Chapitre 2

Rappels d'algèbre linéaire

Pour pouvoir aborder sereinement la suite de ce cours sur l'API OpenGL® , nous mettons en relation les notions de bases d'algèbre linéaire nécessaires et la géométrie 2D/3D. Il ne faut pas hésiter à faire des schémas.

2.1 Définitions et notations

Notation 2.1.1 *Pour pouvoir se repérer dans l'espace, on choisit un point que l'on appelle l'origine O et un triplet de vecteurs linéairement indépendants $(\overrightarrow{i}, \overrightarrow{j}, \overrightarrow{k})$. $(O, \overrightarrow{i}, \overrightarrow{j}, \overrightarrow{k})$est un repère de l'espace. Lorsque les vecteurs sont orthogonaux entre eux deux à deux, on parle de repère orthogonal. Lorsque les vecteurs sont de longueur unitaire, on parle de repère normé. Lorsque les vecteurs sont orthogonaux et de longueur unitaire, on parle de repère orthonormé. Les axes du système sont les droites O_x, O_y, O_z (voir la figure 2.1).*

Notation 2.1.2 *Un point M de l'espace est identifié de façon unique par ses coordonnées cartésiennes. On emploie indifféremment l'une des notations suivantes :*

- $M : x_M \overrightarrow{i} + y_M \overrightarrow{j} + z_M \overrightarrow{k}$
- $M : \begin{pmatrix} x_M \\ y_M \\ z_M \end{pmatrix}$

En particulier, dans le repère $(O, \overrightarrow{i}, \overrightarrow{j}, \overrightarrow{k})$, l'origine a pour coordonnées $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$. La figure 2.2 montre un point M de coordonnées $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ dans le repère $(O, \overrightarrow{i}, \overrightarrow{j}, \overrightarrow{k})$.

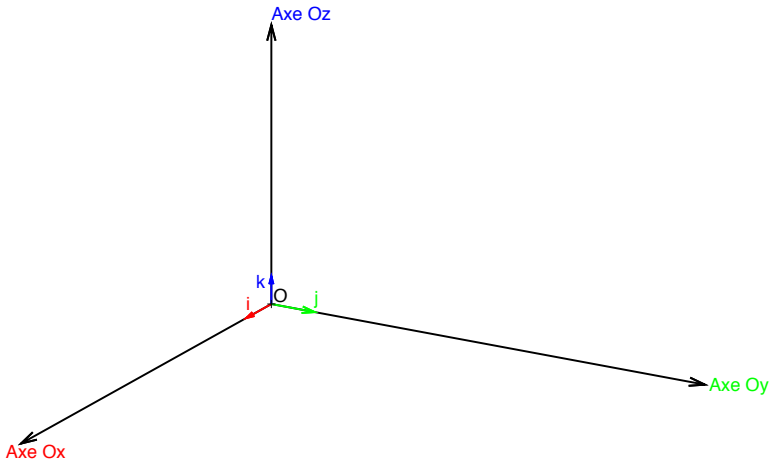

FIGURE 2.1 – Repère .

Exercice 2.1.1 *En vous reportant à la figure 2.2 et sachant que le point M a pour coordonnées* $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$*, donner les coordonnées de H_x, H_y et H_z.*

Notation 2.1.3 *Un vecteur $\overrightarrow{v}$ est identifié par ses coordonnées cartésiennes. On utilise indifféremment l'une des notations suivantes :*

- $\overrightarrow{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$
- $\overrightarrow{v} = v_x \overrightarrow{i} + v_y \overrightarrow{j} + v_z \overrightarrow{k}$

On utilise indifféremment les notations suivantes pour désigner un vecteur $\overrightarrow{v}$ entre deux points A et B :

- $\overrightarrow{v} = \overrightarrow{AB} = \begin{pmatrix} x_B - x_A \\ y_B - y_A \\ z_B - z_A \end{pmatrix}$
- $B = A + \overrightarrow{v}$

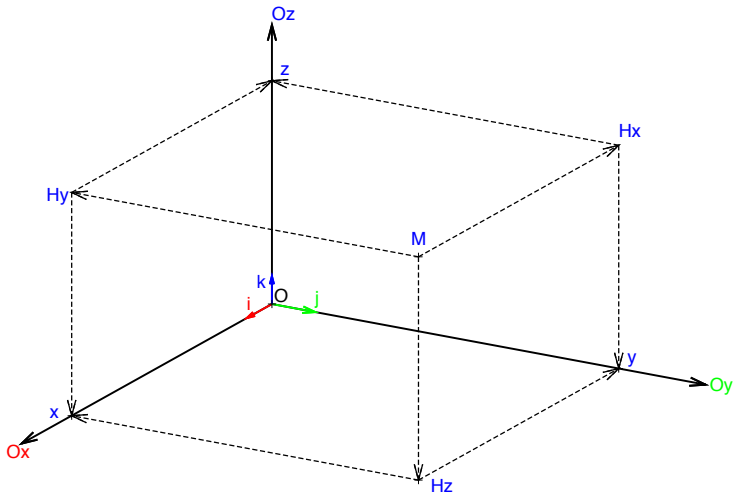

FIGURE 2.2 – Repère .

Par exemple, on a :

$$A : \begin{pmatrix} 0 \\ 2 \\ 3 \end{pmatrix} \quad B : \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}$$

$$\overrightarrow{AB} = \begin{pmatrix} 3-0 \\ 1-2 \\ 0-3 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \\ -3 \end{pmatrix}$$

2.2 Opérations vectorielles

Théorème 2.2.1 *La multiplication d'un vecteur $\overrightarrow{v}$ par un scalaire λ est définie par :*

$$\lambda \overrightarrow{v} = \begin{pmatrix} \lambda v_x \\ \lambda v_y \\ \lambda v_z \end{pmatrix}$$

Par exemple :

$$3 \begin{pmatrix} -1 \\ 2 \\ 5 \end{pmatrix} = \begin{pmatrix} -3 \\ 6 \\ 15 \end{pmatrix}$$

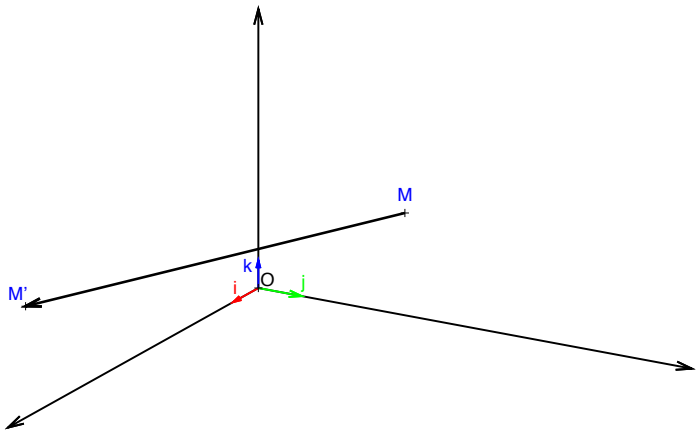

FIGURE 2.3 – Vecteurs .

Théorème 2.2.2 *L'addition de deux vecteurs est définie par :*

$$\overrightarrow{u} + \overrightarrow{v} = \begin{pmatrix} u_x + v_x \\ u_y + v_y \\ u_z + v_z \end{pmatrix}$$

Par exemple :

$$\begin{pmatrix} -1 \\ 2 \\ 5 \end{pmatrix} + \begin{pmatrix} -3 \\ 6 \\ 15 \end{pmatrix} = \begin{pmatrix} -4 \\ 8 \\ 20 \end{pmatrix}$$

Théorème 2.2.3 *La norme euclidienne d'un vecteur $\overrightarrow{v}$ est définie par :*

$$\|\overrightarrow{v}\| = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2}$$

On parle également de longueur du vecteur $\overrightarrow{v}$.

La distance d'un point A à un point B est $\|\overrightarrow{AB}\|$. C'est-à-dire :

$$\|\overrightarrow{AB}\| = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

Par exemple, la distance entre $A = \begin{pmatrix} -6 \\ 10 \\ 10 \end{pmatrix}$ et $B = \begin{pmatrix} -10 \\ 18 \\ 30 \end{pmatrix}$ est :

$$\|\overrightarrow{AB}\| = \| \begin{pmatrix} -4 \\ 8 \\ 20 \end{pmatrix} \| = \sqrt{(-4)^2 + 8^2 + 20^2} = \sqrt{480}$$

Notation 2.2.1 *Un vecteur $\overrightarrow{u}$ est dit unitaire lorsque sa norme euclidienne est égale à* 1.

Théorème 2.2.4 *Le produit scalaire de deux vecteurs $\overrightarrow{u}$ et $\overrightarrow{v}$ est un scalaire noté et défini par :*

- $\overrightarrow{u}.\overrightarrow{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} . \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$
- $\overrightarrow{u}.\overrightarrow{v} = u_x.v_x + u_y.v_y + u_z.v_z$
- $\overrightarrow{u}.\overrightarrow{v} = \|\overrightarrow{u}\|.\|\overrightarrow{v}\|.cos(\widehat{\overrightarrow{u}, \overrightarrow{v}})$

En particulier, deux vecteurs sont orthogonaux si et seulement si leur produit produit scalaire est nul :

$$\overrightarrow{u}.\overrightarrow{v} = \|\overrightarrow{u}\|.\|\overrightarrow{v}\|. \cos(\frac{\pi}{2}) = 0$$

Par exemple, les vecteurs $\overrightarrow{i}$ et $\overrightarrow{j}$ sont orthogonaux :

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} . \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 0$$

Exercice 2.2.1 *La norme d'un vecteur $\overrightarrow{v}$ est donnée par :*

$$\|\overrightarrow{v}\| = \sqrt{\overrightarrow{v}.\overrightarrow{v}}$$

Théorème 2.2.5 *Le produit vectoriel de deux vecteurs $\overrightarrow{u}$ et $\overrightarrow{v}$ est un vecteur $\overrightarrow{n}$ noté et défini par :*

- $\overrightarrow{u} \wedge \overrightarrow{v} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \wedge \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$
- $\overrightarrow{u} \wedge \overrightarrow{v} = \begin{pmatrix} u_y v_z - v_y u_z \\ u_z v_x - v_z u_x \\ u_x v_y - u_y v_x \end{pmatrix}$

Le produit vectoriel $\overrightarrow{n}$ de $\overrightarrow{u}$ et $\overrightarrow{v}$ est perpendiculaire (on dit aussi normal) à $\overrightarrow{u}$ et $\overrightarrow{v}$. Par exemple,

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \wedge \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Autrement dit,

$$\overrightarrow{i} \wedge \overrightarrow{j} = \overrightarrow{k}$$

Notation 2.2.2 *Deux vecteurs $\overrightarrow{u}$ et $\overrightarrow{v}$ sont de même direction s'il existe un scalaire λ tel que $\overrightarrow{u} = \lambda\overrightarrow{v}$*

Théorème 2.2.6 *La normalisation d'un vecteur $\overrightarrow{v}$ consiste à trouver un vecteur unitaire de même direction que $\overrightarrow{v}$. norme 1. La normalisation $\overrightarrow{n}$ de $\overrightarrow{v}$ est donnée par :*

$$\frac{1}{\|\overrightarrow{v}\|}\overrightarrow{v}$$

Par exemple, La normalisation de $\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$ donne $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$.

Exercice 2.2.2 *Normaliser* $\begin{pmatrix} 3 \\ 7 \\ -9 \end{pmatrix}$.

2.3 Transformation de $\mathbb{R}^3$

On considère des applications $\mathcal{A}$ de $\mathbb{R}^3$ dans $\mathbb{R}^3$:

$$\begin{array}{ccc} \mathbb{R}^3 & \to & \mathbb{R}^3 \\ M & \overset{\mathcal{A}}{\mapsto} & M' = \mathcal{A}(M) \end{array}$$

Théorème 2.3.1 *Une application linéaire de $\mathbb{R}^3$ est une application qui vérifie :*

$$\begin{array}{ccc} \mathbb{R}^3 & \to & \mathbb{R}^3 \\ M & \overset{\mathcal{A}}{\mapsto} & M' = \mathcal{L}(M) \end{array}$$

où les coordonnées du point M' sont :

$$\begin{cases} x' = & ax + by + cz + d \\ y' = & ex + fy + gz + h \\ z' = & ix + jy + kz + l \end{cases}$$

On peut noter cette application sous forme matricielle :

$$M' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \mathcal{M}_{\mathcal{L}}.M + \mathcal{C} = \begin{pmatrix} a & b & c \\ e & f & g \\ i & j & k \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} d \\ h \\ l \end{pmatrix}$$

Théorème 2.3.2 *Une translation de vecteur $\overrightarrow{T}$ est une application linéaire définie par :*

$$\begin{array}{ccc} \mathbb{R}^3 & \to & \mathbb{R}^3 \\ M & \overset{\mathcal{T}}{\mapsto} & M' = \mathcal{T}(M) \end{array}$$

avec :

$$M' = \mathcal{T}(M) = M + \overrightarrow{T}$$

Les coordonnées $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ *de M' sont données par :*

$$\begin{cases} x' = & x + T_x \\ y' = & y + T_y \\ z' = & z + T_z \end{cases}$$

ou encore :

$$M' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}$$

Exercice 2.3.1 *Calculer les coordonnées du point M' résultat de la translation de vecteur* $\overrightarrow{\mathcal{T}} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ *du point* $A = \begin{pmatrix} 3 \\ -5 \\ 2 \end{pmatrix}$.

Théorème 2.3.3 *Une homothétie de rapport λ de centre C est une application linéaire définie par :*

$$\begin{array}{ccc} \mathbb{R}^3 & \to & \mathbb{R}^3 \\ M & \overset{\mathcal{H}}{\mapsto} & M' = \mathcal{H}(M) \end{array}$$

avec :

$$\overrightarrow{CM'} = \lambda \overrightarrow{CM}$$

Les coordonnées de M' sont données par :

$$\begin{cases} x' - x_C = & \lambda x \\ y' - y_C = & \lambda y \\ z' - z_C = & \lambda z \end{cases}$$

ou encore :

$$M' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Exercice 2.3.2 *Donner les coordonnées de l'image de* $A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ *par l'homothétie de centre l'origine et de rapport $\lambda = -4$.*

Théorème 2.3.4 *Une rotation d'angle α d'axe $\overrightarrow{Oz}$ est une application linéaire définie par :*

$$\begin{array}{ccc} \mathbb{R}^3 & \to & \mathbb{R}^3 \\ M & \overset{\mathcal{R}_Z}{\mapsto} & M' = \mathcal{R}_Z(M) \end{array}$$

avec :

$$M' = \mathcal{R}_z(M) = \mathcal{M}_{\mathcal{R}_Z}.M = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Exercice 2.3.3 *Donner les coordonnées de l'image de $A = \begin{pmatrix} 5 \\ -2 \\ -4 \end{pmatrix}$ par la rotation d'axe Oz d'angle $\alpha = 45$. Faire un schéma.*

Théorème 2.3.5 *Une rotation d'angle α d'axe $\overrightarrow{Oy}$ est une application linéaire définie par :*

$$\begin{array}{ccc} \mathbb{R}^3 & \to & \mathbb{R}^3 \\ M & \overset{\mathcal{R}_Y}{\mapsto} & M' = \mathcal{R}_Y(M) \end{array}$$

avec :

$$M' = \mathcal{R}_Y(M) = \mathcal{M}_{\mathcal{R}_Y}.M = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Exercice 2.3.4 *Donner les coordonnées de l'image de $A = \begin{pmatrix} 5 \\ -2 \\ -4 \end{pmatrix}$ par la rotation d'axe Oy d'angle $\alpha = 45$. Faire un schéma.*

Théorème 2.3.6 *Une rotation d'angle α d'axe $\overrightarrow{Ox}$ est une application linéaire définie par :*

$$\begin{array}{ccc} \mathbb{R}^3 & \to & \mathbb{R}^3 \\ M & \overset{\mathcal{R}_X}{\mapsto} & M' = \mathcal{R}_X(M) \end{array}$$

avec :

$$M' = \mathcal{R}_x(M) = \mathcal{M}_{\mathcal{R}_X}.M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Exercice 2.3.5 *Donner les coordonnées de l'image de $A = \begin{pmatrix} 5 \\ -2 \\ -4 \end{pmatrix}$ par la rotation d'axe Ox d'angle $\alpha = 45$. Faire un schéma.*

2.4 Programmation

2.4.1 Column-major vs Row-major

Un vecteur ou une matrice est un tableau en mémoire. C'est une suite de valeurs $V = \{v_0, v_1, v_2\}$ ou $M = \{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}$. On peut noter de deux façons différentes les vecteurs et les matrices : en colonne d'abord ou en ligne d'abord.

En colonne d'abord,

$$V_{colonne} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix}$$

$$M_{colonne} = \begin{pmatrix} a_0 & a_3 & a_6 \\ a_1 & a_4 & a_7 \\ a_2 & a_5 & a_8 \end{pmatrix}$$

En ligne d'abord,

$$V_{ligne} = \begin{pmatrix} v_0 & v_1 & v_2 \end{pmatrix}$$

$$M_{ligne} = \begin{pmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{pmatrix}$$

2.4.2 Les notations en mathématiques

En mathématique, on utilise les vecteurs en colonne d'abord et les matrices en ligne d'abord :

$$V = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

$$M = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$$

La multiplication d'une matrice par un vecteur ou un point est donnée par :

$$V' = M.V = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix} . \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} m_{00}v_x + m_{01}v_y + m_{02}v_z \\ m_{10}v_x + m_{11}v_y + m_{12}v_z \\ m_{20}v_x + m_{21}v_y + m_{22}v_z \end{pmatrix}$$

2.4.3 Les notations avec OpenGL

Avec OpenGL® , on utilise les vecteurs en ligne d'abord et les matrices en colonne d'abord :

$$V'' = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix}$$

$$M'' = \begin{pmatrix} m_{00} & m_{10} & m_{20} \\ m_{01} & m_{11} & m_{21} \\ m_{02} & m_{12} & m_{22} \end{pmatrix}$$

La multiplication d'une matrice par un vecteur ou un point est donnée par :

$$V''' = V''.M'' = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix} . \begin{pmatrix} m_{00} & m_{10} & m_{20} \\ m_{01} & m_{11} & m_{21} \\ m_{02} & m_{12} & m_{22} \end{pmatrix}$$

$$= \quad ((m_{00}v_x + m_{01}v_y + m_{02}v_z) \quad (m_{10}v_x + m_{11}v_y + m_{12}v_z) \quad (m_{20}v_x + m_{21}v_y + m_{22}v_z))$$

2.4.4 Mathématiques vs OpenGL

On voit donc que les conventions OpenGL® et mathématiques pour l'écriture des vecteurs et des matrices est différentes.

OpenGL®	Mathématique
$(m_{00}v_x + m_{01}v_y + m_{02}v_z$ $m_{10}v_x + m_{11}v_y + m_{12}v_z$ $m_{20}v_x + m_{21}v_y + m_{22}v_z)$	$\begin{pmatrix} m_{00}v_x + m_{01}v_y + m_{02}v_z \\ m_{10}v_x + m_{11}v_y + m_{12}v_z \\ m_{20}v_x + m_{21}v_y + m_{22}v_z \end{pmatrix}$
$\begin{pmatrix} a \\ b \\ c \end{pmatrix}$	$\begin{pmatrix} a & b & c \end{pmatrix}$
$\begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$	$\begin{pmatrix} m_{00} & m_{10} & m_{20} \\ m_{01} & m_{11} & m_{21} \\ m_{02} & m_{12} & m_{22} \end{pmatrix}$

Mais comme nous allons le voir dans la suite, bien que les notations soient différentes, les opérations sont exactement les mêmes.

Théorème 2.4.1 *La transposée d'un vecteur colonne est un vecteur ligne :*

$$V = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \quad {}^tV = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix}$$

La transposée d'un vecteur ligne est un vecteur colonne :

$$V = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix} \quad {}^tV = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Théorème 2.4.2 *La transposée d'une matrice est donnée par :*

$$M = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix} \quad {}^tM = \begin{pmatrix} m_{00} & m_{10} & m_{20} \\ m_{01} & m_{11} & m_{21} \\ m_{02} & m_{12} & m_{22} \end{pmatrix}$$

Théorème 2.4.3 *La transposée d'une multiplication est la multiplication des transposées dans l'ordre inverse :*

$$^t(M.M'') =^t M''.^tM$$

L'ordre des multiplication est très important.

Finalement, on montre facilement que la manière dont les vecteurs et matrices sont représentés avec OpenGL® et en mathématiques sont équivalentes :

OpenGL®	Mathématiques
$V'' =^t V.^tM$	$V' = M.V$

avec $^tV = V''$.

2.5 Problème

Un polyèdre est un solide composé de surfaces planes. On considère les polyèdres suivant :

- pyramide à base triangulaire (4 faces, 4 sommets)
- pyramide à base rectangulaire (5 faces, 5 sommets)
- cube (6 faces, 8 sommets)
- dodécaèdre (12 faces, 20 sommets)
- icosaèdre (20 faces, 12 sommets)

L'exercice suivant a pour but vous familiariser avec la géométrie dans l'espace et les problématiques que vous rencontrerez dans la suite de ce document. Plus le nombre de sommets et de faces sera important, plus les calculs seront nombreux. Hormis le choix du repère et des sommets, vous pouvez écrire un programme pour les autres questions. Adaptez votre choix au temps dont vous disposez et à votre aisance. Vous pourrez par la suite choisir des solides plus difficile à modéliser.

Exercice 2.5.1 *choisir un des polyèdres de la liste précédente. Construire ce polyèdre et répondre aux questions :*

- *choisir un repère orthonormé judicieusement !*
- *donner les coordonnées de tous ses sommets*
- *donner les coordonnées des centres de chaque face (point situé à equidistance des sommets de la face)*
- *donner les coordonnées des deux normales (vers l'intérieur et vers l'extérieur) à chaque face*
- *pour un point M de coordonnées $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$, donner une méthode pour savoir si M est à l'intérieur, sur la surface ou à l'extérieur du solide*

- *Soient un point M de coordonnées $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$, un vecteur $\overrightarrow{v}$ de coordonnées $\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ et un angle θ inférieur à 90 degrés. On considère le cône constitué des points M' vérifiant $(\overrightarrow{v}, \widehat{\overrightarrow{MM'}}) = \theta$. Donner une méthode pour savoir si un sommet du solide est sur, à l'intérieur ou à l'extérieur de ce cône.*
- *donner une méthode pour animer ce solide en effectuant des rotations autour de son centre.*

Chapitre 3

Introduction au processus de traitement

3.1 Le Pipeline OpenGL®

L'API OpenGL® fournie un Pipeline fixe [1] permettant de calculer un rendu en fonction du flux de données fournis en entrée. Ces données sont composées d'une part d'un enchevêtrement de commandes OpenGL® et de données géométriques décrivant les objets et leurs placement dans la scène et d'autre part de données bitmaps généralement utilisées pour décrire les textures des objets. La figure 3.1 reprend les étapes principales de ce Pipeline.

1. A partir de la version 3.0 d'OpenGL® , une partie du Pipeline fixe tend à devenir obsolète au bénéfice d'une utilisation de plus en plus systématique des shaders et ceci même pour les exemples les plus basiques. Les fonctionnalités obsolètes sont maintenues mais deviennent dépréciées (*depcrecated*). Ici, dans une optique purement pédagogique, et pour une introduction en douceur à la programmation OpenGL® , nous continuerons à utiliser la partie dépréciée de l'API. Nous aborderons, vers la fin de l'ouvrage, les nouvelles techniques mises en place dans les dernières versions de l'API.

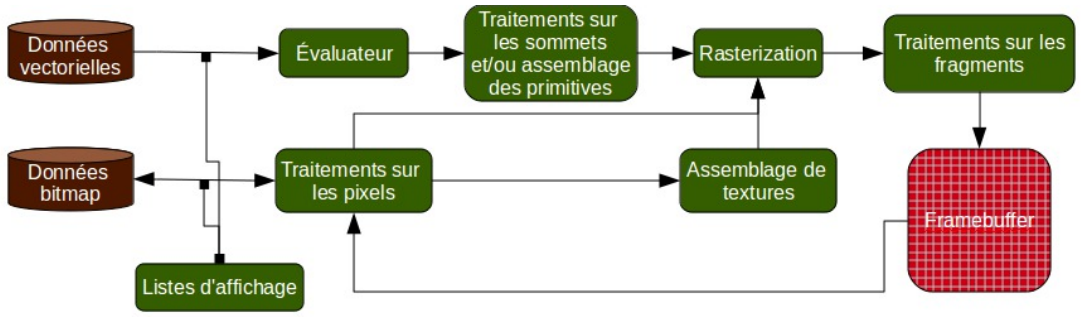

FIGURE 3.1 – Principales étapes du Pipeline fixe OpenGL® .

3.1.1 Points clés du processus de traitement

3.1.1.1 Les listes d'affichage

Cette fonctionnalité permet de rassembler des données de modélisation (aussi bien les données vectorielles que bitmap) sous une même étiquette. Ainsi, une liste d'affichage pourrait être vue comme un sous-ensemble *packagé* et prêt à l'emploi des données d'entrée, ce sous-ensemble représenterait un objet ou une partie d'objet pouvant être *exécuté* une ou plusieurs fois dans la scène. Ici, le terme *exécuté* signifie l'envoi, dans le Pipeline OpenGL® , des données contenues dans la liste d'affichage. Une analogie facile pourrait être faite avec les langages de programmation où l'utilisation des listes d'affichage se rapporterait aux langages compilés et où la description *narrative* de la scène (dite mode immédiat) se rapporterait aux langages interprétés.

3.1.1.2 L'évaluateur

L'évaluateur est un moteur permettant de produire des approximations de courbes ou de surfaces à l'aide d'arètes ou de facettes. Pour cela il utilise des courbes paramétriques (la courbe de Bézier en est une) discrétisées à un pas donné pour en faire un ensemble de primitives géométriques utilisables par le la suite du Pipeline OpenGL® . Ces courbes permettent aussi de calculer simplement et en chaque sommet les vecteurs normaux pour le calcul de l'"éclairage et les coordonnées de textures.

3.1.1.3 Traitement des sommets

A ce stade s'appliquent un ensemble de transformations géométriques (rotations, translations, changement de repère / d'échelle, projection, etc) principalement appliquées aux sommets mais aussi aux coordonnées de textures et aux normales aux faces.

3.1.1.4 Rasterization

Cette étape consiste à transformer des données encore vectorielles, représentées dans un pseudo-espace bidimensionnel prennant en compte la profondeur (un cube dans le quel les objets sont projetés orthogonalement sur l'une des face du cube), en fragments. Un fragment est l'élément qui, *in fine*, sera le pixel, à un détail prêt qui est qu'un pixel peut être composé (calculé à partir) de plusieurs fragments.

3.2 Projection

Maintenant, nous allons tenter de donner au lecteur une façon de se représenter l'espace dans lequel OpenGL® modélise les objets présents dans une scène puis expliquer les techniques mises en œuvre pour projeter la scène à l'écran.

Dans l'espace de modélisation, l'observateur peut être comparé à une caméra placée à l'origine du repère OpenGL® ; c'est un repère main droite tel que schématisé en

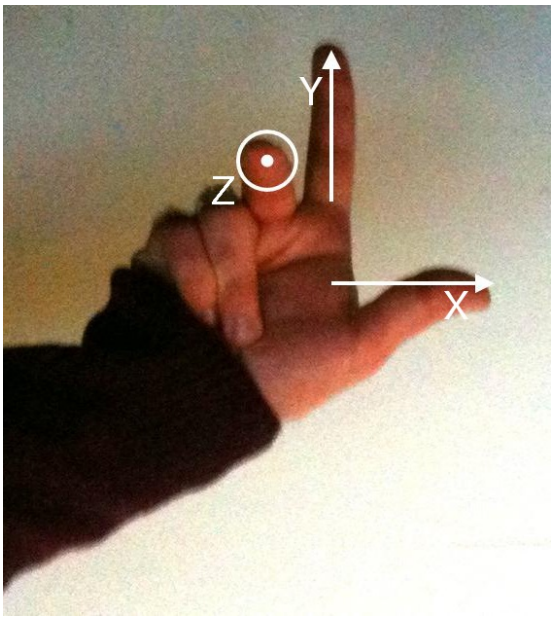

FIGURE 3.2 – Repère direct main droite OpenGL® .

figure 3.2. Cette caméra regarde vers les z négatifs, son vecteur *haut* correspond à l'axe des y et les x positifs sont à sa droite. Ainsi, la scène visualisée à l'écran est composée des objets se trouvant dans un volume *virtuel* de type hexaèdre situé en face de l'observateur, et le résultat final sera différent selon la projection choisie. OpenGL® propose deux types de projections, la projection perspective (§3.2.2) et la projection orthogonale (cf. §3.2.2).

3.2.1 Repère OpenGL®

OpenGL® adopte un repère dit *direct* : il s'agit de positionner les vecteurs de la base $(\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z})$ de manière à avoir le vecteur $\overrightarrow{Z}$ comme la résultante du produit vectoriel entre les vecteurs $\overrightarrow{X}$ et $\overrightarrow{Y}$. Ce repère est aussi appelé repère main droite, la figure 3.2 l'illustre dans le cas de l'API OpenGL® , ici l'axe $\overrightarrow{X}$ (les abscisses) est donné par le pouce et est orienté vers la droite, l'axe des $\overrightarrow{Y}$ (les ordonnées) est donné par l'index et est orienté vers le haut et enfin l'axe des $\overrightarrow{Z}$ (la profondeur) est donné par le majeur et est orienté vers l'arrière (ou l'observateur).

3.2.2 Projection perspective

OpenGL® donne ici la possibilité de calculer une projection dont le point de perspective est l'œil de l'observateur (ou l'objectif de la caméra). Pour cela, il est nécessaire de décrire le plan *virtuel* (correspondant en définitif à l'écran de l'utilisateur) sur lequel le volume sera projeté ainsi que la profondeur du volume à projeter. C'est la

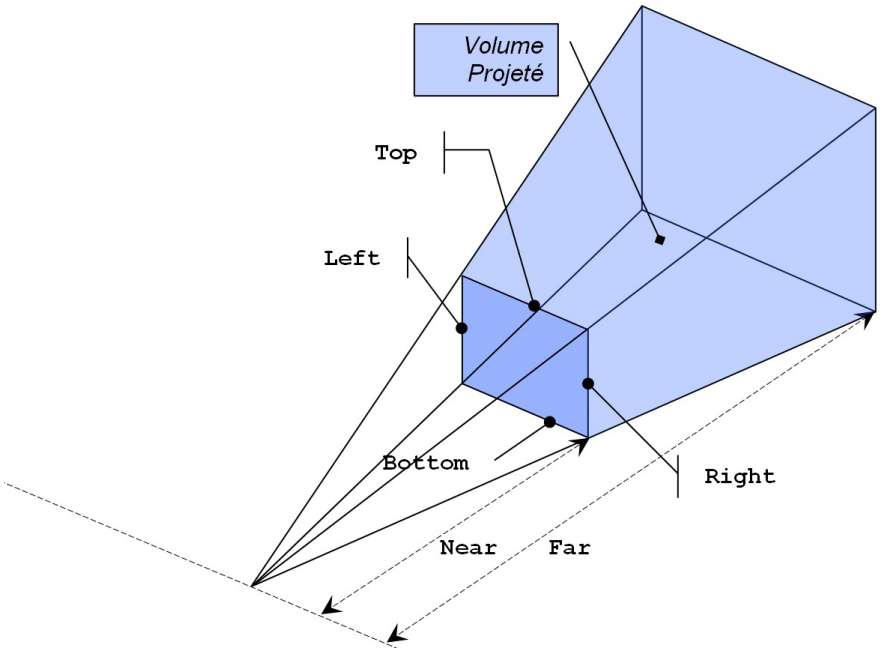

FIGURE 3.3 – Projection perspective.

fonction `glFrustum` qui permet de calculer les coefficients de la matrice de projection [2] en fonction des paramètres qui lui sont passés :

```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom,
               GLdouble top, GLdouble near, GLdouble far);
```

La figure 3.3 illustre le résultat attendu lors de l'utilisation du `glFrustum`. Comme nous pouvons le voir sur ce schéma, le volume projeté s'élargie en fonction de la distance, ainsi, plus nous partons dans les z négatifs plus le volume nous permet d'y placer des objets. D'où, pour des objets de dimensions comparables, notons que, une fois projetés, ils parraissent plus petits pour une distance du point de vue qui est plus grande ; le volume subit une compression plus forte en fonction de la distance, c'est l'effet perspective (pensez à l'image de la route qui fuit vers l'horizon) qui est la conséquence de la transformation du volume projeté initial [3] en un Parallélépipède.

2. En coordonnées homogènes, la projection perspective est donnée par la matrice :

$$\mathcal{P} = \begin{pmatrix} \frac{2\times n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2\times n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2f\times n}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

où l, r, b, t, n et f correspondent respectivement aux paramètres `left`, `right`, `bottom`, `top`, `near` et `far` de la fonction `glFrustum` avec $r \neq l$, $t \neq b$ et $f \neq n$.

3. Quel est le nom de cet hexahèdre ressemblant à une pyramide dont la cime a été tronquée ?

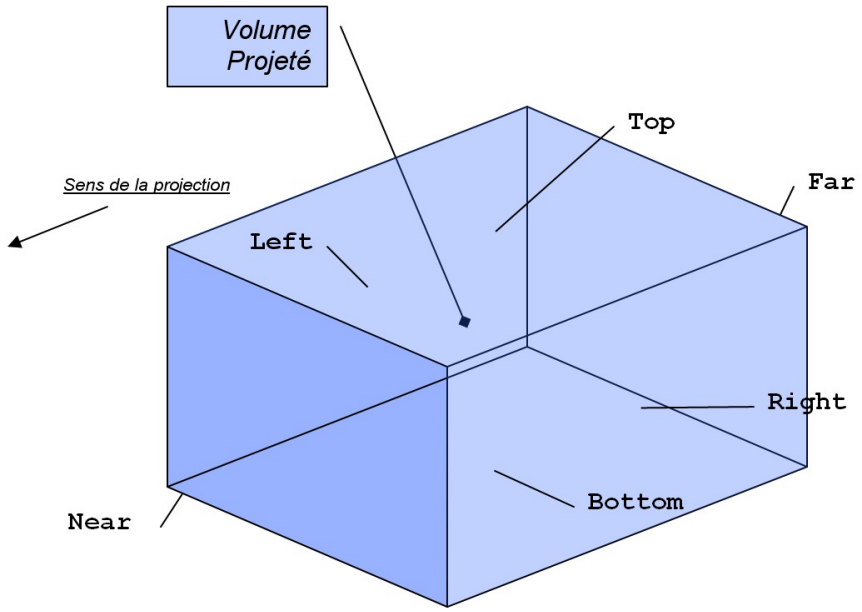

FIGURE 3.4 – Projection orthogonale.

3.2.3 Projection orthogonale

Ce second type de projection, fourni par l'API OpenGL® , peut être vu comme un cas particulier de la projection perpective où le point de perpective fuit à l'infini vers les z positifs. Ici, l'impression d'éloignement liée à la distance par rapport au plan de projection (le plan perpendiculaire à l'axe des z et placé en $z = Near$) n'est plus préservée car tout est projeté orthogonalement à ce plan. Ainsi, la fonction `glOrtho` permet de calculer les coefficients de la matrice de projection [4] en fonction des paramètres qui lui sont passés :

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
             GLdouble top, GLdouble near, GLdouble far);
```

et la figure 3.4 illustre le résultat produit par l'utilisation de cette fonction.

4. En coordonnées homogènes, la projection orthogonale est représentée par la matrice :

$$\mathcal{P} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

où l, r, b, t, n et f correspondent respectivement aux paramètres `left`, `right`, `bottom`, `top`, `near` et `far` de la fonction `glOrtho` avec $r \neq l$, $t \neq b$ et $f \neq n$.

3.3 Description du modèle

Pour modéliser des objets, l'élément de base utilisé par l'API OpenGL® est le sommet — *vertex* en anglais. Cette approche donne trois possibilités de rendu de l'objet à modéliser :

- Le mode **point** permet de visualiser l'objet modélisé comme un nuage de points ; une largeur est affectée aux points, elle est utilisée au moment de la *rasterization* pour lui donner une épaisseur lors du rendu sur le *framebuffer* (pour rappel, en géométrie le point n'a pas d'épaisseur) ;
- Le mode **fil de fer** permet de visualiser l'objet modélisé à l'aide de segments de droites obtenus en reliant les sommets entre-eux. Ici, le choix de relier deux sommets n'est pas obtenu par calcul (comme une triangulation), il est spécifié dans le modèle lui-même. Ainsi, en plus des sommets, il est nécessaire de décrire la façon dont ces sommets sont reliés. Comme pour le mode point, une largeur est affectée aux segments, elle est utilisée au moment de la *rasterization* pour leurs donner une épaisseur lors du rendu sur le *framebuffer* (pour rappel, en géométrie le segment n'a pas d'épaisseur) ;
- Le mode **solide** ou **plein** permet de visualiser l'objet modélisé à l'aide de facettes obtenues en reliant les sommets entre-eux. Afin de décrire ces facettes et pour des raisons d'optimisation de vitesse de rendu, OpenGL® utilise uniquement des polygones convexes [5] ; dans OpenGL® , le rendu d'un polygone non convexe n'est absolument pas garanti et peut varier selon l'implémentation de l'API. Comme pour les segments, la façon dont les sommets sont composés pour obtenir des polygones n'est pas obtenue par calcul, une fois encore, c'est dans le modèle lui-même que cela doit être spécifié.

Ainsi, en pratique, pour modéliser un objet ou une partie d'objet, l'API OpenGL® fournie un ensemble de commandes permettant de décrire en détail la composition du modèle. Ces commandes donnent la possibilité d'introduire des données géométriques dans le Pipeline OpenGL® . Pour cela, ces données doivent [6] être déclarée entre la paire de commandes `glBegin` et `glEnd` dont voici les prototypes :

```
void glBegin(GLenum mode);
void glEnd(void);
```

Chaque sommet est alors déclaré à l'aide des commandes `glVertex*` et le modèle est construit en fonction du mode (argument de `glBegin`) choisi. Nous donnons, ci-après, les valeurs possibles pour le paramètre `mode` ansi que les l'interprétation qui en découle. La figure 3.5 donne un aperçu des résultats pouvant être obtenus dans chaque

5. La restriction de l'API à la seule utilisation de polygones convexes permet de simplifier les calculs nécessaires au remplissage des faces et optimiser la vitesse d'exécution du rendu. Par ailleurs, cette restriction n'interdit pas la représentation des autres classes de polygones (concaves, croisés ou étoilés) car ces derniers pourront toujours être représentés par une union de polygones convexes.

6. Il existe des alternative à cette méthode de déclaration, nous les aborderons plus tard.

situation.
Ainsi pour un ensemble de n sommets $\{S_0, S_1, ..., S_{n-1}\}$ déclarés (à l'aide de `glVertex*`) entre la paire `glBegin` et `glEnd`, nous obtenons, quand le mode choisi est :

- `GL_POINTS` : Tous les sommets sont rendus sous la forme de points ;
- `GL_LINES` : Des segments reliant chaque paire de sommets sont dessinés. Si le nombre de sommets déclarés entre les commandes `glBegin` et `glEnd` est impair alors le dernier sommet sera omis ;
- `GL_LINE_STRIP` : La suite comprenant au minimum deux sommets est reliée par des segments et sans aucune contrainte. Ainsi, pour $n \geq 2$, les segments $[S_0, S_1]$, $[S_1, S_2]$, ..., $[S_{n-2}, S_{n-1}]$ sont dessinés ;
- `GL_LINE_LOOP` : Identique à `GL_LINE_STRIP` à l'exception de l'ajout d'un segment entre les sommets S_{n-1} et S_0 pour compléter la boucle ;
- `GL_POLYGON` : Dessine le polygone convexe constitué de l'ensemble des sommets $\{S_0, ..., S_{n-1}\}$ avec $n \geq 3$. Il n'est pas nécessaire de clôturer la forme en répétant le premier sommet en fin de série ;
- `GL_TRIANGLES` : Dessine pour chaque triplet de sommets un triangle ; les sommets sont pris dans l'ordre et ne sont utilisés qu'une seule fois. Si le nombre de sommets déclarés entre les commandes `glBegin` et `glEnd` n'est pas un multiple de trois, le ou les deux sommets restants sont ignorés ;
- `GL_QUADS` : Dessine pour chaque quadrulet de sommets un quadrilatère convexe ; les sommets sont pris dans l'ordre et ne sont utilisés qu'une seule fois. Si le nombre de sommets déclarés entre les commandes `glBegin` et `glEnd` n'est pas un multiple de quatre, l'unique, les deux ou les trois sommets restants sont ignorés ;
- `GL_TRIANGLE_STRIP` : Dessine une bande (le mot *stip* signifie bande) à l'aide d'un série de triangles. Nous prenons, pour chaque triangle pair de la série les trois premiers sommets dans l'ordre *un, deux, trois* et pour chaque triangle impair de la série les trois premiers sommets dans l'ordre deux, un, trois, à chaque fois, nous décalons d'une position le point de départ de la série avant de continuer sur le reste. Ainsi, pour les six sommets $\{S_0, S_1, S_2, S_3, S_4, S_5\}$ nous obtenons quatre triangles $\{S_0, S_1, S_2\}$, $\{S_2, S_1, S_3\}$, $\{S_2, S_3, S_4\}$ et $\{S_4, S_3, S_5\}$, en général, pour n sommets nous obtenons $n - 2$ triangles. Les sommets sont omis si leur nombre est inférieur à trois ;
- `GL_QUAD_STRIP` : Dessine une bande (le mot *stip* signifie bande) à l'aide d'une série de quadrilatères convexes. Nous prenons, pour chaque quadrilatère, les quatre premiers sommets dans l'ordre *premier, second, quatrième, troisième*, puis nous décalons d'une position le début de la série et continuons sur le reste. Ainsi, pour les huit sommets $\{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$ nous obtenons trois quadrilatères $\{S_0, S_1, S_3, S_2\}$, $\{S_2, S_3, S_5, S_4\}$ et $\{S_4, S_5, S_7, S_6\}$, en général, pour n sommets nous obtenons $\frac{n}{2} - 1$ quadrilatères. Les sommets sont omis si leur nombre est inférieur à quatre ;
- `GL_TRIANGLE_FAN` : Dessine un éventail (le mot *fan* signifie éventail) à l'aide d'une série de triangles ayant tous pour premier sommet le premier som-

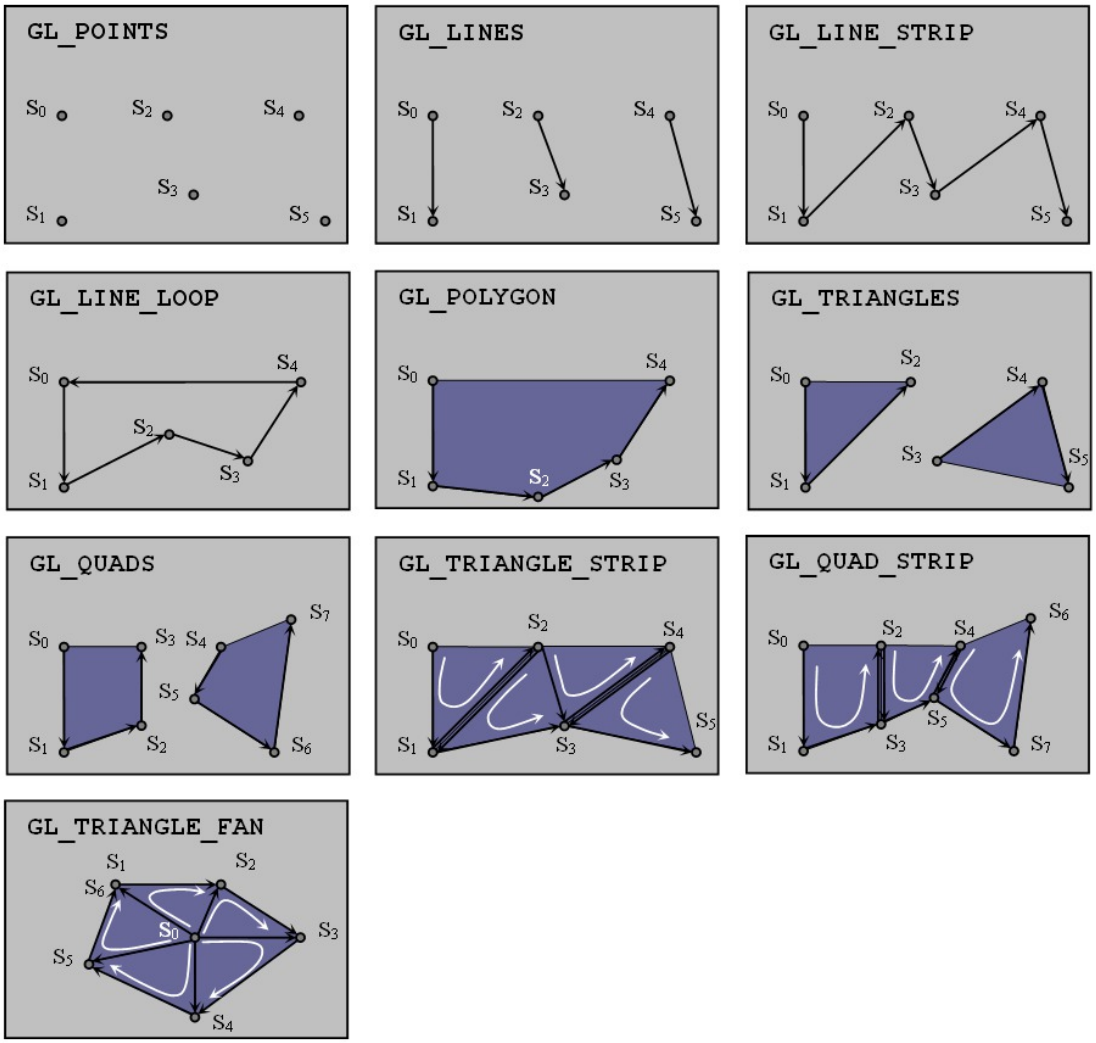

FIGURE 3.5 – Primitives OpenGL® .

met de toute la série, soit S_0. Pour les deux autres sommets, nous prenons respectivement le deuxième et le troisième sommet du reste de la série (tous les sommets sauf S_0) en décalant cette série d'une position à chaque nouveau triangle à créer. Ainsi, pour les sept sommets $\{S_0, S_1, S_2, S_3, S_4, S_5, S_6\}$ nous obtenons cinq triangles $\{S_0, S_1, S_2\}$, $\{S_0, S_2, S_3\}$, $\{S_0, S_3, S_4\}$, $\{S_0, S_4, S_5\}$ et $\{S_0, S_5, S_6\}$ (dans l'exemple donné en figure 3.5, le sommet S_6 est confondu avec le sommet S_1 afin d'obtenir un disque), en général, pour n sommets nous obtenons $n - 2$ triangles. Les sommets sont omis si leur nombre est inférieur à trois.