ÉLIMINATION DES PARTIES CACHÉES

Q. QUADRAT

TABLE DES MATIÈRES

1. INTRODUCTION

L'un des problèmes importants lors d'une projection d'une scène graphique, qu'elle soit en 2D ou 3D, est d'éliminer les parties cachées de la scène. Cet exposé va présenter deux algorithmes pour résoudre ce problème. D'une part, l'algorithme de z-buffer qui affiche les pixels les plus proches de l'observateur dans une projection parallèle, d'autre part, un algorithme de partitionnement binaire de l'espace, qui permet d'afficher les objets les plus éloignés avant d'afficher les plus proches.

2. Scène 3D

Une scène 3D notée $\mathcal{S}$, est définie par un ensemble d'objets $\{\mathcal{O}_1, \mathcal{O}_2, \cdots, \mathcal{O}_n\}$ dans l'espace à trois dimensions. La surface extérieure d'un objet $\mathcal{O}$, appelée bord, est notée $\partial\mathcal{O}$. On se limite à des scènes dont les objets sont polyèdraux. Le bord d'un polyèdre est donc une union de faces qui sont des polygones :

$$\partial\mathcal{O} = \{\partial\mathcal{O}^1, \partial\mathcal{O}^2, \cdots, \partial\mathcal{O}^n\}.$$

Un polygone peut être triangulé c'est à dire que chaque polygone est une union de triangles :

$$\partial\mathcal{O}_0^k = \{t_0^{k,1}, \cdots, t_0^{k,l}\}.$$

En fin de compte pour visualiser une scène nous avons besoin de représenter uniquement les bords des objets. L'ensemble des bords des objets d'une scène sera appele bord de la scène $\partial\mathcal{S}$ qui sera qu'un ensemble de triangles dans l'espace 3D :

$$\partial\mathcal{S} = \{t_0^{k,l}\}.$$

Un triangle, est une surface colorée, défini par ses trois sommets :

$$t = (p_1, p_2, p_3),$$

où chaque sommet est défini par le quadruplet : $p = (x, y, z, c)$, où x, y, z sont les coordonnées du sommet et c sa couleur donc un triangle est défini par une matrice de quatres lignes et trois colonnes. On peut ainsi représenter une scène par un ensemble de telles matrices.

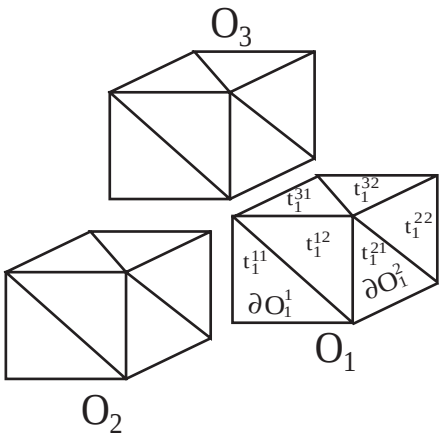

Fig. 1 – Exemple de scène 3D

De même on aura besoin de scènes 2D polygonales. Dans ce cas, chaque objet est un polygone. Le bord d'un objet est une union de faces où chaque face est un segment. Un segment est défini par ses deux extrémités donc une matrice 2 colonnes 3 lignes.

3. Rappel de géométrie et introduction au calcul matriciel

3.1. **Calcul matriciel.** Les matrices sont des tableaux de nombres réels. On dira que la matrice est $m \times n$ si elle a m lignes et n colonnes. Sur des tableaux de tailles compatibles, on définit une addition et une multiplication [FOLE95].

– *Addition matricielle* : Soit A et B deux matrices $m \times n$, la somme des matrices A et B est la matrice C $m \times n$ définie par :

$$C_{ij} = A_{ij} + B_{ij}, \quad i = 1 \cdots m, \ j = 1 \cdots n.$$

– *Multiplication matricielle* : Soit A une matrice $m \times n$ et B une matrice $n \times p$, le produit des matrices A et B est la matrice C $m \times n$ définie par :

$$C_{ik} = \sum_{j=1}^{n} A_{ij} B_{jk}, \quad i = 1 \cdots m, \ k = 1 \cdots p.$$

– *Multiplication par un scalaire* : Soit A une matrice $m \times n$ et λ un nombre réels, le produit de $C = A\lambda = \lambda A$ est défini par :

$$C_{ij} = \lambda A_{ij}, \quad i = 1 \cdots m, \ j = 1 \cdots n.$$

Remarque 1. *Le calcul matriciel a de bonnes propriétés. En particulier les matrices $m \times m$ munies de ces deux opérations a une structure d'anneau.*

3.2. **Géométrie 2D.** On considère un plan muni de son système de coordonnées (O, x, y). Un point P est défini par ses coordonnés (x, y) que l'on notera matriciellement :

$$P = \begin{bmatrix} x \\ y \end{bmatrix}.$$

On définit sur ces points les transformations géometriques suivantes qui à P (x, y) associe P' (x', y') :

– *Translation* : la translation de vecteur T est la transformation définie par $P' = P + T$.

– *Rotation* : la rotation d'angle θ est la transformation définie par :

$$x' = x \cos\theta - y \sin\theta, \qquad y' = x \sin\theta + y \cos\theta.$$

Sous forme matricielle nous avons : $P' = RP$ avec

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

– *Homothétie* : l'homothétie de rapport h est la transformation définie par :

$$x' = hx, \qquad y' = hy.$$

Sous forme matricielle nous avons : $P' = hP$.

– *Perspective* : la perspective V de centre O et de plan de projection $\mathcal{P}$ (d'équation $y = 1$) est définie par $x' = x/y, \quad y' = 1)$. P' est l'intersection de OP avec $\mathcal{P}$. Elle est définie lorsque $P \neq 0$. Elle ne s'exprimera en terme de calcul matriciel que dans le cadre de la géométrie projective [FOLE95].

3.3. **Géométrie projective 2D.** Pour unifier les problèmes apparaissant en géométries, on introduit la géométrie projective. On se place dans un espace à trois dimensions. Au lieu de raisonner avec les points d'un espace 2D, on raisonne avec les *rayons* (droites passant par l'origine) d'un espace 3D. On peut voir alors la géométrie 2D comme la géométrie des intersections des rayons avec un plan. Un rayon est la classes de points proportionnels entre eux. Deux points P de coordonnées (X, Y, Z) et P' de coordonnées (X', Y', Z') sont équivalents ($P \sim P'$) si seulement si $P' = \lambda P$ où λ est un nombre réel. Lorsque $Z \neq 0$ un représentant d'un rayon est $(x = X/Z, y = Y/Z, 1)$. C'est l'intersection du rayon avec le plan $Z = 1$. Les points $Q = (x, y, 1)$ du plan $Z = 1$ sont les points de la géométrie 2D du paragraphe précédant. En géométrie projective les transformations du paragraphe précédant s'écrivent comme des produits matriciels 3×3. (Figure 2) [FOLE95].

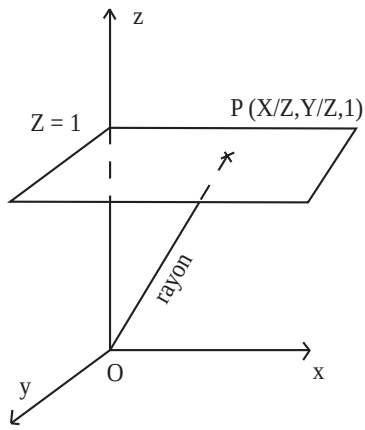

FIG. 2 – Géométrie projective 2D

Aux quatre opérations vues précédemment sont respectivement associées une matrice : la translation T, la rotation R, l'homothétie H et la perspective V.

$$T = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$
$$H = \begin{bmatrix} h & 0 & 0 \\ 0 & h & 0 \\ 0 & 0 & h \end{bmatrix}, \quad V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Vérifions que la dernière opération réalise bien la perspective :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ y \end{bmatrix} \sim \begin{bmatrix} x/y \\ 1 \\ 1 \end{bmatrix}.$$

3.4. **Géométrie projective 3D.** De nouveau, on définit une géométrie 3D projective en considérant les rayons dans un espace 4D. Les coordonnées homogènes seront les quadruplets (X, Y, Z, T) et on peut associer àux opérations une matrice. Sont associés par une matrice : la translation de vecteur (T_x, T_y, T_z), la rotation d'angle θ et d'axe z, l'homothétie de rapport h.

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad H = \begin{bmatrix} h & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & h & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Nous nous interessons à deux sortes de perspective dans un espace 3D :

– *projection perspective* de centre O sur $Z = 1$ (figure 2) :

$$V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

– *projection parallèle* à (Oz) sur le plan $Z = 1$ (figure 3) :

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

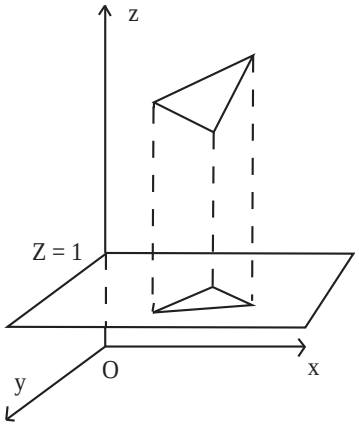

FIG. 3 – perspective parallèle

Remarque 2. *Les transformations géométriques définies par des matrices conservent les alignements. C'est à dire une droite est transformée en une droite, un plan en un plan et l'ordre des points sur une droite est préservé.*

Il est plus façile de faire une projection parallèle qu'une projection perspective. On peut ramener une projection perspective à une projection parallèle en utilisant la transformation géométrique définie par la matrice suivante :

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1-m} & \frac{-m}{1-m} \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ avec } 1 > m > 0.$$

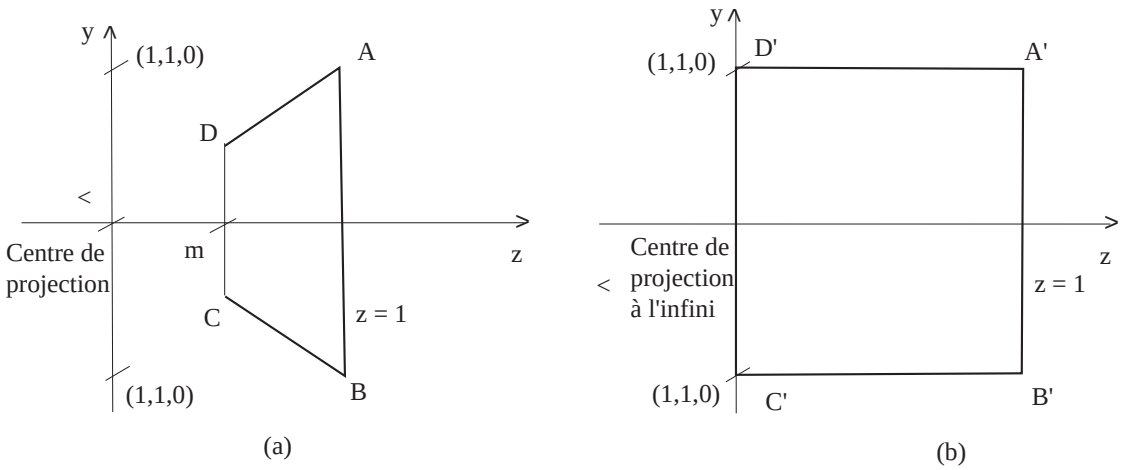

FIG. 4 – Transformation du volume canonique par la matrice M

L'inverse de cette matrice est :

$$M^{-1} = \frac{1-m}{m}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{m}{1-m} \\ 0 & 0 & -1 & \frac{1}{1-m} \end{bmatrix} \text{ avec } 1 > m > 0.$$

Dans la figure 4 on montre les vues de côté du volume canonique avant (a) et aprés (b) l'application de la matrice M (voir [FOLE95]).

Cette transformation a les propriétés suivantes :

– Les points du plan $z = 1$ ne sont pas modifiés. Pour preuve :

$$M\begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix}.$$

– Le plan $z = m$ est envoyé sur le plan $z = 0$:

$$M\begin{bmatrix} x \\ y \\ m \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ m \end{bmatrix}.$$

– L'origine O est envoyé à l'infini dans la direction $x = 0,\ y = 0$:

$$M\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{-m}{1-m} \\ 0 \end{bmatrix}.$$

C'est mathématiquement incorrect, mais si on divise tous les éléments par 0, le centre de projection va à l'infini.

On peut résumer ces transformations par la figure 5. On vérifie la formule

$$V = M^{-1}WM = WM.$$

Les deux méthodes suivantes sont équivalentes :

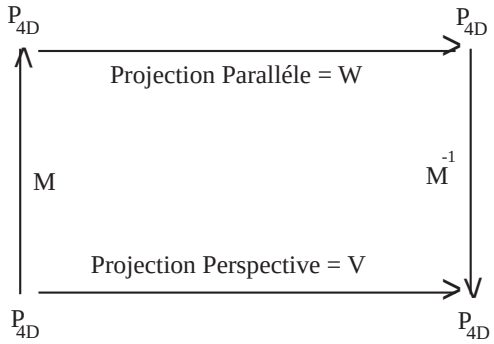

FIG. 5 – Schéma

– éliminer les surfaces cachées avec l'œil à l'origine et faire la perspective de centre 0,
– transformer la scène par M, faire l'élimination pour la projection parallèle et faire la projection parallèle voir [FOLE95].

4. L'ALGORITHME DE PARTITIONNEMENT BINAIRE DE L'ESPACE

4.1. **Partition binaire d'un ensemble.** Un *graphe orienté* $\mathcal{G} = (N, A)$ est défini comme un ensemble fini de *noeuds* N et d'*arcs* A, où un arc a est une paire ordonnée de noeuds (n_1, n_2). On dira que n_1 est *prédécesseur* de n_2 et que n_2 est un *successeur* de n_1.

Un *arbre* est un graphe qui posséde un unique noeud n'ayant aucun prédécesseur, appelé *racine*, et tel que tout noeud autre que la racine a un prédécésseur unique.

Soit un ensemble A on appelle *coupure* une application α qui à A associe deux ensembles non vides A^+, A^- tels que $A = A_\alpha^+ \cup A_\alpha^-$.

Une partition binaire $\mathcal{P}$ d'un ensemble A est une partition obtenue par une suite récursive de coupures. A est coupé éventuellement par α. A_α^+ est éventuellement coupée par β ($A_\alpha^+ = A_{\alpha^+\beta^+} \cup A_{\alpha^+\beta^-}$) et A_α^- est éventuellement coupé par $\gamma \cdots$ (figure 6).

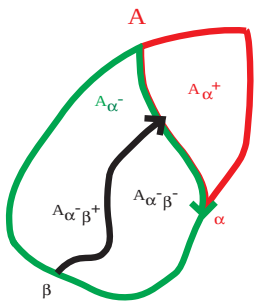

FIG. 6 – Exemple de coupures : elles ne sont pas forcement droites !

Dans un espace à n dimensions. Les coupures sont des hyperplans (voir [faqs.org], [geocities.com], [cs.wpi.edu]). c'est à dire des sous espaces à $n - 1$ dimensions. Par exemple dans un espace 3D l'hyperplan est un plan et pour un espace 2D, c'est une droite.

4.2. **Représentation d'une scène 2D par un BSP.** Une scène 2D polygonales est définie par les segments des bords des objets qui la composent. Pour construire l'arbre BSP associé à la scène, on utilise, comme coupure, les droites définies par les faces des objets. Chaque noeud de l'arbre est associé à une coupure définie par un segment orienté et représenté par une matrice 3×2 contenant les 2 coordonnées et la couleur des extrémités du segment. Chaque coupure coupe l'espace en deux. Le demi-espace positif est celui qui contient la normale au segment orienté faisant un angle de $\pi/2$ avec celui-ci. L'autre demi-espace sera dit négatif.

Soit une scène $\mathcal{S}$ (figure 7) constituée deux objets : un triangle et un rectangle. Orientons toutes les faces (flèches noires). On définit, ici, une scène comme un ensemble de segments.

Les segments constituant ces deux figures appartiennent aux coupures respectives : $\alpha, \delta, \varepsilon, \gamma, \xi, \beta, \nu$. Par exemple prenons α l'hypothénuse du triangle comme premier segment. Par la suite, nous ne distingueront plus coupures et segments : ils porteront le même nom. Donc $\mathcal{S} = (\alpha, \delta, \varepsilon, \gamma, \xi, \beta, \nu)$. Coupons $\mathcal{S}$ par α, notre scène est à présent formée de deux demi-espaces : A_{α^+} et A_{α^-}. La construction

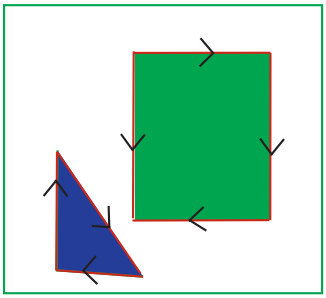

FIG. 7 – Scène orientée

d'un arbre se fera selon le principe suivant. Si α est la racine de l'arbre alors A_{α^+} sera la branche gauche et A_{α^-} la branche droite. On va donc former un arbre dont les noeuds sont les faces des objets de la scène et les branches des ensembles. Les feuilles sont les ensembles finaux de la partition.

Dans notre exemple, nous aurons donc $A_{\alpha^-} = (\beta, \nu)$ et $A_{\alpha^-} = (\delta, \varepsilon, \gamma, \xi)$. Pour savoir par exemple si $\beta \in A_{\alpha^-}$, voir annexe 1 sous-section 1.

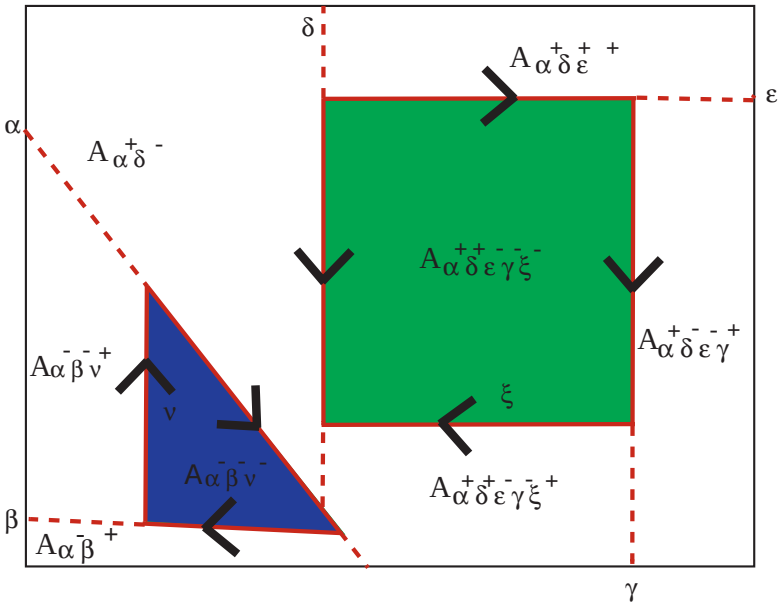

FIG. 8 – Partitionnement d'une scène 2D

L'algorithme de construction de l'arbre sera [faqs.org], [geocities.com], [cs.wpi.edu] :

$$\mathrm{bsp}(\mathcal{S}) = \begin{cases} \bullet & \text{si } \mathcal{S} = \mathrm{nil}\ , \\ \mathcal{S} & \text{si } \ \mathrm{card}(\mathcal{S}) = 1\ , \\ (\text{tête}(\mathcal{S})\ \mathrm{bsp}(\mathcal{S}^+)\ \mathrm{bsp}(\mathcal{S}^-)) & \text{sinon}\ . \end{cases}$$

Avec $\mathcal{S}^+$ la scène gauche et $\mathcal{S}^-$ la scène droite par rapport à une coupure.

En développant la récursion sur la scène de l'exemple, on obtient :

$$\begin{aligned}(\alpha\ \delta\ \varepsilon\ \gamma\ \xi\ \beta\ \nu\ ) &= (\alpha\ (\delta\ \varepsilon\ \gamma\ \xi\ )\ (\beta\ \nu\ )\ )\\ &= (\alpha\ (\delta\ (\varepsilon\ \gamma\ \xi\ )\ \bullet\ )\ (\beta\ \bullet\ \nu\ )\ )\\ &= (\alpha\ (\delta\ (\varepsilon\ \bullet (\gamma\ \xi\ )\ \bullet\ )\ (\beta\ \bullet\ \nu)\ )\\ &= (\alpha\ (\delta\ (\varepsilon\ \bullet\ (\gamma\ \bullet \xi\ )\ )\ \bullet\ )\ (\beta\ \bullet\ \nu)\ ).\end{aligned}$$

Ce qui correspond à l'arbre de la figure 9.

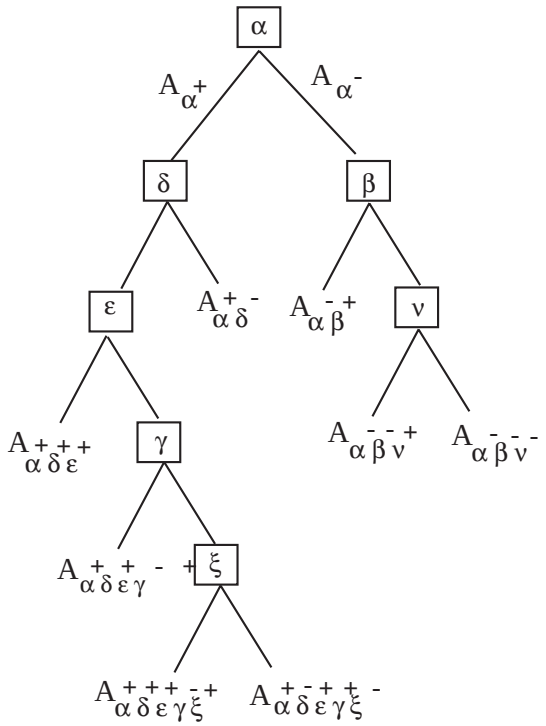

FIG. 9 – Arbre BSP final

4.3. **Algorithme du peintre.** Nous appelons *algorithme du peintre* un algorithme récursif de parcourt un arbre, qui affiche les objets des plus éloignés au plus prés de l'oeil, ce qui permetrra ne de pas voir les objets normalement cachés [faqs.org].

Soit α une coupure et A un ensemble représenté par un arbre binaire tel que : $A = (\alpha, A_{\alpha^-}, A_{\alpha^+})$ et P la position de l'oeil, où $P \in A$. La fonction peintre est définie récursivement par :

$$\text{peintre}\ (A, P) = \begin{cases} \text{peintre}\ (A_{\alpha^-}, P), \text{afficher}\ \alpha, \text{peintre}\ (A_{\alpha^+}, P) & \text{si } P \in A_{\alpha^+}, \\ \text{peintre}\ (A_{\alpha^+}, P), \text{afficher}\ \alpha, \text{peintre}\ (A_{\alpha^-}, P) & \text{si } P \in A_{\alpha^-}, \\ \text{peintre}\ (nil, P) = \bullet & \text{si } A = nil. \end{cases}$$

Pour savoir si $P \in A_{\alpha^+}$ ou $P \in A_{\alpha^-}$, se reporter à l'annexe 1, sous-section 1.

Dans notre exemple l'oeil P est située dans l'espace $A_{\alpha^+\delta^+\varepsilon^+}$. Nous allons appliquer peintre(α, P). Comme $P \in \alpha^+$, nous affichons en premier, la branche A_{α^-} puis α puis A_{α^+}. Dans la brache A_{α^-} nous réapliquons l'algorithme du peintre.

Comme β est plus éloigné que ν et comme on affiche pas les feuilles, on va donc afficher en premier β, vient ensuite ν puis α. Dans la branche A_{α^+}, comme $P \in \delta^+$ on affiche en premier δ. On arrive à la branche ε, on applique peintre(ε, P) c'est à dire on va afficher la branche $A_{\alpha^+\delta^+\epsilon^-}$ puis ε. Comme ξ est plus éloigné que γ on affiche en premier ξ puis γ et enfin ε. En conclusion, nous allons afficher dans l'orde : β, ν, α, δ, ξ, γ, ε. (Figure 10).

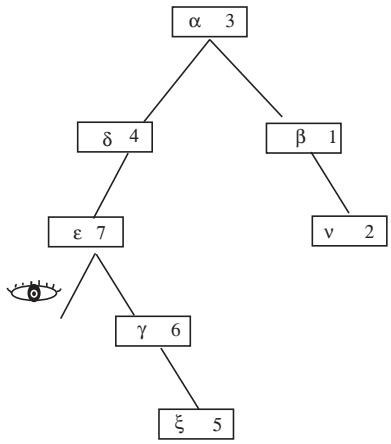

FIG. 10 – Ordre de parcours

Pour simplifier l'écriture, nous noterons α pour peintre(α, P) et $\dot{\alpha}$ pour afficher α dans le développement de récursion du peintre, on obtient ainsi :

$$\begin{aligned}
\alpha &= \alpha^- \dot{\alpha} \alpha^+ \\
&= \cancel{\beta}^+ \dot{\beta} \beta^- \dot{\alpha} \alpha^+ \\
&= \dot{\beta} \cancel{\nu}^+ \dot{\nu} \cancel{\nu}^- \dot{\alpha} \alpha^+ \\
&= \dot{\beta} \dot{\nu} \dot{\alpha} \cancel{\delta}^- \dot{\delta} \delta^+ \\
&= \dot{\beta} \dot{\nu} \dot{\alpha} \dot{\delta} \varepsilon^- \dot{\varepsilon} \cancel{\varepsilon}^+ \\
&= \dot{\beta} \dot{\nu} \dot{\alpha} \dot{\delta} \gamma^- \dot{\gamma} \cancel{\gamma}^+ \dot{\varepsilon} \\
&= \dot{\beta} \dot{\nu} \dot{\alpha} \dot{\delta} \cancel{\xi}^+ \dot{\xi} \cancel{\xi}^- \dot{\gamma} \dot{\varepsilon} \\
&= \dot{\beta} \dot{\nu} \dot{\alpha} \dot{\delta} \dot{\xi} \dot{\gamma} \dot{\varepsilon}
\end{aligned}$$

Nous obtenons ainsi le parours de l'arbre donné dans la figure 10.

Ainsi, nous venons d'afficher tous les objets des plus "éloignés" aux plus "proches". Nous avons ainsi éliminé toutes les parties cachées. Il ne reste plus qu'à effectuer les projections perspectives des faces lorsqu'on affiche une face de la scène pour pouvoir la visualiser sur l'écran (figure 11).

La construction de l'arbre BSP associé à une scène 3D est analogue à la construction 2D. Les coupures sont, maintenant, les plans définis par les faces des objets. L'algorithme du peintre reste identique.

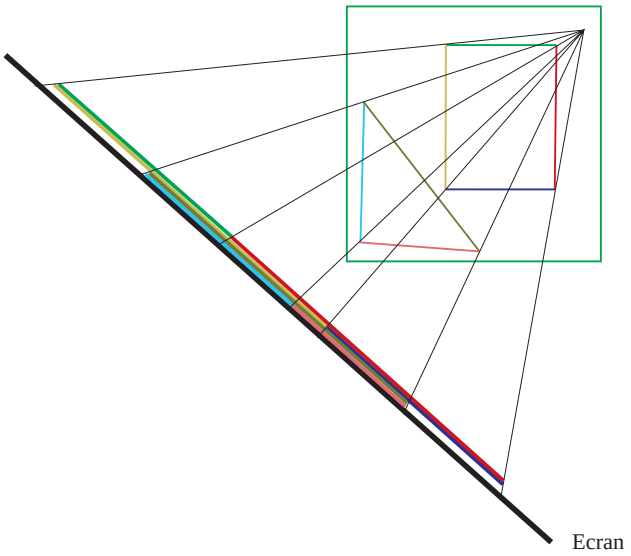

FIG. 11 – Projection sur l'écran

5. CONCLUSION

5.1. **BSP.** L'arbre BSP est un arbre où les noeuds représentent les polygones d'une scène 2D ou 3D.

L'arbre BSP est utilisé, pour la gestion dans un moteur 3D, d'un monde, constitué de nombreux polygones et en grande partie statique (Quake, Doom, Flight Simulator etc...). Il permet un ordre d'affichage des polygones éliminant les parties cachées évitant ainsi l'utilisation de z-buffer.

Un arbre BSP a plusieurs avantages :

- L'arbre donne à peu de frais un ordre exact d'affichage des polygones grâce à l'algorithme du peintre.
- L'arbre permet de sélectionner efficacement les faces visibles dans le cône de vision de la caméra.
- L'arbre permet d'effectuer des tests de collisions sur un nombre minimal de polygones.

Mais il a aussi quelques inconvénients :

- La construction de l'arbre engendre un grand nombre de polygones. En effet, de nombreux polygones vont apparaître à la suite de découpage.
- Une fois construit, l'arbre ne peut pas (ou très difficilement) être modifié.
- Un arbre BSP est dur à débuguer.

5.2. **Zbuffer.** Algorithme standard pour l'obtention rapide d'images de qualité moyenne. Application à la conception et fabrication assistée par ordinateur (station de travail graphique), à la réalité virtuelle et au jeu.

Les avantages du z-buffer sont :

- L'algorithme est facilement implantable au niveau logiciel et matériel.
- Il peut être optimisé en utilisant exclusivement des entiers, ouen utilisant des variantes de l'algorithme de Bresenham pour le tracé de segments.

– L'algorithme est facilement pipelinable et parallélisable.
– Gestion simple des recoupements entre objets.
– Exécution en un temps en $o(n)$ du nombre de facettes donc une bonne scalabilité.

Les inconvénients du Z-buffer sont :
– Les deux zones tampon peuvent avoir des tailles très importantes (plusieurs Mo dans le cas de grands écrans).
– La décomposition de chaque surface élémentaire en pixels nécessite une puissance de calcul importante (de l'ordre de 50 Mips pour 100000 facettes/s) et une vitesse d'accés mémoire très élevée.
– On n'a pas de gestion intrinsèque des réflexions et des transmissions.
– Des difficultés existent pour implanter la modélisation de phénomnes optiques complexes.

6. Annexes

6.1. **Détermination de l'appartenance d'un point à un demi-espace.** On rappelle que le *produit scalaire* de deux vecteurs $\overrightarrow{v_1}(a_1, b_1)$ et $\overrightarrow{v_2}(a_2, b_2)$, est la constante

$$\overrightarrow{v_1}.\overrightarrow{v_2} = a_1a_2 + b_1b_2.$$

Soit $M(a_1, b_1)$ et $N(a_2, b_2)$ deux points. Ces deux points définissent le vecteur $\overrightarrow{MN} = (a_2 - a_1, b_2 - b_1)$. Le vecteur normal $\overrightarrow{n}$ de $\overrightarrow{MN}$ est :

$$\overrightarrow{n} = (b_1 - b_2, a_2 - a_1).$$

Tous les points $P(x, y)$ appartenant à la droite $\alpha = (MN)$ vérifient $\overrightarrow{OP}.\overrightarrow{n} = k$, avec k une constante et O l'origine du repère (figure 12).

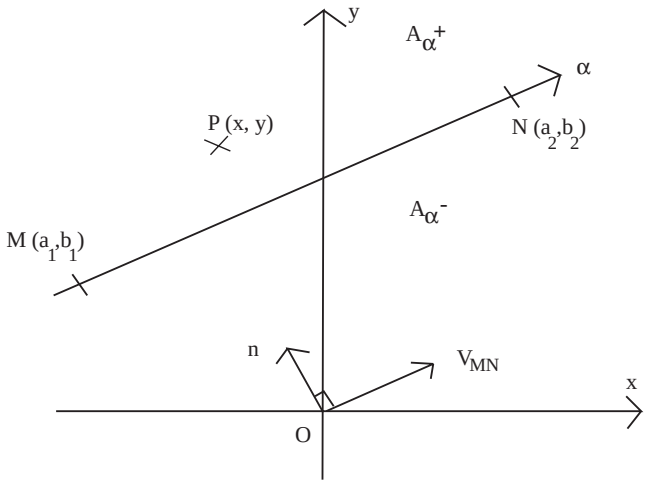

FIG. 12 – Détermination de l'appartenance d'un point à un demi-espace

Dans notre exemple nous avons :

$$\overrightarrow{OP}.\overrightarrow{n} = x(b_1 - b_2) + y(a_2 - a_1) = k = a_1(b_1 - b_2) + b_1(a_2 - a_1).$$

$$\boxed{P \in \begin{cases} A_{\alpha^+} & \text{Si } \overrightarrow{OP}.\overrightarrow{n} > k, \\ A_{\alpha^-} & \text{Si } \overrightarrow{OP}.\overrightarrow{n} < k, \\ \alpha & \text{Si } \overrightarrow{OP}.\overrightarrow{n} = k. \end{cases}}$$

6.2. **Partition d'une scène 2D en deux scènes par une coupure.** On veut savoir si un segment β appartient à l'ensemble positif ou négatif d'une coupure. On se donne d'abord P_1, P_2 les sommets du segment β. Ensuite, on applique deux produits scalaires : un entre le vecteur $\overrightarrow{v_\alpha}$ de la coupure α et $\overrightarrow{OP_1}$, puis un autre produit scalaire avec $\overrightarrow{v_\alpha}$ et $\overrightarrow{OP_2}$.

– Si les deux produits scalaires sont plus grands que la constante, alors $\beta \in A_{\alpha^+}$.

– Si les deux produits scalaires sont plus petits que la constante, alors $\beta \in A_{\alpha^-}$ (Exemple 1).
– Si un des produits scalaires est plus grand et que l'autre est plus petit, alors on recommence la même opération sur les segments $[P_1P']$ et $[P'P_2]$, où P' est l'intersection entre α et β. (Exemple 2).

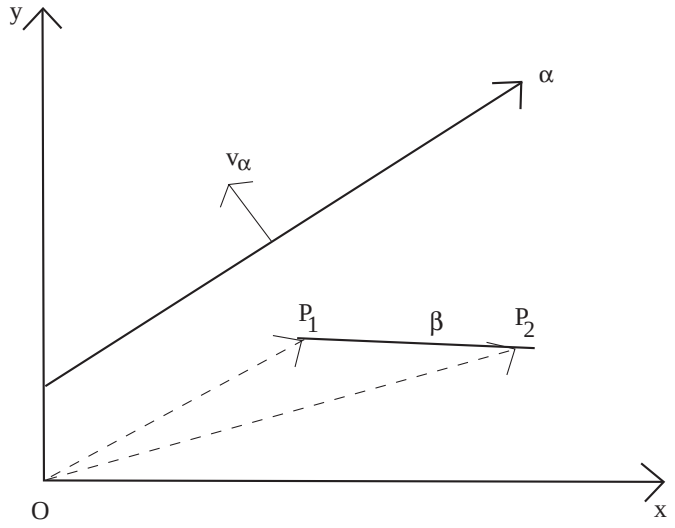

FIG. 13 – Exemple1

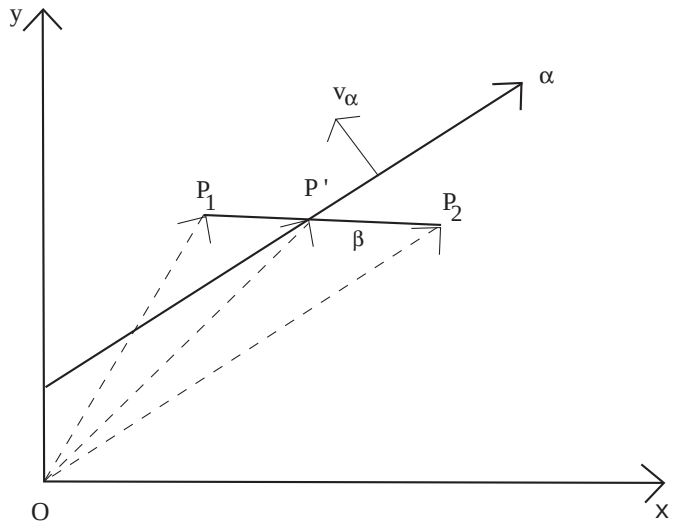

FIG. 14 – Exemple 2

RÉFÉRENCES

[FOLE95] J. Foley, A. van Dam, S. Feiner, J. Hughes, R. Phillips, *Introduction à l'infographie*, Addison-Wesley France, 1995.

[ABVA97] M. Abvash, *Zen de la programmation graphique* International Thomson Publishing France, 1997.

[EBEV02] D. Ebevly, *3D Game Engine and Design* Morgan Kaufman Publishers, 2002.

[ROSS86] J.R. Rossignac and A.A.G. Requicha, *Depth-Buffering Display Techniques for Constructive Solid Geometry* CG & A,6(9), Septembre 1986, 29-39.

[CATM74] E,A Catmull, *Subdivision Algorithm for Computer Display of Curved Surfaces* Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Departement, University of Utah, Salt Lake City, Décembre 1974.

[ATHE81] P.R. Atherton, *A Method of Interactive Visualisation of CAD Surface Models on a Color Video Display* SIGGRAPH 81, 279-287.

[faqs.org] *wwww.faqs.org\faqs\graphics \bsptree-faq\index.html*

[geocities.com] *wwww.geocities.com\SiliconValley\ 2151\bsp.html*

[cs.wpi.edu] *wwww.cs.wpi.edu\m̃att\courses \cs563\talks\bsp\bsp.html*