Si on ne souhaite pas gérer les ombres dans une scène, on fixera $S_i(P, A^{(i)}) = 1$ pour toute source et tout point P.
– Contribution de la $i^{\text{ème}}$ source de lumière ponctuelle.
Ce terme modèlise l'interaction entre une source de lumière ponctuelle et la surface. Il se décompose en deux termes distincts :

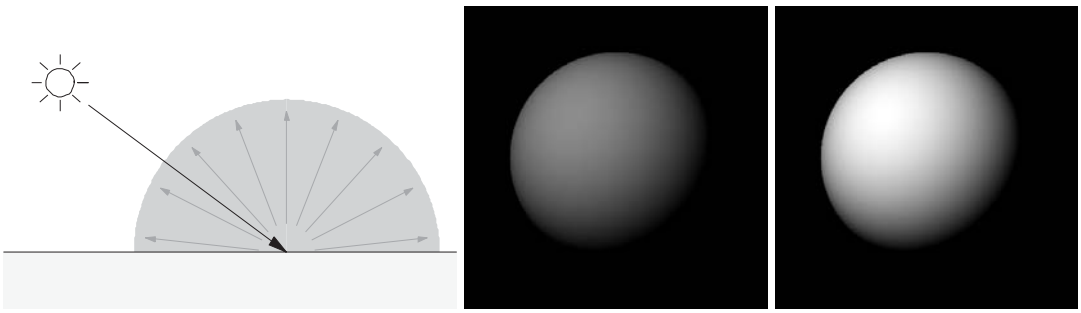
$$\underbrace{I_p^{(i)}.k_d(P).\cos\theta(P, A^{(i)})}_{\text{réflexion diffuse}} + \underbrace{I_p^{(i)}.k_s(P).\left(\cos^+\alpha(P, A^{(i)})\right)^{n(P)}}_{\text{réflexion spéculaire}}$$

– La réflexion diffuse
Ce terme est celui qui joue le rôle le plus important pour un objet terne ou mat. L'idée intuitive derrière ce terme est qu'un objet réfléchit plus de lumière aux endroits où sa normale est orientée vers la source de lumière. La signification des différents paramètres est la suivante :

$k_d(P)$ est le coefficient de réflexion diffuse. Sa valeur est comprise entre 0 et 1 ($0 \leq k_d(P) \leq 1$), et indique la capacité de l'objet à réfléchir la lumière diffuse.

$\theta(P, A^{(i)})$ est l'angle, au point P, entre la direction de la source lumineuse (*i.e.* le vecteur $\overrightarrow{PA^{(i)}}$) et la normale $\mathbf{N}(B)$ à la surface au point P. L'utilisation du cosinus de cet angle permet d'assurer le comportement recherché.

Dans le cas où le cosinus de l'angle est négatif, donc si θ est supérieur à $\pi/2$, alors l'orientation des polygones permet de déduire immédiatement que cette source n'est pas visible depuis ce point (voir section 4), et qu'elle a ni contribution diffuse ou spéculaire en ce point.



(a) Principe (b) $k_d = 0.55$ (c) $k_d = 1.0$

FIG. 4.2: Principe et effets d'une réflexion diffuse sur une sphère.

Par conséquent, le terme $k_d(P).\cos\theta(P, A^{(i)})$ donne la proportion de lumière renvoyée dans toutes les directions par une facette orientée en direction de la $i^{\text{ème}}$ source de lumière. Le terme complet :

$$I_p^{(i)}.k_d(P).\cos\theta(P, A^{(i)})$$

donne l'intensité de lumière correspondante. En pratique, c'est ce terme qui donne sa "couleur" à l'objet.
On en déduit les propriétés de la réflexion diffuse :
- Elle ne dépend pas de la position de l'observateur (la réflexion diffuse dans toutes les directions).
- Elle dépend seulement de l'angle entre la normale à la surface et la direction de la source, ainsi qu'évidemment de l'intensité de la source.

– La réflexion spéculaire
Cette réflexion est celle observée à la surface des objets lisses ou brillants sous forme d'une tache brillante plus ou moins étendue (voir figure 4.3). On observe que ce phénomène dépend de la position de l'observateur, et indique une direction de réflexion privilégiée. La signification des différents paramètres est la suivante :

$k_s(P)$ est le coefficient de réflexion spéculaire. Sa valeur est comprise entre 0 et 1 ($0 \leq k_d(P) \leq 1$), et indique la capacité de l'objet à réfléchir la lumière.

$\alpha(P, A^{(i)})$ est l'angle entre la direction de réflexion privilégiée de la source lumineuse au point P (voir figure 4.4) et la direction de l'observateur.

$n(P)$ est l'exposant de réflexion spéculaire. Cette valeur est positive. Pour $n(P) = +\infty$ (en théorie) correspond à un réflecteur parfait (un miroir). On choisit généralement $1 \leq n(P) \leq 200$.

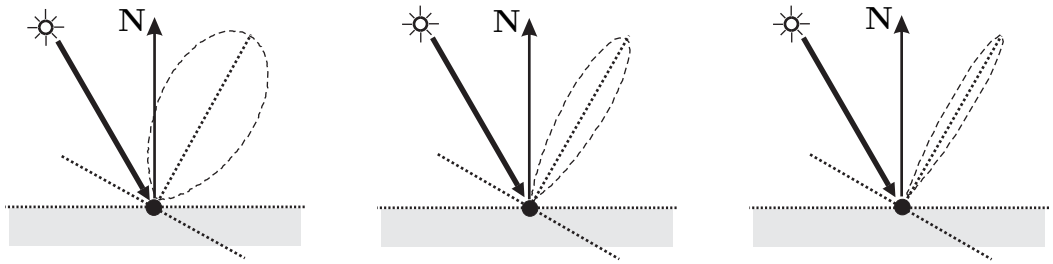
A noter que ce terme ne gère par la réflexion d'un objet sur un autre.
Par conséquent, le terme $k_s(P).\cos^{n(P)}\alpha(P, A^{(i)})$ donne la proportion de lumière renvoyée au point P par la $i^{\text{ème}}$ source de lumière, principalement dans la direction de réflexion privilégiée. Le terme complet :

$$I_p^{(i)}.k_s(P).\left[\cos^{n(P)}\alpha(P, A^{(i)})\right]^+$$

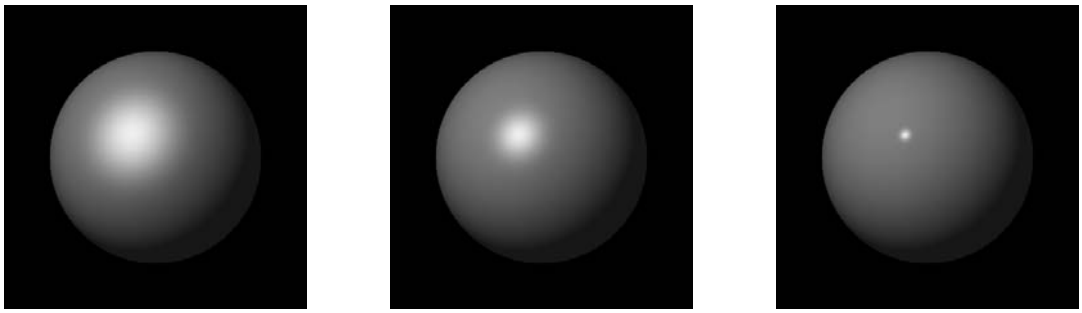
est l'intensité de lumière correspondante, et où la notation $\cos^+\alpha$ a pour sens (cf figure 4.3) :

$$\cos^+\alpha = \begin{cases} \cos\alpha & \text{si } \cos\alpha \text{ est positif} \\ 0 & \text{sinon} \end{cases}$$

On en déduit les propriétés de la réflexion spéculaire :
- Elle dépend de la position de l'observateur et est plus intense dans la direction de réflexion privilégiée.



(a) Lobe spéculaire pour $n = 5$. (b) Lobe spéculaire pour $n = 30$. (c) Lobe spéculaire pour $n = 100$.



(d) $k_s = 0.5$ et $n = 3$ (e) $k_s = 0.5$ et $n = 10$ (f) $k_s = 0.5$ et $n = 200$

FIG. 4.3: Principe et effets d'une réflexion spéculaire sur une sphère avec $k_a = 0.1$ et $k_d = 0.5$

– Ce terme ne donne pas de couleur à l'objet. La tache brillante observée est de la couleur de la source lumineuse (son reflet).

Dans le cadre d'un modèle plus simple, ce terme peut être omis. Il ne sera alors pas possible de modéliser des objets brillants.

Le signe $\sum$ permet d'ajouter l'ensemble des contributions lumineuses de toutes les sources agissant au point P. Il est important de bien comprendre la provenance et le rôle de ces différents paramètres :

– Les paramètres caractéristiques de l'objet :

  k_a le coefficient de réflexion ambiant.

  k_d le coefficient de réflexion diffus.

  k_s le coefficient de réflexion spéculaire.

  n l'exposant de réflexion spéculaire.

  A noter que si intuitivement, on pourrait écrire $k_d + k_s = 1$ (l'énergie en provenance de la source est partagée entre les réflexions diffuse et spéculaire). Cette contrainte souvent citée n'a pas besoin d'être strictement respectée dans le modèle de Phong car celui-ci n'est pas normalisé.

– Les paramètres caractéristiques de la source :

  $I_p^{(i)}$ l'intensité de la source.

  $A^{(i)}$ sa position.

– Les paramètres géométrique à recalculer en chaque point de l'objet.

S_i le paramètre d'occultation de la source.

θ l'angle entre la normale au point et la direction de la source.

α l'angle entre la direction de réflexion privilégiée et la direction de l'observateur.

1.2.2 Calcul des vecteurs et des angles du modèle

Rappel : calcul de l'angle entre deux vecteurs On rappelle que si $\mathbf{V}_1$ et $\mathbf{V}_2$ sont deux vecteurs, et θ l'angle entre ces deux vecteurs, alors $\mathbf{V}_1.\mathbf{V}_2 = |\mathbf{V}_1|\,.\,|\mathbf{V}_2|\cos\theta$. Ce qui signifie que :

$$\cos\theta = \frac{\mathbf{V}_1.\mathbf{V}_2}{|\mathbf{V}_1|\,.\,|\mathbf{V}_2|}$$

Or, si les vecteurs $\mathbf{V}_1$ et $\mathbf{V}_2$ sont normalisés, alors $|\mathbf{V}_1| = |\mathbf{V}_2| = |\mathbf{V}_1|\,.\,|\mathbf{V}_2| = 1$, et par conséquent :

$$\cos\theta = \mathbf{V}_1.\mathbf{V}_2$$

Propriétés vérifiées par l'intensité L'intensité $I(P)$ est toujours positive. Par ailleurs, les termes suivants dans l'équation d'intensité vérifie :

$$\begin{array}{rcccl} 0 & \leq & k_a & \leq & 1 \\ 0 & \leq & k_d.\cos\theta & \leq & 1 \\ 0 & \leq & k_s.\cos^n\alpha & \leq & 1 \end{array}$$

Ces contraintes peuvent être utilisées afin de trouver l'origine d'un problème lors de l'implémentation du calcul de l'intensité.

Calcul du vecteur réfléchi On dispose de deux vecteurs : la normale $\mathbf{N}$ et la direction de la source de lumière $\mathbf{L}$. On désire obtenir le vecteur réfléchi $\mathbf{R}$ qui est le miroir de $\mathbf{L}$ par rapport à $\mathbf{N}$ (voir la figure 4.4 ci-dessous). On note θ l'angle entre $\mathbf{N}$ et $\mathbf{L}$. On suppose que les vecteurs sont normalisés (i.e. de norme 1).

La projection de $\mathbf{L}$ sur $\mathbf{N}$ est $\mathbf{N}\cos\theta$ (car $|\mathbf{N}| = |\mathbf{L}|$) ; et d'après la section précédente $\cos\theta = \mathbf{N}.\mathbf{L}$. Soit $\mathbf{S}$ le vecteur défini comme sur la figure. Alors, on a les relations suivantes :

$$\left\{ \begin{array}{l} \mathbf{L} + \mathbf{S} = \mathbf{N}\cos\theta \\ \mathbf{R} = \mathbf{N}\cos\theta + \mathbf{S} \\ \cos\theta = \mathbf{N}.\mathbf{L} \end{array} \right.$$

Par conséquent,

$$\begin{array}{rcl} \mathbf{R} & = & 2\mathbf{N}.\cos\theta - \mathbf{L} \\ & = & 2\mathbf{N}.(\mathbf{N}.\mathbf{L}) - \mathbf{L} \end{array}$$

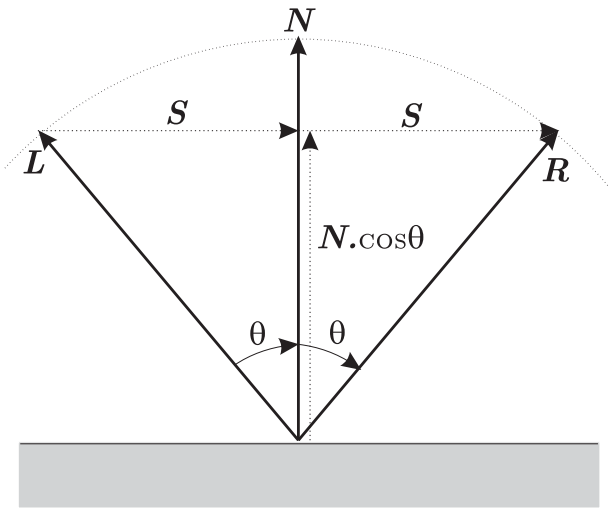


FIG. 4.4: Méthode de calcul du vecteur réfléchi.

Calcul des angles dans le modèle de Phong

– Calcul de $\cos\theta$
θ est l'angle entre la normale $\mathbf{N}$ et la direction $\mathbf{L}$ de la source. On a alors :

$$\cos\theta = \mathbf{N}.\mathbf{L}$$

– Calcul de $\cos\alpha$
α est l'angle entre le rayon réfléchi $\mathbf{R}$ et la direction de l'observateur $\mathbf{V}$. Pour calculer le rayon réfléchi, nous avons besoin de la normale $\mathbf{N}$ et la direction $\mathbf{L}$ de la source. On a donc :

$$\begin{cases} \mathbf{R} = 2\left(\mathbf{N}.\mathbf{L}\right)\mathbf{N} - \mathbf{L} \\ \cos\alpha = \mathbf{R}.\mathbf{V} \end{cases}$$

Par conséquent,

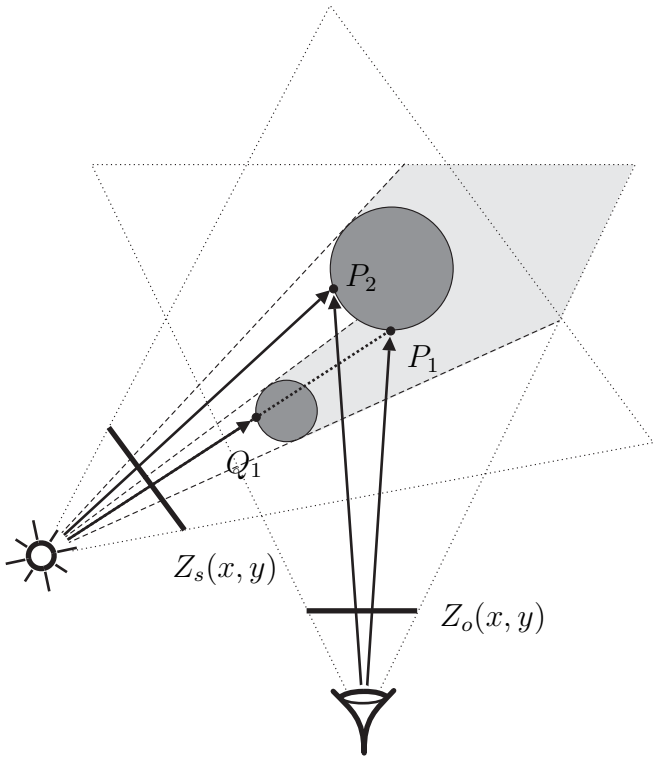
$$\cos\alpha = \Big(2\left(\mathbf{N}.\mathbf{L}\right)\mathbf{N} - \mathbf{L}\Big).\mathbf{V}$$

1.3 Ombres

Il n'y a pas de différences essentielles entre les algorithmes de suppression des parties cachées et les algorithmes de calcul des ombres portées :

– Les algorithmes de suppression des parties cachées déterminent quelles sont les surfaces visibles depuis la position de l'observateur.
– Les algorithmes de calcul des ombres portées déterminent quelles sont les surfaces **non visibles depuis la position des sources de lumière.**

Ces deux types d'algorithme sont donc duaux. On notera que si la scène ne change pas, les parties visibles par la source de lumière ne changent pas non plus. Si l'algorithme d'ombrage le permet, il est donc possible de réutiliser les calculs d'ombrage si l'observateur se déplace dans la scène.



Initialisation : construire le z-buffer $Z_s(x,z)$ associé à la source.

Critère de décision : lors du rendu, un point P est éclairé par la source si la profondeur de ce point calculée dans le repère de la source est la même que celle stockée dans le z-buffer de la source.

Exemple :

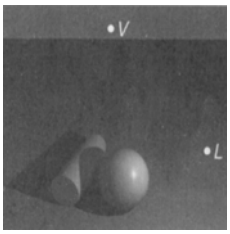
- Pour le point P_1, si ses coordonnées sont (x'_1, y'_1, z'_1) dans le repère de la source, alors z'_1 est strictement supérieur à $Z_s(x'_1, y'_1)$, nous informant qu´un autre point plus proche est visible par la source aux coordonnées (x'_1, y'_1) du z-buffer (en l'occurence Q_1 dont les coordonnées dans le repère de la source sont (x'_1, y'_1, z'_1) aux erreurs d'arrondi près). Donc, le point P_1 est dans l'ombre.
- Pour le point P_2, si ses coordonnées sont (x'_2, y'_2, z'_2) dans le repère de la source, alors z'_2 est égal à $Z_s(x'_2, y'_2)$, car P_2 est effectivement le premier point visible dans le pixel (x'_2, y'_2). Donc le point P_2 est éclairé par la source.

FIG. 4.5: Principe de calcul des ombres avec Z-buffer.

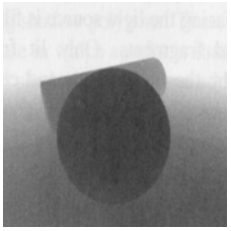
L'algorithme présenté dans cette section n'utilise qu'une seule source de lumière, mais il peut être aisément étendu à plusieurs sources. On ne considère également que des sources ponctuelles.

On présente un méthode de génération des ombres en deux passes basé sur un algorithme du Z-buffer. Le principe est le suivant :

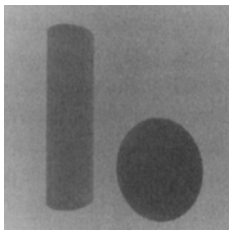
- Calculer l'image de profondeur Z_s de la scène vue depuis la source de lumière.
- Calculer le Z-buffer de la scène vu depuis l'observateur avec les modifications suivantes : pour le pixel (i, j) considéré, on note Q le point visible dans ce pixel (de coordonnées (x_Q, y_Q, z_Q) dans le repère de l'observateur). Q est donc, dans le pixel, le point qui a la plus petite profondeur z_Q (étape $2(b)$ de l'algorithme du Z-buffer) :
  - On calcule les coordonnés de Q dans le repère de la source (on note le point transformé $Q' = (x_{Q'}, y_{Q'}, z_{Q'})$).
  - Si $Z'_Q > Z_s(x_{Q'}, y_{Q'})$ (la profondeur de Q dans le repère de la source est supérieure à la valeur présente dans le Z-buffer de la source) alors il existe un objet entre la source de lumière et P. Le rendu du point est alors effectué sans la source de lumière (i.e. $S_i = 0$).
    sinon le point est éclairé et son rendu est effectué avec la source de lumière.



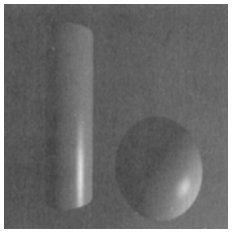
(a) Présentation de la scène



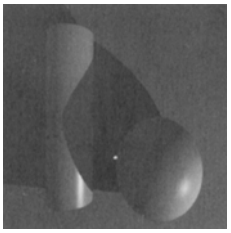
(b) Tampon de profondeur pour la source lumineuse



(c) Tampon de profondeur pour l'observateur



(d) Rendu sans les ombres



(e) Rendu avec les ombres

FIG. 4.6: Calcul des ombres : Z-buffer de l'observateur, résultat du rendu.

Cette méthode soulève deux problèmes :

- **Les erreurs d'arrondi :** le test de comparaison de profondeur dans le repère de la source doit accepter une certaine tolérance, car le point dont le z est stocké dans le z-buffer de la source n'est certainement pas celui qui est considéré par l'observateur ; même s'il se projète dans le pixel du z-buffer de la source.

– **La résolution du z-buffer de la source :** plus il est petit, plus la tolérance doit être importante, et plus elle est succeptible de générer des erreurs (surtout si ce z est stocké en entier). Typiquement, des tailles de 512x512 ou 1024x1024 sont utilisées.

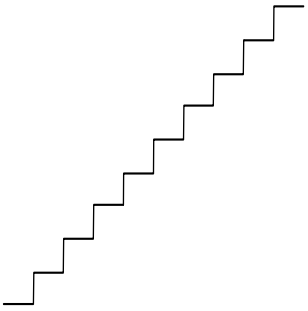
2 Cas particulier des polygones

On s'intéresse dans cette section à des techniques rapides destinées à générer un ombrage sur un polygone à partir de la connaissance de l'ombrage en quelques points de celui-ci. Tous les points du polygone sont **coplanaires** par hypothèse.

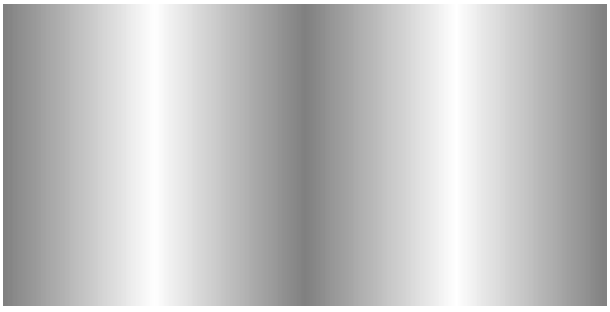
2.1 Rendu des surfaces polygonales

Dans le cas où la surface d'un objet lisse est définie par un ensemble de polygones (première approximation), il est souhaitable que l'illumination de cet objet approximé produise un objet lisse.

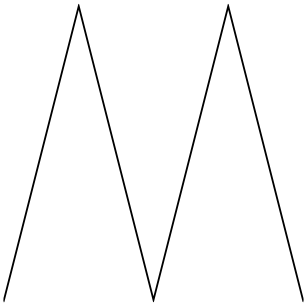
(a) Exemple 1 : Accentuation des transitions



(b) Niveaux de gris pour l'exemple 1



(c) Exemple 2 : Inhibition latérale



(d) Niveaux de gris pour l'exemple 2

FIG. 4.7: Exemples d'effets Mach

Les approches suivantes augmentent généralement l'impression de facettisation des objets à cause des effets d'inhibition latérale des récepteurs de l'oeil humain. Cet effet

d'inhibition (dit effet Mach) accentue les discontinuités et les changements de variations comme le montre la figure 4.7. Ces approches sont :

– Ombrage constant
  Il s'agit du modèle le plus simple d'ombrage. Il est basé sur les constatations suivantes.

  1. la taille du polygone est généralement petite devant sa distance à la source lumineuse.
  2. l'observateur est généralement situé loin du polygone.
  3. la normale à un polygone est constante sur tout le polygone.
  4. la couleur d'un polygone est généralement uniforme.

  Ceci implique que les produits scalaires $\mathbf{N}.\mathbf{L}$ et $\mathbf{N}.\mathbf{V}$ sont quasiment constants sur le polygone. On se trouve dans le cas de la figure 4.7(a).
– Ombrage calculé en chaque point du polygone.
  On se trouve dans le cas de la figure 4.7(b) car la normale est constante en chaque point du polygone et peut changer brusquement d'orientation sur un polygone voisin.

Des techniques plus élaborées (dites ombrages de Gouraud et de Phong) permettent d'obtenir de meilleurs rendus. Quoi qu'il en soit, elles permettent de n'apporter qu'une solution acceptable à ce problème par des techniques d'approximation linéaire. Des effets Mach restent toujours visibles par endroit (mais ils sont beaucoup moins flagrants).

2.2 Ombrage de Gouraud

L'idée de l'ombrage de Gouraud est d'interpoler les valeurs de l'intensité lumineuse sur un polygone à partir de l'intensité de ses sommets. On considère un polygone à n sommets $\{P_i\}_{i=1\ldots n}$. L'algorithme d'ombrage de Gouraud est présenté ici dans le cas des polygones convexes mais peut facilement se généraliser au cas de polygones quelconques :

1. Calculer les normales $\mathbf{N}_i$ en chaque sommet P_i (voir section 2.5).
2. Calculer l'intensité I_i de la lumière au point P_i.
3. Appliquer l'algorithme de remplissage des polygones (voir page 37) avec les modifications suivantes :
   La scanline courante de position x, intersecte deux segments S_H et S_B dont les extrémités respectives sont P_u, P_{u+1} et P_v, P_{v+1}. On note l'intersection R_H (resp. R_B) entre la scanline et S_H (resp. S_B). Alors l'ombrage sur la scanline se calcule en 3 étapes (voir figure 4.8) :

   (a) Calcul de l'intensité au point R_H par interpolation linéaire des intensités lumineuses des extrémités de S_H :

$$I_H = \lambda_H . I_u + (1 - \lambda_H) . I_{u+1} \text{ avec } \lambda_H = \frac{R_H P_{u+1}}{P_u P_{u+1}}$$

(b) Calcul de l'intensité au point R_B par interpolation linéaire des intensités lumineuses des extrémités de S_B :

$$I_B = \lambda_B.I_v + (1 - \lambda_B).I_{v+1} \text{ avec } \lambda_B = \frac{R_B P_{v+1}}{P_v P_{v+1}}$$

(c) Calcul de l'intensité I_Q en tout point Q situé entre R_B et R_H de la scanline par interpolation linéaires des intensités lumineuses en R_B et R_H :

$$I_Q = \lambda.I_B + (1 - \lambda).I_H \text{ avec } \lambda = \frac{QR_H}{R_B R_H}$$

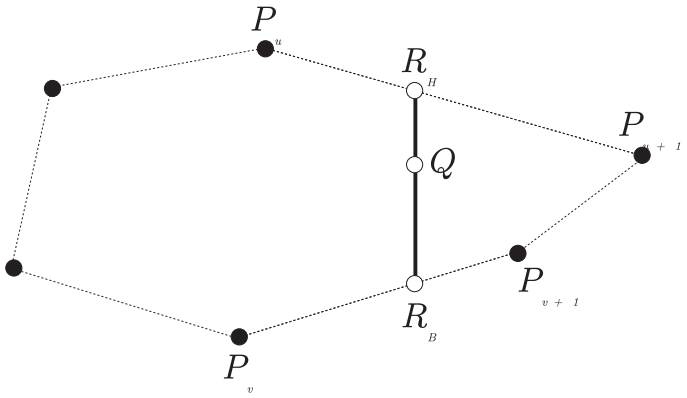


FIG. 4.8: Calcul d'un point par interpolation bilinéaire en utilisant l'algorithme *ScanLine*.

Les étapes $3a$, $3b$ et $3c$ peuvent être implémentées de façon incrémentale. Par exemple, pour $3c$, pour une même scanline, si on note $n + 1$ le nombre de points entre R_H et R_B, λ est de la forme $k.\delta$ (avec $\delta = n^{-1}$ et $k = 0...n$). Et donc, l'intensité du $k^{\text{ème}}$ point est :

$$I_k = I_H + k.\Delta I$$

avec $\Delta I = I_B - I_H$.

L'inconvénient principal de ce modèle est que les reflets spéculaires seront très influencés par la décomposition en polygones à l'endroit où la réflexion spéculaire se produit.

Ce modèle a l'avantage d'être simple, rapide et donne de bons résultats pour les modèles simples d'illumination n'intégrant pas la réflexion spéculaire.

2.3 Ombrage de Phong

L'ombrage de Phong constitue une amélioration importante du modèle de Gouraud mais est plus gourmand en temps de calcul. Le principe est le même : celui de l'interpolation ; sauf qu'au lieu d'interpoler les intensités, on interpole les normales aux

(a) Ombrage constant (b) Ombrage de Gouraud

FIG. 4.9: Comparaison entre l'ombrage constant et l'ombrage de Gouraud sur un objet construit à base de figure polygonale.

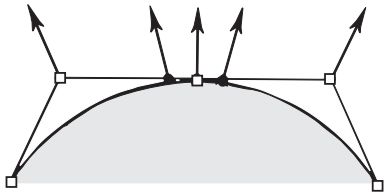


FIG. 4.10: Principe de l'interpolation des normales.

surfaces. Ce principe revient à reconstruire une surface à partir de l'interpolation des normales.

On considère un polygone à n sommets $\{P_i\}_{i=1...n}$. L'algorithme d'ombrage de Phong est présenté ici dans le cas des polygones convexes mais peut facilement se généraliser au cas de polygones quelconques :

1. Calculer les normales $\mathbf{N}_i$ en chaque sommet P_i (voir section 2.5).
2. Appliquer l'algorithme de remplissage des polygones (voir page 37) avec les modifications suivantes :

   La scanline courante de position x intersecte deux segments S_H et S_B dont les extrémités respectives sont P_u, P_{u+1} et P_v, P_{v+1}. On note l'intersection R_H (resp. R_B) entre la scanline et S_H (resp. S_B). Alors l'ombrage sur la scanline se calcule en 3 étapes (voir figure 4.8) :

(a) Calcul de la normale $\mathbf{N}_H$ au point R_H par interpolation linéaire des normales des extrémités de S_H :

$$\mathbf{N}_H = \lambda_H.\mathbf{N}_u + (1 - \lambda_H).\mathbf{N}_{u+1} \text{ avec } \lambda_H = \frac{R_H P_{u+1}}{P_u P_{u+1}}$$

(b) Calcul de la normale $\mathbf{N}_B$ au point R_B par interpolation linéaire des normales des extrémités de S_B :

$$\mathbf{N}_B = \lambda_B.\mathbf{N}_v + (1 - \lambda_B).\mathbf{N}_{v+1} \text{ avec } \lambda_B = \frac{R_B P_{v+1}}{P_v P_{v+1}}$$

(c) Calcul de la normale $\mathbf{N}_Q$ en tout point Q situé entre R_B et R_H de la scanline par interpolation linéaires des normales en R_B et R_H :

$$\mathbf{N}_Q = \lambda.\mathbf{N}_B + (1 - \lambda).\mathbf{N}_H \text{ avec } \lambda = \frac{Q R_H}{R_B R_H}$$

(d) Calcul de l'intensité de la lumière au point Q.

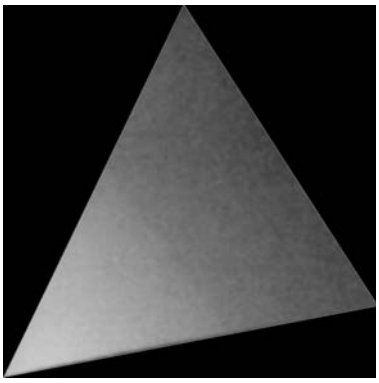
`Attention`, même si toutes les normales sont normalisées (et elles doivent l'être), les vecteurs normaux $\mathbf{N}_Q$ obtenus par interpolation **ne sont pas normalisés**.

Comme pour l'ombrage de Gouraud, les étapes $3a$ à $3c$ peuvent être implémentées de manière incrémentale. Ce modèle résout les problèmes de Gouraud pour la réflexion spéculaire (voir figure 4.11).

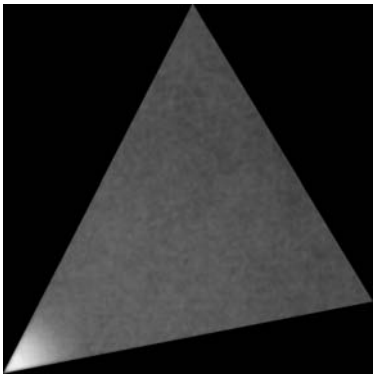
2.4 Problème avec ces modèles

Les ombrages de Gouraud et de Phong ont plusieurs inconvénients majeurs (voir figure 4.12) :

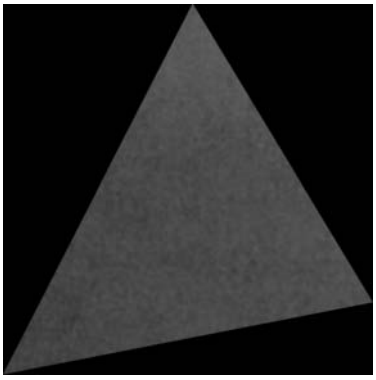
1. L'utilisation de polygones non convexes ou de plusieurs polygones partageant le même coté d'un autre polygone génère presque sûrement des anomalies. Ce type de polygones est donc à proscrire.
2. Dans le cas où la normale présente des oscillations d'un polygone à ses voisins, le calcul de la normale au sommet par moyennage a généralement pour conséquence d'atténuer fortement ces oscillations, et donc de ne pas être représentatif de la géométrie de la surface.
3. L'utilisation de l'algorithme scanline a un prix : si un polygone subit une rotation, alors les interpolations linéaires ne se font plus nécessairement de manière identique. Il en résulte que des changements très visibles d'intensité sur une même facette peuvent survenir pendant une animation.
4. L'interpolation linéaire engendre des déformations de perspective : il n'y a pas de différence entre l'ombrage d'un polygone parallèle au plan de projection et celui d'un polygone non parallèle si leurs projections sur le plan de projection est identique.



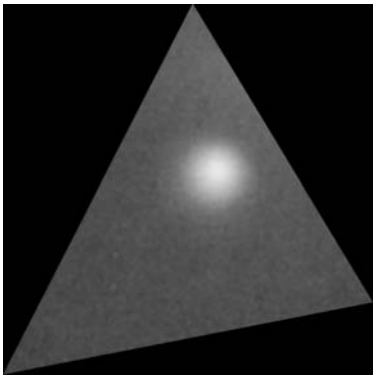
(a) Cas 1 : Gouraud



(b) Cas 1 : Phong

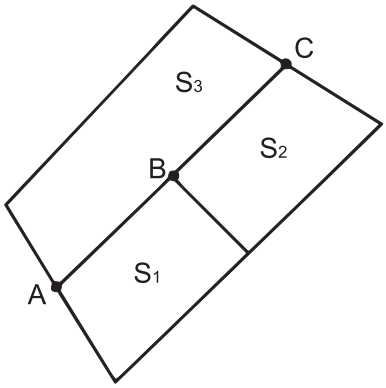


(c) Cas 2 : Gouraud

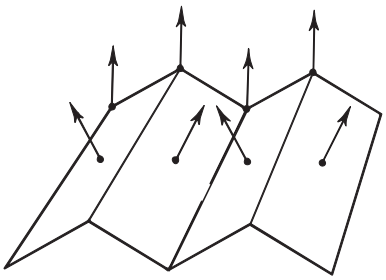


(d) Cas 2 : Phong

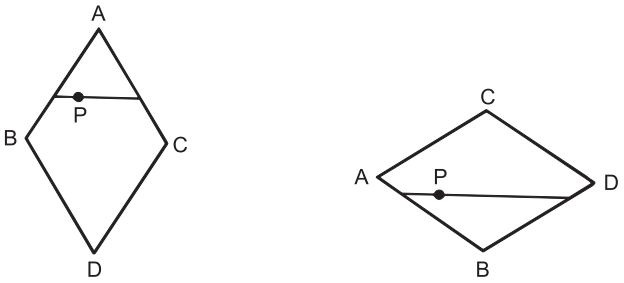
FIG. 4.11: Comparaison des modèles de Gouraud et Phong pour la réflexion spéculaire sur une facette. On présente deux cas. Pour le cas 1, la réflexion spéculaire maximale a lieu à l'un des sommets de la facette. Pour le cas 2, la réflexion spéculaire maximale tombe à l'intérieur de la facette.



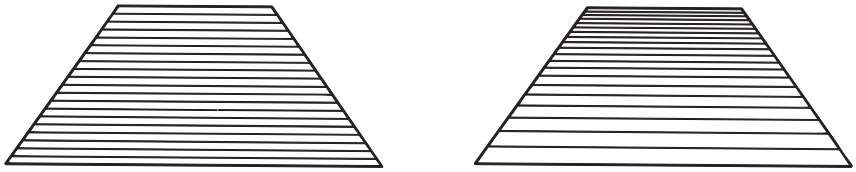
(a) Partage du même côté d'un polygone S_3 par deux polygones S_1 et S_2. L'interpolation sur S_3 se fait de la valeur en A à la valeur en C. L'interpolation sur S_1 se fait de la valeur en A à la valeur en B. La valeur en B est généralement différente de la valeur interpolée en B à partir de celle en A et C.



(b) La normale calculée par moyennage aux sommets donne une apparence de plat à une surface oscillante.



(c) Problème de l'interpolation bilinéaire : la valeur obtenue dépend de l'orientation de la facette.



(d) Déformation de perspective. A gauche, les z sont calculés aux sommets, et interpolé dans le plan de l'écran (interpolation après projection). A droite, les z sont calculé en chaque pixel à partir de l'équation réelle du plan.

FIG. 4.12: Les problèmes engendrés par les méthodes d'ombrage interpolé

5. Si la surface obtenue parait visuellement lisse, le contour de l'objet reste lui polygonal.

La solution à ce type de problème est souvent d'augmenter le nombre de polygones pour décrire l'objet afin de limiter les effets décrits.

2.5 Calcul des normales

Nous avons vu que dans les modèles d'ombrage des surfaces à base de polygones, les normales aux sommets des polygones décrivant la surface sont nécessaires pour effectuer les calculs. On a alors deux cas :

- Si l'équation de la surface originale est connue, alors il est toujours préférable de calculer la véritable valeur de la normale à partir de l'équation plutôt que de l'estimer à partir de son approximation polygonale.
- Sinon, la normale est estimée par moyennage des normales sur les polygones adjacents.
  On rappelle tout d'abord que si A, B et C sont trois points quelconques du polygone P alors $\mathbf{AB} \wedge \mathbf{AC}$ est un vecteur normal au polygone. On note $\mathbf{N}(P) = \frac{\mathbf{AB}}{AB} \wedge \frac{\mathbf{AC}}{AC}$ la normale normalisée du polygone P.
  Si on suppose qu'un objet est construit à partir d'un ensemble de polygones $\{F_j\}$, et que $\{P_i\}$ est l'ensemble des sommets de ces polygones, alors la normale $\mathbf{N}_i$ au sommet P_i est approximée par :

$$\mathbf{N}_i = \frac{1}{\left|\sum_{P_i \in F_j} N(F_j)\right|} \sum_{P_i \in F_j} \mathbf{N}(F_j)$$

  où $P_i \in F_j$ signifie que le point P_i est un sommet du polygone F_j. Attention, il est important que les vecteurs $\mathbf{N}(F_j)$ soient normalisés afin de donner autant de poids à chaque polygone.

3 Exercices

Exercice 16 (Exemple avec le modèle de Phong) *On considère le cube unité de paramètre $k_a = 0.3$, $k_d = 0.4$, $k_s = 0.6$, et $n = 2$. L'observateur est placé en $(4, 0, 5)$, regarde vers le centre $(0, 0, 0)$ du repère. Le haut approximatif pour l'observateur est parallèle à l'axe Oz. La lumière ambiante a pour intensité $I_a = 50$. Une source de lumière ponctuelle est placée en $(-6, 6, 6)$ et a pour intensité $I = 100$.*

1. *Choisir une face adjacente à la face supérieure du cube où le phénomène lumineux observés sont de nature différent à ceux de la face supérieure.*
2. *Calculer l'intensité au centre de ces deux faces en utilisant le modèle élémentaire de rendu.*
3. *Même question en utilisant l'ombrage de Gouraud.*

4. *Même question en utilisant l'ombrage de Phong.*

Exercice 17 (Calcul des ombres avec le z-buffer) *Le but de cet exercice est de vous faire pratiquer numériquement le calcul des ombres avec l'algorithme du z-buffer.*

1. *Est-il possible d'utiliser la scène mise en perspective pour effectuer un calcul des ombres avec le z-buffer ? Si non, dessiner un contre-exemple.*
2. *On considère l'exemple 2D de la figure 4.13. L'origine du repère global est en O. L'observateur est placé en P. La source de lumière se trouve en A. Pour simplifier, on effectuera des projections orthogonales. L'échelle à choisir pour la discrétisation de l'espace est donnée par la grille unitaire surimposée. Les plans de projection sont indiqués par des traits gras.*

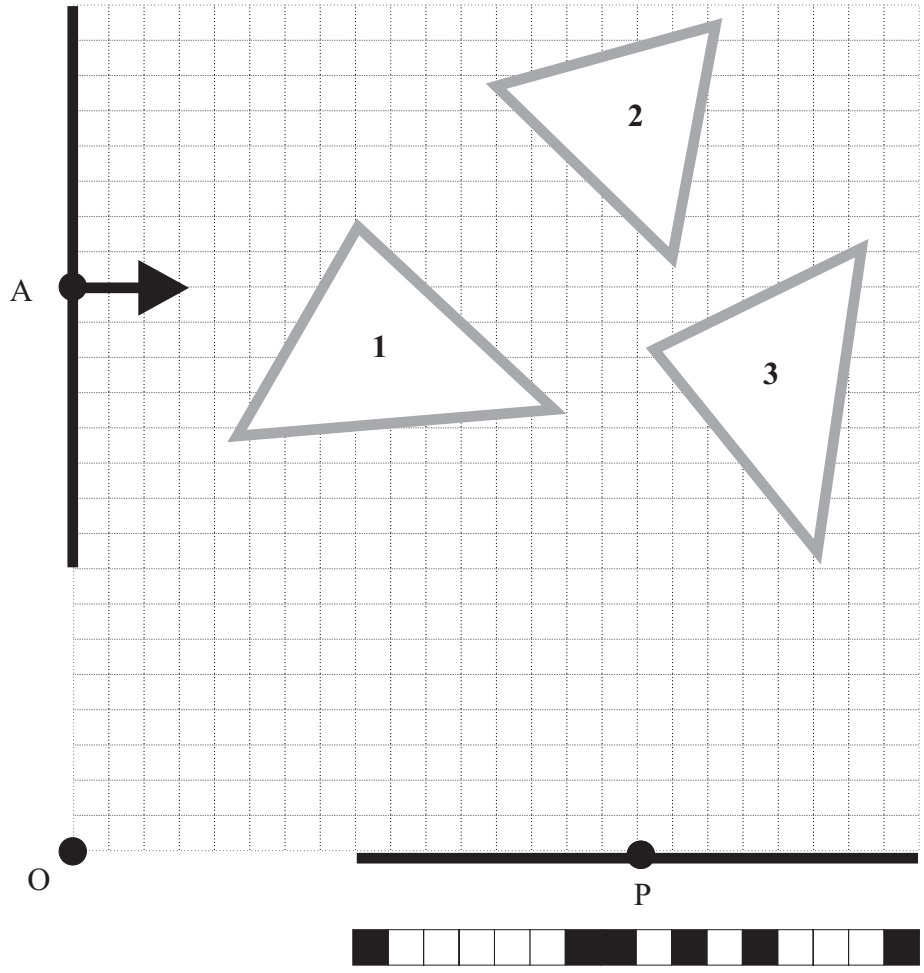


FIG. 4.13: pratique de l'ombrage avec le z-buffer de l'exercice 17.

(a) *Calculer le résultat du z-buffer pour l'observateur et la source de lumière. Traiter les polygones dans l'ordre indiqué.*

(b) *Calculer la transformation qui permet de passer du repère de la source de lumière au repère de l'observateur.*

(c) *Pour les colonnes marquées d'un carré noir dans le plan de l'observateur, décider numériquement si le pixel est ombré ou pas.*

Exercice 18 (Ombrage de Gouraud sur un polygone) *On souhaite implémenter l'ombrage de Gouraud sur un polygone convexe avec un algorithme n'utilisant que des entiers. On suppose que l'on dispose de l'ensemble des n sommets $\{P_k = (i_k, j_k)\}_{k=1...n}$ du polygone, ainsi que pour chacun de ces sommets de la valeur de luminance $\{L_k\}_{k=1...n}$ (nombre entier entre 0 et 255).*

1. *Dans la chaîne de traitement, quand peut-on calculer au plus tôt la valeur de la luminance en tous les sommets ? Est-il raisonnable de le faire ? Justifier.*
2. *Expliquer comment l'algorithme de Bresenham est utilisé pour interpoler les valeurs de luminance lors de l'algorithme scanline de remplissage du polygone. On décomposera le problème de la façon suivante pour chacun des deux cas d'interpolation : passage à la ligne suivante et déplacement sur la même ligne.*
   (a) *Conditions initiales*
      – *Quelle est la première valeur de la luminance L ?*
      – *Quelle est la valeur initiale de la fonction d'erreur ?*
   (b) *Test incrémental*
      – *Quel est le test entier qu'il faut réaliser pour choisir la valeur suivante de L ?*
      – *Une fois la valeur choisie, comment mettre à jour la fonction d'erreur ?*
   (c) *Écrire l'algorithme correspondant.*
3. *Écrire l'algorithme général correspondant.*