

## Image de Synthèse



**Année scolaire  
2015-2016**



**4 avril 2016**

**Éric Guérin**

---

## TABLE DES MATIÈRES

<b>1 Introduction</b>	<b>5</b>
1.1 Introduction . . . . .	5
1.2 Applications . . . . .	5
1.3 Pipeline graphique . . . . .	6
<b>2 Transformations géométriques</b>	<b>7</b>
2.1 Introduction . . . . .	7
2.2 Perspective . . . . .	9
<b>3 Modélisation géométrique</b>	<b>11</b>
3.1 Introduction . . . . .	11
3.2 Exemple : le plan . . . . .	11
3.3 Quelle représentation utiliser ? . . . . .	11
3.4 Primitives . . . . .	12
3.5 Maillages polygonaux . . . . .	12
3.6 Heightmap . . . . .	13
3.7 Autres modèles . . . . .	13
<b>4 Modélisation paramétrique</b>	<b>15</b>
4.1 Introduction . . . . .	15
4.2 Courbes de Bézier . . . . .	15
4.3 B-Splines . . . . .	15
4.4 Carreaux de Bézier . . . . .	16
<b>5 Surfaces implicites</b>	<b>17</b>
5.1 Introduction . . . . .	17
5.2 Surfaces algébriques . . . . .	17
5.3 Squelettes . . . . .	17
5.4 Visualisation . . . . .	18
<b>6 Modélisation itérative</b>	<b>19</b>
6.1 Introduction . . . . .	19
6.2 De Casteljau . . . . .	19
6.3 IFS . . . . .	19
6.4 Subdivision fractale . . . . .	20

---

6.5	Diamond-square . . . . .	20
<b>7</b>	<b>Boîte à outils mathématique</b>	<b>23</b>
7.1	Distances . . . . .	23
7.2	Intersections . . . . .	24
7.3	Triangles . . . . .	24
<b>8</b>	<b>Affichage</b>	<b>25</b>
8.1	Introduction . . . . .	25
8.2	Lancer de rayon . . . . .	25
8.3	Affichage direct . . . . .	27
<b>9</b>	<b>Illumination</b>	<b>31</b>
9.1	Introduction . . . . .	31
9.2	Illumination globale . . . . .	32
9.3	Illumination simplifiée . . . . .	32
9.4	Calcul de couleur . . . . .	34
<b>10</b>	<b>Génération procédurale</b>	<b>35</b>
10.1	Introduction . . . . .	35
10.2	Textures . . . . .	35
10.3	Contenu 3D . . . . .	37

# CHAPITRE 1

## INTRODUCTION

### 1.1 Introduction

L'image de synthèse permet de générer des images 2D ou 3D, de manière réaliste ou non (notion de photo-réalisme).

Ces images sont le reflet d'une réalité concrète ou abstraite.

De nombreuses personnes sont impliquées dans le processus de création d'une image de synthèse parmi lesquelles ont trouvé des graphistes, *designer* mais aussi des informaticiens.

#### ► Scène 3D

Une scène 3D est composée au minimum :

- d'objets (exemple une table) situés dans un environnement (exemple un terrain avec sa végétation)
- d'un ou plusieurs éclairages naturels (soleil) ou artificiels
- d'un point de vue

#### ► Animation

Tous ce qui compose une scène est paramétré et ces paramètres évoluent dans le temps. Exemples :

- Un objet bouge
- La caméra (point de vue) bouge
- Un objet se transforme
- Un objet change d'aspect (texture par exemple)
- L'éclairage est modifié (le soleil bouge par exemple)
- Un nouvel objet apparaît

Cette notion d'évolution décrit ce que l'on appelle l'animation.

### 1.2 Applications

Les applications de l'image de synthèse sont multiples :

- Industrie du film (effets spéciaux, films d'animation, etc.)
- Jeu vidéo
- CAO/éditeurs
- Réalité virtuelle
- Visualisation scientifique
- Imagerie médicale

On distingue deux grandes catégories d'applications : temps réel et hors-ligne.

#### ► Temps-réel

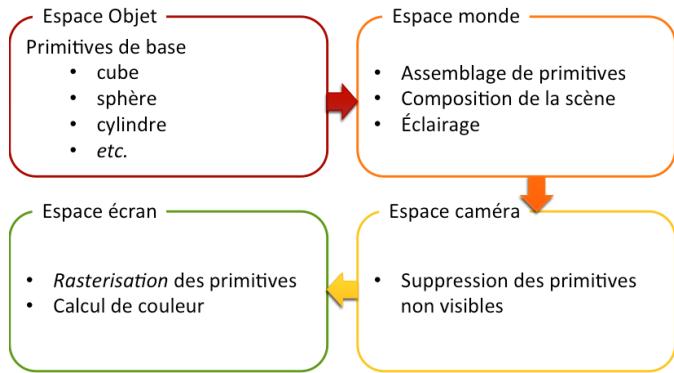
Dans les applications temps-réel, la contrainte principale est le taux de rafraîchissement (intervalle entre deux images affichées) qui doit être au pire d'un vingtième de seconde. Pour ces applications, on exploite le matériel (GPU), on utilise souvent des modèles d'illumination simplifiés et une géométrie simplifiée. Exemple d'applications temps-réel : jeu vidéo, CAO, simulateur, etc.

#### ► Offline

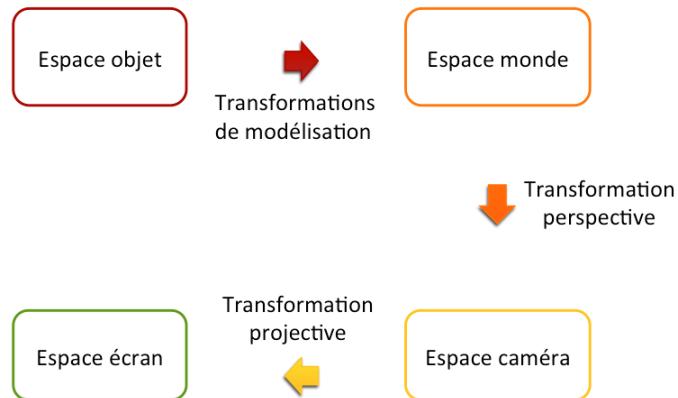
La contrainte des applications dites hors-ligne ou *offline* en anglais est d'avoir une qualité maximum. Cette qualité se traduit au niveau de la géométrie (détails), des textures haute définition, et des modèles d'illumination globale mettant en jeu plusieurs rebonds de lumière. Souvent, le rendu est fait sur des clusters et on utilise le compositing pour former l'image finale (calques superposés).

### 1.3 Pipeline graphique

La modélisation d'une scène 3D se fait grâce à un processus (*pipeline* en anglais) qui ressemble à cela :



On passe d'un espace à un autre par des transformations :



Les transformations de modélisation sont des translations, rotations et mises à l'échelle. Elles permettent de positionner des objets dans une scène.

La transformation perspective permet d'introduire un ou plusieurs points de fuite.

La dernière transformation dite projective permet d'enlever une coordonnée (la profondeur) afin de pouvoir visualiser sur un écran 2D.

Remarque : un terminal de visualisation 3D (lunette, moniteur ou télévision) prend en entrée deux images 2D avec deux points de vue différents.

## CHAPITRE 2

### TRANSFORMATIONS GÉOMÉTRIQUES

#### 2.1 Introduction

En géométrie Euclidienne, un point 2D est représenté par un vecteur de dimension 2. Un point 3D est représenté par un vecteur de dimension 3.

On peut modéliser les transformations de ces points grâce à des applications de  $\mathbb{R}^n$  vers  $\mathbb{R}^n$  avec  $n = 2$  ou  $n = 3$ .

Exemple, une translation de vecteur  $v$  se fera avec l'application  $T_v$  suivante :

$$\begin{aligned} T_v : \quad E &\rightarrow E \\ p &\mapsto p + v \end{aligned}$$

Une homothétie de rapport  $a$  :

$$\begin{aligned} H_a : \quad E &\rightarrow E \\ p &\mapsto ap \end{aligned}$$

Une rotation 2D d'angle  $\theta$  :

$$R_\theta : \quad \begin{aligned} E_2 &\rightarrow E_2 \\ \begin{pmatrix} x \\ y \end{pmatrix} &\mapsto \begin{pmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{pmatrix} \end{aligned}$$

Que l'on peut noter de manière matricielle par :

$$R_\theta(p) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Pour les rotations en 3D, il faut en plus choisir quel est l'axe de rotation. Exemple avec  $z$  :

$$R_\theta^z(p) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

On voit bien que la coordonnée  $z$  n'est pas modifiée et que les coordonnées  $(x, y)$  subissent une rotation 2D.

Lorsque l'on veut composer (c'est-à-dire enchaîner) plusieurs rotations, on peut multiplier les matrices de rotation correspondantes :

$$R_\theta^z \circ R_\alpha^x(p) = \mathcal{R}_\theta^z \mathcal{R}_\alpha^x p$$

Cette représentation matricielle existe aussi pour les mises à l'échelle :

$$H_a(p) = \mathcal{H}_a p = \begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

On peut généraliser les mises à l'échelle avec un coefficient différent suivant les axes concernés :

$$H_a(p) = \mathcal{H}_a p = \begin{pmatrix} a_x & 0 & 0 \\ 0 & a_y & 0 \\ 0 & 0 & a_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Et au final, la composition de mises à l'échelle et de rotations se traduit par la multiplication des matrices correspondantes :

$$H_a \circ R_\alpha^x(p) = \mathcal{H}_a \mathcal{R}_\alpha^x p$$

La notation matricielle est pratique car elle permet de composer les transformations de manière simple grâce à des multiplications de matrices. Cependant, cette représentation matricielle est limitée aux mises à l'échelle et aux rotations. On ne peut pas représenter la translation car il s'agit d'une transformation affine et non linéaire.

La solution réside dans l'utilisation de coordonnées homogènes et les espaces projectifs.

## ► Coordonnées homogènes

L'idée est d'ajouter une composante au système de coordonnées. En 2D, on passe à trois composantes et en 3D à quatre composantes.

La dernière coordonnée (dite homogène et notée  $w$ ) a un rôle particulier de diviseur par rapports aux autres.

Il existe un lien (non bijectif) entre les systèmes de coordonnées cartésien et homogène. Exemple en 2D :

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix}_h \Leftrightarrow_{w \neq 0} \begin{pmatrix} x/w \\ y/w \end{pmatrix}_c$$

$w$  ayant un rôle de diviseur, l'équivalent cartésien n'existe pas lorsqu'il est nul. Lorsque  $w = 1$ , les représentations correspondent. On voit aussi que lorsque l'on multiplie toutes les coordonnées homogènes par un facteur non nul, elles représentent le même point.

## ► En 2D

Une multiplication matricielle dans l'espace projectif se traduit par (exemple en 2D) :

$$\begin{pmatrix} x'_h \\ y'_h \\ w'_h \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ w_h \end{pmatrix}$$

On a donc :

$$\begin{cases} x'_h = ax_h + by_h + cw_h \\ y'_h = dx_h + ey_h + fw_h \\ w'_h = gx_h + hy_h + iw_h \end{cases}$$

Or :

$$\begin{aligned} x'_c &= \frac{x'_h}{w'_h} \\ &= \frac{ax_h + by_h + cw_h}{gx_h + hy_h + iw_h} \end{aligned}$$

Faisons pour l'instant la simplification en prenant  $g = h = 0$  et  $i = 1$ , on arrive à :

$$\begin{aligned} x'_c &= \frac{ax_h + by_h + cw_h}{w_h} \\ &= a\frac{x_h}{w_h} + b\frac{y_h}{w_h} + c \\ &= ax_c + by_c + c \end{aligned}$$

Ainsi la matrice  $\begin{pmatrix} a & b \\ d & e \end{pmatrix}$  correspond à la matrice de rotation et mise à l'échelle habituelle et les coefficients  $c$  et  $f$  représentent le vecteur de translation.

Mais cela implique de respecter la structure suivante :

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

La multiplication de deux matrices de transformations homogènes va effectivement produire la composition des rotations, mises à l'échelle et translations associées :

$$\mathcal{T}_1 = \begin{pmatrix} A_{00} & A_{10} & S_x \\ A_{01} & A_{11} & S_y \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathcal{T}_2 = \begin{pmatrix} B_{00} & B_{10} & T_x \\ B_{01} & B_{11} & T_y \\ 0 & 0 & 1 \end{pmatrix}$$

Le résultat de cette multiplication :

$$\mathcal{T} = \mathcal{T}_1 \mathcal{T}_2 = \begin{pmatrix} C_{00} & C_{10} & U_x \\ C_{01} & C_{11} & U_y \\ 0 & 0 & 1 \end{pmatrix}$$

Avec :

$$\begin{cases} C = AB \\ U = AT + S \end{cases}$$

On voit sur ce résultat que l'on applique la rotation/mise à l'échelle de  $T_1$  à la translation de  $T_2$ , ce qui est logique.

## ► En 3D

C'est exactement la même chose !

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}_h \Leftrightarrow_{w \neq 0} \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}_c$$

Les transformations de type translation/rotation/- mise à l'échelle ont donc cette forme :

$$\mathcal{T} = \begin{pmatrix} A_{00} & A_{10} & A_{20} & S_x \\ A_{01} & A_{11} & A_{21} & S_y \\ A_{02} & A_{12} & A_{22} & S_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

où  $A$  est la matrice de rotation/mise à l'échelle et  $S$  est le vecteur de translation.

Cette matrice  $A$  peut contenir d'autres types de transformations :

- Des cisaillements (les objets transformés sont "penchés")

$$A_{C_x} = \begin{pmatrix} 1 & c_1 & c_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Des symétries/miroirs

$$A_{m_x} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

## 2.2 Perspective

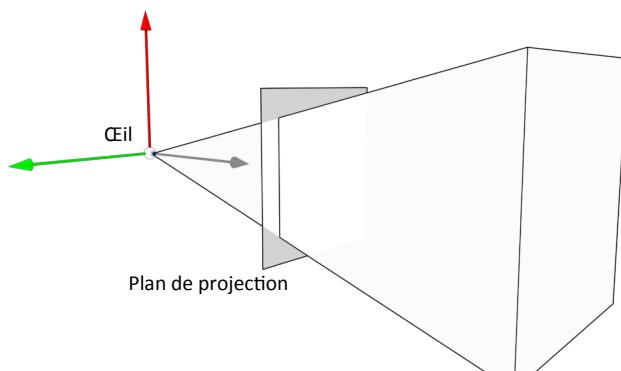
Nous n'avons pas pour l'instant utilisé les coefficients de la matrice homogène qui se trouvent sur la dernière ligne. Ils ont un rôle particulier car permettent d'injecter dans la coordonnée homogène une information issue des autres coordonnées.

C'est comme cela que l'on va décrire des transformations perspective, c'est-à-dire qui modélisent la vision de l'œil (point de fuite).

### ► Intuition

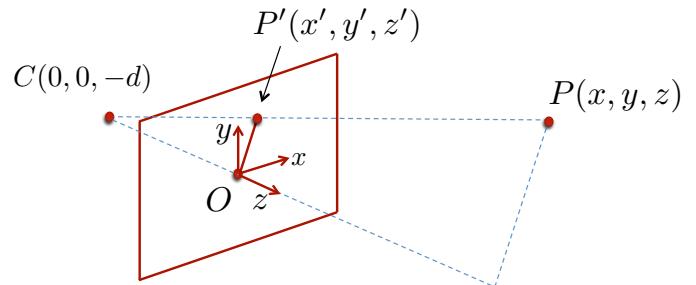
Des droites qui sont parallèles dans l'espace vont s'intersecter dans notre système de vision. La transformation perspective utilise une coordonnée (par exemple  $z$ ) qu'elle va injecter dans la coordonnée homogène. Étant donné que la coordonnée homogène agit comme un diviseur, plus l'objet sera loin, plus il apparaîtra petit.

Plaçons-nous dans le contexte suivant :



On place l'œil (donc la caméra) à un endroit et on considère que l'on projette la scène sur un plan appelé plan de projection.

Nous allons maintenant essayer de calculer les coordonnées d'un point projeté sur le plan  $(0, x, y)$  qui serait vu d'un point  $C$  sur l'axe  $(O, z)$  à une distance  $d$  de  $O$  :



Le théorème de Thalès nous dit :

$$\left\{ \begin{array}{l} \frac{x'}{x} = \frac{y'}{y} = \frac{d}{d+z} \\ z' = 0 \end{array} \right.$$

On a donc :

$$\left\{ \begin{array}{l} x' = x/(1 + \frac{z}{d}) \\ y' = y/(1 + \frac{z}{d}) \\ z' = 0 \end{array} \right.$$

Nous sommes dans le plan  $(O, x, y)$ , c'est pour cela que l'on a  $z' = 0$ .

Essayons maintenant de nous arranger pour que  $z$  ait le même changement d'échelle que les autres coordonnées :

$$\left\{ \begin{array}{l} x' = x/(1 + \frac{z}{d}) \\ y' = y/(1 + \frac{z}{d}) \\ z' = z/(1 + \frac{z}{d}) \end{array} \right.$$

Dont l'équivalent en coordonnées homogènes est :

$$\begin{pmatrix} x/(1 + \frac{z}{d}) \\ y/(1 + \frac{z}{d}) \\ z/(1 + \frac{z}{d}) \end{pmatrix}_c \Leftrightarrow \begin{pmatrix} x \\ y \\ z \\ 1 + \frac{z}{d} \end{pmatrix}_h$$

Cette transformation peut s'écrire sous forme matricielle homogène :

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}$$

Cette transformation conserve la coordonnée de profondeur  $z$  donc ce n'est pas une projection.

La distance  $d$  qui sépare l'œil du plan de projection s'appelle la distance focale. La coordonnée de profondeur  $z$  qui reste disponible permet de faire des tests de visibilité d'objets (un objet cache un autre lorsqu'il est plus proche).

Afin de pouvoir afficher cela sur un écran il faut faire une projection qui consiste à supprimer une coordonnée.

## ► Projection

L'opération de projection est très simple puisqu'elle se contente d'enlever une dimension sans modifier les autres.

On peut la traduire par un matrice dont une colonne est nulle :

$$P_{\parallel} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ceci va uniquement mettre à zéro une coordonnée. Si on voulait vraiment enlever cette coordonnées, il faudrait utiliser une matrice non carrée.

On peut utiliser ce type de projection sans transformation perspective préalable, c'est souvent le cas pour le dessin industriel où l'on veut voir toutes les droites parallèles. Ce type de visualisation n'est pas très réaliste toutefois et devra être évité dans un cadre plus général.

## CHAPITRE 3

### MODÉLISATION GÉOMÉTRIQUE

#### 3.1 Introduction

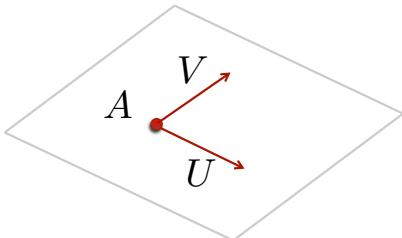
La modélisation géométrique est la représentation mathématique et/ou informatique d'objets du monde réel.

Il existe deux grandes familles de représentations :

- Explicite : on donne directement les informations sur la géométrie de l'objet
- Implicite : ce sont des équations qui décrivent l'objet, pour connaître cet objet il faut les résoudre

#### 3.2 Exemple : le plan

Pour représenter un plan dans l'espace, on utilise un point  $A$  et deux vecteurs directeurs  $U$  et  $V$  :



La représentation explicite (ici paramétrique) de ce plan est :

$$P = A + aU + bV, \quad a, b \in \mathbb{R}$$

La représentation implicite est :

$$\vec{AP} \cdot N = 0$$

où  $N$  est le vecteur normal au plan et  $\cdot$  est le produit scalaire.

En représentation explicite, on sous-entend :

$$\mathcal{P} = \{P(a, b), \quad a, b \in \mathbb{R}\}$$

avec :

$$\begin{aligned} P : \quad \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\ (a, b) &\mapsto A + aU + bV \end{aligned}$$

alors qu'en représentation implicite, on sous-entend :

$$\mathcal{P} = \{P \in \mathbb{R}^3 \mid \vec{AP} \cdot N = 0\}$$

Les objets mathématiques qui se cachent derrière les deux représentations sont vraiment différents (fonctions/ensembles).

Par la suite, on notera :

$$\vec{AB} = B - A$$

ce qui correspond à représenter les points et les vecteurs géométriques par leurs coordonnées.

#### 3.3 Quelle représentation utiliser ?

Il n'y a pas de bonne ou mauvaise représentation. Tout dépend du contexte dans lequel on veut l'utiliser.

Pour savoir si un point appartient ou non à un plan, il suffit d'injecter ses coordonnées dans la représentation implicite. Le résultat (nul ou non-nul) indiquera si le point appartient ou non au plan. Dans ce même exemple, la représentation explicite paramétrique n'est pas du tout adaptée.

Dans d'autres cas, la représentation explicite est mieux adaptée. Exemple, pour générer une grille de points sur un plan, il suffit de faire une boucle sur les paramètres  $a$  et  $b$  à intervalles réguliers.

Parfois il est très simple de passer d'une représentation à une autre. Dans le cas du plan, il suffit de faire :

$$N = U \wedge V$$

où  $\wedge$  représente le produit vectoriel.

### 3.4 Primitives

Nous allons voir dans cette section quelques exemples d'objets géométriques simples et leurs représentations explicite et implicite.

#### ► Sphère

Pour une sphère de rayon  $r$  et de centre  $O$ .

Représentation implicite :

$$\|P - O\| = r$$

Représentation explicite :

$$P = O + \begin{pmatrix} r \cos \alpha \cos \beta \\ r \cos \alpha \sin \beta \\ r \sin \alpha \end{pmatrix}, \quad \alpha \in [0, \pi/2], \beta \in [-\pi, \pi]$$

#### ► Droite

Pour une droite passant par  $A$  et de vecteur directeur  $U$ .

Représentation implicite (peu utilisée) : les vecteurs  $U$  et  $AP$  sont colinéaires, leur produit scalaire est égal au produit de leurs normes.

$$|(P - A).U| = \|P - A\|. \|U\|$$

Représentation explicite :

$$P = A + aU, \quad a \in \mathbb{R}$$

#### ► Autres

Pour décrire d'autres primitives géométriques (cylindre, cône, tore, etc.), on appliquera la méthode suivante :

- Représentation implicite. Il faut trouver une propriété géométrique qui est tout le temps valable pour cet objet et la traduire sous la forme d'une ou plusieurs équations.
- Représentation explicite. Il faut représenter un point sur l'objet en fonction d'un ou plusieurs paramètres. Le nombre de paramètres dépend de la dimension topologique de l'objet (1 : linéique, 2 : surfacique, 3 : volumique).

Exemple : si l'on veut trouver la représentation du volume (et non plus de sa surface) d'une sphère de centre  $O$  et de rayon  $R$ .

Représentation implicite :

$$\|P - O\| \leq R$$

Représentation explicite, il faut rajouter un paramètre  $r$  qui est une dimension supplémentaire :

$$P = O + \begin{pmatrix} r \cos \alpha \cos \beta \\ r \cos \alpha \sin \beta \\ r \sin \alpha \end{pmatrix}$$

avec  $\alpha \in [0, \pi/2]$ ,  $\beta \in [-\pi, \pi]$  et  $r \in [0, R]$ .

### 3.5 Maillages polygonaux

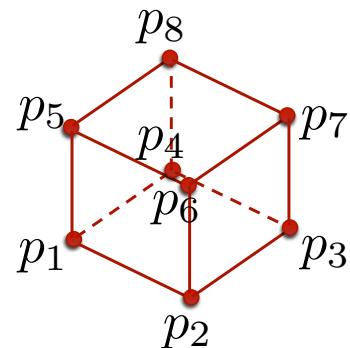
Il existe une représentation assez répandue pour décrire les objets qui s'appelle le maillage (*mesh* en anglais). Il consiste à décrire tout d'abord une liste numérotée de sommets (*vertex/vertices* en anglais) :

$$(p_i)_{i=1 \dots n}$$

et d'utiliser les indices de ces sommets pour décrire les polygones représentant les faces de l'objet :

$$(k_j^1, k_j^2, k_j^3, \dots)_{j=1 \dots m}$$

Exemple avec un cube, on a 8 sommets :



Les facettes quadrangulaires seront représentées par :

$$\begin{pmatrix} 1 & 4 & 3 & 2 \\ 1 & 2 & 6 & 5 \\ 2 & 3 & 7 & 6 \\ 3 & 4 & 8 & 7 \\ 1 & 5 & 8 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

Il est possible d'ajouter des informations au maillage comme par exemple la normale à chaque sommet ou bien des informations de voisinage. On peut par exemple indiquer quelles sont les facettes voisines, ou alors décrire la facette par une liste de segments qui sont partagés par les facettes voisines. Ceci permet d'accélérer certains calculs et parcours au prix d'un surcoût mémoire et d'une gestion plus difficile.

- Enumération spatiale
- Nuages de points
- Systèmes de particules
- Maillages volumiques
- Surfaces de subdivision
- Imposteurs
- Piles de matières
- etc.

### 3.6 Heightmap

Les *heightmap* ou *heightfield* ou images de profondeur sont un moyen de représenter une surface dont la projection sur un plan est homéomorphe à une grille. Dans ce cas là, la surface n'est pas réellement 3D (on utilise parfois le qualificatif 2,5D pour cela).

Prenons l'exemple d'un ensemble de points :

$$p_{ij} = (x_{ij}, y_{ij}, z_{ij}), i, j \in [1 \dots n]^2$$

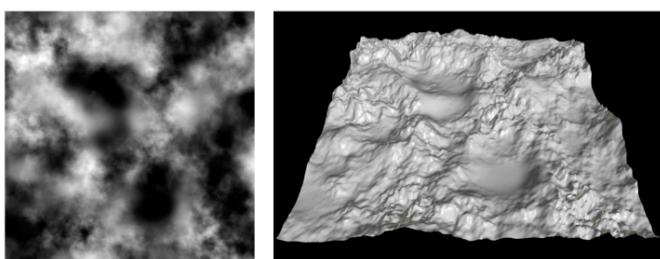
avec :

$$\begin{cases} x_{ij} = i \\ y_{ij} = j \\ z_{ij} = h_{ij} \end{cases}$$

La simple donnée de la matrice  $h$  permet de retrouver les coordonnées complètes de tous les points. Cette matrice peut être représentée par une image en niveaux de gris (d'où le terme image de profondeur).

Cette représentation est très utilisée pour les terrains mais ces terrains ne pourront pas avoir de surplombs ni de grottes.

Exemple :



Ici les zones noires de l'image sont les altitudes les plus basses.

### 3.7 Autres modèles

Il existe bien d'autres modèles géométriques que nous n'aborderons pas dans ce cours :



## CHAPITRE 4

### MODÉLISATION PARAMÉTRIQUE

#### 4.1 Introduction

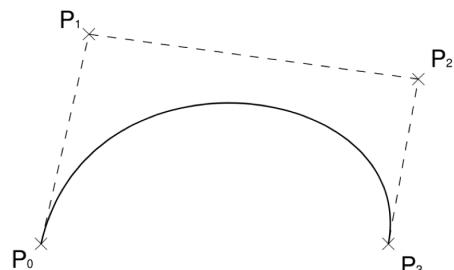
La modélisation paramétrique est un type de modélisation explicite qui utilise des fonctions.

Cela sert de manière historique à décrire des courbes et des surfaces. On peut facilement identifier les propriétés de continuité grâce à une analyse différentielle des fonctions mises en œuvre. Les fonctions utilisées sont souvent des polynômes pour leur caractère lisse. Plus le degré du polynôme est élevé, plus la courbe ou la surface sont lisses.

Ainsi, chaque point de la courbe est une combinaison barycentrique des points de contrôle. Cela rend la courbe invariante aux rotations (elle aura le même aspect mais sera totalement tournée).

Exemple, avec  $n = 3$ , on a 4 points de contrôle et des polynômes de degré 3 :

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$



#### 4.2 Courbes de Bézier

Une courbe de Bézier est une courbe contrôlée par  $n + 1$  points  $P_i$  appelés points de contrôle.

La fonction décrivant la courbe de Bézier est :

$$P(t) = \sum_{i=0}^n B_i^n(t) P_i$$

où les  $B_i^m$  sont les polynômes de Bernstein :

$$B_i^m(u) = \binom{m}{i} u^i (1-u)^{m-i}$$

Les coefficients de ces polynômes sont les coefficients binomiaux :

$$\binom{m}{i} = \frac{m!}{i!(m-i)!}$$

Une propriété importante des polynômes de Bernstein est :

$$\sum_{i=0}^m B_i^m(u) = 1$$

Les courbes de Bézier de degré 2 et 3 sont très utilisées pour décrire les fontes de caractères ainsi que dans les logiciels de dessin 2D.

Lorsque l'on veut contrôler une courbe qui a plus de points, on raccorde des bouts de Bézier en faisant attention aux raccords de tangentes. Cela évite d'avoir à augmenter le degré (et la complexité) de la courbe. On peut aussi recourir aux B-Splines.

#### 4.3 B-Splines

Les courbes de Bézier font apparaître un lien direct entre le degré de la courbe et le nombre de points de contrôle. Les B-Splines proposent une généralisation des courbes de Bézier qui permet de supprimer cette contrainte.

Le formalisme est un peu plus complexe. On se donne en entrée une liste de  $m + 1$  nœuds qui sont des scalaires :

$$0 \leq t_0 < t_1 < \dots < t_m \leq 1$$

on se donne aussi  $p = m - n$  points de contrôles (où  $n$  sera le degré de la courbe).

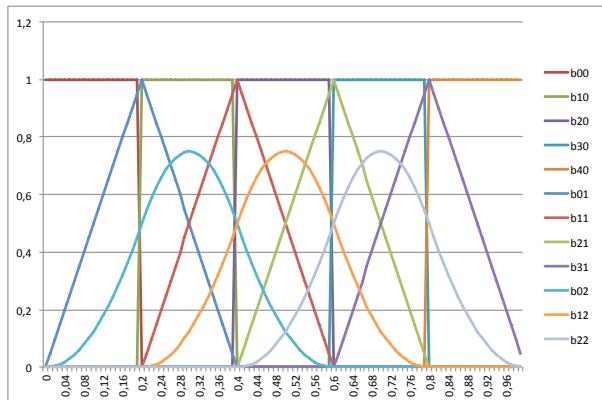
La formule de la courbe est donnée par :

$$S(t) = \sum_{i=0}^{m-n-1} b_{i,n}(t) P_i, \quad t \in [0, 1]$$

où les  $b_{i,n}$ ,  $i = 0 \dots m - n - 1$  sont des polynômes de degré  $n$  définis de manière récursive :

$$\begin{cases} b_{i,0}(t) = \begin{cases} 1 & \text{si } t_i \leq t < t_{i+1} \\ 0 & \text{sinon} \end{cases} \\ b_{i,n}(t) = \frac{t - t_i}{t_{i+n} - t_i} b_{i,n-1}(t) \\ \quad + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} b_{i+1,n-1}(t) \text{ pour } n > 0 \end{cases}$$

Par exemple, avec  $m = 5$  et  $n = 2$  (soit 3 points de contrôle) :



Ceci est facile à implémenter :

```
double spline(
    vector<double> & k,
    double t, double n, int i) {
if (n==0) {
    return (t>=k[i] && t<k[i+1])?1.:0.;
}
return spline(k,t,n-1,i)
    *(t-k[i])/(k[i+n]-k[i])
+spline(k,t,n-1,i+1)
    *(k[i+n+1]-t)/(k[i+n+1]-k[i+1]);
}
```

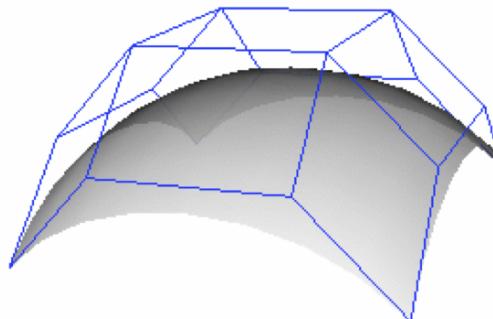
Une implémentation plus efficace consisterait à calculer les coefficients des polynômes par morceaux (pour chaque intervalle entre les nœuds), cela éviterait les diverses multiplications par zéro.

On voit que si on espaces les nœuds de manière régulière ( $t_i = i/m$ ) on arrive à une courbe de Bézier.

#### 4.4 Carreaux de Bézier

Pour définir une surface, on remplace la liste de points de contrôle par une grille rectangulaire de taille  $m \times n$  et on obtient :

$$P(s, t) = \sum_{i=0}^m \sum_{j=0}^n B_i^m(s) B_j^n(t) P_{ij}$$



Si on fixe une valeur de  $s$  ou  $t$ , on obtient une courbe de Bézier traditionnelle, ceci est vrai notamment pour les courbes qui bordent la surface.

## CHAPITRE 5

### SURFACES IMPLICITES

#### 5.1 Introduction

Dans une modélisation de type surface implicite, on a une fonction dite fonction potentiel qui change de signe lorsque l'on traverse la surface (on passe de l'intérieur à l'extérieur).

La surface est définie comme étant l'ensemble des points qui annulent la fonction potentiel (notion d'iso-surface).

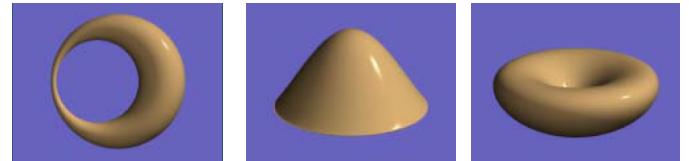
Exemple : avec une sphère de rayon  $R$  :

$$f(x, y, z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - R^2$$

La surface est définie par :

$$S = \{P / f(P) = c\}$$

L'inconvénient est que pour produire une surface, il faut trouver des points qui ont la propriété. En pratique, on va plonger l'espace dans une grille et on regarde sur chaque segment les changements de signe de la fonction (algorithme du *Marching Cube*). L'avantage est que la normale à la surface peut être calculée directement à partir du gradient de la fonction  $f$ .



#### 5.3 Squelettes

Les surfaces algébriques sont assez difficiles à définir. L'idée des modèles à squelette est de pouvoir à partir d'une forme simple guider la fonction potentiel.

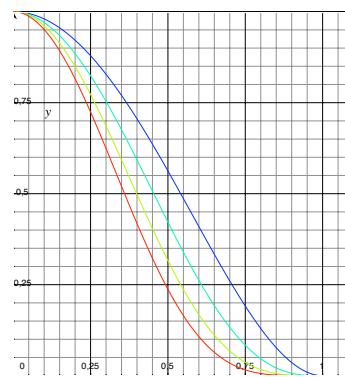
On se dote des éléments suivants :

- $S$  : un squelette (une forme, donc un ensemble de points)
- $d_S$  : une distance par rapport au squelette
- $g$  : une fonction potentiel

Exemple de fonction potentiel :

$$\begin{cases} g(r) = (1 - r^2)^n, & r \leq 1 \\ g(r) = 0, & r > 1 \end{cases}$$

avec  $n \geq 2$ .



#### 5.2 Surfaces algébriques

Les surfaces algébriques sont une catégorie de surface implicite pour lesquelles la fonction potentiel est un polynôme de degré  $n$  :

$$f(p) = f(x, y, z) = \sum_{0 \leq i+j+k \leq n} a_{ijk} x^i y^j z^k$$

Elles sont faciles à calculer et le calcul de la normale peut être analytique.

On peut ensuite faire le mélange de plusieurs primitives en faisant la somme (ou un autre opéra-

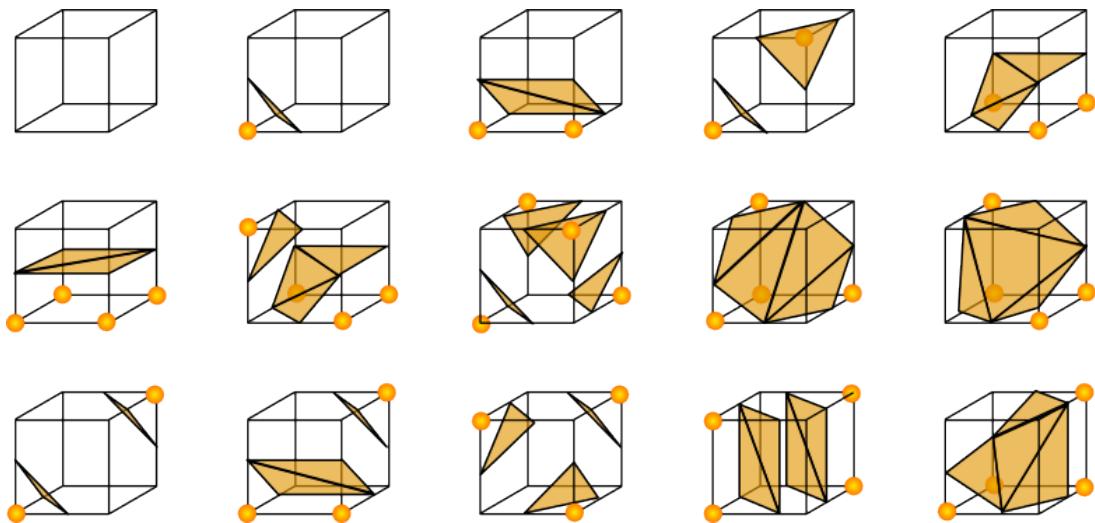
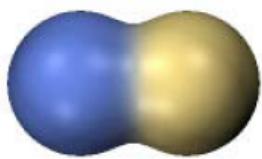


FIGURE 5.1 – Les configurations du marching cube

teur plus complexe) des fonctions de plusieurs squelettes :

$$f(p) = \sum_{0 \leq i < n} f_{S_i}(p) - T$$



Le type d'interpolation utilisé dans ce cas là est une interpolation linéaire :

$$I = \lambda P_1 + (1 - \lambda)P_2$$

$P_1$  —————  $P$  —————  $P_2$

$$\lambda = \frac{|f(P_2)|}{|f(P_1)| + |f(P_2)|}$$

## 5.4 Visualisation

L'algorithme du *marching cube* a été publié en 1987 à la conférence SIGGRAPH. Le principe est de plonger le domaine de visualisation dans une grille régulière. Chaque sommet de cette grille possède une valeur associée avec la fonction implicite. Chaque cube de la grille a une configuration qui est une variante des 15 types de configuration de base. La configuration donne la forme de la surface à cet endroit de la grille (voir figure 5.1).

Si la surface n'a qu'une seule composante connexe, il suffit de propager la reconstruction de la surface (d'où le terme *marching cube*). Il n'y a donc pas besoin de calculer la fonction aux endroits où la surface ne passe pas.

On peut approximer la normale grâce au calcul du gradient de la fonction implicite (approximé par une interpolation).

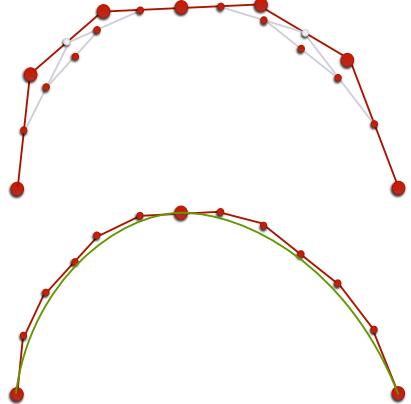
# CHAPITRE 6

## MODÉLISATION ITÉRATIVE

### 6.1 Introduction

Pour créer des formes, on peut aussi passer par des algorithmes itératifs qui convergent vers une figure. L'idée qui sera exploitée est de trouver un opérateur dans la figure qui est invariant : lorsque l'on applique ce dernier, la figure reste identique. Si on part d'une figure de départ et que l'on applique cet opérateur plusieurs fois, on converge vers l'invariant (appelé point fixe ou attracteur).

L'exemple typique est l'algorithme de De Casteljau qui permet de créer une courbe de Bézier de manière itérative.



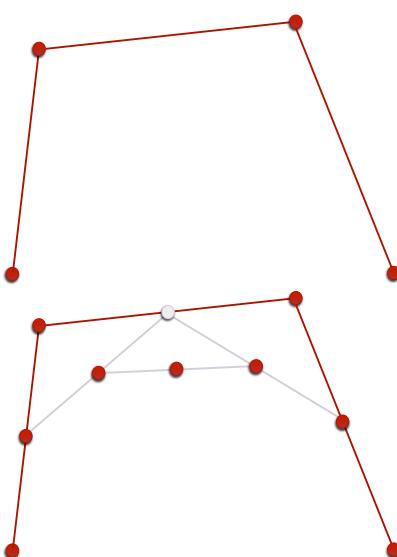
L'algorithme converge vers la courbe de Bézier de degré 3 (avec 4 points de contrôle).

Cet algorithme est assez particulier car on aurait pu utiliser l'expression analytique (polynôme) pour tracer la courbe.

D'une manière plus générale, les modèles itératifs sont utilisés lorsque l'on ne peut pas obtenir de description explicite ou analytique de la figure.

Quelques exemples :

- IFS : Iterated Function Systems
- L-systems
- Surfaces de subdivision
- Terrains fractals



### 6.3 IFS

Les IFS (Iterated Function Systems) ont été développés par Hutchinson et Barnsley dans les années 1980. Une transformation géométrique contractante dans un espace métrique complet possède un point fixe.

L'idée est de définir un opérateur sur les compacts

(un compact est un ensemble que l'on peut assimiler à une figure) de cet espace métrique complet. La théorie permet de définir un nouvel espace métrique complet dont le point fixe (appelé attracteur) sera un compact donc une figure composée de point du premier espace.

Voici un exemple dans le plan 2D de trois transformations affines de rapport  $\frac{1}{2}$  :

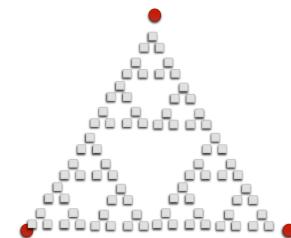
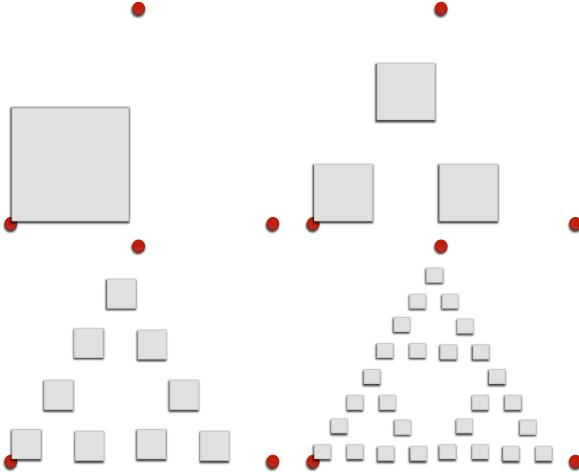
$$\begin{aligned} T_0(p) &= \frac{1}{2}p \\ T_1(p) &= \frac{1}{2}p + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ T_2(p) &= \frac{1}{2}p + \begin{pmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{pmatrix} \end{aligned}$$

Chacune de ces transformations possède un point fixe :

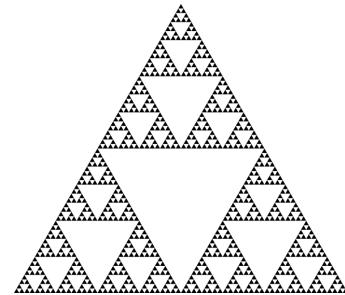
$$p_2 = \begin{pmatrix} \frac{\sqrt{3}}{2} \\ 1 \end{pmatrix}$$

$$p_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad p_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

Un algorithme de visualisation consiste à prendre un compact (une figure simple) de départ. Ensuite, on applique chacune des transformations, et la nouvelle figure est l'union des trois figures. On recommence ensuite à appliquer les trois transformations et ce de manière récursive jusqu'à ce que les figures de départ soient suffisamment petites. Illustration :

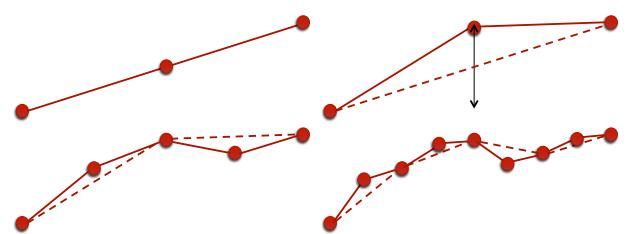


Au final, on obtient ce genre de figure :



## 6.4 Subdivision fractale

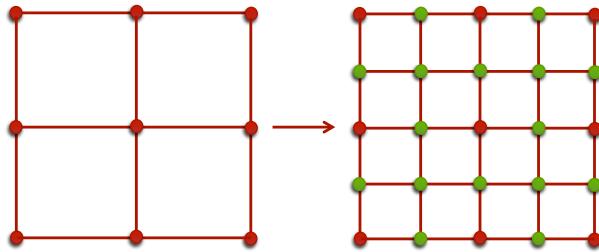
Un algorithme très classique de subdivision fractale s'appelle *mid-point* subdivision. Le principe sur une courbe est de partir d'un segment, de le couper en deux parties égales. Ensuite, on déplace le point créé d'une valeur aléatoire de variance  $s$  et on applique récursivement l'algorithme sur les deux segments créés avec une variance  $\lambda s$  (où  $\lambda \in [0, 1]$ ). Cet algorithme est très simple à implémenter. En voici une illustration :



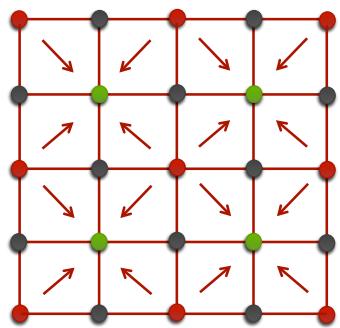
## 6.5 Diamond-square

L'algorithme *diamond-square* est une généralisation de l'algorithme précédent. Il permet de générer une image de profondeur (matrice) de manière itérative. Chaque itération va faire passer la matrice d'une taille  $m \times n$  à une taille  $(2m - 1) \times (2n - 1)$ . On ajoute donc  $m - 1$  lignes et  $n - 1$  colonnes dans les intervalles.

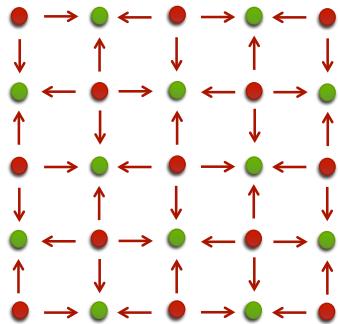
Exemple, de  $3 \times 3$ , on passe à  $5 \times 5$  :



Les nouveaux points sont calculés en faisant la moyenne de leurs voisins et en ajoutant une valeur aléatoire comme pour la subdivision *mid-point*.  
L'originalité de l'algorithme est de procéder en deux passes. La première passe est dite *diamond* :



On fait la moyenne des 4 voisins à laquelle on ajoute une valeur aléatoire.  
La deuxième passe est dite *square* :



Ici, on fait la moyenne des 4 voisins (ou 3 sur les bords) à laquelle on ajoute encore une valeur aléatoire. L'algorithme est itéré jusqu'à la précision voulue.



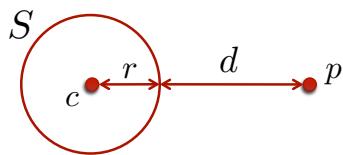
## CHAPITRE 7

### BOÎTE À OUTILS MATHÉMATIQUE

#### 7.1 Distances

Dans tout le chapitre on considère que  $\|u\| = 1$ .

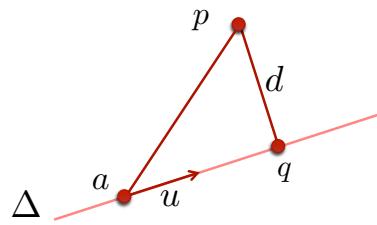
##### ► Point-sphère



$$d(p, S) = \begin{cases} 0 & \text{si } \|p - c\| \leq r \\ \|p - c\| - r & \text{sinon} \end{cases}$$

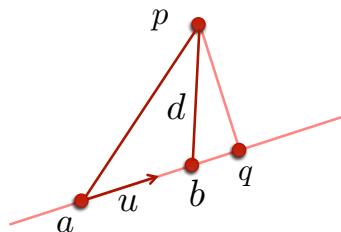
##### ► Point-droite

On applique le théorème de Pythagore au triangle  $apq$ .



$$d^2(p, \Delta) = \|p - a\|^2 - ((p - a).u)^2$$

##### ► Point-segment

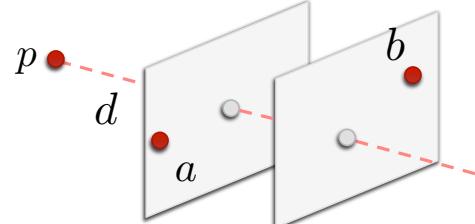


On définit  $l = (p - a).u$  et :

$$d(p, [ab]) = \begin{cases} \|p - a\| & \text{si } l \leq 0 \\ d(p, (ab)) & \text{si } 0 < l < \|b - a\| \\ \|p - b\| & \text{si } l \geq \|b - a\| \end{cases}$$

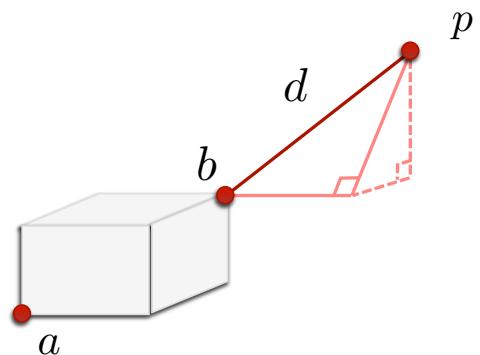
##### ► Point-boîte

On commence par faire la distance d'un point  $p$  à une plaque passant par  $a$  et  $b$  suivant un axe  $x, y$  ou  $z$ .



$$d(p, S_{ab}^x) = \begin{cases} 0 & \text{si } a_x \leq p_x \leq b_x \\ a_x - p_x & \text{si } p_x < a_x \\ p_x - b_x & \text{si } p_x > b_x \end{cases}$$

On combine ensuite les trois composantes :



$$d^2(p, B_{ab}) = d^2(p, S_{ab}^x) + d^2(p, S_{ab}^y) + d^2(p, S_{ab}^z)$$

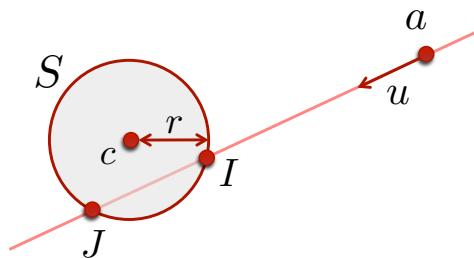
## 7.2 Intersections

Le principe est d'injecter une équation paramétrique dans une formulation implicite puis de résoudre l'équation. Dans le cas d'une intersection d'une droite avec un autre objet, on va utiliser l'équation de la droite suivante (passant par  $a$  et de vecteur directeur  $u$ ) :

$$p(t) = a + tu$$

### ► Droite-sphère

On utilise la formulation implicite de la sphère :



$$\|p - c\|^2 = r^2$$

Dans laquelle on injecte celle de la droite paramétrique :

$$(p(t) - c)^2 - r^2 = 0$$

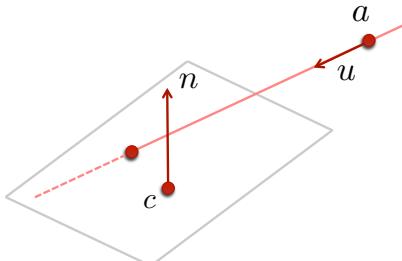
qui donne :

$$u^2t^2 + 2(a - c).ut + (a - c)^2 - r^2 = 0$$

qui est une équation du second degré en  $t$ . La résolution donnera 0, une ou deux solutions suivant le cas de figure.

*Remarque : il existe une autre méthode géométrique qui consiste à calculer la distance de  $c$  à la droite*

### ► Droite-plan



L'équation du plan est :

$$(p - c).n = 0$$

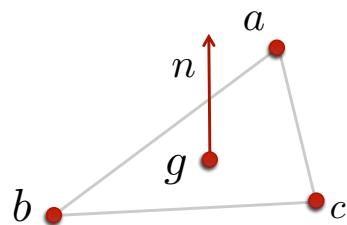
qui mène à l'équation du premier degré suivante :

$$u.n t + (a - c).n = 0$$

Lorsque  $u.n = 0$ , on a un cas dégénéré, la droite est parallèle au plan. Dans ce cas là, tous les points de la droite sont solution (si  $(a - c).n = 0$  aussi donc  $a$  appartient au plan) ou alors il n'y a pas de solution.

## 7.3 Triangles

### ► Normale



$$n = (b - a) \wedge (c - a)$$

### ► Centre de gravité

C'est le barycentre des trois points.

$$g = (a + b + c)/3$$

### ► Aire

La moitié de la norme de la normale :

$$A = \frac{1}{2}\|n\| = \frac{1}{2}\|(b - a) \wedge (c - a)\|$$

## CHAPITRE 8

### AFFICHAGE

### 8.1 Introduction

Le pipeline de modélisation que nous avons étudié permet de calculer les coordonnées de points dans l'espace écran.

Mais cela ne suffit pas pour afficher une scène. Il nous manque en effet le remplissage des formes géométriques et la couleur d'un point en fonction des paramètres de la scène.

Il existe fondamentalement deux grandes manières de faire pour effectuer le remplissage :

1. Lancer de rayon. On prend chacun des pixels de l'image résultat et on lance un rayon depuis l'œil vers ce pixel en direction de la scène. L'objet de plus proche est celui dont la couleur sera affichée (en version simplifiée).
2. Affichage direct. On décompose chaque objet en primitives simples (triangles ou quadrangles) dont on fournit les coordonnées à un système d'affichage. C'est le système d'affichage qui va lui-même décomposer chaque primitive en pixels.

### 8.2 Lancer de rayon

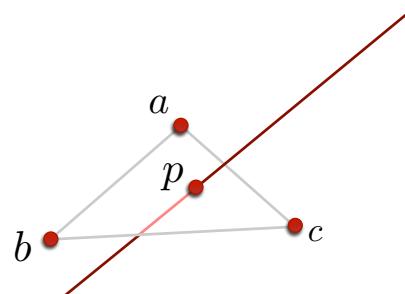
Le principe du lancer de rayon est de tester l'intersection avec tous les objets de la scène. L'objet qui sera visible à cet endroit sera celui pour lequel l'intersection est la plus proche.

Un rayon est caractérisé par un point et un vecteur de direction. La seule chose à implémenter pour chaque type d'objet est la fonction d'intersection.

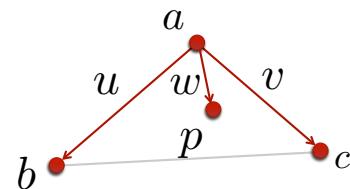
#### ► Intersections

L'intersection d'une droite (donc un rayon) avec une sphère et un plan a déjà été abordée dans le chapitre précédent.

Pour un triangle, on commence par calculer le point d'intersection avec le plan du triangle. Ensuite, on exprime le point d'intersection en fonction des vecteurs  $ab$  et  $ac$  c'est-à-dire  $p = a + s(b - a) + t(c - a)$ .



Pour cela on prendra les notation suivantes :



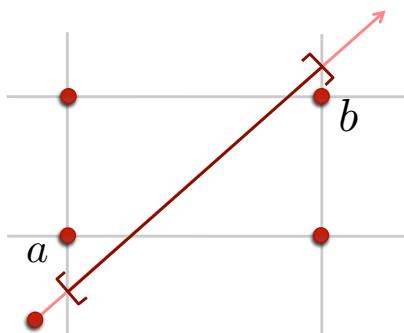
On commence par calculer  $s$  et  $t$ .

$$\left\{ \begin{array}{l} s = \frac{(u.v)(w.u) - (u.u)(w.v)}{(u.v)^2 - (u.u)(v.v)} \\ t = \frac{(u.v)(w.v) - (v.v)(w.u)}{(u.v)^2 - (u.u)(v.v)} \end{array} \right.$$

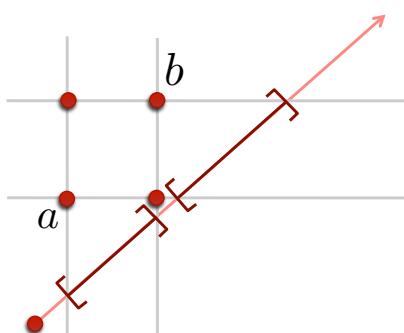
Ensuite, pour savoir si le point  $p$  appartient au triangle, il suffit de faire le test suivant :

$$\left\{ \begin{array}{l} s \geq 0 \\ t \geq 0 \\ s + t \leq 1 \end{array} \right.$$

Pour calculer l'intersection avec une boîte, on va utiliser les intersections avec les plans qui composent cette boîte. Pour chacun des axes, cela va donner un intervalle  $t_{\min}$  et  $t_{\max}$ . Il faut ensuite faire l'intersection (au sens ensembliste) des intervalles. Illustration en 2D :



Si l'intersection apparaît vide (avant même d'avoir étudié toutes les coordonnées), alors on sait que le rayon ne touche pas la boîte :



## ► Accélérations

Le gros problème du lancer de rayon est sa lenteur. En effet, pour chaque rayons (et il y en a autant que de pixels), il faut tester l'intersection avec tous les objets de la scène...

Il existe plusieurs méthodes d'accélération :

- Hiérarchie de volumes englobants
- Subdivisions de l'espace

Il existe plusieurs types de volumes englobants :

- AABB : Axis-Aligned Bounding Box
- Sphère
- Polygone convexe (enveloppe convexe)

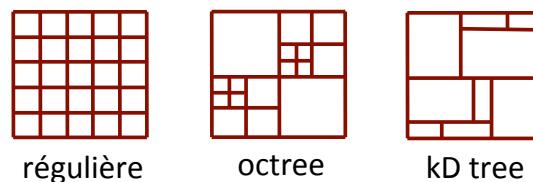
Chaque nœud de l'arbre contient les informations d'englobant, et chaque feuille décrit un ou plusieurs objets de la scène. Si on détecte une intersection avec le volume englobant, on teste les nœuds fils, sinon, on sait que l'on peut s'arrêter et on a économisé beaucoup de temps...

Il existe trois stratégies de construction de la hiérarchie de volumes englobants :

- Top-down : on part d'un volume contenant tous les objets et on subdivise pour minimiser un coût
- Bottom-up : on commence par créer les feuilles et on les regroupe jusqu'à un seul nœud racine
- Insertion : méthode incrémentale (on ajoute les objets un par un). On décide à la volée s'il faut créer un nouveau nœud et à quel endroit l'insérer. L'avantage est qu'il n'y a pas besoin de connaître tous les objets dès le début de l'algorithme (aspect dynamique)

Une autre méthode d'accélération est de subdiviser l'espace. Cette méthode se rapproche de la stratégie top-down des volumes englobants mais on cherche à chaque subdivision à faire une tessellation (l'union des sous-parties est la partie). Il existe plusieurs manières de subdiviser l'espace, certaines sont plus efficaces que d'autres dans la discrimination des objets mais sont souvent plus difficiles à calculer.

Il existe des subdivisions qui conservent l'alignement aux axes principaux :



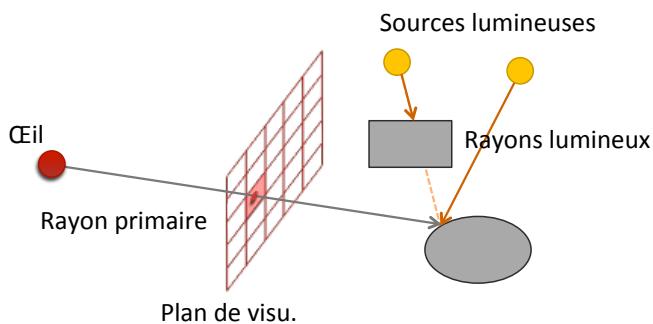
Il est possible aussi de ne pas conserver cette propriété :



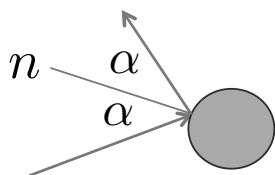
BSP-tree

## ► Calcul d'illumination

Un rayon est lancé depuis la position de l'œil vers un pixel de l'écran. Vers l'intersection la plus proche trouvée, on lance des rayons lumineux depuis toutes les sources de lumière de la scène. On cumule ensuite les valeurs de radiance pour chacune des sources :

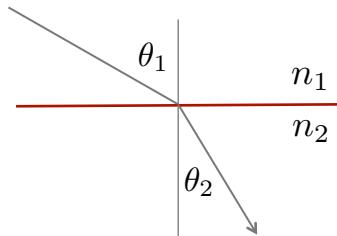


Si le matériau est un miroir, on va effectuer un rebond du rayon suivant la normale à l'objet.

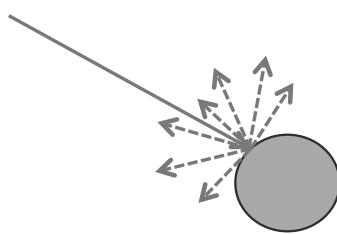


Si l'objet est transparent mais avec un changement d'indice (réfraction), on applique la loi de Descartes :

$$n_1 \sin \theta_1 = n_2 \sin \theta_2$$



Si on veut calculer de manière précise l'illumination, il faut lancer depuis le point d'intersection avec l'objet plusieurs autres rayons de manière stochastique et cumuler les radiances obtenues par chacun d'entre eux.



Cette méthode est très couteuse mais permet de simuler des comportements très complexes (ombres douces, sources de lumière non ponctuelles, etc.).

Remarque : les modèles d'illumination seront étudiés en détail dans le chapitre suivant.

## 8.3 Affichage direct

### ► Principe

Ce type d'affichage est adapté à une application rapide pour du temps-réel (jeux vidéo par exemple). L'utilisateur modélise la scène avec :

- Des informations de perspective et projection
- Des informations générales sur l'éclairage
- Des primitives simples de type triangle dans une pile de transformations

### ► Pile de transformations

À chaque instant, le système possède une transformation courante (sous la forme d'une matrice homogène  $4 \times 4$ ).

Il existe une série de fonctions qui permettent de modifier cette transformation (rotations, translations, mises à l'échelle). Pour ne pas avoir à appliquer des transformations inverse afin de revenir à un état précédent, l'utilisateur a la possibilité de sauvegarder l'état courant dans la pile (push) et de le restaurer en écrasant l'état courant (pop).

Par exemple, on peut imaginer qu'on a codé une fonction `Affichage` permettant l'affichage d'un objet centré sur l'origine. Pour afficher cet objet à plusieurs endroits il suffit de faire ces opérations :

- Charger la matrice identité
- push
- Translation1
- Affichage
- pop
- push
- Translation2
- Affichage
- pop
- etc.

### ► Affichage de primitive

Une primitive simple (triangle) est fournie au système d'affichage en coordonnées monde + matrice de modélisation/perspective/projection. On peut calculer facilement en coordonnées écran les positions de chaque sommet du triangle mais comment remplir l'intérieur du triangle ?

C'est ce que l'on appelle la *rasterisation*.

## ► Algorithme de Bresenham

Avant de commencer à afficher un triangle, on va commencer par voir comment tracer une ligne joignant deux pixels. L'astuce va consister à se placer dans un octant de référence pour lequel on aura toujours une pente dans l'intervalle  $[0, 1]$ .

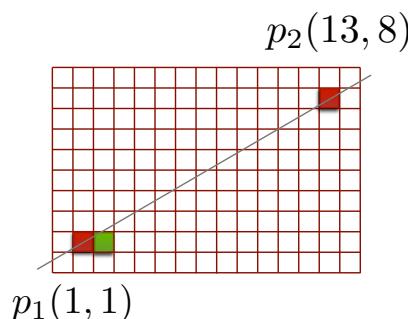
Ensuite, l'algorithme va incrémenter  $x$ . Si l'erreur introduite au niveau de  $y$  par ce déplacement est supérieure à  $\frac{1}{2}$ , alors on incrémentera aussi  $y$ . L'étape suivante consiste à reboucler jusqu'à ce que le point final soit atteint.

La pente est définie comme suit :

$$p = \frac{y_2 - y_1}{x_2 - x_1}$$

Lorsque  $x$  augmente de 1,  $y$  augmente de la valeur de la pente. L'erreur introduite par l'incrémentation de  $x$  est donc égale à cette pente si on ne touche pas à  $y$ . Cette erreur correspond géométriquement au décalage entre la droite et le centre du pixel.

Exemple :



La pente :

$$p = \frac{8 - 1}{13 - 1} = \frac{7}{12}$$

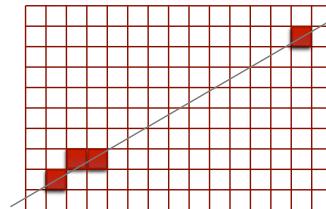
On commence par incrémenter  $x$  et voir l'erreur introduite :

$$\begin{cases} q_{try} = (2, 1) \\ e = \frac{7}{12} \end{cases}$$

L'erreur introduite est supérieure à  $\frac{1}{2}$ , il faut incrémenter  $y$  aussi.

$$\begin{cases} q_1 = (2, 2) \\ e = -\frac{5}{12} \end{cases}$$

Le prochain décalage en  $x$  n'aura pas besoin de décalage en  $y$  :

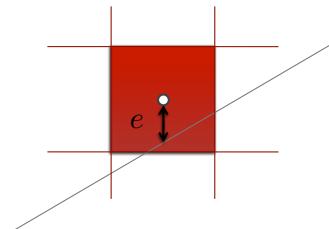


$$\begin{cases} q_2 = (3, 2) \\ e = \frac{2}{12} \end{cases}$$

Et voici le résultat final :

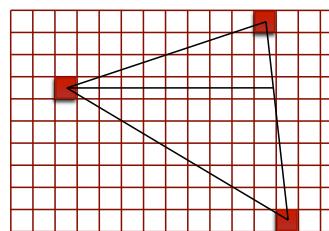


La valeur de  $e$  correspond à l'écart vertical (en pixel) entre le centre du pixel et la droite à tracer :

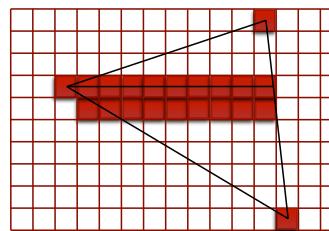


## ► Remplissage de triangle

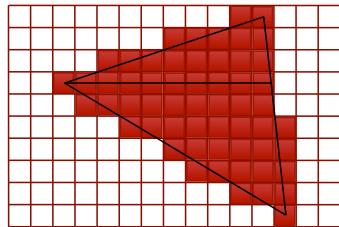
Pour remplir un triangle, on commence par le décomposer en deux triangles de cette manière :



Ensuite, on utilise Bresenham pour s'appuyer sur un des côtés du triangle :



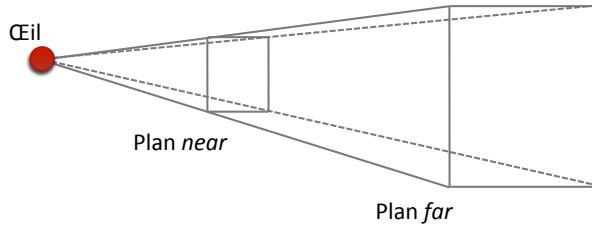
Et voici le résultat final :



Cet algorithme est très simple et hautement parallélisable.

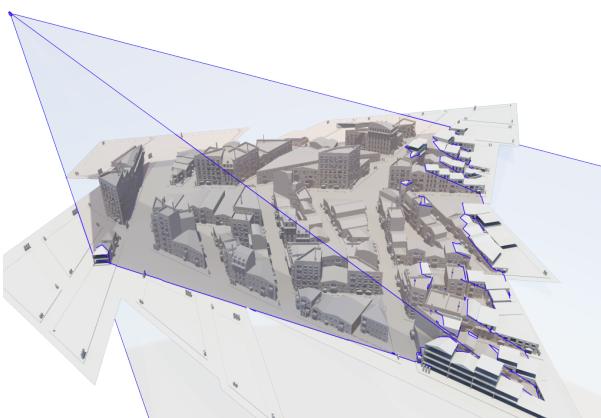
### ► Visibilité

Il ne sert à rien d'afficher des primitives qui sont hors du champ de la caméra. Pour cela, on modélise un volume visuel lié à la caméra appelé cône ou pyramide de vision (*view frustum* en anglais).



Les plans *near* et *far* indiquent les profondeurs minimum et maximum que l'on souhaite visualiser. Ce volume, une fois qu'il a subit la transformation perspective, est un pavé. Le test d'appartenance d'un point devient alors trivial.

Dans cet exemple, seuls les objets dont une partie est dans le cône de vision ont été représentés (d'un autre point de vue) :



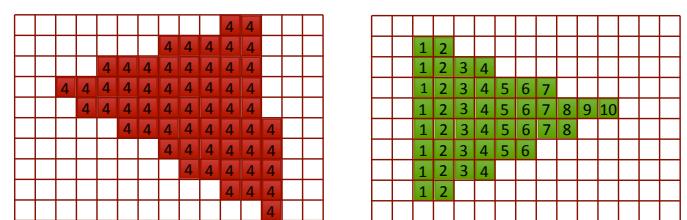
Un autre problème lorsque l'on utilise une méthode d'affichage direct est que l'on manipule des primitives de manière séquentielle. Si plusieurs primitives

ont des pixels communs, il faut afficher la primitive la plus proche de la caméra, et ce pour chaque pixel. Il existe plusieurs méthodes pour parvenir à afficher les bons pixels :

- Algorithme du peintre. On commence par classer les primitives par ordre de profondeur. Ensuite, on les affiche dans l'ordre inverse. Cela ne permet pas l'entrecroisement de deux primitives (chacune est entièrement devant ou derrière).
- Z-buffer. Un tampon de profondeur de la même taille que l'image est créé. Chaque fois qu'un pixel est plus proche on met à jour le tampon et on affiche ce pixel dans l'image finale.

### ► Z-buffer

Voici une illustration de l'algorithme avec deux triangles (les profondeurs de chaque pixel sont indiquées).



L'affichage du premier triangle va remplir les pixels de l'image et les profondeurs dans le z-buffer :

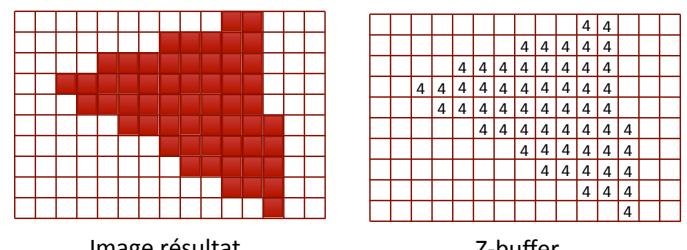
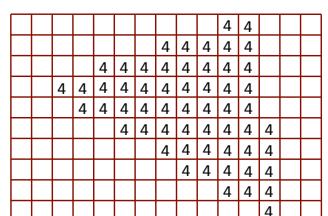


Image résultat



Z-buffer

Pour l'affichage du deuxième triangle, les pixels dont le z-buffer était vide ou supérieur sont affichés (et le z-buffer est mis à jour) :

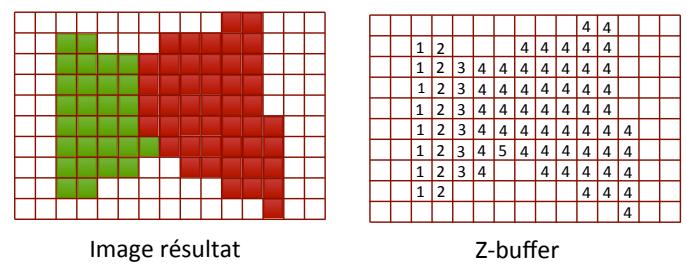
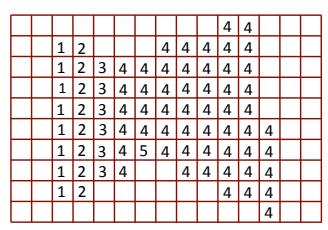


Image résultat



Z-buffer

Lorsque les valeurs de profondeur sont égales, c'est le pixel de la dernière primitive qui est affiché. Avec les cartes graphiques actuelles qui parallélisent fortement l'affichage des pixels, si deux primitives sont dans un même plan, le pixel de l'une ou l'autre sera affiché (c'est très moche et à éviter). Ce phénomène est appelé *z-fight*.

## CHAPITRE 9

### ILLUMINATION

#### 9.1 Introduction

L'éclairage est un phénomène complexe qui contribue largement à la qualité de l'image produite.

Divers paramètres contribuent à l'éclairage :

- Éclairage direct : la source de lumière éclaire une surface que l'œil voit de manière directe.
- Ombre : la source de lumière n'atteint pas certaines parties d'un objet.
- Réflexion : une partie de la lumière incidente est absorbée par le matériaux, une autre est réfléchie.
- Milieux participants : phénomène de réflexion volumique (fumées, nuages, *etc.*).

Certains de ces phénomènes dépendent de la longueur d'onde, ce qui complexifie encore un peu le problème.

- Quelconque. La distribution de la réflexion est complexe car le matériau a des propriétés complexes.

#### ► Effets

On parle de caustiques lorsque des taches lumineuses apparaissent par accumulation de lumière :



#### ► Types d'ombres

On parle d'ombres dures lorsque le contour de l'ombre est très précis. Ceci est visible lorsque la source de lumière est ponctuelle (source de lumière idéale). Les ombres douces quant à elles laissent apparaître des contours plus flous car les sources de lumières sont étendues.

Bien entendu, dans la réalité, les sources de lumière sont rarement ponctuelles (même le soleil).

Lorsqu'un matériau d'une certaine couleur reflète la lumière est éclairé à nouveau un autre objet, on a un phénomène de diffusion de couleur. C'est un phénomène assez subtil et peu perceptible.

Les milieux participants (volumiques) permettent de mettre en œuvre des effets volumiques. C'est le cas lorsque des particules sont en suspension, ou lorsqu'il y a la présence de poussières ainsi que dans des matériaux translucides (mais pas totalement) :



#### ► Types de réflexions

On distingue trois types de réflexions :

- Réflexion diffuse. Le rayon incident est reflété dans toutes les directions de manière identique.
- Spéculaire. Le rayon est reflété dans la direction symétrique à la normale.

## 9.2 Illumination globale

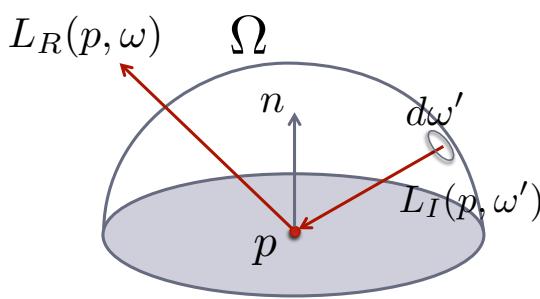
Nous allons maintenant aborder l'équation générale de l'illumination globale :

$$L_R(p, \omega) : \int_{\Omega} f_R(p, \omega', \omega) L_I(p, \omega') \cos \theta_I d\omega'$$

où :

- $L_R$  est la radiance (quantité de lumière) émise
- $f_R$  est la fonction BRDF<sup>1</sup>
- $L_I$  est la radiance incidente
- $\theta_i$  est l'angle incident
- $d\omega'$  est l'angle solide incident considéré

Illustration avec le schéma suivant :



Pour chaque direction émise, il faut prendre en compte toutes les directions incidentes (intégrale sur toute la demi-sphère).

La BRDF varie aussi en fonction du point sélectionné !

Mais la vraie complexité du calcul de l'illumination globale vient du fait que chaque direction incidente est aussi issue de la même équation. Certaines directions de rayons incidents sont donc aussi des directions de rayons émis. La solution générale est donc extrêmement complexe. Les algorithmes de lancer de rayons permettent d'en faire une approximation mais au prix de calculs très coûteux.

Nous allons donc voir dans la suite de ce chapitre comment simplifier cette équation afin de rendre les calculs moins complexes et tout simplement solubles.

## 9.3 Illumination simplifiée

### ► Ombrage de Phong

Ce modèle est une simplification de l'illumination basée sur des observations (modèle empirique). Il n'y

1. Bidirectional Reflectance Distribution Function

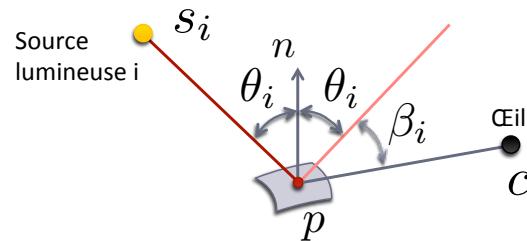
a donc pas de réalité physique derrière ce modèle. Cependant, les résultats obtenus sont assez convaincants et sont encore très utilisés. Ce modèle ne permet pas de modéliser les matériaux complexes.

Le calcul de l'illumination avec le modèle d'ombrage de Phong est la somme de trois composantes :

- Composante ambiante
- Composante diffuse (reflets dans toutes les directions)
- Composante spéculaire (reflets dans la direction symétrique à la normale)

Ce type d'illumination ne dépend que des sources de lumières directes et ne modélise pas les rebonds successifs de la lumière.

Dans la suite des explications, on utilisera les notations suivantes :



$\theta_i$  est l'angle que fait la source de lumière  $i$  avec la normale à la surface.

$\beta_i$  est l'angle que fait le rayon avec le symétrique de la source lumineuse  $i$  par rapport à la normale.

### Composante ambiante

Dans le modèle de Phong, c'est probablement la plus grosse approximation. Cela consiste à considérer que la radiance due aux rebonds successifs de la lumière sur toutes les parois de la scène est constante. Pour chaque matériau, on définit un coefficient multiplicateur  $k_a$  qui va moduler la quantité de lumière ambiante  $I_a$  :

$$L_a = I_a k_a$$

### Composante diffuse

Pour chaque source lumineuse, la lumière est diffusée dans toutes les directions (du demi-espace délimité par le plan tangent). Un coefficient multiplicateur en  $\cos \theta_i$  est inséré où  $\theta_i$  est l'angle incident. Plus la lumière arrive dans la direction de la normale, plus ce coefficient est grand. Inversement, lorsque la lumière est rasante, le coefficient tend vers 0. Pour

chaque matériau, on a aussi un coefficient multiplicateur  $k_d$ .

$$L_d = k_d \sum_i I_d^i \cos \theta_i$$

### Composante spéculaire

Pour chaque source lumineuse, la lumière est reflétée dans la direction symétrique à la normale. Pour obtenir une tâche plutôt qu'un seul point de reflet un coefficient en  $\cos^\alpha \beta_i$  est appliqué.

Pour chaque matériau, on applique un coefficient multiplicateur  $k_s$ .

$$L_s = k_s \sum_i I_s^i \cos^\alpha \beta_i$$

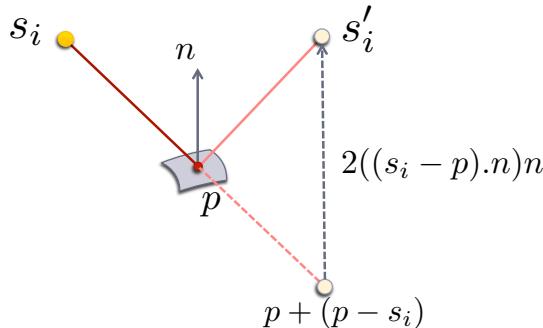
### Formule complète

Pour obtenir la formule complète, on peut calculer les cosinus avec un produit scalaire :

$$\begin{aligned} L = & I_a k_a \\ & + k_d \sum_i I_d^i \frac{s_i - p}{\|s_i - p\|} \cdot n \\ & + k_s \sum_i I_s^i \left( \frac{c - p}{\|c - p\|} \cdot \frac{s'_i - p}{\|s'_i - p\|} \right)^\alpha \end{aligned}$$

où  $s'_i$  est le symétrique de  $s_i$  par rapport à la droite  $(p, n)$  :

$$s'_i = p + (p - s_i) + 2((s_i - p) \cdot n)n$$



La radiance peut être calculée dans un espace de couleur (RVB par exemple). Dans ce cas-là les intensités des sources lumineuses pourront être exprimées dans cet espace couleur (pour simuler un éclairage de couleur). On pourra aussi indiquer des coefficients diffus et spéculaire qui dépendent de la couleur pour simuler la couleur du matériau (cette couleur peut dépendre de la position dans le matériau grâce à une texture).

### ► Méthodes d'interpolation

Les objets sont souvent décrits par des primitives de type polygone. Les calculs précédents permettent de connaître l'illumination en un point d'un matériau en fonction des sources lumineuses, des caractéristiques du matériau ainsi que de la normale à la surface. On connaît la normale aux sommets des polygones, mais comment calculer l'illumination entre ces sommets ? Il faut recourir à des méthodes d'interpolation.

#### Interpolation de Gouraud

C'est historiquement le premier mais aussi le plus simple. Cette méthode commence par un calcul de l'illumination à chaque sommet de l'objet. Ensuite, on fait une interpolation linéaire sur chaque arête et au moment de la *rasterisation* sur chaque ligne. Avec ce type d'interpolation, il est quasiment impossible d'avoir les effets spéculaires (reflets). Il faudrait en effet beaucoup de chance pour qu'un sommet tombe exactement dans la direction de la réflexion. Et quand bien même ce serait le cas, la forme du reflet sera forcément imprécise.

#### Interpolation de Phong

L'auteur est le même mais il s'agit bien d'autre chose que l'ombrage éponyme. Les deux peuvent d'ailleurs être utilisés conjointement.

L'interpolation de Phong est plus complexe à calculer mais beaucoup plus réaliste.

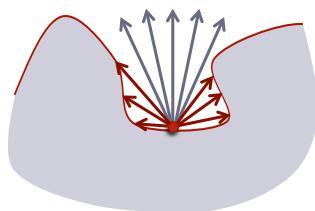
La méthode consiste à faire une interpolation linéaire sur tous les pixels du triangle de la normale. En pratique, il faut donc que chaque sommet du triangle ait une normale différente pour que l'effet soit visible. Les effets spéculaires sont maintenant visibles car l'interpolation des normales fera apparaître l'angle de réflexion.

### ► Lumière ambiante

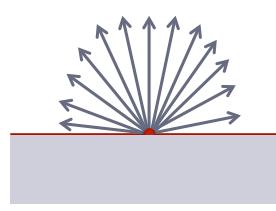
Un des gros soucis du modèle d'ombrage de Phong est son approximation de l'illumination ambiante qui est considérée comme constante. Pour pallier à ce défaut, on a parfois recours à l'occlusion ambiante (*ambient occlusion* en anglais). Cette méthode bien que donnant des résultats très réalistes n'est pas pour autant exacte. On notera au passage que pour

avoir des résultats exacts ou presque, on ne pourra échapper aux méthodes d'illumination globales qui sont très coûteuses.

L'idée de l'occlusion ambiante est simple : plus on se trouve dans un recoin, moins la lumière ambiante va pouvoir s'y aventurer. On effectue alors un calcul du degré d'occlusion d'une paroi par rapport au reste de la scène. Cette information est assez locale (on peut borner la recherche à une certaine distance) :



Occlusion ambiante forte



Occlusion ambiante nulle

On utilise ensuite cette information pour modifier le terme d'éclairage ambiant. Les résultats sont impressionnantes sans forcément avoir à recourir à de l'illumination globale.



Lorsque l'on veut aller encore plus vite, on peut faire ces calculs directement dans l'espace de l'écran (SSAO : *Screen Space Ambient Occlusion*). On utilise alors le z-buffer pour avoir une idée de la géométrie locale. L'approximation est encore plus grossière mais offre la présence d'effets réalistes quand même. Ce principe est très utilisé dans le domaine du jeu vidéo.

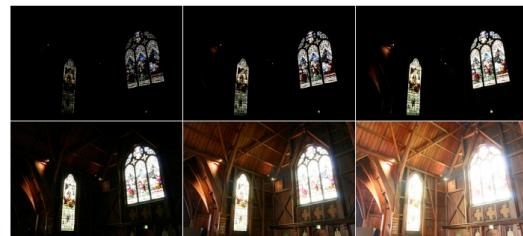
## 9.4 Calcul de couleur

Les algorithmes étudiés jusqu'à présent permettent d'obtenir une valeur d'illumination (radiance) qui n'est pas bornée car il s'agit d'une énergie.

Le problème qui se pose alors est qu'un terminal de visualisation ne permet pas d'avoir une dynamique

aussi étendue (sauf à y mettre une très grosse somme d'argent). Il faut au final afficher une couleur avec une dynamique bornée. Le standard d'un écran est 8 bits par composante RVB. Afin de faire l'affichage de la radiance, il faut avoir recours à un algorithme de *tone mapping*.

Ce problème est exactement le même que l'affichage d'une photo dite HDR (*High Dynamic Range*) prise avec plusieurs temps de pose :



Auteur : Dean S. Pemberton (CC)

Ce genre d'algorithme ne travaille en général que sur la composante luminance de l'image.

Il existe plusieurs algorithmes de *tone mapping* :

- Le plus simple est de faire du *clipping*, consistant à borner la valeur à une valeur maximum.
- On peut appliquer une fonction globale, par exemple la fonction suivante :

$$L' = \frac{L}{L+1}$$

(fonction qui est monotone croissante sur  $[0, \infty[$  avec des valeurs bornées à 1).

- On peut aussi utiliser une fonction qui s'adapte au comportement local de l'image (basé sur le gradient par exemple). Cette fonction peut ou non être basée sur le système de vision de l'œil humain. Les résultats sont en général assez impressionnantes mais manquent parfois de réalisme dans les zones à faible contraste.

## ► Logiciels de modélisation

On peut accéder assez facilement à deux logiciels de modélisation qui permettent de faire du rendu de haute qualité :

- Blender : projet open source.
- Autodesk Maya : une des références dans le domaine, dont la licence est gratuite pour les étudiants et les enseignants.

# CHAPITRE 10

## GÉNÉRATION PROCÉDURALE

### 10.1 Introduction

Les données 3D ainsi que les textures que l'on plaque dessus sont très gourmandes en mémoire. Le domaine de la génération procédurale vise à créer des algorithmes permettant de générer avec très peu d'information au départ une grande quantité de données avec si possible la présence de beaucoup de détails.

Les applications se situent dans la création de mondes virtuels. Nous allons passer en revue quelques techniques liées à la génération de textures procédurales ainsi que la génération de modèles 3D.

#### ► Générateur aléatoire

La première difficulté que l'on rencontre en matière de génération automatique est la reproductibilité des algorithmes que l'on utilise. La génération de beaucoup de données passe souvent par l'utilisation de générateurs aléatoires, mais il faut pouvoir reproduire deux fois la même chose. On peut pour cela avoir recours à l'utilisation d'un générateur standard avec une graine identique. Le souci, c'est qu'il faudra toujours exécuter les actions exactement dans le même ordre.

Une autre manière de faire est d'utiliser un générateur aléatoire déterministe. Cela peut paraître contradictoire mais en fait tous les générateurs aléatoires le sont. Pour cela on va donner un paramètre (spatial souvent) à une fonction dont le but va être de produire une valeur identique pour deux appels avec le même paramètre.

Exemple, une fonction dont la valeur dépend de la

position dans le plan :

$$\begin{aligned} f : \quad \mathbb{R}^2 &\rightarrow [0, 1] \\ (x, y) &\mapsto f(x, y) \end{aligned}$$

La distribution des valeurs de cette fonction suit par exemple une loi uniforme sur l'intervalle  $[0, 1]$ . Si on rappelle la fonction avec les mêmes valeurs de  $x$  et  $y$ , on obtient le même résultat. La fonction est donc bien déterministe.

Exemple de fonction satisfaisant ces contraintes :

```
float 2DNoise(int x, int y)
{
    long long n = x + y*57;
    n = (n<<13) ^ n;
    n = n*(n*n*15731+789221)+1376312589;
    return 1.0-((n*(n*n*15731+789221)+1376312589)&0X7fffffff)/1073741824.0;
}
```

On utilise la combinaison d'opérateurs arithmétiques ou sur les bits.

### 10.2 Textures

Une texture permet de donner une valeur de couleur en un point  $(x, y)$  ou plus généralement  $(u, v)$  en coordonnées de textures. Dans le cas d'une texture procédurale, la valeur de couleur (ou niveau de gris) est calculée par une procédure ou une fonction. Dans le cas idéal, cette fonction sera dépourvue de mémoire (exécutable pur).

Dans le domaine de la génération procédurale de textures, on utilise souvent des briques de base que l'on appelle de façon imbriquée (équivalent à des compositions au sens mathématique).

La fonction que nous avons étudiée précédemment 2DNoise est un exemple de brique de base.

## ► Interpolation

Une autre brique de base est celle qui permet d'utiliser une fonction qui normalement ne marche que dans le domaine discret dans un domaine continu : la fonction d'interpolation.

Si on veut uniquement interpoler dans une dimension, la fonction peut ressembler à cela :

```
float Interpolate(float x,
    float y,
    float ratio) {
    return ratio*x+(1.0-ratio)*y;
}
```

Dans le cas d'une texture, on a besoin d'une interpolation 2D. On utilise souvent l'interpolation bilinéaire suivante :

```
float Interpolate2(
    float a00, float a10,
    float a01, float a11,
    float r1, float r2) {
return Interpolate(
    Interpolate(a00,a10,r1),
    Interpolate(a01,a11,r1),
    r2);
}
```

On interpose deux fois linéairement d'où le terme bilinéaire.

Si maintenant on veut disposer d'un bruit 2D comme tout à l'heure mais dans le domaine continu, alors on peut utiliser la fonction d'interpolation précédemment construite :

```
float 2DNoise(double x, double y) {
    int X = int(x);
    int Y = int(y);
    return Interpolate2(
        2DNoise(X,Y),2DNoise(X+1,Y),
        2DNoise(X,Y+1),2DNoise(X+1,Y+1),
        1.0-x+X,1.0-y+Y);
}
```

Le ratio est déduit de l'erreur d'arrondi.

## ► Perlin

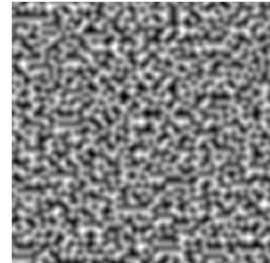
L'idée des textures de Perlin est de mélanger des bruits procéduraux à des fréquences différentes (notion d'octave). Les paramètres de ce bruit sont :

- $p$  : la persistance qui correspond à l'atténuation entre chaque octave
- $o$  : de nombre d'octaves

Exemple de code :

```
float PerlinNoise(double x, double y,
    double p, int o) {
    float n=0;
    for(int i=0;i<o;i++) {
        float f = float(1<<i);
        float a = pow(p,i);
        n += 2DNoise(x*f,y*f)*a;
    }
    return n;
}
```

Exemple de texture générée :



En général, la valeur du bruit n'est pas utilisée directement pour la texture mais est composée avec une fonction de couleur. Exemple de rendu :

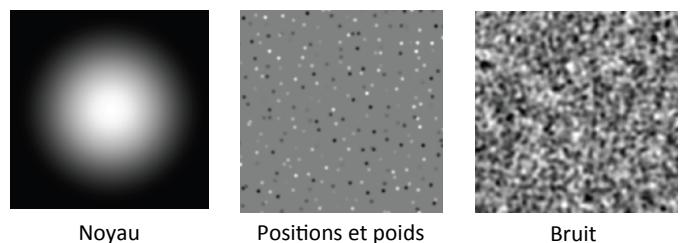


## ► Noyaux

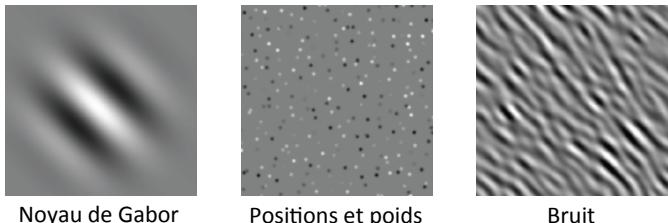
Une autre manière de créer un bruit est de sommer des contributions de noyaux :

$$N(x, y) = \sum_i w_i k(x - x_i, y - y_i)$$

Chaque noyau est identique mais pondéré par un poids  $w_i$  et positionné en  $(x_i, y_i)$ . Tout cela est assez bien adapté au procédural car il s'agit d'une somme locale de petites fonctions (sur un domaine borné). Exemple :



Lorsque l'on utilise un noyau qui possède une décomposition fréquentielle non symétrique, on obtient un bruit anisotrope :



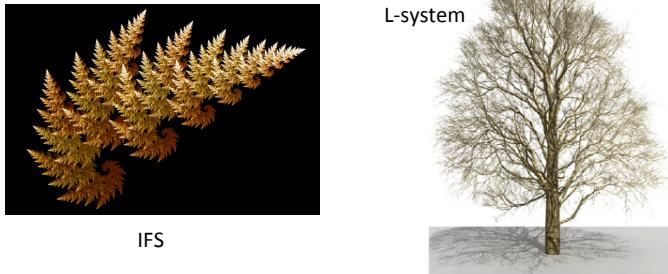
Noyau de Gabor

Positions et poids

Bruit

## 10.3 Contenu 3D

Les algorithmes procéduraux peuvent aussi servir à générer du contenu graphique. La modélisation itérative étudiée dans un chapitre précédent en est un parfait exemple :



Nous allons nous attarder sur les méthodes permettant de générer tout ce qui a trait aux villes :

- Plan de ville (rues, blocs, quartiers)
- Bâtiments
- Routes

Dans tous ces domaines, il n'y a pas vraiment de procédural pur (qui n'aurait pas de mémoire). Souvent, des pré-traitements sont nécessaires à la génération du contenu.

### ► Bâtiments

Diverses techniques à base de grammaires existent. Elles consistent à utiliser des règles de production avec des attributs géométriques. Par exemple, un bâtiment génère une façade, des murs de côté, un fond et un toit. La façade génère un rez-de-chaussée et plusieurs étages. Chaque étage génère un balcon, une fenêtre, etc. On appelle cela des *shape-grammar* ou *CGA-shape*.

Exemple de résultat :



Les bâtiments peuvent aussi être générés à l'aide d'assets (modèles) paramétrés. Cela passe par le codage de modèle géométriques complets sous forme de classes. Exemple : un pont est paramétré par ses deux extrémités (paramètres du constructeur de la classe), et tout le reste est totalement automatique.



### ► Plans de ville

Diverses techniques à base de champs de tenseur permettent de générer des plans de ville. Un tenseur est défini en tout point par interpolation de contraintes utilisateur. Ce champ donne la direction principale des rues.

Les rues secondaires sont ensuite générées grâce aux directions perpendiculaires.

## ► Routes

Le tracé automatique de routes peut se faire grâce à l'optimisation d'une fonction de coût anisotrope sur un graphe implicite (donc non stocké). Cette fonction de coût permet de prendre en compte tout un tas de paramètres tels que la pente, la présence de végétation, d'eau, la proximité avec d'autres routes existantes, *etc.*

