Contents

Chapter 1

Part1

Infographie pour la Science des Données

1.1 Introduction : Infographie et Science des Données

L'informatique graphique, ou infographie, est la discipline qui s'intéresse à la création, au traitement et à l'exploitation des images numériques à l'aide de l'informatique. Elle joue un rôle croissant dans la science des données, notamment pour la visualisation et l'interprétation de grands ensembles de données complexes.

Definition 1.1.1. Une **Scène 3D** est une représentation virtuelle d'un environnement tridimensionnel. Elle est composée de plusieurs éléments essentiels :

- **Objets 3D :** Les éléments géométriques qui peuplent la scène (personnages, bâtiments, objets divers).
- **Matériaux :** Les propriétés de surface des objets 3D qui déterminent comment ils interagissent avec la lumière (couleur, texture, brillance, transparence).
- **Lumières :** Les sources lumineuses qui éclairent la scène (directionnelles, ponctuelles, ambiantes).
- **Caméra :** Le point de vue virtuel à partir duquel la scène est observée et rendue en une image 2D.

1.2 Historique de l'Infographie

L'infographie a connu une évolution rapide depuis ses débuts.

1.2.1 Sketchpad (1963)

Ivan Sutherland développe le **Sketchpad**, considéré comme le premier système graphique interactif complet. Il permettait de :

- Sélectionner, pointer, dessiner, éditer des formes géométriques directement sur un écran.
- Utiliser des structures de données et des algorithmes spécifiques pour la manipulation graphique.
- Mettre en œuvre une modélisation hiérarchique des objets.
- Interagir via des menus contextuels (pop-up).

Lien vidéo : `https://youtu.be/6orsmFndx_o`

1.2.2 Affichage

L'évolution des technologies d'affichage a été cruciale :

- **Écran à affichage vectoriel :**
  - 1963 : Oscilloscope modifié utilisé pour les premiers affichages graphiques.
  - 1974 : "Picture System" d'Evans et Sutherland, un système d'affichage vectoriel avancé.
- **Écran à affichage raster :**
  - 1975 : Buffer de frames (mémoire tampon d'image) introduit par Evans et Sutherland, permettant l'affichage point par point (pixel).
  - Années 1980 : Ordinateurs personnels basés sur les bitmaps (images matricielles). L'Apple Macintosh (1984) et les moniteurs CRT (Cathode Ray Tube) en sont des exemples.
  - Années 1990 : Écrans LCD (Liquid-Crystal Displays) sur les ordinateurs portables.
  - Années 2000 : Appareils photo numériques et projecteurs à micro-miroirs.
- **Autres technologies :** Stéréo, casques de Réalité Virtuelle, affichages tactiles, haptiques, audio 3D.

1.2.3 Entrées (Inputs)

Les dispositifs permettant d'interagir avec les systèmes graphiques ont également évolué :

- **2D :** Stylo lumineux, tablette graphique, souris, joystick, trackball, écran tactile.
- **3D :** Traqueurs de position 3D, systèmes à multiples caméras, télémètres actifs (mesure de distance).
- **Autres :** Gants de données (tactiles), reconnaissance vocale, reconnaissance gestuelle.

1.2.4 Rendu (Rendering)

Le rendu est le processus de génération d'une image 2D à partir d'une scène 3D. Les techniques ont évolué pour plus de réalisme :

- **Années 1960 - Problème de visibilité :**
  - Algorithmes de lignes cachées : Roberts (1963), Appel (1967).
  - Algorithmes de surfaces cachées : Warnock (1969), Watkins (1970).
  - Tri par visibilité : Sutherland (1974).
- **Années 1970 - Graphiques en Raster :**
  - Lumière diffuse : Gouraud (1971).
  - Lumière spéculaire : Phong (1974).
  - Texture, surface courbe : Blinn (1974).
  - Z-buffer (tampon de profondeur) : Catmull (1974).
  - Anti-aliasing / anticrénelage : Crow (1977).
- **Début Années 1980 - Illumination globale :**
  - Ray tracing (lancer de rayons) : Whitted (1980).
  - Radiosité : Goral et al. (1984), Cohen (1985).
  - Équation de rendu : Kajiya (1986).
- **Fin Années 1980 - Photoréalisme :**

  - Arbres d'ombrage : Cook (1984).
  - Langage de shading (programmation des effets de surface) : Perlin (1985).
  - RenderMan : Hanrahan & Lawson (1990).

- **Début Années 1990 - Rendu de non-photoréalisme :**
  - Rendu des volumes : Drebin et al. (1988), Levoy (1988).
  - Peinture impressionniste : Haeberli (1990).
  - Illustration automatique à l'encre et stylo : Salesin et al. (1994-).
  - Rendu de peinture : Meier (1996).

- **Fin Années 1990 - Rendu basé sur images :**
  - Interpolation de points de vue : Chen & Williams (1993).
  - Modélisation plénoptique : McMillan & Bishop (1995).
  - Rendu des champs lumineux : Levoy & Hanrahan (1996).

1.2.5 Programmation

L'évolution matérielle et logicielle a permis des avancées majeures :

- **Depuis début Années 1980 - Cartes Graphiques :**
  - 1979 : Premier processeur programmable dédié au graphisme 3D (Clark).
  - 1982 : Création de Silicon Graphics, Adobe et AutoDesk.
  - 1987 : Première carte graphique grand public.
  - 1992 : OpenGL 1.0 (API graphique standard).
  - 1993 : Création de NVIDIA.
  - 1994 : VRML (Virtual Reality Modeling Language).
  - 1996 : DirectX de Microsoft.
  - 1997 : Java3D de Sun.
  - 2007 : OpenGL 3.0.

1.3 Applications de l'Infographie

L'infographie trouve des applications dans de nombreux domaines.

1.3.1 Divertissement

- **Films d'animation :** Création de mondes et personnages virtuels (ex: Shrek).
- **Effets spéciaux (VFX) :** Intégration d'éléments générés par ordinateur dans des prises de vues réelles.
- **Jeux vidéo :** Création d'environnements interactifs en temps réel.

1.3.2 Design industriel et Ingénierie assistée par ordinateur (CAE)

Modélisation, simulation et visualisation de produits avant leur fabrication (ex: conception automobile).

1.3.3 Supervision & Téléopération (online)

Contrôle à distance de systèmes (robots, véhicules) avec retour visuel graphique.

1.3.4 Simulateurs (offline)

- **Conduite/Pilotage :** Entraînement réaliste pour les pilotes et conducteurs.
- **Entraînement :** Simulation de procédures complexes (chirurgie, maintenance).
- **Jeux :** Simulateurs de vol, de course, etc.
- **Téléopération :** Préparation et simulation de missions à distance.

1.3.5 Visualisation de données

Représentation graphique de données complexes pour faciliter leur compréhension :

- **Médicales :** Imagerie 3D (scanner, IRM).
- **Géophysiques :** Modélisation du sous-sol.
- **Biologiques :** Visualisation de molécules, de structures cellulaires.
- **Dynamique des fluides (CFD) :** Simulation d'écoulements.

1.3.6 Navigation

Systèmes de cartographie 3D, aide à la navigation (ex: vues aériennes dans les GPS).

1.3.7 Art & Design

Outils de création numérique pour les artistes et designers.

1.3.8 Communications

Présentations visuelles, illustrations techniques, publicité.

1.4 Définitions Fondamentales

Definition 1.4.1. L'infographie est l'application de l'informatique à la création, au traitement, et à l'exploitation des images numériques .

Remark 1.4.2. Quelques points clés :

- Le mot "Infographie" est formé à partir de "INFOrmatique" et "GRAPHIque". C'est une appellation déposée par la société Benson en 1974.
- C'est un domaine interdisciplinaire faisant appel à la physique (lumière, optique), aux mathématiques (géométrie, algèbre linéaire), à la perception humaine, à l'interaction homme-machine, à l'ingénierie, à la conception graphique et à l'art.
- Distinction avec la **Vision par Ordinateur (Computer Vision)** : l'infographie génère des images à partir de descriptions (Input: description de scène -¿ Output: Image), tandis que la vision par ordinateur analyse des images pour en extraire des informations (Input: Image -¿ Output: description/interprétation).
- La **géométrie** est l'élément constitutif essentiel de toute image de synthèse. L'organisation spatiale des objets forme un **modèle géométrique** complet de la scène virtuelle.

- L'image de synthèse résulte de l'**interaction** entre les différentes sources de lumière de la scène et les objets qui la composent.

Definition 1.4.3. En infographie, le terme **"modèle"** peut se référer à :

- **Modèle géométrique :** Modèle d'un objet à rendre dans une image, enrichi par divers attributs (couleur, texture, réflectance, etc.).
- **Modèle mathématique :** Modèle d'un processus physique ou informatique (par exemple, un modèle de calcul de la lumière qui se reflète sur des surfaces brillantes).

1.5 Pipeline Graphique

Le pipeline graphique est une séquence d'étapes conceptuelles permettant de transformer une description de scène 3D en une image 2D affichable à l'écran. Chaque primitive géométrique (triangle, point, ligne) de la scène passe successivement par ces étapes. Voici les étapes principales du pipeline graphique classique :

1. **Modèles 3D :** Description initiale des objets dans leurs propres systèmes de coordonnées.
2. **Transformations de modélisation :**
   - Application des transformations (translation, rotation, mise à l'échelle) pour positionner et orienter les objets les uns par rapport aux autres.
   - Passage du système de coordonnées local de chaque objet (Object Space) vers un repère global unique (World Space).
3. **Illumination (Shading) :**
   - Calcul de la couleur de chaque point visible des objets en fonction des propriétés de leurs matériaux, des sources de lumière et de la position de l'observateur.
   - Les modèles d'illumination sont souvent locaux (calcul par primitive, sans gestion des ombres portées initialement) : modèles diffus, ambiants, spéculaires (Gouraud, Phong, etc.).
4. **Transformation d'affichage (Viewing Transformation) :**
   - Passage des coordonnées du monde (World Space) aux coordonnées du point de vue (Eye Space ou Camera Space).
   - La scène est décrite par rapport à la position et à l'orientation de la caméra. Le repère est souvent aligné avec l'axe Z.
5. **Clipping (Découpage) :**
   - Élimination des parties de la scène qui se trouvent en dehors du volume de vision de la caméra (View Frustum).
   - Passage en coordonnées normalisées (NDC - Normalized Device Coordinates), généralement un cube où les coordonnées varient entre -1 et 1 ou 0 et 1.
   - Suppression des parties hors du volume de vision.
6. **Transformation écran (Projection) :**
   - Projection des primitives 3D (après clipping) sur un plan 2D, simulant ce que la caméra "voit".
   - Passage des coordonnées NDC vers l'espace image 2D (Screen Space), en pixels. Les coordonnées (left, right, bottom, top, near, far) du volume de vision sont mappées sur les dimensions (width, height) de la fenêtre d'affichage.

7. **Pixelisation (Rasterization) :**
    - Découpage des primitives 2D (projetées) en fragments, correspondant aux pixels de l'écran.
    - Interpolation des valeurs connues aux sommets (couleur, profondeur, coordonnées de texture, etc.) pour chaque fragment généré.
8. **Visibilité / Rendu Final :**
    - Détermination des fragments visibles pour chaque pixel (élimination des parties cachées, souvent via un Z-buffer).
    - Remplissage du frame buffer (mémoire image) avec la couleur finale des fragments visibles pour former l'image finale.

1.5.1 Systèmes de coordonnées

Le passage à travers le pipeline implique plusieurs changements de systèmes de coordonnées :

- **Repère objet (Object Space) :** Coordonnées locales à chaque objet.
- **Repère scène (World Space) :** Coordonnées globales communes à tous les objets.
- **Repère caméra (View/Eye Space) :** Coordonnées relatives à la caméra.
- **Repère caméra normalisé (NDC) :** Coordonnées normalisées après clipping (-1 à 1 ou 0 à 1).
- **Espace écran (Screen Space) :** Coordonnées en pixels sur l'écran.

1.5.2 Vue d'ensemble et Implémentation

Le pipeline graphique traite des :

- **Modèles géométriques :** Objets, surfaces, etc.
- **Modèle d'illumination :** Calcul des interactions lumineuses.
- **Caméra :** Point de vue et ouverture (frustum).
- **Fenêtre (viewport) :** Grille de pixels sur laquelle l'image est plaquée.
- **Couleurs :** Intensités convenant à l'afficheur (ex: 24 bits RVB).

Le pipeline peut être implémenté de diverses manière :

- Entièrement en **software** (logique configurable), sans carte graphique dédiée (lent).
- Avec des **cartes graphiques** (hardware) qui accélèrent certaines étapes :
    - *Première génération :* Hardware pour la rastérisation/pixelisation. Le reste en software configurable.
    - *Deuxième génération :* Hardware configurable pour la transformation/clipping et la rastérisation.
    - *Troisième génération :* Hardware programmable (GPU) pour la plupart des étapes, offrant une grande flexibilité (vertex/pixel shaders).
- À certaines étapes, des **outils de programmation** (ex: vertex program, pixel program/shader) permettent de personnaliser le traitement.

1.6 GPU (Graphics Processing Unit)

Le GPU est un processeur massivement parallèle spécialisé dans les calculs graphiques, mais de plus en plus utilisé pour des calculs généraux (GPGPU).

- **Matériel spécialisé :** Très performant pour certaines opérations graphiques (transformations de matrices, traitement de pixels/vertices). Son catalogue d'opérations natives est plus réduit que celui d'un CPU.
- **Parallélisme massif :** Conçu pour exécuter la même opération (le même code/shader) sur des millions de points de données (vertices, pixels) simultanément (architecture SIMD - Single Instruction, Multiple Data).
- **Efficacité :** Très efficace lorsque le même code est exécuté sur de grandes quantités de données homogènes.
- **Divergence :** Moins performant lorsque le flot d'exécution diverge (par exemple, si des pixels voisins doivent exécuter des branches différentes d'un 'if' dans un shader), car certaines unités de calcul deviennent inactives.

Un GPU typique contient de nombreuses unités d'exécution (Exec) organisées en groupes (SIMD/SIMT), des caches mémoire, des unités spécialisées (Texture, Tessellation, Clip/Cull, Rasterize, Z-buffer/Blend) et un ordonnanceur (Scheduler/Work Distributor) pour distribuer le travail, le tout connecté à une mémoire dédiée (GPU Memory).

1.7 Rappels Mathématiques Essentiels

1.7.1 Pourquoi les mathématiques ?

Remark 1.7.1. De nombreux concepts graphiques font appel aux mathématiques :

- Systèmes de coordonnées
- Transformations (rotation, translation, mise à l'échelle)
- Rayonnement (Ray-casting, lancer de rayons)
- Conversion des couleurs
- Tests d'intersection
- Requêtes géométriques
- Simulations physiques
- Et bien d'autres...

L'algèbre linéaire et l'analyse vectorielle sont particulièrement fondamentales.

1.7.2 Scalaire

Definition 1.7.2. Un **scalaire** est une grandeur totalement définie par un nombre (magnitude) et éventuellement une unité. Il n'a pas d'orientation dans l'espace.

Remark 1.7.3.

- **Exemples :** Masse, distance, température, volume, densité.
- **Propriétés :** Les scalaires obéissent aux lois de l'algèbre ordinaire.

- **Opérations élémentaires :** Addition et multiplication.
- **Propriétés des opérations :**
  - Commutativité : $\alpha + \beta = \beta + \alpha$; $\alpha \cdot \beta = \beta \cdot \alpha$
  - Associativité : $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$; $\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma$
  - Distributivité : $\alpha \cdot (\beta + \gamma) = (\alpha \cdot \beta) + (\alpha \cdot \gamma)$
- **Identité :**
  - Addition : $\alpha + 0 = 0 + \alpha = \alpha$ (0 est l'élément neutre)
  - Multiplication : $\alpha \cdot 1 = 1 \cdot \alpha = \alpha$ (1 est l'élément neutre)

1.7.3 Vecteur

Definition 1.7.4. Un **vecteur** est une entité mathématique définie par n valeurs numériques (composantes) extraites du même ensemble E (par exemple $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{C}$). Ces valeurs décrivent le module (longueur) et l'orientation du vecteur.

Remark 1.7.5.
- n est appelé la **dimension** du vecteur. On dit que le vecteur est défini dans E^n, qui est un espace vectoriel de dimension n.
- **Exemple :** Un vecteur dans $\mathbb{R}^3$ est noté $\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ ou (x, y, z).
- **Exemples physiques :** Déplacement, vitesse, accélération, force.
- Les vecteurs obéissent aux lois de l'**algèbre vectorielle**.

Vecteurs unitaires et base

Remark 1.7.6.
- Dans un repère, les vecteurs sont décrits dans une **base**, généralement composée de **vecteurs unitaires** (norme 1) orthogonaux entre eux.
- Dans $\mathbb{R}^3$, la base canonique est $(\vec{i}, \vec{j}, \vec{k})$, où $\vec{i} = (1, 0, 0)$, $\vec{j} = (0, 1, 0)$, $\vec{k} = (0, 0, 1)$.
- Un vecteur $\vec{V}$ est représenté par l'addition des vecteurs de base multipliés par ses **projections** (composantes) sur chaque axe :
$$\vec{V} = x'\vec{i} + y'\vec{j} + z'\vec{k}$$
où x', y', z' sont les composantes de $\vec{V}$ dans la base $(\vec{i}, \vec{j}, \vec{k})$.

Opérations sur les vecteurs

Remark 1.7.7.
- Opérations élémentaires
- Produit scalaire
- Produit vectoriel
- Addition de vecteurs
- Produit vecteur-scalaire
- Normalisation

1.7.4 Addition de deux vecteurs

Remark 1.7.8.
- **Géométriquement :** L'addition de deux vecteurs $\vec{A}$ et $\vec{B}$ obéit à la règle du parallélogramme (ou règle du triangle). Le vecteur résultant $\vec{A}+\vec{B}$ va de l'origine de $\vec{A}$ à l'extrémité de $\vec{B}$ lorsque $\vec{B}$ est placé à l'extrémité de $\vec{A}$.
- **Algébriquement :** Cela se fait en additionnant les composantes correspondantes. Si $\vec{A} = (A_x, A_y, A_z) = A_x\vec{i} + A_y\vec{j} + A_z\vec{k}$ et $\vec{B} = (B_x, B_y, B_z) = B_x\vec{i} + B_y\vec{j} + B_z\vec{k}$, alors :

$$\vec{A} + \vec{B} = (A_x + B_x)\vec{i} + (A_y + B_y)\vec{j} + (A_z + B_z)\vec{k}$$

$$\vec{A} + \vec{B} = (A_x + B_x, A_y + B_y, A_z + B_z)$$

Example 1.7.9 (Addition de composantes)**.** Un vecteur $\vec{A}$ peut être décomposé en ses composantes rectangulaires $A_x = A\cos\theta_A$ et $A_y = A\sin\theta_A$ (en 2D). Pour additionner $\vec{R} = \vec{A}+\vec{B}$: $R_x = A_x + B_x = A\cos\theta_A + B\cos\theta_B$ $R_y = A_y + B_y = A\sin\theta_A + B\sin\theta_B$ Le module de $\vec{R}$ est $R = \sqrt{R_x^2 + R_y^2}$ et son angle θ_R est tel que $\tan\theta_R = R_y/R_x$.

1.7.5 Norme d'un vecteur

Definition 1.7.10. La **norme** (ou module, longueur) d'un vecteur $\vec{V}$, notée $||\vec{V}||$ ou $|\vec{V}|$, est sa taille.

Remark 1.7.11.
- Elle est calculée par le théorème de Pythagore en utilisant ses composantes.
- En 2D, pour $\vec{V} = x'\vec{i} + y'\vec{j}$, $||\vec{V}|| = \sqrt{(x')^2 + (y')^2}$.
- En 3D, pour $\vec{V} = x'\vec{i} + y'\vec{j} + z'\vec{k}$, $||\vec{V}|| = \sqrt{(x')^2 + (y')^2 + (z')^2}$.
- En dimension n, on applique le même principe récursivement.
- La norme du vecteur $\vec{AB}$ (vecteur allant du point A au point B) est la distance entre A et B.

1.7.6 Normalisation d'un vecteur

Definition 1.7.12. La **normalisation** est le processus qui transforme un vecteur non nul en un vecteur unitaire (de norme 1) ayant la même direction et le même sens.

Remark 1.7.13.
- Pour normaliser un vecteur $\vec{V}$, il suffit de diviser chacune de ses composantes par sa norme $||\vec{V}||$.
- Le vecteur normalisé $\vec{v}$ est : $\vec{v} = \frac{\vec{V}}{||\vec{V}||}$.
- Si $\vec{V} = (x', y', z')$, alors $\vec{v} = \left(\frac{x'}{||\vec{V}||}, \frac{y'}{||\vec{V}||}, \frac{z'}{||\vec{V}||}\right)$, avec $||\vec{V}|| = \sqrt{(x')^2 + (y')^2 + (z')^2}$.

1.7.7 Produit scalaire (Dot Product)

Definition 1.7.14. Le **produit scalaire** de deux vecteurs $\vec{A}$ et $\vec{B}$, noté $\vec{A} \cdot \vec{B}$, est un scalaire.

Remark 1.7.15.

- **Définition géométrique :** C'est le produit des modules des deux vecteurs par le cosinus de l'angle θ entre eux :

$$\vec{A} \cdot \vec{B} = ||\vec{A}||||\vec{B}|| \cos\theta$$

  Il correspond aussi au produit du module de l'un par la projection du second sur le premier.

- **Définition par composantes :** C'est la somme des produits des composantes correspondantes :

$$\vec{A} \cdot \vec{B} = A_x B_x + A_y B_y + A_z B_z$$

- **Propriétés :**
  - Commutativité : $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$
  - Distributivité par l'addition : $(\vec{u} + \vec{v}) \cdot \vec{w} = \vec{u} \cdot \vec{w} + \vec{v} \cdot \vec{w}$
  - Distributivité par un scalaire k : $\vec{u} \cdot (k\vec{v}) = k(\vec{u} \cdot \vec{v})$
- **Signe et angle :** Le signe du produit scalaire renseigne sur l'angle θ entre les vecteurs (supposés non nuls) :
  - $\vec{A} \cdot \vec{B} > 0 \implies \cos\theta > 0 \implies -90° < \theta < 90°$ (angle aigu)
  - $\vec{A} \cdot \vec{B} = 0 \implies \cos\theta = 0 \implies \theta = \pm 90°$ (vecteurs orthogonaux)
  - $\vec{A} \cdot \vec{B} < 0 \implies \cos\theta < 0 \implies 90° < \theta < 180°$ ou $-180° < \theta < -90°$ (angle obtus)

  On peut calculer l'angle entre deux vecteurs via : $\cos\theta = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}||||\vec{B}||}$
- **Applications :**
  - Projection d'un vecteur sur un autre.
  - Calcul d'angle entre deux vecteurs.
  - Élimination des faces cachées (back-face culling) : si la normale à une face pointe à l'opposé de la direction de vue (produit scalaire négatif), la face est cachée.
  - Calcul de la quantité de lumière perçue (loi de Lambert : intensité proportionnelle au cosinus de l'angle entre la normale et la direction de la lumière, i.e., au produit scalaire de leurs vecteurs unitaires).
  - Ombres simples.

1.7.8 Produit vectoriel (Cross Product)

Definition 1.7.16. Le **produit vectoriel** de deux vecteurs $\vec{A}$ et $\vec{B}$ dans $\mathbb{R}^3$, noté $\vec{A} \times \vec{B}$ ou $\vec{A} \wedge \vec{B}$, est un vecteur.

Remark 1.7.17.

- **Direction :** Le vecteur résultant $\vec{A} \times \vec{B}$ est perpendiculaire au plan formé par $\vec{A}$ et $\vec{B}$. Son sens est donné par la règle de la main droite (si les doigts de la main droite se courbent de $\vec{A}$ vers $\vec{B}$, le pouce indique la direction de $\vec{A} \times \vec{B}$).
- **Module :** Le module du produit vectoriel est égal au produit des modules des deux vecteurs par le sinus de l'angle θ entre eux :

$$||\vec{A} \times \vec{B}|| = ||\vec{A}||||\vec{B}|| \sin\theta|$$

Ceci correspond à l'aire du parallélogramme formé par $\vec{A}$ et $\vec{B}$. Le module est nul si les vecteurs sont parallèles ($\theta = 0$ ou $\theta = 180°$) et maximal s'ils sont perpendiculaires ($\theta = 90°$).

- **Calcul par composantes :**

$$\begin{aligned}\vec{A} \times \vec{B} &= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix} \\ &= (A_y B_z - A_z B_y)\vec{i} - (A_x B_z - A_z B_x)\vec{j} + (A_x B_y - A_y B_x)\vec{k} \\ &= (A_y B_z - A_z B_y, A_z B_x - A_x B_z, A_x B_y - A_y B_x)\end{aligned}$$

- **Propriétés :**
  - Anticommutativité : $\vec{u} \times \vec{v} = -(\vec{v} \times \vec{u})$
  - Distributivité sur l'addition : $(\vec{u} + \vec{v}) \times \vec{w} = \vec{u} \times \vec{w} + \vec{v} \times \vec{w}$
  - Distributivité par un scalaire k : $(k\vec{u}) \times \vec{v} = \vec{u} \times (k\vec{v}) = k(\vec{u} \times \vec{v})$
  - Non-associativité : $(\vec{u} \times \vec{v}) \times \vec{w} \neq \vec{u} \times (\vec{v} \times \vec{w})$ en général.
- **Application principale :** Calcul de la normale à un plan ou à une surface (par exemple, la normale à un triangle défini par les sommets P1, P2, P3 peut être calculée par $(\vec{P2} - \vec{P1}) \times (\vec{P3} - \vec{P1})$).

Example 1.7.18 (Calcul de produit vectoriel). Soit $\vec{A} = (1, 2, -4)$ et $\vec{B} = (3, -1, 5)$.

$$\begin{aligned}\vec{A} \times \vec{B} &= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & 2 & -4 \\ 3 & -1 & 5 \end{vmatrix} \\ &= \vec{i}(2 \times 5 - (-4) \times (-1)) - \vec{j}(1 \times 5 - (-4) \times 3) + \vec{k}(1 \times (-1) - 2 \times 3) \\ &= \vec{i}(10 - 4) - \vec{j}(5 + 12) + \vec{k}(-1 - 6) \\ &= 6\vec{i} - 17\vec{j} - 7\vec{k} \\ &= (6, -17, -7)\end{aligned}$$

1.7.9 Matrice

Definition 1.7.19. On appelle **matrice** M un tableau à deux indices de $n \times m$ valeurs numériques extraites du même ensemble E. n est le nombre de lignes et m est le nombre de colonnes. On note $M = (m_{ij})$ où $1 \leq i \leq n$ et $1 \leq j \leq m$.

Remark 1.7.20.

- **Exemple :** Une matrice 4×5 :

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} & m_{15} \\ m_{21} & m_{22} & m_{23} & m_{24} & m_{25} \\ m_{31} & m_{32} & m_{33} & m_{34} & m_{35} \\ m_{41} & m_{42} & m_{43} & m_{44} & m_{45} \end{pmatrix}$$

- Si $n = m$, la matrice est dite **carrée**.

Opérations sur les matrices

Remark 1.7.21.

- **Addition (Matrice-matrice, Scalaire-matrice) :** Se fait terme à terme pour des matrices de même dimension.
- **Multiplication :**
  - *Scalaire-matrice :* On multiplie chaque élément de la matrice par le scalaire.
  - *Matrice-vecteur :* Le produit d'une matrice M $(m \times n)$ par un vecteur colonne V $(n \times 1)$ est un vecteur colonne W $(m \times 1)$. La i-ème composante de W est le produit scalaire de la i-ème ligne de M par le vecteur V.

$$w_i = \sum_{k=1}^{n} m_{ik} v_k \quad (1 \leq i \leq m)$$

Example 1.7.22 (Produit Matrice-Vecteur).

$$\begin{pmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{pmatrix} \times \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 2 \times 1 + 4 \times 3 \\ 6 \times 1 + 8 \times 3 \\ 10 \times 1 + 12 \times 3 \end{pmatrix} = \begin{pmatrix} 2 + 12 \\ 6 + 24 \\ 10 + 36 \end{pmatrix} = \begin{pmatrix} 14 \\ 30 \\ 46 \end{pmatrix}$$

  - *Matrice-matrice :* Le produit d'une matrice M_1 $(n \times m)$ par une matrice M_2 $(m \times p)$ est une matrice M $(n \times p)$. L'élément m_{ij} de M est le produit scalaire de la i-ème ligne de M_1 par la j-ème colonne de M_2.

$$m_{ij} = \sum_{k=1}^{m} (m_1)_{ik} (m_2)_{kj} \quad (1 \leq i \leq n, 1 \leq j \leq p)$$

Example 1.7.23 (Produit Matrice-Matrice).

$$\begin{pmatrix} 1 & 3 \\ 7 & 9 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{pmatrix} \quad \text{(impossible: dimensions incompatibles)}$$

$$\begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{pmatrix} = \begin{pmatrix} 1(2) + 3(6) + 5(10) & 1(4) + 3(8) + 5(12) \\ 7(2) + 9(6) + 11(10) & 7(4) + 9(8) + 11(12) \end{pmatrix}$$

$$= \begin{pmatrix} 2 + 18 + 50 & 4 + 24 + 60 \\ 14 + 54 + 110 & 28 + 72 + 132 \end{pmatrix} = \begin{pmatrix} 70 & 88 \\ 178 & 232 \end{pmatrix}$$

- **Inversion :** Pour certaines matrices carrées, il existe une matrice inverse M^{-1} telle que $MM^{-1} = M^{-1}M = I$ (matrice identité).
- **Transposition :** La transposée M^T d'une matrice M est obtenue en échangeant les lignes et les colonnes ($(M^T)_{ij} = M_{ji}$).

Les matrices sont fondamentales en infographie pour représenter les transformations géométriques (rotation, translation, mise à l'échelle, projection).

1.8 Espace Vectoriel et Espace Affine

Remark 1.8.1.

- **Espace vectoriel :** C'est l'espace où vivent les **vecteurs** (représentant des déplacements ou des directions). Ses propriétés principales sont l'existence d'un vecteur nul (origine) et la stabilité par combinaison linéaire (toute combinaison linéaire de vecteurs de l'espace appartient à l'espace).
- **Espace affine :** C'est l'espace où vivent les **points**. Il est construit à partir d'un point de référence (l'origine) et d'un espace vectoriel associé (qui définit les déplacements autorisés à partir de ce point). On peut y définir des objets géométriques comme les droites, les plans, etc. La différence entre deux points d'un espace affine est un vecteur de l'espace vectoriel associé. L'addition d'un point et d'un vecteur donne un point.

Remark 1.8.2. En infographie, on travaille constamment avec ces deux types d'espaces. Les sommets des objets sont des points (espace affine), tandis que les normales, les directions de lumière ou les déplacements sont des vecteurs (espace vectoriel). Les transformations sont souvent représentées par des matrices qui agissent différemment sur les points et les vecteurs (notamment via l'utilisation de coordonnées homogènes).

Chapter 2

Part2

2.1 Introduction à la Modélisation 3D

La modélisation 3D est un domaine fondamental de l'informatique graphique qui consiste à créer des représentations mathématiques d'objets ou de scènes tridimensionnelles. Elle trouve ses racines dans la géométrie.

Definition 2.1.1 (Géométrie)**.** Le terme **géométrie** dérive du grec ancien (geômétrês), composé de "gê" (Terre) et "metron" (mesure). La géométrie peut être définie comme :

1. L'étude des formes, des tailles, des motifs et des positions des objets.
2. L'étude des espaces où l'on peut mesurer certaines quantités, telles que les longueurs, les angles, etc.

En modélisation 3D, nous cherchons à décrire numériquement la géométrie d'objets virtuels. Ces objets peuvent être simples (formes géométriques de base) ou complexes (personnages, terrains, bâtiments, scènes naturelles).

2.1.1 Comment décrire la géométrie ?

Il existe plusieurs manières de décrire une forme géométrique :

- **Implicite :** Par une équation ou une condition que les points de la forme doivent satisfaire. Par exemple, un cercle unité dans le plan peut être décrit par l'équation $x^2 + y^2 = 1$.
- **Explicite :** Par une paramétrisation, c'est-à-dire une fonction qui génère les points de la forme. Pour le cercle unité, une paramétrisation est $(\cos\theta, \sin\theta)$ pour θ variant de 0 à 2π.
- **Linguistique :** Par une description textuelle, comme "cercle unitaire".
- **Discrète :** En approximant la forme par un ensemble fini de points ou de segments.
- **Dynamique :** En décrivant le mouvement qui génère la forme, par exemple, un point tournant autour d'un centre $\frac{d^2\mathbf{x}}{dt^2} = -\mathbf{x}$.
- **Par Symétrie :** En décrivant les opérations de symétrie qui laissent la forme inchangée.

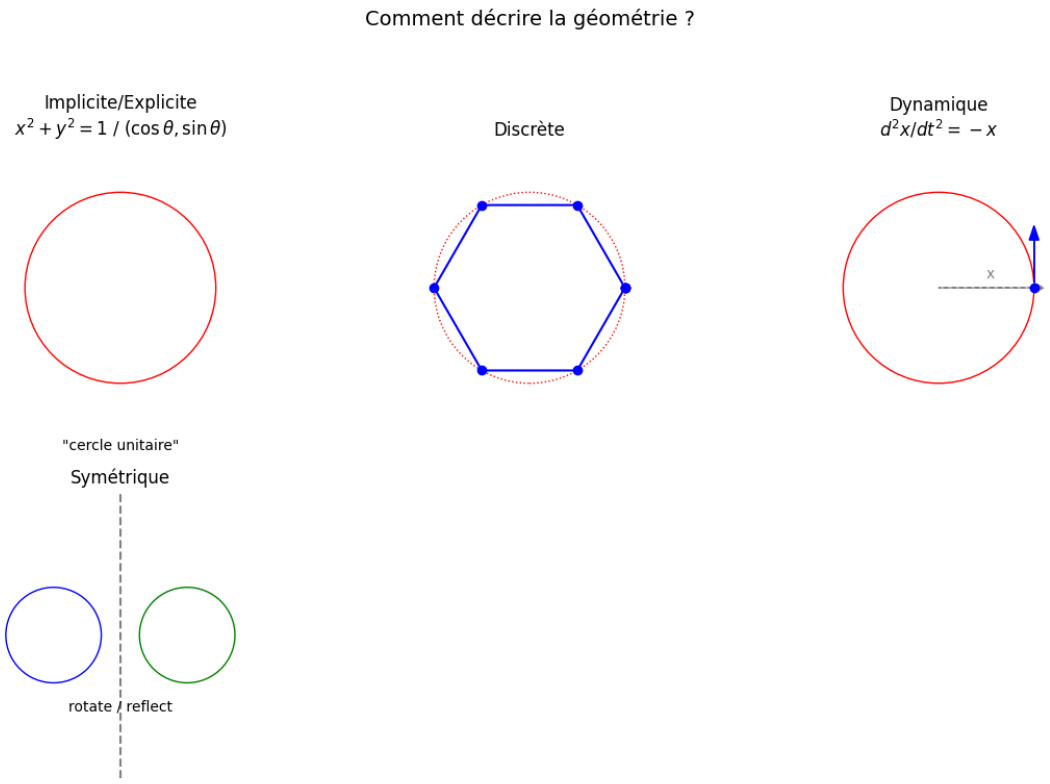

Figure 2.1: Différentes manières de décrire la géométrie.

2.2 Représentations de la Géométrie

En informatique graphique, on distingue principalement deux types de représentations pour la géométrie 3D.

2.2.1 Représentations Implicites

Definition 2.2.1 (Représentation Implicite)**.** Une surface implicite est définie comme l'ensemble des points (x, y, z) qui satisfont une équation de la forme $f(x, y, z) = 0$. Les points ne sont pas donnés directement, mais ils doivent vérifier une certaine relation.

Example 2.2.2 (Sphère Unitaire)**.** La sphère unité est l'ensemble des points (x, y, z) tels que $x^2+y^2+z^2 = 1$. Ici, $f(x, y, z) = x^2+y^2+z^2-1$. Pour un point donné, il est facile de vérifier s'il appartient à la surface (si $f(x, y, z) = 0$), s'il est à l'intérieur ($f(x, y, z) < 0$) ou à l'extérieur ($f(x, y, z) > 0$). Cependant, il est plus difficile de générer directement des points se trouvant sur la surface.

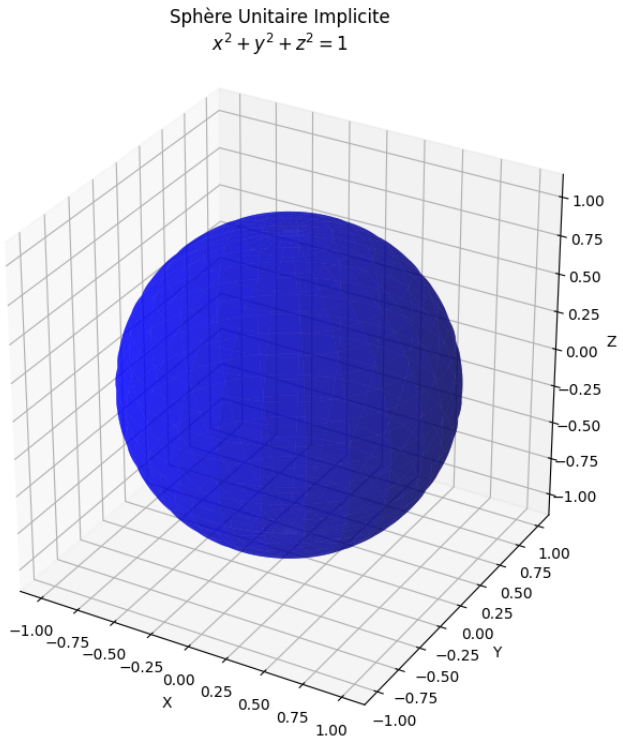

Figure 2.2: Représentation implicite d'une sphère unité.

2.2.2 Représentations Explicites

Definition 2.2.3 (Représentation Explicite)**.** Une surface explicite (ou paramétrique) est définie par une fonction f qui mappe un domaine de paramètres (souvent un sous-ensemble de $\mathbb{R}^2$) vers l'espace 3D ($\mathbb{R}^3$). Tous les points de la surface sont donnés directement par l'évaluation de la fonction $f(u, v) = (x(u, v), y(u, v), z(u, v))$ pour des valeurs de paramètres (u, v) dans le domaine.

Example 2.2.4 (Sphère Unitaire)**.** Les points sur la sphère unité peuvent être générés explicitement par la fonction $f(u, v) = (\cos(u)\sin(v), \sin(u)\sin(v), \cos(v))$ pour $0 \leq u \leq 2\pi$ et $0 \leq v \leq \pi$. Il est facile de générer des points sur la surface, mais il peut être plus complexe de vérifier si un point donné appartient à la surface ou de déterminer les relations spatiales (intérieur/extérieur).

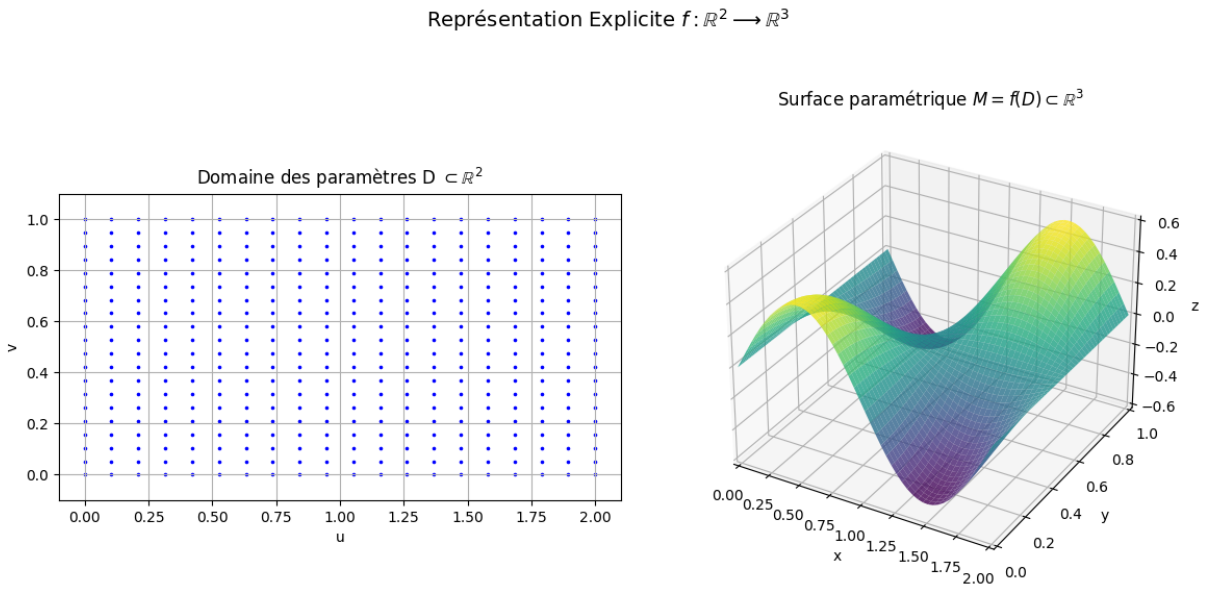

Figure 2.3: Représentation explicite : mapping d'un domaine 2D vers une surface 3D.

2.3 Encoder Numériquement la Géométrie

Pour utiliser la géométrie dans un ordinateur, nous devons l'encoder numériquement. Le choix de l'encodage dépend de l'application et du type de géométrie.

Principales méthodes d'encodage :

- **Explicite :**
  - **Nuage de points :** Une collection de points (x, y, z) dans l'espace. Simple mais sans information de connectivité ou de surface.
  - **Maillage polygonal :** Représentation par des sommets connectés pour former des polygones (souvent des triangles ou des quadrilatères) qui approximent la surface. Très courant en infographie.
  - **Subdivision :** Surfaces lisses générées en subdivisant récursivement un maillage de contrôle grossier.
  - **NURBS (Non-Uniform Rational B-Splines) :** Surfaces lisses définies mathématiquement, utilisées en CAO (Conception Assistée par Ordinateur).
- **Implicite :**
  - **Ensemble de niveaux (Level Set) :** La surface est définie comme le niveau zéro d'une fonction de distance signée définie sur une grille. Utile pour les formes complexes et les changements de topologie.
  - **Surface algébrique :** Définie par une équation polynomiale $P(x, y, z) = 0$.
  - **L-systems (Lindenmayer Systems) :** Grammaires formelles utilisées pour générer des structures fractales, souvent employées pour modéliser des plantes.

Chaque choix est mieux adapté à une tâche ou à un type de géométrie différent. Par exemple, les maillages sont polyvalents pour le rendu, tandis que les NURBS sont précis pour la conception industrielle.

2.4 Techniques de Modélisation 3D

La création de modèles 3D peut se faire selon différentes approches :

- **Reconstruction :** Création d'un modèle 3D à partir d'un objet réel, souvent par scan 3D ou photogrammétrie (utilisation de multiples photos).
- **Modélisation automatique :** Génération algorithmique de modèles, souvent pour des objets naturels complexes comme les arbres, ou des scènes étendues comme les terrains.
- **Modélisation interactive :** Utilisation d'outils logiciels spécialisés (CAO, modeleurs 3D comme Blender, 3ds Max, Maya) où un utilisateur sculpte, assemble ou modifie la géométrie directement.

La plupart des objets virtuels (terrains, personnages, objets manufacturés, etc.) sont représentés par leur surface.

2.5 Modélisation Surfacique

La modélisation surfacique se concentre sur la représentation de la "peau" extérieure d'un objet.

2.5.1 Caractéristiques

- Représente l'extérieur visuel d'un objet et ses contours.
- Ne définit intrinsèquement aucune propriété de masse ou d'épaisseur (les objets sont considérés comme creux).
- Les modèles surfaciques ne peuvent pas être directement "découpés" comme des objets solides, car ils n'ont pas d'intérieur défini.

2.5.2 Approches

Plusieurs techniques sont utilisées en modélisation surfacique :

- **Primitives solides :** Utilisation de formes géométriques de base (sphères, cubes, cylindres...) qui sont ensuite combinées (voir CSG plus loin, bien que CSG soit souvent considérée volumique).
- **Maillage (Mesh) :** L'approche la plus courante, décrite ci-dessous.
- **Surfaces paramétrées :** Utilisation de fonctions mathématiques (comme les NURBS) pour définir des surfaces lisses.
- **Balayage de surface (Sweep) :** Création d'une surface en déplaçant une courbe de profil le long d'une trajectoire.

2.6 Maillage (Mesh)

Definition 2.6.1 (Maillage)**.** Un maillage (ou "mesh" en anglais) représente une forme 3D comme un ensemble de **sommets** (vertices), d'**arêtes** (edges) connectant ces sommets, et de **facettes** (faces) délimitées par les arêtes. Les facettes sont généralement des triangles ou des quadrilatères.

Plus le nombre de polygones (souvent des triangles) est élevé, plus le réalisme de la surface peut être grand, mais plus le coût en calcul et en mémoire est important.

2.6.1 Avantages et Inconvénients

- **Avantages :**
  - Permet de représenter des surfaces et des courbes complexes.
  - Très flexible et largement supporté par les logiciels et le matériel graphique.
- **Désavantages :**
  - Peut avoir un aspect angulaire ("facettisé") si la résolution est faible.
  - Une haute résolution (beaucoup de polygones) nécessite des performances de calcul élevées et une grande capacité mémoire.

2.6.2 Représentation des Données : Énumération de Facettes

Une manière simple de stocker un maillage est d'énumérer toutes les facettes, en listant les coordonnées des sommets pour chaque facette.

Example 2.6.2 (Énumération simple)**.** Considérons une pyramide simple à base triangulaire (tétraèdre) avec 4 sommets P0, P1, P2, P3. Les 4 facettes triangulaires seraient : F1 = (P0, P1, P2) F2 = (P0, P2, P3) F3 = (P3, P1, P0) F4 = (P3, P2, P1)
Objet = (F1, F2, F3, F4)

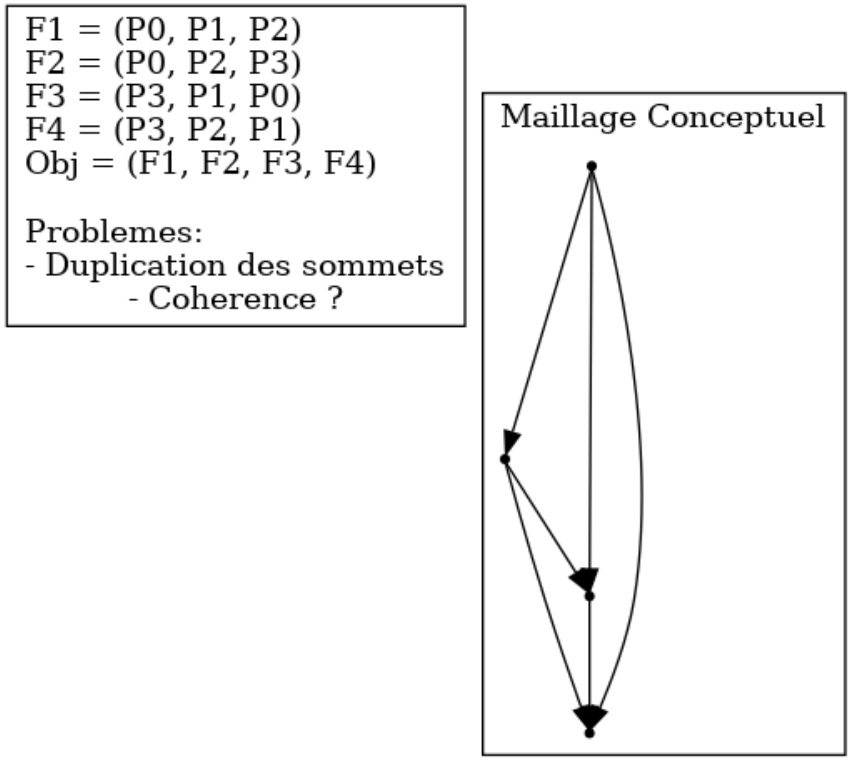

Figure 2.4: Énumération simple des facettes d'un tétraèdre.

Cette approche simple a des inconvénients majeurs :

- **Duplication de données :** Les coordonnées de chaque sommet sont stockées plusieurs fois (une fois pour chaque facette à laquelle il appartient).

- **Problèmes de cohérence :** Il est difficile de garantir que les différentes copies d'un même sommet ont exactement les mêmes coordonnées, ou que les arêtes partagées sont définies de manière cohérente.

2.6.3 Représentation avec Partage de Sommets

Une meilleure approche consiste à stocker une seule liste de sommets (avec leurs coordonnées) et à définir ensuite les facettes en utilisant les indices des sommets dans cette liste.

Example 2.6.3 (Partage de Sommets). Liste des Sommets (LS) : LS = {P0, P1, P2, P3} (chaque Pi a des coordonnées x,y,z)
Facettes (indices dans LS, base 0) : F1 = (0, 1, 2) F2 = (0, 2, 3) F3 = (3, 1, 0) F4 = (3, 2, 1)
Objet = (LS, F1, F2, F3, F4)

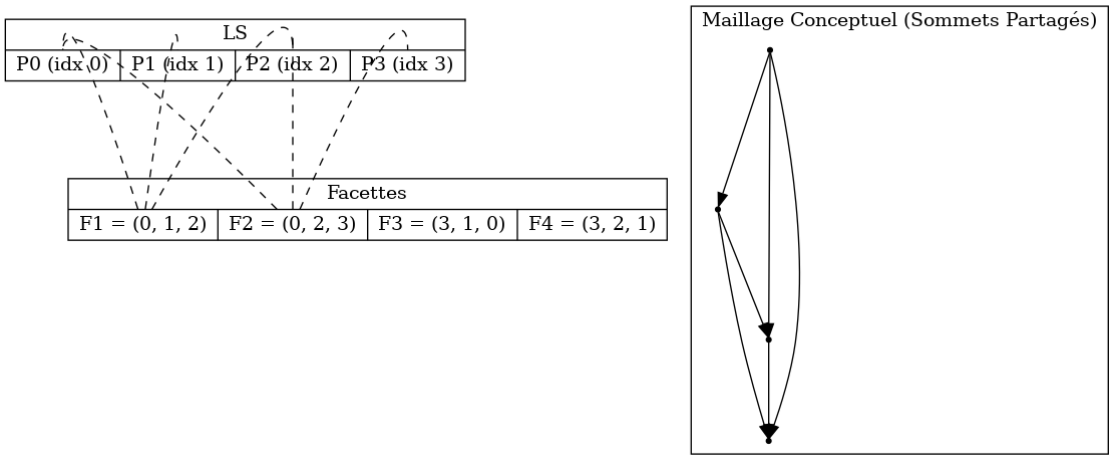

Figure 2.5: Énumération des facettes avec partage de sommets.

Cette méthode évite la duplication des données de sommets et facilite la manipulation de la topologie du maillage.

2.6.4 Orientation des Surfaces

La **normale** à une face est un vecteur perpendiculaire à cette face, indiquant son orientation (par exemple, vers l'"extérieur" de l'objet). L'ordre dans lequel les sommets d'une face sont listés détermine la direction de la normale, typiquement via la règle de la main droite.

- **Convention Sens trigonométrique (counter-clockwise) :** Les sommets sont listés dans le sens inverse des aiguilles d'une montre lorsqu'on regarde la face de l'extérieur. La normale est alors "sortante".
- **Convention Sens anti-trigonométrique (clockwise) :** Les sommets sont listés dans le sens des aiguilles d'une montre. La normale est alors "rentrante".

Il est crucial de maintenir une orientation cohérente sur tout le maillage pour des opérations comme l'éclairage et le rendu corrects.

2.7 Surfaces de Subdivision

Les surfaces de subdivision sont une technique pour créer des surfaces lisses à partir d'un maillage polygonal grossier, souvent appelé "maillage cage" ou "maillage de contrôle". Le principe est de subdiviser

récursivement chaque polygone du maillage cage en polygones plus petits, et d'ajuster la position des nouveaux sommets (et éventuellement des anciens) selon des règles précises. Cela produit une surface qui converge vers une limite lisse. La forme finale de la surface est directement contrôlée par la géométrie du maillage cage initial (positions relatives des sommets et des segments). C'est une technique puissante pour combiner le contrôle polygonal avec la création de formes organiques lisses.

2.8 Surfaces Paramétrées

Comme mentionné précédemment (Représentations Explicites), une surface paramétrée est définie par une fonction $f : D \subset \mathbb{R}^2 \to \mathbb{R}^3$. La surface est calculée directement par l'évaluation de cette fonction $f(u, v) = (x(u, v), y(u, v), z(u, v))$.

- **Avantages :** Permet de définir mathématiquement des surfaces lisses et précises (ex: NURBS).
- **Inconvénients :** Peut être limité dans les types d'objets pouvant être générés facilement. La conversion vers un maillage polygonal est souvent nécessaire pour le rendu temps réel.

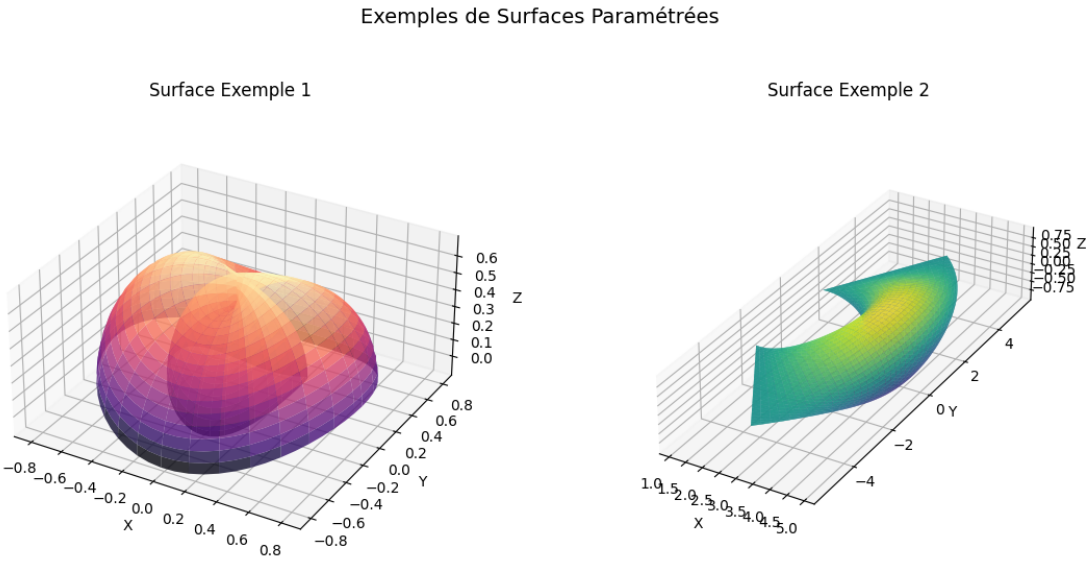

Figure 2.6: Exemples de surfaces générées par des équations paramétriques.

2.9 Balayage de Surface (Sweep)

Le balayage (ou extrusion) est une technique de modélisation où une forme 2D (le profil) est déplacée le long d'une trajectoire (le chemin). La surface balayée est l'ensemble des points parcourus par le profil.

- Si le profil est une ligne et le chemin est une ligne, on obtient un quadrilatère.
- Si le profil est un cercle et le chemin est une ligne droite perpendiculaire au plan du cercle, on obtient un cylindre.
- Si le profil est un cercle et le chemin est un cercle, on obtient un tore (donut).
- Si le profil est une courbe et le chemin est une autre courbe, on obtient une surface plus complexe.

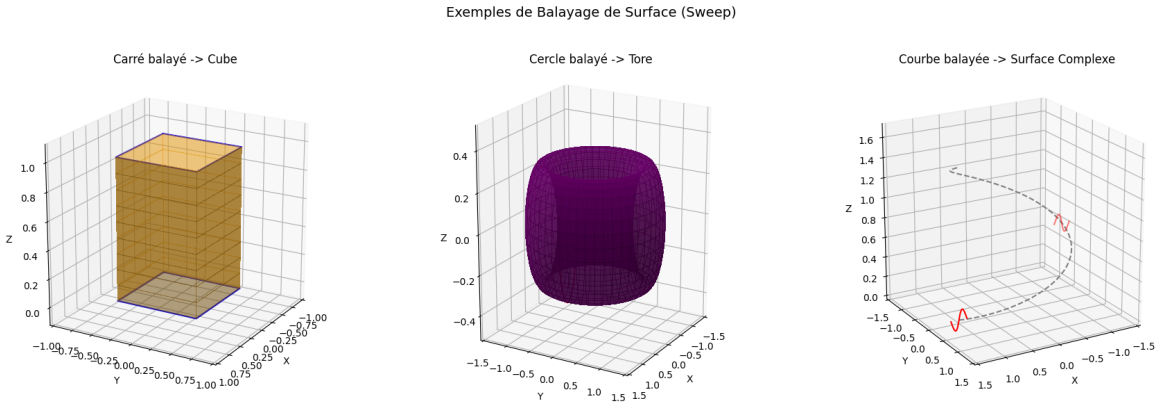

Figure 2.7: Exemples de surfaces générées par balayage.

2.10 Modélisation Volumique

Contrairement à la modélisation surfacique qui ne représente que l'extérieur, la modélisation volumique représente l'objet entier, y compris son intérieur.

2.10.1 Caractéristiques

- Le modèle contient des informations sur chaque point dans l'espace (ou du moins, permet de déterminer si un point est à l'intérieur, à l'extérieur ou sur la frontière de l'objet).
- L'espace (ou l'objet) est souvent décrit en termes mathématiques purs, ou via des opérations booléennes (union, intersection, soustraction) et des mélanges ("blending") entre des formes de base.
- Il n'y a pas nécessairement de géométrie explicite (sommets, faces) tant que le modèle n'a pas besoin d'être visualisé (rendu) ou exporté.

2.10.2 Approches

Voxélisation

Definition 2.10.1 (Voxélisation)**.** Représentation d'un objet 3D par une grille régulière de **voxels** (pixels volumiques). Chaque voxel contient une information indiquant s'il fait partie de l'objet (et potentiellement d'autres propriétés comme la couleur, la densité, etc.). C'est un modèle discret.

- **Applications :** Imagerie médicale (IRM, CT scans), simulation de fluides (CFD), sculpture virtuelle.
- **Paramètres :** Intérieur/extérieur, couleur, réfraction/absorption.
- **Avantages :** Topologie abstraite facile à gérer, bon pour les données naturelles (scans), permet le rendu volumétrique direct (visualisation de l'intérieur).
- **Désavantages :** Nécessite de grands ensembles de données (mémoire), peut être difficile de générer des données complexes non scannées, peut souffrir d'anisotropie (artefacts liés à l'orientation de la grille).

Géométrie Solide Constructive (Constructive Solid Geometry – CSG)

Definition 2.10.2 (CSG)**.** Méthode de modélisation où des formes géométriques simples (**primitives** : sphères, cubes, cylindres, cônes, etc.) sont combinées à l'aide d'opérations booléennes ensemblistes

(**union** $\cup$, **intersection** $\cap$, **différence** $-$).

La structure est souvent représentée par un arbre binaire (CSG tree) où les feuilles sont les primitives et les nœuds internes sont les opérations booléennes.

- **Applications :** CAO (SolidWorks, etc.), modélisation pour le rendu (Pov-Ray).

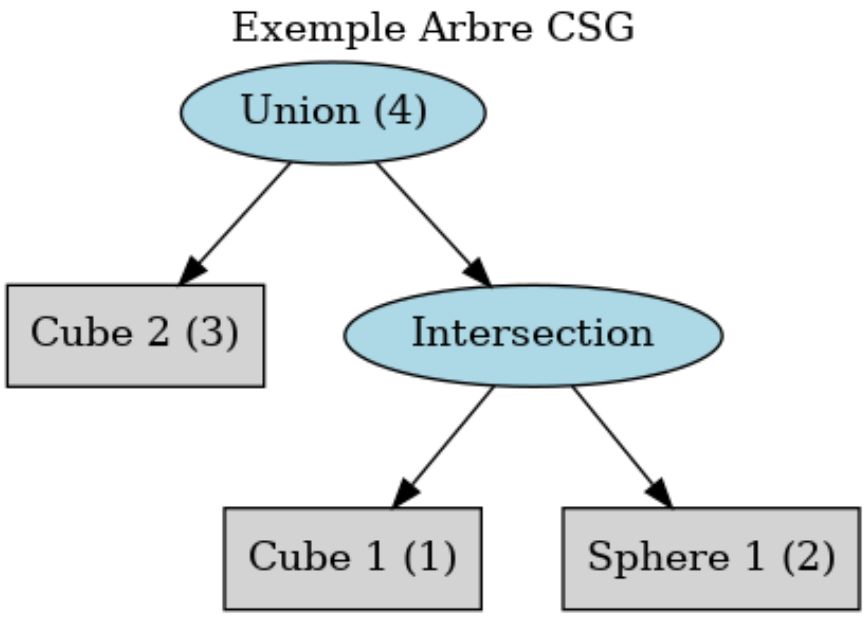

Figure 2.8: Exemple d'arbre CSG représentant une combinaison d'opérations booléennes sur des primitives.

Example 2.10.3 (Exercice)**.** Identifier les opérations booléennes correspondant aux numéros 1 à 4 dans la figure CSG pour obtenir la forme finale. *(Solution basée sur l'interprétation visuelle de la diapositive 8)* 1: Cube A 2: Sphère B 3: Cube C (plus petit) Intersection(A, B) -¿ Forme intermédiaire D Union(C, D) -¿ Forme finale (4)

Blobtree

Definition 2.10.4 (Blobtree)**.** Structure arborescente similaire à l'arbre CSG, mais qui combine des primitives (qui peuvent être implicites ou paramétriques) en utilisant non seulement des opérations booléennes, mais aussi des opérations de **mélange** (blending, smooth union) et des **déformations** globales (torsion, étirement, etc.).

Permet de créer des formes organiques complexes tout en gardant une structure hiérarchique.

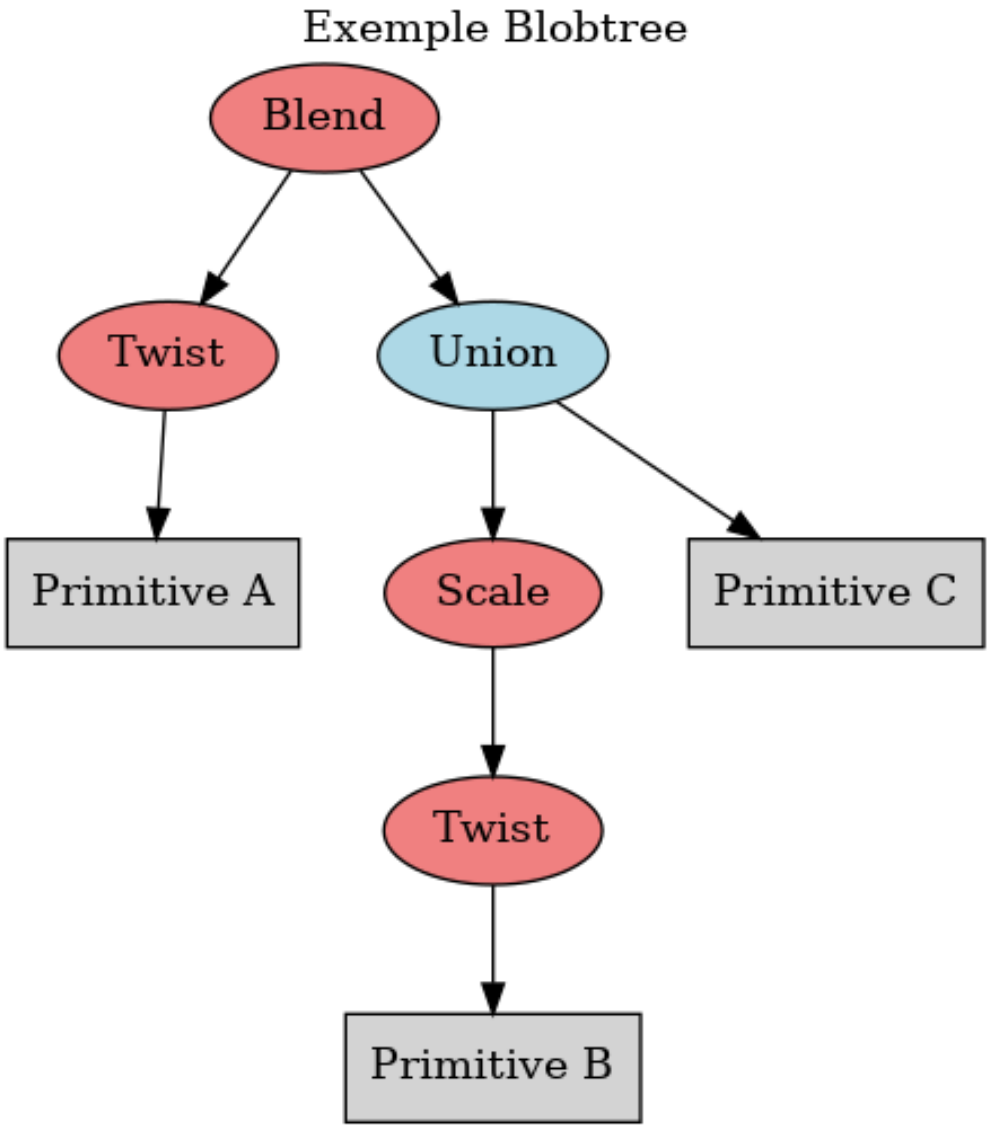

Figure 2.9: Exemple de structure Blobtree combinant primitives, opérations booléennes et déformations.

2.11 Nuage de Points 3D

Definition 2.11.1 (Nuage de Points)**.** Un nuage de points est une collection de points (x, y, z) représentant des positions sur la surface d'objets dans l'espace 3D. Il ne contient généralement pas d'information explicite sur la connectivité entre les points ou sur les surfaces sous-jacentes.

2.11.1 Sources de Données

Les nuages de points sont souvent générés par :

- **Scanner 3D :** Dispositifs utilisant la lumière structurée ou le laser (LiDAR) pour mesurer la distance à de nombreux points sur la surface d'un objet réel.
- **Reconstruction à partir d'images/vidéos 2D :** Techniques comme la Stéréo Vision ou la Structure from Motion (SfM) qui estiment la géométrie 3D à partir de multiples vues 2D. (Exemple : VisualSFM).

2.11.2 Traitements Courants sur les Nuages de Points

Une fois acquis, les nuages de points bruts nécessitent souvent des traitements pour être utilisables :

- **Tâche 1 : Simplification :** Réduire le nombre de points tout en préservant la forme globale. Des approches locales (fusion de points proches) ou globales (ajustement de surface) existent.
- **Tâche 2 : Classification / Reconnaissance d'objets :** Assigner une étiquette sémantique (ex: "chaise", "table", "mur") à l'ensemble du nuage ou à des sous-ensembles.
- **Tâche 3 : Segmentation :** Diviser le nuage de points en différentes parties significatives (ex: ailes, fuselage, queue d'un avion).
- **Tâche 4 : Estimation des normales :** Calculer un vecteur normal (perpendiculaire à la surface locale) pour chaque point, information cruciale pour le rendu et l'analyse de forme.

Ces tâches sont essentielles pour passer d'un nuage de points brut à un modèle 3D structuré et interprétable.

2.12 Transformations Géométriques

Les transformations géométriques sont des fonctions qui modifient la position, l'orientation ou la forme des objets dans l'espace.

Definition 2.12.1 (Transformation Spatiale)**.** Une transformation spatiale $f : \mathbb{R}^n \to \mathbb{R}^n$ est une fonction qui assigne à chaque point de l'espace un nouvel emplacement.

Nous nous concentrerons sur les transformations de l'espace $\mathbb{R}^3$. Celles-ci peuvent être décomposées en transformations **linéaires** (rotation, mise à l'échelle, cisaillement) et **non-linéaires** (la plus courante étant la translation).

2.12.1 Pipeline Graphique

Les transformations sont fondamentales dans le pipeline graphique standard, qui décrit les étapes pour afficher une scène 3D à l'écran : Transformations de modélisation $\longrightarrow$ Illumination (Shading) $\longrightarrow$ Transformation d'affichage $\longrightarrow$ Clipping $\longrightarrow$ Transformation écran (Projection) $\longrightarrow$ Pixelisation (Rasterization) $\longrightarrow$ Visibilité / Rendu. Les **transformations de modélisation** permettent de positionner et d'orienter chaque objet dans la scène globale (passage de l'"object space" au "world space").

2.12.2 Rappel : Transformation Linéaire

Une transformation $f : \mathbb{R}^n \to \mathbb{R}^n$ est dite **linéaire** si elle satisfait deux conditions :

1. Additivité : $f(\mathbf{u} + \mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$ pour tous vecteurs $\mathbf{u}, \mathbf{v}$.
2. Homogénéité : $f(c\mathbf{u}) = cf(\mathbf{u})$ pour tout scalaire c et tout vecteur $\mathbf{u}$.

Géométriquement, une transformation linéaire :

- Fait correspondre des lignes droites à des lignes droites.
- Préserve l'origine ($f(\mathbf{0}) = \mathbf{0}$).

Algébriquement, elle préserve les opérations de l'espace vectoriel (addition de vecteurs et multiplication par un scalaire).

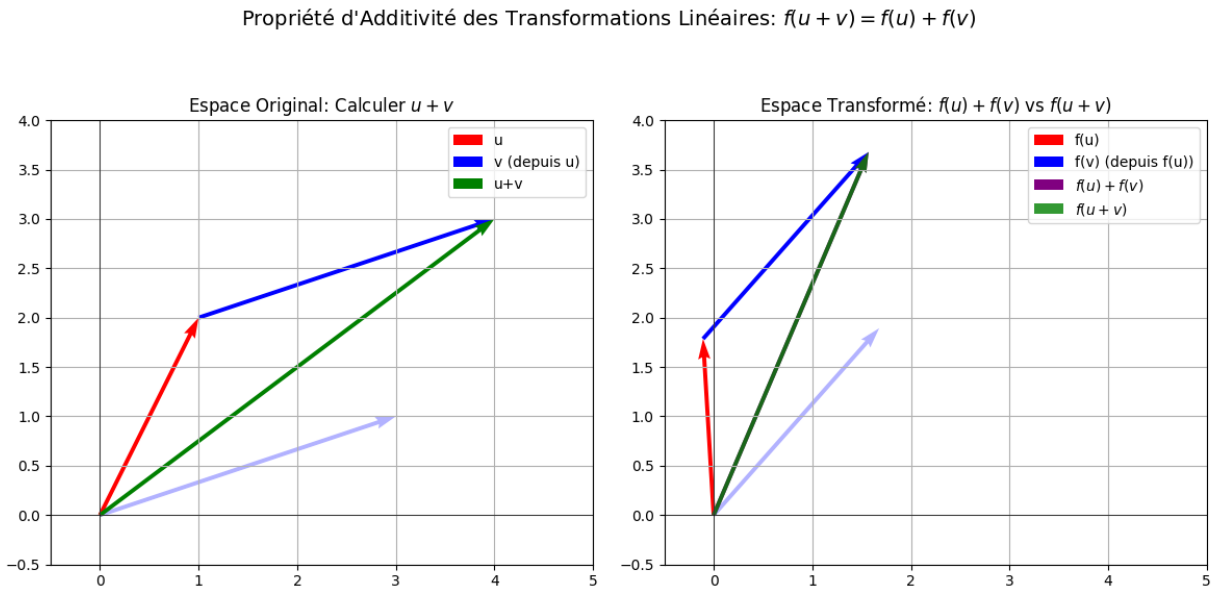

Figure 2.10: Illustration de la propriété d'additivité $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u})+f(\mathbf{v})$ pour une transformation linéaire.

2.12.3 Utilisations des Transformations

Les transformations géométriques sont utilisées pour de nombreuses opérations en modélisation 3D :

- **Déplacement d'un objet** dans une scène (translation, rotation).
- **Déplacement de l'observateur** (caméra) par rapport à une scène (équivalent à transformer la scène inversement).
- **Réplication** d'un motif ou d'un objet (combinaison de translation, rotation, échelle).
- **Déformation** d'un objet (mise à l'échelle non uniforme, cisaillement, ou transformations plus complexes).
- **Projection :** Transformer la scène 3D en une image 2D vue depuis la caméra.

2.12.4 Représentation des Objets pour la Transformation

Un objet 3D est souvent décrit par un ensemble de sommets (points). Appliquer une transformation à un objet revient à appliquer cette transformation à chacun de ses sommets. On utilise la notation vectorielle pour représenter les sommets : un point P de coordonnées (x, y, z) est représenté par un vecteur colonne $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

2.13 Types de Transformations Élémentaires

Les transformations les plus courantes sont :

- **Translation :** Déplacement de tous les points d'une distance et d'une direction constantes. Non linéaire car l'origine n'est pas fixe ($f(\mathbf{0}) \neq \mathbf{0}$).
- **Rotation :** Pivotement des points autour d'un axe fixe ou d'un point fixe. Linéaire.

- **Homothétie (Scaling) :** Agrandissement ou réduction des distances par rapport à un point fixe (souvent l'origine). Linéaire. Peut être uniforme (même facteur dans toutes les directions) ou non uniforme.
- **Cisaillement (Shear) :** Déformation qui incline les points, comme si on poussait sur le côté d'un livre. Linéaire.

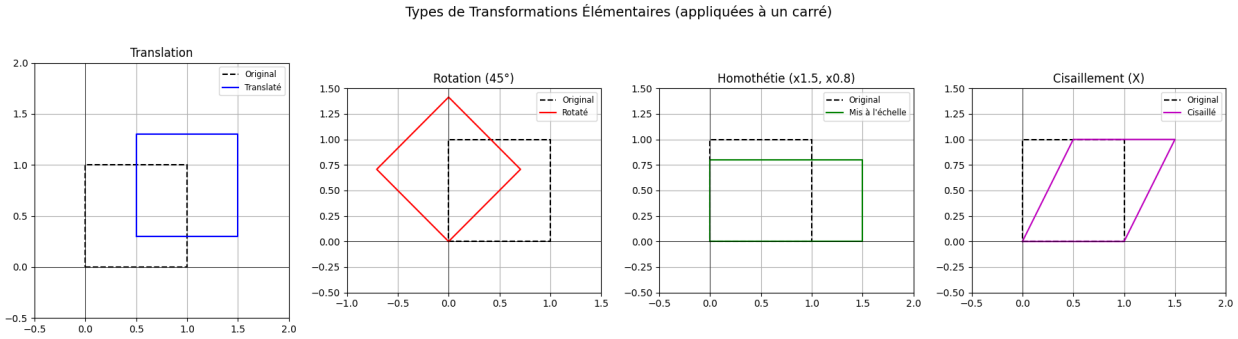

Figure 2.11: Illustration des transformations élémentaires.

2.14 Transformations : Représentation Matricielle

Les transformations linéaires (rotation, échelle, cisaillement) peuvent être représentées par des matrices. Si $P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ est un point et M est la matrice de transformation (3x3 pour les transformations linéaires en 3D), le point transformé P' est donné par $P' = M \cdot P$.

2.14.1 Rotation

Les rotations sont définies par trois propriétés fondamentales :

- L'origine (ou le centre de rotation) reste fixe.
- Les distances entre les points sont conservées.
- L'orientation (ex: règle de la main droite) est conservée.

Les deux premières propriétés impliquent que les rotations sont linéaires.

Rotation autour de l'axe Z

Une rotation d'un angle θ autour de l'axe Z transforme un point (x, y, z) en (x', y', z') selon :

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \\ z' &= z \end{aligned}$$

La matrice de rotation correspondante (en 3D) est :

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

Le point transformé est $P' = R_z(\theta) \cdot P$.

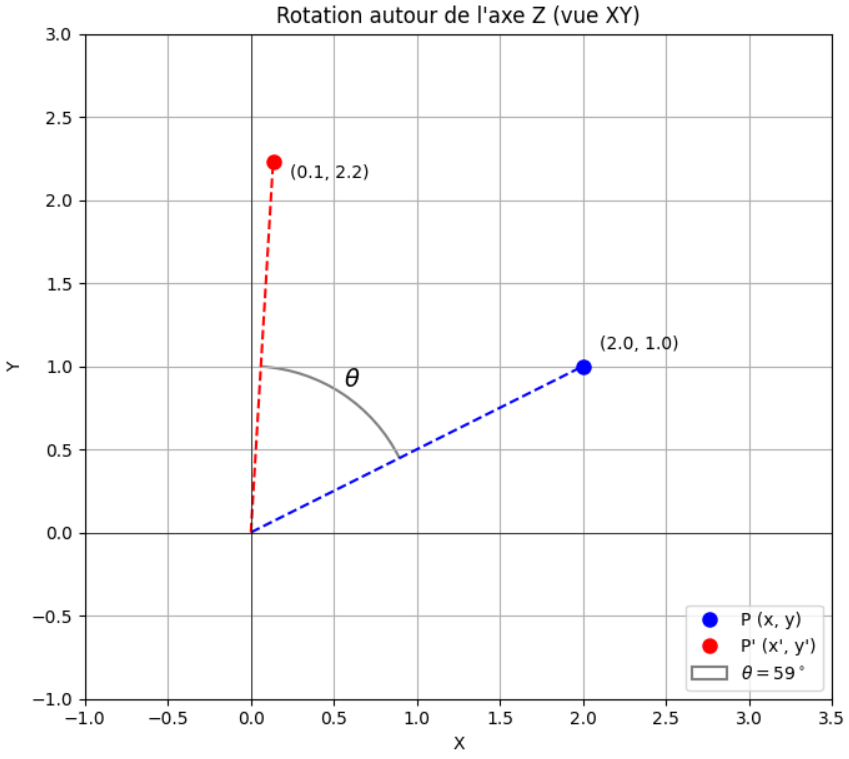

Figure 2.12: Rotation d'un point P autour de l'origine dans le plan XY (équivalent à une rotation autour de l'axe Z).

2.14.2 Homothétie (Scaling)

Une homothétie (mise à l'échelle) par des facteurs S_x, S_y, S_z le long des axes transforme un point (x, y, z) en (x', y', z') :

$$x' = S_x \cdot x$$
$$y' = S_y \cdot y$$
$$z' = S_z \cdot z$$

La matrice d'homothétie est :

$$S = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{pmatrix}$$

Le point transformé est $P' = S \cdot P$. L'homothétie est une transformation linéaire (elle satisfait $f(c\mathbf{u}) = cf(\mathbf{u})$ et $f(\mathbf{u} + \mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$).

2.14.3 Translation

Une translation par un vecteur $T = (T_x, T_y, T_z)$ transforme un point (x, y, z) en (x', y', z') :

$$x' = x + T_x$$
$$y' = y + T_y$$
$$z' = z + T_z$$

En notation vectorielle : $P' = P + T$. La translation n'est **pas** une transformation linéaire car l'origine n'est pas préservée ($f(\mathbf{0}) = T \neq \mathbf{0}$ si $T \neq \mathbf{0}$). Elle ne peut donc pas être représentée par une simple multiplication matricielle 3x3.

> **Remark 2.14.1.** Q: La translation est-elle une transformation linéaire ? R: Non, car elle ne préserve pas l'origine (sauf si le vecteur de translation est nul).

2.14.4 Coordonnées Homogènes

Pour unifier le traitement des transformations linéaires (rotation, échelle) et non-linéaires (translation) sous une forme matricielle unique, on utilise les **coordonnées homogènes**. Un point 3D (x, y, z) est représenté par un vecteur 4D $(x, y, z, 1)$. Plus généralement, le point (X, Y, Z, W) en coordonnées homogènes correspond au point 3D $(X/W, Y/W, Z/W)$ si $W \neq 0$. On utilise généralement $W = 1$.

Avec les coordonnées homogènes, les transformations 3D (y compris la translation) peuvent être représentées par des matrices 4x4. Si $P_h = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ est le point en coordonnées homogènes et M_H est la matrice de transformation 4x4, le point transformé $P'_h = \begin{pmatrix} X' \\ Y' \\ Z' \\ W' \end{pmatrix}$ est donné par $P'_h = M_H \cdot P_h$. Le point 3D correspondant est $(X'/W', Y'/W', Z'/W')$.

Structure de la Matrice 4x4

Une matrice de transformation 4x4 M_H peut être décomposée en sous-matrices :

$$M_H = \begin{pmatrix} M_{3x3} & T_{3x1} \\ P_{1x3} & S_{1x1} \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ P_x & P_y & P_z & S \end{pmatrix}$$

- M_{3x3} (coin supérieur gauche) : Contient les effets linéaires (rotation, échelle, cisaillement).
- T_{3x1} (colonne de droite) : Contient les effets de translation (T_x, T_y, T_z).
- P_{1x3} (ligne du bas) : Utilisée pour les projections perspectives. Souvent $(0, 0, 0)$ pour les transformations affines.
- S_{1x1} (coin inférieur droit) : Facteur d'échelle global homogène. Souvent 1 pour les transformations affines.

2.15 Manipulations Géométriques avec Matrices Homogènes

Soit un point p en 3D, représenté par $P_h = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ en coordonnées homogènes. Soit une transformation géométrique M définie par une matrice 4x4 M_H. Le point transformé p' (en coordonnées homogènes P'_h) est obtenu par multiplication matricielle :

$$P'_h = M_H \cdot P_h$$

Cela permet de combiner plusieurs transformations (ex: translation, puis rotation, puis mise à l'échelle) en multipliant leurs matrices respectives. L'ordre des multiplications est important ! $M_3 \cdot M_2 \cdot M_1 \cdot P_h$ signifie appliquer M_1 d'abord, puis M_2, puis M_3.

2.15.1 Matrice de Translation

La translation par (T_x, T_y, T_z) est représentée par la matrice 4x4 :

$$T(T_x, T_y, T_z) = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En appliquant cette matrice à P_h:

$$T \cdot P_h = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}$$

Ce qui correspond bien au point translaté.

2.15.2 Matrice de Rotation (angles d'Euler)

Une rotation 3D quelconque peut être décomposée en une séquence de trois rotations autour des axes principaux (X, Y, Z). Les matrices de rotation 4x4 autour de chaque axe sont :

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

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

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

Une rotation composite R est obtenue en multipliant ces matrices, par exemple $R = R_z(\gamma)R_y(\beta)R_x(\alpha)$. La matrice résultante R (le bloc 3x3 supérieur gauche de la matrice 4x4 totale) contiendra les cosinus directeurs de la rotation combinée.

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Règles pour Construire les Matrices de Rotation Axiales

Pour construire la matrice 4x4 de rotation d'angle θ autour d'un axe principal (X, Y, ou Z) :

- La ligne et la colonne correspondant à l'axe de rotation contiennent des 0, sauf un 1 sur la diagonale. (Ex: pour R_z, ligne 3 et colonne 3 sont $(0, 0, 1, 0)$ et $(0, 0, 1, 0)^T$).
- Le 1 dans le coin inférieur droit (coordonnée homogène) reste 1. Les autres éléments de la 4ème ligne et 4ème colonne sont 0.
- Dans le bloc 2x2 restant correspondant aux deux autres axes, on place $\cos\theta$ sur la diagonale.
- On place $\sin\theta$ et $-\sin\theta$ sur l'anti-diagonale pour "compléter le carré". Le signe '-' est placé sur le $\sin\theta$ de la ligne correspondant à l'axe *suivant* l'axe de rotation dans l'ordre cyclique (X $\to$ Y $\to$ Z $\to$ X).

- Rotation X : affecte Y, Z. L'axe suivant est Y. $-\sin\theta$ en ligne Y, colonne Z.
- Rotation Y : affecte X, Z. L'axe suivant est Z. $-\sin\theta$ en ligne Z, colonne X.
- Rotation Z : affecte X, Y. L'axe suivant est X. $-\sin\theta$ en ligne X, colonne Y.

2.16 Autres Transformations

2.16.1 Homothétie Isotrope

C'est une mise à l'échelle uniforme avec le même facteur S dans toutes les directions. La matrice 3x3 est $S \cdot I$, où I est la matrice identité. La matrice 4x4 est :

$$S_{iso}(S) = \begin{pmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.16.2 Affinités Orthogonales (Homothétie Non-Uniforme)

Ce sont des mises à l'échelle avec des facteurs potentiellement différents le long des axes principaux S_x, S_y, S_z. La matrice 4x4 générale est :

$$S(S_x, S_y, S_z) = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Des cas particuliers sont les affinités par rapport à un plan (mise à l'échelle le long d'un seul axe) :

- Affinité d'axe X par rapport au plan yOz : $S(S_x, 1, 1)$

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Affinité d'axe Y par rapport au plan xOz : $S(1, S_y, 1)$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Affinité d'axe Z par rapport au plan xOy : $S(1, 1, S_z)$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Chapter 3

Part3

3.1 Transformations de Modélisation

En infographie, les transformations géométriques sont fondamentales pour manipuler la position, l'orientation, et la taille des objets dans une scène 2D ou 3D. Ces transformations sont généralement représentées par des matrices et appliquées aux coordonnées des points définissant les objets. Ce chapitre explore différentes transformations de modélisation, leur composition, l'organisation hiérarchique des scènes et les projections pour visualiser des objets 3D sur un écran 2D.

3.1.1 Glissement (Shear)

Definition 3.1.1. Le **glissement**, aussi appelé cisaillement (shear), est une transformation qui déforme un objet en déplaçant ses points le long de lignes parallèles. C'est un étirement non uniforme suivant un axe, dont l'amplitude dépend de la distance à une ligne ou un plan de référence.

Glissement parallèle à l'axe x

La matrice d'un glissement parallèle à l'axe des x, où la coordonnée x d'un point est augmentée proportionnellement à sa coordonnée y avec un rapport k, est donnée par :

$$M_{shear\_x} = \begin{pmatrix} 1 & k & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Un point $P = (x, y, z, 1)^T$ est transformé en $P' = (x + ky, y, z, 1)^T$.

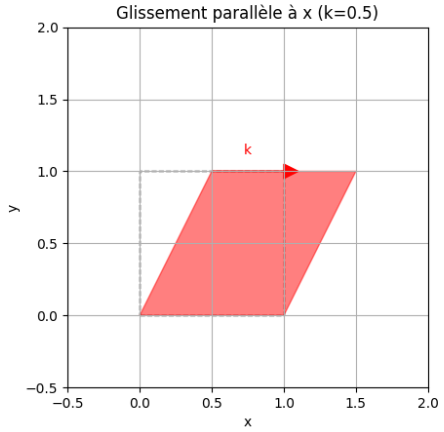

Figure 3.1: Illustration d'un glissement parallèle à l'axe x.

Glissement parallèle à l'axe x avec ligne de base

Si le glissement parallèle à l'axe x de rapport k s'effectue par rapport à une ligne de base horizontale $y = y_{ref}$, la matrice de transformation devient :

$$M_{shear\_x\_ref} = \begin{pmatrix} 1 & k & 0 & -k \cdot y_{ref} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Un point $P = (x, y, z, 1)^T$ est transformé en $P' = (x + k(y - y_{ref}), y, z, 1)^T$.

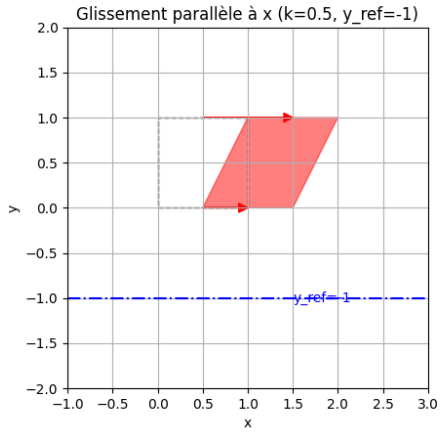

Figure 3.2: Illustration d'un glissement parallèle à l'axe x par rapport à une ligne de base y_{ref}.

3.2 Composition de Transformations

Il est souvent nécessaire d'appliquer plusieurs transformations successives à un objet. La composition de transformations consiste à combiner plusieurs matrices de transformation en une seule matrice équivalente.

3.2.1 Multiplication de matrices

Si un point p est d'abord transformé par une matrice M_1 puis par une matrice M_2, le point résultant p'' est obtenu par $p'' = M_2 \cdot (M_1 \cdot p)$. Grâce à l'associativité de la multiplication matricielle, cela peut s'écrire $p'' = (M_2 \cdot M_1) \cdot p$. La matrice composée $M = M_2 \cdot M_1$ représente l'effet combiné des deux transformations.

Example 3.2.1 (Homothétie puis Translation)**.** Considérons un point p auquel on applique d'abord une mise à l'échelle $S(s_x, s_y)$ puis une translation $T(t_x, t_y)$. Le point transformé p' est $p' = T \cdot (S \cdot p) = (T \cdot S) \cdot p$.

Soit $S = S(2,2)$ et $T = T(3,1)$. Les matrices correspondantes en coordonnées homogènes 2D (en étendant à 3x3 pour simplifier) sont :

$$S = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

La matrice composée $T \cdot S$ est :

$$T \cdot S = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} (1 \cdot 2 + 0 \cdot 0 + 3 \cdot 0) & (1 \cdot 0 + 0 \cdot 2 + 3 \cdot 0) & (1 \cdot 0 + 0 \cdot 0 + 3 \cdot 1) \\ (0 \cdot 2 + 1 \cdot 0 + 1 \cdot 0) & (0 \cdot 0 + 1 \cdot 2 + 1 \cdot 0) & (0 \cdot 0 + 1 \cdot 0 + 1 \cdot 1) \\ (0 \cdot 2 + 0 \cdot 0 + 1 \cdot 0) & (0 \cdot 0 + 0 \cdot 2 + 1 \cdot 0) & (0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1) \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Appliquons cela à un point $p = (1,1)^T$, représenté en coordonnées homogènes par $p_{hom} = (1,1,1)^T$. $S \cdot p_{hom} = (2,2,1)^T$. Le point est mis à l'échelle à $(2,2)$. $T \cdot (S \cdot p_{hom}) = T \cdot (2,2,1)^T = (2+3, 2+1, 1)^T = (5,3,1)^T$. Le point final est $(5,3)$. Vérifions avec la matrice composée : $(T \cdot S) \cdot p_{hom} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 + 0 \cdot 1 + 3 \cdot 1 \\ 0 \cdot 1 + 2 \cdot 1 + 1 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \\ 1 \end{pmatrix}$. Le résultat est identique.

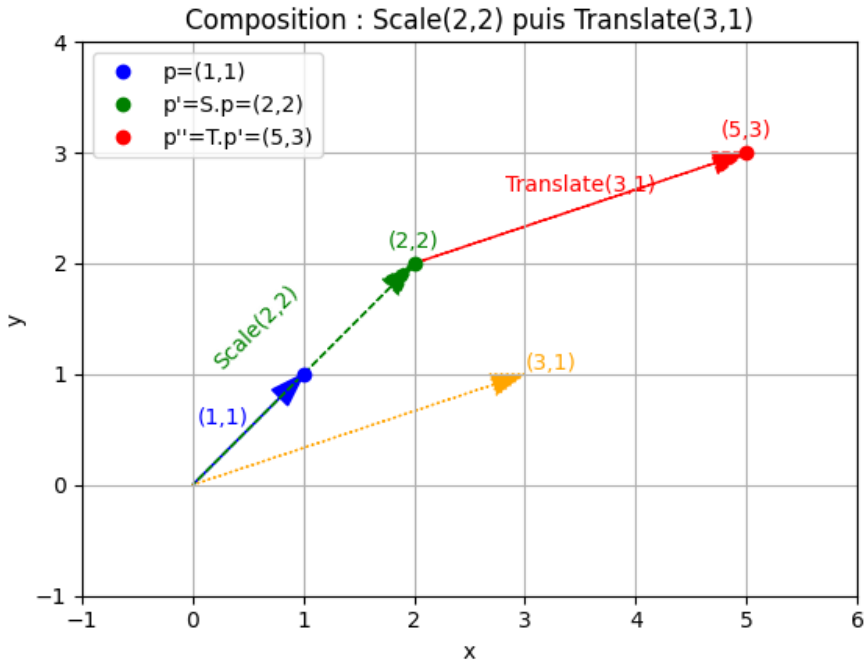

Figure 3.3: Composition d'une homothétie suivie d'une translation.

3.2.2 Non-commutativité

La multiplication de matrices n'est généralement pas commutative, c'est-à-dire $M_1 \cdot M_2 \neq M_2 \cdot M_1$. L'ordre dans lequel les transformations sont appliquées est donc crucial.

Example 3.2.2 (Translation puis Homothétie). Reprenons les transformations $S = S(2,2)$ et $T = T(3,1)$. Appliquons-les dans l'ordre inverse : d'abord la translation, puis l'homothétie. Le point transformé est $p' = S \cdot (T \cdot p) = (S \cdot T) \cdot p$.

La matrice composée $S \cdot T$ est :

$$S{\cdot}T = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} (2\cdot 1 + 0\cdot 0 + 0\cdot 0) & (2\cdot 0 + 0\cdot 1 + 0\cdot 0) & (2\cdot 3 + 0\cdot 1 + 0\cdot 1) \\ (0\cdot 1 + 2\cdot 0 + 0\cdot 0) & (0\cdot 0 + 2\cdot 1 + 0\cdot 0) & (0\cdot 3 + 2\cdot 1 + 0\cdot 1) \\ (0\cdot 1 + 0\cdot 0 + 1\cdot 0) & (0\cdot 0 + 0\cdot 1 + 1\cdot 0) & (0\cdot 3 + 0\cdot 1 + 1\cdot 1) \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

On observe que $S \cdot T \neq T \cdot S$.

Appliquons cela au point $p = (1,1)^T$, soit $p_{hom} = (1,1,1)^T$. $T \cdot p_{hom} = (1+3, 1+1, 1)^T = (4,2,1)^T$. Le point est translaté à $(4,2)$. $S \cdot (T \cdot p_{hom}) = S \cdot (4,2,1)^T = (2\cdot 4, 2\cdot 2, 1)^T = (8,4,1)^T$. Le point final est $(8,4)$. Vérifions avec la matrice composée : $(S{\cdot}T) \cdot p_{hom} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2\cdot 1 + 0\cdot 1 + 6\cdot 1 \\ 0\cdot 1 + 2\cdot 1 + 2\cdot 1 \\ 0\cdot 1 + 0\cdot 1 + 1\cdot 1 \end{pmatrix} = \begin{pmatrix} 8 \\ 4 \\ 1 \end{pmatrix}$. Le résultat est $(8,4)$, ce qui est différent de $(5,3)$ obtenu précédemment.

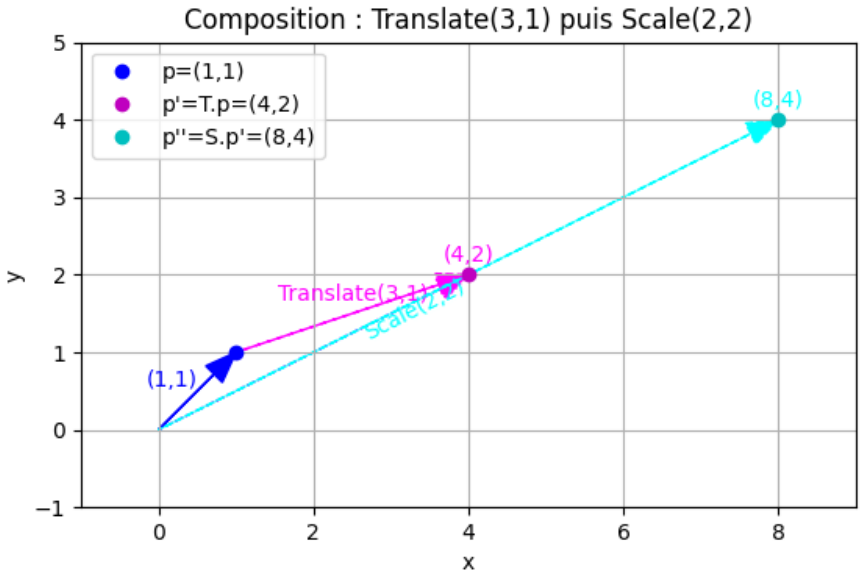

Figure 3.4: Composition d'une translation suivie d'une homothétie (comparez avec la Figure 3.3).

3.2.3 Ordre d'application

Comme $p'' = M_2 \cdot M_1 \cdot p$, la transformation M_1 est appliquée en premier au point p, suivie de la transformation M_2 appliquée au résultat. L'ordre d'application des transformations se lit donc **de droite à gauche** dans

le produit matriciel.

3.2.4 Cas général

Pour une séquence de transformations représentées par les matrices $M_1, M_2, ..., M_n$, appliquées dans cet ordre à un point p, le point transformé p' est donné par :

$$p' = M_n \cdot M_{n-1} \cdot ... \cdot M_2 \cdot M_1 \cdot p$$

On peut précalculer la matrice de transformation globale $M = M_n \cdot M_{n-1} \cdot ... \cdot M_2 \cdot M_1$. Alors, $p' = M \cdot p$.

3.2.5 Transformations par rapport à un point arbitraire

Les transformations élémentaires (rotation, mise à l'échelle) sont définies par rapport à l'origine du repère. Pour effectuer une transformation par rapport à un point arbitraire $P(t_x, t_y, t_z)$, on utilise une composition de trois transformations :

1. Ramener le point P à l'origine en appliquant une translation $T(-t_x, -t_y, -t_z)$.
2. Appliquer la transformation souhaitée (par exemple, une rotation $R(\theta)$ ou une mise à l'échelle $S(s_x, s_y, s_z)$).
3. Replacer le point P à sa position initiale en appliquant la translation inverse $T(t_x, t_y, t_z)$.

La matrice de transformation résultante M est le produit de ces trois matrices :

$$M = T(t_x, t_y, t_z) \cdot R(\theta) \cdot T(-t_x, -t_y, -t_z)$$

ou

$$M = T(t_x, t_y, t_z) \cdot S(s_x, s_y, s_z) \cdot T(-t_x, -t_y, -t_z)$$

Example 3.2.3 (Rotation autour d'un axe passant par P)**.** On désire effectuer une rotation d'angle θ_z autour d'un axe parallèle à l'axe z et passant par le point $P(T_x, T_y, T_z)$. La matrice de transformation M est :

$$M = T(T_x, T_y, T_z) \cdot R_z(\theta_z) \cdot T(-T_x, -T_y, -T_z)$$

où $R_z(\theta_z)$ est la matrice de rotation autour de l'axe z :

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

Les matrices de translation sont :

$$T(T_x, T_y, T_z) = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T(-T_x, -T_y, -T_z) = \begin{pmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Effectuons le produit matriciel $M = T \cdot R_z \cdot T^{-1}$:

$$R_z \cdot T^{-1} = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & -T_x\cos\theta_z + T_y\sin\theta_z \\ \sin\theta_z & \cos\theta_z & 0 & -T_x\sin\theta_z - T_y\cos\theta_z \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M = T \cdot (R_z \cdot T^{-1}) = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & -T_x\cos\theta_z + T_y\sin\theta_z \\ \sin\theta_z & \cos\theta_z & 0 & -T_x\sin\theta_z - T_y\cos\theta_z \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & -T_x\cos\theta_z + T_y\sin\theta_z + T_x \\ \sin\theta_z & \cos\theta_z & 0 & -T_x\sin\theta_z - T_y\cos\theta_z + T_y \\ 0 & 0 & 1 & -T_z + T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & T_x(1-\cos\theta_z) + T_y\sin\theta_z \\ \sin\theta_z & \cos\theta_z & 0 & T_y(1-\cos\theta_z) - T_x\sin\theta_z \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cette matrice M applique directement la rotation souhaitée autour de l'axe passant par $P(T_x, T_y, T_z)$. (Note : La diapositive 4 contenait une erreur dans le calcul final de la matrice M, elle a été corrigée ici.)

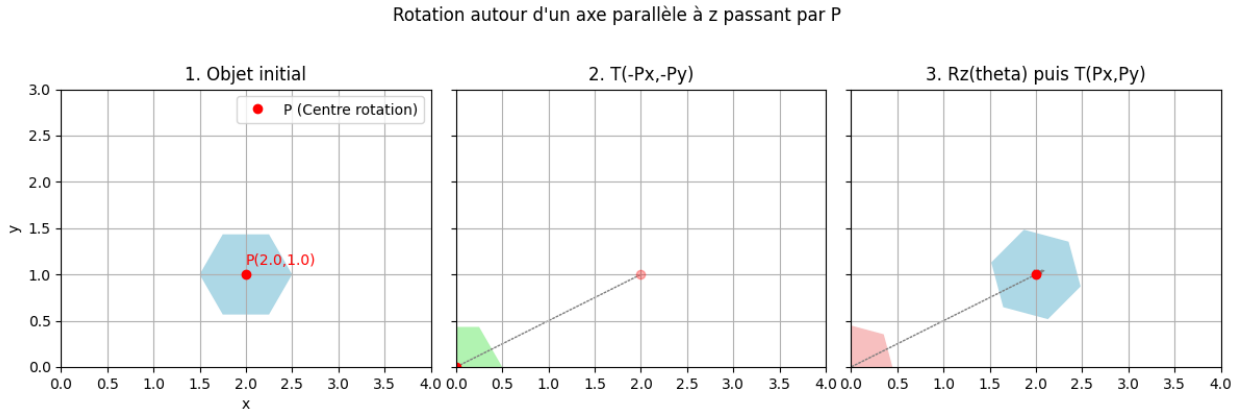

Figure 3.5: Étapes de la rotation d'un objet autour d'un point P.

3.3 Modélisation Hiérarchique

Pour décrire des objets ou des scènes complexes composés de plusieurs parties, on utilise souvent une approche de modélisation hiérarchique.

3.3.1 Concept et avantages

- Un modèle hiérarchique permet de décrire facilement des objets complexes composés d'objets plus simples (primitives).
- La scène est organisée sous forme d'une structure arborescente (ou plus généralement, un graphe).
- Les objets ne sont plus définis par leur transformation absolue par rapport au repère global (monde), mais par leur transformation relative par rapport à leur parent dans l'arbre.
- Le repère associé à la racine de l'arbre est le repère de la scène (repère monde).
- À chaque nœud de l'arbre est associé un repère local.
- À chaque arc (reliant un parent à un enfant) est associée une transformation géométrique qui positionne l'objet fils dans le repère de son père.
- Un même objet (ou sous-arbre) peut être inclus plusieurs fois dans la hiérarchie (instanciation), ce qui permet de réutiliser des modèles.
- La structure résultante est un graphe orienté acyclique (Directed Acyclic Graph - DAG), car un objet ne peut pas être son propre ancêtre.

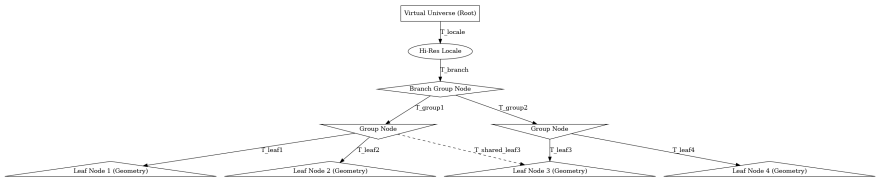

Figure 3.6: Exemple de structure hiérarchique sous forme de Graphe Orienté Acyclique (DAG).

3.3.2 Construction du graphe

La construction d'un modèle hiérarchique se fait souvent par un processus descendant ("top-down") :

1. On part de l'objet complet (racine).
2. On le décompose récursivement en parties plus simples (nœuds internes).
3. On continue la décomposition jusqu'à obtenir des objets élémentaires indécomposables, appelés primitives géométriques (feuilles de l'arbre).

La séquence de transformations le long d'un chemin depuis la racine jusqu'à un nœud feuille définit la position et l'orientation finale de la primitive correspondante dans le repère monde. Cette séquence est appelée la chaîne cinématique.

Example 3.3.1 (Modélisation d'une maison simple)**.** On peut modéliser une maison comme un assemblage d'un corps principal et d'un toit. Le corps principal peut être décomposé en murs, fenêtres, porte. Les fenêtres peuvent être composées d'un cadre et d'une vitre, etc.

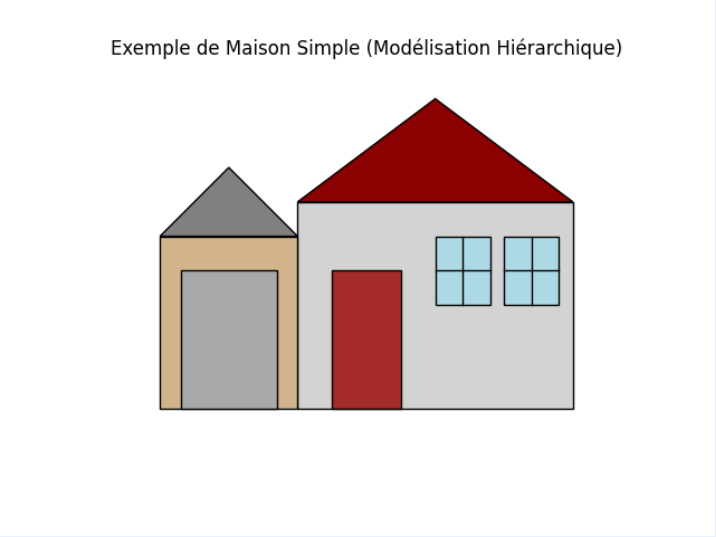

Figure 3.7: Illustration d'une maison simple pouvant être modélisée hiérarchiquement.

Le graphe de scène correspondant pourrait ressembler à ceci :

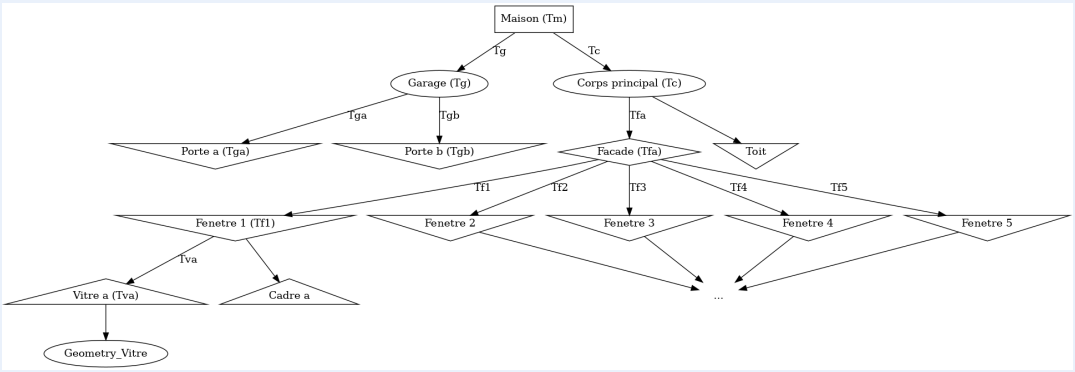

Figure 3.8: Exemple de graphe de scène pour la modélisation d'une maison.

3.3.3 Pile de transformations

Pour déterminer la transformation globale d'un objet feuille dans le repère monde, on multiplie toutes les matrices de transformation rencontrées le long du chemin depuis la racine jusqu'à la feuille. C'est la "Current Transformation Matrix" (CTM). Par exemple, pour trouver la transformation de la 'Vitre a' dans la Figure 3.8 par rapport à la 'Maison' (racine), on calcule :

$$M_{vitreA} = T_m \cdot T_c \cdot T_{fa} \cdot T_{f1} \cdot T_{va}$$

où chaque T représente la matrice de transformation associée à l'arc correspondant dans le graphe. L'implémentation se fait souvent à l'aide d'une pile (stack) pour gérer la CTM lors du parcours du graphe.

3.3.4 Instanciation (Instancing)

L'instanciation permet de réutiliser une définition d'objet (géométrie ou sous-graphe) à plusieurs endroits dans la scène sans dupliquer sa description.

- On définit l'objet une seule fois.

- Dans le graphe de scène, on utilise des nœuds "pointeurs" ou des références vers cette définition unique.
- Chaque instance (chaque pointeur) peut avoir sa propre transformation associée sur l'arc entrant, permettant de positionner, orienter et mettre à l'échelle chaque copie différemment.

Cela économise de la mémoire et facilite les modifications : changer la définition originale affecte toutes ses instances.

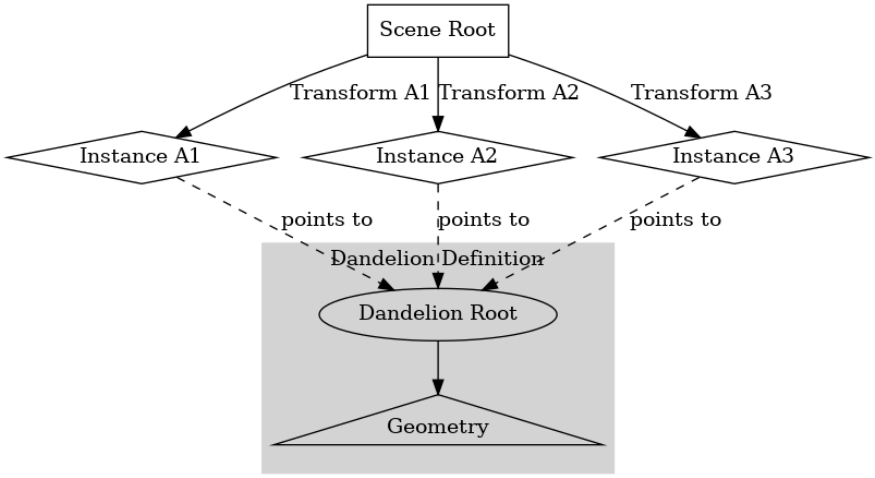

Figure 3.9: Graphe de scène illustrant l'instanciation : plusieurs nœuds (Instances A1, A2, A3) pointent vers la même définition d'objet (Dandelion) mais avec des transformations différentes.

3.4 Changement de Repère

Il est souvent nécessaire d'exprimer les coordonnées d'un point ou la description d'un objet dans différents systèmes de coordonnées (repères). Par exemple :

- Repère objet : Coordonnées locales à l'objet.
- Repère scène (ou monde) : Coordonnées globales de la scène.
- Repère caméra (ou vue) : Coordonnées par rapport à l'observateur.

Le passage d'un repère à un autre s'effectue à l'aide d'une matrice de changement de repère.

3.4.1 Matrice de passage

Soient deux repères R_1 (origine O_1, axes X_1, Y_1, Z_1) et R_2 (origine O_2, axes X_2, Y_2, Z_2). Soit un point P dont les coordonnées sont $P_1 = (x_1, y_1, z_1, 1)^T$ dans R_1 et $P_2 = (x_2, y_2, z_2, 1)^T$ dans R_2.

La matrice de passage $M_{2\to1}$ permet de passer des coordonnées dans R_2 aux coordonnées dans R_1 :

$$P_1 = M_{2\to1} \cdot P_2$$

La matrice $M_{2\to1}$ exprime les vecteurs de base et l'origine de R_2 dans le repère R_1. Si $O_2 = (T_x, T_y, T_z)$ dans R_1 et les vecteurs directeurs des axes X_2, Y_2, Z_2 ont pour coordonnées (R_{11}, R_{21}, R_{31}), (R_{12}, R_{22}, R_{32}), et (R_{13}, R_{23}, R_{33}) respectivement dans R_1, alors :

$$M_{2\to1} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_2|_{R1} & Y_2|_{R1} & Z_2|_{R1} & O_2|_{R1} \\ \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La sous-matrice 3x3 contenant les R_{ij} est une matrice de rotation (si les repères sont orthonormés et de même orientation) et le vecteur (T_x, T_y, T_z) est le vecteur de translation de O_1 vers O_2 exprimé dans R_1.

Pour passer des coordonnées de R_1 à R_2, on utilise la matrice inverse $M_{1\to2} = (M_{2\to1})^{-1}$:

$$P_2 = M_{1\to2} \cdot P_1 = (M_{2\to1})^{-1} \cdot P_1$$

Si $M_{2\to1}$ représente une transformation rigide (rotation + translation), son inverse est relativement simple à calculer :

$$(M_{2\to1})^{-1} = \begin{pmatrix} R^T & -R^T \cdot T \\ 0 & 1 \end{pmatrix}$$

où R est la matrice de rotation 3x3 et T est le vecteur de translation $(T_x, T_y, T_z)^T$.

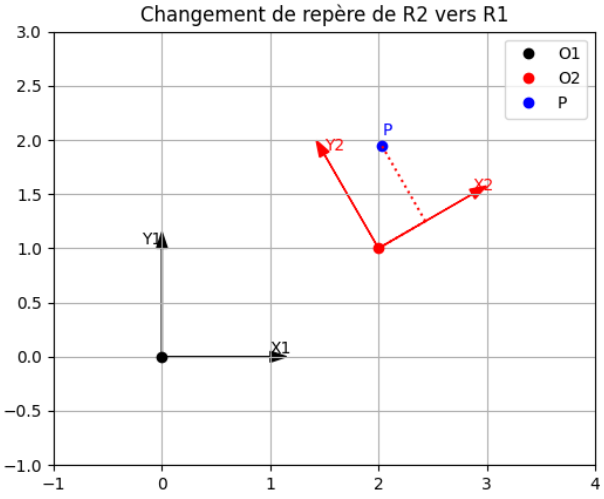

Figure 3.10: Illustration du changement de repère. Les coordonnées du point P peuvent être exprimées dans R1 ou R2.

3.5 Projections

La projection est le processus qui transforme les objets d'un espace 3D en une représentation 2D, telle qu'elle apparaîtrait sur un écran ou une image. C'est une étape essentielle du pipeline graphique.

3.5.1 Pipeline Graphique simplifié

Les étapes principales pour afficher une scène 3D incluent :

1. **Transformations de modélisation :** Positionner/orienter/mettre à l'échelle les objets dans le repère monde.
2. **Transformation d'affichage (vue) :** Placer la caméra et transformer les coordonnées du repère monde vers le repère caméra.
3. **Illumination (Shading) :** Calculer les couleurs des objets en fonction des lumières et des matériaux (pas traité ici).
4. **Clipping :** Éliminer les parties de la scène qui sont en dehors du volume de vue de la caméra.
5. **Projection :** Transformer les coordonnées 3D (du repère caméra) en coordonnées 2D sur le plan de projection.

6. **Transformation écran (Viewport) :** Mapper les coordonnées 2D projetées vers les coordonnées pixels de l'écran.

7. **Pixelisation (Rasterization) :** Déterminer quels pixels sont couverts par chaque primitive projetée.

8. **Visibilité / Rendu :** Déterminer quelles surfaces sont visibles (élimination des parties cachées) et affecter les couleurs finales aux pixels.

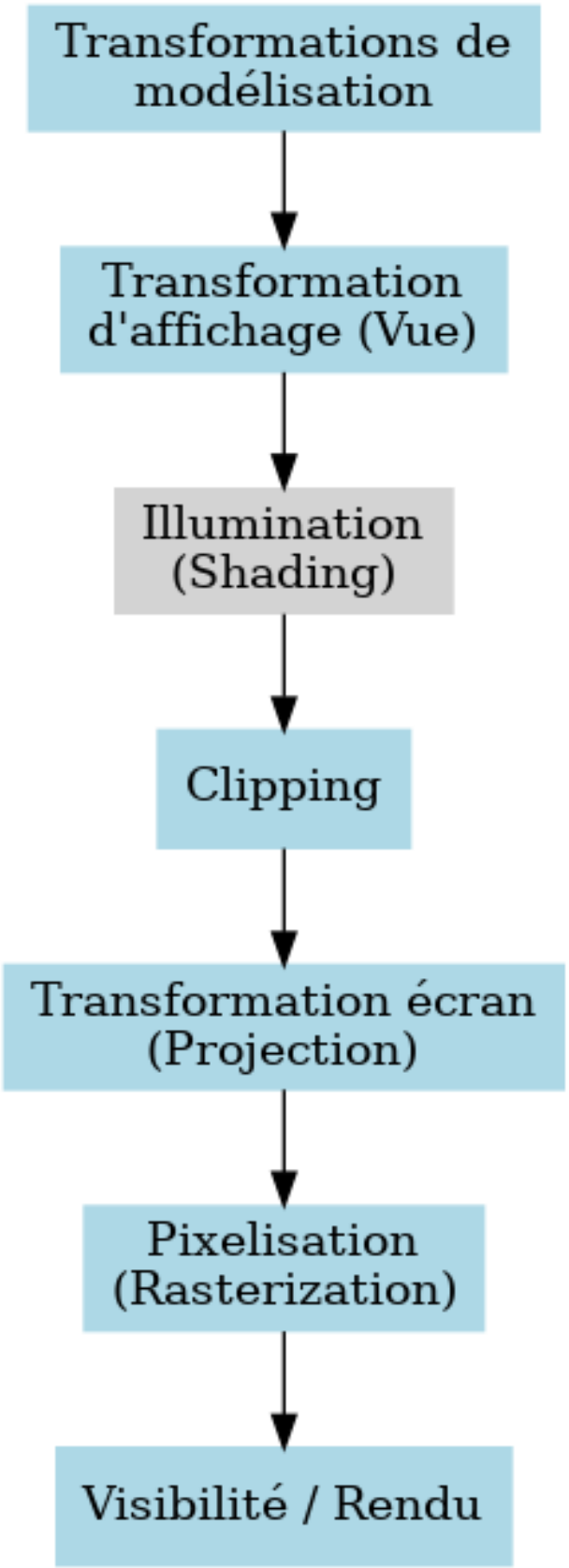

Figure 3.11: Étapes principales du pipeline graphique.

3.5.2 Définition et Types

La projection consiste à définir des **projecteurs** (lignes de projection) qui passent par chaque point de l'objet 3D et intersectent un **plan de projection** (ou plan de vue). L'ensemble de ces intersections forme l'image 2D projetée de l'objet.

Il existe deux grandes familles de projections géométriques :

1. **Projection Perspective :** Les projecteurs convergent en un point unique appelé **Centre de Projection (CP)** ou point de vue. Ce CP est situé à une distance finie du plan de projection. Elle simule la vision humaine et photographique (les objets lointains paraissent plus petits).

2. **Projection Parallèle :** Les projecteurs sont parallèles entre eux. Cela correspond à un CP situé à l'infini. Elle préserve les rapports de taille et les angles pour les faces parallèles au plan de projection, utile pour les dessins techniques.

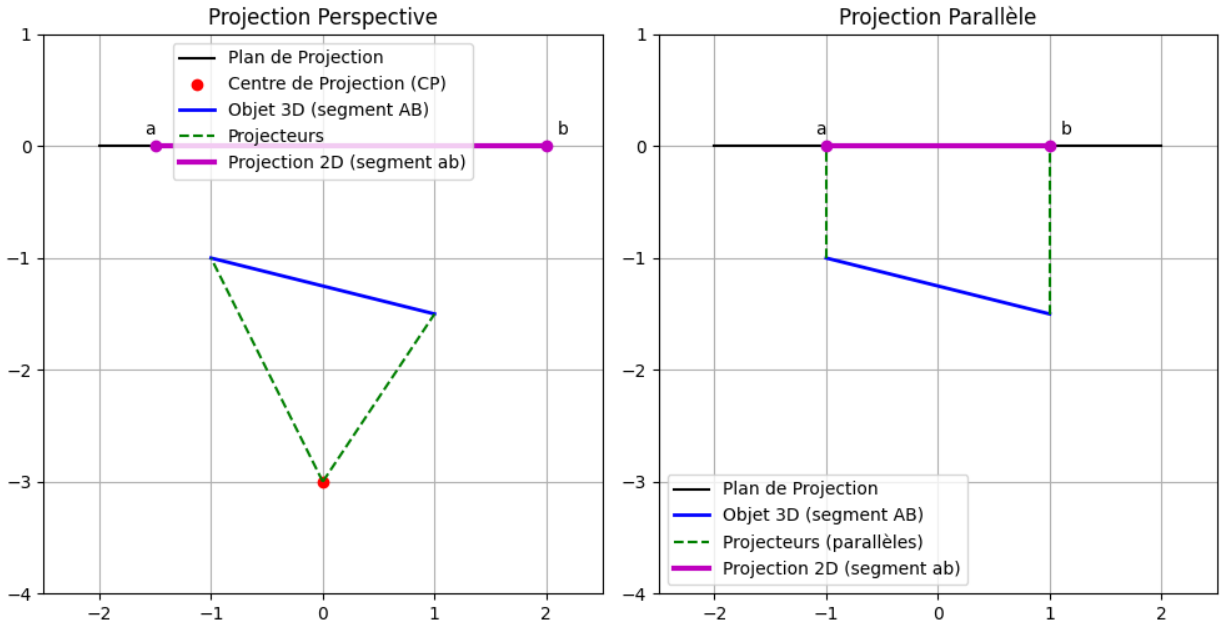

Figure 3.12: Comparaison entre projection perspective (gauche) et parallèle (droite).

3.5.3 Taxonomie des Projections

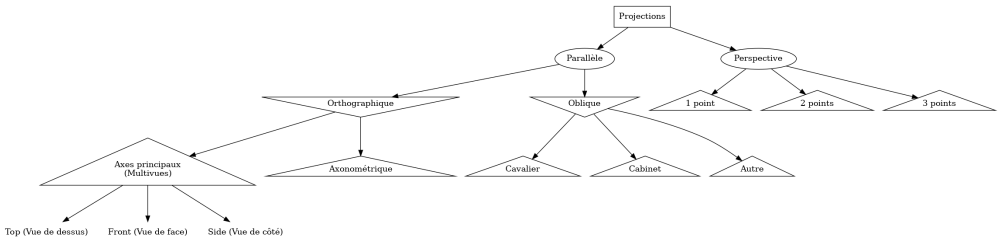

Figure 3.13: Classification des projections géométriques.

3.5.4 Projections Parallèles

Dans une projection parallèle, la direction de projection (DP) est constante pour tous les points. **Propriétés :**

- Les lignes parallèles dans l'espace 3D restent parallèles dans l'image 2D projetée.
- Les rapports des distances le long d'une direction donnée sont conservés.
- Peu réaliste (pas d'effet de perspective), mais utile pour les dessins techniques et les mesures exactes car les dimensions relatives sont préservées pour les faces parallèles au plan de projection.

Projection Orthographique

Les projecteurs sont parallèles entre eux ET perpendiculaires au plan de projection.

- **Vues Multiples (Axes principaux) :** Le plan de projection est parallèle à l'un des plans de coordonnées principaux (xy, yz, xz). On obtient les vues de face (Front), de dessus (Top), et de côté (Side). Mathématiquement, cela revient à annuler une des coordonnées. Par exemple, pour une projection sur le plan z=0 (vue de dessus), on met la coordonnée z à 0.
- **Axonométrique :** Le plan de projection n'est pas parallèle aux plans de coordonnées. Plusieurs faces de l'objet sont visibles simultanément. Le vecteur normal au plan de projection n'est parallèle à aucun des axes de coordonnées. (Isométrique, Dimétrique, Trimétrique selon les angles).

Matrice de Projection Orthographique (sur plan z=0) : Pour projeter un point $P = (x, y, z, 1)^T$ sur le plan $z = 0$ (vue de dessus), le point projeté est $P' = (x, y, 0, 1)^T$. La matrice correspondante est :

$$M_{ortho\_z} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

De même pour les projections sur les plans x=0 et y=0 :

$$M_{ortho\_x} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad M_{ortho\_y} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Projection Oblique

Les projecteurs sont parallèles entre eux mais **non perpendiculaires** au plan de projection. L'angle ϕ entre les projecteurs et le plan est différent de 90°.

- Les faces parallèles au plan de projection sont projetées sans déformation.
- Les faces perpendiculaires au plan de projection sont déformées (lignes projetées avec un angle α et un facteur de raccourcissement f).
- **Projection Cavalière :** L'angle $\phi = 45$. Les lignes perpendiculaires au plan de projection ne sont pas raccourcies ($f = 1$). L'angle α est souvent choisi à 30° ou 45°. Apparence déformée.
- **Projection Cabinet :** L'angle $\phi \approx 63.4$ ($\cot \phi = 1/2$). Les lignes perpendiculaires au plan sont raccourcies de moitié ($f = 1/2$). Apparence plus réaliste que la cavalière. L'angle α est souvent choisi à 30° ou 45°.

La matrice d'une projection oblique est une combinaison d'une transformation de glissement (shear) suivie d'une projection orthographique.

3.5.5 Projection Perspective

Dans une projection perspective, les projecteurs émanent d'un point unique, le Centre de Projection (CP).
Propriétés :

- Non-linéaire (en coordonnées cartésiennes, mais linéaire en coordonnées homogènes).
- Ne conserve pas le parallélisme (sauf pour les lignes parallèles au plan de projection).
- Ne conserve pas les angles ni les rapports de distance (sauf cas particuliers).
- La taille projetée d'un objet diminue lorsque sa distance au CP augmente (effet de perspective).
- Les ensembles de lignes parallèles dans l'espace 3D (qui ne sont pas parallèles au plan de projection) convergent vers un point unique dans l'image projetée, appelé **point de fuite**.
- Le nombre de points de fuite (1, 2 ou 3) dépend de l'orientation des axes principaux de l'objet par rapport au plan de projection.

Calcul de la Projection Perspective (Vue de côté simplifiée)

Considérons une caméra simple (sténopé) avec le CP à l'origine (0,0,0) et le plan de projection à $z = -f$ (ou $z = f$ selon la convention, ici utilisons $z = f$ comme sur certaines slides, bien que $z = -f$ soit plus courant pour un système droitier regardant vers -z). Soit un point 3D $P(x, y, z)$ et son projeté $q(u, v)$ sur le plan $z = f$.

Par Thalès (triangles semblables) :

$$\frac{u}{x} = \frac{f}{z} \implies u = \frac{x \cdot f}{z} = \frac{x}{z/f}$$

$$\frac{v}{y} = \frac{f}{z} \implies v = \frac{y \cdot f}{z} = \frac{y}{z/f}$$

Si $f = 1$ (plan de projection unitaire) :

$$u = \frac{x}{z}, \quad v = \frac{y}{z}$$

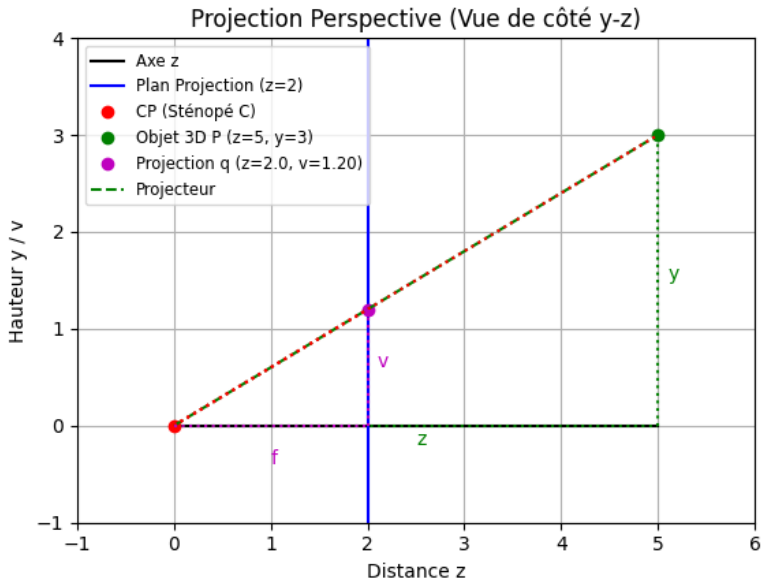

Figure 3.14: Calcul de la projection perspective en utilisant les triangles semblables (vue de côté y-z).

Matrice de Projection Perspective (simple)

En coordonnées homogènes, la projection perspective $u = xf/z$, $v = yf/z$ peut être représentée par une matrice. L'astuce est que la division par z est effectuée lors de la renormalisation des coordonnées homogènes (division par la 4ème composante w). On cherche une matrice M_{persp} telle que :

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = M_{persp} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \text{et} \quad u = \frac{x'}{w'}, v = \frac{y'}{w'}$$

Une matrice possible (qui projette sur $z = 0$ après division, et préserve z pour le Z-buffering, avec CP à l'origine) est :

$$M_{persp} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \text{ou plus couramment} \quad M_{persp} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{pmatrix}$$

Appliquons la seconde matrice :

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/f \end{pmatrix}$$

Après division par $w' = z/f$, on obtient les coordonnées cartésiennes projetées :

$$u = \frac{x'}{w'} = \frac{x}{z/f} = \frac{xf}{z}$$

$$v = \frac{y'}{w'} = \frac{y}{z/f} = \frac{yf}{z}$$

La coordonnée $z_{proj} = z'/w' = z/(z/f) = f$ est constante (tous les points se projettent sur le plan $z = f$), ce qui n'est pas idéal pour la gestion de la profondeur (Z-buffer). Des matrices plus complexes sont utilisées en pratique pour mapper l'intervalle de profondeur $[z_{near}, z_{far}]$ vers un intervalle normalisé (e.g., $[-1, 1]$).

La matrice présentée sur la diapositive 16 (droite) semble correspondre à une projection sur $z = 0$ avec le CP à $(0, 0, -f)$ regardant vers $+z$. $x_p = \frac{x}{-z/f+1}$, $y_p = \frac{y}{-z/f+1}$. La matrice homogène correspondante est :

$$M_{persp} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/f & 1 \end{pmatrix}$$

Vérifions :

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = M_{persp} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ -z/f + 1 \end{pmatrix}$$

Après division par $w' = -z/f + 1$:

$$x_p = \frac{x}{-z/f + 1}, \quad y_p = \frac{y}{-z/f + 1}$$

Ceci correspond aux formules données.

Effet de la distance focale (f)

- Plus f est **grand** (téléobjectif), plus la projection tend vers une projection parallèle (orthographique). La profondeur a peu d'effet sur la taille projetée, la perspective est moins marquée ("écrasée").
- Plus f est **petit** (grand angle), plus la projection diverge. La profondeur a une influence considérable, les effets de perspective sont exagérés (distorsion possible près des bords).

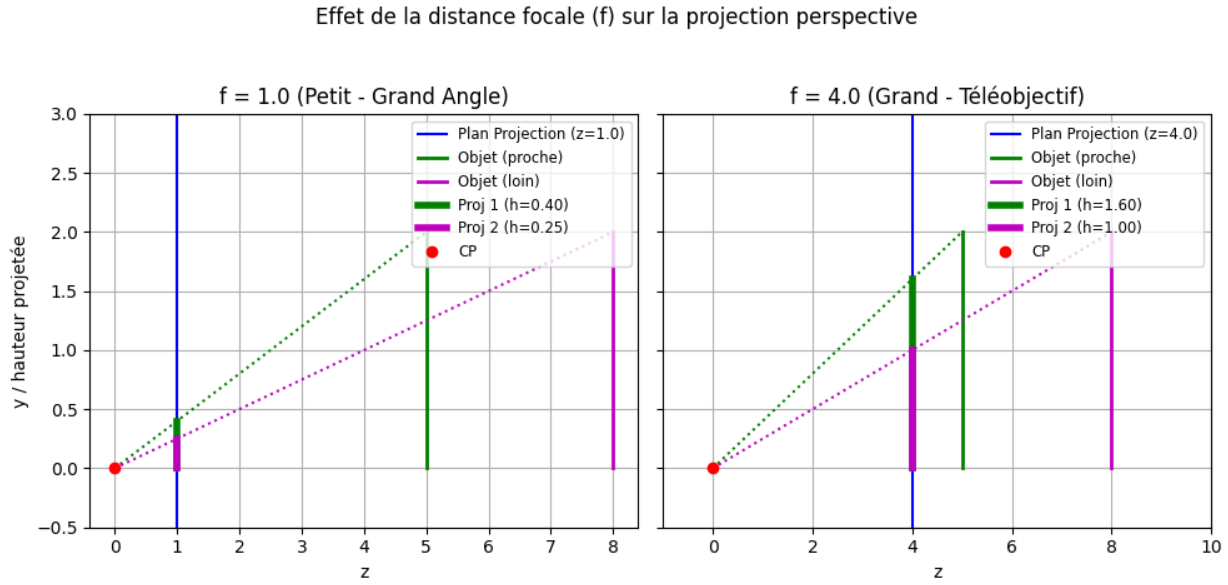

Figure 3.15: Comparaison de la taille projetée pour un objet à deux distances différentes, avec une petite distance focale (gauche) et une grande distance focale (droite). La différence de taille projetée est plus marquée avec un petit f.

Volume de vue (View Frustum) et Clipping

En pratique, on ne projette pas tout l'espace 3D. On définit un volume de vue (View Frustum), généralement une pyramide tronquée, qui délimite la portion de l'espace visible par la caméra. Ce volume est défini par 6 plans :

- Plan proche (Near plane) : $z = z_{near}$ (ou $z = -n$)
- Plan lointain (Far plane) : $z = z_{far}$ (ou $z = -f$)
- Plans gauche/droite/haut/bas (Left/Right/Top/Bottom) : Ces plans passent par le CP et les bords de la "fenêtre" définie sur le plan proche.

Le processus de **clipping** consiste à éliminer toutes les primitives géométriques (ou parties de primitives) qui se trouvent en dehors de ce volume de vue.

Pourquoi les plans Near et Far ?

- Éviter la division par zéro (ou un nombre très petit) dans les calculs de projection si z est proche de 0 (CP). Le plan Near impose $z \geq z_{near} > 0$.
- Gérer les primitives qui pourraient se trouver derrière la caméra.
- Limiter la plage de valeurs de profondeur à traiter, ce qui est crucial pour la précision du **Z-buffer** (tampon de profondeur). Le Z-buffer stocke la valeur de profondeur pour chaque pixel afin de déterminer

quelle surface est visible. Il utilise une précision limitée (e.g., float 16/24/32 bits). Une plage de profondeur $[z_{near}, z_{far}]$ trop grande, surtout avec un z_{near} très petit, peut entraîner des erreurs d'arrondi et des artefacts visuels ("Z-fighting") où des surfaces proches se "battent" pour savoir laquelle est devant.

Passage en Coordonnées Normalisées (NDC)

Avant la projection finale en 2D et le mapping vers l'écran, il est courant de normaliser le volume de vue (le frustum perspectif) en un volume canonique, généralement un cube $[-1, 1]^3$. Cet espace est appelé Normalized Device Coordinates (NDC). Cette transformation simplifie les étapes ultérieures comme le clipping (qui se fait alors contre les faces du cube unité) et la projection orthographique finale (qui devient triviale).

La matrice de transformation $M_{persp \to NDC}$ qui mappe le frustum perspectif défini par $(left, right, bottom, top, near, far)$ vers le cube NDC $[-1, 1]^3$ est donnée par :

$$M_{persp \to NDC} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

(Note: Cette matrice suppose un système de coordonnées caméra regardant vers -z, avec near=n¿0 et far=f¿n. Des variations existent selon les conventions +/-z, n/f positifs/négatifs).

Après application de cette matrice et division par w, un point (x_c, y_c, z_c) dans le repère caméra qui était à l'intérieur du frustum se retrouvera avec des coordonnées $(x_{ndc}, y_{ndc}, z_{ndc})$ comprises entre -1 et 1.

Transformation complète

La transformation complète d'un point P_w en coordonnées monde vers ses coordonnées finales P_{screen} sur l'écran implique la chaîne suivante :

$$P_{screen} = M_{viewport} \cdot M_{ortho} \cdot M_{persp \to NDC} \cdot M_{world \to cam} \cdot P_w$$

où :

- $M_{world \to cam}$: Transformation du repère monde au repère caméra (changement de base).
- $M_{persp \to NDC}$: Transformation du frustum perspectif vers le cube NDC.
- M_{ortho} : Projection orthographique triviale (ignorer z, ou la matrice identité si z est déjà géré).
- $M_{viewport}$: Transformation des coordonnées NDC vers les coordonnées pixels de la fenêtre d'affichage (Viewport).

Souvent, $M_{ortho} \cdot M_{persp \to NDC}$ est combinée en une seule matrice de projection. La combinaison $M_{proj} \cdot M_{view} = M_{persp \to NDC} \cdot M_{world \to cam}$ est fréquemment précalculée.

Chapter 4

Part 4

4.1 Illumination

4.1.1 Pipeline Graphique

L'illumination est une étape clé dans le pipeline graphique 3D. Elle intervient après les transformations de modélisation et avant la transformation d'affichage.

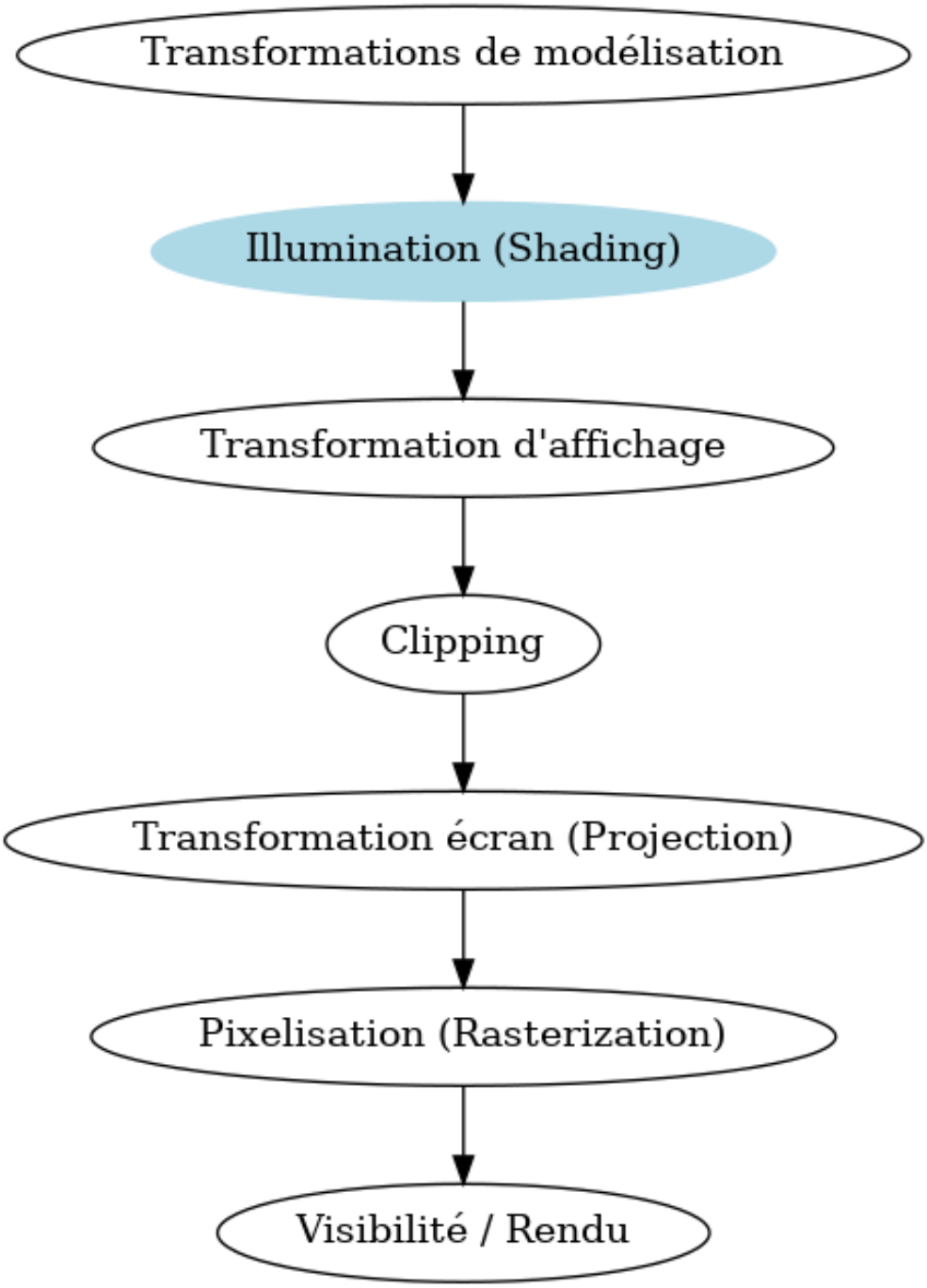

Figure 4.1: Place de l'Illumination dans le Pipeline Graphique.

4.1.2 Introduction à l'Illumination

- Les primitives (points, lignes, polygones) sont éclairées selon leur matériau, le type de surface et les sources de lumière environnantes.
- Les modèles d'illumination sont généralement locaux, ce qui signifie que le calcul de l'éclairage pour une primitive ne prend pas en compte les ombres portées par d'autres objets. Le calcul est effectué par primitive en utilisant des modèles comme diffus, ambiant, Gouraud, Phong, etc.

L'illumination comprend deux aspects principaux :

- **Ombrage (éclairement & shading):** Détermine comment la lumière interagit avec la surface de l'objet pour définir sa couleur et sa luminosité.
- **Texture (plaquage, mappage):** Applique des détails de surface (images, motifs) sur les objets pour augmenter le réalisme.

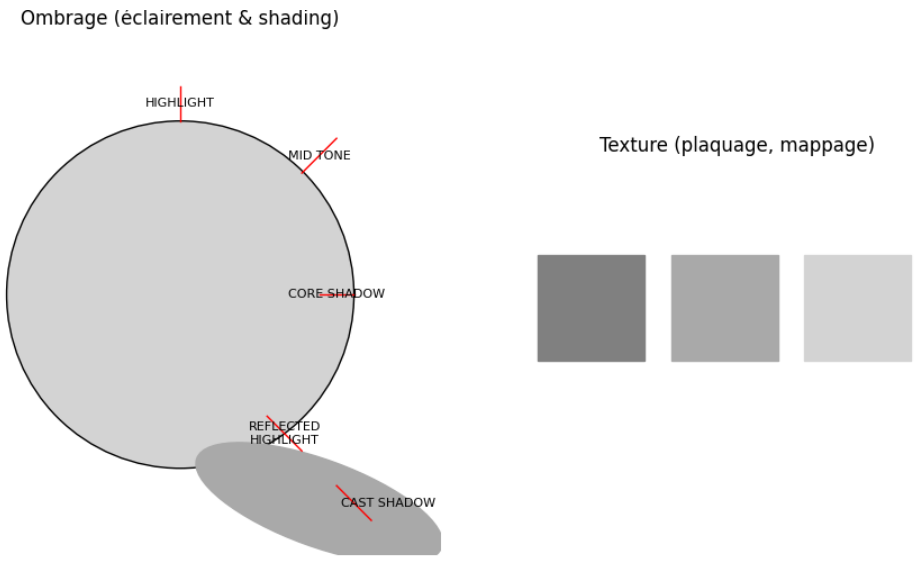

Figure 4.2: Illustration de l'Ombrage et de la Texture.

4.2 Définitions

Definition 4.2.1 (Illumination)**.** Transport du flux lumineux direct ou indirect depuis les sources lumineuses vers les surfaces de la scène. On distingue les modèles locaux (interaction directe source-surface) des modèles globaux (prenant en compte les interactions entre objets, comme les ombres portées ou les réflexions indirectes).

Definition 4.2.2 (Éclairement)**.** Calcul de l'intensité lumineuse en un point précis de la scène. Il modélise l'interaction entre une source lumineuse et le point éclairé.

Definition 4.2.3 (Ombrage (Shading))**.** Utilisation du modèle d'éclairement pour déterminer la couleur finale d'un pixel à l'écran, souvent par interpolation des valeurs calculées aux sommets des primitives.

4.3 Éclairement et Ombrage

Le calcul de l'éclairement et de l'ombrage en un point dépend de plusieurs facteurs :

- La position du point dans l'espace.
- L'orientation de la surface au niveau de ce point (définie par la normale à la surface).
- Les caractéristiques de la surface (comment elle diffuse ou réfléchit la lumière, sa transparence, sa couleur, sa texture...).

- Les sources de lumière présentes dans la scène (leur type : directionnelles, ponctuelles, etc., leur intensité, leur couleur).
- La position et l'orientation de la "caméra" ou du point de vue de l'observateur.

4.4 Sources de Lumières

4.4.1 Lumière ambiante

- Représente une lumière de fond qui éclaire toute la scène uniformément, simulant la lumière indirecte réfléchie par l'environnement.
- Elle n'a pas de direction ni de position spécifique.
- Est uniquement caractérisée par son **intensité** (et sa couleur).

4.4.2 Sources ponctuelles

- Sources de lumière placées en un point précis de l'espace.
- Elles émettent de la lumière radialement dans toutes les directions (isotropique) ou dans des directions privilégiées (anisotropique).
- Caractérisées par leur **intensité**, leur **position** et une fonction d'**atténuation** ('falloff') qui décrit comment l'intensité diminue avec la distance.
- Lorsque la source ponctuelle prend du volume (n'est plus un point mathématique), elle devient une "source étendue".

4.4.3 Sources directionnelles

- Simulent une source de lumière très éloignée (comme le soleil).
- Éclairent la scène avec des rayons lumineux parallèles provenant tous d'une même direction.
- Caractérisées par leur **intensité** et leur **direction**. La position n'est pas pertinente.

4.4.4 Sources projecteur ou spot

- Sources de lumière qui émettent un cône de lumière dans une direction spécifique.
- Caractérisées par leur **position**, leur **direction**, un **angle d'ouverture** du cône et un **facteur de concentration** (comment l'intensité diminue du centre vers les bords du cône).

Note: Les diagrammes illustrant les sources lumineuses ne sont pas recréés ici, mais leur description est fournie ci-dessus.

4.5 Modèles d'Éclairement

Plusieurs modèles sont utilisés pour simuler l'interaction de la lumière avec les surfaces :

- Lumière émise
- Lumière ambiante
- Réflexion diffuse
- Réflexion spéculaire
- Brillance (associée à la réflexion spéculaire)

4.5.1 Lumière Émise

- Certains objets peuvent eux-mêmes émettre de la lumière (par exemple, une lampe, un écran).
- Dans les modèles locaux simples, on considère souvent que les objets ne sont pas intrinsèquement émetteurs de lumière. Ils ne font que réfléchir la lumière des sources externes.
- La lumière émise peut être considérée comme un niveau minimum d'éclairement propre à l'objet, indépendant des sources externes.

4.5.2 Lumière Ambiante

- Correspond au modèle d'éclairage le plus simple.
- On considère qu'il existe une source lumineuse virtuelle présente partout, éclairant de manière égale dans toutes les directions.
- Physiquement, cela approxime la lumière indirecte réfléchie par l'environnement global (par exemple, la lumière du ciel ou les réflexions multiples dans une pièce).
- Ce modèle assure qu'aucun objet, même non directement éclairé par une source ponctuelle ou directionnelle, ne soit complètement noir. Il fournit un niveau minimum d'éclairage appliqué sur tous les objets.
- On définit l'intensité résultant de la lumière ambiante réfléchie par une surface (I_p) comme :

$$I_p = p_a \cdot I_a$$

  où :

  - I_a est l'intensité de la lumière ambiante incidente.
  - p_a est le coefficient de réflexion ambiante de la surface (une propriété du matériau, comprise entre 0 et 1, souvent de la même couleur que la réflexion diffuse).
  - I_p est l'intensité (couleur) résultante réfléchie par la surface due à la composante ambiante.
- Cette intensité I_p est constante sur toute la surface de l'objet, quelle que soit son orientation ou la position de l'observateur.

Propriétés :

- Ne permet pas de percevoir la forme 3D des objets car l'éclairage est uniforme.
- Modélise simplement l'interréflexion globale entre toutes les surfaces d'une scène.
- Évite qu'un objet dans l'ombre soit complètement noir.

Note: L'illustration des théières avec augmentation de p_a montre simplement des objets devenant uniformément plus clairs.

4.5.3 Réflexion Diffuse

- Simule la réflexion de la lumière sur des surfaces mates ou rugueuses (non brillantes).
- Si on considère une surface **Lambertienne** (idéalement diffuse), la lumière incidente est réfléchie de manière égale dans toutes les directions, indépendamment de la position de l'observateur.
- L'intensité de la lumière réfléchie dépend de :

  - I_l : L'intensité de la source lumineuse incidente.

- θ : L'angle entre le vecteur pointant vers la source lumineuse ($\vec{L}$) et la normale à la surface ($\vec{N}$) au point considéré.
- p_d : Le coefficient de réflexion diffuse de la surface (propriété du matériau, $0 \leq p_d \leq 1$).

- Principe physique : Une source lumineuse émet une certaine énergie par unité de surface. Selon l'angle d'incidence des rayons lumineux, cette énergie se répartit sur une surface plus ou moins grande de l'objet. Si les rayons sont perpendiculaires à la surface ($\theta = 0$), l'énergie est concentrée sur la plus petite surface possible, résultant en une intensité réfléchie maximale. Plus l'angle θ augmente, plus l'énergie se répartit sur une grande surface, et l'intensité réfléchie diminue. Cette dépendance est modélisée par le terme $\cos(\theta)$.

- La formule pour l'intensité diffuse réfléchie (I_p) est :

$$I_p = p_d \cdot I_l \cdot \cos(\theta)$$

ou en utilisant le produit scalaire des vecteurs normalisés :

$$I_p = p_d \cdot I_l \cdot (\vec{N} \cdot \vec{L})$$

Note : Si $\vec{N} \cdot \vec{L} < 0$, cela signifie que la lumière vient de derrière la surface, donc l'intensité diffuse est nulle ($\cos(\theta)$ est pris comme $\max(0, \cos(\theta))$).

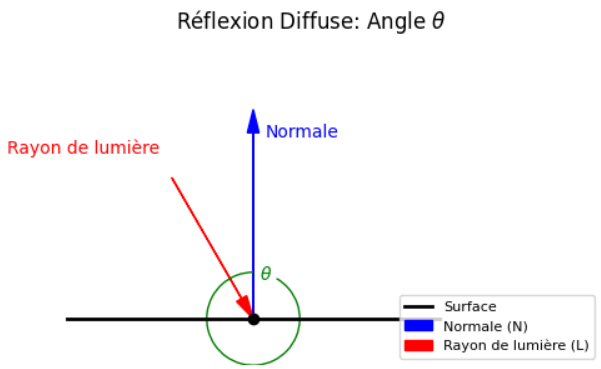

Figure 4.3: Angle θ utilisé dans le calcul de la réflexion diffuse.

Note: L'illustration des théières avec $p_a = 0$ et augmentation de p_d montre des objets dont la luminosité dépend de l'orientation par rapport à la lumière, avec des surfaces face à la lumière plus claires et des surfaces perpendiculaires ou orientées à l'opposé plus sombres.

4.5.4 Modèle Diffuse + Ambiante

On combine souvent les composantes ambiante et diffuse pour obtenir un éclairage plus réaliste :

$$I_{total} = I_{ambiant} + I_{diffus} = p_a I_a + p_d I_l \max(0, \vec{N} \cdot \vec{L})$$

Note: Le diagramme de grille de théières montre l'effet combiné: l'augmentation de p_a (verticalement) augmente la luminosité globale, tandis que l'augmentation de p_d (horizontalement) augmente le contraste dû à l'orientation.

4.5.5 Réflexion Spéculaire

- Simule la réflexion sur des surfaces brillantes ou polies (effet "miroir").
- Contrairement à la réflexion diffuse, la lumière est réfléchie préférentiellement dans une certaine direction.
- **Modèle de Phong [1973]:** Le modèle le plus connu. Il fait intervenir la position de l'observateur. Les rayons lumineux sont réfléchis de manière symétrique par rapport à la normale à la surface.
- L'intensité spéculaire observée dépend de :
  - I_l : L'intensité de la source lumineuse incidente.
  - θ' : L'angle entre le rayon réfléchi idéal ($\vec{R}$) et le vecteur pointant vers le point de vue ($\vec{V}$).
  - p_s : Le coefficient de réflexion spéculaire de la surface ($0 \leq p_s \leq 1$).
  - n : L'exposant de **rugosité** (ou brillance, *shininess*), qui contrôle la netteté du reflet.
- Le rayon réfléchi $\vec{R}$ est calculé par : $\vec{R} = 2(\vec{N} \cdot \vec{L})\vec{N} - \vec{L}$ (avec $\vec{L}$ pointant *vers* la source).
- L'intensité spéculaire est maximale lorsque l'observateur regarde exactement dans la direction du rayon réfléchi ($\theta' = 0$). Elle diminue rapidement lorsque l'angle θ' augmente. Cette diminution est modélisée par $\cos^n(\theta')$.
- La formule pour l'intensité spéculaire de Phong (I_s) est :

  $$I_s = p_s \cdot I_l \cdot (\cos(\theta'))^n$$

  ou en utilisant le produit scalaire des vecteurs normalisés :

  $$I_s = p_s \cdot I_l \cdot (\vec{R} \cdot \vec{V})^n$$

  Note : Si $\vec{R} \cdot \vec{V} < 0$, l'intensité spéculaire est nulle ($\cos(\theta')$ est pris comme $\max(0, \cos(\theta'))$).
- L'exposant n (rugosité / shininess) :
  - Une valeur **élevée** de n (ex: ¿ 100, voire ∞ ou 1024 pour un miroir parfait) produit une petite tâche spéculaire très intense (surface très lisse).
  - Une valeur **faible** de n (ex: 1 à 10) produit une tâche spéculaire large et diffuse (surface plus rugueuse).

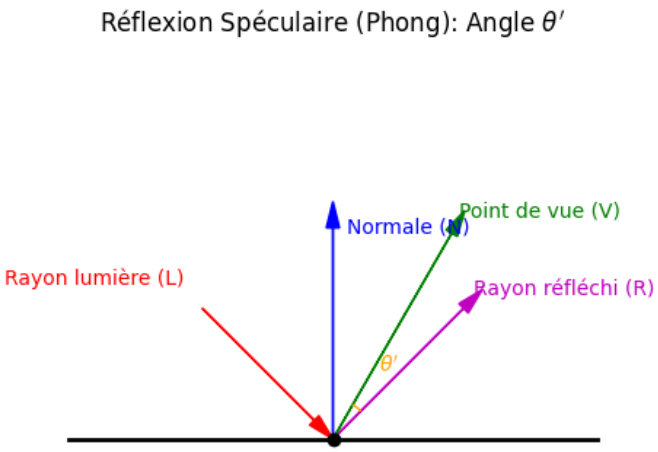

Figure 4.4: Angles et vecteurs pour la réflexion spéculaire de Phong.

- **Modèle Blinn-Phong [1977]:** Une version simplifiée et souvent plus rapide à calculer.
- Au lieu de calculer le rayon réfléchi $\vec{R}$, on calcule le vecteur **demi-angle** (halfway vector) $\vec{H}$, qui est à mi-chemin entre le vecteur lumière $\vec{L}$ et le vecteur vue $\vec{V}$.

$$\vec{H} = \frac{\vec{L} + \vec{V}}{\|\vec{L} + \vec{V}\|}$$

- L'intensité spéculaire dépend alors de l'angle θ'' entre la normale $\vec{N}$ et le vecteur demi-angle $\vec{H}$.
- La formule devient :

$$I_s = p_s \cdot I_l \cdot (\cos(\theta''))^n$$

ou en utilisant le produit scalaire des vecteurs normalisés :

$$I_s = p_s \cdot I_l \cdot (\vec{N} \cdot \vec{H})^n$$

Note : Si $\vec{N} \cdot \vec{H} < 0$, l'intensité spéculaire est nulle ($\cos(\theta'')$ est pris comme $\max(0, \cos(\theta''))$).

- Ce modèle est souvent préféré car $\vec{H}$ est plus simple à calculer que $\vec{R}$, surtout quand $\vec{V}$ et $\vec{L}$ sont constants (vue ou source à l'infini). Les résultats visuels sont très similaires à ceux de Phong, bien que l'exposant n puisse nécessiter un ajustement pour obtenir le même aspect.

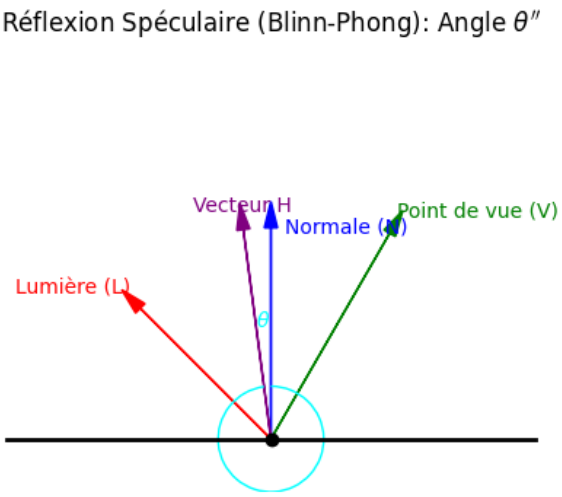

Figure 4.5: Vecteur demi-angle $\vec{H}$ et angle θ'' pour Blinn-Phong.

Note: Les sphères comparant Blinn-Phong et Phong montrent des résultats visuels très similaires.

4.5.6 Modèle d'Éclairement Complet (Phong ou Blinn-Phong)

Le modèle d'illumination complet combine les composantes ambiante, diffuse et spéculaire pour calculer l'intensité finale (I) en un point P :

$$I(P) = I_{ambiant} + I_{diffus} + I_{speculaire}$$

En utilisant les notations précédentes et en introduisant un facteur d'atténuation F_d qui peut dépendre de la distance à la source lumineuse (pour les sources ponctuelles/spots), l'équation pour une seule source lumineuse l devient :

$$I(P) = p_a I_a + F_{d,l} \cdot (p_d \cdot I_l \cdot \max(0, \vec{N} \cdot \vec{L}_l) + p_s \cdot I_l \cdot \max(0, \vec{N} \cdot \vec{H}_l)^n)$$

(Ici, la version Blinn-Phong est utilisée pour le terme spéculaire).

Note: Le diagramme de grille de théières montre l'effet combiné : l'augmentation de p_s (marquée 'ps' ou 'n' sur le slide) ajoute ou renforce les reflets brillants.

Note: Les images des théières de la page 7 illustrent le résultat visuel de l'application du modèle complet avec différents paramètres.

4.5.7 Modèle coloré

Pour obtenir des couleurs, les calculs sont effectués séparément pour chaque composante de couleur (Rouge, Vert, Bleu - RVB) :

- L'intensité de la lumière ambiante I_a et de chaque source I_l ont des composantes (I_{ar}, I_{ag}, I_{ab}) et (I_{lr}, I_{lg}, I_{lb}).
- Les coefficients de réflexion p_a, p_d, p_s sont également définis par composante : (p_{ar}, p_{ag}, p_{ab}), (p_{dr}, p_{dg}, p_{db}), (p_{sr}, p_{sg}, p_{sb}). Ces coefficients représentent la couleur intrinsèque du matériau.

- L'équation complète est appliquée pour chaque canal R, V, B :

$$I_r = p_{ar} I_{ar} + \sum_l F_{d,l}(p_{dr} I_{lr}(\vec{N} \cdot \vec{L}_l) + p_{sr} I_{lr}(\vec{N} \cdot \vec{H}_l)^n)$$
$$I_g = p_{ag} I_{ag} + \sum_l F_{d,l}(p_{dg} I_{lg}(\vec{N} \cdot \vec{L}_l) + p_{sg} I_{lg}(\vec{N} \cdot \vec{H}_l)^n)$$
$$I_b = p_{ab} I_{ab} + \sum_l F_{d,l}(p_{db} I_{lb}(\vec{N} \cdot \vec{L}_l) + p_{sb} I_{lb}(\vec{N} \cdot \vec{H}_l)^n)$$

  (Les termes $\max(0, \cdot)$ sont omis pour la lisibilité).

Si plusieurs sources lumineuses sont présentes, leurs contributions (diffuse et spéculaire) sont sommées. La composante ambiante est généralement ajoutée une seule fois.

4.5.8 Transparence

Pour gérer la transparence, un paramètre t (souvent appelé alpha, α) est introduit ($0 \leq t \leq 1$, où $t = 1$ est opaque et $t = 0$ est transparent). La couleur finale I d'un fragment semi-transparent est calculée en combinant sa propre couleur calculée $I(P)$ avec la couleur $I(\text{derriere P})$ de ce qui se trouve derrière lui :

$$I = t \cdot I(P) + (1 - t) \cdot I(\text{derriere P})$$

4.5.9 Halo

Le halo simule un effet de lueur autour des objets lumineux ou réfléchissants. La couleur et l'intensité du halo dépendent souvent de l'épaisseur de l'objet traversé ou de l'intensité de la source.

4.6 Modèles d'Ombrage (Shading)

L'ombrage (*shading*) est la technique utilisée pour appliquer les modèles d'éclairement aux surfaces des objets (souvent représentés par des maillages de polygones, typiquement des triangles) afin de déterminer la couleur de chaque pixel.

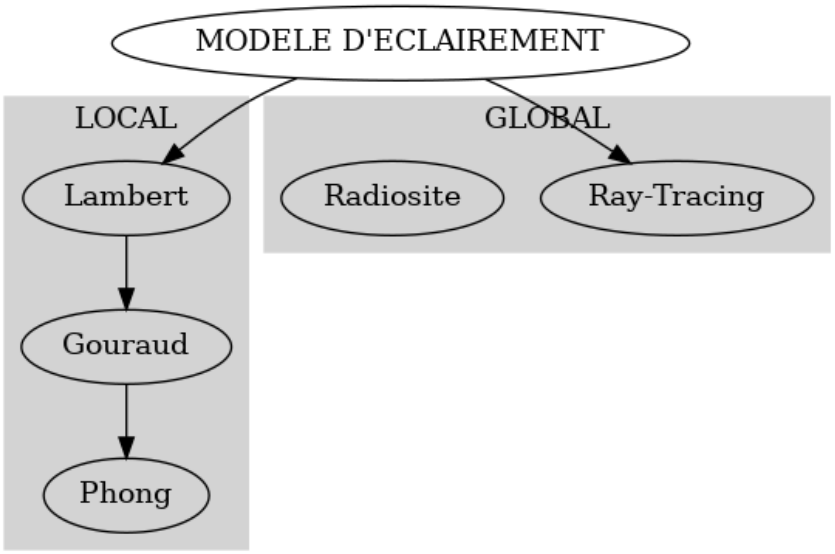

Figure 4.6: Classification des modèles d'ombrage.

- **Modèles locaux :** La luminance à la surface d'un objet est calculée en considérant l'objet comme isolé. On utilise uniquement les paramètres de l'objet lui-même et les paramètres des sources de lumière directes. (Ex: Lambert, Gouraud, Phong).
- **Modèles globaux :** La luminance est calculée en tenant compte des interactions avec tous les objets de la scène (ombres portées, réflexions indirectes, etc.). (Ex: Ray Tracing, Radiosité).

Nous nous concentrons ici sur les modèles locaux.

4.6.1 Ombrage de Lambert (Plat / Constant)

- La méthode d'ombrage la plus simple pour les facettes polygonales.
- Une seule valeur d'illumination est calculée pour l'ensemble de la facette et appliquée uniformément à tous ses pixels.
- Le calcul est typiquement fait au centre de la facette, en utilisant la normale géométrique de la facette elle-même (qui est constante sur toute la facette).
- Formule (pour la composante diffuse) : $I_{facette} = p_d \cdot I_l \cdot \max(0, \vec{N}_{facette} \cdot \vec{L})$
- **Problème :** Crée des discontinuités d'intensité visibles aux arêtes entre les facettes, donnant un aspect facetté. L'œil humain a tendance à exagérer ces changements brusques d'intensité (effet de *Mach Banding*).
- **Solution :** Utiliser une interpolation pour lisser les transitions (voir Gouraud et Phong).

Note: L'illustration des sphères montre clairement l'aspect facetté avec 16, 32 et 64 facettes. Note: Le diagramme d'intensité illustre la discontinuité perçue (marches d'escalier).

4.6.2 Ombrage de Gouraud [1971]

- Développé par Henri Gouraud pour éliminer les discontinuités de l'ombrage plat.
- Principe : Interpoler les valeurs d'*intensité* (couleur) aux sommets de la facette.
- Étapes :
  1. **Calculer les normales aux sommets :** Pour chaque sommet, calculer une normale qui représente la moyenne de l'orientation de la surface autour de ce sommet. Cela se fait généralement en moyennant les normales des facettes qui partagent ce sommet. $N_s = \frac{\sum N_i}{\|\sum N_i\|}$, où N_i sont les normales des faces incidentes au sommet s.
  2. **Calculer l'intensité aux sommets :** Appliquer le modèle d'éclairement (par exemple, Ambiant + Diffus + Spéculaire) à chaque sommet en utilisant sa normale calculée (N_s) pour obtenir une intensité (I_{s1}, I_{s2}, I_{s3} pour un triangle). $I_s = p_a I_a + p_d I_l(\vec{N}_s \cdot \vec{L}) + p_s I_l(\vec{N}_s \cdot \vec{H})^n$.
  3. **Interpoler les intensités à l'intérieur de la facette :** Pour chaque pixel à l'intérieur de la facette, interpoler linéairement (ou bilinéairement) les intensités calculées aux sommets. Ceci est souvent fait par scan-line :
     - Interpoler les intensités le long des arêtes gauche et droite de la facette pour la ligne de balayage actuelle (y) pour obtenir I_{gauche} et I_{droite}.

       $$I_{gauche} = I_{s1}\frac{y - y_2}{y_1 - y_2} + I_{s2}\frac{y - y_1}{y_2 - y_1}$$

       (Exemple pour l'arête S1-S2, adaptation pour les autres arêtes).

– Interpoler horizontalement entre I_{gauche} et I_{droite} pour la coordonnée x du pixel.

$$I_{pixel}(x,y) = I_{gauche}\frac{x - x_{droite}}{x_{gauche} - x_{droite}} + I_{droite}\frac{x - x_{gauche}}{x_{droite} - x_{gauche}}$$

(Simplifié, utilise souvent les coordonnées barycentriques).

- **Caractéristiques :**
  - Produit un aspect lisse et continu.
  - Efficace en termes de calcul (calcul du modèle d'éclairement uniquement aux sommets).
  - Facile à implémenter avec des algorithmes de remplissage de polygones comme le Z-buffer (scan-line).
  - **Inconvénients :** N'est qu'une approximation. Peut manquer ou déformer les reflets spéculaires s'ils se produisent au milieu d'une facette (car l'interpolation se fait sur les intensités finales). Peut produire des artefacts visuels (comme des bandes de Mach si la tessellation n'est pas assez fine).

Note: Le pseudo-code pour l'interpolation de Gouraud est omis mais les étapes sont décrites ci-dessus.

4.6.3 Ombrage de Phong [1973]

- Développé pour améliorer la gestion des reflets spéculaires par rapport à Gouraud.
- Principe : Interpoler les vecteurs *normaux* aux sommets, puis appliquer le modèle d'éclairement complet à chaque pixel en utilisant la normale interpolée.
- Étapes :
  1. **Calculer les normales aux sommets :** Identique à Gouraud (N_{s1}, N_{s2}, N_{s3}).
  2. **Interpoler les normales à l'intérieur de la facette :** Pour chaque pixel, interpoler linéairement (ou bilinéairement) les vecteurs normaux des sommets pour obtenir une normale interpolée $\vec{N}_{pixel}$. L'interpolation se fait de la même manière que pour les intensités dans Gouraud (par scan-line ou coordonnées barycentriques).

     $$\vec{N}_{pixel}(x,y) = Interpolation(\vec{N}_{s1}, \vec{N}_{s2}, \vec{N}_{s3})$$

     (Important: le vecteur normal interpolé doit être *renormalisé* après interpolation).
  3. **Calculer l'intensité au pixel :** Appliquer le modèle d'éclairement complet (Ambiant + Diffus + Spéculaire) pour le pixel courant en utilisant la normale interpolée et renormalisée $\vec{N}_{pixel}$.

     $$I_{pixel} = p_a I_a + p_d I_l(\vec{N}_{pixel} \cdot \vec{L}) + p_s I_l(\vec{N}_{pixel} \cdot \vec{H})^n$$

- **Caractéristiques :**
  - Produit des reflets spéculaires plus précis et réalistes que Gouraud, même s'ils apparaissent au milieu des facettes.
  - Est généralement considéré comme donnant un meilleur rendu visuel que Gouraud, même pour des modèles sans spécularité forte.
  - **Inconvénients :** Nettement plus coûteux en calcul que Gouraud, car le modèle d'éclairement complet (incluant potentiellement des racines carrées pour la normalisation et des exponentiations pour le spéculaire) doit être évalué pour chaque pixel, et non juste aux sommets.

Note: Les comparaisons visuelles (sphères, visages) montrent que Lambert est facetté, Gouraud est lisse mais avec des reflets spéculaires diffus/manquants, et Phong est lisse avec des reflets nets et bien placés.

4.6.4 Ajout des Couleurs et Plusieurs Sources

Comme pour les modèles d'éclairement, l'ombrage est appliqué par composante de couleur (RVB). Pour gérer plusieurs sources lumineuses, les contributions diffuses et spéculaires de chaque source sont additionnées à la composante ambiante unique.

$$I_{pixel} = \text{Ambiant} + \sum_{l \in \text{lumières}} (\text{Diffus}_l + \text{Spéculaire}_l)$$

où Ambiant, Diffus_l, Spéculaire_l sont calculés en utilisant les normales appropriées (facette pour Lambert, sommet pour Gouraud, pixel interpolé pour Phong) et les propriétés de la lumière l.

4.7 Interpolation Linéaire et Coordonnées Barycentriques

L'interpolation est fondamentale pour les ombrages de Gouraud et Phong.

4.7.1 Interpolation Linéaire en 1D

Pour interpoler une valeur f entre deux points x_i et x_j où les valeurs sont f_i et f_j, on utilise un paramètre $t \in [0, 1]$:

$$f(t) = f_i + t(f_j - f_i) = (1 - t)f_i + tf_j$$

où $t = (x - x_i)/(x_j - x_i)$. Cela peut être vu comme une combinaison linéaire de deux fonctions de base, $(1 - t)$ et t.

4.7.2 Interpolation Linéaire en 2D (pour les triangles)

Pour interpoler une valeur (intensité, normale, coordonnée de texture, etc.) f à l'intérieur d'un triangle défini par les sommets p_i, p_j, p_k avec les valeurs f_i, f_j, f_k, on utilise les **coordonnées barycentriques**.

Un point p à l'intérieur du triangle peut s'écrire comme une combinaison affine des sommets : $p = \alpha p_i + \beta p_j + \gamma p_k$, avec $\alpha, \beta, \gamma \geq 0$ et $\alpha + \beta + \gamma = 1$. Les coefficients (α, β, γ) sont les coordonnées barycentriques de p. Ils peuvent être calculés comme des rapports d'aires :

$$\alpha = \frac{\text{area}(p, p_j, p_k)}{\text{area}(p_i, p_j, p_k)}$$
$$\beta = \frac{\text{area}(p, p_i, p_k)}{\text{area}(p_i, p_j, p_k)}$$
$$\gamma = \frac{\text{area}(p, p_i, p_j)}{\text{area}(p_i, p_j, p_k)} \quad (= 1 - \alpha - \beta)$$

La valeur interpolée $f(p)$ est alors :

$$f(p) = \alpha f_i + \beta f_j + \gamma f_k$$

Ces coordonnées sont calculées lors de la rastérisation (remplissage) du triangle et permettent d'interpoler n'importe quel attribut défini aux sommets.

4.8 Interpolation en Projection Perspective

4.8.1 Le Problème

L'interpolation barycentrique directe des attributs dans l'espace écran 2D après une projection perspective ne correspond pas à une interpolation linéaire dans l'espace 3D d'origine. Cela est dû à la division par la profondeur (ou coordonnée w) lors de la projection. Si on interpole linéairement en 2D des attributs (comme les coordonnées de texture), on obtient des déformations incorrectes (non linéaires en 3D).

4.8.2 Interpolation avec Correction de Perspective

Pour obtenir une interpolation correcte en perspective, il faut interpoler les attributs d'une manière qui tienne compte de la profondeur.

- Soit q l'attribut à interpoler (ex: coordonnée de texture u ou v, couleur R, G ou B) et z la profondeur (distance à la caméra) au sommet.
- Au lieu d'interpoler q directement, on calcule $P = q/z$ et $Z = 1/z$ pour chaque sommet.
- On interpole P et Z linéairement dans l'espace écran 2D en utilisant les coordonnées barycentriques (α, β, γ) :

$$P_{interp} = \alpha P_i + \beta P_j + \gamma P_k$$
$$Z_{interp} = \alpha Z_i + \beta Z_j + \gamma Z_k$$

- La valeur finale de l'attribut interpolé correctement en perspective pour le pixel est obtenue en divisant P_{interp} par Z_{interp} :

$$q_{pixel} = \frac{P_{interp}}{Z_{interp}}$$

Cette méthode assure que l'interpolation est linéaire dans l'espace 3D. Elle est essentielle pour obtenir des textures et des couleurs correctes sur des surfaces inclinées par rapport à l'observateur.

Note: L'illustration du quadrilatère divisé montre la discontinuité qui apparaît si la correction de perspective n'est pas utilisée lors de l'interpolation (par exemple, de coordonnées de texture).

4.9 Textures (Plaquage et Mappage)

Le **Texture Mapping** (plaquage de texture) consiste à appliquer une image (la texture) sur la surface d'un objet 3D pour ajouter des détails visuels sans augmenter la complexité géométrique du modèle.

4.9.1 Motivation

Calculer l'éclairage par sommet et interpoler (Gouraud/Phong) peut nécessiter une géométrie très fine pour représenter des détails fins. Le plaquage de texture est une alternative plus efficace : on calcule un éclairage plus simple sur une géométrie moins complexe, puis on module le résultat avec une image de texture détaillée.

4.9.2 Types de Textures

Il existe de nombreux types de textures utilisés pour différents effets :

- **Couleurs (Colour map / Diffuse map / Albedo) :** Définit la couleur de base de la surface.
- **Transparence (Alpha map) :** Contrôle l'opacité de la surface (souvent dans le canal alpha de la texture de couleur).
- **Relief (Bump maps, Normal maps, Displacement maps) :** Simulent des détails de relief en modifiant la normale perçue (Normal map) ou en déplaçant réellement les sommets (Displacement map), sans ajouter de polygones.
- **Spéculaires (Specular map, Gloss map, Roughness map, Metallic map) :** Contrôlent comment la surface réfléchit la lumière spéculaire (intensité, couleur, netteté du reflet, caractère métallique).
- **Environnement (Environment map, Cube map) :** Utilisées pour simuler des réflexions de l'environnement sur des surfaces réfléchissantes.
- **Lumière (Light map) :** Stocke un éclairage précalculé (souvent l'illumination globale indirecte) qui est ensuite ajouté à l'éclairage dynamique.
- **Autres :** Occlusion ambiante, émissivité, épaisseur (pour le subsurface scattering), etc.

4.9.3 Attributs des Textures et Combinaison

Les textures peuvent être combinées pour créer des effets complexes :

- **Diffuse + Spéculaire :** Une texture de couleur (diffuse) et une texture contrôlant la brillance (spéculaire).
- **Diffuse + Transparence :** Utilisation du canal alpha pour rendre certaines parties invisibles ou semi-transparentes.
- **Illumination précalculée (Light map) :** Multipliée ou ajoutée à la couleur de base pour simuler un éclairage statique complexe.
- **Simulation d'environnement :** Une texture cubique (Cube map) représentant l'environnement autour de l'objet est utilisée pour calculer les réflexions.
- **Normal Mapping :** Une texture spéciale (Normal map) stocke des vecteurs normaux. Lors du calcul de l'éclairage, la normale lue dans cette texture est utilisée à la place de la normale géométrique, créant une illusion de relief détaillé sur une surface plane.
- **Displacement Mapping :** Similaire au Normal Mapping, mais les valeurs de la texture sont utilisées pour déplacer réellement les sommets de la géométrie le long de leur normale, créant un relief réel (plus coûteux).

4.9.4 Fonction de Plaquage (Mapping)

Le processus consiste à établir une correspondance entre les points de la surface 3D de l'objet $P(x, y, z)$ et les points (coordonnées) (s, t) ou (u, v) dans l'image de texture 2D $I(s, t)$. Cette correspondance est la fonction de plaquage $f : P(x, y, z) \to (s, t)$.

Coordonnées de Texture (UV Mapping) : La méthode la plus courante consiste à assigner des coordonnées de texture (u, v) à chaque sommet du modèle 3D. Ces coordonnées (u, v) se situent généralement dans l'intervalle $[0, 1]$ et indiquent quelle partie de l'image de texture correspond à ce sommet. Lors de la rastérisation, ces coordonnées (u, v) sont interpolées sur la surface du polygone (en utilisant l'interpolation avec correction de perspective !) pour chaque pixel. La coordonnée (u, v) interpolée est ensuite utilisée pour lire la couleur (ou autre valeur) dans l'image de texture.

Méthodes de projection pour générer les UV : Pour les objets simples, les coordonnées UV peuvent être générées par des projections mathématiques :

- **Planaire :** Projection sur un plan. $f(x, y, z) = (k_x x + c_x, k_y y + c_y)$. Convient aux objets plats.
- **Cylindrique :** Enroulement autour d'un cylindre. Les coordonnées sont basées sur l'angle θ autour de l'axe du cylindre et la hauteur y. $f(\theta, y) = (\theta/(2\pi), y/H)$, où $\theta = \text{atan2}(x, z)$.
- **Sphérique :** Basé sur les angles de longitude (θ) et latitude (ϕ). $f(\theta, \phi) = (\theta/(2\pi), \phi/\pi)$, où $\theta = \text{atan2}(x, z)$, $\phi = \text{asin}(y/R)$.

Pour les modèles complexes, les coordonnées UV sont souvent créées manuellement par un artiste 3D dans un processus appelé "UV unwrapping" (dépliage UV).

4.9.5 Interprétation des Coordonnées Hors [0, 1]

Que faire si les coordonnées (u, v) interpolées tombent en dehors de l'intervalle $[0, 1]$? Plusieurs modes d'adressage de texture existent :

- **GL_CLAMP_TO_EDGE :** La coordonnée est limitée à l'intervalle [0, 1]. La couleur du bord de la texture est répétée.
- **GL_REPEAT :** La partie entière de la coordonnée est ignorée, seule la partie fractionnaire est utilisée. La texture se répète.
- **GL_MIRRORED_REPEAT :** Similaire à REPEAT, mais la texture est répétée en miroir à chaque fois.

4.9.6 Problèmes et Limites

- **Distorsion :** Selon la méthode de plaquage UV, la texture peut apparaître étirée ou compressée sur certaines parties du modèle. Un bon dépliage UV vise à minimiser ces distorsions.
- **Crénelage (Aliasing) :** Apparaît lorsque la résolution de la texture ne correspond pas bien à la taille à laquelle elle est affichée à l'écran.

4.9.7 Magnification vs. Minification

- **Magnification (Agrandissement) :** La caméra est proche de l'objet texturé. Un pixel à l'écran correspond à une petite zone (moins d'un texel) dans la texture.
  - *Problème :* Peut rendre la texture pixellisée si on prend simplement le texel le plus proche (*Nearest neighbor filtering*).
  - *Solution :* Interpolation bilinéaire (moyenne pondérée des 4 texels voisins) pour un résultat plus lisse.
- **Minification (Réduction) :** La caméra est loin de l'objet texturé. Un pixel à l'écran correspond à une grande zone (plusieurs texels) dans la texture.
  - *Problème :* Si on échantillonne juste un texel au centre de la zone couverte par le pixel, on manque beaucoup d'informations, ce qui conduit à des artefacts scintillants (aliasing) et des motifs moirés.
  - *Solution :* Il faut calculer une couleur moyenne de tous les texels couverts par le pixel. C'est coûteux à faire dynamiquement. Une solution est le **MIP-Mapping**.

4.9.8 Aliasing et MIP-Mapping

Le **MIP-Mapping** est une technique de préfiltrage pour lutter contre l'aliasing de minification.

- **Idée :** Créer une pyramide d'images pré-filtrées de la texture originale, où chaque niveau est une version réduite de moitié et moyennée du niveau précédent. Le niveau 0 est la texture originale (X_0, Y_0). Le niveau 1 est $(X_0/2, Y_0/2)$, le niveau 2 est $(X_0/4, Y_0/4)$, etc., jusqu'à une texture 1x1 pixel. (MIP vient de *multum in parvo*, "beaucoup de choses dans un petit espace").
- **Utilisation :** En fonction de la distance de l'objet ou de la taille projetée de la texture à l'écran, le système graphique choisit le niveau de MIP-map le plus approprié (celui où un pixel écran correspond à environ un texel de ce niveau de MIP-map).
- **Filtrage inter-niveaux :** On peut encore améliorer la qualité en interpolant linéairement entre les deux niveaux de MIP-map les plus proches (*filtrage trilinéaire*).
- **Modes OpenGL :**
  - 'GL$_N EAREST$'/'$GL_L INEAR$' : $Pas de MIP-Mapping (bon pour magnification)$.'$GL_N EAREST_M IPMAP_N E$
    $Choisit le meilleur niveau MIP, prend le texel le plus proche$.
  - 'GL$_L INEAR_M IPMAP_N EAREST$' : $Choisit le meilleur niveau MIP, interpole bilinéairement dans ce niveau$.'$GL_N EARES$
    $Interpole linéairement entre les texels les plus proches de deux niveaux MIP$.
  - 'GL$_L INEAR_M IPMAP_L INEAR$' : $Interpole bilinéairement dans deux niveaux MIP et interpole linéairement entre les$

4.10 Ray Tracing

Le Ray Tracing (lancer de rayons) est une technique d'illumination globale alternative au pipeline de rastérisation standard.

4.10.1 Introduction

- Approche basée sur le pixel (*image-order rendering*) et non sur l'objet (*object-order rendering* comme la rastérisation).
- Principe "inverse-mapping" : Au lieu de projeter les objets vers l'écran, on lance des rayons depuis la caméra (l'œil) à travers chaque pixel de l'écran et on détermine ce que ce rayon rencontre dans la scène.
- Étapes de base :
  1. Pour chaque pixel de l'écran, construire un rayon partant de la caméra et passant par ce pixel.
  2. Trouver la première intersection de ce rayon avec un objet de la scène.
  3. Si une intersection est trouvée, déterminer la couleur en ce point d'intersection.

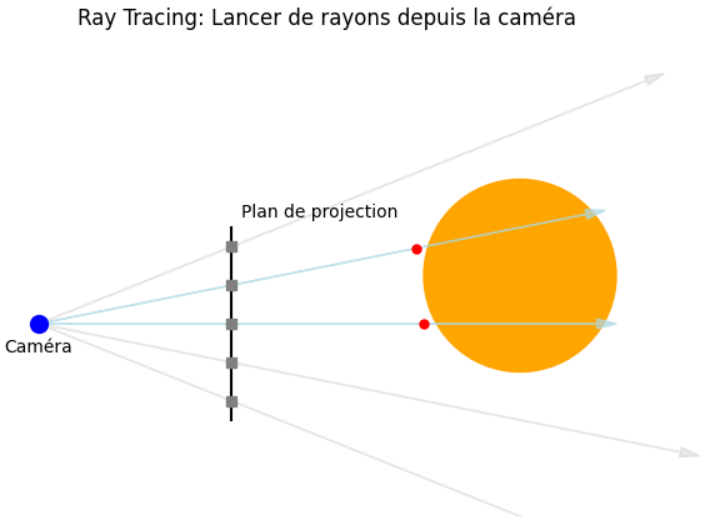

Figure 4.7: Principe du lancer de rayons.

4.10.2 Calcul de la Couleur au Point d'Intersection

Pour déterminer la couleur au point d'intersection, on applique un modèle d'éclairement (similaire à Phong), mais on peut aussi simuler des phénomènes globaux :

- **Ombres :** Depuis le point d'intersection, tracer un rayon ("rayon d'ombre") vers chaque source lumineuse. Si ce rayon rencontre un autre objet avant d'atteindre la source, le point est dans l'ombre de cette source (sa contribution directe est nulle).
- **Réflexion :** Si la surface est réfléchissante, lancer un nouveau rayon dans la direction de la réflexion (calculée comme dans Phong : $\vec{R} = 2(\vec{N} \cdot \vec{L})\vec{N} - \vec{L}$, où $\vec{L}$ est ici la direction opposée au rayon incident). La couleur obtenue par ce rayon réfléchi est ajoutée à la couleur du point d'intersection, pondérée par le coefficient de réflexion de la surface.
- **Réfraction :** Si la surface est transparente ou translucide, lancer un nouveau rayon dans la direction de la réfraction (calculée par la loi de Snell-Descartes). La couleur obtenue par ce rayon réfracté est ajoutée, pondérée par le coefficient de transparence/réfraction.

4.10.3 Ray Tracing Récursif

Les rayons de réflexion et de réfraction peuvent eux-mêmes intersecter d'autres objets, générant à leur tour de nouveaux rayons d'ombre, de réflexion et de réfraction. Ce processus récursif permet de simuler des réflexions multiples, des réfractions complexes et des ombres douces (si on utilise des sources étendues). La récursion s'arrête après un certain nombre de rebonds ou lorsque l'intensité du rayon devient négligeable.

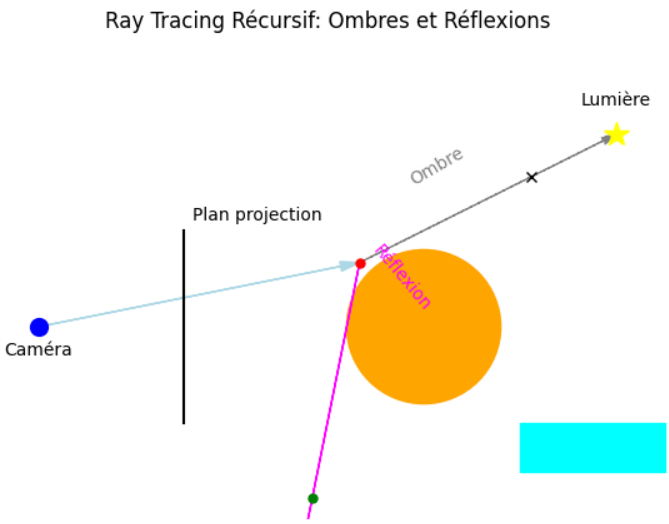

Figure 4.8: Illustration du lancer de rayons récursif avec ombres et réflexions.

Le Ray Tracing est capable de produire des images très réalistes mais est traditionnellement beaucoup plus coûteux en calcul que la rastérisation, bien que les accélérations matérielles récentes (RTX) le rendent de plus en plus viable pour le temps réel.

Chapter 5

Part5

5.1 Rastérisation

La rastérisation, aussi appelée pixélisation, est un processus fondamental en infographie qui consiste à convertir des descriptions géométriques de primitives (comme les points, les lignes et les triangles) en une représentation basée sur des pixels sur un écran ou dans une image bitmap. C'est une étape clé du pipeline graphique qui permet d'afficher des scènes 3D sur des écrans 2D.

5.1.1 Pipeline Graphique

Le pipeline graphique est une séquence d'étapes qui transforment une description de scène 3D en une image 2D affichable. La rastérisation est l'une des dernières étapes majeures de ce pipeline.

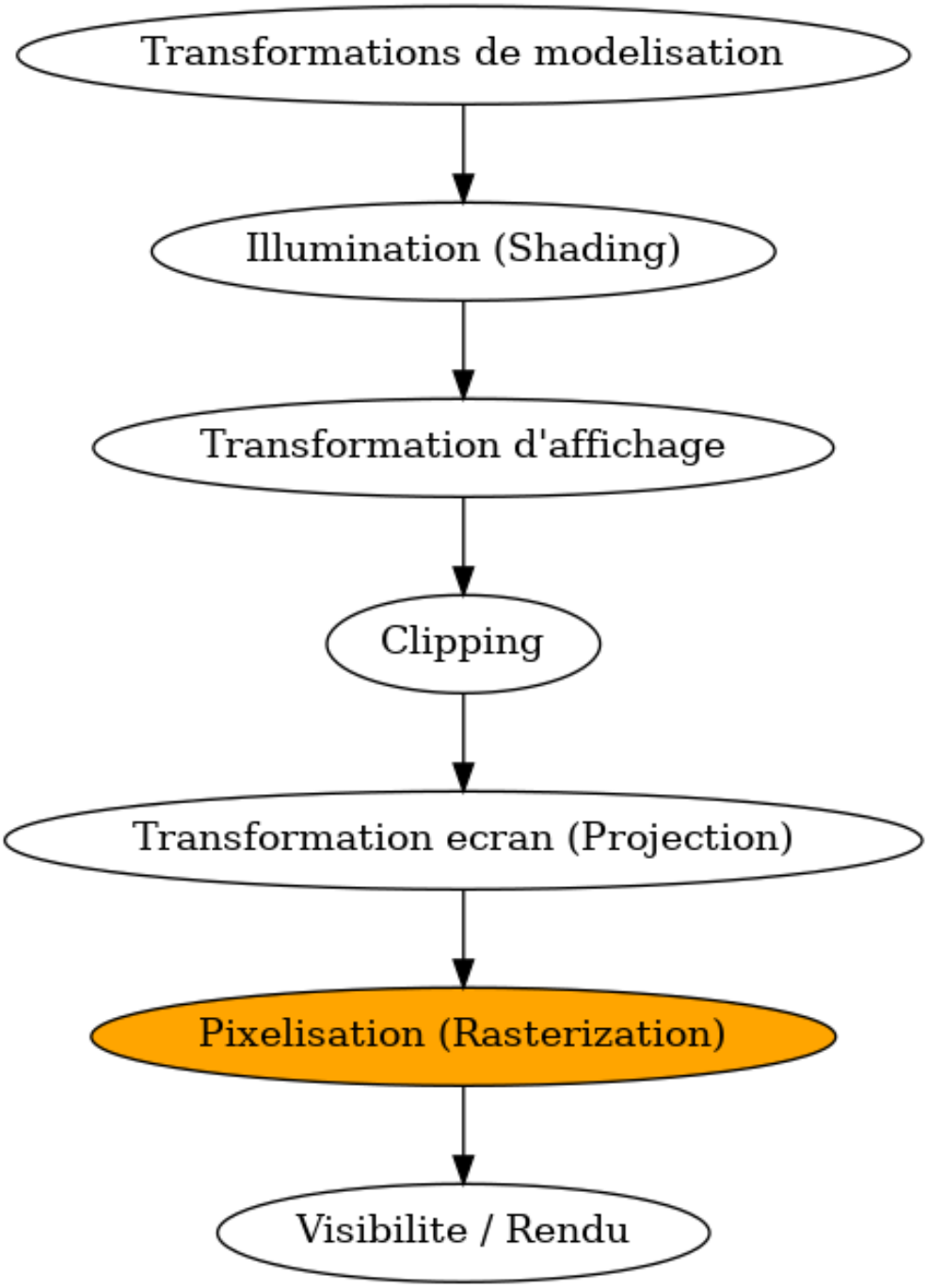

Figure 5.1: Étapes simplifiées du pipeline graphique mettant en évidence la Rastérisation.

La rastérisation implique deux actions principales :

- **Découpe des primitives 2D en pixels :** Déterminer quels pixels de la grille de l'écran sont couverts par chaque primitive géométrique (après projection).
- **Interpolation des valeurs connues aux sommets :** Calculer les attributs (comme la couleur, la profondeur, les coordonnées de texture) pour chaque pixel couvert (fragment) en interpolant les valeurs définies aux sommets de la primitive.

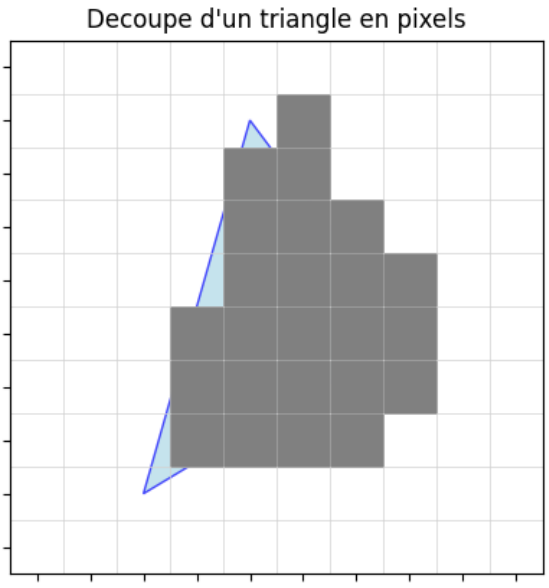

Figure 5.2: Illustration de la découpe d'une primitive (triangle) en pixels sur une grille.

5.1.2 Pipeline de Rastérisation

Le pipeline de rastérisation moderne est conçu pour la génération d'images en temps réel.

- **Input:** Primitives 3D, essentiellement des triangles, potentiellement avec des attributs supplémentaires (couleur, normales, coordonnées de texture). Par exemple, un objet peut être défini par une liste de sommets (LS) et une liste de faces (F), chaque face référençant des sommets:
  - LS = {P0, P1, P2, P3}
  - F1 = (LS[0], LS[1], LS[2])
  - F2 = (LS[0], LS[2], LS[3])
  - ...
  - Obj = {F1, F2, F3, F4}
- **Output:** Une image bitmap (un tableau 2D de pixels), potentiellement avec des informations supplémentaires par pixel (profondeur pour le test de visibilité, alpha pour la transparence).
- **Objectif:** Comprendre les étapes intermédiaires qui mènent de la géométrie aux pixels.
- **Exécution:** En pratique, ces étapes sont massivement parallélisées et exécutées par le GPU (Graphics Processing Unit).

5.1.3 Pourquoi les triangles?

Le pipeline de rastérisation est fortement optimisé pour le traitement des triangles. Toutes les primitives géométriques, y compris les points et les lignes, sont généralement converties en triangles avant ou pendant la rastérisation.

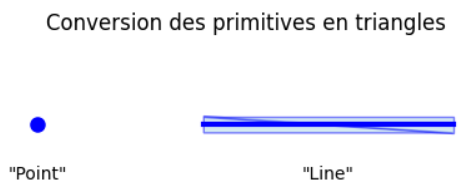

Figure 5.3: Conversion conceptuelle de points et lignes en triangles pour la rastérisation.

Pourquoi cette focalisation sur les triangles?

- **Approximation universelle:** N'importe quelle forme 3D complexe peut être approximée par un maillage de triangles.
- **Planaire:** Un triangle est toujours planaire, ce qui garantit un vecteur normal bien défini (utile pour l'éclairage).
- **Interpolation facile:** Les attributs (couleur, etc.) peuvent être facilement et efficacement interpolés sur la surface du triangle en utilisant les coordonnées barycentriques.

5.1.4 Pixels sur l'écran

L'étape finale de la rastérisation consiste à déterminer la couleur de chaque pixel sur l'écran.

- Chaque élément d'image (pixel) est affiché comme un petit carré de lumière avec la couleur appropriée (approximativement).
- Pixel signifie "picture element".

5.1.5 Rastérisation en bref

La question centrale de la rastérisation est: **Quels pixels le triangle (projeté) chevauche-t-il?**

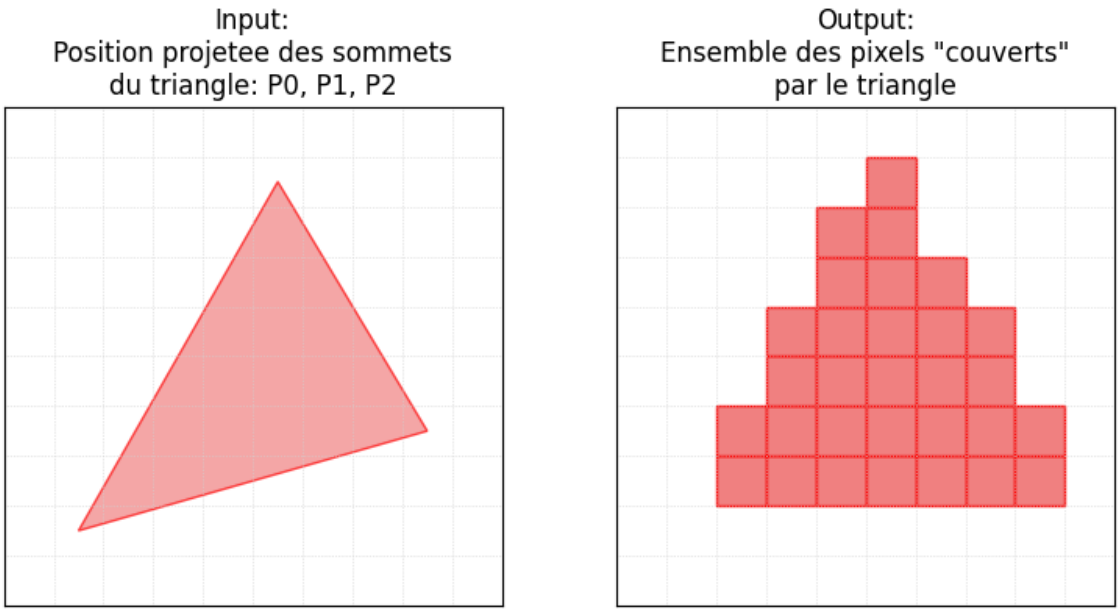

Figure 5.4: Entrée (sommets du triangle projeté) et sortie (pixels couverts) de la rastérisation.

Une autre question importante gérée pendant ou après la rastérisation est la visibilité : **Quel est le triangle le plus proche de la caméra dans chaque pixel ?** Ceci est généralement résolu à l'aide d'un Z-buffer (tampon de profondeur).

Les principaux aspects abordés dans le contexte de la rastérisation incluent :

- **Traçage :** Algorithmes pour dessiner des lignes et des courbes (segments de droites, cercles).
- **Anticrénelage (Antialiasing) :** Techniques pour réduire les artefacts visuels (jaggies) dus à la discrétisation.
- **Remplissage des primitives projetées :** Algorithmes pour remplir l'intérieur des polygones (triangles).

5.1.6 Traçage : Segments de Droites

Le traçage de segments de droite est un algorithme de base essentiel pour de nombreux traitements en infographie, tels que le dessin en fil de fer, le remplissage de polygones, et l'élimination des parties cachées. L'objectif est de déterminer quels pixels doivent être allumés pour représenter au mieux un segment de droite continu entre deux points (x_1, y_1) et (x_2, y_2) sur une grille de pixels discrète.

Il y a trois impératifs pour un bon algorithme de traçage de segment discret :

1. Tout point du segment discret est traversé par le segment continu.
2. Le segment discret doit être connexe. Tout point du segment discret touche au moins un autre point, soit par l'un de ses côtés (4-connexité), soit par un de ses sommets (8-connexité). La 4-connexité est souvent préférée pour éviter des lignes trop épaisses en diagonale.
3. On trace le moins de points possibles (idéalement, un seul pixel par colonne ou par ligne de la grille, selon la pente).

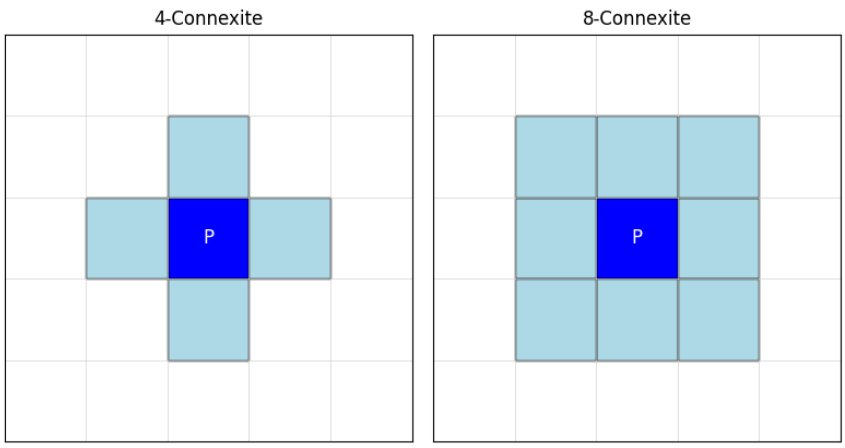

Figure 5.5: Voisins d'un pixel (P) en 4-connexité (côtés) et 8-connexité (côtés et sommets).

Tracé de segments par l'équation cartésienne

Une approche simple consiste à utiliser l'équation cartésienne de la droite : $y = ax + b$. Soit un segment entre (x_1, y_1) et (x_2, y_2). La pente a et l'ordonnée à l'origine b sont :

$$a = \frac{y_2 - y_1}{x_2 - x_1}$$
$$b = y_1 - ax_1$$

Pour simplifier, supposons $x_2 > x_1$, $y_2 \geq y_1$ et que la pente est inférieure ou égale à 1, c'est-à-dire $(x_2 - x_1) \geq (y_2 - y_1)$. Dans ce cas, pour chaque valeur entière de x entre x_1 et x_2, on calcule la valeur y correspondante et on arrondit à l'entier le plus proche pour déterminer le pixel à allumer.

Incrémentation suivant l'axe x (pente ≤ 1) Si la valeur absolue de la pente $|a| \leq 1$, on incrémente x de x_1 à x_2 et on calcule $y = \text{round}(ax + b)$ pour chaque x.

Listing 5.1: Algorithme simple basé sur l'équation cartésienne (incrémentation en x)

```
DroiteSimple (int x1, int x2, int y1, int y2)
{
  a = (y2 - y1) / (x2 - x1);
  b = y1 - a * x1;
  x = x1;
  while (x <= x2) // Inclure x2
  {
    AfficherPixel(x, round(a * x + b)); // Arrondi nécessaire
    x = x + 1;
  }
}
```

Figure 5.6: Exemple de tracé par incrémentation suivant l'axe x (pour une pente ≤ 1).

Incrémentation suivant l'axe y (pente > 1) Si la valeur absolue de la pente $|a| > 1$, on doit incrémenter y de y_1 à y_2 et calculer $x = \text{round}((y - b)/a)$ pour chaque y.

Listing 5.2: Algorithme simple basé sur l'équation cartésienne (incrémentation en y)

```
DroiteSimpleY (int x1, int x2, int y1, int y2)
{
  a = (y2 - y1) / (x2 - x1);
  b = y1 - a * x1;
  y = y1;
  while (y <= y2) // Inclure y2
  {
    AfficherPixel(round((y - b) / a), y); // Arrondi nécessaire
    y = y + 1;
  }
}
```

Figure 5.7: Exemple de tracé par incrémentation suivant l'axe y (pour une pente > 1). Les pixels sont ceux indiqués sur la diapositive.

Il faut donc "switcher" entre ces deux versions (incrémentation en x ou en y) en fonction de la pente de la droite. Les autres cas (pentes négatives, $x_1 > x_2$, etc.) peuvent être traités par symétrie.

Caractéristiques

- **Simplicité algorithmique :** L'idée de base est facile à comprendre.
- **Lenteur :** Cette méthode nécessite des calculs en virgule flottante (division pour a, multiplication ax, addition $ax + b$), ainsi qu'une opération d'arrondi ('round' ou 'cast' implicite après ajout de 0.5), ce qui est coûteux en termes de performance par rapport aux opérations sur entiers.

Algorithmes de Bresenham

L'algorithme de Bresenham (développé par Jack Bresenham en 1962) est une méthode beaucoup plus efficace pour tracer des lignes car il utilise uniquement des opérations sur entiers (additions, soustractions, comparaisons et décalages binaires implicites par multiplication par 2).

L'idée clé est de maintenir une variable d'erreur e qui représente la distance (ou une valeur proportionnelle à la distance) entre la position y du pixel courant et la position y exacte sur la ligne idéale, pour le x courant. À chaque étape (incrémentation de x), on met à jour e. Si e dépasse un certain seuil (0.5 pour l'erreur

normalisée, ou dx pour l'erreur entière), cela signifie que la ligne est passée plus près du pixel supérieur $(x_k + 1, y_k + 1)$ que du pixel est $(x_k + 1, y_k)$. Dans ce cas, on incrémente y et on ajuste e en conséquence.

Considérons le cas simple $0 \leq a \leq 1$. À l'étape k, on a tracé le pixel (x_k, y_k). Pour $x_{k+1} = x_k + 1$, on doit choisir entre le pixel Est $E = (x_k + 1, y_k)$ et le pixel Nord-Est $NE = (x_k + 1, y_k + 1)$. On choisit le pixel le plus proche de la ligne idéale.

Soit y la coordonnée y exacte sur la ligne pour x_{k+1}. La décision est basée sur la distance verticale entre y et le point milieu $M = (x_k + 1, y_k + 0.5)$.

- Si $y < y_k + 0.5$, on choisit E.
- Si $y \geq y_k + 0.5$, on choisit NE.

Cette condition est équivalente à tester le signe d'un paramètre de décision $d = f(x_k + 1, y_k + 0.5)$, où $f(x, y) = (y_2 - y_1)x - (x_2 - x_1)y + c$ est dérivé de l'équation implicite de la droite. L'astuce de Bresenham est de calculer ce paramètre de décision de manière incrémentale en utilisant uniquement des entiers.

Algorithme de base (avec flottants pour l'erreur) Une première version conceptuelle utilise une erreur e initialisée à 0. À chaque pas en x, on ajoute la pente $a = dy/dx$ à e. Si e dépasse 0.5, on incrémente y et on soustrait 1 à e (pour ramener l'erreur par rapport au nouveau y).

Listing 5.3: Algorithme de Bresenham (conceptuel, avec erreur flottante)

```
DroiteBresenhamFloat (int x1, int x2, int y1, int y2)
{
  dx = x2 - x1;
  dy = y2 - y1;
  a = dy / dx; // Pente (flottant)
  x = x1;
  y = y1;
  e = 0.0;      // Erreur (flottant)
  while (x <= x2)
  {
    AfficherPixel (x, y);
    e = e + a;
    x = x + 1;
    if (e > 0.5)
    {
      y = y + 1;
      e = e - 1.0;
    }
  }
}
```

Optimisation 1 : Éliminer la division initiale On peut éviter la division dy/dx en multipliant toute l'équation de l'erreur par dx. L'erreur e devient $e' = e \times dx$. L'incrément devient $a \times dx = dy$. Le seuil 0.5 devient $0.5 \times dx$. L'ajustement devient $1.0 \times dx = dx$.

Listing 5.4: Bresenham sans division initiale (seuil flottant)

```
DroiteBresenhamNoDiv (int x1, int x2, int y1, int y2)
{
  dx = x2 - x1;
  dy = y2 - y1;
  x = x1;
  y = y1;
  e = 0; // Erreur * dx (entier)
```

```
  while (x <= x2)
  {
    AfficherPixel (x, y);
    e = e + dy;     // Incrémenter l'erreur (entier)
    x = x + 1;
    // Comparaison avec seuil flottant dx/2
    if (e * 2 > dx) // Equivalent à e > dx/2 en entiers si dx > 0
    {
      y = y + 1;
      e = e - dx;   // Ajuster l'erreur (entier)
    }
  }
}
```

Optimisation 2 : Éliminer la comparaison flottante (ou la multiplication par 2) On peut éliminer la comparaison $e > dx/2$ (ou $2e > dx$) en initialisant l'erreur différemment. L'algorithme classique de Bresenham utilise un paramètre de décision $p_k = 2dx \times e_k = 2dy \cdot (x_k - x_1) - 2dx \cdot (y_k - y_1)$. La valeur initiale est $p_0 = 2dy - dx$. La mise à jour se fait comme suit :

- Si $p_k < 0$, le prochain pixel est $E = (x_k + 1, y_k)$, et $p_{k+1} = p_k + 2dy$.
- Si $p_k \geq 0$, le prochain pixel est $NE = (x_k + 1, y_k + 1)$, et $p_{k+1} = p_k + 2dy - 2dx$.

Listing 5.5: Algorithme de Bresenham classique (entiers seulement)

```
DroiteBresenhamInt (int x1, int x2, int y1, int y2)
{
  dx = x2 - x1;
  dy = y2 - y1;
  x = x1;
  y = y1;
  p = 2 * dy - dx; // Parametre de decision initial
  incE = 2 * dy;       // Increment si E choisi
  incNE = 2 * (dy - dx); // Increment si NE choisi

  AfficherPixel (x, y);
  while (x < x2) // Boucle dx fois
  {
    x = x + 1;
    if (p < 0)
    {
      p = p + incE;
      // y ne change pas
    }
    else
    {
      y = y + 1;
      p = p + incNE;
    }
    AfficherPixel (x, y);
  }
}
```

Cet algorithme final n'utilise que des additions, soustractions et comparaisons sur des entiers, le rendant extrêmement rapide. Des adaptations similaires existent pour les autres pentes et directions.

Rapidité due à :

- Utilisation exclusive d'entiers courts (les valeurs de dx, dy, p restent généralement petites, de l'ordre de la résolution de l'écran).
- Opérations arithmétiques simples sur ces entiers (additions, soustractions, comparaisons).

5.2 Anticrénelage (Antialiasing)

Les algorithmes de tracé comme celui de Bresenham produisent des lignes qui peuvent apparaître comme des "escaliers" (appelés "jaggies"), en particulier sur les lignes à faible pente. Ce phénomène est appelé **crénelage** (aliasing). Il est dû à la nature discrète de la grille de pixels qui sous-échantillonne le signal continu de la ligne idéale. D'autres artefacts, comme les motifs de Moiré, peuvent aussi apparaître.

L'**anticrénelage** (antialiasing) regroupe les techniques visant à réduire ces artefacts pour améliorer la qualité visuelle des images. L'idée générale est de simuler une couverture partielle des pixels par la primitive, en utilisant des nuances de couleur ou d'intensité.

Image: Side-by-side comparison showing aliased lines on the left and antialiased lines on the right.

Figure 5.8: Effet du crénelage (gauche) et résultat après anticrénelage (droite).

Les deux principales approches sont :

- Le sur-échantillonnage de l'image.
- Les algorithmes de tracé de segments corrigés.

5.2.1 Antialiasing: Sur-échantillonnage de l'Image

Le principe est de calculer l'image à une résolution plus élevée que la résolution finale désirée (par exemple, 3x3 ou 5x5 sous-pixels pour chaque pixel final), puis de combiner les valeurs des sous-pixels pour obtenir la valeur du pixel final. Cette combinaison se fait par **filtrage**.

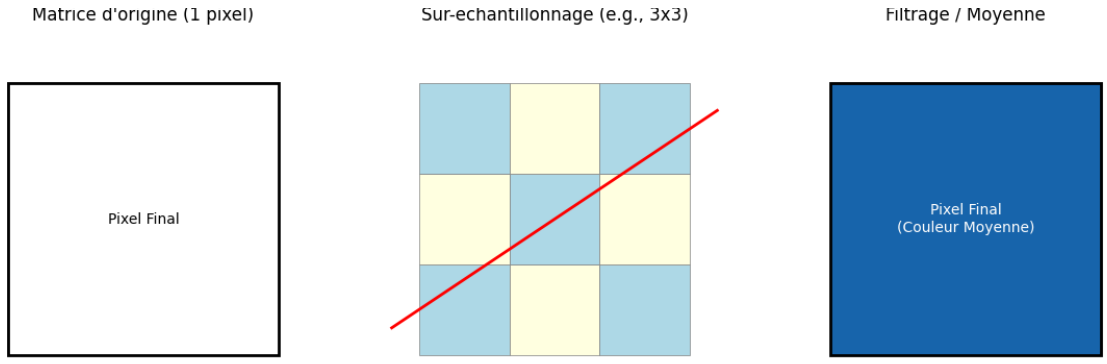

Figure 5.9: Concept du sur-échantillonnage : calculer sur une grille plus fine, puis filtrer.

Plusieurs paramètres influencent le sur-échantillonnage :

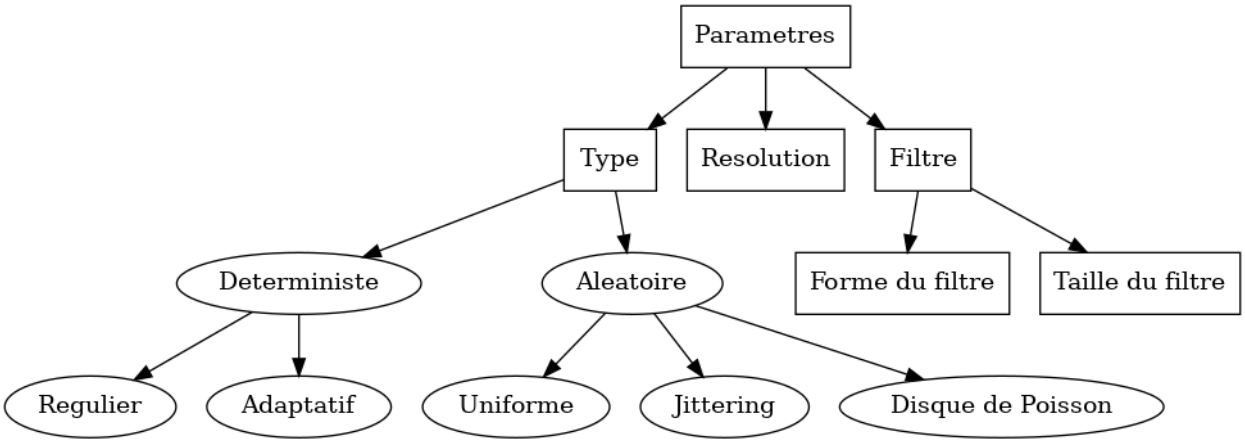

Figure 5.10: Paramètres clés des techniques d'anticrénelage par sur-échantillonnage.

- **Type d'échantillonnage:**
  - **Déterministe:** Les positions des sous-pixels sont fixes.
    - *Régulier:* Grille régulière de sous-pixels (e.g., 2x2, 4x4). Simple mais peut réintroduire des motifs réguliers. Augmente le temps de calcul par n^2 (si $n \times n$ sous-pixels).
    - *Adaptatif:* Raffinement progressif. On échantillonne plus finement seulement dans les zones où c'est nécessaire (e.g., près des contours, zones à haute fréquence de texture), basé sur certains critères (différence de couleur/profondeur entre échantillons voisins). Plus efficace que le régulier mais plus complexe.
  - **Aléatoire:** Les positions des sous-pixels sont choisies aléatoirement ou pseudo-aléatoirement à l'intérieur du pixel. Transforme l'aliasing cohérent en bruit, ce qui est souvent moins gênant pour l'œil humain.
    - *Uniforme:* Chaque position est tirée indépendamment et uniformément. Simple, mais peut créer des amas et des vides.
    - *Jittering (Grille perturbée):* On part d'une grille régulière et on perturbe aléatoirement la position de chaque échantillon à l'intérieur de sa cellule. Bon compromis entre régularité et aléatoire.
    - *Disque de Poisson:* Garantit une distance minimale entre les échantillons, imitant la distribution des photorécepteurs dans la rétine. Donne des résultats de haute qualité mais plus coûteux à générer.
- **Résolution:** Le nombre de sous-pixels utilisés (e.g., 4x, 8x, 16x). Plus la résolution est élevée, meilleur est l'anticrénelage, mais plus le coût de calcul est important.
- **Filtre:** La manière dont les valeurs des sous-pixels sont combinées.
  - *Forme du filtre:* Définit comment pondérer les échantillons.
    - Boîte (Box): Moyenne simple (tous les échantillons ont le même poids). Le plus simple, mais peut flouter l'image.
    - Cône/Tente (Tent): Poids décroissant linéairement avec la distance au centre du pixel. Meilleur compromis netteté/flou que la boîte.
    - Gaussienne: Poids décroissant selon une fonction gaussienne. Donne des résultats doux, souvent perçus comme naturels.
  - *Taille du filtre:* La zone sur laquelle le filtre est appliqué (souvent liée à la taille du pixel final).

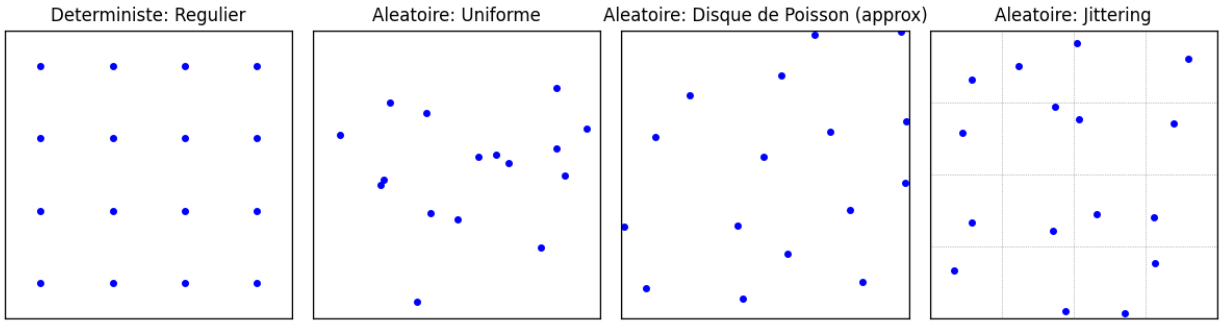

Figure 5.11: Différents types de motifs d'échantillonnage à l'intérieur d'un pixel.

Forme du Filtre

Le filtre détermine comment les contributions des échantillons sont pondérées pour calculer la couleur finale du pixel.

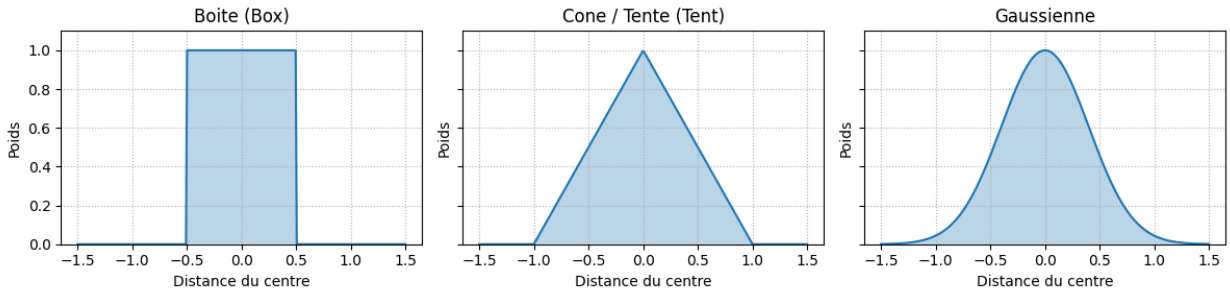

Figure 5.12: Profils 1D de filtres courants : Boîte, Cône/Tente, Gaussienne.

Taille du Filtre

La taille du filtre (souvent exprimée comme un noyau de convolution) affecte également le résultat. Des filtres plus larges peuvent produire plus de flou mais mieux lisser les hautes fréquences. Les filtres gaussiens sont souvent définis par leur écart-type et peuvent être de différentes tailles (e.g., 3x3, 5x5 pixels).

Exemples de noyaux de filtre Gaussiens (non normalisés):

3x3

1	2	1
2	4	2
1	2	1

5x5

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

Filtrage Aléatoire : Monte-Carlo Pondérée

Pour les échantillons aléatoires, une méthode consiste à utiliser un diagramme de Voronoï basé sur les points d'échantillonnage à l'intérieur du pixel. L'aire de la cellule de Voronoï de chaque échantillon peut être utilisée pour pondérer sa contribution à la couleur finale du pixel.

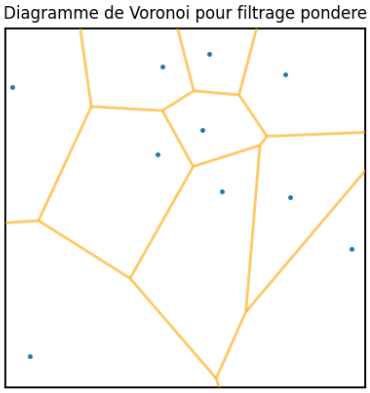

Figure 5.13: Filtrage Monte-Carlo : pondération par l'aire des cellules de Voronoï.

5.2.2 Antialiasing: Algorithmes de tracé de segments corrigés

Une alternative (ou un complément) au sur-échantillonnage est de modifier directement les algorithmes de tracé (comme Bresenham) pour qu'ils calculent une intensité ou une couverture pour chaque pixel proche de la ligne, au lieu de simplement l'allumer ou l'éteindre.

Le principe est d'allumer plusieurs pixels à chaque itération, avec des intensités différentes suivant leur proximité par rapport à la primitive (ligne) idéale.

Approche par Aire de Recouvrement

On peut considérer le segment de droite comme ayant une épaisseur non nulle (par exemple, 1 pixel de large). L'intensité d'un pixel est alors proportionnelle à l'aire d'intersection entre ce pixel (carré) et le segment épaissi.

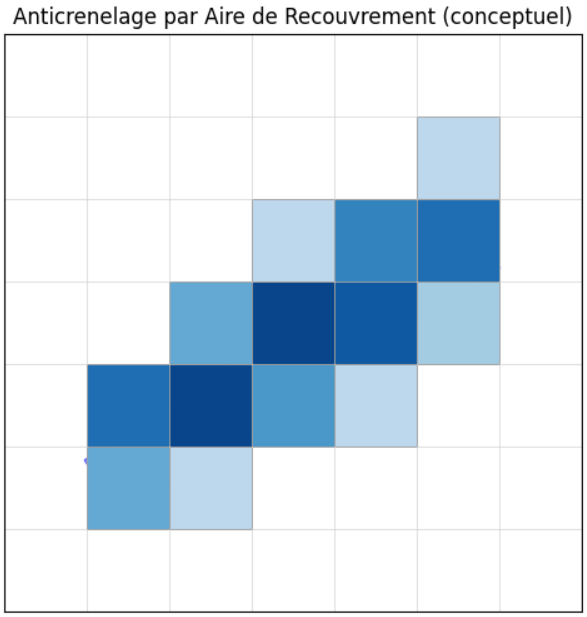

Figure 5.14: Intensité des pixels proportionnelle à l'aire de recouvrement par une ligne épaisse.

Approche par Distance

Plutôt que de calculer des aires d'intersection (qui peuvent être complexes), on peut calculer l'intensité d'un pixel en fonction de sa distance à la ligne idéale. Plus le pixel est proche, plus son intensité est élevée.

- **Approche non-pondérée (simplifiée) :** L'intensité pourrait être liée à la fraction de la ligne passant dans le pixel.
- **Approche pondérée :** L'intensité est calculée par une fonction (souvent gaussienne) de la distance entre le centre du pixel et la ligne.

Algorithme Gupta-Sproull

Cet algorithme est une modification de Bresenham qui implémente l'anticrénelage basé sur la distance.

- À chaque itération de l'algorithme de Bresenham (qui détermine le pixel principal (x, y)), on active non seulement ce pixel, mais aussi ses voisins (généralement un voisinage 3x3).
- Le niveau de gris (intensité) de chaque pixel activé (le pixel principal et ses voisins) est proportionnel (via une fonction, souvent de type cône ou gaussienne) à sa distance perpendiculaire à la ligne idéale.

L'équation de la droite peut s'écrire $ax + by + c = 0$. La distance D d'un point (x_p, y_p) à cette droite est donnée par :

$$D = \frac{|ax_p + by_p + c|}{\sqrt{a^2 + b^2}}$$

L'intensité I du pixel (x_p, y_p) peut être calculée comme $I = f(D)$, où f est une fonction décroissante (e.g., $f(D) = \max(0, 1 - kD)$ pour un filtre Cône, ou une gaussienne $f(D) = e^{-kD^2}$).

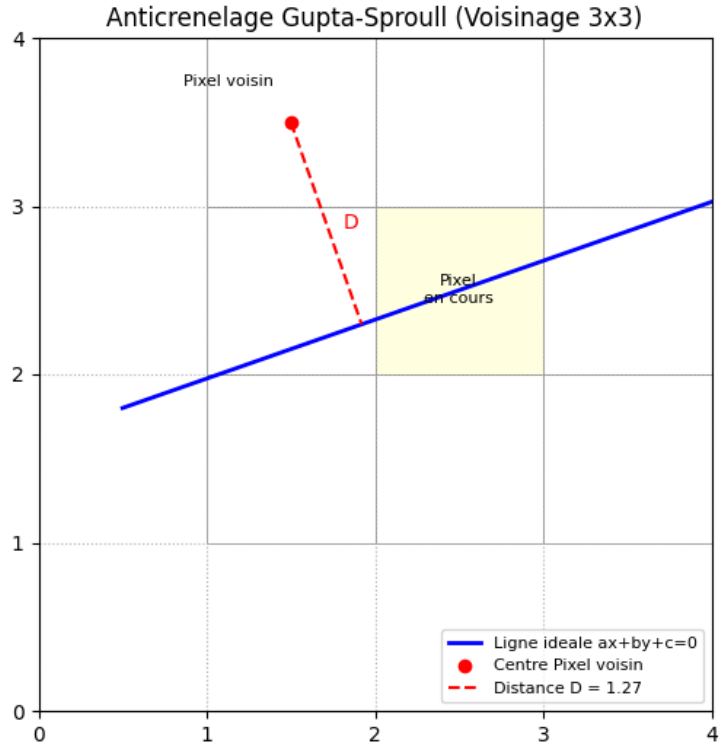

Figure 5.15: Calcul de l'intensité d'un pixel voisin basée sur sa distance D à la ligne idéale (Algorithme Gupta-Sproull).

Il est facile de modifier l'algorithme de trace de segments (Bresenham) pour calculer et appliquer ces intensités. Classiquement, on applique une fonction de filtrage Gaussienne pour passer de la distance D à l'intensité.

5.2.3 Antialiasing: Résultats

Les techniques d'anticrénelage améliorent significativement la qualité visuelle des lignes et des contours dans les images rastérisées.

- **Sans antialiasing :** Les lignes présentent des artefacts d'escalier ("jaggies").
- **Avec antialiasing (Sur-échantillonnage / Moyennage) :** Les jaggies sont réduits, les lignes apparaissent plus lisses mais peuvent être légèrement floues, surtout avec un filtre Boîte ou un filtre Gaussien large.
- **Avec antialiasing (Basé sur la distance / Gupta-Sproull) :** Les jaggies sont également réduits, donnant des lignes lisses. La netteté dépend de la fonction de distance utilisée.

Le choix de la méthode dépend du compromis souhaité entre qualité visuelle, performance et complexité de mise en œuvre.

Image: Three images side-by-side: No antialiasing, Supersampling antialiasing, and Distance-based antialiasing.

Figure 5.16: Comparaison visuelle : (Gauche) Sans anticrénelage, (Centre) Anticrénelage par sur-échantillonnage, (Droite) Anticrénelage basé sur la distance.

5.3 Conclusion

La rastérisation est le processus fondamental de conversion de la géométrie vectorielle en une image pixelisée. Les algorithmes de tracé de lignes, comme celui de Bresenham, sont optimisés pour effectuer cette tâche efficacement en utilisant des calculs entiers. Cependant, la nature discrète de la rastérisation introduit des artefacts de crénelage.

L'anticrénelage vise à atténuer ces artefacts. Le sur-échantillonnage calcule une image à haute résolution puis la filtre, offrant diverses options (régulier, adaptatif, aléatoire) et filtres (boîte, cône, gaussien). Les algorithmes de tracé corrigés, comme Gupta-Sproull, modifient directement le tracé pour calculer des intensités de pixels basées sur la proximité de la primitive. Chaque approche présente des avantages et des inconvénients en termes de qualité, de performance et de complexité. Le choix dépend des exigences spécifiques de l'application graphique.

Chapter 6

Part6

6.1 Clipping (Découpage)

6.1.1 Introduction

Le clipping, ou découpage, est une étape fondamentale du pipeline graphique. Il consiste à éliminer les parties des objets géométriques qui se trouvent en dehors d'une région spécifiée, appelée fenêtre de clipping.

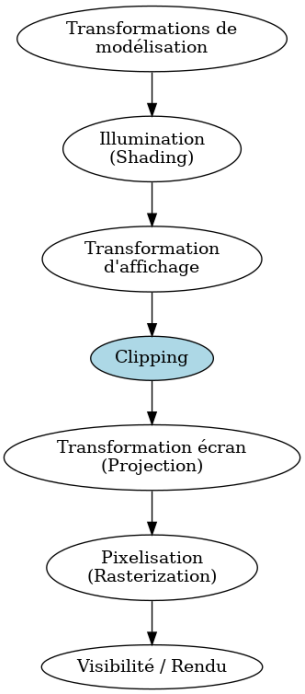

Figure 6.1: Le Clipping dans le Pipeline Graphique.

Le processus implique souvent :

- Le passage en coordonnées normalisées (NDC - Normalised Device Coordinates) : Les coordonnées des objets sont transformées dans un espace canonique, souvent un cube ou un carré unitaire, indépendant de la résolution de l'écran.
- La suppression des parties hors du volume de vision : Seules les parties de la scène visibles depuis le point de vue de la caméra et situées à l'intérieur du volume de vision défini sont conservées.

Schéma conceptuel :
Espace Oeil (Pyramide/Tronc de vision) → Transformation →
Espace NDC (Cube [-1,1] ou [0,1])

Figure 6.2: Passage de l'espace Oeil à l'espace NDC.

Le clipping d'écran est un traitement permettant de réduire le dessin d'un objet graphique à une région de l'écran. Cette région est classiquement un rectangle mais peut être de toute autre forme :

- Carré : Écran standard
- Trapèze : Pare-brise / Rétroviseur
- Circulaire : Lunette / Jumelle

C'est un traitement de base de l'Infographie qui permet d'économiser des opérations inutiles en ne traitant pas les primitives ou parties de primitives invisibles. En 3D, cela permet d'éliminer les calculs en dehors de l'espace visible.

6.1.2 Familles d'algorithmes

Plusieurs familles d'algorithmes existent pour différents types de primitives :

- Clipping Point / Fenêtre
- Clipping Segment / Fenêtre
  - Cohen-Sutherland
  - Liang-Barsky
- Clipping Polygone / Fenêtre
  - Sutherland-Hodgeman

Clipping de points : Point / Fenêtre

Le test est très simple : un point $P(x, y)$ est conservé si et seulement s'il se trouve à l'intérieur des limites de la fenêtre rectangulaire définie par (x_{min}, y_{min}) et (x_{max}, y_{max}).

$$(x_{min} < x < x_{max}) \quad \& \quad (y_{min} < y < y_{max})$$

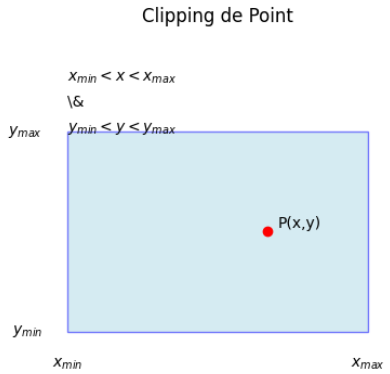

Figure 6.3: Test de clipping pour un point.

Clipping de segments : Segment / Fenêtre

L'objectif est de déterminer les portions de segments de droite qui se trouvent à l'intérieur de la fenêtre de clipping.

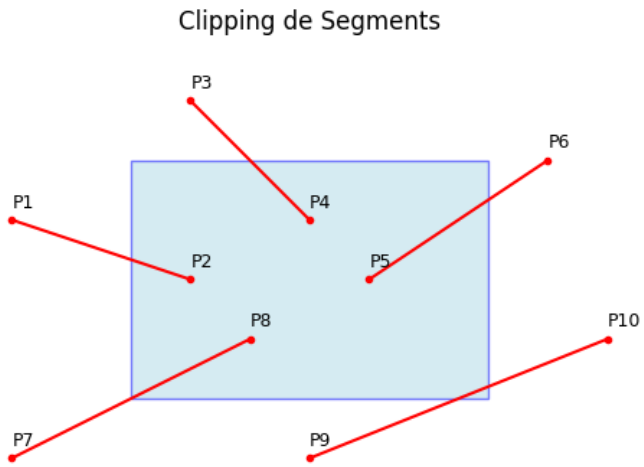

Figure 6.4: Exemples de segments par rapport à une fenêtre.

Algorithme de Cohen & Sutherland Cet algorithme (dû à Ivan Sutherland et collaborateurs) accélère le clipping de segments en utilisant des "codes de zones" (outcodes). L'espace est divisé en 9 régions par les lignes prolongeant les bords de la fenêtre. Chaque région a un code binaire de 4 bits.

- Bit 1 (droite) : 1 si $x > x_{max}$
- Bit 2 (gauche) : 1 si $x < x_{min}$
- Bit 3 (bas) : 1 si $y < y_{min}$
- Bit 4 (haut) : 1 si $y > y_{max}$

Le code pour la fenêtre elle-même est 0000.

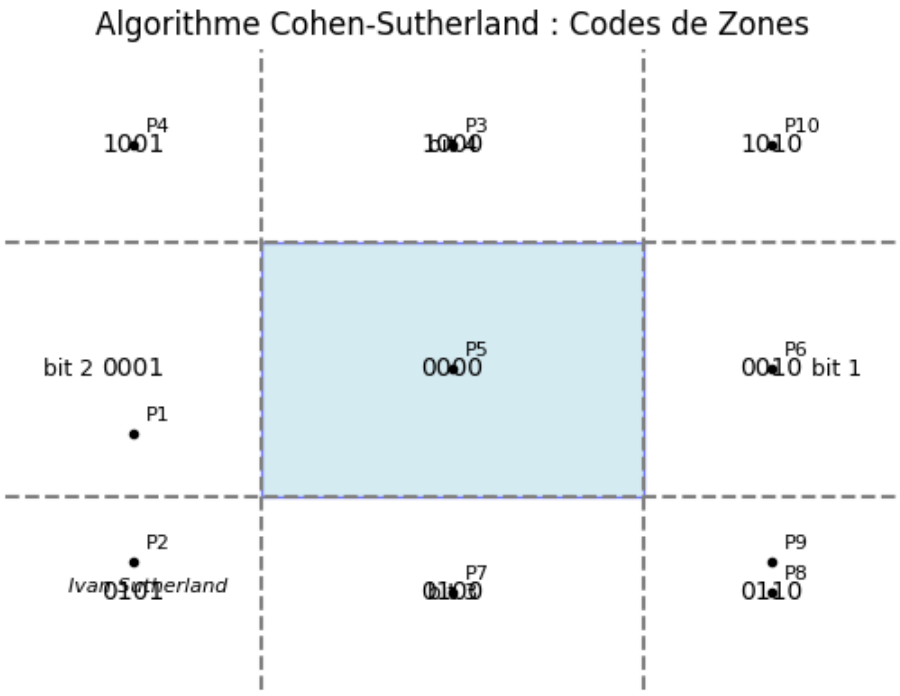

Figure 6.5: Codes de zones utilisés par l'algorithme de Cohen-Sutherland.

L'algorithme traite chaque segment $[P_1, P_2]$ en calculant les codes $c_1 = \text{code}(P_1)$ et $c_2 = \text{code}(P_2)$. Il y a 4 cas :

- **Cas 1 : Acceptation triviale.** Si $c_1 = 0$ et $c_2 = 0$ (ou $c_1|c_2 = 0$), les deux points sont dans la fenêtre. Le segment est entièrement visible.
- **Cas 3 : Rejet trivial.** Si $c_1 \& c_2 \neq 0$ (ET logique bit à bit), les deux points sont dans la même région extérieure (gauche, droite, haut, ou bas). Le segment est entièrement invisible.
- **Cas 2 : Partiellement dedans.** Si un point est dehors ($c_1 \neq 0$ ou $c_2 \neq 0$) et l'autre est dedans ($c_1 = 0$ ou $c_2 = 0$), le segment coupe la fenêtre. Il faut calculer l'intersection.
- **Cas 4 : Potentiellement dedans.** Si $c_1 \& c_2 = 0$ et aucun des points n'est dans la fenêtre, les extrémités sont dans des zones différentes, mais le segment pourrait traverser la fenêtre. Il faut calculer l'intersection.

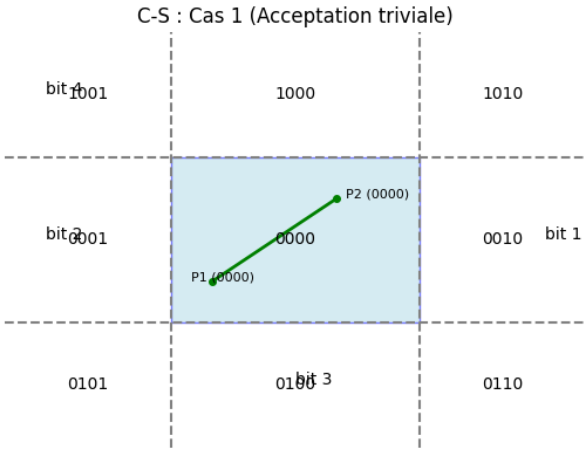

Figure 6.6: Cas 1 : Segment entièrement dans la fenêtre.

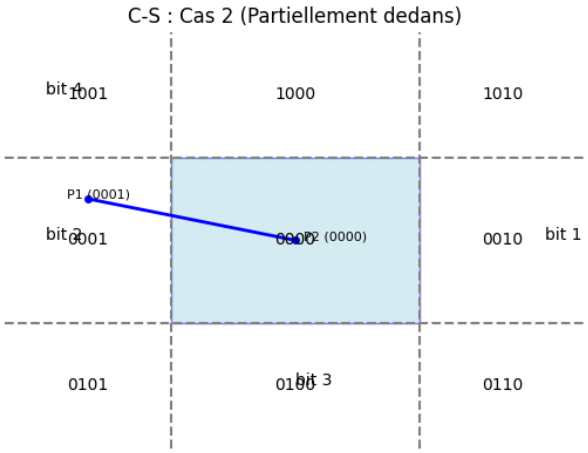

Figure 6.7: Cas 2 : Segment partiellement dedans.

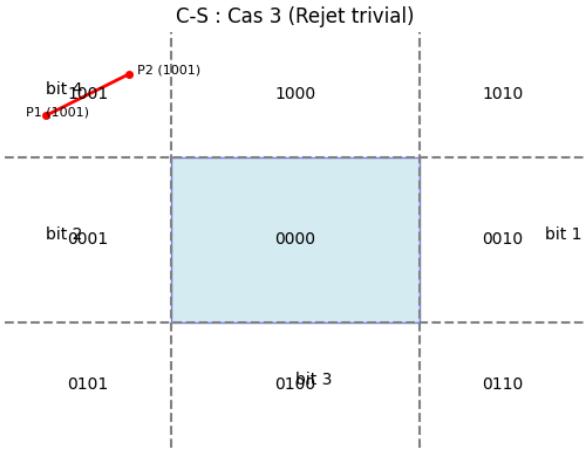

Figure 6.8: Cas 3 : Segment entièrement dehors.

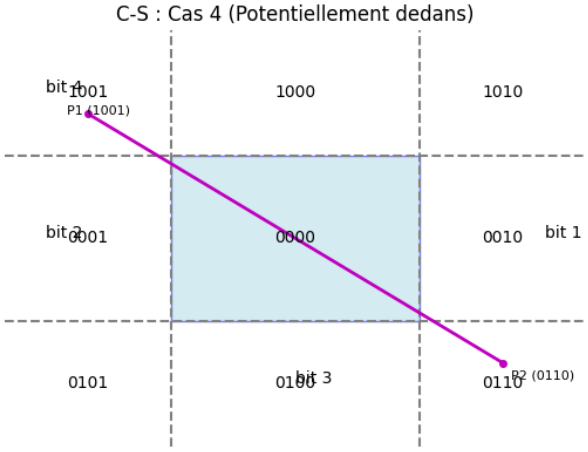

Figure 6.9: Cas 4 : Extrémités dans des zones différentes.

Calcul d'intersection (Cas 2 et 4) : Si un segment n'est ni trivialement accepté ni rejeté, on calcule son intersection avec les bords de la fenêtre. On choisit un point extérieur $P_{ext}(x, y)$ et on calcule l'intersection avec une ligne de clipping (par exemple, $y = y_{max}$). L'équation paramétrique du segment $[P_1(x_1, y_1), P_2(x_2, y_2)]$ est :

$$\begin{aligned} x(t) &= (1-t)x_1 + tx_2 \\ y(t) &= (1-t)y_1 + ty_2 \quad \text{avec } t \in [0,1] \end{aligned}$$

Pour l'intersection avec $y = y_{max}$:

$$(1-t)y_1 + ty_2 = y_{max} \implies t = \frac{y_{max} - y_1}{y_2 - y_1}$$

On calcule ensuite le x correspondant : $x = (1-t)x_1 + tx_2$. Le point (x, y_{max}) remplace P_{ext}. On recalcule le code du nouveau point et on réitère le test (Cas 1, 2, 3, 4) avec le segment réduit. Ce processus est répété jusqu'à ce que le segment soit trivialement accepté ou rejeté.

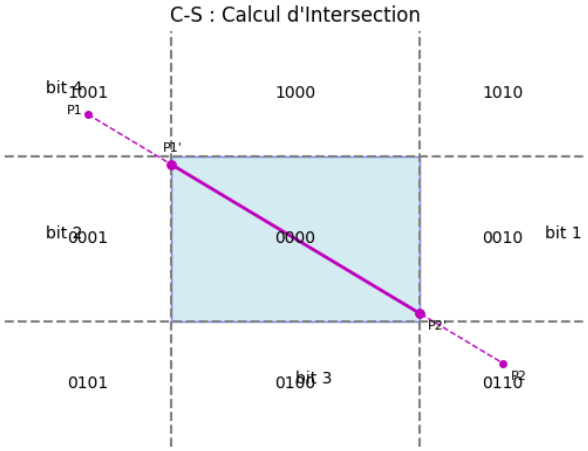

Figure 6.10: Calcul des intersections et réitération des tests.

Extension 3D : L'algorithme se généralise en 3D avec 6 bits pour le code de zone (outcode) :

- Bit 1: $x > x_{max}$ (RIGHT)
- Bit 2: $x < x_{min}$ (LEFT)
- Bit 3: $y > y_{max}$ (TOP)
- Bit 4: $y < y_{min}$ (BOTTOM)
- Bit 5: $z > z_{max}$ (FAR)
- Bit 6: $z < z_{min}$ (NEAR)

Cela nécessite des pré-calculs supplémentaires pour les intersections avec les 6 plans du volume de vue.

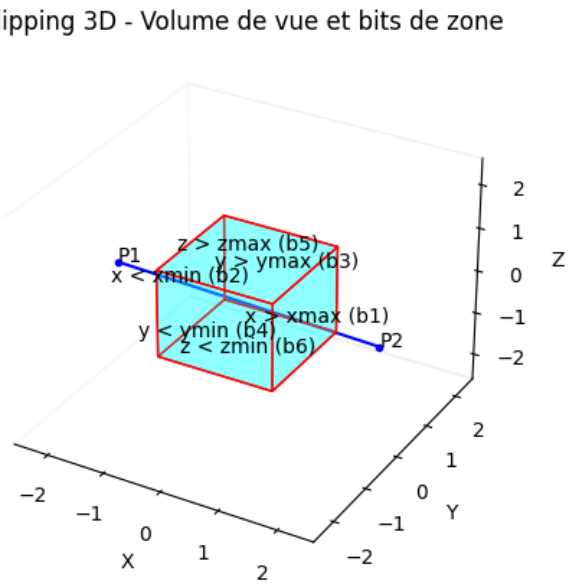

Figure 6.11: Extension 3D de Cohen-Sutherland avec 6 bits.

Algorithme de Liang & Barsky Cette approche découpe une droite en exploitant sa forme paramétrique $P(\alpha) = (1-\alpha)P_1 + \alpha P_2$.

$$x(\alpha) = (1-\alpha)x_1 + \alpha x_2$$
$$y(\alpha) = (1-\alpha)y_1 + \alpha y_2$$

Le paramètre α indique la position sur la droite :

- $\alpha \in [0,1]$: $P(\alpha)$ est sur le segment $[P_1, P_2]$.
- $\alpha < 0$: $P(\alpha)$ est sur la demi-droite à gauche de P_1.
- $\alpha > 1$: $P(\alpha)$ est sur la demi-droite à droite de P_2.

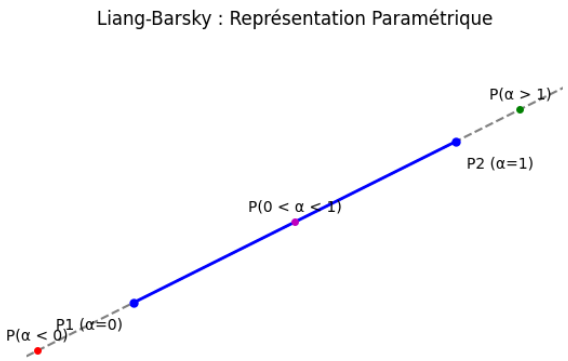

Figure 6.12: Signification du paramètre α dans l'algorithme de Liang-Barsky.

L'algorithme trouve les intersections de la droite infinie $P(\alpha)$ avec les quatre lignes de la fenêtre ($x = x_{min}, x = x_{max}, y = y_{min}, y = y_{max}$). Cela donne jusqu'à quatre valeurs de α. Ces valeurs définissent l'intervalle $[\alpha_{min}, \alpha_{max}]$ correspondant à la partie visible du segment. Initialement, $[\alpha_{min}, \alpha_{max}] = [0, 1]$. Deux cas peuvent se présenter pour les intersections avec les bords de la fenêtre :

- La droite n'est parallèle à aucun côté de la fenêtre : 4 intersections potentielles.
- La droite est parallèle à un des côtés de la fenêtre : 2 intersections potentielles.

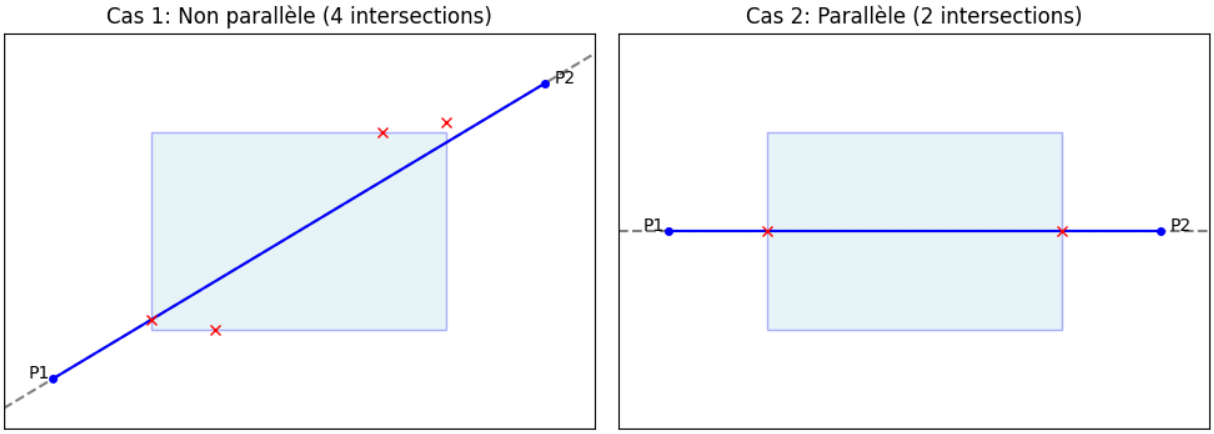

Figure 6.13: Cas d'intersection dans l'algorithme de Liang-Barsky.

Exemple d'exécution : Considérons le segment $P_0 = (30, 20)$ à $P_1 = (280, 160)$ et une fenêtre [70, 230] x [60, 150]. L'algorithme teste les intersections avec chaque bord, mettant à jour l'intervalle $[\alpha_{min}, \alpha_{max}]$ (initialement [0, 1]).

(Note : Les diagrammes suivants illustrent le processus, mais les calculs exacts ne sont pas détaillés ici.)

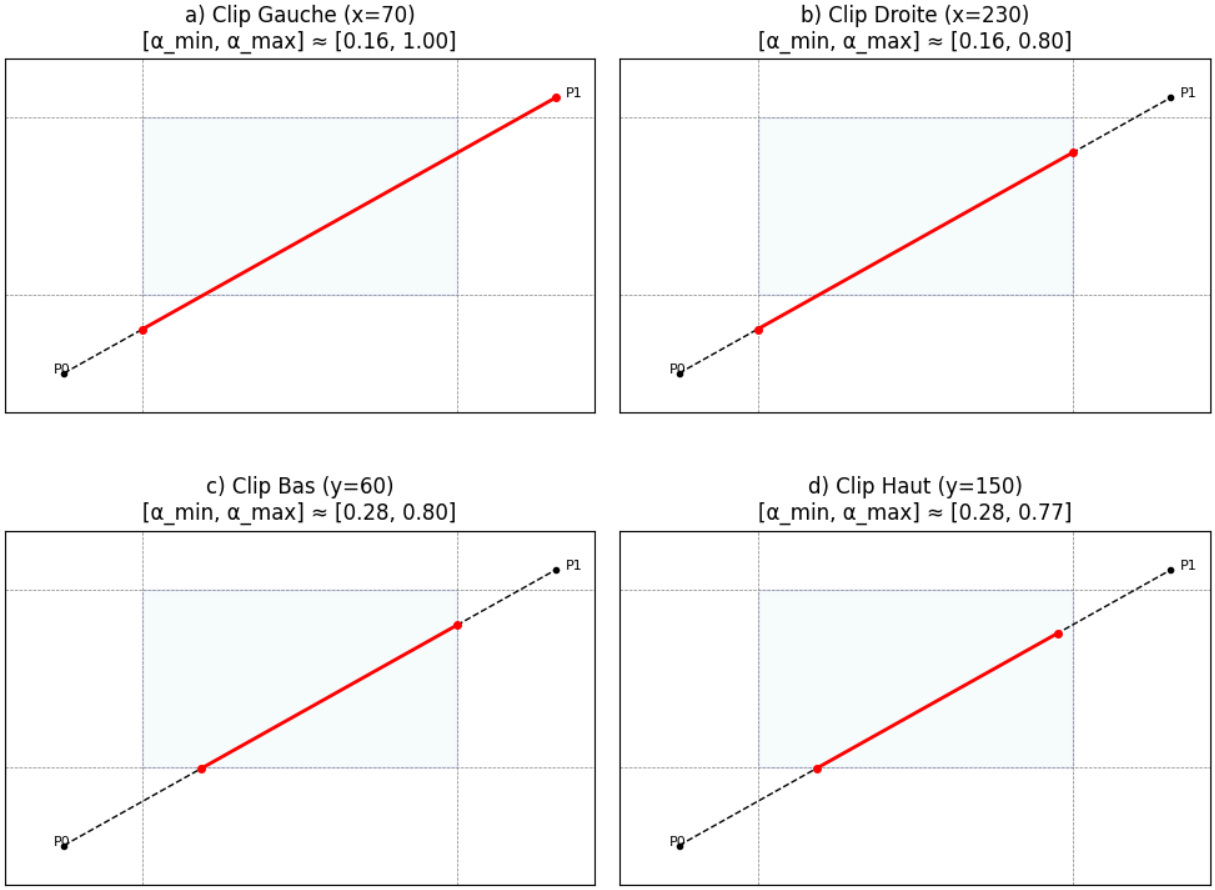

Figure 6.14: Exemple de clipping successif avec Liang-Barsky.

L'ordre d'intersection avec les frontières peut varier (ex: Bas, Gauche, Haut, Droite). L'algorithme calcule les 4 paramètres d'intersection $\alpha_1, \alpha_2, \alpha_3, \alpha_4$. Il trie ensuite ces paramètres et les compare avec l'intervalle [0, 1] pour déterminer la portion visible finale $[\alpha_{vis\_min}, \alpha_{vis\_max}]$. Si $\alpha_{vis\_min} < \alpha_{vis\_max}$, le segment est (partiellement) visible, sinon il est rejeté.

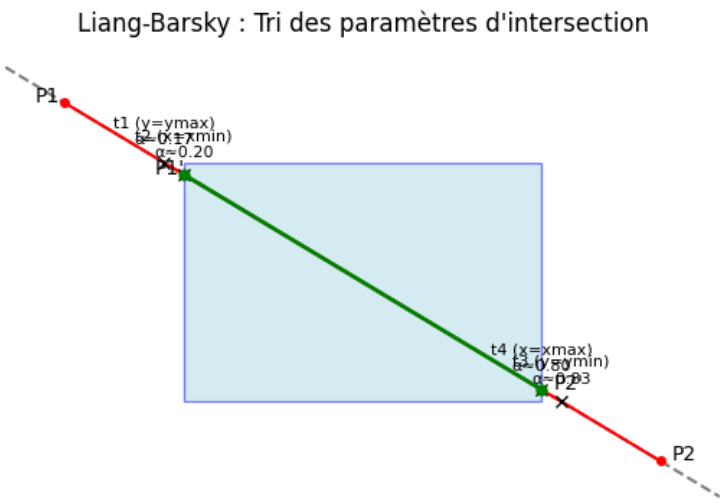

Figure 6.15: Ordre des intersections et détermination du segment visible.

Clipping de polygones : Polygone / Fenêtre

L'objectif est de déterminer la ou les parties d'un polygone qui se trouvent à l'intérieur de la fenêtre de clipping. Le résultat peut être un ou plusieurs polygones.

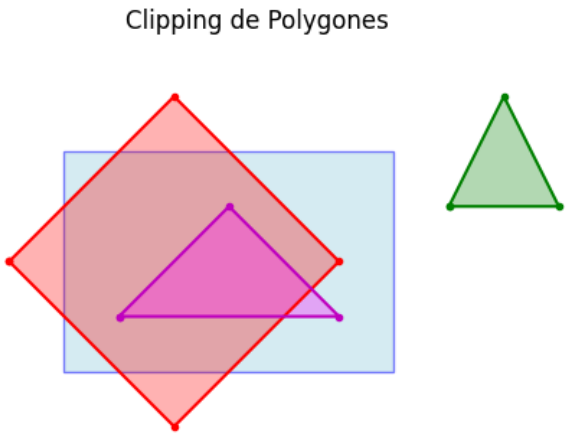

Figure 6.16: Exemples de polygones par rapport à une fenêtre.

Algorithme de Sutherland & Hodgeman Cet algorithme clippe un polygone en le testant successivement contre chaque frontière (bord) de la fenêtre de clipping (Gauche, Droite, Bas, Haut). Pour chaque frontière, l'algorithme prend une liste de sommets en entrée (le polygone original ou le résultat du clipping précédent) et produit une nouvelle liste de sommets en sortie.

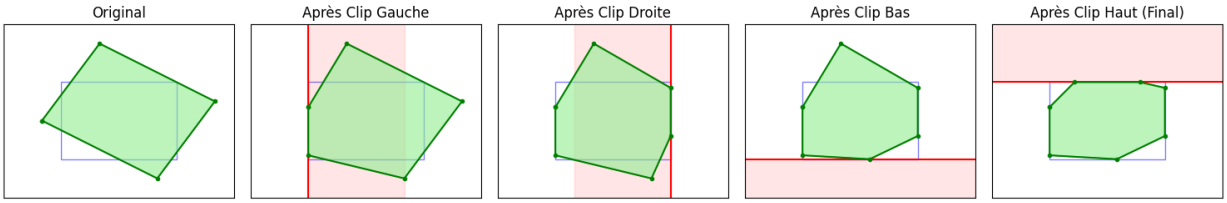

Figure 6.17: Étapes successives du clipping avec Sutherland-Hodgeman.

Pour chaque frontière et chaque arête (P_i, P_{i+1}) du polygone en entrée :

- Tester l'appartenance des extrémités P_i et P_{i+1} par rapport à la frontière (dedans ou dehors).
- Il y a 4 cas pour l'arête (P_i, P_{i+1}):
  1. P_i dedans, P_{i+1} dedans : Garder P_{i+1}.
  2. P_i dedans, P_{i+1} dehors : Calculer l'intersection P', garder P'. (Cas 2)
  3. P_i dehors, P_{i+1} dehors : Ne rien garder.
  4. P_i dehors, P_{i+1} dedans : Calculer l'intersection P', garder P' puis P_{i+1}. (Cas 4)
- Insérer les nouveaux points calculés (intersections) et les points conservés dans la liste de sortie.
- Supprimer les points en dehors de la fenêtre.

La liste de sortie d'une étape devient l'entrée de l'étape suivante. Le résultat final est le polygone clippé.

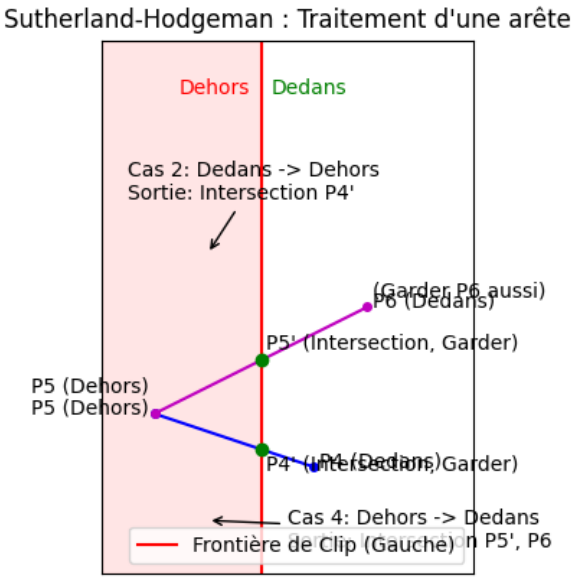

Figure 6.18: Détail du traitement d'une arête par Sutherland-Hodgeman (Cas 2 et 4).

6.2 Élimination des Parties Cachées (Visibilité / Rendu)

6.2.1 Introduction

L'élimination des parties cachées (ou détermination de la surface visible, HSR - Hidden Surface Removal) est une étape cruciale du rendu 3D. Elle consiste à déterminer quelles lignes, arêtes, surfaces ou volumes d'une scène 3D sont visibles depuis le point de vue de l'observateur (caméra).

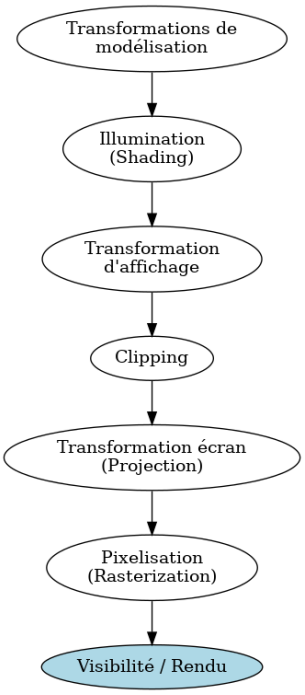

Figure 6.19: La Visibilité / Rendu dans le Pipeline Graphique.

Le but est d'assurer la cohérence visuelle de la scène et de réduire le nombre de primitives traitées par le pipeline graphique, en ne dessinant que ce qui est réellement visible. Cela implique souvent de remplir le framebuffer (tampon d'image) avec la bonne couleur pour chaque pixel, en tenant compte de l'occultation.

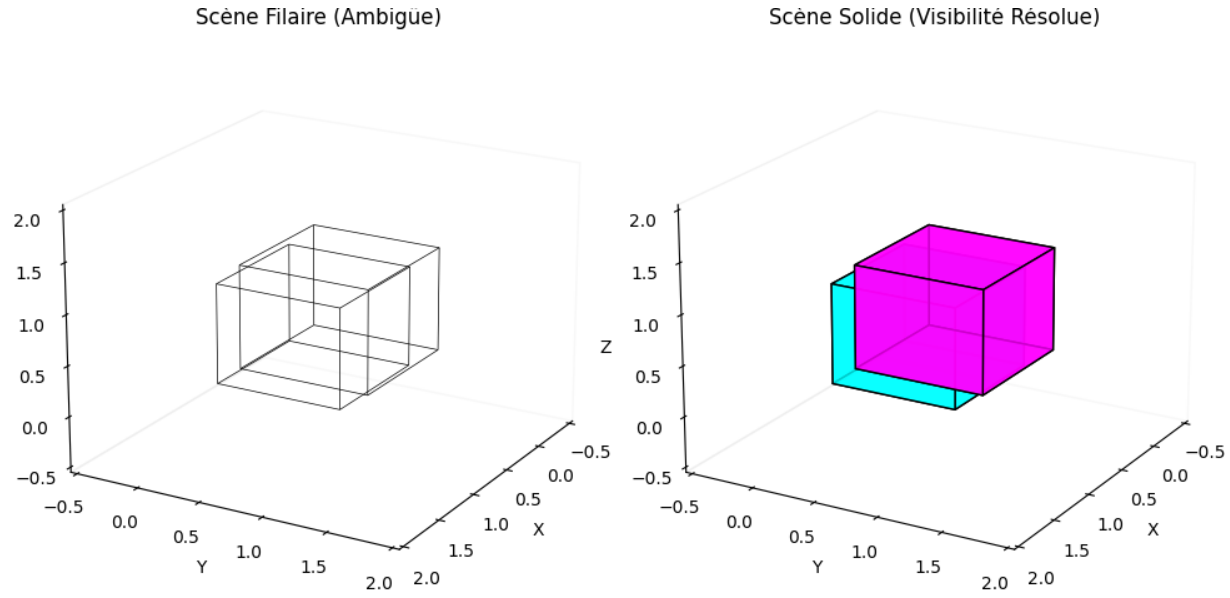

Figure 6.20: Objectif de l'élimination des parties cachées.

6.2.2 Familles d'algorithmes

Il existe deux grandes familles d'algorithmes HSR :

- **Algorithmes Objet (Espace Scène) :**
  - Le masquage se fait sur le modèle de données physiques (calcul en coordonnées du monde).
  - La détermination de la visibilité est valable quelle que soit l'échelle du dessin (résolution de l'image).
  - Caractéristiques : Test objet par objet (potentiellement $O(n^2)$), précision dépend de la résolution des objets, techniques souvent plus complexes à mettre en œuvre.
  - Méthodes : Backface Culling, Algorithme du peintre, Arbres BSP.
- **Algorithmes Image (Espace Image) :**
  - Le masquage se fait au niveau des pixels de l'écran (calcul en coordonnées fenêtre).
  - Caractéristiques : Test de visibilité en chaque pixel (souvent $O(n \times \text{pixels})$), précision dépend de la résolution de l'espace image, recalcul nécessaire si la caméra ou la scène change, techniques souvent plus simples.
  - Méthodes : Warnock, Ligne de balayage (Scan-line Watkins, Scan-line Z-buffer), Test de profondeur (Z-buffer).

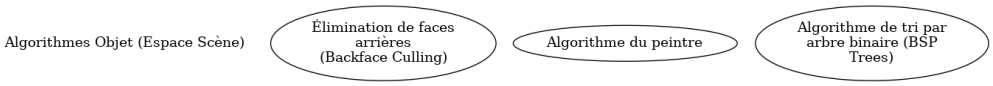

Figure 6.21: Principaux algorithmes HSR dans l'espace objet.

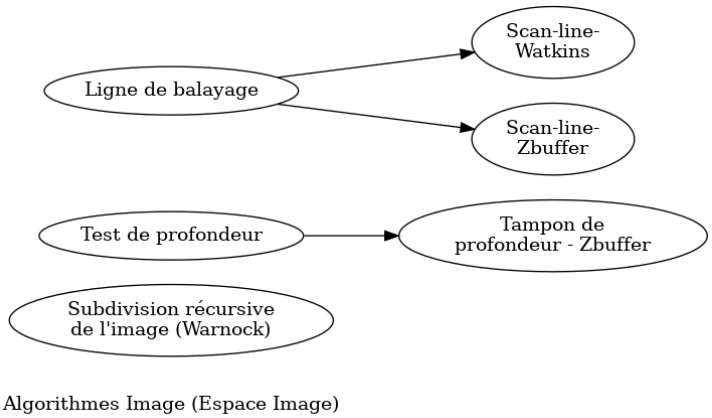

Figure 6.22: Principaux algorithmes HSR dans l'espace image.

Algorithmes Objet (Espace Scène)

Élimination des Faces Arrières (Backface Culling)

Definition 6.2.1. Principe : Garder uniquement les parties de la surface (facettes) dont la normale pointe *vers* l'observateur (ou dans une direction opposée au vecteur de vue). Les faces dont la normale pointe à l'opposé de l'observateur sont considérées comme "arrière" et sont éliminées.

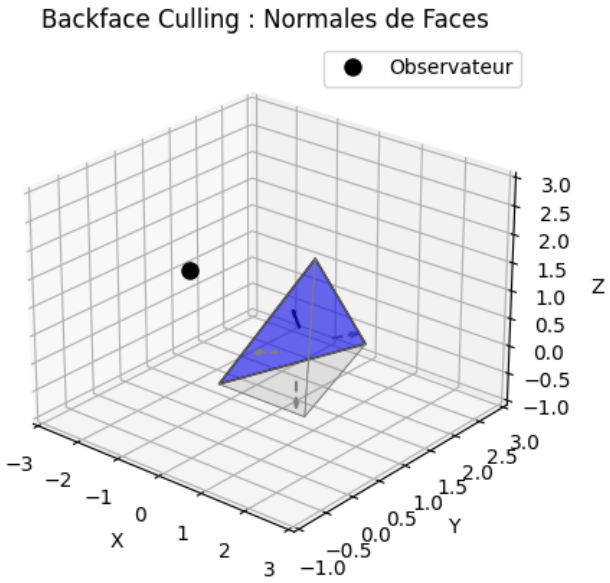

Figure 6.23: Principe du Backface Culling.

Remark 6.2.2. Conditions : La suppression des faces arrières suffit à éliminer les parties cachées si :

- L'objet est seul dans la scène (pas d'occultation par d'autres objets).
- L'objet est convexe. Pour un objet concave, des faces avant peuvent être masquées par d'autres parties de l'objet lui-même.

Definition 6.2.3. Calcul : On utilise le produit scalaire entre la normale à la face $\vec{N}$ et le vecteur allant d'un point de la face (ou son centre) vers l'oeil (ou la caméra) $\vec{V} = \text{Oeil} - \text{Face}$.

- Si $\vec{N} \cdot \vec{V} > 0$: La face est visible (orientée vers l'oeil). On la garde. *(Note: La diapo dit ≤ 0 on garde, ce qui implique que V va de l'oeil vers la face ou que N est la normale intérieure. En suivant la convention usuelle Oeil-¿Face et normale extérieure, N.V ¿ 0 est visible. Si V = Face - Oeil, N.V ¡ 0 est visible. La diapo utilise (Oeil - Face) . Normale ≤ 0 =¿ on garde. Adoptons cette convention.)*
- Si $(\text{Oeil} - \text{Face}) \cdot \vec{N} \leq 0$: On garde le polygone.
- Si $(\text{Oeil} - \text{Face}) \cdot \vec{N} > 0$: On élimine le polygone (face arrière).

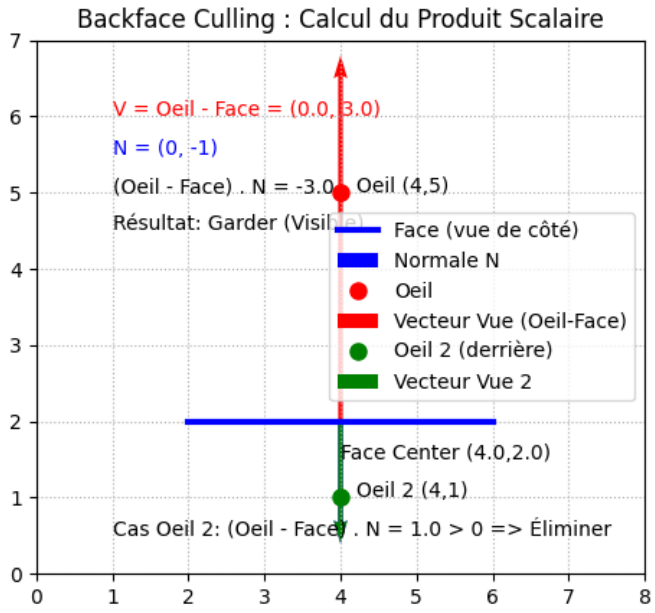

Figure 6.24: Calcul pour déterminer si une face est visible.

Remark 6.2.4. Avantages :

- Économise en moyenne 50% du temps de calcul (pour des objets fermés).
- Faible coût par polygone.
- Souvent utilisé comme étape préliminaire pour d'autres algorithmes HSR plus complexes.

Algorithme du Peintre (Depth-Sort)

Definition 6.2.5. Aussi appelé "Algorithme de Tri par la Profondeur" (Newell, Newell & Sancha 1972). Principe :

1. Trier les polygones de la scène en fonction de leur distance à l'observateur (profondeur, souvent coordonnée Z après projection). On trie du plus éloigné au plus proche.
2. Peindre (dessiner) les polygones dans la mémoire vidéo (framebuffer) dans cet ordre : du plus loin au plus près.

Ainsi, les polygones plus proches recouvrent correctement les polygones plus éloignés qu'ils occultent.

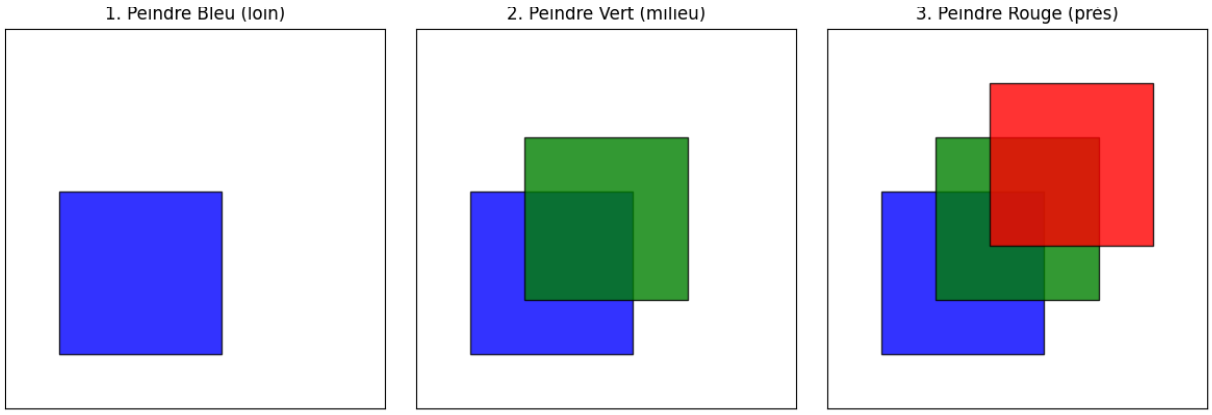

Figure 6.25: Principe de l'algorithme du peintre : peindre du fond vers l'avant.

Remark 6.2.6. Cas ambigu (chevauchement) : Le tri simple basé sur une seule valeur de profondeur (ex: Zmax) n'est pas toujours suffisant. Des polygones peuvent avoir des étendues en Z qui se chevauchent, ou des dépendances cycliques.

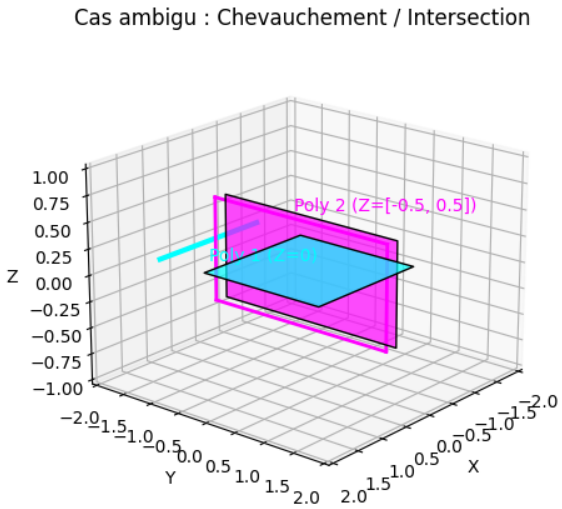

Figure 6.26: Exemple de cas ambigu pour l'algorithme du peintre.

Remark 6.2.7. Résolution des cas ambigus :

- Trier les polygones en fonction de leur plus grande coordonnée en Z (distance max à la caméra).
- Si deux polygones P et Q ont des étendues en Z qui se recouvrent ($[Z^P_{min}, Z^P_{max}]$ et $[Z^Q_{min}, Z^Q_{max}]$ overlap) :
  - Test des boîtes englobantes : Si les projections 2D (XY) des boîtes englobantes sont séparées, l'ordre n'importe pas.

– Test d'orientation : Tester si P est entièrement "derrière" Q ou vice-versa par rapport au plan de l'autre polygone.

– Test de projection : Si les projections 2D se chevauchent, vérifier si P occulte Q ou Q occulte P.

– Si rien ne marche (intersection ou dépendance cyclique) : Couper l'un des polygones (ou les deux) par le plan de l'autre. On obtient de nouveaux polygones plus petits qu'il faut réinsérer dans la liste triée.

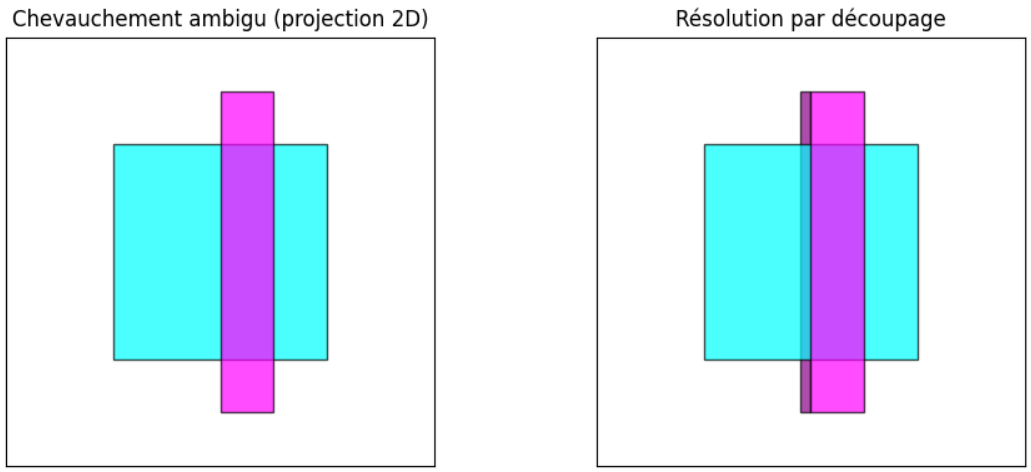

Figure 6.27: Résolution des ambiguïtés par découpage de polygones.

Remark 6.2.8. Caractéristiques :

- Le plus intuitif des algorithmes.
- Coût en mémoire : Peut nécessiter de stocker tous les polygones. L'affichage direct à l'écran évite un grand buffer ($O(p)$ où p est le nombre de polygones).
- Coût de calcul : Dominé par le tri ($O(n \log n)$ ou $O(n^2)$ en cas de tests complexes) et potentiellement le découpage.
- Efficace surtout sur des petites scènes où les découpages sont rares.

Partition Binaire de l'Espace (BSP-Trees)

Definition 6.2.9. Un arbre BSP (Binary Space Partition) est un arbre binaire utilisé pour trier des primitives (polygones) dans l'espace 3D. Il permet une détermination efficace de la visibilité.

Principe (Construction de l'arbre - Schumaker 1969, Fuchs 1980)

1. Choisir un polygone (ou une face) de la scène comme plan séparateur (nœud racine).
2. Ce plan divise l'espace en deux demi-espaces : "avant" (in front) et "arrière" (behind) par rapport à l'orientation du plan/polygone.
3. Répartir tous les autres polygones de la scène dans ces deux demi-espaces :
   - Ceux entièrement "devant" vont dans le sous-arbre "avant" (fils droit/gauche).

- Ceux entièrement "derrière" vont dans le sous-arbre "arrière" (fils gauche/droit).
- Ceux qui sont coupés ("à cheval") par le plan sont divisés en deux nouveaux polygones, chacun étant placé dans le demi-espace correspondant.

4. Réitérer le processus récursivement sur les ensembles de polygones dans chaque demi-espace, jusqu'à ce que chaque feuille de l'arbre contienne au plus un polygone (ou fragment).

Schémas conceptuels (voir slides 42-47) :
1. Ensemble initial d'objets.
2. Choix d'un plan (associé à un objet/polygone) → Nœud racine.
3. Répartition des objets : 'avant' (P - proche/positif) / 'arrière' (L - loin/négatif).
4. Objet coupé ('Ouch') → Division en deux, répartition.
5. Répétition récursive pour les sous-ensembles avant/arrière → Construction de l'arbre.

Figure 6.28: Illustration conceptuelle de la construction d'un arbre BSP.

Utilisation pour le rendu (Parcours de l'arbre) Le parcours de l'arbre BSP pour le rendu dépend de la position de l'observateur par rapport aux plans séparateurs des nœuds. L'ordre de parcours garantit que les objets sont dessinés du plus loin au plus près (similaire à l'algorithme du peintre, mais sans tri explicite à chaque frame).

Algorithme de rendu récursif ('renderBSP(node, eye$_p$os)') :

1. Si 'node' est vide (feuille), retourner.
2. Déterminer si 'eye$_p$os'*est*"*devant*"*ou*"*derrière*"*leplandu*'*node*'.*Si*'*eye$_p$os*'*estdevant* :
3. 4. 'Proche = node-¿avant'
   5. 'Loin = node-¿arriere'

Sinon ('eye$_p$os'*estderrière*) :
'Proche = node-¿arriere'

'Loin = node-¿avant'

Appeler récursivement 'renderBSP(Loin, eye$_p$os)'(*traiterlesous* − *arbrelointaind*′*abord*).*Dessinerlaprimitive*(*polygone*)*associéeau*'*node*'*courant*.

Appeler récursivement 'renderBSP(Proche, eye$_p$os)'(*traiterlesous* − *arbreprocheensuite*).

Listing 6.1: Pseudo-code du rendu avec un arbre BSP

```
renderBSP(BSPtree *T, Point eye_pos) {
  if (T == NULL) return; // Feuille vide

  BSPtree *Proche, *Loin;
  Plane plane = T->plane; // Plan du noeud courant

  // Determiner si l'oeil est devant ou derriere le plan
  if (is_in_front(eye_pos, plane)) {
    Proche = T->front_child; // Sous-arbre avant
    Loin = T->back_child;   // Sous-arbre arriere
  } else {
    Proche = T->back_child;   // Sous-arbre arriere
    Loin = T->front_child; // Sous-arbre avant
  }
```

```
    // 1. Traiter le sous-arbre lointain
    renderBSP(Loin, eye_pos);

    // 2. Dessiner la primitive du noeud courant
    if (T->polygon != NULL) { // Verifier si c'est un noeud interne ou
        ↪ feuille avec polygone
        renderObject(T->polygon);
    }

    // 3. Traiter le sous-arbre proche
    renderBSP(Proche, eye_pos);
}
```

Schémas conceptuels (voir slides 49-64) :
1. Position de l'observateur (Oeil) par rapport aux plans.
2. Parcours récursif : Loin → Dessin Nœud → Proche.
3. L'ordre de dessin final dépend du point de vue (Viewpoint A vs Viewpoint B).
4. Résultat : Scène rendue avec occultation correcte.

Figure 6.29: Illustration conceptuelle du rendu avec un arbre BSP.

Algorithmes Image (Espace Image)

Ces algorithmes opèrent au niveau des pixels dans l'espace image (fenêtre/écran).

Remark 6.2.10. Caractéristiques :

- Test de visibilité effectué pour chaque pixel.
- Précision limitée par la résolution de l'image.
- Doit être recalculé si la caméra ou la scène change.
- Souvent plus simples à implémenter que les algorithmes objet complexes.

Méthodes principales (voir Figure 6.22) :

- **Z-buffer (Tampon de profondeur)** : Le plus commun. Maintient un tampon stockant la profondeur (Z) du pixel visible le plus proche pour chaque pixel de l'écran. Chaque polygone est rastérisé (converti en pixels), et chaque pixel généré n'est écrit dans le framebuffer que si sa profondeur est inférieure à celle déjà stockée dans le Z-buffer.
- **Algorithmes Scan-line** : Traitent l'image ligne par ligne. Pour chaque ligne de balayage, ils déterminent les segments visibles en calculant les intersections des polygones avec la ligne et en gérant les informations de profondeur le long de la ligne. (Watkins, Scan-line Z-buffer).
- **Algorithme de Warnock** : Algorithme récursif de subdivision de l'image. Divise récursivement les régions de l'écran jusqu'à ce que la visibilité dans une région soit simple à déterminer (ex: région vide, couverte par un seul polygone, etc.).

Chapter 7

Part7

7.1 Introduction : Élimination des Parties Cachées

Lors de la création d'images de synthèse tridimensionnelles, plusieurs objets peuvent occuper la même position sur l'écran (pixel). Le problème de l'élimination des parties cachées consiste à déterminer quelles parties de quels objets sont visibles depuis le point de vue de l'observateur (la caméra) afin de ne dessiner que celles-ci. Sans ce processus, les objets les plus éloignés pourraient être dessinés par-dessus les objets plus proches, créant une image incorrecte. Plusieurs approches algorithmiques existent pour résoudre ce problème. On les classe souvent en deux catégories : les algorithmes opérant dans l'espace objet et ceux opérant dans l'espace image. Ce chapitre se concentre sur les **algorithmes images**, qui déterminent la visibilité au niveau de chaque pixel de l'image finale.

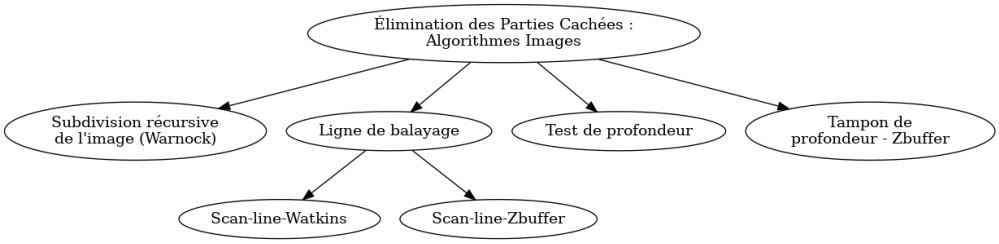

Figure 7.1: Classification des algorithmes d'élimination des parties cachées opérant dans l'espace image.

Parmi les algorithmes images, nous allons détailler l'algorithme de subdivision récursive de l'image (Warnock) et l'algorithme du tampon de profondeur (Z-buffer). Les algorithmes basés sur la ligne de balayage (Scan-Line), tels que ceux de Watkins ou utilisant un Z-buffer par ligne, seront brièvement évoqués.

7.2 Subdivision Récursive de l'Image : Algorithme de Warnock

L'algorithme de Warnock, développé par John Warnock, est un algorithme d'élimination des parties cachées basé sur le principe "diviser pour régner" (divide and conquer). L'idée est de subdiviser récursivement l'image en quadrants (structure de quadtree) jusqu'à ce que le contenu de chaque quadrant soit suffisamment simple pour être affiché directement.

7.2.1 Principe "Diviser pour Régner"

L'algorithme commence avec la fenêtre d'affichage entière.

1. On examine le contenu du quadrant courant.
2. Si le contenu est simple (voir cas simples ci-dessous), on affiche le quadrant et on arrête la subdivision pour cette branche.
3. Si le contenu est complexe, on subdivise le quadrant en quatre sous-quadrants de taille égale.
4. On applique récursivement l'algorithme à chacun des sous-quadrants.

Ce processus est illustré ci-dessous, montrant les étapes successives de subdivision de l'image.

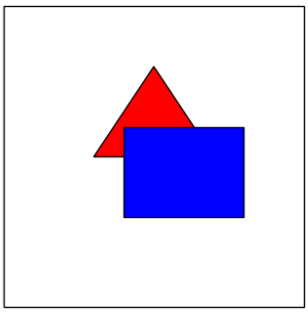

Figure 7.2: Image initiale avec deux polygones.

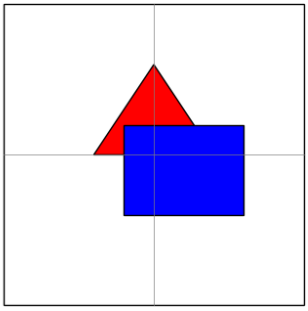

Figure 7.3: Première subdivision en quadtree.

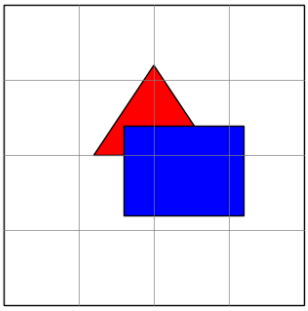

Figure 7.4: Deuxième niveau de subdivision dans les quadrants complexes.

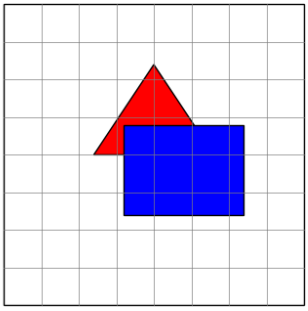

Figure 7.5: Troisième niveau de subdivision.

7.2.2 Traitement des Quadrants

Pour chaque quadrant, l'algorithme détermine la liste des polygones potentiellement visibles (ceux qui intersectent le quadrant). Ensuite, il analyse la situation :

Cas Simples

Le contenu du quadrant est considéré comme simple et peut être tracé directement dans les cas suivants :

- **Rien dans le quadrant :** Le quadrant est vide. On le remplit avec la couleur du fond.
- **Un seul polygone :**
  - Si le polygone **couvre totalement** le quadrant, on remplit le quadrant avec la couleur du polygone (éventuellement après calcul d'ombrage).
  - Si le polygone **couvre partiellement** le quadrant, on remplit le quadrant avec la couleur du fond, puis on dessine la partie du polygone contenue dans le quadrant. (Note : une

approche alternative est de subdiviser même si un seul polygone est partiellement dedans, mais le cas simple ici suggère de le dessiner directement).

– Si le polygone est **contenu** dans le quadrant, on le dessine avec sa couleur sur le fond.

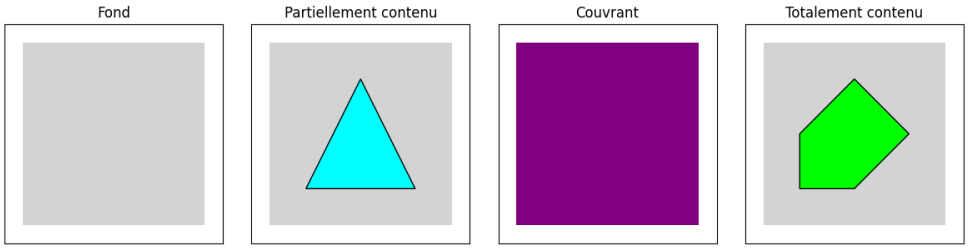

Figure 7.6: Cas simples de traitement d'un quadrant dans l'algorithme de Warnock.

Cas Complexes : Plusieurs Polygones

Si plusieurs polygones sont présents dans le quadrant, la situation est complexe et nécessite une analyse plus poussée ou une subdivision :

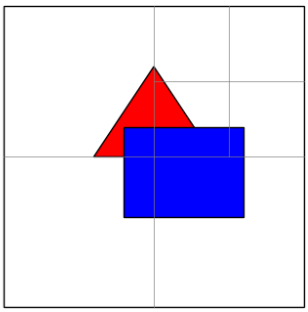

Figure 7.7: Cas complexe : quadrant contenant plusieurs polygones.

1. **Trier les polygones :** Les polygones dans le quadrant sont triés selon leur profondeur (distance à l'observateur).
2. **Vérifier le masquage complet :** On vérifie si le polygone le plus proche (après tri) masque complètement tous les autres polygones à l'intérieur du quadrant.
   - Si un polygone P couvre entièrement le quadrant et est plus proche que tous les autres polygones intersectant le quadrant, alors on remplit le quadrant avec la couleur de P. Ce test peut parfois être effectué sans calculs complexes de profondeur si P est clairement devant les autres.
3. **Subdiviser :** Si aucun polygone ne masque entièrement les autres de manière simple, on subdivise le quadrant en quatre sous-quadrants et on réitère le processus pour chacun.

Critère d'Arrêt

La subdivision récursive s'arrête lorsque le quadrant devient plus petit qu'un pixel. À ce niveau de résolution :

- On calcule la profondeur de chaque polygone intersectant le pixel (au centre du pixel, par exemple).
- On détermine le polygone le plus proche de l'observateur.
- On affiche le pixel avec la couleur de ce polygone le plus proche.

7.2.3 Découpage Récursif et Problèmes Potentiels

Le découpage récursif continue jusqu'à ce qu'une décision puisse être prise pour chaque région.

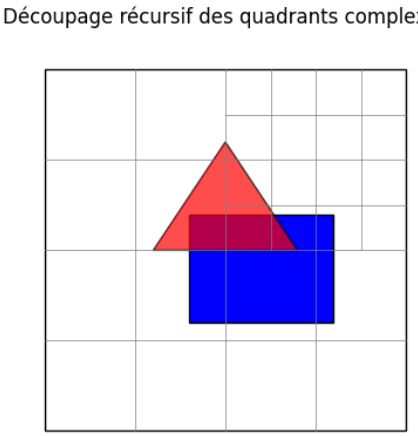

Figure 7.8: Exemple de découpage récursif pour résoudre les cas complexes.

Un problème peut survenir si les tests simples de masquage échouent à détecter une situation de masquage réel. Par exemple, un polygone S1 peut masquer entièrement d'autres polygones S2, S3, S4 à l'intérieur d'un quadrant donné, mais si S1 ne remplit pas *entièrement* le quadrant, les tests simples (comme vérifier si un polygone couvre le quadrant) peuvent ne pas le détecter. Le test de profondeur au niveau du pixel résout ce problème, mais le but de la subdivision est d'éviter ces calculs coûteux autant que possible.

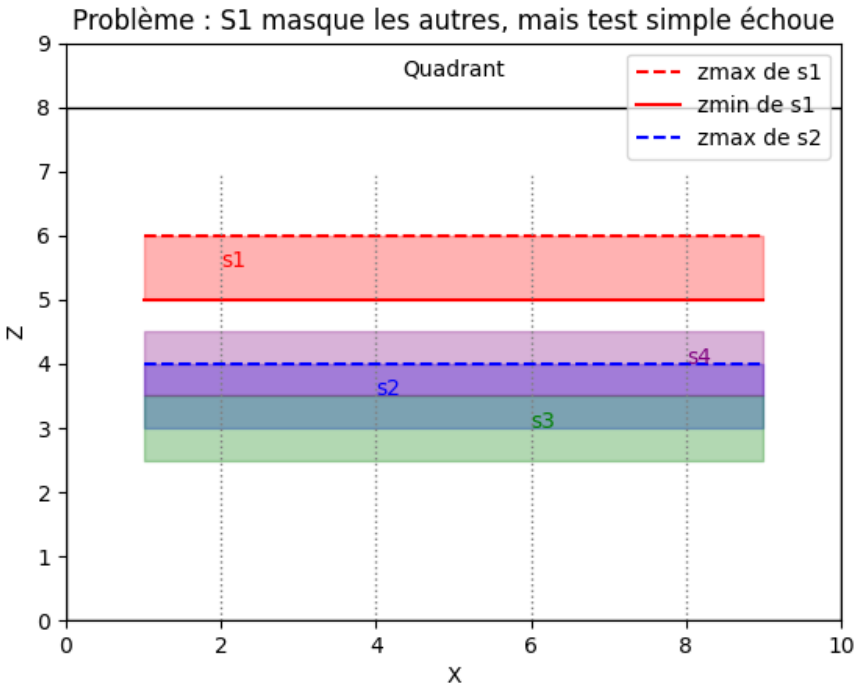

Figure 7.9: Illustration du problème où un polygone S1 masque d'autres polygones (S2, S3, S4) dans le quadrant selon l'axe Z, mais les tests simples peuvent ne pas le détecter si S1 ne couvre pas entièrement le quadrant.

Dans ce cas (Figure 7.9), la subdivision est nécessaire car le test simple (S1 couvre-t-il le quadrant ?) échoue, même si S1 est effectivement le seul polygone visible dans cette zone.

7.3 Algorithme de Balayage (Scan-Line) - Aperçu

Les algorithmes de balayage traitent l'image ligne par ligne (horizontalement, le long de l'axe y). Le principe général est d'éliminer les parties cachées pour chaque ligne de balayage individuellement.

- **Principe :** Pour une ligne de balayage donnée (y constant), on détermine tous les segments de polygones qui intersectent cette ligne.
- **Visibilité :** On calcule ensuite, pour chaque segment le long de la ligne (en x), quel segment est le plus proche de l'observateur (en utilisant la coordonnée z).
- **Affichage :** Seuls les segments visibles sont affichés sur la ligne de balayage courante.

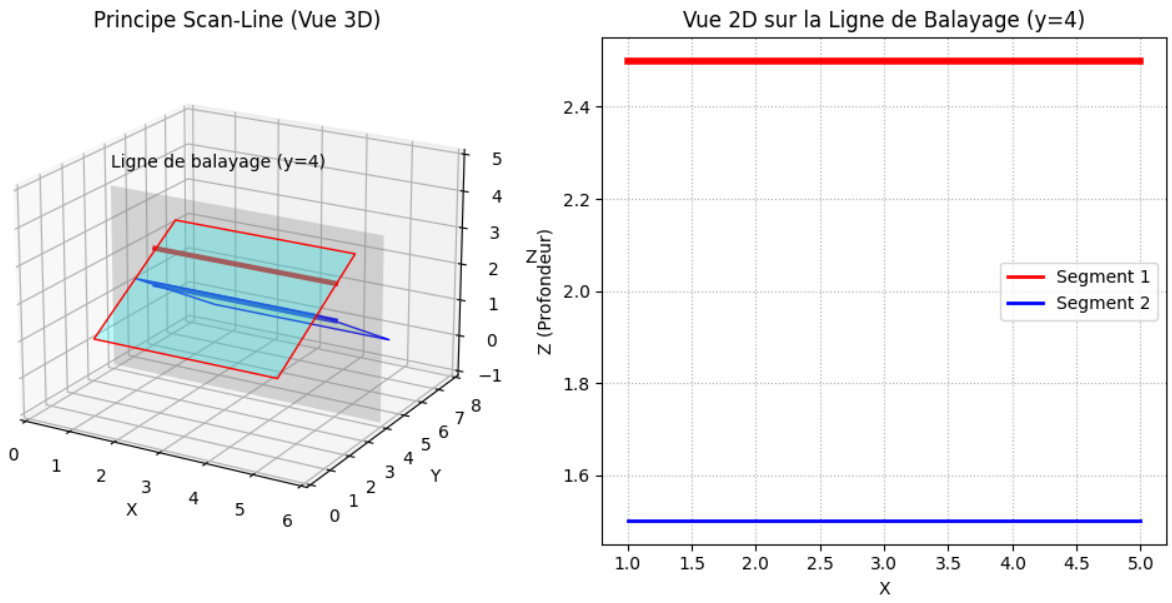

Figure 7.10: Principe de l'algorithme de balayage (Scan-Line). La scène 3D est coupée par un plan correspondant à la ligne de balayage courante (gauche). Les intersections forment des segments dans le plan X-Z (droite), où la visibilité est déterminée par la profondeur Z.

Pour une ligne de balayage donnée, plusieurs cas peuvent se présenter lors de la comparaison des segments issus de différents polygones. La gestion des intervalles de segments et de leur profondeur est cruciale.

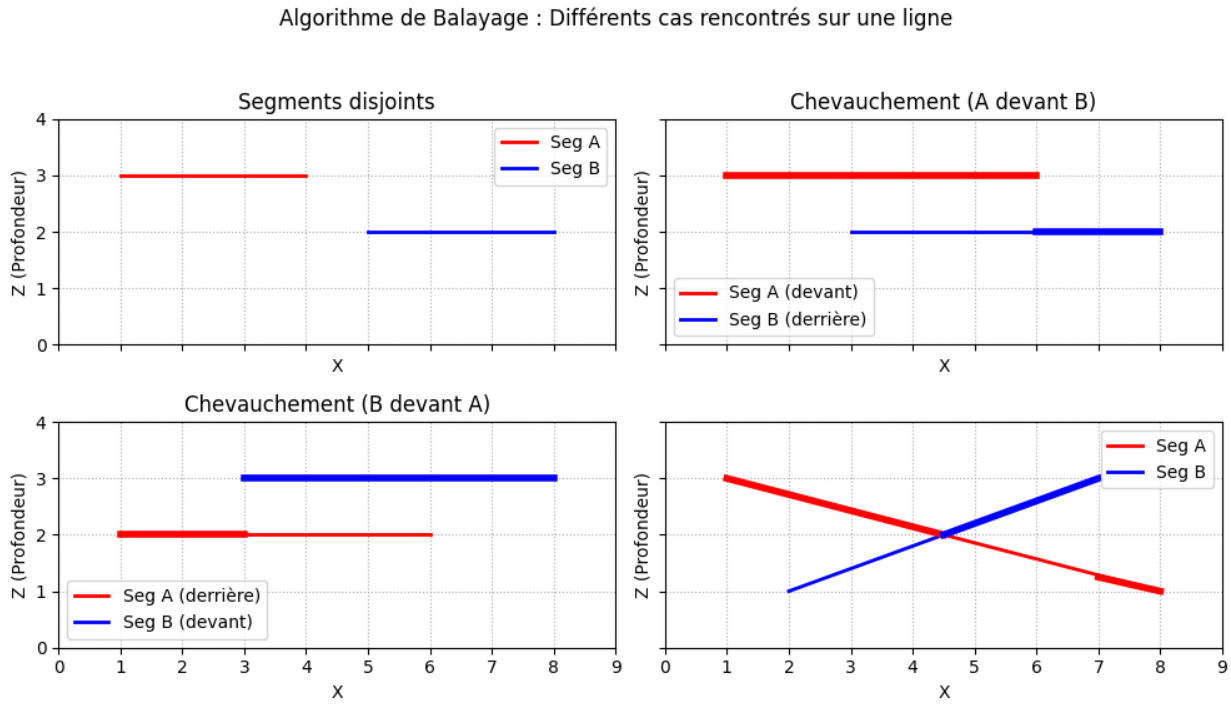

Figure 7.11: Différents cas de segments rencontrés sur une ligne de balayage et détermination de la visibilité basée sur la profondeur Z.

Des algorithmes spécifiques comme celui de Watkins gèrent ces cas en maintenant une liste active d'arêtes et en calculant les intersections et la visibilité le long de chaque ligne. Une variante utilise un tampon de profondeur (Z-buffer) unidimensionnel pour chaque ligne de balayage.

7.4 Tampon de Profondeur (Z-buffer)

L'algorithme du Z-buffer, développé par Edwin Catmull en 1974, est l'un des algorithmes d'élimination des parties cachées les plus simples et les plus utilisés, notamment car il est souvent implémenté directement dans le matériel graphique.

7.4.1 Concept

L'idée principale est de maintenir une mémoire tampon bidimensionnelle, appelée tampon de profondeur ou Z-buffer, de même taille que l'image à générer. Chaque élément de ce tampon stocke la profondeur (coordonnée Z) de l'objet le plus proche trouvé jusqu'à présent pour le pixel correspondant.

- Le calcul de l'image se fait en traitant les objets (polygones/facettes) séquentiellement, les uns après les autres, sans ordre particulier (pas de tri préalable nécessaire).
- Pour chaque polygone, on détermine les pixels qu'il recouvre sur l'écran.
- Pour chaque pixel (x, y) recouvert par le polygone, on calcule sa profondeur Z.
- On compare cette profondeur Z avec la valeur Z déjà stockée dans le Z-buffer à la position (x, y).

- Si la nouvelle profondeur Z est inférieure (plus proche de l'observateur, selon la convention utilisée) à la valeur stockée, cela signifie que ce polygone est devant ce qui avait été dessiné précédemment à ce pixel. On met alors à jour le Z-buffer avec la nouvelle profondeur Z et on écrit la couleur du polygone dans le pixel (x, y) de l'image (tampon couleur).
- Sinon (si Z est supérieur ou égal), le polygone est derrière ou au même niveau que ce qui est déjà visible, donc on ne modifie ni le Z-buffer ni le tampon couleur pour ce pixel.

Il ne s'agit donc pas d'un tri global des objets, mais d'une série de comparaisons et de mises à jour locales (calcul de maximum ou minimum, selon la convention Z) pour chaque pixel.

7.4.2 Zones Mémoire

Deux zones de mémoire principales sont utilisées :

- **Tampon de Profondeur (Z-buffer)** : Tableau 2D de la taille de l'écran, stockant la profondeur Z du pixel visible le plus proche. Il est initialisé avec une valeur de profondeur maximale (infinie, ou la valeur la plus éloignée possible) pour chaque pixel.
- **Tampon Couleur (Framebuffer)** : Tableau 2D de la taille de l'écran, stockant la couleur finale de chaque pixel. Il est initialisé avec la couleur de fond souhaitée.

Figure 7.12: Initialisation des tampons pour l'algorithme Z-buffer.

7.4.3 Algorithme

Le pseudo-code de l'algorithme Z-buffer est le suivant :

```
// Initialisation
for all pixels (i, j) {
    Depth[i, j] = MAX_DEPTH  // Ou +infini
    Image[i, j] = BACKGROUND_COLOUR
}
// Traitement des polygones
for all polygons P {
    for all pixels (i, j) covered by polygon P {
        Calculate Z_pixel for pixel (i, j) on polygon P // (via interpolation)
        // Convention: Z plus petit est plus proche
        if (Z_pixel < Depth[i, j]) {
```

```
                Calculate C_pixel for pixel (i, j) on polygon P // (couleur,
                    ↪ ombrage)
                Image[i, j] = C_pixel
                Depth[i, j] = Z_pixel
            }
        }
    }
```

Le calcul de la profondeur ‘Z_pixel‘$pour chaque pixel à l'intérieur du polygone projeté se fait généralement par interpolation bilinéa$

7.4.4 Exemple d'Exécution

Considérons le traitement de deux polygones, P1 (violet, Z=5) puis P2 (orange, Z=7), projetés sur l'écran. Le Z-buffer est initialisé à ∞ et le tampon couleur au gris.

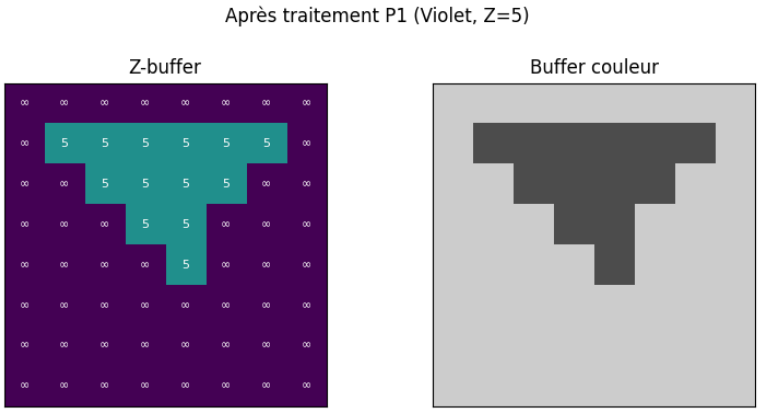

Figure 7.13: État des tampons après traitement du polygone P1 (Violet, Z=5).

Maintenant, traitons le polygone P2 (orange, Z=7).

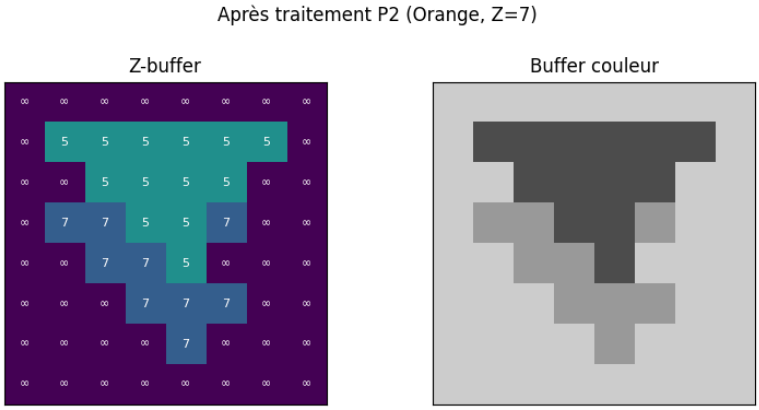

Figure 7.14: État des tampons après traitement du polygone P2 (Orange, Z=7). Notez que P2 n'écrase P1 que là où P2 est devant (jamais dans cet exemple car Z=7 ¿ Z=5) ou là où P1 n'était pas présent.

Si nous avions traité un troisième polygone P3 (disons, bleu, Z=3) qui recouvre certains pixels de P1 et P2, il aurait écrasé P1 et P2 là où il était présent, car sa profondeur Z=3 est inférieure à 5 et 7.

7.4.5 Calcul de la Profondeur par Interpolation

Pour déterminer la profondeur Z_p d'un point $P(x_p, y_p)$ à l'intérieur d'un triangle projeté défini par les sommets $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$, $P_3(x_3, y_3, z_3)$, on utilise généralement l'interpolation bilinéaire. Une méthode courante consiste à interpoler linéairement le long des arêtes, puis à interpoler entre les points interpolés. Par exemple, si P se trouve sur le segment V_sP_3, où V_s est sur le segment P_1P_2, on peut d'abord interpoler Z_s en V_s à partir de Z_1 et Z_2, puis interpoler Z_p en P à partir de Z_s et Z_3. Soit V_s le point d'intersection de la droite horizontale passant par P avec l'arête P_1P_2, et V_t l'intersection avec P_1P_3 (ou P_2P_3). Les profondeurs Z_s et Z_t peuvent être calculées par interpolation linéaire le long des arêtes :

$$Z_s = Z_1 + (Z_2 - Z_1)\frac{y_s - y_1}{y_2 - y_1} \quad \text{(si } V_s \text{ sur } P_1P_2\text{)}$$
$$Z_t = Z_1 + (Z_3 - Z_1)\frac{y_t - y_1}{y_3 - y_1} \quad \text{(si } V_t \text{ sur } P_1P_3\text{)}$$

(Attention aux cas où les dénominateurs sont nuls - arêtes horizontales). Ensuite, la profondeur Z_p au pixel $P(x_p, y_p)$ est interpolée linéairement entre Z_s et Z_t en fonction de la coordonnée x_p :

$$Z_p = Z_s + (Z_t - Z_s)\frac{x_p - x_s}{x_t - x_s}$$

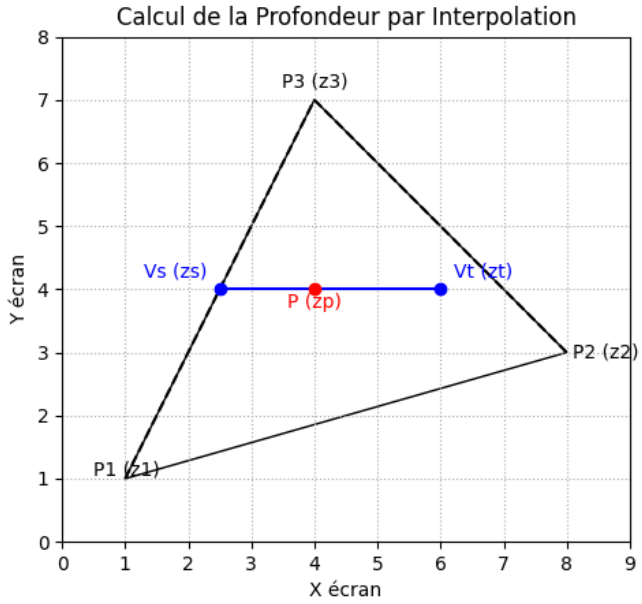

Figure 7.15: Illustration de l'interpolation bilinéaire pour calculer la profondeur Zp d'un pixel P à l'intérieur d'un triangle projeté.

Note : L'interpolation linéaire de Z dans l'espace écran n'est correcte que pour la projection parallèle. Pour la projection perspective, il faut interpoler 1/Z linéairement, puis inverser le résultat pour obtenir Zp.

7.4.6 Avantages et Inconvénients

- **Avantages :**
  - Simplicité de mise en œuvre.

- Pas de tri préalable des polygones nécessaire.
- Fonctionne pour des scènes complexes avec des polygones entrelacés.
- Rapidité : Facilement implémentable en matériel, parallélisable au niveau des pixels.

- **Inconvénients :**
  - Traitement de tous les polygones, même ceux qui seront finalement complètement cachés.
  - Taille mémoire nécessaire pour le Z-buffer (bien que de moins en moins un problème avec la mémoire graphique actuelle). La précision du Z-buffer (nombre de bits) peut être limitante pour des scènes très profondes.
  - Ne traite pas nativement les effets particuliers comme la transparence, les inter-réflexions ou la réfraction (des extensions existent mais complexifient l'algorithme).
  - Peut souffrir d'artefacts de "Z-fighting" si la précision du tampon est insuffisante pour distinguer des surfaces très proches.

7.5 Remplissage de Polygones

Une fois les polygones projetés sur l'écran 2D et leur visibilité déterminée (par ex. via Z-buffer ou Warnock), l'étape suivante est de les "remplir", c'est-à-dire de colorer tous les pixels intérieurs au polygone projeté. Ce processus donne un aspect solide et réaliste aux objets. Le remplissage se fait en utilisant un modèle d'ombrage (shading) pour déterminer la couleur de chaque pixel intérieur, basé sur les propriétés du matériau, les sources de lumière et la géométrie de la surface (par exemple, ombrage plat, Gouraud, Phong, Lambert, etc.). Plusieurs méthodes existent pour déterminer quels pixels sont à l'intérieur d'un polygone projeté et pour les colorer.

7.5.1 Test d'Appartenance d'un Point à un Polygone

Cette méthode consiste à tester, pour chaque pixel potentiellement couvert par le polygone, s'il se trouve à l'intérieur ou à l'extérieur.

Polygones Convexes : Test des Demi-Plans

Pour un polygone convexe, un point P est à l'intérieur si et seulement s'il se trouve du même côté (par exemple, "à gauche" ou "à l'intérieur") de toutes les droites définies par les arêtes du polygone (orientées de manière cohérente, par ex., dans le sens horaire). On peut déterminer la normale "extérieure" $\vec{N}_{i,i+1}$ pour chaque arête P_iP_{i+1}. Un point P est à l'intérieur si le produit scalaire $\vec{N}_{i,i+1} \cdot \vec{P_iP}$ est négatif ou nul pour toutes les arêtes i. S'il existe une arête pour laquelle ce produit scalaire est positif, le point est à l'extérieur.

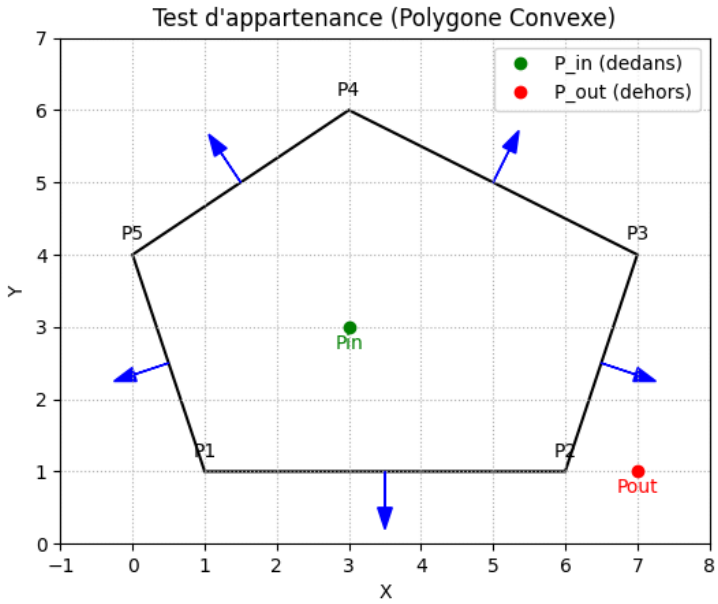

Figure 7.16: Test d'appartenance pour un polygone convexe. Un point est intérieur s'il est du même côté de toutes les arêtes (par ex., produit scalaire négatif avec les normales extérieures).

Cas Général : Test des Angles

Pour un polygone quelconque (convexe ou concave), on peut calculer la somme des angles orientés formés par les vecteurs reliant le point P à chaque paire de sommets consécutifs (P_i, P_{i+1}).

$$\sum_i \alpha_i = \sum_i \text{angle}(\vec{PP_i}, \vec{PP_{i+1}})$$

Si le point P est à l'intérieur du polygone, la somme des angles est $\pm 2\pi$ (ou 360°). Si le point est à l'extérieur, la somme des angles est 0.

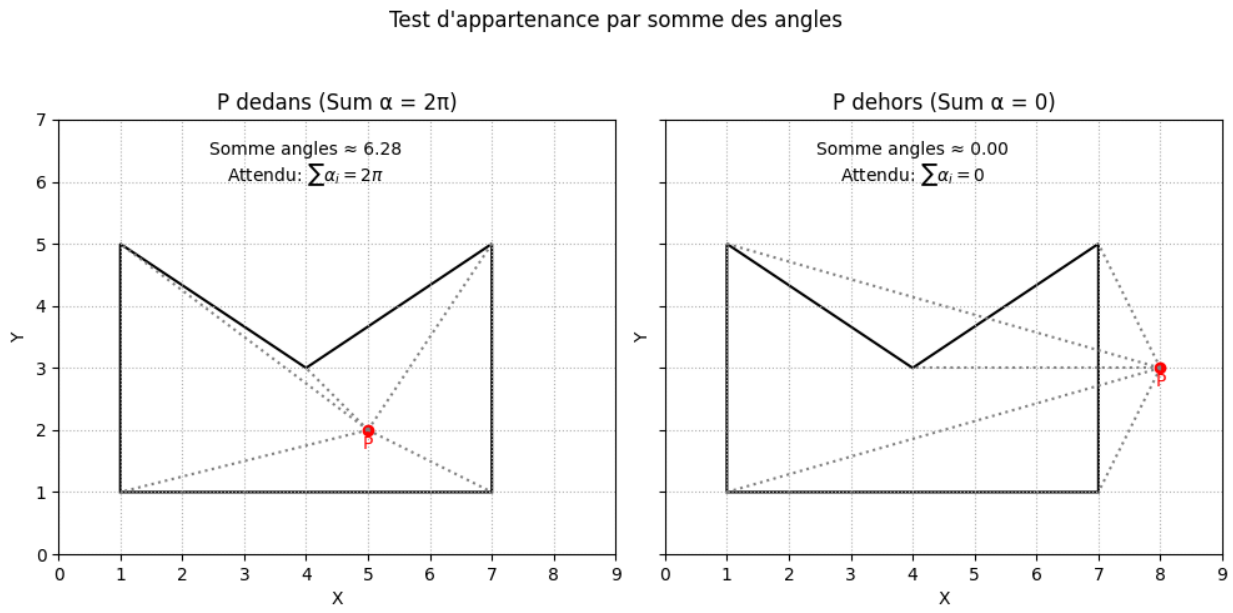

Figure 7.17: Test d'appartenance par la somme des angles. La somme est 2π si le point est intérieur, 0 s'il est extérieur.

Cas Général : Test du Nombre d'Intersections (Ray Casting)

Une méthode très générale consiste à tracer un rayon (une demi-droite) à partir du point P dans une direction quelconque (par exemple, vers $+X$) et à compter le nombre de fois où ce rayon intersecte les arêtes du polygone.

- Si le nombre d'intersections est **impair**, le point P est à l'intérieur.
- Si le nombre d'intersections est **pair** (y compris zéro), le point P est à l'extérieur.

Il faut traiter attentivement les cas limites :

- Le rayon passe par un sommet : Compter l'intersection uniquement si les deux arêtes adjacentes au sommet sont de part et d'autre du rayon.
- Le rayon chevauche une arête horizontale : Ne pas compter ces intersections ou utiliser une règle cohérente.
- Le point P est sur une arête : Il est considéré comme intérieur ou extérieur selon la convention choisie.

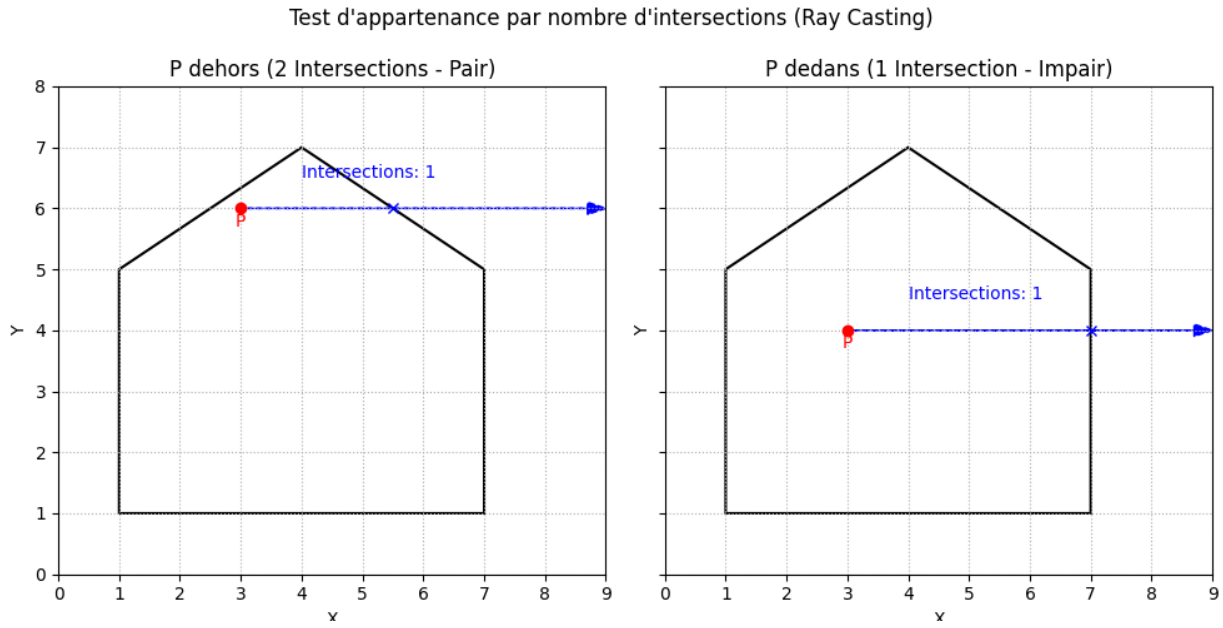

Figure 7.18: Test d'appartenance par comptage d'intersections. Un nombre impair d'intersections signifie que le point est intérieur, un nombre pair signifie qu'il est extérieur. (Note: les cas limites doivent être gérés avec soin).

Identification d'un Polygone

Pour certains algorithmes, il est utile de savoir si un polygone est convexe ou concave, ou de connaître l'orientation des arêtes. On peut utiliser le produit vectoriel (ou produit mixte en 2D) pour cela. Pour trois sommets consécutifs P_1, P_2, P_3, le signe de $(P_2 - P_1) \times (P_3 - P_2)$ indique le sens de la rotation (gauche ou droite). Si tous les virages sont dans le même sens, le polygone est convexe. Le produit vectoriel entre deux vecteurs $p1 = (p1_x, p1_y)$ et $p2 = (p2_x, p2_y)$ en 2D est $p1_x p2_y - p1_y p2_x$.

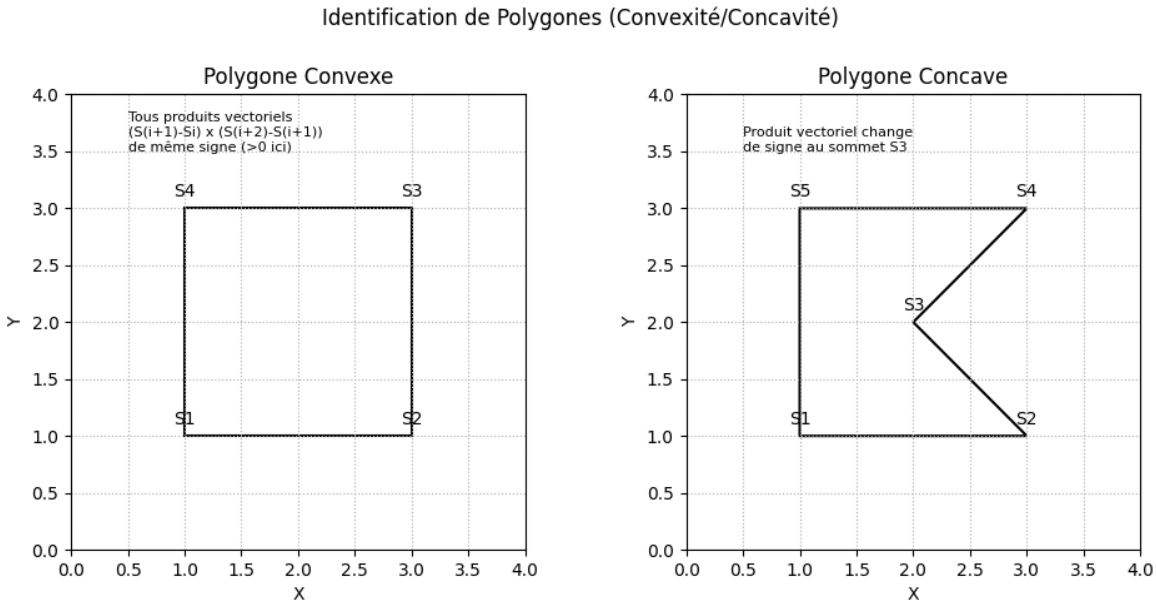

Figure 7.19: Utilisation du produit vectoriel pour déterminer la convexité (signe constant) ou la concavité (changement de signe) d'un polygone.

7.5.2 Algorithme de Balayage de Lignes (Scan-Line Filling)

Cette méthode est très efficace pour le remplissage de polygones. Elle combine la détermination des pixels intérieurs et leur coloriage.

- On balaie l'image horizontalement, une ligne de pixels (scan-line) à la fois, typiquement de bas en haut.
- Pour chaque ligne de balayage qui intersecte le polygone :
  1. Trouver toutes les intersections de la ligne de balayage avec les arêtes du polygone.
  2. Trier ces intersections par coordonnée X croissante.
  3. Remplir les pixels entre les paires successives d'intersections (1ère et 2ème, 3ème et 4ème, etc.). C'est la règle de parité : on est à l'intérieur du polygone entre une intersection impaire et une intersection paire.
- Pour optimiser la recherche des intersections, on utilise souvent une structure de données appelée "Table des Arêtes Actives" (Active Edge Table - AET). L'AET contient les arêtes qui sont intersectées par la ligne de balayage courante. Pour chaque arête dans l'AET, on stocke des informations utiles comme Y_{max} (coordonnée Y maximale de l'arête), X_{min} (coordonnée X de l'intersection avec la ligne courante), et dx/dy (l'inverse de la pente, pour mettre à jour X_{min} efficacement lorsque l'on passe à la ligne suivante $y + 1$).
- La liste des intersections pour la ligne courante est obtenue à partir des X_{min} des arêtes dans l'AET. Après avoir rempli les segments de la ligne, on met à jour l'AET : on supprime les arêtes dont Y_{max} est atteint, on ajoute les nouvelles arêtes commençant à $y + 1$, et on met à jour X_{min} pour les arêtes restantes ($X_{min} = X_{min} + dx/dy$).

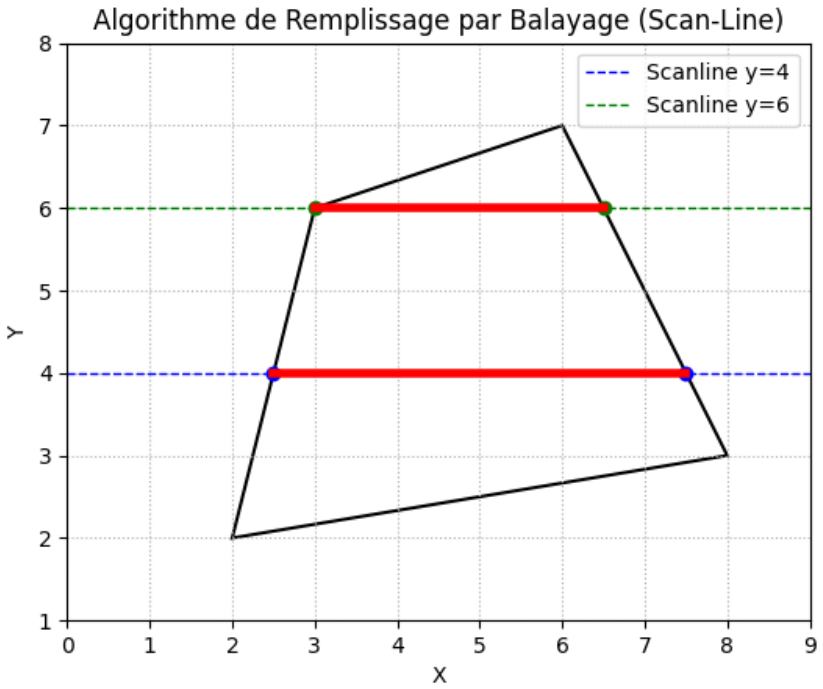

Figure 7.20: Principe du remplissage par balayage. Pour chaque ligne (ex: y=4, y=6), les intersections avec les arêtes sont trouvées, triées, puis les segments entre paires d'intersections sont remplis.

La gestion des détails, comme les sommets horizontaux, les sommets qui sont des extrema locaux en Y, et la mise à jour de l'AET, est essentielle pour une implémentation correcte.

Gestion des Conflits Frontaliers

Lors de l'utilisation d'algorithmes de traçage de segments (comme Bresenham) pour approximer les arêtes et trouver les intersections X_{min}, des erreurs d'arrondi peuvent survenir. Cela peut conduire à des points d'intersection calculés légèrement à l'extérieur du polygone réel ou à des incohérences entre arêtes partageant un sommet. Pour éviter les chevauchements ou les trous entre polygones adjacents, on peut adopter des conventions strictes : par exemple, toujours prendre les points intérieurs au polygone, ou utiliser les informations de l'équation de la droite (Y_{max}, X_{min}, dx/dy) pour une détermination plus précise des appartenances.

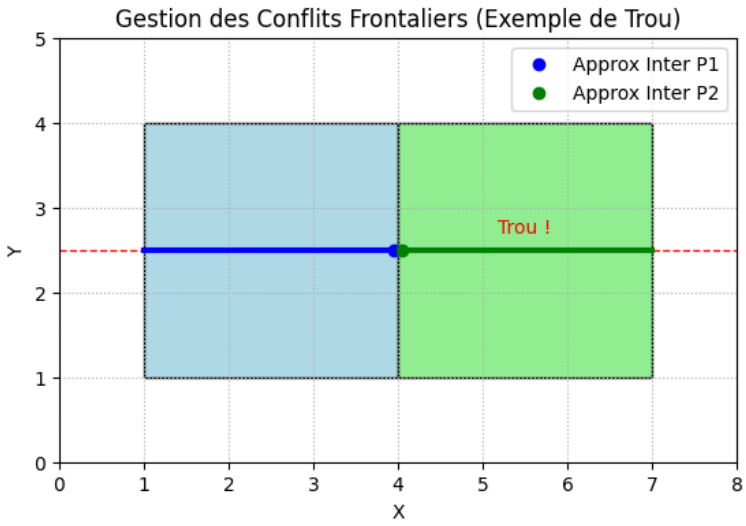

Figure 7.21: Exemple de conflit frontalier dû à l'approximation. Les intersections calculées pour la même arête partagée peuvent différer légèrement, créant des trous ou des chevauchements si non gérées correctement.

7.5.3 Remplissage de Régions (Méthode du Germe - Seed Fill)

Cette approche est différente du balayage. Elle fonctionne sur des régions définies par une couleur de frontière explicite dans le tampon d'image (framebuffer).

- On commence avec un pixel "germe" (seed) dont on sait qu'il est à l'intérieur de la région à remplir.
- L'algorithme propage récursivement (ou itérativement avec une pile) la couleur de remplissage aux pixels voisins, tant qu'ils ne sont pas de la couleur de la frontière (ou déjà remplis avec la couleur de remplissage).
- Une version efficace (Scanline Seed Fill) optimise en remplissant des segments horizontaux (spans) de pixels contigus de même couleur initiale, puis en cherchant des pixels germes pour les lignes adjacentes (au-dessus et au-dessous) uniquement au début et à la fin de ces segments.

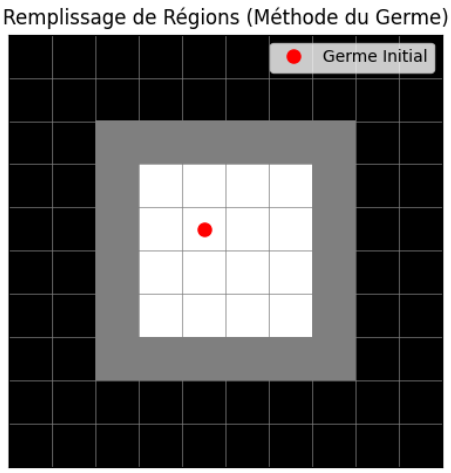

Figure 7.22: Principe du remplissage par germe. À partir d'un pixel intérieur (germe), la couleur se propage aux voisins jusqu'à rencontrer la frontière.

Cette méthode est utile lorsque les régions sont définies par des frontières de pixels plutôt que par des définitions géométriques de polygones.

7.6 Coûts Comparés des Algorithmes

Les performances des différents algorithmes d'élimination des parties cachées dépendent fortement de la complexité de la scène (nombre de polygones, profondeur, etc.). Le graphique suivant donne une idée qualitative des coûts relatifs.

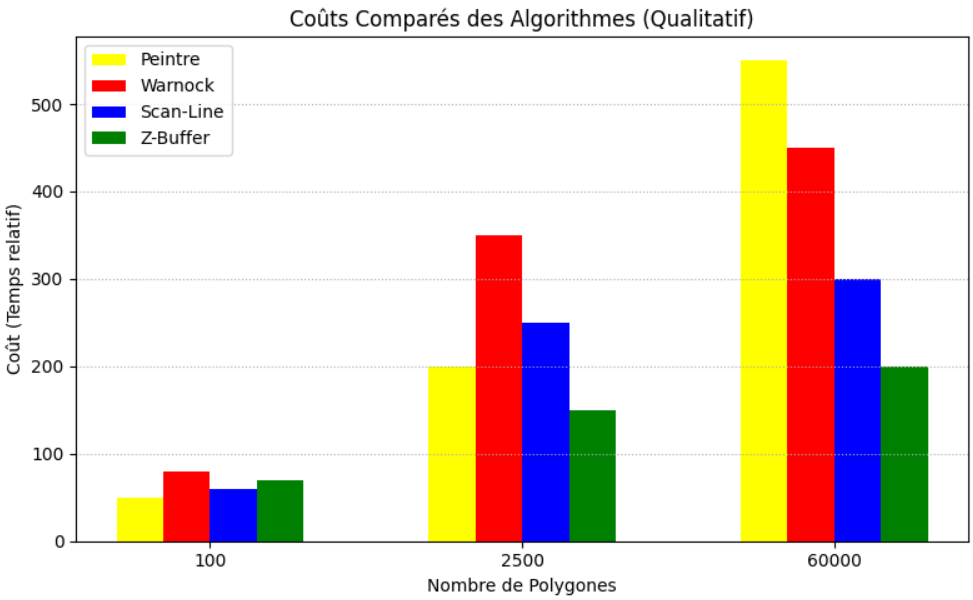

Figure 7.23: Comparaison qualitative des coûts (temps d'exécution) des algorithmes en fonction du nombre de polygones dans la scène. (Adapté de l'image 95).

On observe que le Z-buffer a tendance à avoir un coût qui augmente moins rapidement avec le nombre de polygones que d'autres méthodes comme l'algorithme du peintre ou Warnock, surtout pour des scènes complexes. Scan-Line se situe souvent entre les deux. Cependant, ces coûts dépendent aussi fortement de l'implémentation (matérielle vs logicielle) et des caractéristiques spécifiques de la scène.

7.7 Choix de l'Algorithme

Le choix de l'algorithme idéal pour l'élimination des parties cachées et le remplissage dépend de plusieurs facteurs :

- **Complexité de la scène :** Nombre de polygones, profondeur moyenne, distribution spatiale.
- **Matériel disponible :** Présence d'accélération matérielle (GPU) pour certains algorithmes (notamment Z-buffer).
- **Effets souhaités :** Besoin de gérer la transparence, les réflexions, etc., qui peuvent être plus faciles à intégrer dans certains algorithmes.
- **Pré-traitements :** Certains algorithmes nécessitent des tris préalables (Peintre) ou des structures de données complexes (Warnock, Scan-Line AET).
- **Coût mémoire :** Compromis entre la mémoire nécessaire (ex: Z-buffer) et le temps de calcul.

7.7.1 Scénarios d'Utilisation

- **Z-Buffer :**
  - **Usage général :** C'est l'algorithme le plus couramment utilisé aujourd'hui, surtout grâce à son implémentation matérielle généralisée dans les GPU.

  - **Avantages :** Simple à implémenter (si logiciellement), pas de pré-traitement complexe, performance relativement indépendante de l'ordre des polygones, fourni "gratuitement" par la librairie graphique / le matériel.
  - **Inconvénients :** Peut nécessiter beaucoup de mémoire (moins un problème aujourd'hui), moins élégant pour gérer certains effets (transparence), peut avoir des problèmes de précision (Z-fighting).

- **Scan-Line :**
  - **Usage spécifique ("pour les hackers") :** Peut être très efficace si implémenté soigneusement en logiciel, surtout lorsque l'accès direct au matériel graphique n'est pas possible ou souhaité.
  - **Avantages :** Faible coût mémoire (ne stocke que les informations de la ligne courante), cohérence spatiale exploitée (les informations d'une ligne aident pour la suivante), peut être très efficace car lié directement à la fonction d'affichage ligne par ligne.
  - **Inconvénients :** Plus complexe à implémenter correctement (gestion de l'AET, cas particuliers), moins facilement parallélisable que le Z-buffer au niveau pixel.

- **Warnock :**
  - **Usage historique/pédagogique :** Intéressant pour son approche "diviser pour régner".
  - **Avantages :** Peut être rapide pour des scènes simples ou avec de grandes zones uniformes.
  - **Inconvénients :** La subdivision peut devenir coûteuse pour des scènes complexes, la gestion des polygones aux frontières des quadrants ajoute de la complexité. Moins utilisé dans les systèmes modernes comparé au Z-buffer.

En pratique, l'algorithme Z-buffer domine largement dans le rendu en temps réel grâce à son efficacité et son intégration matérielle. Les algorithmes de type Scan-Line peuvent encore trouver leur place dans des contextes spécifiques ou pour des rendus logiciels optimisés.