République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la recherche scientifique

Université Saad DAHLAB de BLIDA

Faculté des Sciences

Département Informatique

Mémoire pour l'obtention d'un diplôme de Master en Informatique

Option :

Ingenierie du logiciel

Thème:

Conception et réalisation d'une application pour le calcul de structure en Génie Civil

Présenté par :

- Benmiloud Mohamed Sabri
- Benhoura Abderrahmane

Promotrice :

- Mme Abed

Juillet 2011

Mr. - Cherif Zahar

Mm - Aouichiche

Mr. - Oldaissa

Dédicaces

Je dédie ce travail en signe de reconnaissance à mes très chers parents qui on tout fait pour me donner une bonne éducation et me soutenir dans mes études.

- ✓ A mes frères.
- ✓ A toute ma famille.
- ✓ A Mme Abed
- ✓ A Mr. Abed
- ✓ A tous mes amis pour leur encouragement, aide et soutien en l'occurrence, aide et soutien en l'occurrence : Roudhane, Abdelhak, Mustapha, Bilal, Mahfoud et Asma.

Benhoura Abderrahmane

Dédicaces

Je dédie ce mémoire :

- ✓ A mes chères parents pour leur sacrifice, l'éducation qu'ils m'ont inculqué et leurs efforts consentis à mon égard tout au long de mes études.
- ✓ A Ma sœur.
- ✓ A toute ma famille.
- ✓ A Mme Abed
- ✓ A Mr. Abed
- ✓ A tous mes amis pour leur encouragement, aide et soutien en l'occurrence.

Benmiloud Mohamed Sabri

Remerciements

Merci à toutes les personnes qui nous ont encouragé et soutenu tout au long de la réalisation de ce modeste travail, qu'elles trouvent ici l'expression de notre profonde gratitude.

On tient à remercier particulièrement notre promotrice Mme Abed pour sa disponibilité, ses précieux conseils ainsi que le soin qu'elle a apporté à nous guider dans notre travail tout en m'inculquant la notion de recherche.

On remercie infiniment les membres du jury pour avoir bien voulu accepter d'évaluer notre travail.

Liste des figures

Liste des figures

Liste des algorithmes et codes

Liste des algorithmes et codes

Sommaire

Sommaire

Chapitre 4 : conception

Chapitre 5 : Implémentation

Introduction générale

Introduction générale :

Les logiciels d'aujourd'hui, comme toutes les technologies de l'information et de la communication (T.I.C.), progressent à grands pas et constituent un outil incontournable dans le travail des spécialistes de tous les domaines de l'activité humaine. Ils permettent une productivité et un rendement plus ou moins performent selon l'élaboration de ces derniers. Ce marché de logiciels représente une grande part de plus value de l'économie mondiale.

Notre travail est d'élaborer un logiciel dans le domaine de la construction de bâtiment. Ce logiciel prend en charge une interface graphique souple à manipuler et un stockage des informations techniques contenant les règles de calcul de structures.

Le calcul de structures constitue une étape importante pour la projection d'éléments de construction de bâtiments en Génie Civil. L'ingénieur peut prévoir la réalisation de son projet de Génie Civil avant sa concrétisation. Cela lui permet de dimensionner sa structure, de paramétrer la stabilité, et la solidité ainsi que les critères de fonctionnalité et de sécurité d'un bâtiment, d'optimiser les coûts, le temps, la performance.

Cette phase de conception peut être menée à l'aide d'outils informatiques permettant d'assister l'ingénieur dans la prise de décision.

Dans ce contexte, nous nous sommes fixés pour but de réaliser une application pour le calcul de structure pour l'ingénieur de génie civil avec les objectifs suivants :

- Générer une base de données, contenant toute les informations techniques d'un projet de Génie Civil.
- Permettre une visualisation 3D et des vues en plan.
- Interagir avec les objets affichés en toute simplicité.
- Coloration des objets suivant des critères donnés.
- Permettre des méthodes de sélections d'objets.

Par conséquent, nous avons choisi comme thème : **« conception et réalisation d'une application d'aide pour le calcul de structures en génie civil».**

Ce sujet s'inscrit dans un même projet composé d'une partie A et une partie B.la premier partie représente le présent travail ayant trait a l'aspect informatique. La deuxième partie sera réaliser par un binôme du département de génie civil qui vent se servir de l'application de la partie pour former un système complet de calcul de structures.

Pour réaliser ce travail, nous l'avons partagé en cinq chapitres qui sont comme suit:

Chapitre 1 : Calcul de structures.

Présentation du domaine de génie civil avec la nécessité de garantir la sécurité d'une structure grâce à un bon dimensionnement et une bonne connaissance des matériaux, sections et type des appuis utilisé dans les structures.

Chapitre 2 : Notions mathématiques et synthèse d'images.

Dans ce chapitre, nous allons aborder les concepts mathématiques et quelques algorithmes pour bien se lancer dans le domaine de la 3D.

Chapitre 3 : Outil d'implémentation Java 3D et moteurs 3D.

Cette partie décrit les moteur3D Java3D, on montrera quelques classes en se basant sur un exemple simple.

Chapitre 4 : Conception.

Nous avons utilisé le formalisme UML pour modéliser les données et les traitements de notre application.

Chapitre 5 : Implémentation.

Ce chapitre est consacré à l'implémentation de l'application basée sur l'environnement Java et le Java 3D pour tous les affichages en 3D et la communication entre la base de données et l'application.

Chapitre 1

Calcul de structure

1.1 Introduction :

Le Génie civil représente l'ensemble des techniques concernant les constructions civiles. Les ingénieurs en génie civil s'occupent de la conception, de la réalisation, de l'exploitation et de la réhabilitation d'ouvrages de construction et d'infrastructures dont ils assurent la gestion afin de répondre aux besoins de la société, tout en assurant la sécurité du public et la protection de l'environnement. Très variées, leurs réalisations se répartissent principalement dans cinq grands domaines d'intervention: structures, géotechnique, hydraulique, transport, et environnement.

1.2 Les structures :

Une structure se rapporte à un système de pièces reliées, employées pour soutenir une charge. Les exemples importants liés au génie civil incluent des bâtiments, des ponts, et des tours. Afin de concevoir une structure, on doit remplir une fonction spécifique pour l'usage public. Elle permet d'assurer à la construction son indéformabilité, donc sa solidité et sa stabilité donc l'ingénieur doit expliquer sa sûreté, son esthétique, et son utilité, tout en prenant les contraintes économiques et environnementales en considération **[CM]**.

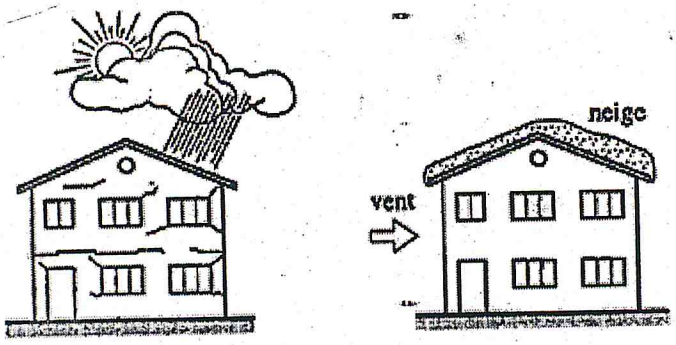


Figure 1.1 La sécurité structurale d'une construction.

1.3 Calcul de Structures :

Le calcul de structures comporte l'ensemble des lois physiques et des mathématiques exigées pour étudier et prévoir le comportement des structures. Les sujets de calcul de structures sont des objets façonnés de technologie qui sont jugés sur leur capacité de résister à des charges. Les éléments finis, une des méthodes de calcul de structures, incluent généralement des bâtiments, des ponts, des avions, des bateaux et des véhicules. D'un point de vue théorique, le but de l'analyse structurale est le calcul des déformations, des forces internes, et des efforts.

1.4 Charges et actions :

Une bonne connaissance des charges et de la résistance est nécessaire pour effectuer le calcul d'une structure. On distingue quatre types de charges et actions **[CM]**:

1.4.1 Poids propre de la structure porteuse :

Le poids propre de la structure porteuse à considérer tant pour la vérification de l'aptitude au service que de la sécurité structurale se base sur la valeur moyenne G_m obtenue à partir des dimensions théoriques de la structure. Les charges volumiques moyennes des principaux matériaux de construction sont contenues dans le tableau.

Matériau	Charge volumique [kN/m^3]
Aluminium	27
Acier	78.5
Béton – armé	25
– non armé	24
– léger	≤ 20
Bois	≤ 8

Figure 1.2 Charges volumiques moyenne des principaux matériaux de construction.

1.4.2 Poids propre des éléments non porteurs :

Pour la sécurité structurale, la valeur représentative Q_r du poids propre des éléments non porteurs est égale à la valeur moyenne. Tout comme pour le poids propre de la structure porteuse, celle-ci est obtenue en multipliant les dimensions théoriques par les charges volumiques moyennes des éléments non porteurs tels que les galandages, les revêtements, les toitures, les façades et les seconds œuvres.

1.4.3 Charges utiles dans les bâtiments :

Selon la fonction et l'affectation des bâtiments, on peut distinguer entre les trois groupes de surfaces utilisables suivantes :

- Les surfaces des locaux habitables, commerciaux ou administratifs.
- Les surfaces des entrepôts, les locaux de fabrication, des archives et des silos.
- Les surfaces des parkings et les surfaces accessible au trafic.

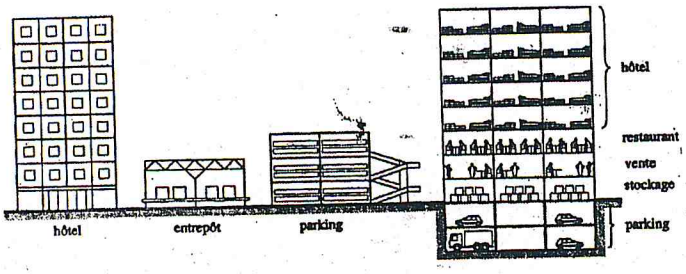


Figure 1.3 Charges utiles dans les bâtiments.

1.4.4 Actions climatiques et accidentelles :

Les actions climatiques comme la charge de la neige sur un terrain horizontal ou l'action due au vent sont directement proportionnelles à sa pression dynamique q, qui dépend elle-même de sa vitesse v.

On a : $q=\rho/2 * v^2$ avec ρ densité de l'aire

Pour tenir compte des actions climatiques dues aux variations de température, il est en général suffisant, pour les bâtiments et les constructions industrielles, de considérer une variation uniforme de température ΔT.

Par action accidentelle, on comprend les actions de très courte durée dont la présence avec une grandeur significative est peu ou pas probable au cours de la durée de service prévue.les chocs, déraillements, incendies, explosions et séismes constituent des actions accidentelles.

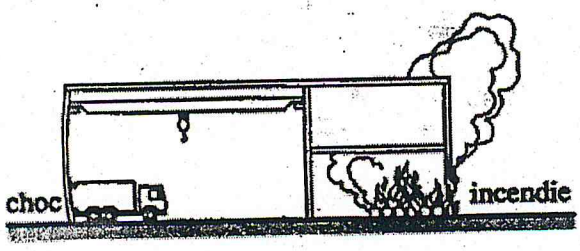


Figure 1.4 Exemple actions accidentelles.

1.5 Les matériaux dans une structure :

1.5.1 Définition d'un matériau :

Un matériau est une matière d'origine naturelle ou artificielle que l'homme façonne pour en faire des objets. C'est donc une matière de base sélectionnée en raison de propriétés particulières et mise en œuvre en vue d'un usage spécifique.

1.5.2 Caractéristiques des matériaux :

Un matériau peut être caractérisé selon de nombreux paramètres :

- Sa contrainte
- Sa déformation

La contrainte est proportionnelle à la déformation, le facteur de proportionnalité est noté E et appelé module d'élasticité (ou module de Young).

On a : contrainte = (Module d'élasticité) × (déformation)^{2}.

1.5.3 Propriétés physiques des matériaux :

- **Masse volumique :**

La masse volumique est une grandeur physique qui caractérise la masse d'un matériau par unité de volume. Elle est déterminée par le rapport P=M/V, où M est la masse de la substance homogène occupant un volume V. L'unité de mesure de la masse volumique dans le système international est le kilogramme par mètre cube ($kg \cdot m^{-3}$ ou kg/m^3). Les matériaux à masse volumique importante sont utilisés à la fabrication de contrepoids (équilibrage), volants d'inertie, etc. Ceux à faible masse volumique sont utilisés dans l'aéronautique par exemple.

- **Coefficient de dilatation :**

Le coefficient de dilatation mesure l'augmentation relative de volume d'un système lorsque l'on ne fait varier qu'un seul paramètre, en général la pression ou la température, mais également la concentration. Entre en jeu, par exemple pour des matériaux soumis à des écarts de température importants.

- **Chaleur massique :**

La chaleur massique ou chaleur spécifique1 (symbole c ou s), qu'il convient d'appeler capacité thermique massique, est déterminée par la quantité d'énergie à apporter par échange thermique pour élever d'un Kelvin la température de l'unité de masse d'une substance. C'est donc une grandeur intensive égale à la capacité thermique rapportée à la masse du corps étudié. L'unité du système international est alors le joule par kilogramme-kelvin ($J \cdot kg^{-1} \cdot K^{-1}$), elle est utilisée dans les accumulateurs thermiques des habitations.

- **Température de fusion :**

La température de fusion ou le point de fusion d'un corps représente la température à une pression donnée, à laquelle un élément pur ou un composé chimique passe de l'état solide à l'état liquide.

- **Module de Young :**

Le module de Young ou module d'élasticité (longitudinale) ou encore module de traction est la constante qui relie la contrainte de traction (ou de compression) et la déformation pour un matériau élastique isotrope. Le physicien britannique Thomas Young (1773-1829) avait remarqué que le rapport entre la contrainte de traction appliquée à un matériau et la déformation qui en résulte (un allongement relatif) est constant, tant que cette déformation reste petite et que la limite d'élasticité du matériau n'est pas atteinte. La loi d'élasticité est la loi de Hooke :

On a : $\sigma = E * \xi$

Avec :

- σ est la contrainte (en unité de pression),
- E est le module de Young (en unité de pression),
- ξ est l'allongement relatif, ou déformation (adimensionnel).

- **Coefficient de Poisson :**

Le coefficient de Poisson ou coefficient principal de Poisson permet de caractériser la contraction de la matière perpendiculairement à la direction de l'effort appliqué. Le coefficient de Poisson fait partie des constantes élastiques. Il est compris entre -1 et 0,5. Les valeurs expérimentales obtenues dans le cas d'un matériau parfaitement isotrope sont très proches de la valeur théorique (1/4). Pour un matériau quelconque, on obtient en moyenne 0,3. Il existe également des matériaux à coefficient de Poisson négatif : on parle alors parfois de matériaux auxétiques.

V=contraction transversale unitaire /allongement axial unitaire

- **Conductivité thermique :**

La conductivité thermique est une grandeur physique caractérisant le comportement des matériaux lors du transfert thermique par conduction. Notée λ (ou k en anglais), cette constante apparaît par exemple dans la loi de Fourrier. Elle représente la quantité de chaleur transférée par unité de surface et par une unité de temps sous un gradient de température de 1 degré par mètre.

- **conductivité électrique :**

La conductivité électrique est l'aptitude d'un matériau ou d'une solution à laisser les charges électriques se déplacer librement, autrement dit à permettre le passage du courant

électrique, unité est $S.m^{-1}$ (siemens par mètre) mais la mesure avec un conductimètre donne le résultat en $mS.cm^{-1}$ (milli siemens par centimètre) avec 1 $mS.cm^{-1}$ = 0,1 $S.m^{-1}$.

1.6 Les matériaux utilisés dans une structure :

1.6.1 Acier :

L'acier est un alliage métallique utilisé dans les domaines de la construction métallique et de la construction mécanique. Il est constitué d'au moins deux éléments, majoritairement le fer puis le carbone dans des proportions comprises entre 0,02 % et 2 % en masse. Les aciers sont élaborés pour résister à des sollicitations mécaniques ou des agressions chimiques ou une combinaison des deux. Pour résister à ces sollicitations et/ou agressions, des éléments chimiques peuvent être ajoutés en plus du carbone **[MAT]**.

1.6.2 Béton :

Le béton est un matériau de construction composite fabriqué à partir de granulats naturels (sable, gravillons) ou artificiels (granulats légers) agglomérés par un liant. Le liant peut être qualifié d'« hydrique », lorsque sa prise se fait par hydratation. Les 4 ingrédients essentiels de tout béton sont le sable, la pierre concassée, le ciment et l'eau **[MAT]**.

Le choix des proportions de chacun des constituants d'un béton afin d'obtenir les propriétés mécaniques et de mise en œuvre souhaitées s'appelle la formulation. Plusieurs méthodes de formulations existent, dont notamment :

- La méthode Baron ;
- La méthode Bolomey ;
- La méthode de Féret ;
- La méthode de Faury ;
- La méthode Dreux-Gorisse.

La formulation d'un béton doit intégrer avant tout les exigences de la norme NF EN 206-1, laquelle, en fonction de l'environnement dans lequel sera mis en place le béton, sera plus ou moins contraignante vis-à-vis de la quantité minimale de ciment à insérer dans la formule ainsi que la quantité d'eau maximum tolérée dans la formule. De même, à chaque environnement donné, une résistance garantie à 28 jours sur éprouvettes sera exigée aux producteurs, pouvant justifier des dosages de ciments plus ou moins supérieurs à la recommandation de la norme, et basée sur l'expérience propre à chaque entreprise, laquelle étant dépendante de ses matières premières dont la masse volumique peut varier, notamment celle des granulats.

1.6.3 Aluminium :

L'aluminium est un élément chimique, de symbole Al et de numéro atomique 13. C'est un métal pauvre, malléable, de couleur argentée. Il est remarquable pour sa résistance à l'oxydation et sa faible densité. C'est le métal le plus abondant de l'écorce terrestre et le troisième élément le plus abondant après l'oxygène et le silicium ; il représente en moyenne 8 % de la masse des matériaux de la surface solide de notre planète. L'aluminium est trop réactif pour exister à l'état natif dans le

milieu naturel : on le trouve au contraire sous forme combinée dans plus de 270 minéraux différents, son minerai principal étant la bauxite, où il est présent sous forme d'oxyde hydraté dont on extrait l'alumine. Il peut aussi être extrait de la néphéline, de la leucite, de la sillimanite, de l'andalousite et de la muscovite.

L'aluminium est un métal mou, léger, mais résistant avec un aspect argent-gris mat, dû à une mince couche d'oxydation de cinq à dix nanomètres qui se forme rapidement quand on l'expose à l'air et qui empêche la corrosion de progresser dans des conditions normales d'exposition chimiques **[MAT]** .

1.7 Les sections dans une structure :

1.7.1 Définition d'une section :

Une section est une vue en coupe, mais limitée à celle-ci, c'est-à-dire sans représentation de ce qui se trouve derrière ou en avant de cette coupe, et une section plane est l'intersection d'un solide ou d'une surface avec un plan.

1.7.2 Type de section :

Dans une structure, on utilise divers sections mais on va parler que des sections utilisées dans notre projet :

- **Section rectangulaire :**

  Elle est caractérisée par une forme rectangulaire et qui a comme dimension la longueur (L) et la largeur (l).

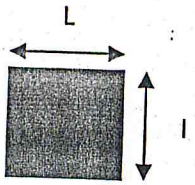


Figure 1.5 Section rectangulaire à arêtes vives.

Figure 1.6 Section rectangulaire avec bords arrondis.

Cette section est utilisée généralement avec le matériau béton frais associé à de l'acier qui permetten d'obtenir le béton armé « un matériau de construction courant », on appelle cette acier des barres d'armature (en anglais rebar) utilisée pour le renforcement du béton parce que Le béton est un matériau très résistant en compression, mais faible en traction. Pour compenser ce déséquilibre, on incorpore dans la masse de béton des barres d'armature destinées à reprendre les efforts de traction, et possède une surface nervurée pour améliorer son adhérence. Sa distribution peut être rectangulaire ou circulaire.

Figure 1.7 Section rectangulaire avec une distribution rectangulaire de renforcement.

Figure 1.8 Section rectangulaire avec une distribution circulaire de renforcement.

- **Section circulaire :**

Elle est caractérisée par une forme circulaire et qui a comme dimension son diamètre (D).

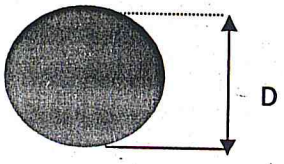


Figure 1.9 Section circulaire.

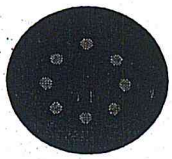


Figure 1.10 Section circulaire avec renforcement.

- **Section tube :**

Elle est caractérisée par une forme rectangulaire mais vide à l'intérieur et qui a comme dimension sa longueur (L), sa largeur (l) et son épaisseur(W).

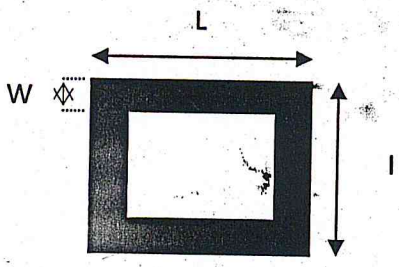


Figure 1.10 Section tube.

- **Section pipe :**

Elle est caractérisée par une forme circulaire vide à l'intérieur et qui a comme dimension son diamètre (D) et son épaisseur(W).

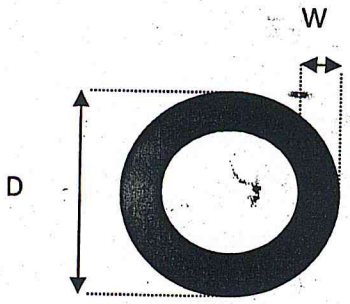


Figure 1.11 Section pipe.

- **Section Poutrelles (Section i):**

Elle est caractérisée par une forme polygone à 12 sommets.

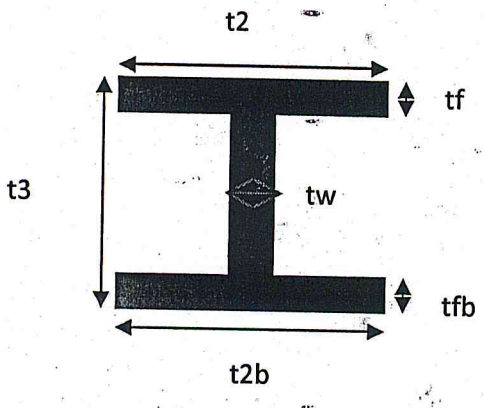


Figure 1.12 Section Poutrelles.

- **Section Cornières (angle ou L):**

Figure 1.13 Section Cornières à ailes égales.

Figure 1.14 Section Cornières à ailes inégales

- **Section tee (T)**

Figure 1.15 Section Tee à ailes égales.

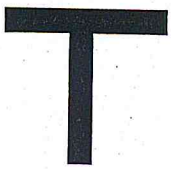


Figure 1.16 Section Tee à ailes inégales.

- **Section channel :**

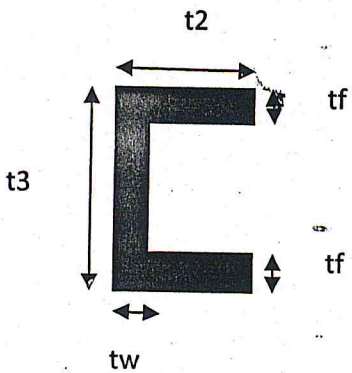


Figure 1.17 Section channel

1.8 Systèmes de forces et couples:

1.8.1 Définition d'une force :

A. Définition statique : Une force est une cause capable de maintenir un corps au repos ou de le déformer **[ADS]**.

B. Définition dynamique : Une force est une cause capable de provoquer ou de modifier le mouvement d'un corps **[ADS]**

Le vecteur force :

Un représentant du vecteur force est caractérisé par 4 éléments :

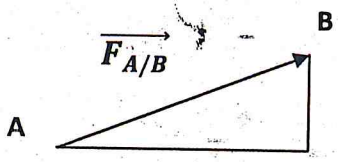


Figure 1.17 Représentation d'une force F

- la direction : orientation de la force
- le sens : vers où la force agit
- la norme : grandeur de la force, elle est mesurée en newton (N)
- le point d'application : endroit où la force s'exerce

Une force est complètement définie par ces quatre données

Systèmes de forces :

A. Forces concourantes

On dit que les forces sont concourantes quand leurs lignes d'actions se coupent au même point, leur résultante se calcule par les formules suivantes :

$$R_x = \sum F_x \quad \text{et} \quad R_y = \sum F_y \quad \text{et} \quad R_z = \sum F_z$$

$$R = \sqrt{(R_x^2 + R_y^2 + R_z^2)}$$

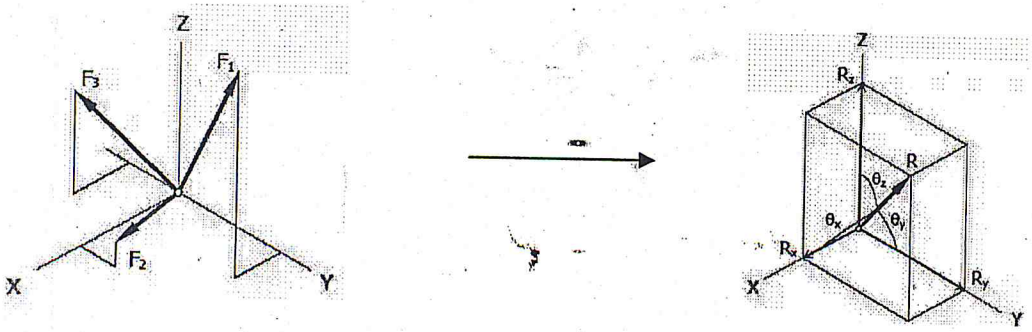


Figure 1.18 Forces concourantes.

B. **Forces parallèles**

On dit que les forces sont parallèles quand leurs lignes d'actions sont parallèles et leur résultante est égale à leur somme algébrique.

$$R= \sum_{i=0}^{i=n} Fn$$

1.8.2 Définition d'un moment:

Le moment d'une force est l'aptitude d'une force à faire tourner un système mécanique autour d'un point donné, que l'on nomme pivot **[ADS]**.

Le moment M de la force F par rapport à un point O se calcule par l'équation suivante :

$M_{F/O}=F * d$ d est dite le bras levier du moment.

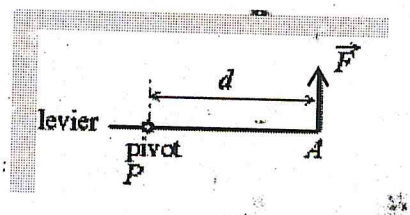


Figure 1.19 Représentation d'un moment

1.8.3 Définition d'un couple:

C'est deux forces égales, parallèles, sens opposé, séparées par une distance dite bras du couple, la valeur de ce couple (M) se calcule par l'équation suivante :

M =F * d

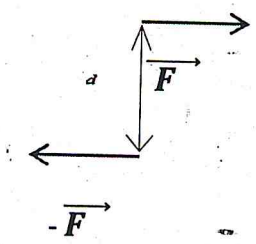


Figure 1.20 Représentation d'un couple

1.8.4 Condition d'équilibre :

Un solide est en équilibre si le système de forces qui lui sont appliquées, est égal à zéro **[ADS]**. Donc le solide dans l'espace pour qu'il soit en équilibre, les conditions suivantes doivent être vérifiées :

$$\sum R_x=0 \ , \ \sum R_y=0 \ , \ \sum R_z=0$$

$$\sum M_x=0 \ , \ \sum M_y=0 \ , \ \sum M_z=0$$

1.9 Réaction d'appui :

Les appuis sont destinés à bloquer les déplacements d'une structure. Dans l'espace, six déplacements sont possibles : trois translations et rotations suivants les axes X, Y et Z **[ADS]**.

Les appuis peuvent être :

a. **Appui simple:** le seul déplacement bloqué se trouve perpendiculaire à l'appui (pas de translation selon l'axe Z) **[ADS]**.

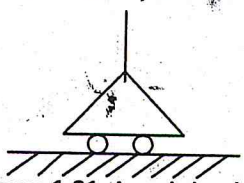


Figure 1.21 Appui simple.

b. **Appui double ou articulation :** les déplacements sont bloqués dans la direction perpendiculaire et parallèle à l'appui (pas de translation selon les axes X, Y, Z) **[ADS]**.

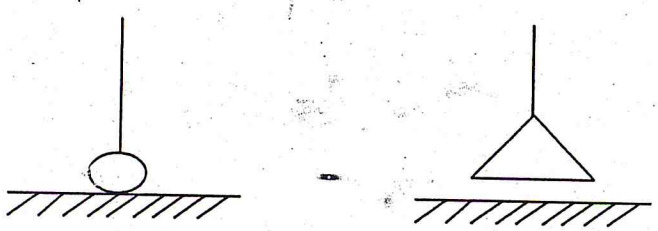


Figure 1.22 Appui double.

c. **L'encastrement :** tous les déplacements possibles dans l'espace sont bloqués **[ADS]**.

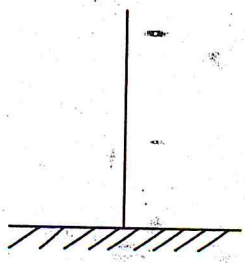


Figure 1.23 L'encastrement.

d. **La bielle:** elle accepte seulement les forces axiales, compression ou traction **[ADS]**.

Figure 1.24 La bielle.

1.10 Quelques logiciels de calcul de structures:

- **SAP (Structural Analysis Program)** a été développé par l'université de Berkley / Californie, depuis les années 60. Le premier logiciel fût le SAP 4, qui fonctionnait dans de gros systèmes informatiques (premier génération de calculateur). Ce logiciel a été utilisé dans la modélisation mathématique (éléments finis) de structures complexes telles que : buildings, ouvrages d'art, aéronautique, l'aérospatial (NASA). L'introduction des données se faisaient par fichier, jusqu'aux années 90, une nouvelle génération est apparue. En effet, l'introduction des données d'une structure se faisait par une interface graphique, et notamment le SAP2000, qui a été réalisé par la société CSI (Computers and Structures,Inc). Ce logiciel, qui en est à sa quatorzième édition, est fréquemment utilisé par les ingénieurs de génie civil lors de la conception et le calcul de ponts, d'édifices, de barrages, de bâtiments industrialisés, bâtiment sociaux et autres équipements de génie civil.

- **Autodesk® Robot™ Structural Analysis Professional** est un logiciel de calcul et d'optimisation des structures. Il utilise la méthode d'analyse par les éléments finis pour étudier les structures planes et spatiales. Réalisé par Robobat. L'interopérabilité du logiciel Autodesk® Revit® Structure étend le processus de modélisation des données du bâtiment (BIM), permettant ainsi aux ingénieurs d'exécuter plus rapidement des calculs de bâtiments et d'ingénierie complèts sur de nombreuses structures.

- **ICAB** est une société qui développe le logiciel de calcul de structures ICAB basé sur la méthode des éléments finis, créé à Toulouse (France) en 1991, ses logiciels permettent de concevoir des structures selon les règles DTU ou Eurocodes.

1.11 Conclusion :

Notre avons présenté, dans ce chapitre, les différentes notions du domaine du génie civil que nous devons connaitre, pour implémenter notre logiciel, nous avons cité quelques logiciels de calcul de structures. Dans notre réalisation, nous avons essayé de nous inspirés des interfaces du SAP 2000.

La vérification par le calcul représente une des mesures possibles pour garantir la sécurité structurale. Une bonne connaissance des matériaux utilisés en construction est indispensable pour la réalisation d'une structure, aussi bien pour sa conception, son dimensionnement que lors de son exécution.

Chapitre 2

Notions mathématique et synthèse d'images

2.1. Introduction :

La synthèse d'images consiste à générer des images en utilisant des ordinateurs. Cela va des primitives graphiques de base, telles que le tracé d'une droite sur un écran d'ordinateur, au calcul d'images très complexes, telles qu'on en voit dans les effets spéciaux de certains films à gros budget. Dans ce dernier cas, les limites au réalisme peuvent venir des imperfections de la *modélisation de la réalité*, qui sont liées au problème de la puissance de calcul.

Ce chapitre décrit les modèles mathématiques et les algorithmes fondamentaux permettant de générer des images 2D ou 3D ; ces méthodes sont à la base de toute conception d'un logiciel d'infographie.

Bien que notre propos soit orienté vers la 3D, il nous a paru utile d'exposer quelques principes d'infographie 2D, principalement un algorithme de traçage de droite du type Bresenham et un algorithme de remplissage de polygone, dans une image formée de pixels. Ces primitives graphiques interviennent, en effet, dans de très nombreuses techniques graphiques. On verra par exemple l'utilité de la primitive de remplissage de polygone lors de l'étude de l'algorithme du z-buffer.

La partie 2 est consacrée à la modélisation géométrique. La modélisation consiste en un ensemble d'outils permettant de construire et de coder des objets graphiques, puis des scènes plus ou moins complexes.

Enfin ce chapitre explique comment spécifier un point de vue quelconque sur une scène3D, et des techniques permettant la navigation dans une scène (déplacement et rotation d'une camera).

Vous remarquerez qu'il n'ya pas de référence bibliographique citer dans ce chapitre, c'est parce que ce chapitre a été pris depuis un même livre [SIA3D]. On a tiré les concepts fondamentaux sur la 3D.

2.2 Images 2D et quelque Primitives graphiques :

2.2.1 Dessiner une droite :

2.2.1.1 Droite et segments dans le plan :

Repère, coordonnées cartésiennes dans le plan :

Le plan est muni d'un repère orthonormé$(o, \vec{\imath}, \vec{\jmath})$. Ainsi chaque point M du plan est muni de deux coordonnées (x_m, y_m) qui déterminent sa position : $\overrightarrow{OM} = x_m\vec{\imath} + y_m\vec{\jmath}$. nous écrirons M=$(x_m y_m)$ et nous identifierons complètement le point M à ses coordonnées. En d'autre terme, le plan s'identifie à l'ensemble $\mathbb{R}^2$des couples (x, y) de nombre réels.

Pente d'un segment :

Considérons deux points distincts $A = (x_a y_a)$ et $B = (x_b y_b)$ du plan. Par ces deux points, passe une seule droite, que l'on désignera par D. Notant aussi $dx = (x_a - x_b)$, $dy = (y_a - y_b)$ les différences entre les coordonnées de A et B. supposant que dx est non nul, ce qui revient à dire que la droite D n'est pas verticale. On note alors $m = \frac{dy}{dx}$ .le nombre m s'appelle la pente de la droite D.

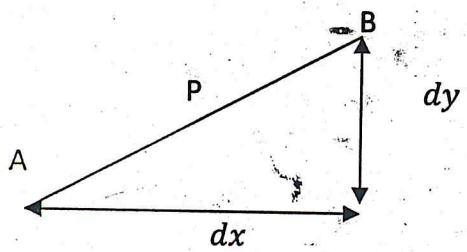


Figure 2.1 Le segment [A, B]

Equation cartésienne $F(x, y) = 0$:

Considérons maintenant L'équation (E)suivante :

$(E) : F(x, y) = 0$ Avec $F(x, y) = dy(x - x_a) - dx(y - y_a)$.

L'équation (E) est l'équation d'une droite.

On a $F(x_a, y_a) = 0.dy - 0.dx = 0$ et $F(x_b, y_b) = 0.dy - 0.dx = 0$

Autrement dit, les points A et B vérifient l'équation(E). l'équation (E) est précisemment l'équation de la droite D passant par A et B

Position d'un point par rapport à un segment :

Soit $M = (x_m, y_m)$ un point du plan qui ne ce trouve pas sur la droite D.On ce pose la question suivante : le point M ce trouve t'il à droite ou à gauche du segment [A,B] orienté de A vers B ?On peut répendre facilement a cette question on considérons l'équation (E) de la droite D donnée ci-

dessus, et plus précisément en évaluant $F(x_m, y_m)$ .en effet si $F(x_m, y_m) < 0$, le point M ce trouve a gauche du segment $[A, B]$ et si $F(x_m, y_m) < 0$ le point M ce trouve a droite du segment $[A, B]$.Notant que ce critère vaut aussi si $[A, B]$est verticale, c'est-à-dire si $dx = 0$.dans le cas ou $dx > 0$,ce critère permet de savoir si le point M ce trouve au-dessus de la droite D(on doit pour cela avoir $F(x_m, y_m) < 0$) ou au dessous de la droite D (on doit pour cela avoir $F(x_m, y_m) > 0$) .

Augmentation de y le long d'un segment :

Soit un point $P = (x_p, y_p)$ ayant pour abscisse $x_p = x_a + \Delta x$. Le point M doit vérifier l'équation (E) de la droite D, c'est-à-dire que $F(x_p, y_p) = 0$

On a donc

$$0 = F(x_p, y_p)$$

$$= dy(x_p - x_a) - dy(y_p - y_a)$$

$$= dy(x_a + \Delta x - x_a) - dy(y_p - y_a)$$

On en déduit que $dy.\Delta x = dx\,(y_p - y_a)$, c'est-à-dire que $y_p = y_a + \frac{dy}{dx}.\Delta x$, ou encore $y_p = y_a + m.\Delta x$

En d'autre termes, l'augmentation de y le long de la droite D, est proportionnel à l'augmentation de x, le coefficient de proportionnalité étant égale à la pente m de D.

2.2.1.2 Position du problème :

Les objets géométriques d'une image sont des approximations discrètes d'objets idéaux continus. Il faut des méthodes efficaces pour passer de l'objet idéal à sa discrétisation, laquelle sera représentée sur l'image.

On suppose qu'on dispose d'une primitive :

void PutPixel(int x,int y,unsigned char couleur) ;

Qui permet d'afficher le pixel (x, y) avec la couleur, et on désire dessiner dans un premier temps des segments de droite, plus précisément on se limitera ici à des segments de droites d'épaisseur 1. En d'autres termes, pour des pentes comprises entre –1 et 1, on veut obtenir un pixel sur chaque verticale. Pour des pentes de valeur absolue supérieure à 1, on veut un pixel sur chaque horizontale. Il nous faut donc un procédé rapide pour parcourir cet ensemble de pixels. La figure 2.2 représente une discrétisation de droite. Sur cette figure, les pixels sont représentés par des gros points sur une grille.

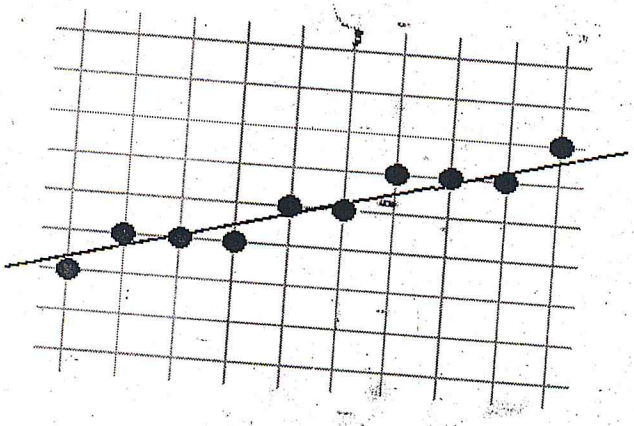


Figure 2.2 Exemple de segment de droite discrète.

On se place dans le cas d'une pente comprise entre 0 et 1, ce qui correspond au premier octant (figure 2.3). On fera de même pour d'autres valeurs de la pente. Dans le cas où l'on se place, on doit afficher exactement un pixel sur chaque verticale.

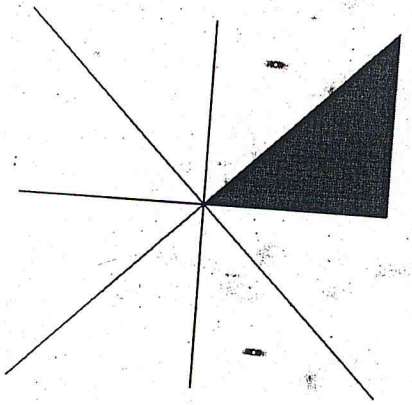


Figure 2.3 premier octant Hachurer

On dispose de la fonction d'arrondie suivante

```
Int Round (Float x){return (int)(x+0 ,5) ;}
```

Qui, à x associe la partie entière de x +0,5

On cherche à dessiner un segment de droite d'un point $A = (x_0, y_0)$ à un point $B = (x_0 + dx, y_0 + dy)$ avec x_0, y_0 , dx et dy entiers.

2.2.1.3 Algorithme NAÎF :

On note $m = \frac{dy}{dx}$ la pente réelle de la droite D passant par les points A et B. On a l'algorithme décrit au code 2.1, qui consiste à calculer, pour chaque valeur entière de x, la valeur y correspondante par une opération d'arrondi.

```
void dessin_droite(int x0, int y0, int dx, int dy, unsigned char couleur){
int x,y,z ;
float m=((float) dy)/((float)dx);   //pente de la droite
for(i=0 ;i<dx ;i++)
        {
        x=x0+i ;
        y=Round(y0+m*i) ;
        PutPixel(x,y,couleur) ;
        }
}
```

Code 2.1 Algorithme naîf de tracer d'une droite.

Cette méthode a de gros défauts : pour chaque pixel à dessiner, On effectue une multiplication en virgule flottante, une opération d'arrondi et une addition. C'est très couteux surtout la multiplication et l'arrondi. Nous pouvons déjà supprimer la multiplication en virgule flottante.

Si l'on pose $y_i = y_0 + m.i$ et $x_i = x_0 + i$, on doit afficher les pixels de la forme $(x_i, Round(y_i))$ pour $i = 0, \dots dx$. Or on a :

$$y_{i+1} = y_0 + m(i+1) = y_i + m$$

D'où la relation de récurrence :

$$y_{i+1} = y_i + m$$

Donc on peut calculer y_{i+1}apré avoir calculé y_i

On a supprimé le plus couteux : la multiplication entre flottants à chaque étape, il reste cependant une addition en virgule flottante et un calcul d'arrondi. Ce n'est pas l'objectif de notre projet.

2.2.2 Remplissage de polygones :

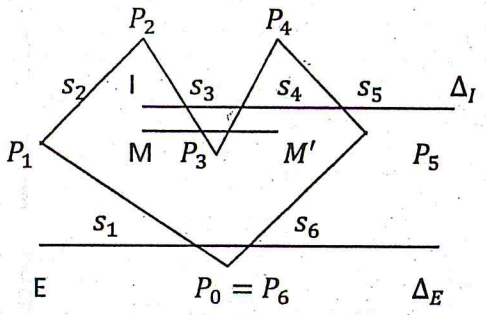
2.2.2.1 Polygones dans le plan :

Définition :

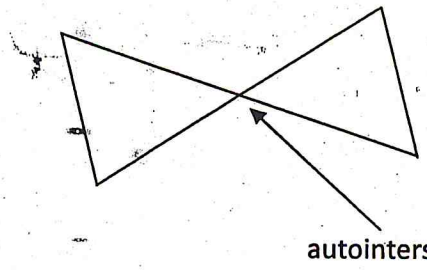
Un polygone Π dans le plan est une suite de segments $\Pi = (s_1, s_2, \dots, s_n)$, avec $n \in \mathbb{N}^*$ et pour $i = 1, \dots, n$ le segment s_i est de la forme $s_i = [P_{i-1}, P_i]$, ou P_i est un point du plan pour $i = 0, \dots n$, avec de plus $P_n = P_0$. Les points P_i pour $i = 1, \dots n$ s'appelles les sommets du polygoneΠ, et les segments s_i s'appellent les arrêtes du polygone Π.

Notons que pour $i = 2 \ldots n$ l'origine P_{i-1} du segment s_i est égale à l'extrémité du segment s_{i-1}. De plus l'origine P_0 du segment $s_1$est égale à l'extrémité $P_n$du segment s_n.Dans la figure 2.4 on peut voir un exemple de polygone avec 6 sommets (c'est-à-dire que n=6).

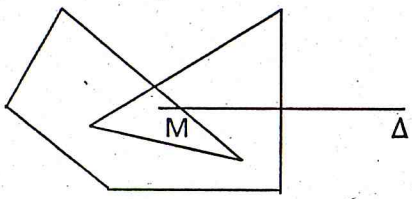
On dit qu'un polygone possède une autointersection si deux des sommets qui compose le polygone, qui ne sont pas des segments consécutifs, s'intersectent. Dans la Figure 2.4, on peut voir un polygone avec 6 sommets, qui possède une autointersection puisque les segments s_2 *et* s_4 , qui ne sont pas consécutif s'intersectent.



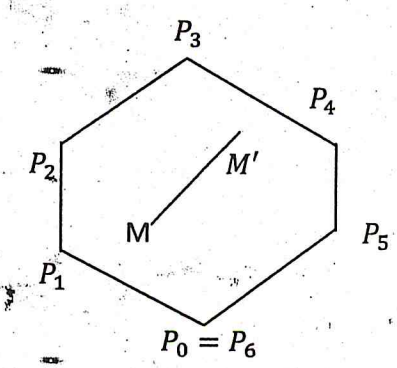
(a)Exemple de polygone



(b)Exemple de polygone avec autointersection



(c)Le point M est extérieur au polygone



(d) Exemple de polygone convexe

Figure 2.4 Exemple de polygones

Points intérieurs et point extérieurs :

Soit $\Pi = (s_1, s_2, \ldots, s_n)$ un polygone et soit M un point du plan qui ne se trouve sur aucun des segments du polygone Π. Considérant une demi droite Δ issu de M qui ne contient aucun des sommets du polygone Π. On dit que le point M est intérieur au polygone Π si la demi-droite Δ intersecte un nombre impair de segments du polygone Π . On dit que le point M est extérieur au polygone Π si la demi-droite Δ intersecte un nombre pair de segments du polygone Π.

Considérant les points I et E du plan représenté à la figure 2.4, le point I est intérieur au polygone Π, le point E est extérieur au polygone. On démontre que la parité du nombre d'intersection d'une demi-droite issue d'un point M avec les arêtes d'un polygone Π ne dépend pas de la demi-droite issue de M choisi. Cela montre que les notions de points intérieurs et points extérieurs sont bien définies de manière cohérente. Mais cette notion peut parfois paraitre surprenante pour des

polygones qui possède des autointersection, il n'empêche que cette notion dite fondée sur un critère d'indice est la plus couramment utilisée et la plus pratique à caractériser par des algorithmes.

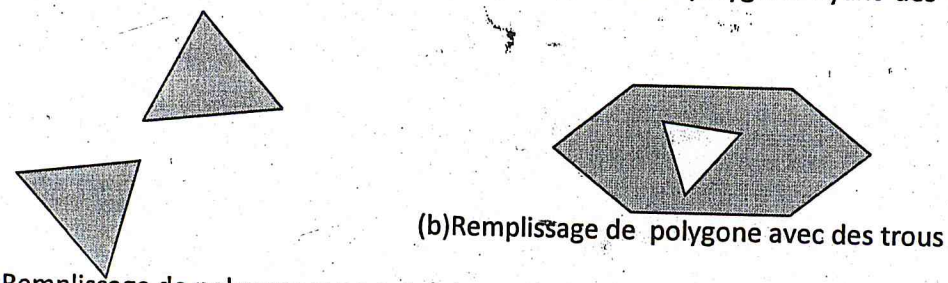
Polygone convexe :

Un polygone Π est dit convexe si pour les points M et M'qui sont intérieur au polygone Π, le segment $[M, M']$ est entiérement composé de points intérieurs au polygone Π. Le polygone représenté dans la figure 2.4.a n'est pas convexe. En revanche, le polygone représenté à la figure 2.4.d est convexe.

2.2.2.2 Principe du remplissage de polygone :

Considérant un polygone Π du plan. Le problème de remplissage consiste à trouver un algorithme permettant de parcourir tous les points du plan qui sont intérieurs au polygone.

Dans ce mémoire, on va simplement décrire un algorithme de remplissage de polygone qui fonctionne pour des polygones convexes, mais il existe des algorithmes généraux fonctionnant pour des algorithmes ayant des autointersection ou même pour des polygone ayant des trous (figure 2.5.b.)



(a)Remplissage de polygone avec autointersection

(b)Remplissage de polygone avec des trous

Figure 2.5 Exemple de polygones remplies

Le principe de remplissage consiste à calculer des segments horizontaux inclus dans le polygone. Par exemple, pour le polygone représenté à la figure 2.6, sur l'horizontale $y = 7$, on a deux segments horizontaux inclus dans le polygone.

Nous devons déterminer, pour chaque horizontale, quels pixels sont à l'intérieur du polygone, et les afficher avec la bonne couleur. En faisant cela itérativement pour toutes les horizontales, On remplit complètement le polygone.

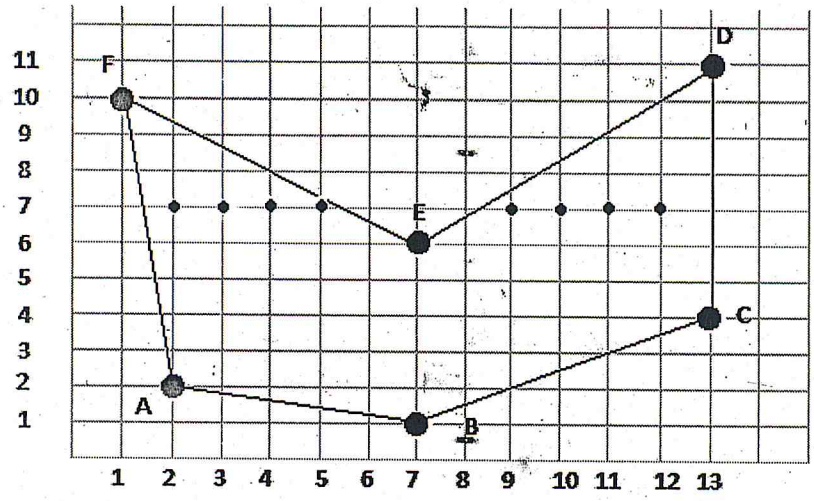


Figure 2.6 Exemple de remplissage de polygone

Le remplissage pour chaque ligne horizontale, sera décomposé en trois étapes :

1. Trouver les extrémités des segments horizontaux inclus dans le polygone. Ces extrémités correspondent à des intersections de la ligne horizontale avec les arêtes du polygone.

2. Trier les extrémités obtenues à l'étape 1 dans l'ordre des coordonnées x croissante.

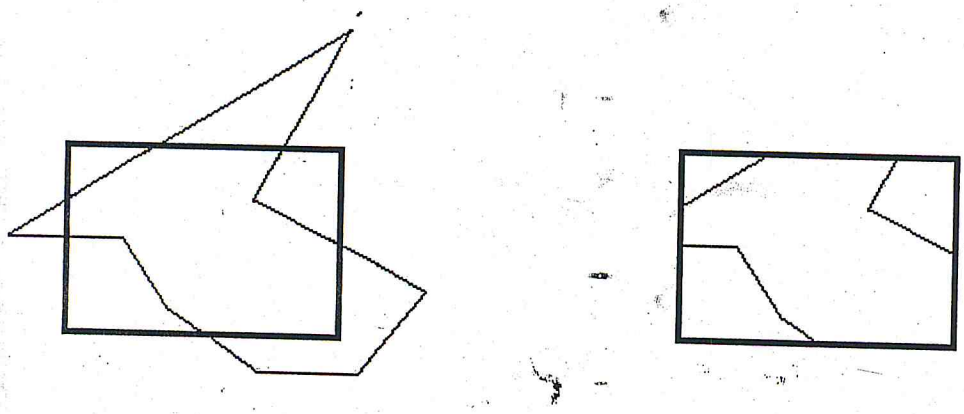
3. Afficher tous les pixels qui sont à l'intérieur du polygone entre paire d'extrémité. On utilise la règle de la parité pour déterminer qu'un pixel se trouve à l'intérieur du polygone. Initialement, on a un bit de parité égale à 0, puis on l'inverse pour chaque nouvelle extrémité. On dessine les pixels lorsque la parité est impaire (bit de parité égale à 1). Ce principe de parité est justifié par la définition même des points intérieurs au polygone.

2.2.2.3 fenêtrages :

Position du problème :

La question est la suivante : étant donnée un polygone Π à afficher dans une fenêtre (rectangulaire ou autre), comment ne parcourir que les partie du polygone qui sont à l'intérieur de la fenêtre ?

On ne va pas trop en détailler, voyant un exemple.



(a) le polygone à intersecter (b) le polygone à construire

Figure 2.7 Exemple de fenêtrage pour une fenêtre rectangulaire

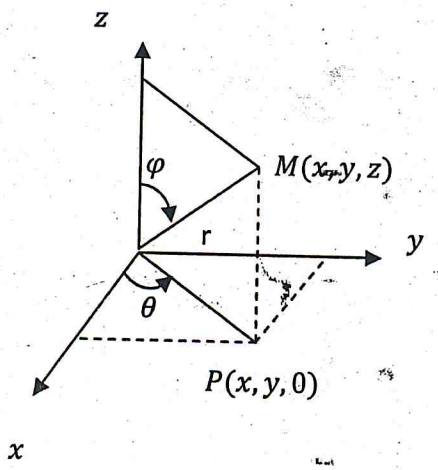


Figure 2.8 Les coordonnées sphériques (r, θ, φ) d'un point M.

2.3.2.2 Polyèdre :

Un polyèdre P dans l'espace tridimentrionnel $\mathbb{R}^3$ est la donnée de :

- Une suite de point $P_0, P_1 \ldots, P_{n-1}$ de $\mathbb{R}^3$ appelée sommet du polyèdre.
- Un ensemble de faces, chaque face étant une suite de numéro de sommet dans $\{0, \ldots, n-1\}$

Exemple :

Par exemple, considérant le tétraèdre construit sur les quatre points $O = (0,0,0)$, $A = (1,0,0)$, $B = (0,1,0)$, $C = (0,0,1)$ (figure 2.9). Le polyèdre correspondant est donnée de :

- Les sommets O, A, B
- Les quatre faces qui sont :
  - La face numéro 0 représentant le triangle OAB

- La face numéro 1 représentant le triangle OAC
- La face numéro 2 représentant le triangle OBC
- La face numéro 3 représentant le triangle ABC

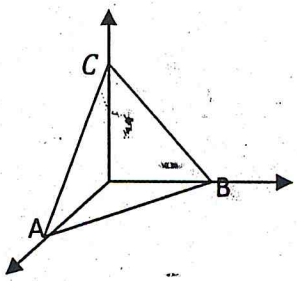


Figure 2.9 les quatre faces d'un tétraèdre

2.3.2.4 Sphère :

Définition de la sphère :

Soit O un point de l'espace, et soit r un élement de $\mathbb{R}^+$. On définie une sphère S_r de centre O, et de rayon r, l'ensemble des points Mde l'espace vérifiant $OM = r$. En d'autres termes, la sphère S_r est constituée de l'ensemble des points qui sont situés à distance r du point O.

Méridien et parallèle sur la sphère :

Un méridien sur la sphère s_r est un demi-cercle formé de l'ensemble des points M de coordonnées sphériques (r, θ_m, φ_m) tel que l'angle θ_m soit fixé égale à une certaine valeur. Le parallèle d'angle λ de la sphère $S_r$est le cercle constitué de l'ensemble des points M de S_r tels que $\varphi_m = \lambda$

Facettisation de la sphère :

Soit $m \geq 3$ et $p \geq 2$ deux nombres entiers. Nous allons découper la sphère S_r suivant m méridient et p parallèles. Soit $N = (0,0,r)$, appelé pôle nord et $S = (0,0,-r)$ appelé pôle sud. Tous les méridiens converge au pôle nord et au pôle sud, les facettes incidentes à N et S seront triangulaire. Toutes les autres facettes auront 4 sommets.

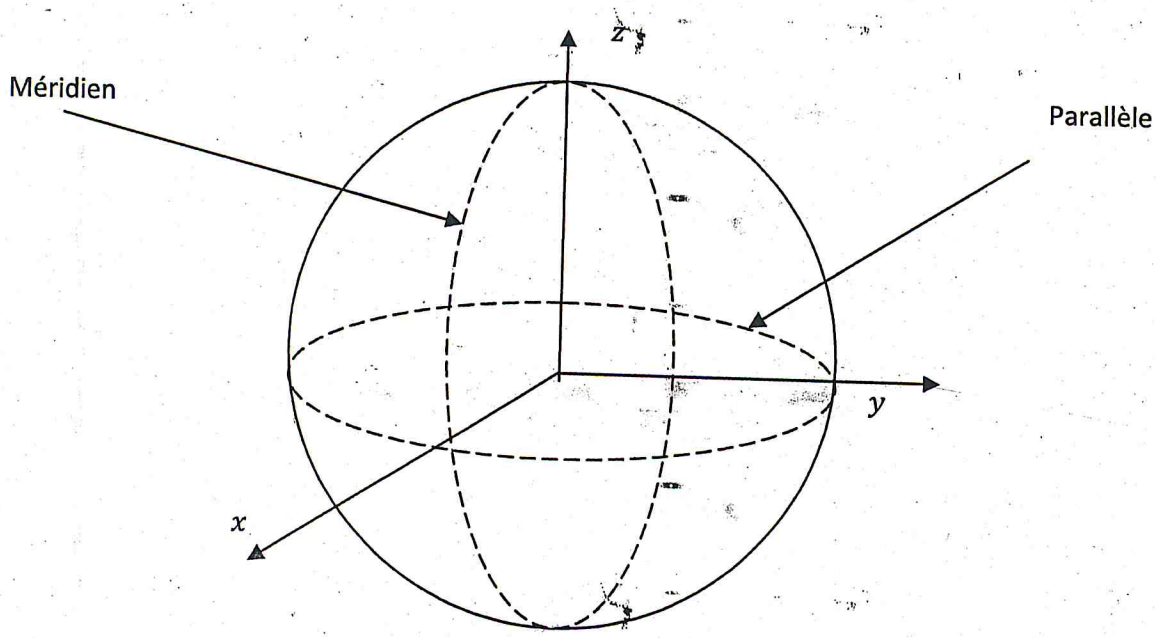


Figure 2.10 La sphère S_r

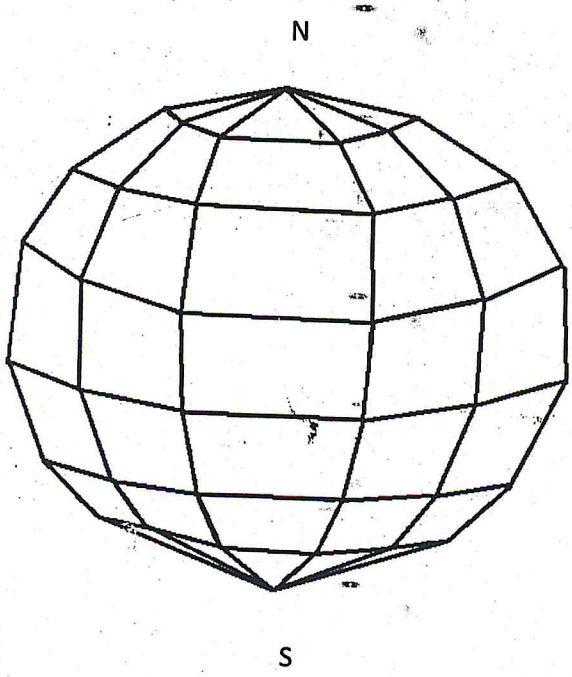


Figure 2.11 Les facettes du polyèdre

2.4 Affichage Et Navigation :

2.4.1 Elimination des parties cachées : Algorithme du z-buffer :

Pour réaliser une interface interactive permettant à un utilisateur de visualiser une scène sous différentes angles, il faut procéder à un affichage rapide. Dans cette partie, on décrit des solutions pour un affichage rapide, fondé sur l'algorithme du z-buffer pour éliminer les parties cachées.

2.4.1.1 Préliminaires sur les projections :

Nous voyons ici comment projeter des données 3D sur un plan dans le but de visualiser ces données. Nous considérons ici des projections très particulières, le plan sur lequel on projette étant perpendiculaire au troisième axe de coordonnées oz. Ceci correspondra à un point de vue très particulier d'un observateur placé sur l'origine O et regardant dans la direction de l'axe des z. Nous verrons par la suite comment ramener par des changements de repères appropriés, le cas d'un point de vue quelconque de l'observateur au cas particulier traité ici.

Projection parallèle :

La projection parallèle est la projection sur un plan parallèlement à une direction (figure 2.12). Cette projection correspond à un observateur placé à l'infini. Elle est simple mais pas très réaliste. Plaçons nous dans un repère $(o, \vec{\imath}, \vec{\jmath}, \vec{k})$ tel que le plan de projection soit le plan $z = 0$ et la direction de la projection soit parallèle au vecteur $\vec{k}$. Pour obtenir le projeté d'un point (x, y, z) il suffit d'oublier la coordonnée z : le projeté a pour coordonnées $x_p = x, y_p = y, z_p = 0$. Les formules de cette projection sont donc particulièrement simples.

La projection en perspective (caméra Pinhole) :

Dans le modèle de caméra Pinhole, il y a un centre de projection (figure 2.12) qui correspond à la position de l'observateur. Ce modèle de caméra est plus réaliste que le modèle avec projection parallèle. Par contre, les formules donnant les coordonnées du projeté (x_p, y_p, z_p) d'un point en fonction des coordonnées (x, y, z) du point sont un peu plus compliquées.

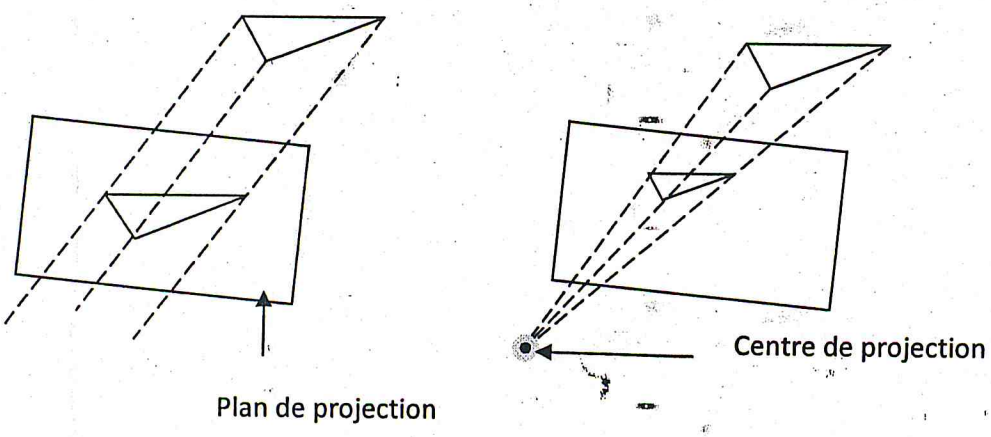


Figure 2.12 les deux principaux types de projection

Calcul des coordonnées du projeté d'un point

Plaçons-nous dans un repère $(o, \vec{i}, \vec{j}, \vec{k})$ lié à la camera, tel que le plan de projection soit le plan d'équation $z = d$ et tel que le centre de projection ait pour coordonnée (0.0.0) (figure 2.13). La troisième coordonnée z_p du projeté sera évidemment égale à 0, par définition du plan de projection.

D'après le théorème de Thalès, on a :

$$\frac{x_p}{d} = \frac{x}{z} \text{ et } \frac{y_p}{d} = \frac{y}{z}$$

En multipliant ces égalités par d on obtient :

$$\begin{cases} x_p = x * \frac{d}{z} \\ y_p = y * \frac{d}{z} \end{cases}$$

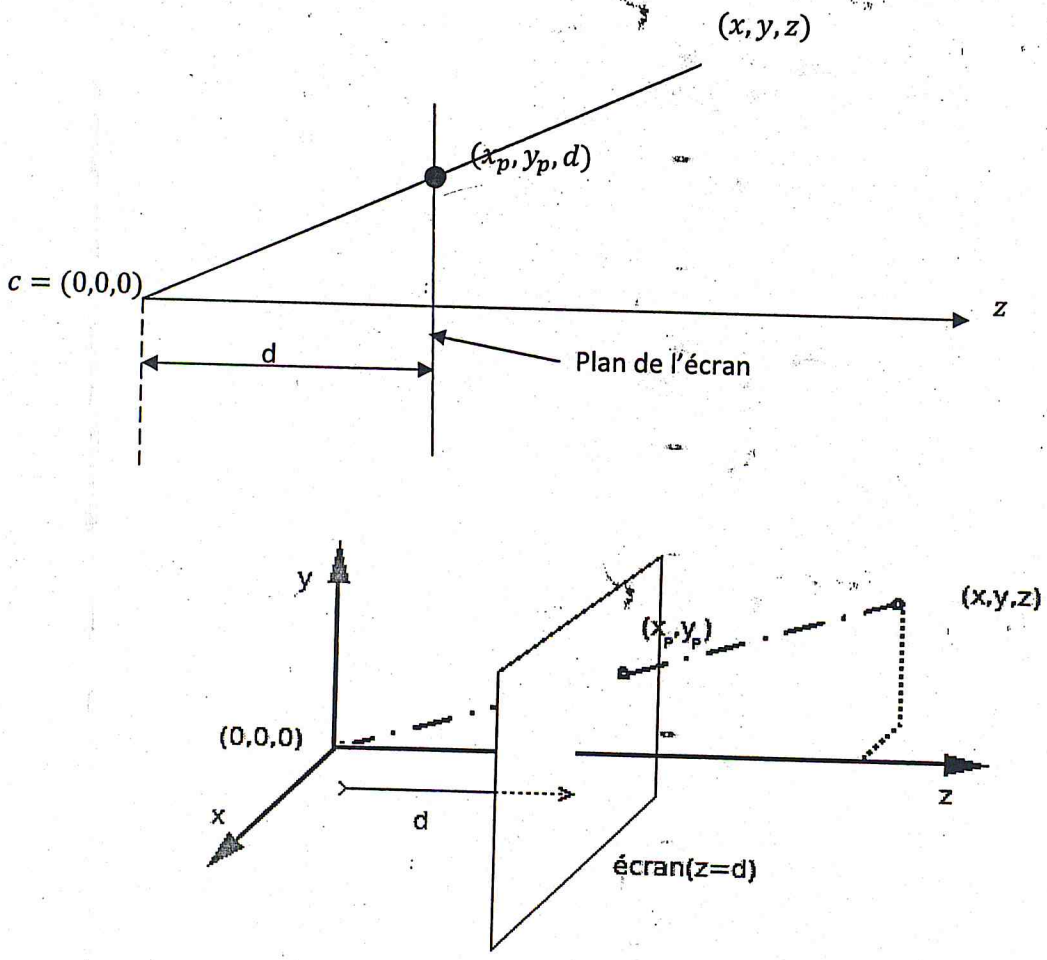


Figure 2.13 Calcul des coordonnées du projeté

Caméra Pinhole :

Une camera « Pinhole » est une spécification d'un point de vue de l'observateur avec projection en perspective. on se place ici dans le cas où la caméra est placée sur le point $= (0,0,0)$, la direction du visé étant l'axe des z. Nous avons vu comment calculer les coordonnées du projeté d'un point en fonction du paramètre d donnant l'équation $z = d$ du plan de projection. Cependant , dans une application graphique 3d, le paramètre d n'est pas données directement, mais doit être déterminé à partir de l'angle d'ouverture θ_c de la camera et de la largeur dim_x de l'image à calculer. L'angle d'ouverture θ_c est l'angle sous lequel l'observateur placé en 0=(0,0,0) doit voir le rectangle de largeur dim_x constitué par l'image à calculer, ce rectangle étant placer dans le plan $z = d$ centré au point $A = (0,0,d)$(figure 2.14). On peut calculer comme suit la valeur de d à partir de $\theta_c$et dim_x

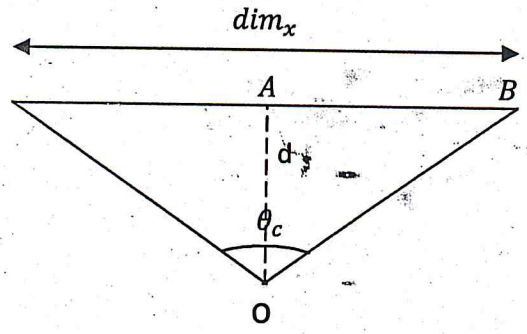


Figure 2.14 Calcul de la profondeur d du plan de l'écran

Soit $B = (\frac{dim_x}{2}, 0, d)$ le point de l'image à calculer situé le plus à droite par rapport à l'observateur. Le triangle OAB est rectangle en A, et l'angle en O de ce triangle est égale à $\frac{\theta_c}{2}$. On a donc :

$$d = OA = \frac{AB}{\tan(\frac{\theta_c}{2})} = \frac{dim_x}{2\tan(\frac{\theta_c}{2})}$$

2.4.1.2 Principe de l'algorithme du Z-buffer :

L'algorithme de z-buffer permet d'afficher des scènes dont les objets sont des polyèdres. Si une scène comprend d'autres types d'objets (sphères, quadriques...), il faut facettiser ces surfaces (c'est-à-dire les approximer par des polyèdres) pour utiliser l'algorithme de z-buffer. Notons que l'on peut adapter cette algorithme pour afficher directement des objets tel que des sphères, mais cette adaptation doit être faite pour chaque type d'objet, et il est donc plus commode de facettiser les objets.

Dans l'algorithme du z-buffer, on calcule une image discrète mémorisée dans un frame-buffer :

Unsigned char image [xmax][ymax];

Dans le cas d'une image couleur, il faut calculer non pas une, mais trois matrices pour les trois composantes rouges, verte et bleue.

L'idée de l'algorithme est de calculer, pour chaque pixel le point de profondeur minimale pour tous les polygones qui se projette sur ce point, ou plus exactement la couleur du polygone en ce point. Rappelons (code 2.2) l'algorithme de calcul d'un minimum de nombres réels.

```
double minimum(double *tab,int n)
{
double min=MAX_DOUBLE ;              /* valeur maximale pour un double */
int i ;

for (i=0 ; i<n ;  i++)
    If (tab[i]<min)  min=tab[i];     return min;
}
```

Code 2.2 l'algorithme de calcul d'un minimum

Dans le cas de l'algorithme du z-buffer, étant donné que l'on souhaite calculer simultanément un minimum pour chaque pixel de l'image, la variable min de l'algorithme (code 2.2) doit exister en autant d'exemplaires qu'il y a de pixels dans l'image à calculer. On introduit donc un « z-buffer » :

double minz [xmax][ymax]

Pour x=0,..., xmax-1 et y=0, ..., ymax-1, le nombre minz[xmax][ymax] correspondra au minimum de la profondeur z des points des objets de la scène rencontré qui se projettent sur le pixel(x, y).

Les valeurs de la matrice image sont initialisés à la couleur du fond (noir par exemple), et les valeurs du z-buffer minz sont initialisées à $+\bowtie$ (c'est-à-dire MAX_DOUBLE).

Ensuite l'algorithme du z-buffer peut être décrit comme suit :

Pour chaque polygone P de la scène (figure 2.15), on calcule tout d'abord la projection des sommets du polygone P dans le plan de l'écran (voir section pour le calcul des projections). On obtient, ainsi par la projection, un polygone dans le plan.

Pour chaque pixel(x, y) de l'intérieur de la projection de P sur le plan (cet ensemble de pixel est parcouru en utilisant un algorithme de remplissage de polygones tel que celui présenté précédemment), On calcule le point M du polygone P qui se projette sur le pixel(x, y).

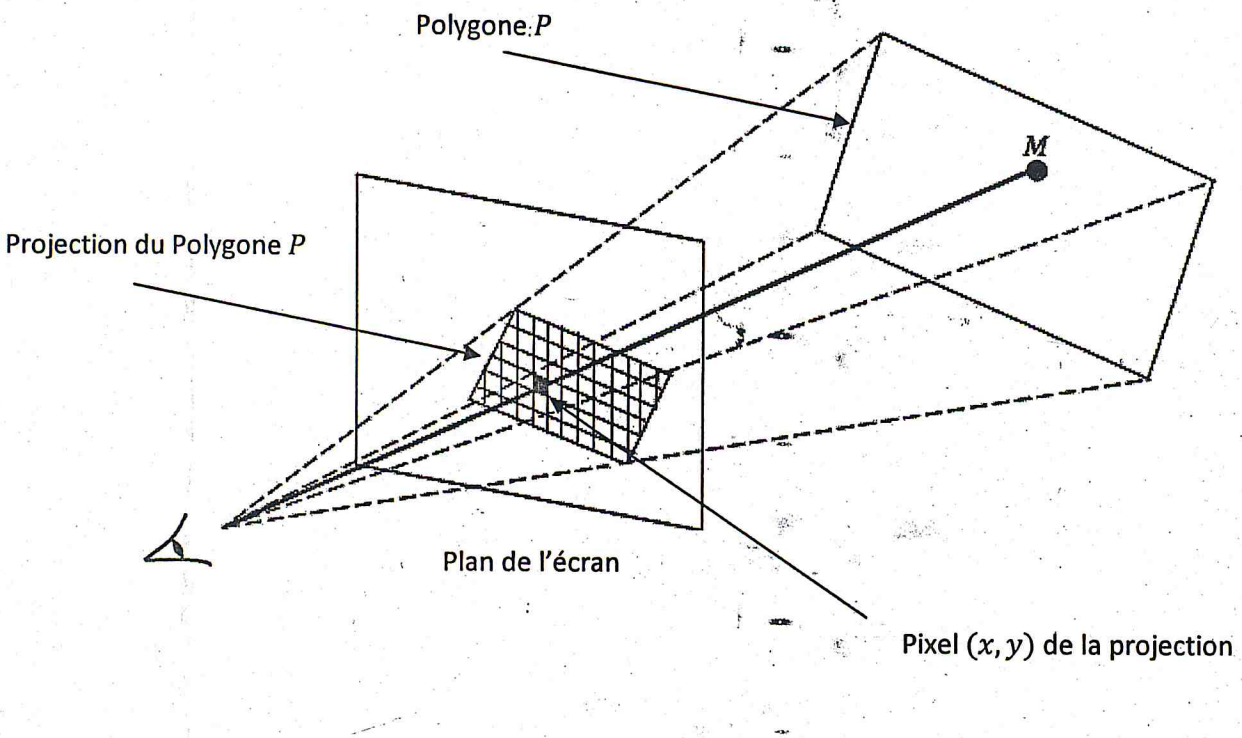


Figure 2.15 principe de l'algorithme du z-buffer

Soit M_z la troisième coordonnée (ou profondeur) du point M. Si $M_z$est inférieur à minz[x][y], le point M est plus proche de l'observateur que les points rencontrés jusqu'à présent .On met alors à jour le minimum minz[x][y]= M_z, et on met image[x][y] à la couleur du polygone P au point M considéré. Le pseudo-code de l'algorithme est donné (code 2.3) .

```
void zbuffer(void)
{
int x, y ;
for (x=0  ;  x<xmax ;  x++)                    /* initialization*/
   for (y=0  ;  y<ymax ;  y++){
      image[x][y]=0;                           /*fond noir*/
      minz[x][y]=MAX_DOUBLE ;
      }

Pour chaque polygone P de la scène{
  Calculer la projection P2D de P sur le plan de l'écran ;
  Appliquer un algorithme de fenêtrage au polygone P2D ;

  Pour chaque pixel (x,y) de l'intérieur de la projection de P2D {
    Point3D  M  =  point de P se projetant sur (x,y) ;
    Si  (M.z>0 et M.z<minz[x][y]){
       minz[x][y]=M.z ;
       image[x][y]=couleur de p au point M ;
      }
   }
}
```

Code 2.3 peudo-code de l'algorithme du z-buffer

2.4.1.3 implémentations dans le cas d'une projection en perspective :

Calcul des coefficients de l'équation du plan de la facette :

Soit $P = (p_0, \dots, p_n)$ un polygone dans l'espace 3D. Voyons comment calculer les coedicient A, B, C et D de l'équation $Ax + By + Cz + d = 0$ du plan contenant le polygone P.

Les coefficients A, B, C et D ne sont autres que les coordonnées de la normale au plan, et peuvent donc être calculées comme le produit vectoriel de deux vecteurs non parallèles contenus dans le plan.

Deux vecteurs du plan sont donnés par $\overrightarrow{P_0P_1}$ et $\overrightarrow{P_0P_2}$. Si par malheur ces deux vecteurs étaient parallèles, on peut essayer $\overrightarrow{P_0P_3}$ et ainsi de suite. Notant que pour savoir si deux vecteurs sont parallèles, il suffit de tester si leur produit vectoriel est nul (ou très petit). Une fois deux vecteurs du plan non parallèles obtenus, les coordonnées (A, B, C) de leur produit vectoriel sont les coordonnées de la normale au plan, et sont donc les trois premiers coefficients de l'équation de ce plan.

Pour calculer D le quatrième coefficient, il suffit d'écrire que le point $p_0 = (x_0, y_0, z_0)$ (par exemple) appartient au plan :

$$Ax_0 + By_0 + Cz_0 + D = 0$$

Les coefficients A, B, C et les coordonnées (x_0, y_0, z_0) de P_0 étant connus, on en déduit la valeur de D.

Calcule du point M :

Expliquant comment calculer les cordonnées du point M d'un polygone P se projetant sur un pixel (x, y) du plan de l'écran. On suppose que les coordonnées du sommet du polygone sont exprimés dans un repère lié à la caméra, tel que le centre de projection est le point(0,0,0), et le plan sur lequel on projette la scène est le plan d'équation $z = d$ (figure 2.15). Le pixel (x, y), dans l'espace correspond donc au point(x, y, d).

Pour un pixel(x, y)de la projection, il faut donc déterminer le point 3D $M = (M.x, M.Y, M.z)$ du polygone 3D P qui se projette sur (x, y, d)sur le plan de projection. Ce point M est l'intersection de la demi-droite Δ issue du centre de projection O et passant par le point (x, y, d) avec le plan contenant le polygone 3D P (figure 2.15). Cette intersection se calcule aisément à partir des coefficients de l'équation cartésienne $Ax_0 + By_0 + Cz_0 + D = 0$ du plan contenant P.

En effet, considérant l'équation paramétrique $M(t) = O + t(x, y, d)$, avec $t > 0$, de la demi-droite Δ. cela signifie que tout point de Δ est de la forme $M(t) = o + t(x, y, d)$ pour un certain $t > 0$. En particulier, le point M recherché est de la forme $M = M(t_0) = t_0(x, y, d)$ pour un certain t_0, qu'il nous faut calculer.

Le point M est aussi un point du plan contenant la facette P, et doit donc vérifier l'équation $Ax + By + Cz + D = 0$ de ce plan. En substituant dans cette équation les coordonnées de $M = M(t_0)$, il vient :

$$A(t_0x) + B(t_0y) + C(t_0d) + D = 0$$

Soit :

$$t_0 = \frac{-D}{Ax + By + Cd}$$

De la valeur de t_0, on déduit les coordonnées du point $M = M(t_0)$

Une fois le point M calculé, la couleur du pixel (x, y) est image[x][y] ,ou bien déterminer en fonction de modèles d'illumination si une source lumineuse existe .

2.4.2 Changement de repère et navigation :

2.4.2.1 Changement de repère :

En synthèse d'images, on a, en générale, plusieurs repères (repère global fixe, repère lié à un objet, lié à une caméra...)

Supposant que l'on ait les coordonnées (x, y, z) d'un point dans un repère$(o, \vec{\imath}, \vec{\jmath}, \vec{k})$. Considérant par ailleurs un autre repère $(\alpha, \vec{\imath_1}, \vec{\jmath_1}, \vec{k_1})$.tel que :

$$\begin{cases} \vec{\imath_1} = a_{1.1}\vec{\imath_1} + a_{1.2}\vec{\jmath_1}\, a_{1.3}\vec{k_1} \\ \vec{\jmath_1} = a_{2.1}\vec{\imath_1} + a_{2.2}\vec{\jmath_1}\, a_{2.3}\vec{k_1} \\ \vec{k_1} = a_{3.1}\vec{\imath_1} + a_{3.2}\vec{\jmath_1}\, a_{3.3}\vec{k_1} \end{cases}$$

Et $\overrightarrow{o\alpha} = \alpha_x\vec{\imath} + \alpha_y\vec{\jmath} + \alpha_z\vec{k}$

On souhaite, connaissant les coordonnées du point (x,y,z) dans le repère $(o, \vec{\imath}, \vec{\jmath}, \vec{k})$, calculer les coordonnées (X,Y,Z) du même point dans l'autre repère .

Cas général :

Notant M= $(a_{i,j})_{i,j=1,2,3}$et $(M^T)^{-1}$ l'inverse de la transposée de cette matrice M. Notant que la matrice de passage, telle qu'elle est classiquement définie, est la transposé $P = M^T$de la matrice M.

A\ Transformation d'un point :

Les nouvelles coordonnées (X, Y, Z)du point (x, y, z)sont données par

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = (M^T)^{-1}.\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix}\right)$$

B\ transformation d'un vecteur :

Soit $\vec{V} = (x_v, y_v, z_v)$ un vecteur exprimée dans le repère $(\vec{\imath}, \vec{j}, \vec{k})$. On souhaite maintenant calculer les coordonnées (X_v, Y_v, Z_v) du vecteur $\vec{V}$ dans le repère $(\alpha, \overrightarrow{\imath_1}, \overrightarrow{j_1}, \overrightarrow{k_1})$. On a :

$$\begin{pmatrix} X_v \\ Y_v \\ Z_v \end{pmatrix} = (M^T)^{-1} \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix}$$

Inversion des formules

Avec les notations précédentes on a :

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = (M^T). \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix}$$

Et

$$\begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = (M^T). \begin{pmatrix} X_v \\ Y_v \\ Z_v \end{pmatrix}$$

Cas de repère orthonormé direct :

Dans le cas où les repères sont tous les deux des repères orthonormés directs, la matrice M est une matrice de rotation, et on a donc $M^{-1} = M^T$. On a alors $(M^T)^{-1} = M$ et donc :

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M. \left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix} \right)$$

$$\begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = M. \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix}$$

2.4.2.2 Positionnement quelconque d'une caméra :

Repère lié à une caméra :

En synthèse d'images, on définit des cameras, qui définissent le point de vue à partir duquel l'observateur voit la scène. Chaque modèle de caméra correspond à un type de projection dans l'espace. Nous avons vu qu'il existe principalement 2 type de projections, la projection parallèle et la projection en perspective.

Lors de l'étude de l'algorithme de z buffer, nous avons toujours supposé que la caméra (ou l'observateur) était placé au point $O = (0,0,0)$, et regardait dans la direction de l'axe O_z. Cette hypothèse peut etre reformulée comme suit : les coordonnées qui définissent les objets (coordonnées des sommets de polyèdres), et les coordonnées des vecteurs normaux aux sommets de polyèdre, sont calculées dans un repère lié à la camera (figure 2.16).

Une scène 3D possède un repère fixe $(o,\vec{\imath},\vec{\jmath},\vec{k})$ dans lequel l'utilisateur définit les objets, les sources lumineuses et la position de la caméra. Ce repère s'appel le repère de la scène.

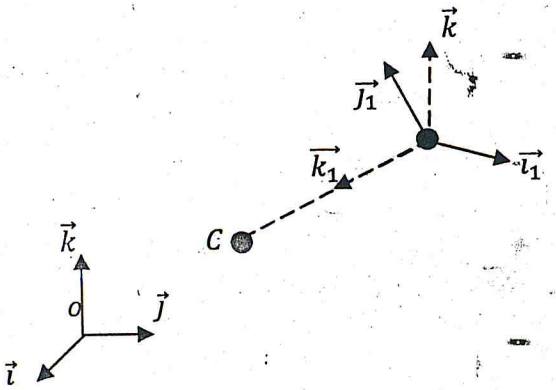


Figure 2.16 le repère lié à la caméra

La caméra possède un repère $(\alpha,\overrightarrow{\imath_1},\overrightarrow{J_1},\overrightarrow{k_1})$ qui lui est propre, appelé repère de la caméra, dont l'origine α coincide avec la position de la caméra, et tel que la direction $\overrightarrow{k_1}$ du troisiéme axe de coordonnées coincide avec la direction de visée de la caméra.

On associe donc à la caméra une matrice $M = (a_{i,j})_{i,j=1,2,3}$ tel que :

$$\begin{cases} \overrightarrow{\imath_1} = a_{1.1}\vec{\imath} + a_{1.2}\vec{\jmath}\, a_{1.3}\vec{k} \\ \overrightarrow{J_1} = a_{2.1}\vec{\imath} + a_{2.2}\vec{\jmath}\, a_{2.3}\vec{k} \\ \overrightarrow{k_1} = a_{3.1}\vec{\imath} + a_{3.2}\vec{\jmath}\, a_{3.3}\vec{k} \end{cases}$$

Rappelons que Mest la transposée de la matrice de passage du repère de la scène au repère de la caméra.

Initialisation d'une caméra :

Le but de cette section est d'expliquer comment on peut initialiser la matrice $M = (a_{i,j})_{i,j=1,2,3}$d'une caméra à partir de la position et de point de focalisation (le centre) de cette caméra.

Soit α la position d'une caméra et C un point, appelé le centre de la caméra, vers lequel est dirigé la caméra. Calculer les coefficients $(a_{i,j})$ de la matrice M associée à la caméra revient à calculer les coordonnées des vecteurs $\overrightarrow{\imath_1},\overrightarrow{J_1}$ et $\overrightarrow{k_1}$ dans le repère $(o,\vec{\imath},\vec{\jmath},\vec{k})$ de la scène.

Le vecteur $\overrightarrow{k_1}$ de la caméra , qui est le vecteur unitaire dans la direction de visé est égale à :

$$\overrightarrow{k_1} = \frac{\overrightarrow{\alpha C}}{\|\overrightarrow{\alpha C}\|}$$

Nous avons un choix pour définir les deuxième et troisième vecteurs du repère de la caméra. En effet, pour le moment, nous n'avons pas fixé exactement la position de la caméra, qui peut tourner au tour de l'axe αC. Nous allons fixer le vecteur $\overrightarrow{\imath_1}$ de sorte que l'axe des X du repère de la caméra

soit perpendiculaire à l'axe des z du repère de la scène. De cette manière, l'axe des z de la scène apparaitra vertical lors de l'affichage, ce qui est assez naturel.

Le vecteur $\overrightarrow{\imath_1}$ doit donc etre orthogonal au troisième vecteur $\vec{k}$ du repère de la scène. De plus, le vecteur $\overrightarrow{\imath_1}$ doit être orthogonal au vecteur $\overrightarrow{k_1}$ de la direction de visée, puisque nous souhaitons obtenir un repère $(\alpha, \overrightarrow{\imath_1}, \overrightarrow{J_1}, \overrightarrow{k_1})$ orthonormé. Nous n'avons pas beaucoup de choix, et nous définirons donc :

$$\overrightarrow{\imath_1} = \frac{\overrightarrow{k_1} \wedge \vec{k}}{\|\overrightarrow{k_1} \wedge \vec{k}\|}$$

Enfin le vecteur $\overrightarrow{J_1}$doit etre à la fois orthogonal à $\overrightarrow{\imath_1}$ et $\overrightarrow{k_1}$,de manière à ce que le repère $(\alpha, \overrightarrow{\imath_1}, \overrightarrow{J_1}, \overrightarrow{k_1})$ soit orthonormé direct . On a donc :

$$\overrightarrow{J_1} = \frac{\overrightarrow{k_1} \wedge \vec{k}}{\|\overrightarrow{k_1} \wedge \overrightarrow{\imath_1}\|}$$

Transformation des objets dans le repère de la caméra

Une fois la matrice M associé à la caméra initialisée, dans le but d'utiliser l'algorithme d'affichage du z-buffer expliqué précédemment, nous devons exprimer les coordonnées des sommets, ainsi que les vecteurs normaux aux sommets dans le cas d'une utilisation du modèle de Phong, dans le repère de la caméra.

Pour cela, on utilise simplement des formules expliquées pour calculer les coordonnées (X, Y, Z) des sommets dans le repère de la caméra à partir de ses coordonnées (x, y, z) dans le repère de la scène.

2.4.2.3 Navigation

En règle générale, la navigation consiste à modifier la position et l'orientation de la caméra. Comme les coordonnées des objets doivent être maintenues dans le repère de la caméra, ces coordonnées doivent être modifiées lorsque la caméra changé. En général, les nouvelles positions et direction de la caméra sont calculées en appliquant des rotations et des translations qui sont définies, soit dans le repère de la caméra, soit dans le repère de la scène.

Translation de la caméra

Supposant que soit donné un vecteur $= (X_v^c, Y_v^c, Z_v^c)$, les coordonnées de v étant exprimées dans le repère de la caméra. Nous souhaitons appliquer à la caméra une translation de vecteur v

Lorsque la caméra se déplace du vecteur v, les objets se déplacent du vecteur $-v$ par rapport à la caméra (tout est relatif !) . Les coordonnées des objets étant exprimées, de même que le vecteur v, dans le repère de la caméra, il suffit d'ajouter le vecteur $-v$ à tous les sommets pour obtenir les coordonnées de ces sommets dans le nouveau repère de la caméra.

Nous devons ensuite mettre à jour les données relatives à la caméra elle-même, à savoir sa position, le centre de la matrice Mqui lui est assiciée.

Les vecteurs $\overrightarrow{\imath_1}, \overrightarrow{\jmath_1}$ et $\overrightarrow{k_1}$qui définissent le repère de la caméra restent inchangés par une translation de celle-ci.par conséquent, la matrice M qui est définie à partir de ces vecteurs, reste aussi inchangée.

Concernant la position $pos = (x_p^s, y_p^s, z_p^s)$ de la caméra, cette dernière est exprimée dans le repère de la scène. On ne peut donc pas se contenter d'ajouter les coordonnées de , puisque celle-ci sont exprimée dans le repère de la caméra. Nous devons donc calculer les coordonnées (x_v^s, y_v^s, z_v^s) du vecteur v dans le repère de la scène en fonction des coordonnées (X_v^c, Y_v^c, Z_v^c) de ce même vecteur dans le repère de la caméra, en utilisant la matrice M de la caméra.

Concernant le centre de la caméra, qui est le point qui définit la direction de visée de la caméra, nous avons le choix entre deux types de navigation :

- La navigation « centrale », la caméra se déplace autour d'un centre fixe, qui est par exemple le centre de la scène. Cela permet à un utilisateur de tourner au tour d'un objet pour le visualiser sous différentes angles. Le centre de la caméra doit alors rester inchangé. Il faut alors s'assurer dès le départ que la coordonnée Z_v du vecteur v n'excède pas la distance entre la position et le centre de la caméra.
- La navigation de type « avatar » telle que les types de navigation employés en réalité virtuelle. Dans ce type de navigation, l'utilisateur se déplace dans une scène 3D, fixant la direction de visée de manière arbitraire. La position et le centre de la caméra se déplace avec l'avatar, censé représenter l'observateur qui se déplace dans une scène. Dans ce cas, le centre de la caméra doit subir la même transformation que la position de la caméra.

Rotations dans le repère de la caméra

Voici des matrices de rotation selon les 3 axes :

Une rotation d'un angle θ selon l'axe x s'écrit

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

Une rotation selon l'axe y s'écrit

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

Une rotation selon l'axe z s'écrit

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

Exemple de rotation dans le cas d'une rotation autour de l'axe des z :

La rotation autour de l'axe des Z peut être utilisée aussi bien pour une navigation centrale que pour une navigation avatar. Dans le cas d'une rotation autour de l'axe de visée de la caméra, c'est-à-dire l'axe entre la position et le centre de la caméra, qui coïncide avec le troisième axe de coordonnée du repère de la caméra, la procédure est un peu plus simple que dans le cas d'un axe

quelconque. En effet, dans ce cas, la position de la caméra reste inchangée, et nous n'avons qu'à mettre à jour la matrice M associée à la caméra et les coordonnées des objets de la scène.

Soit $R_c = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ La matrice de la rotation exprimée dans le repère de la caméra.

Nous devons tout simplement multiplier les positions des sommets de la scène, ainsi que les normales, par la matrice R_c pour leur faire subir la rotation.

La matrice M associée à la caméra doit être multiplié à gauche par la matrice R_c (c'est-à-dire que la matrice M est calculée comme $R_c.M$).

2.5 Conclusion

Dans la modélisation 3D, on a expliqué, un tout petit peu, comment approximer une sphère par un polyèdre. Les polyèdres sont très utilisés en infographie, notamment en raison de leur généralité et du très grand nombre de méthodes disponibles pour les traiter.

Toutefois, nous avons négligé beaucoup d'autres choses, on peut citer les courbes paramétrées, courbes de Béziers, et les courbes B-splines. Elles sont très utilisées dans la conception assistée par ordinateur, elles permettent de représenter des objets aux formes les plus variées. Les modèles de l'illumination ainsi que textures n'ont pas été abordées non plus.

Enfin, ce chapitre a été une introduction aux algorithmes qui permettent la modélisation en 3D. Il fait le tour des modèles et techniques incontournables du domaine.

- Infographie2D : tracé de droites, remplissage de polygones, fenêtrage.
- Modélisation géométrique.
- Affichage interactif : élimination des parties cachées.
- Navigation : déplacement d'une caméra dans une scène 3D.

Chapitre 3
Outil d'implémentation Java3D et moteurs 3D

3.1 Introduction :

Les représentations 3D sont aujourd'hui au cœur de l'univers du graphisme. Le Réalisme et la beauté des images sont de plus en plus éblouissants. L'éventail des Utilisations de la 3D comprend des domaines aussi divers que le cinéma (effet Spéciaux et films d'animations), le jeu vidéo, l'enseignement ou la publicité.

Les moteurs 3D sont la base de toutes ces utilisations, il existe différents outils Logiciels accessible aux développeurs de moteurs 3D, l'apparition de ces outils combinés à l'évolution matérielle des cartes vidéo ont permis à la 3D de se démocratiser, en simplifiant grandement la création de moteurs 3D tout en augmentant leur puissance.

Dans cette partie, nous présenterons les quelques outils existants, leurs caractéristiques communes, et utilisations possibles, ainsi que leurs différences, et nous détaillerons sur le moteur java3D.

3.2 Moteurs 3D

Qu'est-ce qu'un moteur 3D? C'est un ensemble d'algorithmes et de structures de données informatiques permettant la représentation et la manipulation d'un environnement tridimensionnel virtuel en vue de l'affichage bidimensionnel (à l'écran) d'une scène par une camera imaginaire (ou plusieurs...) placée(s) dans cet environnement.

Pour donner cette illusion de profondeur, un moteur 3D représente les objets du monde virtuel par un ensemble de polygones, un polygone pouvant être « décoré » par une(ou plusieurs) image(s) appelée(s) texture(s). La prochaine étape pour augmenter le réalisme d'une scène 3D est l'intégration des lumières dans le processus de rendu d'un objet, c'est ainsi que Henry Gouraud développa le célèbre modèle de *shading* qui porte son nom. Celui-ci fonctionne en trouvant la normale du vecteur de chaque sommet d'une face, calculant la couleur du pixel au sommet, puis en interpolant de façon linéaire cette couleur sur toute la face. Ensuite vient l'intégration du brouillard ou *fog*(changement de la couleur d'un pixel avec la distance à la camera) et des différents procédés de transparence ou *blend.* [Opdlm3d]

3.3 Présentation des outils

Dans cette partie nous allons présenter les 3 outils pour le développement de moteurs 3D que nous avons choisi : OpenGL, DirectX9.0 et Java3D. Pour chacun d'eux, nous présenterons leurs caractéristiques (Historique, Spécifications,...), et nous donnerons quelques exemples concrets d'applications les utilisant.

3.3.1 OpenGL

3.3.1.1 Caractéristique

OpenGL (Open Graphics Library) est une spécification qui définit une API multiplateformes pour la conception d'applications générant des images 3D (mais également 2D).Elle a vu le jour en 1992 pour succéder la bibliothèque de Silicon Graphics IRIS GL (SGI) qui était alors un standard pour les ingénieurs, créateurs d'effets spéciaux, bref pour tout ce qui touche de près ou de loin à de la CAO (*Conception Assistée par Ordinateur*). L'expérience de SGI dans ce domaine a fait d'OpenGL une interface facile d'utilisation, intuitive et surtout portable. Le projet Fahrenheit, initiative de Microsoft et de SGI, tenta d'unifier les interfaces OpenGL et Direct3D. Celui-ci apporta au début l'espoir de mettre de l'ordre dans le monde des APIs 3D, mais pour des contraintes financières de la part de SGI, le projet dût être abandonné. [Opdlm3d]

OpenGL est géré par un consortium *OpenGL Architecture Review Board* (appelé également *ARB*) qui détermine donc les modifications, ajouts et évolutions de la norme OpenGL. L'ARB est composé de membres volants tels que ATI, Nvidia, IBM, Intel, Sun, 3DLabs auxquels s'ajoutent des membres participants tels que Matrox ou encore Id-Software. Microsoft, l'un des membres fondateur, s'est retiré du consortium en mars 2003. [Opdlm3d]

3.3.1.2 Exemples d'application

Dans cette partie nous donnerons quelques exemples d'applications utilisant OpenGL. Le domaine dans lequel on retrouve le plus fréquemment cette librairie est évidemment le domaine des jeux vidéo. En effet, nombres d'entre eux utilisent OpenGL pour développer leurs propres moteurs 3D, comme par exemple :

-l'excellent moteur d'Unreal développé par EpicMegagames. Il est encore en évolution (actuellement *UnrealEngine3*) et utilisé par de très nombreux jeux (*Farcry,Fallen,DeusEx,...*) du fait qu'il est entièrement personnalisable, possédant son propre langage de script. Dans sa dernière version, il prend en compte toutes les techniques 3D les plus avancées comme le *DisplacementMapping*, le *Pixel/Vertex Shader*, et bien d'autres. [Opdlm3d]

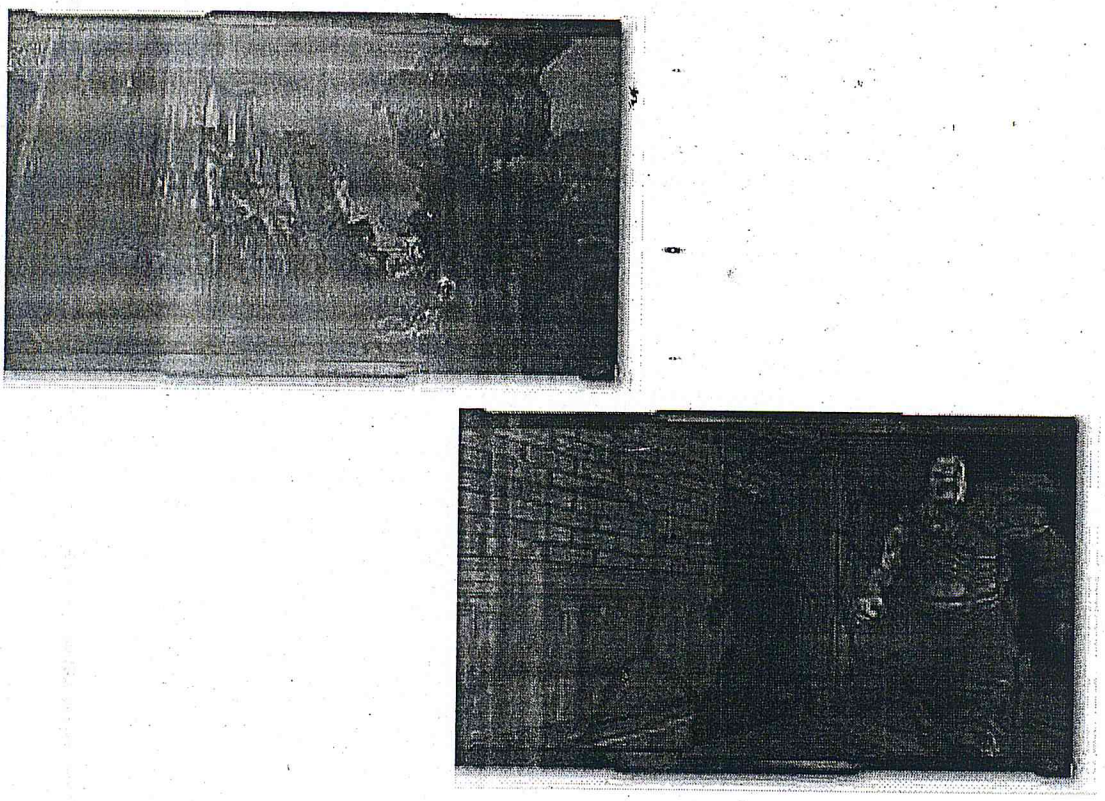


Figure 3.1 exemple d'application du moteur Unreal

Un autre domaine dans lequel OpenGL domine est la CAO(*Conception Assistée par Ordinateur*)utilisée par exemple pour les créateurs d'effets spéciaux de cinéma mais également pour les simulations diverses dans le domaine scientifique(exemple dans Formule1 pour l'aérodynamisme des voitures). Dans ce domaine deux logiciels principaux :

-LightWave3D : Même si sa version actuelle, v9.6, permet de développer en Direct3D, son implémentation OpenGL domine fortement. La puissance de son moteur 3D permet d'avoir des rendus d'une qualité inégalée (une résolution de 8000x6000) mais au détriment du temps de rendu qui peut aller jusqu'à plusieurs semaines pour une seule image ! [Opdlm3d]

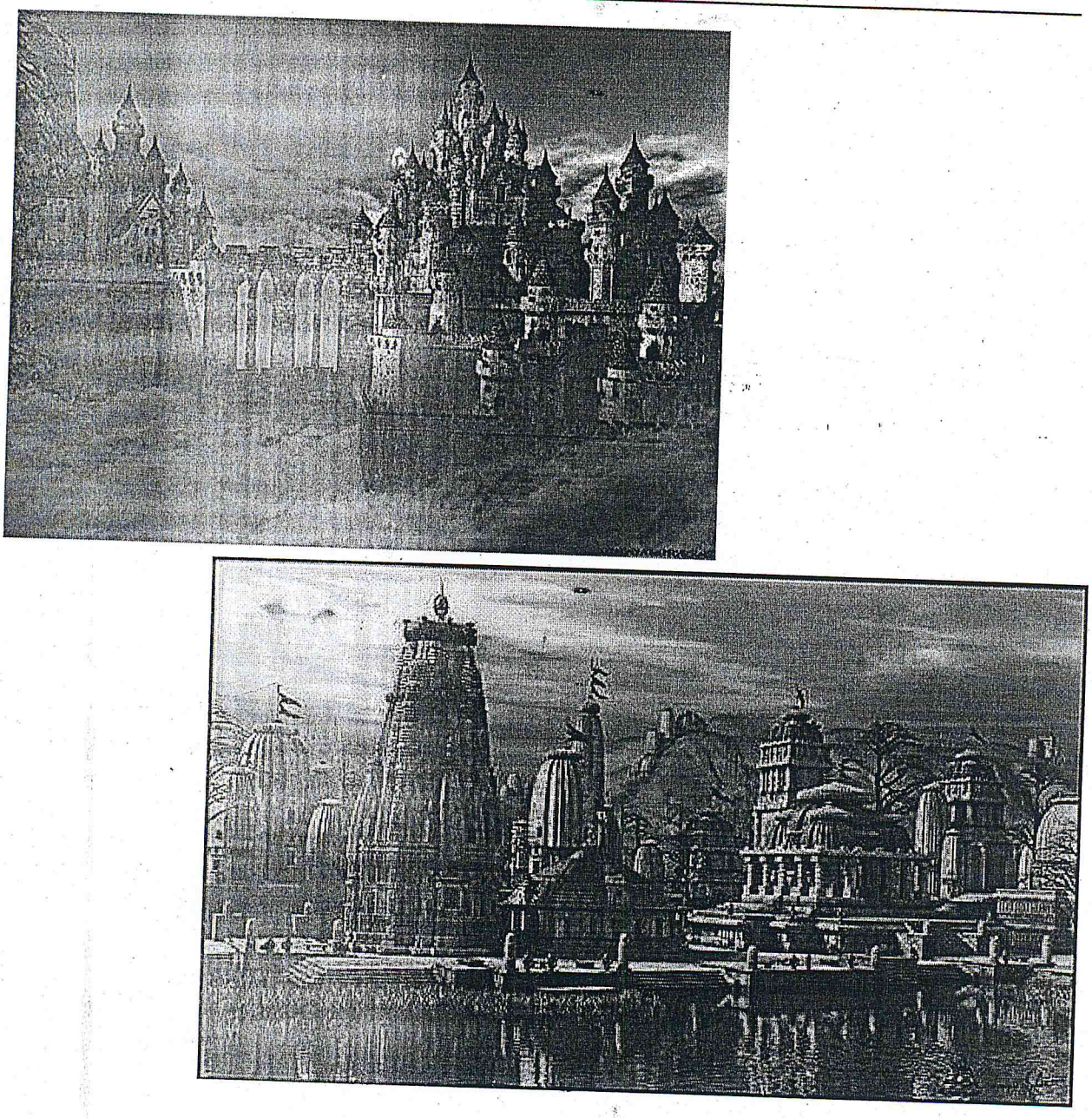


Figure 3.2 exemple d'application du moteur LightWave3D

3.3.2 DirectX

3.3.2.1 Caractéristiques

Initialement nommé Game SDK, DirectX est un ensemble de composants développés par Microsoft pour fournir aux applications Windows (jeux, applications multimédia, ...) un accès temps-réel et haute-performance au matériel disponible sur le système où elles sont exécutées. C'est une API, qui se place entre l'application et le matériel (carte graphique, carte son, etc...) formant une « surcouche » de Windows, évitant théoriquement aux programmeurs de devoir s'occuper des différences matérielles qui existent entre les différents PCs. Par exemple, si une carte vidéo n'a pas de fonctions dédiées à la 3D, DirectX demandera au processeur de s'occuper du rendu d'une image de synthèse ou le rendu 3D en temps réel. Mais c'est avec l'arrivée de Windows95 que le besoin d'une API générique s'est fait ressentir car à cette époque, l'essentiel des jeux étaient développés sous DOS qui ne comportait pas, contrairement à Windows, de nombreuses couches entre lui et le

matériel. C'est donc pour avoir des performances optimales quelque soit le matériel utilisé que les développeurs de jeux sous Windows ont créé DirectX. De nombreuses versions ont été développées jusqu'à ce jour puisque nous sommes actuellement à la version 11.0 et que Microsoft travaille toujours sur la prochaine version. DirectX est (évidemment!) la propriété de Microsoft qui possède tous les droits. Le produit n'étant pas libre, les sources ne sont pas rendues publiques, contrairement à OpenGL. Malgré cela, il devient un outil incontournable dans le domaine du développement de jeux vidéos sous Windows, Microsoft passant des accords technologiques avec les constructeurs de cartes 3D grand public. [Opdlm3d]

3.3.2.2 Exemples d'application

Contrairement à OpenGL, DirectX n'est quasiment utilisé que pour le développement de jeux vidéos même si de nombreuses applications comme les outils de CAO commencent a présenter une implémentation de DirectX en plus d'OpenGL. Une des plus grosses boites de développement utilisant grandement DirectX est *UbiSoft* dont le dernier moteur, utilisé pour le jeu *SplinterCell*, est celui d'Unreal (cité plus haut dans la section OpenGL) mais rendu compatible DirectX9.0 et prenant donc en compte les dernières implémentations de PixelShader3.0 et autres. [Opdlm3d]

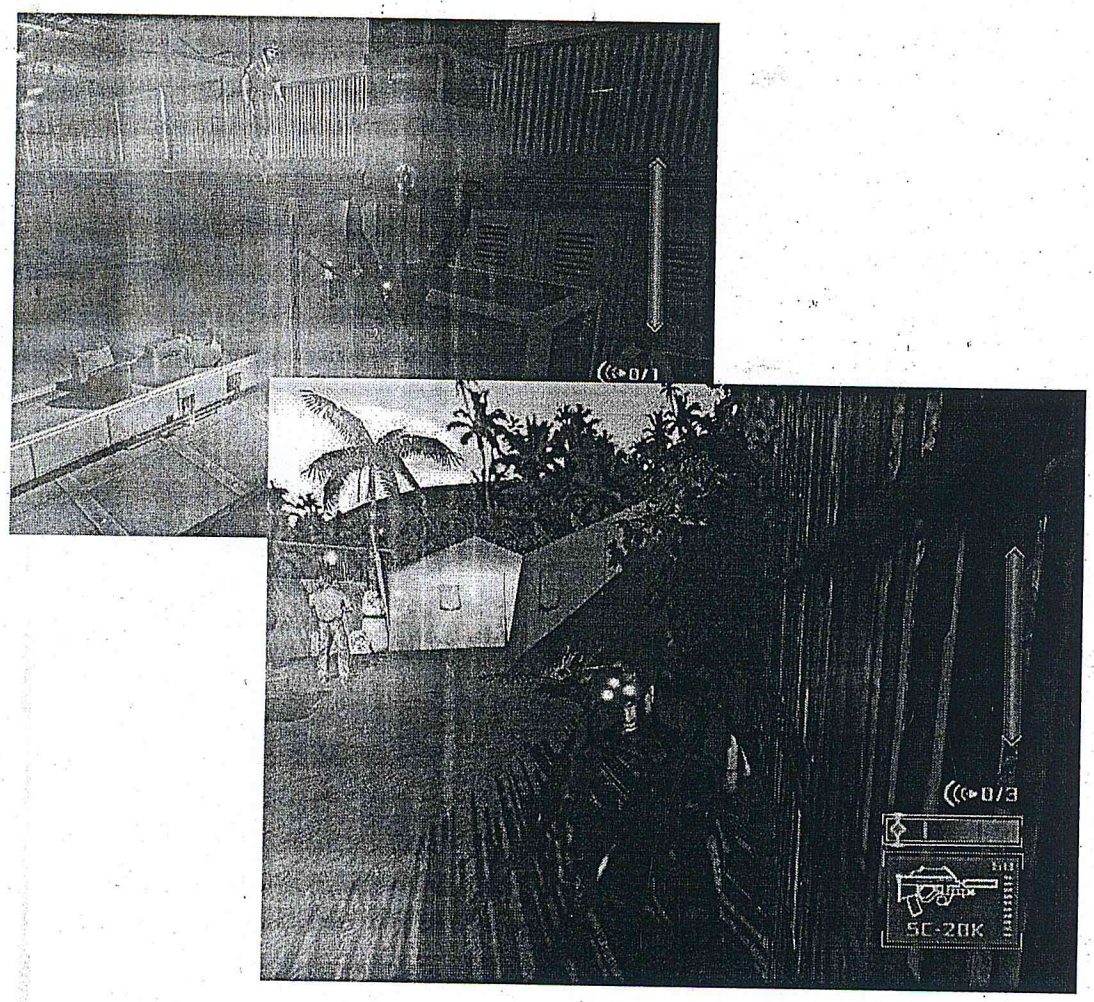


Figure 3.3 le jeu *SplinterCell*

3.3.3 JAVA 3D :

3.3.3.1 Caractéristiques :

Java3D est une bibliothèque de classes destinée à créer des scènes 3D (utilisation de formes complexes, d'éclairages, de textures, d'animations, de sons ...) développée par *SunMicrosystem* en 1998. Contrairement à OpenGL et DirectX qui ont chacun leur implémentation, Java3D possède un support natif OpenGL, DirectX et QuickDraw3D. [Opdlm3d]

Mais pourquoi développer Java3D alors qu'existe déjà OpenGL et DirectX ? Le but est de faciliter la programmation en disposant de fonctions de haut niveau ainsi que de permettre l'écriture d'applets contenant des scènes 3D. Java3D est aussi justifié par la portabilité de Java, ainsi une application ou applet peut être lancée avec le même code compilé dans toutes les plateformes ou Java3D existe (actuellement Solaris, SGI, HP-UX, Linux et Windows). [Opdlm3d]

3.3.3.2 Implémentation :

Java3D est une extension du langage Java destinée à créer des scènes 3D. Il se présente sous la forme d'une bibliothèque de classes (packages J3D). Comme indiqué plus haut, Java3D utilise OpenGL ou DirectX pour fonctionner, rajoutant un niveau d'abstraction entre les bibliothèques 3D de base et le logiciel. [Opdlm3d]

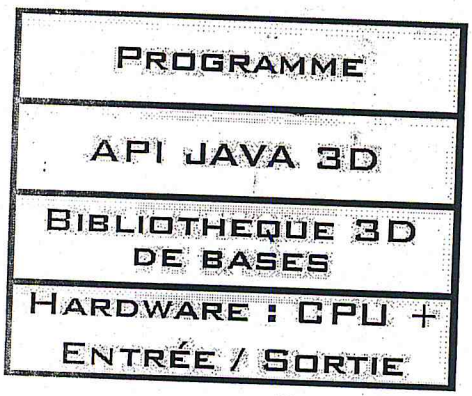


Figure 3.4 structure en couches

3.3.3.3 Exemples :

Peu d'applications industrielles sont développées avec Java3D. On observe plutôt des applications individuelles, du fait de la simplicité d'utilisation. Par exemple Java3D est très utilisée pour la création d'applets des sites scientifiques (représentations de molécules, simulations physiques, ...) [Opdlm3d].

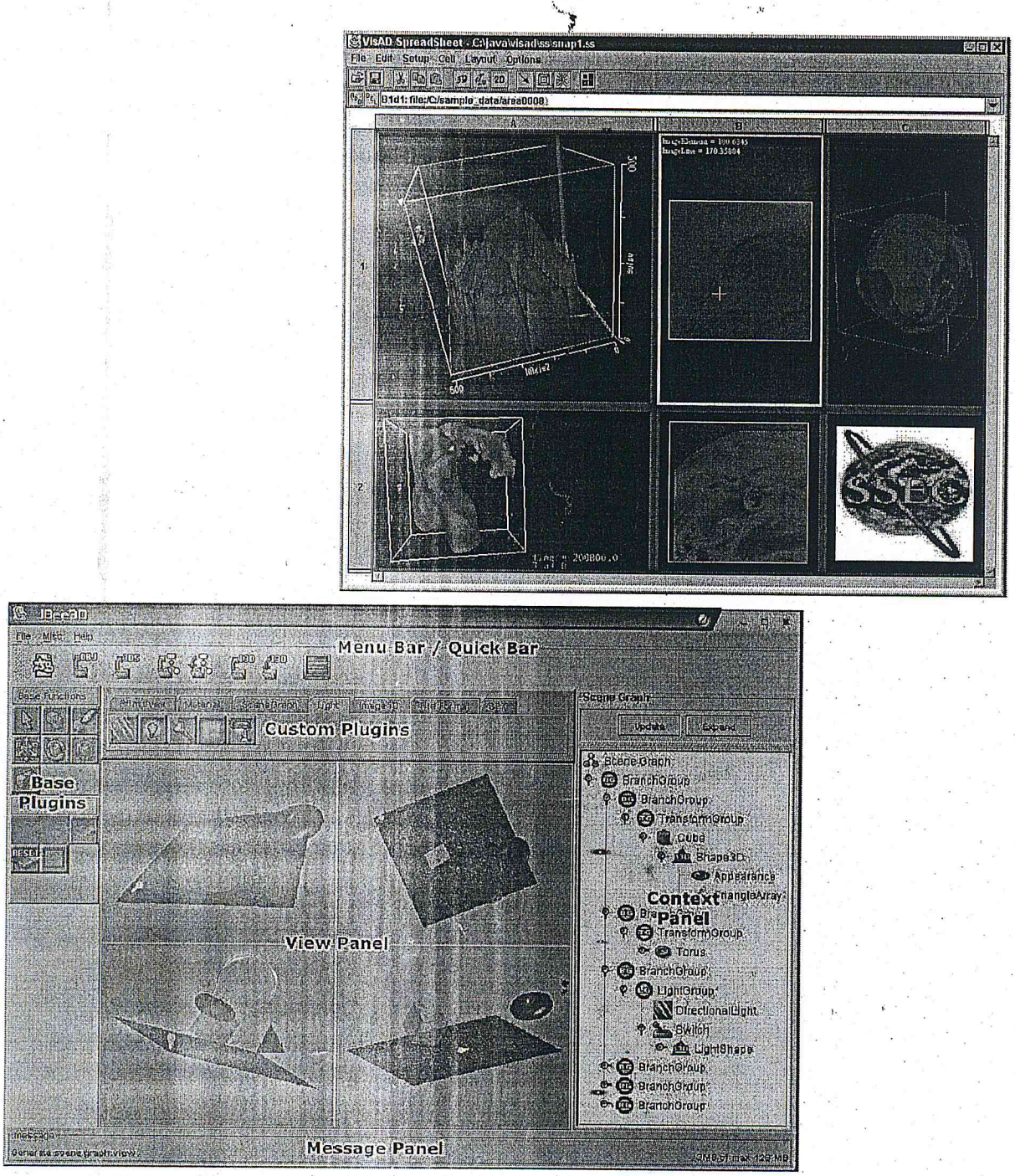


Figure 3.5 exemple d'application du moteur java3D

Java3D est beaucoup plus récent que ses concurrents DirectX et OpenGL, c'est surement une des causes qui font qu'il n'est pas beaucoup utilisé dans le développement d'applications 3D [Opdlm3d].

3.4 Un Bref comparatif des outils :

La première différence notable entre les différents outils concerne le niveau d'abstraction : Java3D se situe à un niveau supérieur à OpenGL et Direct3D, utilisant d'ailleurs ces deux API pour son implémentation. Java3D étant donc une surcouche aux API classiques, nous étudierons ces possibilités techniques séparément [Opdlm3d.]

3.4.1 OpenGL / DirectX :

Historiquement OpenGL et DirectX se sont développées en parallèle, les innovations des uns étant rapidement suivi d'une réponse du concurrent. Aujourd'hui, les différences de possibilités sont minimes entre les deux API; les dernières versions (DirectX 11.0b, et OpenGL 4.6) prennent en compte toutes les fonctionnalités classiques des moteurs 3D, sauf quelques détails dont en va pas en détaillé **[Opdlm3d]**.

3.4.2 Java3D :

La programmation avec Java3D est bien différente de celle avec OpenGL et DirectX. En effet, ces derniers proposent une implémentation de plus bas niveau qui permettent aux développeurs de gérer le moindre pixel de leurs moteurs alors que Java3D possède des fonctions de plus haut niveau, certes plus faciles d'utilisation mais aussi plus limitées. L'avantage de ces fonctions de haut niveau est que le développement d'applications est beaucoup plus facile. Par exemple il existe dans les package de Java3D des fonctions permettant de détecter si deux objets sont en collision, ce qui n'est pas le cas avec OpenGL ou DirectX ou l'on doit tout faire « à la main », ce qui dans ce cas est loin d'être évident. De plus, Java3D existe dans deux versions, une OpenGL et une DirectX, l'utilisateur peut donc choisir d'utiliser son programme avec l'un ou l'autre des moteurs de rendu.

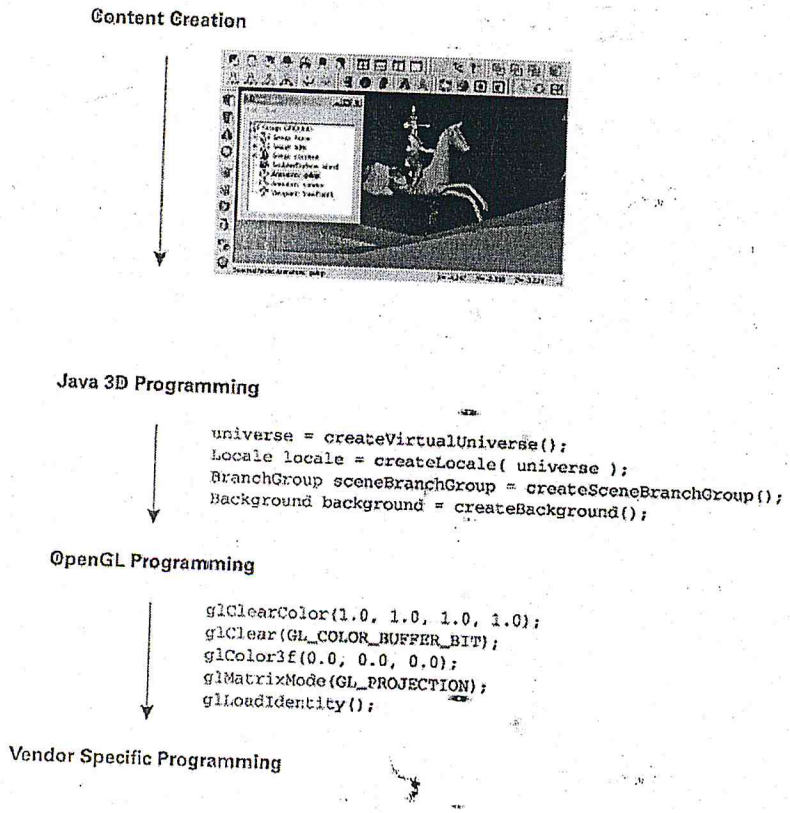


Figure 3.6 Java3D utilise OpenGL ou DirectX pour fonctionner

3.5 L'outil d'implémentation Java3D :

3.5.1 Qu'est ce que l'API Java 3D ?

L'API Java 3D est un classement de structures de données (classes) Java qui sert d'interface à la production d'images précises en trois dimensions, et de sons. Le programmeur travaille avec les

constructeurs lourds pour créer et manipuler des objets 3D géométriques. Ces géométries existent dans un univers virtuel, qui est ensuite rendu. L'API permet de créer des univers précis d'une large variété de dimensions, de l'astronomique au subatomique. **[J3dp]**

Malgré ces multiples fonctionnalités, l'API est néanmoins facile d'utilisation. Le processus de rendu est manipulé automatiquement. Tirant l'avantage de la faculté des threads Java, qui permettent à plusieurs processus de se dérouler en même temps, Java 3D est capable de rendre en parallèle. Il peut également optimiser le rendu de façon automatique afin d'en améliorer les performances. **[J3dp]**

Un programme Java3D crée des instances d'objets Java 3D et les place dans un graphe scénique (scene graph en anglais). Le graphe scénique est un agencement d'objets 3D, sous la forme d'une arborescence en arbre inversé, spécifiant le contenu de l'univers virtuel et comment il doit être rendu. **[J3dp]**

Les programmes en Java 3D peuvent être écrits pour être exécutés soit sous forme d'application, d'applet dans un navigateur configuré pour Java 3D ou bien sous les deux formes.

3.5.2 L'API Java 3D :

Chaque programme Java 3D est composé d'objets de la hiérarchie des classes de l'API Java3D. Cette collection d'objets décrit un *univers virtuel* (Virtual Universe), lequel est ensuite visualisé. L'API définit plus de 100 classes (voir annexe A) présentées dans le package javax.media.j3d. Ces classes sont généralement définies comme les *classes racines* (core class) de Java3D. **[Gswj3d]**

Il y a des centaines de champs et de méthodes dans les classes de l'API Java 3D. Pourtant, un Univers simple incluant de l'animation peut être construit avec un petit nombre de classes. Ce chapitre présente un ensemble minimal, il inclut le développement d'un programme simple, mais complet, nommé HelloJava3D qui affiche un cube en rotation. **[Gswj3d]**

En plus du package racine de Java 3D, d'autres packages sont utilisés dans l'écriture de programmes Java 3D. L'un d'entre eux, le package com.sun.j3d.utils qui est défini comme le package des *classes Utilitaires*. Le package des classes racines comprend seulement les classes de base nécessaires à la programmation Java3D. Les classes utilitaires sont des alliées commodes et puissantes en addition aux classes racines. **[Gswj3d]**

Objet visuel sera utilisé pour faire référence à un « objet déposé dans le graphe scénique » (par exemple un cube ou une sphère). Le terme d'*objet* étant lui utilisé seulement pour l'instance d'une classe. **[Gswj3d]**

3.5.2.1 Construire un graphe scénique :

Un univers virtuel Java 3D est crée depuis le *graphe scénique* travaillant avec des instances de classes Java 3D. Le graphe scénique rassemble des objets pour définir des géométries, des sons, des lumières. Mais aussi l'emplacement, le sens d'orientation, et l'apparence de ces objets visuels et sonores. **[Gswj3d]**

Le graphe scénique est un ensemble de données structurelles composées de nœuds et de liens. Un nœud est une donnée, un lien permet la relation entre les données. Les nœuds du graphe scénique sont des instances des classes Java 3D. Les liens représentent deux sortes de relations d'héritage entre les instances de classes Java 3D. **[Gswj3d]**

La relation principale est une *relation parent-enfant*. Un groupe de nœuds peut avoir n'importe quel nombre d'enfants mais un seul parent. Le dernier nœud d'une arborescence peut avoir un seul parent mais aucun enfant. L'autre rapport d'héritage est la *référence*. La référence relie un objet *NodeComponent* (composant de nœud) à un nœud du graphe scénique. Un objet NodeComponent détermine les attributs de géométrie et d'apparence utilisée pour produire les objets visuels. [Gswj3d]

Un graphe scénique Java 3D contient des objets nœuds relies par un héritage parent-enfant formant ainsi une hiérarchie en arbre inversée. Le point de départ de cette arborescence en arbre inversée est un nœud formant la racine. Le graphe scénique est crée par les objets formant les branches de l'arbre inversé prenant comme racine un objet nommé Locale. Les NodeComponent (composant de nœud) et les liens de références ne font pas partie de l'arborescence du graphe scénique. [Gswj3d]

Un seul lien relie entre eux chaque composant de l'arborescence depuis la racine. Donc, il n'y a qu'un seul chemin depuis la racine jusqu'au dernier nœud de l'arborescence. Le chemin depuis la racine du graphe scénique jusqu'à un nœud de terminaison précis est *le chemin de graphe scénique* de ce nœud de terminaison. Comme le chemin du graphe scénique ne mène qu'a une seule extrémité, il n'y a qu'un seul chemin de graphe scénique pour chaque extrémité du graphe scénique. [Gswj3d]

Chaque chemin du graphe scénique dans un graphe scénique Java 3D détermine entièrement le statut de l'extrémité lui appartenant. Ce statut contient l'emplacement, l'orientation, et la taille de l'objet visuel. Par conséquent, les attributs visuels de chaque objet visuel dépendent entièrement de leur chemin de graphe scénique. Le rendu Java 3D tire l'avantage de cette particularité et rend les objets présents dans l'ordre qu'il choisit comme le plus efficace. [Gswj3d]

La représentation sous forme de schéma d'un graphe scénique peut servir d'outil et/ou de documentation pour les programmes Java 3D. Le graphe scénique est représenté par des symboles, décrits dans la figure 3.7,

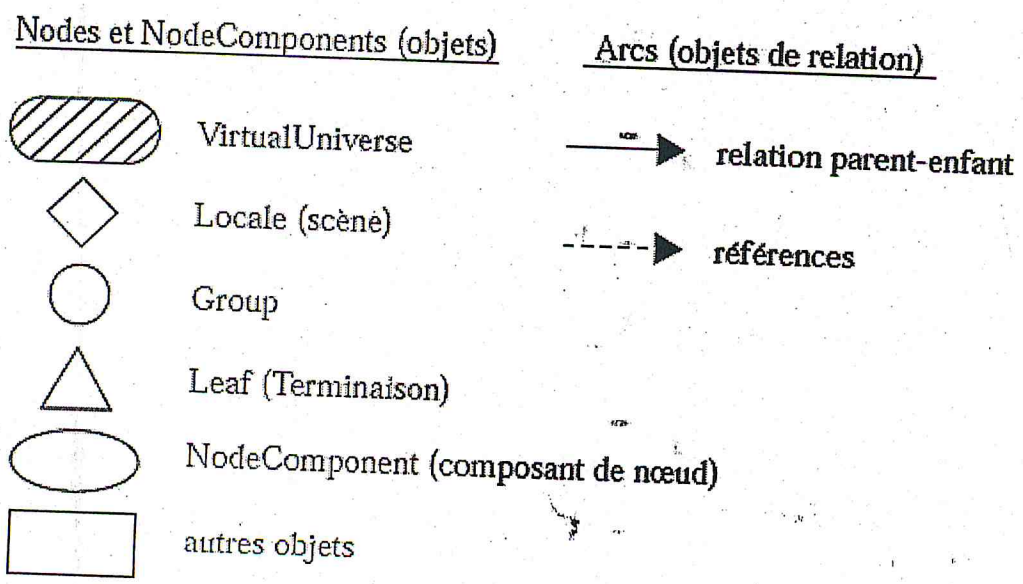


Figure 3.7 Représentation symbolique des objets d'un graphe scénique.

Chaque symbole du coté gauche représente un seul objet utilisé dans le graphe scénique.

Les deux premiers symboles représentent des objets de classes spécifiques : VirtualUniverse (univers virtuel) et Locale (scène). Les trois symboles suivants à gauche représentent les objets des classes Group (groupe), Leaf (terminaison), NodeComponent (composant du nœud). Le dernier symbole est utilisé pour représenter toutes les autres classes d'objet. La flèche pleine représente une relation d'héritage parent-enfant. La flèche pointillée est une référence à tout autre objet. Les objets référencés peuvent être distribués entre différentes branches du graphe scénique.

Un exemple d'un graphe scénique simple est décrit dans la figure 3.8

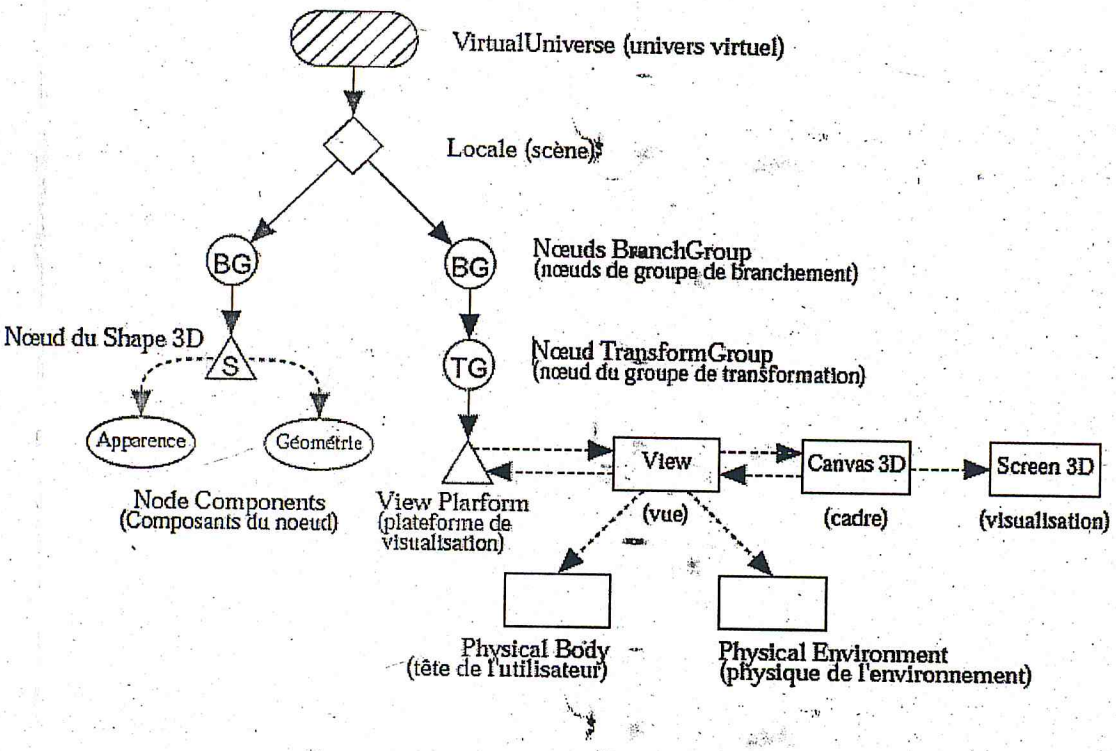


Figure 3.8 Premier exemple de graphe scénique.

Chaque graphe scénique possède un seul VirtualUniverse (univers virtuel). L'objet VirtualUniverse possède une certaine quantité d'objets Locale (scène). Un objet Locale est le point de départ de l'univers virtuel, c'est un point de repère pour déterminer la position des objets visuels dans l'univers virtuel [Gswj3d].

Dans un Local, des doubles ou les floats sont utilisés pour spécifier les positions des objets. Si un double est assez grand pour représenter exactement les positions de tous les objets dans une scène, alors un lieu simple devrait être suffisant. Si ce n'est pas le cas Un Local, emploie 256–bit pour spécifier la position de x, de y, et de z. Les 256–bit sont stockées dans un objet HiResCoord et permettent à un lieu d'être placé dans un l'espace virtuel, pouvant maintenir également une précision d'une longueur de Planck (plus petite que la taille d'un proton) et assez large qu'un univers. [J3dp]

2^n Mètres	Unités
87.29	Univers (20 milliard d'années de lumière
69.68	Galaxy (100,000 années de lumière)
53.07	années de lumière
43.43	Solar system diameter
23.60	Diamètre de la planète terre
10.65	Mille

9.97	Kilomètre
0.00	Mètre
−19.93	Micron
−33.22	Angstrom
−115.57	longueur d'un Planck

Figure 3.9 dimensions physiques exprimée par une puissance de 2

Alors que l'objet VirtualUniverse peut faire référence à plusieurs objets Locale, la plupart des programmes Java 3D possèdent un seul objet Locale. Chaque objet Locale peut servir de racine à de multiples sous arborescences du graphe. [J3dp]

L'objet BranchGroup est la racine d'un point d'articulation ou point de branchement. Il y a deux catégories différentes de point de branchement : le point de branchement de la *visualisation* et le point de branchement du *volume*. Le point de branchement de volume définit le *contenu* de l'univers virtuel - géométrie, apparence, comportement, emplacement, sons et lumières. Le point de branchement de la visualisation définit les paramètres visuels comme la position et la direction de la vue. Ces deux points de branchement forment la plus grande partie de ce que doit rendre le programme. [J3dp]

3.5.2.2 Hiérarchie des classes de haut niveau de l'API Java 3D :

Les classes VirtualUniverse, Locale, Group et Leaf (univers virtuel, scène, groupe, et terminaison) sont présentent dans cette partie de la hiérarchie. En plus des objets VirtualUniverse et Locale, le reste du graphe scénique est composé d'objets sceneGraphObject (objet du graphe scénique). SceneGraphObject est une superclasse de presque toutes les classes de javax.media.j3d (voir Annexe A) et com.sun.j3d.utils (classes racines et utilitaires).

Le sceneGraphObject possède deux sous-classes : Node (nœud) et NodeComponent (composant du nœud). La majeure partie des objets qui composent un graphe scénique sont les sous-classes Node (nœud). Un objet Node est soit un objet Group (groupe de nœuds) ou un objet Leaf (nœud de terminaison). Group et Leaf sont deux super classes d'un grand nombre de sous-classes. La classe Node et ses deux sous-classes sont détaillées ci-après. [Gswj3d]

La classe Node (nœud)
La classe nœud est une super-classe regroupant les classes Group et Leaf. La classe Node définit des méthodes importantes pour ses sous-classes. [Gswj3d]
La classe Group (groupe)
La classe Group est une super-classe utilisée pour déterminer la localisation et l'orientation des objets visuels dans l'univers virtuel. Les classes BranchGroup (groupe de branchement) et TransformGroup (groupe de transformation) sont deux sous-classes. Dans la représentation schématique du graphe scénique, les symboles de Group (cercles) contiennent souvent une annotation BG pour BranchGroup, TG pour TransformGroup, etc. [Gswj3d]
La classe Leaf (terminaison)
La classe Leaf est un super classe utilisé pour déterminer les formes, les sons et les comportements des objets visuels dans l'univers virtuel. Shape3D, Light, Behavior et Sound sont des sous-classes de la classe Leaf. Ces objets ne peuvent pas avoir d'enfant mais doivent se référer à un NodeComponent. [Gswj3d]
La classe NodeComponent (composant de noeud)
La classe NodeComponent est une super-classe utilisée pour déterminer la géométrie, l'apparence, la texture, et les attributs de matière d'un nœud Shape3D (terminaison). Les NodeComponent ne font

pas partie du graphe scénique, mais lui sont référencés. Un NodeComponent peut être référencé par plusieurs Shape3D. [Gswj3d]

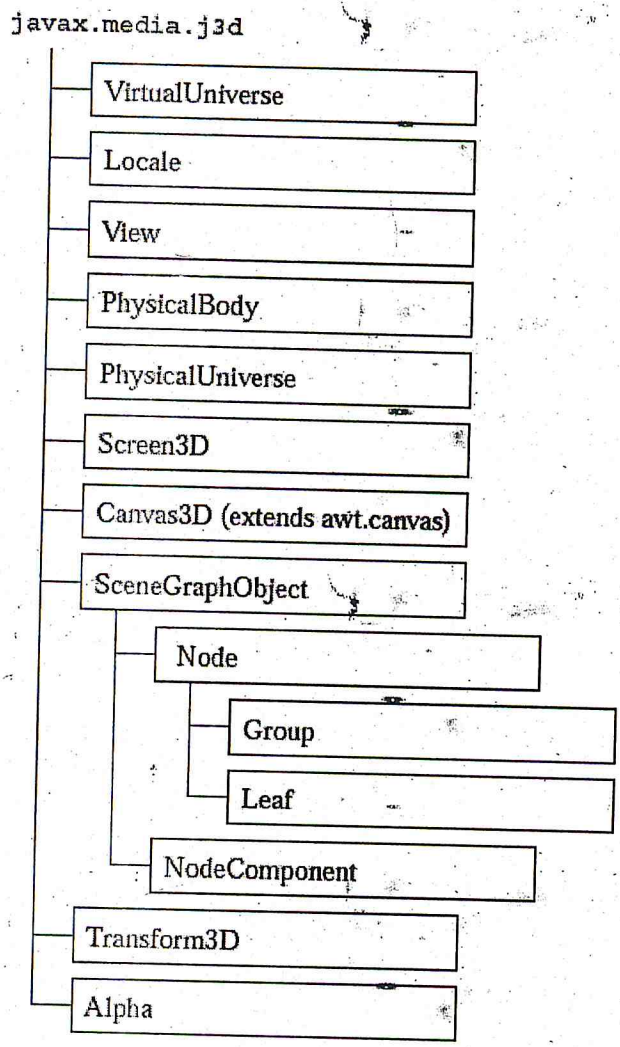


Figure 3.10 Un survol de la hiérarchie des classes de l'API Java 3D.

3.5.2.3 Une recette pour écrire des programmes Java 3D :

Les sous-classes de sceneGraphObject (objet du graphe scénique) sont les blocs de construction qui sont assemblés dans le graphe scénique. Le plan schématique fondamental dans le développement d'un programme Java3D se divise en sept étapes présentées dans le code 3.1. Cette recette peut être utilisée au montage de la plupart des programmes Java 3D [Gswj3d].

1. Création d'un objet Canvas3D.
2. Création d'un objet VirtualUniverse.
3. Création d'un objet Locale, et l'attacher à l'objet VirtualUniverse.
4. Construction d'un point de branchement de visualisation.
    a. Création d'un objet View.
    b. Création d'un objet ViewPlatform.
    c. Création d'un objet PhysicalBody.
    d. Création d'un objet PhysicalEnvironment.
    e. Attacher les objets ViewPlatform, PhysicalBody, PhysicalEnvironment, et Canvas3D à l'objet View.
5. Construction du (des) point de branchement de volume (formes, lumières et sons).

6. Compilation de l'arborescence de la branche(s) volume.
7. Insertion des points de branchement à l'objet Locale.

Code 3.1 Recette pour écrire des programmes Java 3D.

Cette recette ignore quelques détails mais illustre bien les concepts fondamentaux de la programmation Java 3D : la création de chaque point de branchement du graphe scénique représente la plus grande partie de la programmation. Plutôt que de développer cette recette, le prochain point décrit une méthode plus simple pour construire un graphe scénique très similaire avec moins de programmation. [Gswj3d]

3.5.2.4 Une recette simplifiée pour écrire des programmes Java 3D :

Les programmes Java 3D écrits avec la recette de base ont une construction identique de l'arborescence de visualisation. Cette construction identique de la branche visualisation se retrouve dans la classe utilitaire nommée SimpleUniverse. La classe SimpleUniverse accomplit les étapes 2, 3 et 4 de la recette précédente. [Gswj3d]

L'utilisation de la classe SimpleUniverse diminue le temps et l'effort nécessaire à la création de la branche de visualisation. Par conséquent, le programmeur peut consacrer plus de temps aux volumes. La création des volumes devenant ainsi l'essentiel de l'écriture de programmes Java 3D. Le SimpleUniverse est un bon point de départ à la programmation Java 3D, parce qu'il permet au programmeur d'ignorer toute la branche de visualisation. Toutefois, l'utilisation du SimpleUniverse ne permet pas pour autant d'avoir plusieurs points de visualisation de l'univers virtuel. [Gswj3d]

La classe SimpleUniverse

Le constructeur d'objet SimpleUniverse crée un graphe scénique incluant les objets VirtualUniverse et Locale (scène) ainsi qu'une arborescence visuelle complète. [Gswj3d]

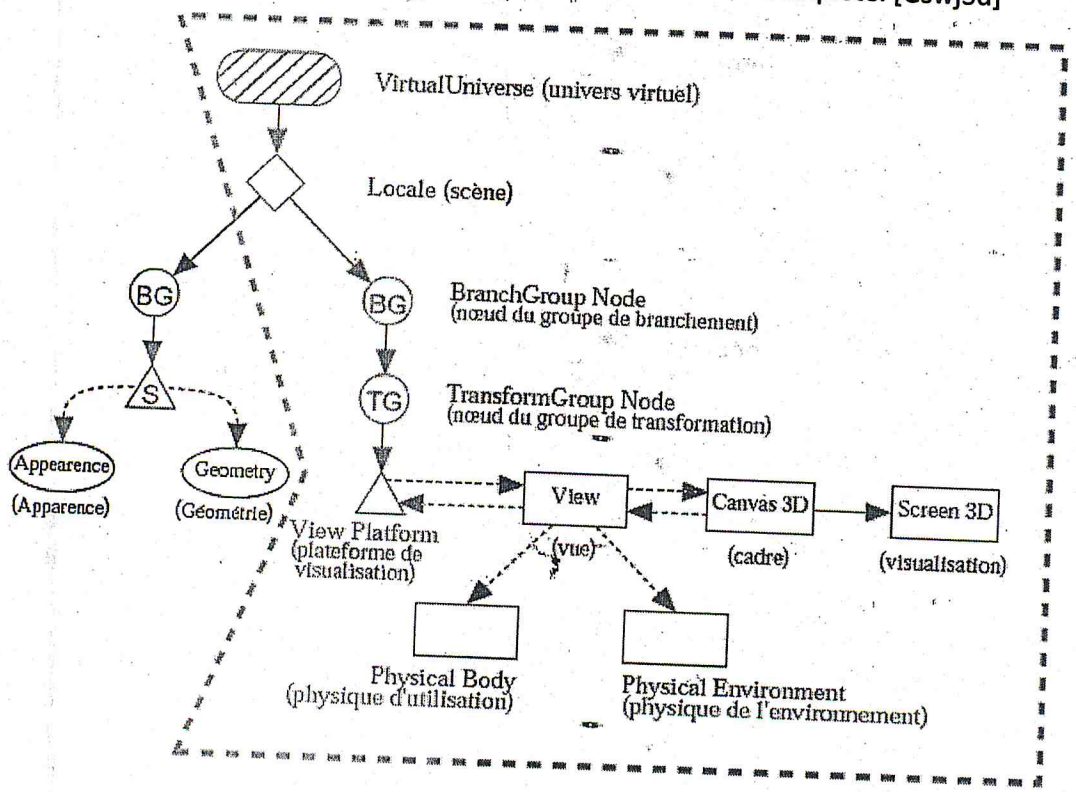


Figure 3.11 Un objet SimpleUniverse fournit un univers virtuel minimal, indiquée par la ligne pointillée.

L'utilisation de l'objet SimpleUniverse simplifie la première recette. Le code 3.2 montre la recette simplifiée, modifiant ainsi la recette de base pour l'utilisation de l'objet SimpleUniverse. Les étapes 2, 3 et 4 de la première recette sont remplacées par l'étape 2 dans la nouvelle recette. [Gswj3d]

1. Création d'un objet Canvas3D
2. Création de l'objet SimpleUniverse faisant référence à l'objet Canvas3D
a. Adaptation de l'objet SimpleUniverse
3. Construction du point de branchement de volume
4. Compilation de la branche volume
5. Insertion du point de branchement volume à la Locale (scène) du SimpleUniverse

Code 3.2 Recette simple pour écrire des programmes Java 3D en utilisant le SimpleUniverse.

Les Constructeurs du SimpleUniverse
SimpleUniverse()
Construit un univers virtuel simple.
SimpleUniverse(Canvas3D canvas3D)
Construit un univers simple avec une référence à l'objet nommé Canvas3D.

L'objet SimpleUniverse crée la branche de visualisation complète pour un univers virtuel. La branche de visualisation contient une image plate. L'image plate est une surface rectangulaire sur laquelle est projeté le volume pour créer l'image de rendu. L'objet Canvas3D, qui produit une image dans une fenêtre sur notre écran d'ordinateur, affiche l'image plate. [Gswj3d]
La figure 3.12, montre les relations d'héritage entre l'image plate, la position de l'œil, et l'univers virtuel.
L'œil est situé derrière l'image plate. Les objets visuels en face de l'image plate sont envoyés à l'image plate. Le rendu peut être vu comme une projection des objets visuels sur l'image plate. Ce concept est illustré par les quatre projecteurs sur le schéma (lignes pointillées)

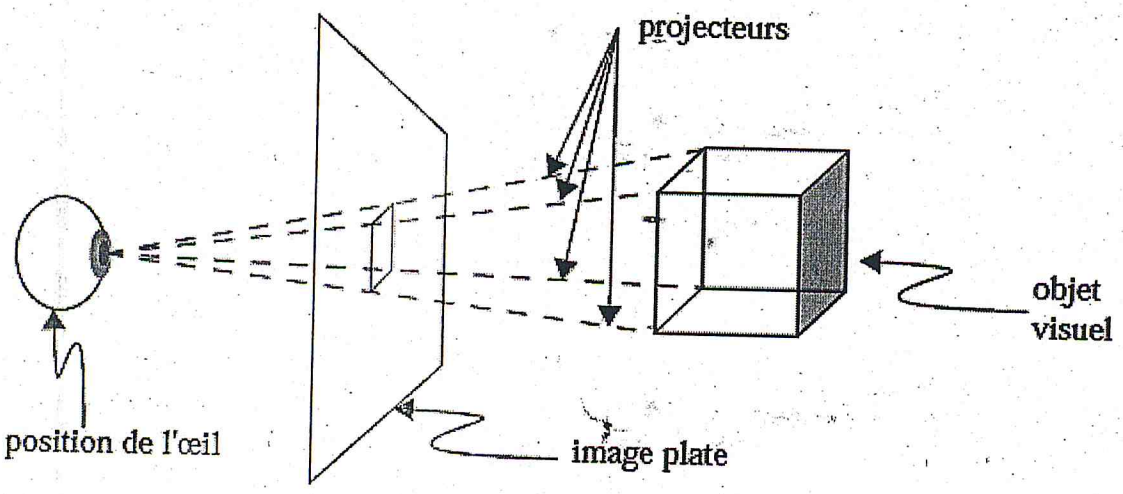


Figure 3.12 Schéma conceptuel d'une Image Plate et de la position de l'œil dans un univers virtuel.

Les Méthodes du SimpleUniverse (liste partielle)
Package: com.sun.j3d.utils.universe
void addBranchGraph(BranchGroup bg)

Utilisée pour ajouter des nœuds (Nodes) à l'objet Locale du graphe scénique crée par le SimpleUniverse.
Utilisé ici pour ajouter une branche de volume à l'univers virtuel.
ViewingPlatform getViewingPlatform()
Utilisé pour instancier l'objet ViewingPlatform au SimpleUniverse. Cette méthode est utilisée avec la méthode setNominalViewingTransform() de la ViewingPlatform pour régler la situation du point de vue.

3.5.3 HelloJava3D : Un exemple qui suit la recette simple :

La classe principale (classe main) d'un programme Java 3D défini une méthode pour construire le point de branchement de volume. Une telle méthode est définie dans l'exemple HelloJava3D et est nommée createSceneGraph().

Toutes les étapes de la recette simple sont appliquées dans le constructeur de la classe HelloJava3D.La première étape, créant un objet Canvas3D, est achevée à la ligne 4. L'étape 2, qui crée un objet SimpleUniverse, est terminée ligne 11. L'étape 2a, adaptant l'objet SimpleUniverse, est accomplie à la ligne 16. L'étape 3, construisant la branche de volume, est accomplie par l'appel de la méthode createSceneGraph(). L'étape 4, compilant la branche volume, est exécutée ligne 9. Pour finir, l'étape 5, insérant le point de branchement de volume à la Locale du SimpleUniverse, est achevée ligne 19.

```
public class HelloJava3D extends Applet {
public HelloJava3D() {
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

// Met en place une branche volume
BranchGroup scène = createSceneGraph();
scene.compile();

// SimpleUniverse est une Classe Utilitaire de complaisance
SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

// Ce qui suit recule un peu la ViewPlatform
//pour que les objets de la scène puissent être vus
simpleU.getViewingPlatform().setNominalViewingTransform();

// Connecte la branche volume au SimpleUniverse
simpleU.addBranchGraph(scene);
} // Fin d'HelloJava3D (constructeur)
```

Code 3.3 la Classe HelloJava3D

La troisième étape de la recette simple doit créer l'arborescence de la branche volume. Un point branchement de volume est créé dans Fragment de code 3.3. C'est certainement le point branchement de volume le plus sommaire possible. Le point branchement de volume crée dans Fragment de code 3.4 contient un objet graphique statique, un ColorCube. Ce ColorCube est placé à l'origine du système de cordonnées du *monde virtuel*. Avec l'emplacement et l'orientation définie par le sens de visualisation et du cube, le cube apparaît alors comme un rectangle au rendu. L'image de rendu sera dévoilée après la présentation de tout le code nécessaire au programme.

```
public BranchGroup createSceneGraph() {
 // Crée le point d'articulation de la branche volume
 BranchGroup objRoot = new BranchGroup();
```

```
// Crée une forme simple, terminaison d'un noeud,
// et l'ajoute au graphe scénique.
// ColorCube est une classe utilitaire de base
objRoot.addChild(new ColorCube(0.4));

return objRoot;
} // fin de la méthode createSceneGraph d'HelloJava3D
} // fin de la classe HelloJava3D
```

Code 3.4 Méthode createSceneGraph pour la classe HelloJava3D.

La classe HelloJava3D est une dérivé d'Applet mais le programme est exécutable comme un application par l'usage de la classe MainFrame. La classe Applet sert de classe de base pour rendre plus facile à écrire un programme Java 3D qui s'exécute dans une fenêtre. La MainFrame donne un cadre AWT (fenêtre) permettant à l'applet de s'exécuter comme une application. La taille de la fenêtre, résultat de l'application, est définie dans le constructeur de la classe MainFrame. Le Fragment de code 3.5 présente l'utilisation de la MainFrame dans HelloJava3D.java.

Le Constructeur MainFrame (liste partielle)
package: com.sun.j3d.utils.applet
La MainFrame produit une applet dans une application. Une classe dérivée d'applet doit avoir une méthode main() qui appelle le constructeur de la MainFrame. La MainFrame est une extension de java.awt.Frame.

MainFrame(java.applet.Applet applet, int width, int height)
Cree un objet MainFrame qui exécute une application comme une applet.
Paramètres:
applet - le constructeur d'une classe dérivée d'une applet. La MainFrame produit un cadre AWT pour cette applet.
width - la largeur du cadre de la fenêtre en pixels.
height- la hauteur du cadre de la fenêtre en pixels

```
// La suite permet d'exécuter cette application comme une applet.
public static void main(String[] args) {
Frame frame = new MainFrame(new HelloJava3D(), 256, 256);
} // fin de la méthode main (méthode principale d'HelloJava3D)
```

Code 3.5 Méthode Main() d'HelloJava3D qui fait appel à la MainFrame

Les trois Fragments précédents (code 3.3, 3.4, et 3 :5) forment un programme Java 3D complet si les déclarations d'importation adéquates sont utilisées. Les groupes de déclaration d'importation sont nécessaires à la compilation de la classe HelloJava3D. Les classes les plus communes se trouvent dans les packages javax.media.j3d, ou javax.vecmath. Dans cet exemple, seul la classe utilitaire ColorCube est situé dans le package com.sun.j3d.utils.geometry. Par conséquent, la plupart des programmes Java 3D utilisent la déclaration d'importation présentée dans le Fragment de code 3.6, à l'exception de l'import du ColorCube.

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.ColorCube;
```

```
import javax.media.j3d.*;
import javax.vecmath.*;
```

Code 3.6 Déclarations d'importation pour HelloJava3D.java

Dans le programme d'exemple HelloJava3D.java, un objet visuel unique est placé dans une seule Scène (Locale). Le graphe scénique qui en résulte est montrée dans la figure 3.13.

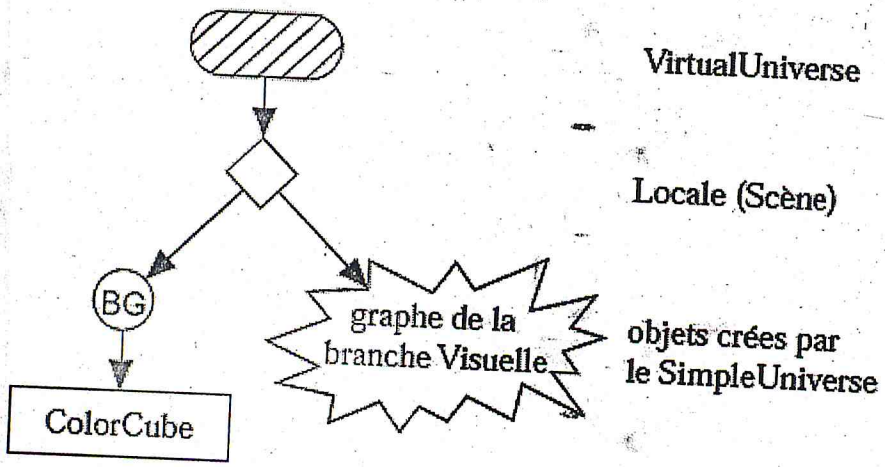


Figure 3.13 Graphe scénique de l'exemple HelloJava3D

On Compilant le code en lançant la commande : javac HelloJava3D.java. Et on exécutant le programme avec la commande : java HelloJava3D. L'image produite est montrée dans la figure 3.14.

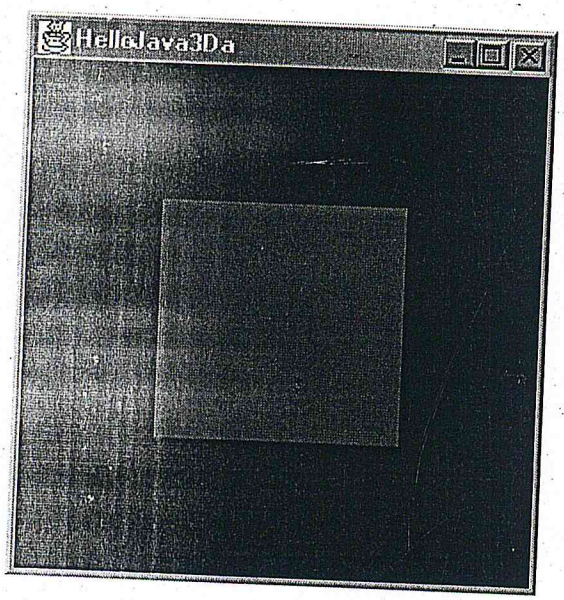


Figure 3.14 Image produite par HelloJava3D

Bien que chaque ligne de code de l'exemple HelloJava3D ne soit pas entièrement expliquée, les notions de base de l'assemblage d'un programme Java 3D doivent êtres claire à la lecture de l'exemple. L'Annexe B détaille les classes utilisées par le programme.

3.6 Conclusion :

Ce chapitre a décrit les premiers pas à faire pour l'utilisation de l'API Java 3D. Java3D se caractérise surtout par sa facilité d'utilisation, notamment grâce à ses primitives de haut niveau, à la représentation d'une scène sous la forme structuré d'un arbre, et également aux avantages du langage Java (orientation objet, API haut niveau très développée, grand nombre d'utilisateurs). Cette facilité d'utilisation, fait de Java3D l'outil idéal dans le développement de petites applications utilisant la 3D.

L'API Java 3D est encore beaucoup plus riche. Il permet de simuler un monde virtuel très riche incluant des objets ayant des formes complexe et plusieurs phénomènes du monde réel (c'est le but des concepteurs de Java 3D) pour des utilisations différentes.

Vu la richesse de Java 3d (il y a beaucoup de choses qu'on n'utilisera pas) et la contrainte de nombre de page sur ce mémoire, on a sélectionné ces point (traité plus haut). D'autre fonctionnalité seront découverte au cours du chapitre d'Implémentation.

Chapitre 4
Conception

4.1 Introduction :

Dans ce chapitre, nous allons aborder la conception de notre projet. On va s'intéresser aux besoins fonctionnels par des diagrammes de cas d'utilisation, et on termine avec le diagramme de classe qui montre un ensemble d'éléments de modélisations déclaratifs (statique).

4.2 Objectif et analyse des besoins :

L'objectif principal de notre projet est de développer une application qui aide les utilisateurs de génie civil à faire le calcul de structure avec une visualisation 3D et des vues en plan, et le pouvoir d'interagir avec les objets affichés en toute simplicité, puis générer une base de données, contenant toute les informations techniques d'un projet de Génie Civil.

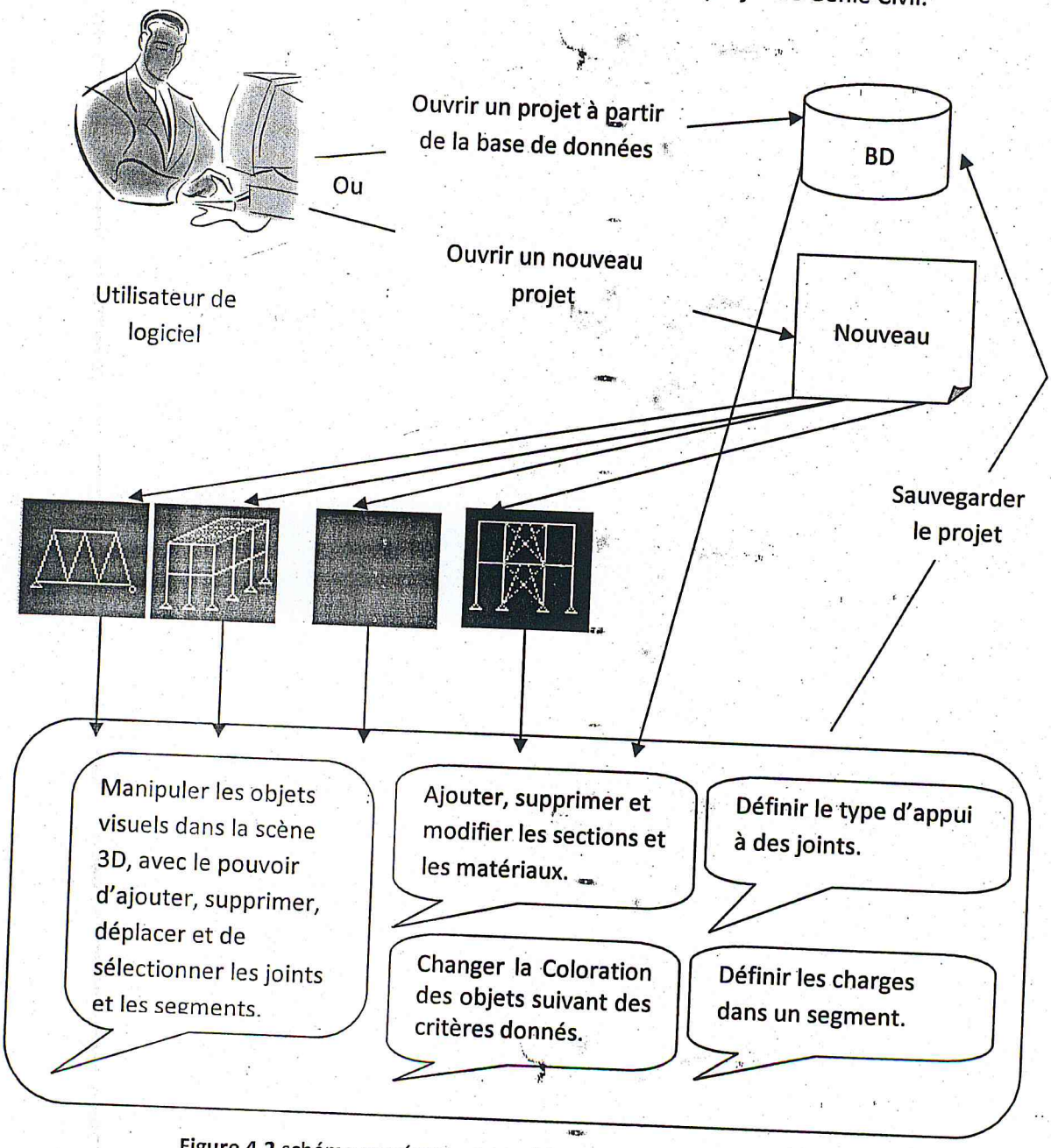


Figure 4.2 schéma représentant les objectifs principaux de notre projet

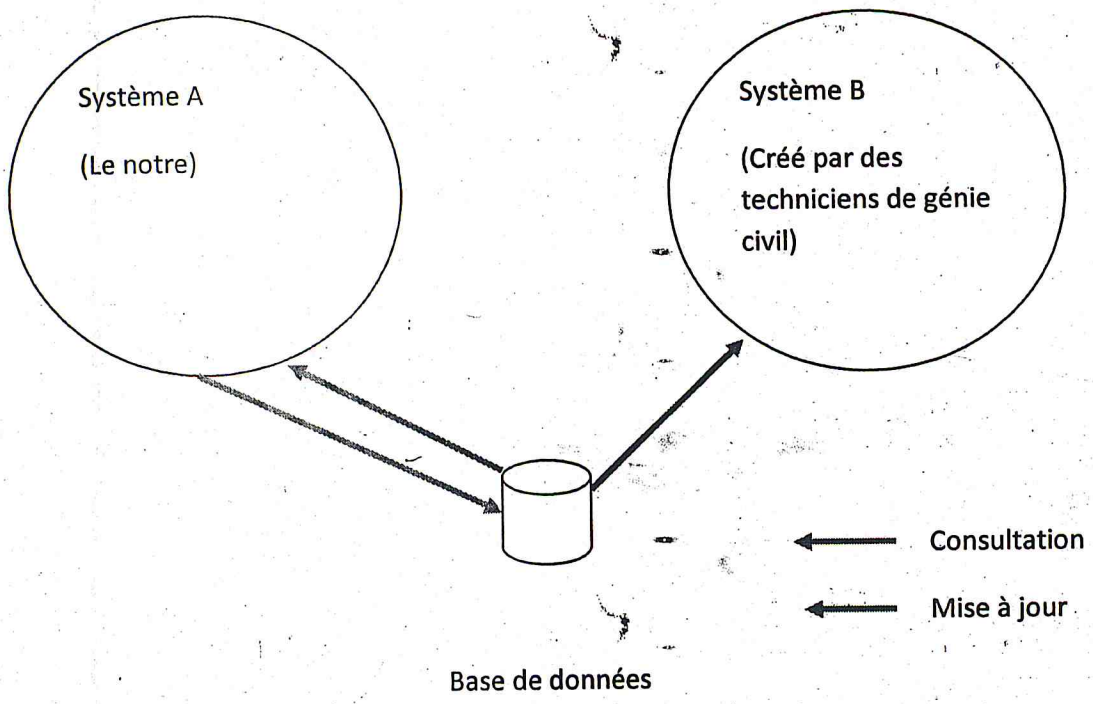


Figure 4.3 conception globale du projet.

Le système A (le notre) enregistre toutes les informations requises (position des poteaux et poutres, ferraillages) dans une base de données. Cette base de données sera par la suite consulter, par le système B, pour extraire des informations, qui seront par la suite des paramètres d'entrée pour le calcul de structure. Après ces calculs, le système B affichera les résultats compréhensibles aux techniciens de génie civil.

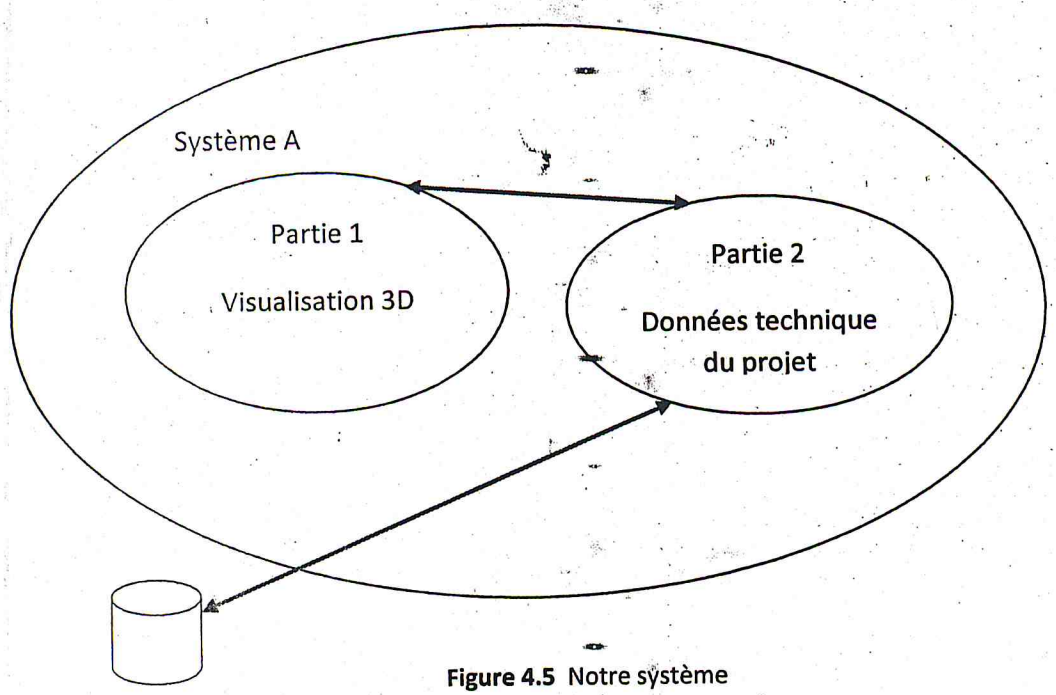


Figure 4.5 Notre système

La partie 1 s'occupe de tout ce qui est en rapport avec la 3D (la création de segment et de joints, la sélection d'objet visuel).

La partie 2 offre une interface permettant de paramétrer les objets visuels, comme le type de matériau utilisé dans un segment par exemple, et d'enregistrer le tout dans la base de données.

4.3 Modélisation de l'application :

Pour modélisation de notre application nous utiliserons un ensemble de diagrammes, un diagramme est une représentation graphique d'une collection d'éléments de modélisation, le plus souvent visualisée comme un graphe d'arcs (relations) et de sommets (autres éléments de modélisation). Le langage UML offre les diagrammes suivants: diagramme de classes, diagramme d'objet, diagramme de cas d'utilisation, diagramme de séquence, diagramme de collaboration, diagramme d'états transitoires, diagramme d'activités, diagramme de composants et diagramme de déploiement.

4.4 Modèle de cas d'utilisation :

Le cas d'utilisation décrit le système et les relations entre le système et l'environnement. Son intérêt:

- Permettre de délimiter les frontières du système
- Constituer un moyen d'exprimer les besoins d'un système
- Exploitation par les utilisateurs finaux pour exprimer leurs attentes et leurs besoins
- Permettre d'impliquer les utilisateurs dès les premiers stades du développement
- Constituer une base pour les tests fonctionnels

Les cas d'utilisation jouent un rôle fondamental dans le cycle de vie d'un projet de développement logiciel.

- En phase initiale, ils permettent d'identifier les utilisateurs et de comprendre leurs attentes.
- En phase d'élaboration, les développeurs s'appuient sur eux pour découvrir les objets « métiers » et constituer le modèle de domaine.

Les acteurs sont des entités externes qui interagissent avec le système, comme une personne humaine ou un robot. Une même personne (ou robot, ...) peut être plusieurs acteurs pour un système, c'est pourquoi les acteurs doivent surtout être décrits par leur rôle. Ce rôle décrit les besoins et les capacités de l'acteur.

Dans notre projet, il y a un seul acteur interagissant avec l'application, on le nomme utilisateur.

4.4.1 Diagramme de cas d'utilisation générale de notre application :

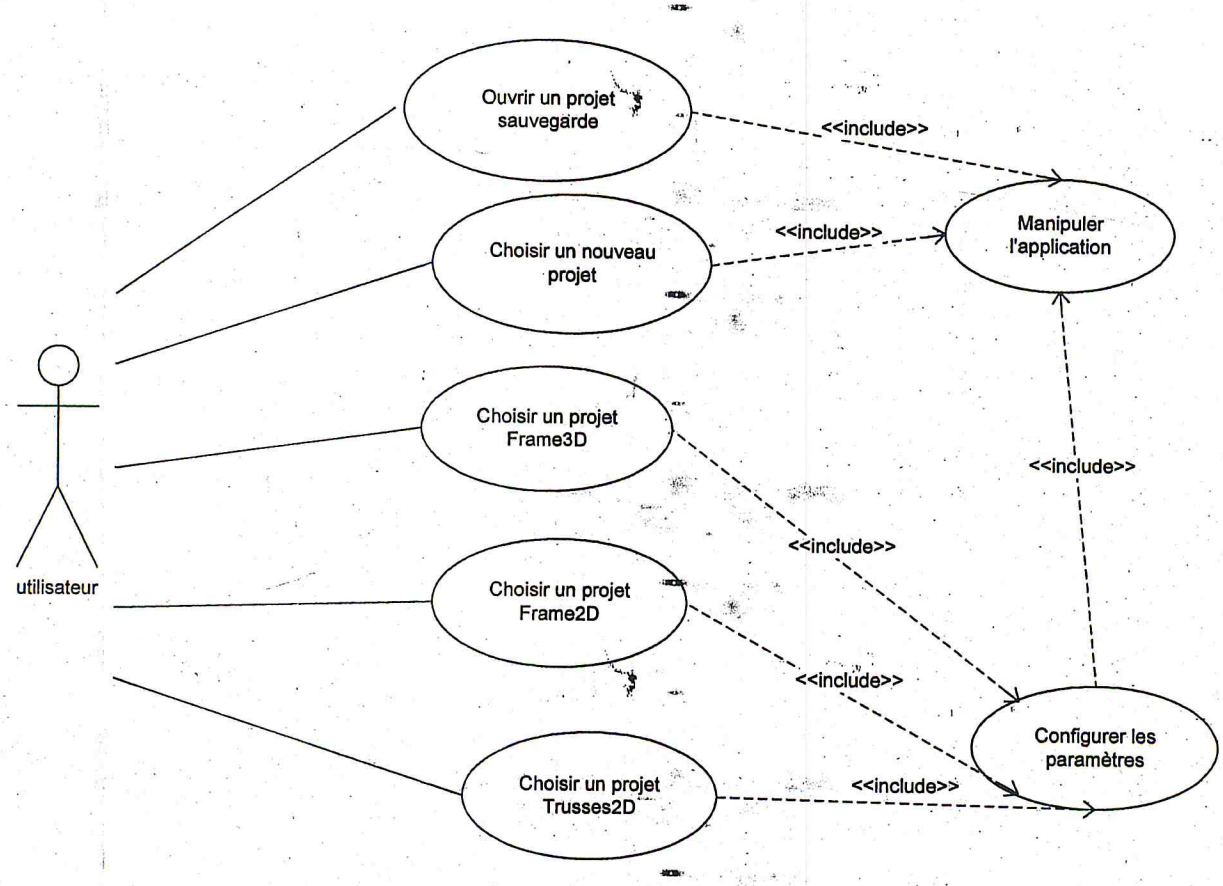


Figure 4.3 Diagramme de cas d'utilisation générale de notre application.

Cas d'utilisation	Description
Ouvrir un projet sauvegarde	C'est un projet qui existe déjà. Il est sauvegardé dans la base de données.
Choisir un nouveau projet	C'est un projet à blanc (tout faire à zéro).
Choisir un projet frame 3D	c'est un projet d'une structure qui ressemble à un bâtiment déjà prédéfini. On configure seulement les paramètres.
Choisir un projet Trusses 2D	c'est un projet d'une structure qui contient des étriers métalliques en 2D déjà prédéfini. On configure seulement les paramètres (nombre de divisions, sa largeur et sa hauteur).
Choisir un projet Frame2D	c'est un projet d'une structure qui contient des poutres et des poteaux dans un seul plan déjà prédéfini. On configure seulement les paramètres.

4.4.2 Diagramme de cas d'utilisation de configuration des paramètres pour le Frame 3D :

Dans le cas de configuration des paramètres pour le Frame 3D, l'utilisateur doit personnaliser la structure 3D qui sera affichée après.

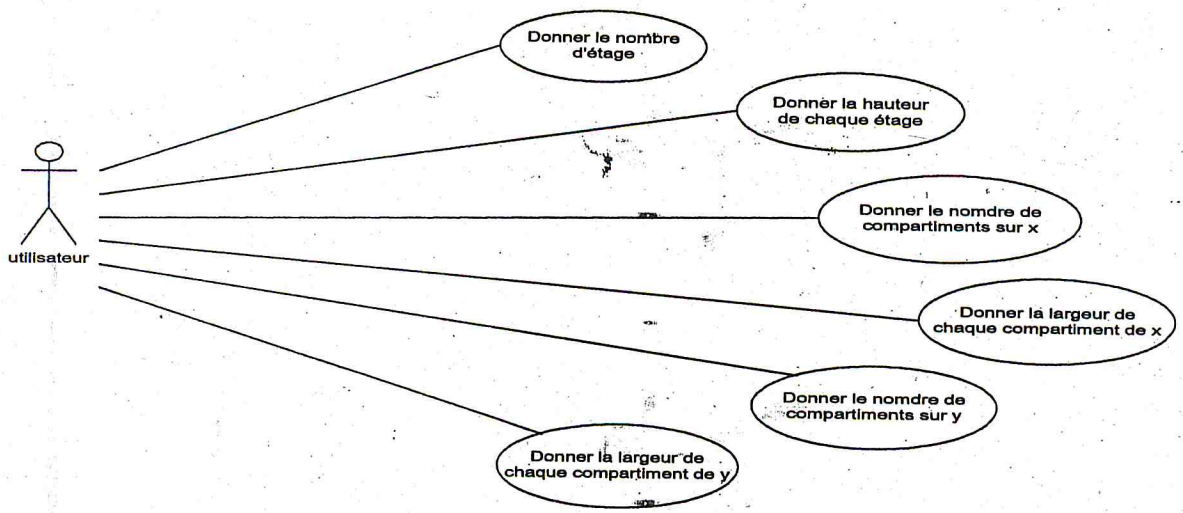


Figure 4.4 Diagramme de cas d'utilisation de configuration des paramètres pour le Frame 3D

4.4.3 Diagramme de cas d'utilisation de manipulation de l'application :

L'utilisateur doit avoir une liberté de manipulation dans l'application, grâce à une interface simple qui satisfait tous ses besoins. La manipulation pour le joint et le segment consiste à ajouter, supprimer et de les déplacer.

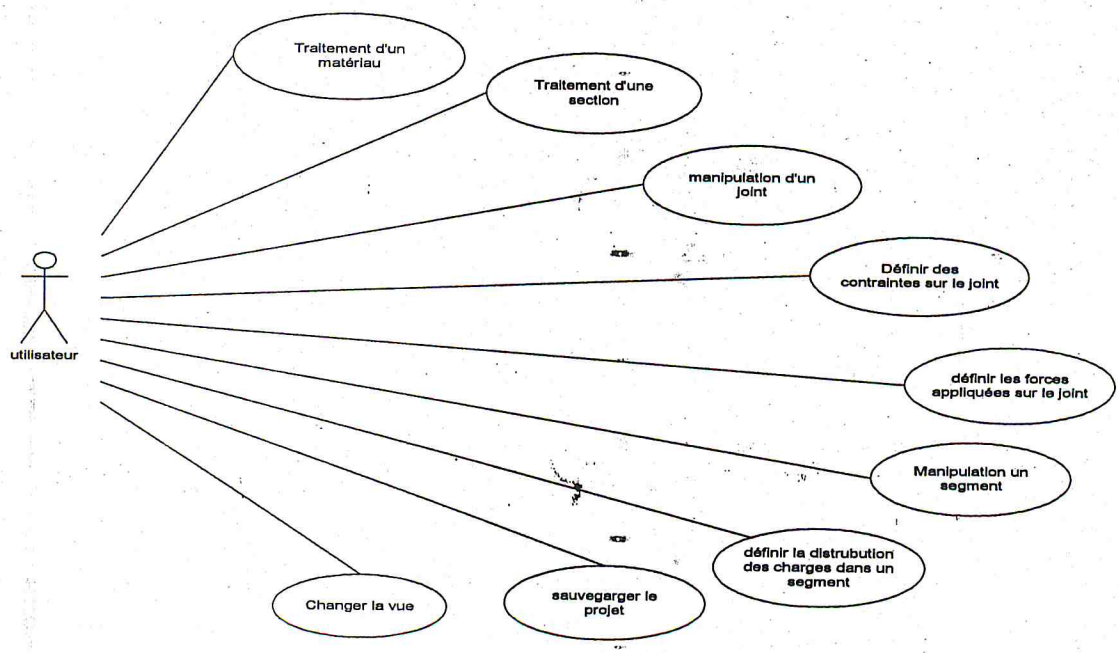


Figure 4.5 Diagramme de cas d'utilisation de manipulation de l'application

Le paragraphe suivant présente la partie données de notre application, nous le modéliserons en utilisant deux diagrammes de classe : un global, ainsi un autre pour le traitement 3D.

4.5. Modèle de classe :

Le diagramme de classes est considéré comme le plus important de la modélisation orientée « objet », il est le seul obligatoire lors d'une telle modélisation.

4.5.1 Diagramme de classe globale :

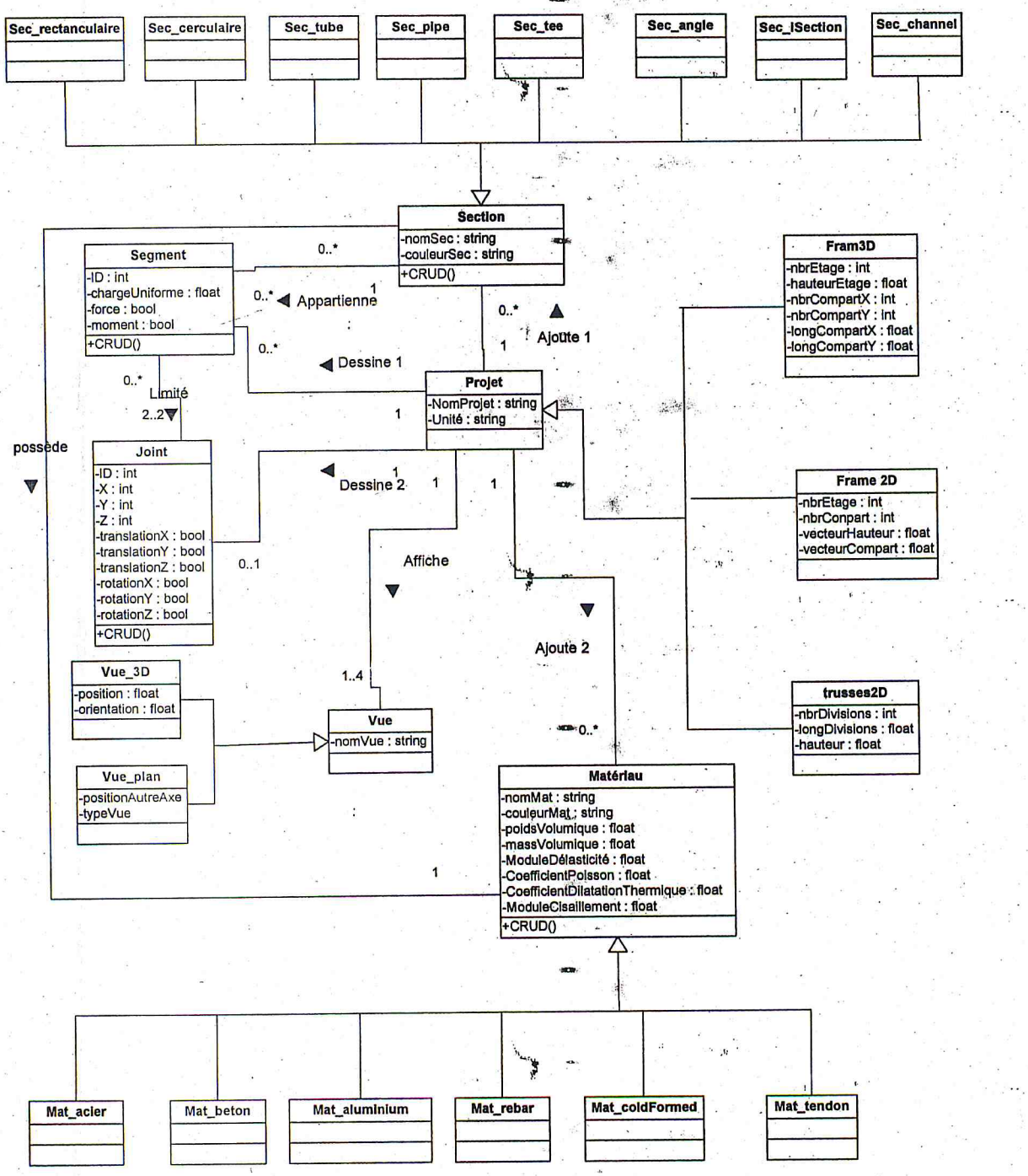


Figure 4.6 Diagramme de classe globale.

Classes	Description
Projet	C'est la classe principale, elle contient des informations sur le projet et son type (Frame3D, Frame2D, Trusses2D).
Matériau	Elle contient des informations sur les matériaux et les propriétés physiques qui dépendent du type de matériau (acier, béton, aluminium, tendon et rebar).
Section	Elle contient les informations sur la vue en coupe (la section), qui concerne les caractéristiques et les dimensions selon le type (rectangulaire, circulaire, tee, pipe, tube, Channel, angle, section I).
Joint	Elle contient les informations sur le joint qui est un point dans l'espace avec les coordonnées X, Y et Z. En plus, il définit le type d'appui, en bloquant les déplacements des joints dans l'espace (translations et rotations suivants les axes X, Y et Z).
Segment	Elle contient les informations sur le segment qui est limité par deux joints.
Vues	Elle contient les informations sur les quatre vues (3D, XY, YZ, XZ), avec le positionnement de caméra dans l'espace

4.5.2 Diagramme de classe pour le traitement 3D :

L'API java3d offre un ensemble de classes (annexe A) qui peuvent être utilisées ou redéfinies pour une utilisation particulière. Ces classes sont nombreuses, on a utilisé quelques unes comme : Behavior, pickMouseBehavior, KeynavigatorBehavior, Shape3d, LineArray, sphere .

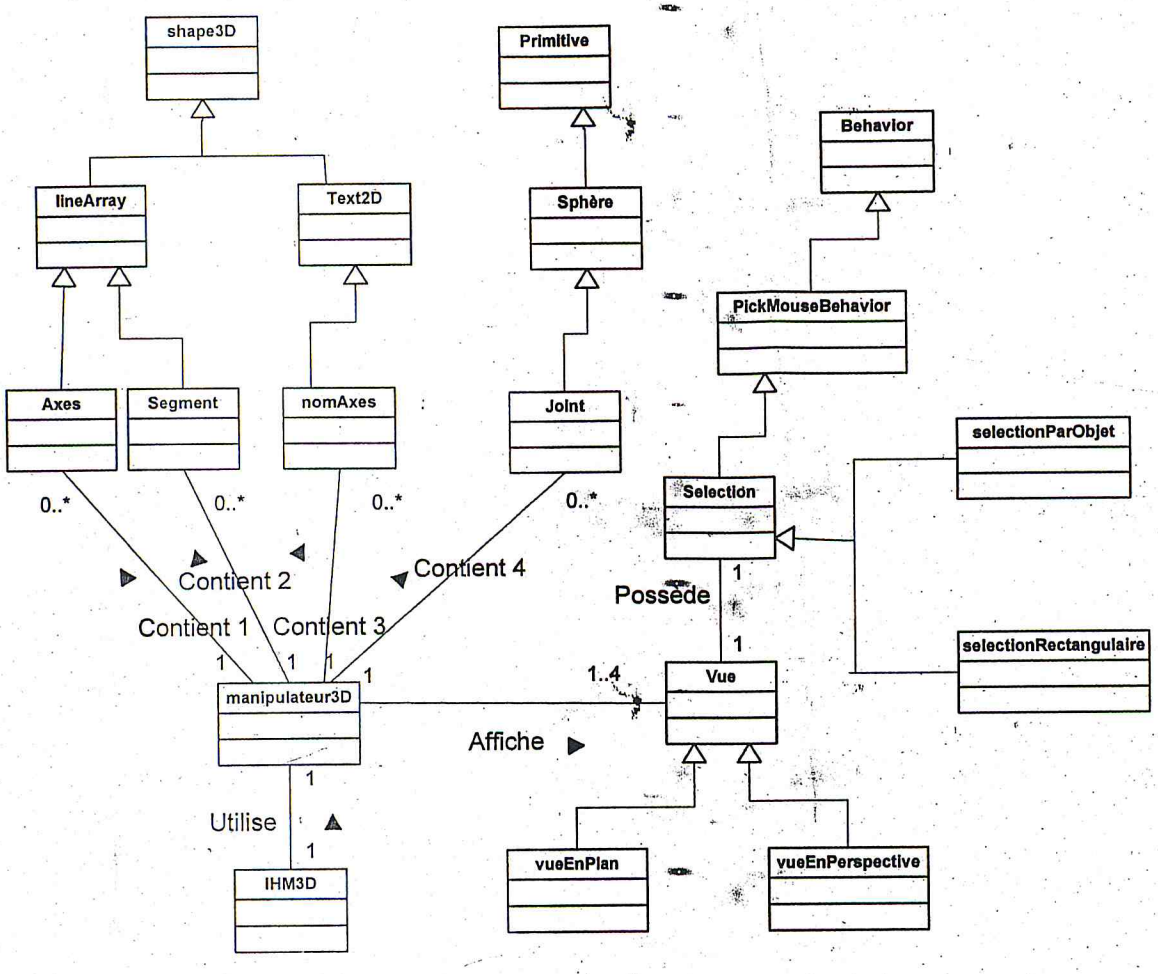


Figure 4.7 Diagramme de classe pour le traitement 3D

classe	Description
Manipulateur3D	Cette classe possède une structure de base pour la création d'un projet avec Java3D
IHM3D	Cette classe redéfinit plusieurs méthodes de manupulateur3D, et c'est la classe interface homme machine.
Behavior	Cette classe définit des objets auditeurs d'un déclenchement, qui peut être le passage d'un intervalle de temps, touches clavier pressées, click souris, collision entre objets visuels et plein d'autres méthodes. Elle est définie par L'API Java3d
PickMouseBehavior	Cette classe est définie par l'API java3D, elle met en place tous les outils entrant dans la sélection d'objets visuels.
Sélection	Cette classe applique les méthodes de sélection dans notre projet.

Shape3D	Cette classe est définie par l'API Java3D, elle définit des formes polygonales. Nous avons utilisé le Text2D et LineArray.
Axes	Cette classe dispose la géométrie modélisant les axes X, Y, Z.
nomAxes	Cette classe possède la géométrie modélisant les noms des trois axes.
Vue	Cette classe définit une vue. Elle se caractérise par la position de la vue et le nom de la vue (3D, X, Y, Z).
Primitive	Est une classe abstraite, elle est définie par l'API java3D, définissant quelques formes géométriques primitives comme sphère et cône.

4.6. Conclusion :

Dans ce chapitre, nous avons modélisé nos données et nos traitements en utilisant la notion UML. Grâce à notre conception, nous pouvons entamer la partie réalisation de notre projet avec l'implémentation.

Chapitre 5
Implémentation

5.1 Introduction :

Dans ce chapitre, nous allons présenter le processus de développement et les outils que nous avons utilisé dans notre travail ainsi que l'environnement de programmation (matériel & logiciel), ensuite nous exposons la communication qui existe entre l'application et la base de données à travers des interfaces graphiques IHM (Interface homme-machine) et un affichage 3D avec des objets interactifs.

Enfin, nous allons exposer quelques imprimés écrans de l'application avec quelques uneS des fonctionnalités qu'elle offre.

5.2 Présentation de l'environnement :

5.2.1 Environnement matériel :

Ordinateur PC :

- CORE 2 DUO 2.4 GHZ
- RAM 2 GO
- Disque dur 250 GO

5.2.2Environnement logiciel :

Dans le but de réaliser notre application, on a utilisé :

Eclipse : est un environnement de développement intégré libre extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

Java Development Kit (JDK) est l'environnement dans lequel le code Java est compilé pour être transformé en bytecode afin que la machine virtuelle Java (JVM) puisse l'interpréter.

Java 3D : est une interface de programmation (Application Programming Interface ou API) pour la plateforme Java visant la synthèse d'image 3D basée sur les graphes de scène. Cet API fait appel soit aux fonctions d'OpenGL ou Direct3D.

Microsoft Access ou MS Access (officiellement *Microsoft Office Access*) : est un logiciel édité par Microsoft. Il fait partie de la suite bureautique MS Office Pro. *MS Access* est composé de trois programmes : le moteur de base de données Microsoft Jet, un éditeur graphique, et le langage de programmation Visual Basic. Les trois ne sont pas séparables.

5.3 La base de données :

Comme on a vu dans la partie de l'environnement logiciel, on a utilisé le Microsoft Access pour la gestion de la base de données, qui contient toutes les informations techniques de notre projet de Génie Civil.

5.3.1 Description de la base de données :

Notre base de données est un stock d'informations décomposées et organisées dans des matrices appelées relations ou tables conformément au modèle de données relationnelles. Son contenu peut ainsi être synthétisé par des opérations d'algèbre relationnelle telles que l'intersection, la jointure et le produit cartésien.

On a le modèle relationnel des principales tables du projet suivant:

Projet (***nomProjet***, unité, typeProjet)

Matériau (***nomProjet, nomMat***, couleurMat,typeMat,PoidsV, MassV, E, G,V, A, fu, fy, fue, fye, fc, ftu, fty, fcy, fsu ,betonL, alumAliage)

Section (***nomProjet ,nomSec***, couleurSec, typeSec, nomMat,L, I, D,W, t2, t3, tw, tf, t2b, tfb, barMatLong, barMatconf, tailleBarConf, tailleBarLong, space, nbrBarConf, nbrBarLong)

Joint (***nomProjet ,idJoint***, X, Y, Z, siTx, siTy, siTz, siRx, siRy, siRz)

Segment(***nomProjet,idSegment*** ,idJointD,idJointF,nomSec,long,chargeUniforme,force,moment)

Vue (***nomProjet,typeVue***, positionCaméra)

- **La Table Projet :**

Attribue	Type	Description
nomProjet	Texte	C'est le nom du projet qui est son identificateur.
Unité	Texte	C'est l'unité de mesure utilisée dans le projet.
typeProjet	Texte	C'est le type de projet qui peut être utilisé au commencement (Frame3D, Frame2D, Trusses2D, nouveau projet).

- **Table Matériau :**

Attribue	Type	Description
nomProjet	Texte	Contient le nom du projet.
nomMat	Texte	Contient le nom de matériau.
couleurMat	Texte	Contient la couleur du matériau.
typeMat	Texte	Contient le type de matériau (béton, acier, aluminium, rebar, tendon)
poidsV	Réel	Cette valeur représente le poids volumique.
massV	Réel	Cette valeur représente la masse volumique.
E	Réel	Cette valeur représente le module d'élasticité
G	Réel	Cette valeur représente le module de cisaillement.
A	Réel	Cette valeur représente le coefficient de dilatation thermique.
V	Réel	Cette valeur représente le coefficient de poisson.
Fu	Réel	Cette valeur représente la contrainte de traction minimum.
Fy	Réel	Cette valeur représente l'effort de fléchissement minimum.
Fue	Réel	Cette valeur représente la contrainte de traction efficace.
Fye	Réel	Cette valeur représente l'effort de fléchissement efficace.
Ftu	Réel	Cette valeur représente la force finale de torsion.
Fty	Réel	Cette valeur représente la limite conventionnelle de l'élasticité de tension.
Fsu	Réel	Cette valeur représente la force finale de cisaillement.

Fc	Réel	Cette valeur représente la résistance à la pression du béton.
Fcy	Réel	Cette valeur représente la limite conventionnelle de l'élasticité compressive.
betonL	Booléen	Pour savoir si le matériau béton est léger ou non.
alumAliage	Texte	Pour désigner l'alliage de l'aluminium.

- **Table de Section :** (voir figure 1.5 à figure 1.15)

Attribue	Type	Description
nomProjet	Texte	Contient le nom du projet.
nomSec	Texte	Contient le nom de la section.
couleurSec	Texte	Contient la couleur de la section
typeSection	Texte	Contient le type de la section (rectangulaire, circulaire, tube, pipe, tee, angle, channel, section I)
nomMat	Réel	Cette valeur représente le nom de matériau attribué à cette section.
barMatLong	Texte	Contient le nom de matériau des barres longitudinales (rebar).
barMatConf	Texte	Contient le nom de matériau des barres de confinement.
tailleBarConf	Texte	C'est la taille des barres confinement de 1# jusqu'à 28d.
tailleBarLong	Texte	C'est la taille des barres longitudinales de 1# jusqu'à 28d
Space	Réel	Cette valeur représente l'espacement entre deux barre de confinement.
nbrBarLong	Entier	Cette valeur représente le nombre des barr longitudinales.
nbrBarConf	Entier	Cette valeur représente le nombre des barres confinement.

- **Table joint :**

Attribue	Type	Description
nomProjet	Texte	C'est le nom du projet qui est son identificateur.
idJoint	Entier	C'est l'identificateur d'un joint dans un projet.
X	Entier	Coordonnée par rapport à l'axe des X.
Y	Entier	Coordonnée par rapport à l'axe des Y.
Z	Entier	Coordonnée par rapport à l'axe des Z.
siTx	Booléen	Pour savoir si le déplacement de translation par rapport à l'axe des X est bloqué ou non.
siTy	Booléen	Pour savoir si le déplacement de translation par rapport à l'axe des Y est bloqué ou non.
siTz	Booléen	Pour savoir si le déplacement de translation par rapport à l'axe des Z est bloqué ou non.
siRx	Booléen	Pour savoir si le déplacement de rotation par rapport à l'axe des X est bloqué ou non.
siRy	Booléen	Pour savoir si le déplacement de rotation par rapport à l'axe des Y est bloqué ou non.
siRz	Booléen	Pour savoir si le déplacement de rotation par rapport à l'axe des Z est bloqué ou non.

- **Table Segment :**

Attribue	Type	Description
nomProjet	Texte	C'est le nom du projet qui est son identificateur.

idSegment	Entier	C'est l'identificateur d'un segment dans un projet.
idJointD	Entier	Représente l'identificateur d'un joint de début dans ce segment.
idJointF	Entier	Représente l'identificateur d'un joint de fin dans ce segment.
nomSec	Entier	Contient le nom de la section qui est attribuée à ce segment.
Long	Réel	Cette valeur représente la longueur de ce segment.

5.3.2 Communication entre l'application et la base de données :

Dans le code source de notre application, on a créé un package DAO qui représente la couche d'accès aux données qui sont persistantes au sein d'un SGBD. Le package DAO ne s'occupe que de la communication entre la base de données et l'application.

Dans chaque classe du package DAO, il faut importer la bibliothèque SQL de Java.

L'établissement de connexion se fait grâce à la classe « connexion.java » qui gère les exceptions en cas de problème de connexion avec 'try' et 'catch' .

```
package dao;
import java.sql.Connection;
import java.sql.DriverManager;

public class connexion {
      public static Connection connect(){
            Connection connectionMySql =null;
            try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            connectionMySql =DriverManager.getConnection("jdbc:odbc:PFE");

            }
            catch(Exception e){
                  System.err.println(e.getMessage());
            }
            return connectionMySql;
      }

}
```

Code 5.1 code de classe connexion.java.

Chaque classe DAO gère une table dans la base de données et implémente une interface IDAO. Cette interface possède un ensemble de méthodes abstraites, représentant l'acronyme

informatique anglais CRUD qui désigne les quatre opérations de base pour la persistance des données : Create (ajouter), Read ou Retrieve(AvoirListe), Update(modifier) et Delete ou Destroy(supprimer)

```
package dao;
import java.util.ArrayList;
                    import java.sql.SQLException;
import entité.materiel;

public interface IdaoMat{
      public void ajouter(materiel a) throws SQLException;
      public void supprimer(materiel a) throws SQLException;
      public void modifier(materiel a) throws SQLException;
      public ArrayList<materiel> getList() throws SQLException;
}
```

Code 5.2 code de l'interface IdaoMat.

Chaque classe DAO redéfinie les signatures de son interface avec une connexion à la base et une requête SQL. Exemple pour l'ajout d'un matériau, on a le code suivant :

```
public static void ajouter(mat_alum a) throws SQLException {

            Statement stmt = connexion.connect().createStatement();
            String req = "insert into materiel(nom_mat,color_mat,p_v,m_v,E,V,A,G";
            req +=      ",fcy,fty,ftu,fsu,alliage__alum,type_alum,unité,type_mat)
";
            req +=
"values('"+a.getNom_mat()+"','"+a.getColor_mat()+"',"+a.getP_v()+","+a.getS_v()+",
"+a.getE()+","+a.getV()+","+a.getA()+","+a.getG()+",";
            req +=
""+a.getFcy()+","+a.getFty()+","+a.getFtu()+","+a.getFsu()+",'"+a.getAlliage_alum(
)+"','"+a.getType_alum()+"','"+a.getUnité()+"','"+a.getType()+"')";
            stmt.executeUpdate(req);
      }
```

Code 5.3 code de la fonction chargé de l'ajout d'un matériau.

Pour la suppression, on a :

```
public static void supprimer(MaterielC a,String n) throws SQLException {


            Statement stmt = connexion.connect().createStatement();
            String req = "delete from materiel where nom_mat = '"+a.getNom_mat()+"'
AND nom_projet='"+n+"'";
            stmt.executeUpdate(req);
      }
```

Code 5.4 code de la fonction chargé de supprimer un matériau.

5.4 IHM (Interface homme-machine) :

Dans le projet, on a utilisé la bibliothèque graphique SWING du langage JAVA. Swing offre la possibilité de créer des interfaces graphiques identiques quel que soit le système d'exploitation sous-jacent, au prix de performances moindres qu'en utilisant Abstract Window Toolkit (AWT). Il utilise le principe Modèle-Vue-Contrôleur (MVC, les composants Swing jouent en fait le rôle du contrôleur au sens du MVC) et dispose de plusieurs choix d'apparence (de vue) pour chacun des composants standards.

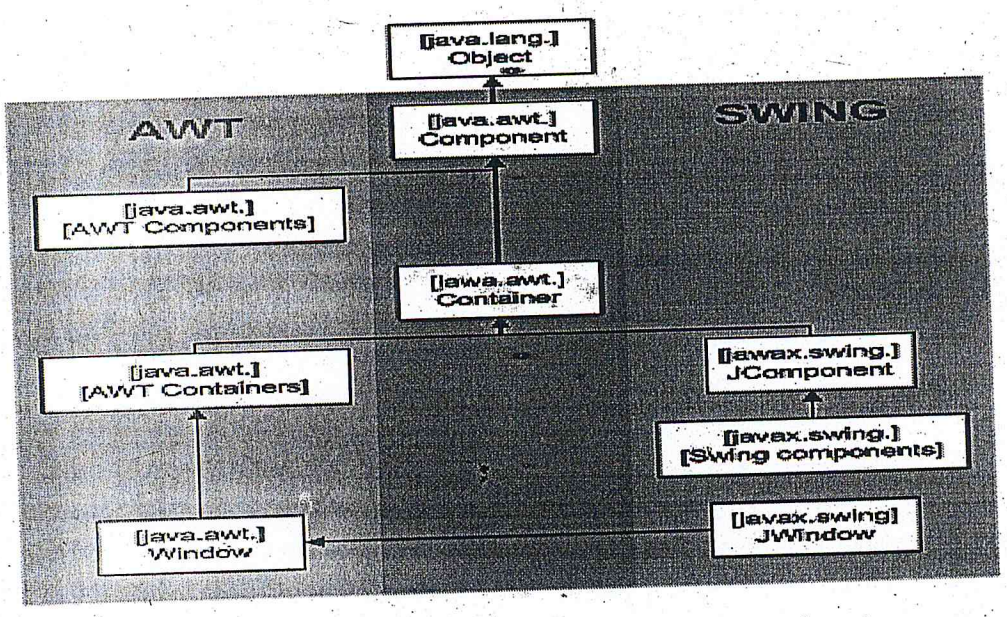


Figure 5.1 Schéma des pseudo-classes AWT et Swing hiérarchisés en Java.

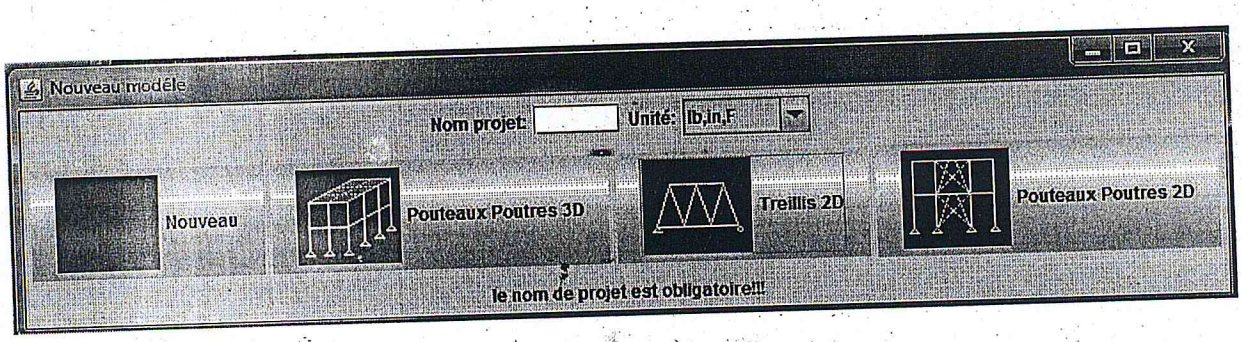


Figure 5.2 La fenêtre pour le choix du projet.

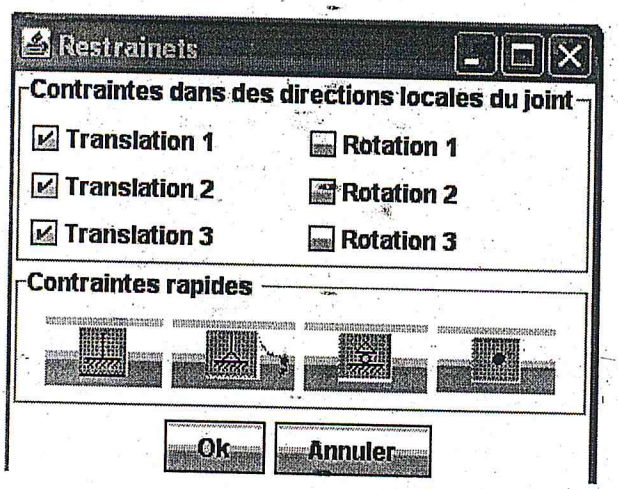


Figure 5.3 Fenêtre pour la gestion des appuis.

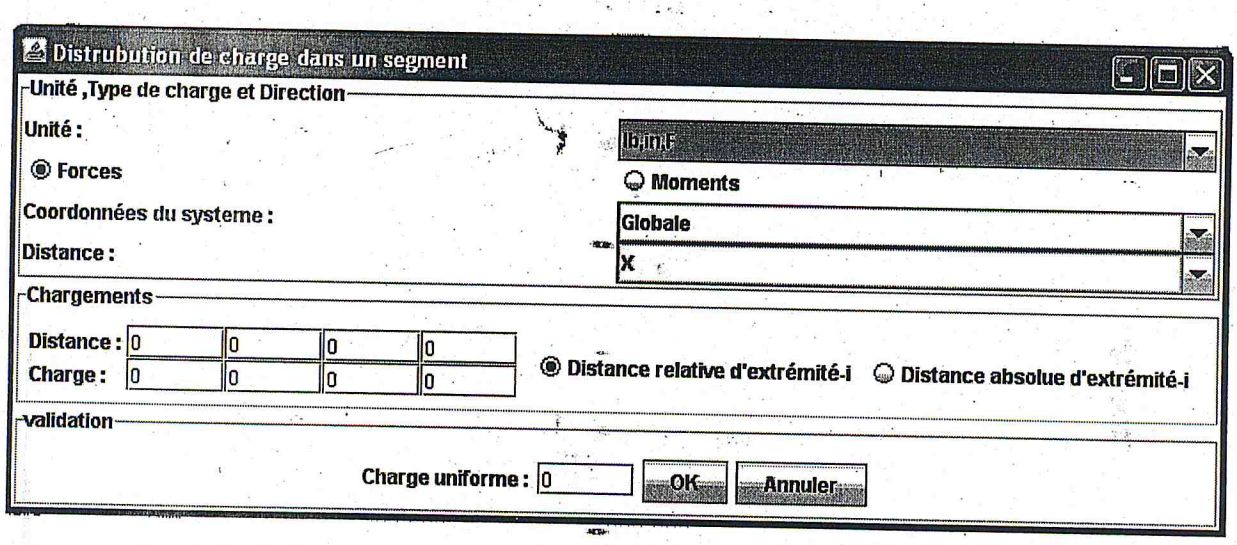


Figure 5.4 Fenêtre de distribution des charges dans un segment

5.4.1 Gestion des matériaux :

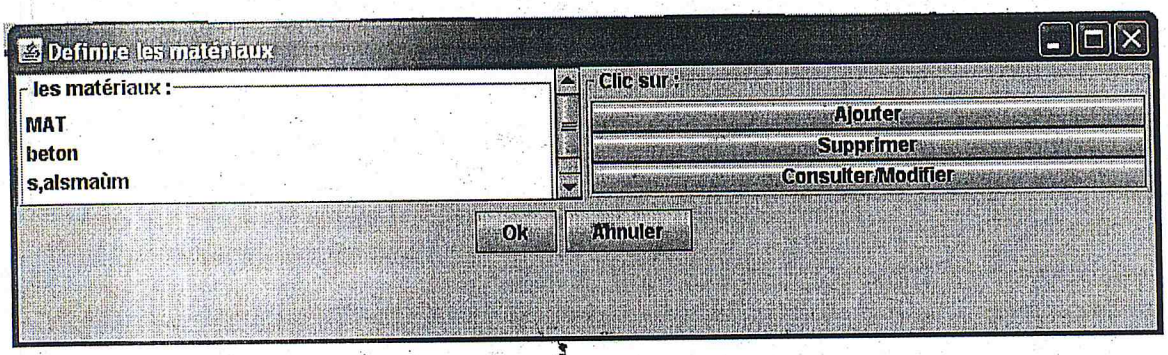


Figure 5.5 La fenêtre qui permet à l'utilisateur d'ajouter, de supprimer, de modifier ou de consulter un matériau.

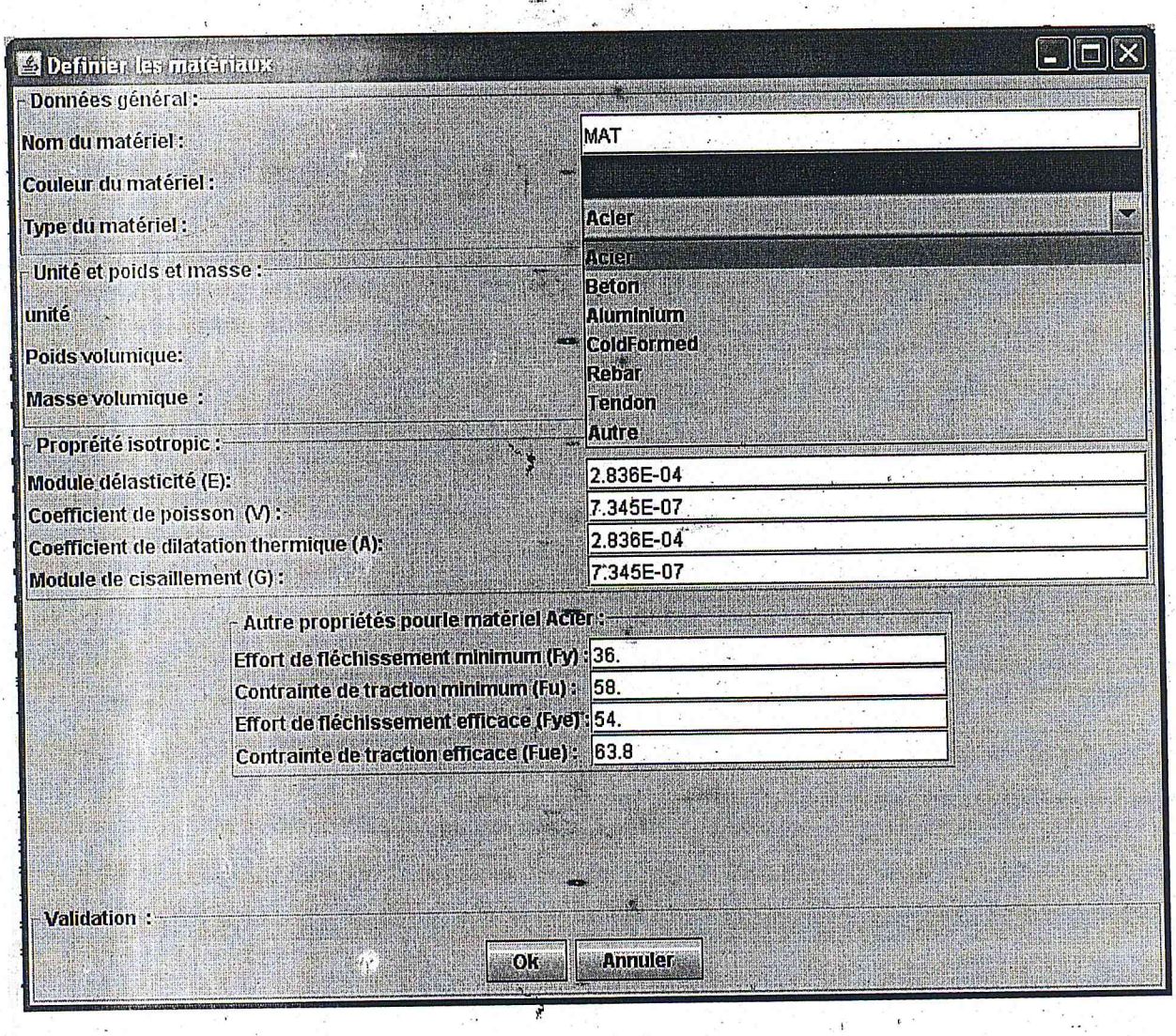


Figure 5.6 La fenêtre qui permet de définir les propriétés d'un matériau.

5.4.2 Gestion des sections:

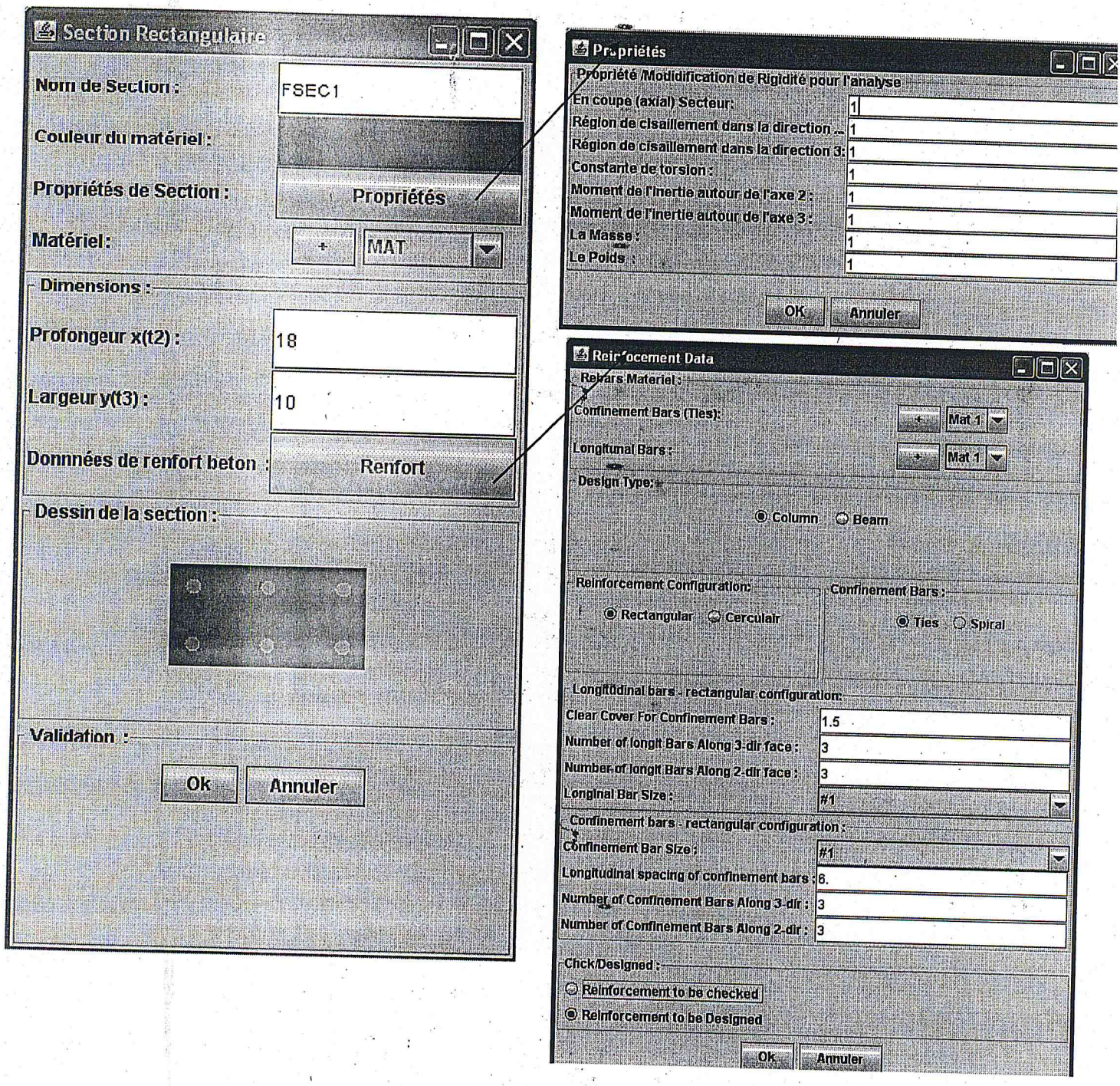


Figure 5.7 Les fenêtres pour la gestion d'une section rectangle de béton armé.

5.6 Implémentation de la partie traitant la 3D :

5.6.1 Création de géométries :

Le Chapitre Java 3D explorait les concepts de base de la construction d'un univers virtuel Java 3D. Il y a trois manières différentes de créer un nouveau volume géométrique. La première utilise les classes utilitaires Box, Cone, Cylinder, et Sphere. Dans la seconde c'est le programmeur qui détermine les cordonnées des sommets pour les points, segments de ligne et/ou surfaces polygonales.

5.6.1.1 Les bases de la définition d'un objet visuel :

Une instance de Shape3D définit un objet visuel :

Un nœud Shape3D du graphe scénique définit un objet visuel. Shape3D est une des sous-classes de la classe Leaf ; L'objet Shape3D ne contient pas d'informations sur la forme et la couleur de l'objet visuel. Ces informations sont stockées dans les objets NodeComponent se référant à cet objet Shape3D. Un objet Shape3D peut se référer à un composant de nœud Geometry et un composant de nœud Appearance.

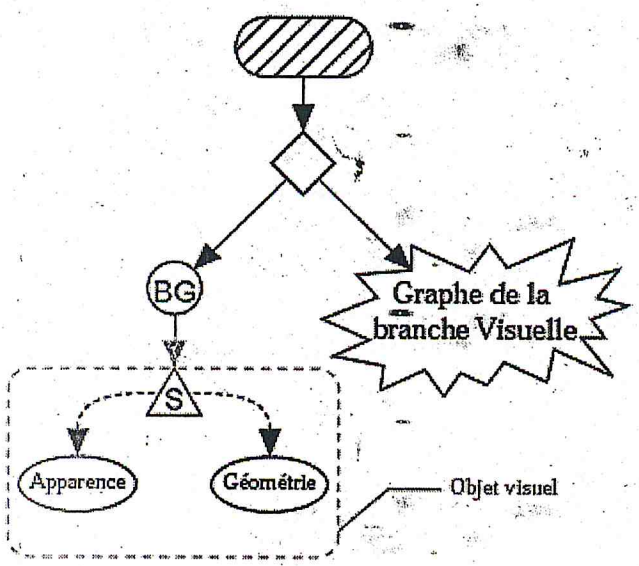


Figure 5.8 Un objet Shape3D définit un objet visuel dans le graphe scénique.

Constructeurs du Shape3D :

Shape3D()

Construit et initialise un objet Shape3D sans composants de noeud de géométrie et d'apparence.

Shape3D(Geometry geometry)

Construit et initialise un objet Shape3D avec le composant de noeud de géométrie spécifié et aucun composant d'apparence.

Shape3D(Geometry geometry, Appearance appearance)

Construit et initialise un objet Shape3D avec le composant de noeud de géométrie spécifié et des composants d'apparence.

5.6.2 Node Components (composants de nœud) :

Les objets NodeComponent contiennent la spécification exacte des attributs d'un objet visuel. Chacune des nombreuses sous-classes de NodeComponent définissent certains attributs visuels. La figure 5.9 montre une partie de la hiérarchie de l'API Java 3D contenant la classe NodeComponent et ces descendants. La Partie 2.5 présente le composant de noeud Geometry. La Partie 2.6 présente le composant de nœud Appearance.

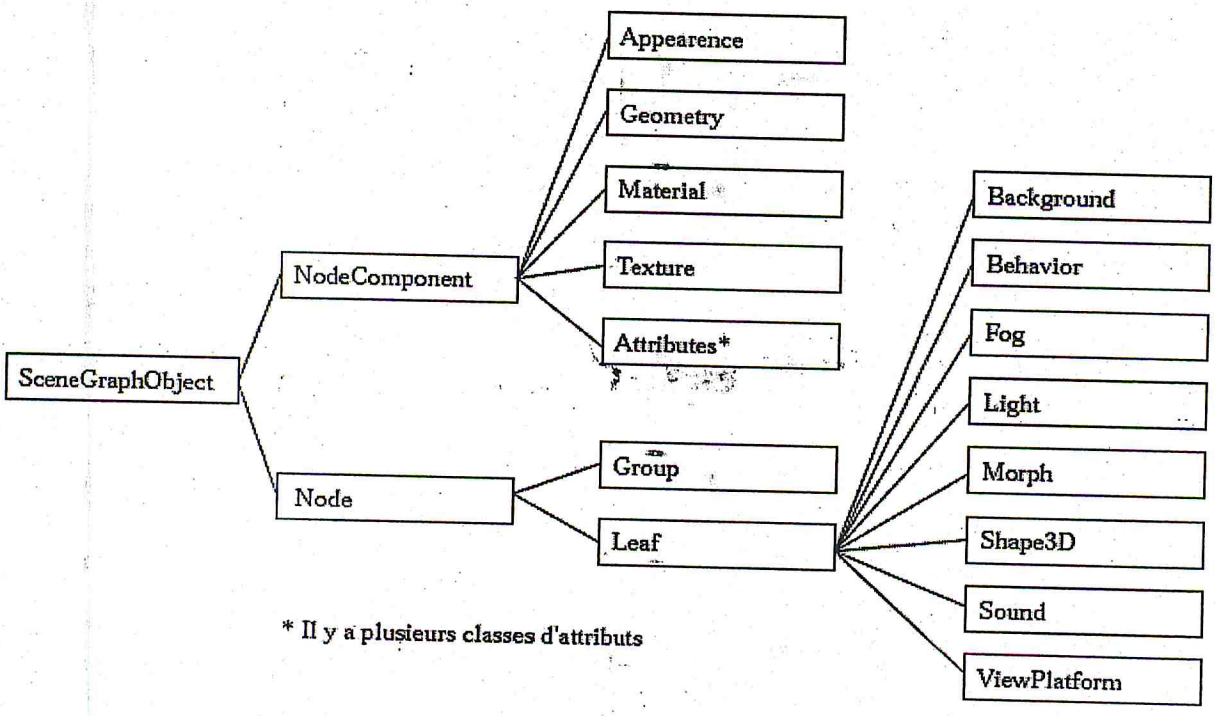


Figure 5.9 Hiérarchie partielle des classes de l'API Java 3D présentant les sous-classes de NodeComponent.

5.6.3 Définition des classes d'objets visuels :

Le même objet visuel pourra souvent apparaître plusieurs fois dans un univers simple. Il semble logique de définir une classe pour y créer les objets visuels à l'intérieur que de reconstruire les objets visuel à chaque fois. Il y a plusieurs manières de concevoir une classe définissant un objet visuel.

Le code 5.1 montre le squelette du code d'une classe VisuelObject, comme un exemple d'organisation générique possible pour une classe d'objet visuel. Les méthodes sont vides dans ce code. Le code du VisualObject ne peut pas apparaître dans les exemples fournis parce qu'il n'est pas particulièrement utile sous cette forme.

```
public class VisualObject extends Shape3D{

private Geometry voGeometry;
private Appearance voAppearance;

// crée un Shape3D avec une géométrie et une apparence
// la géométrie createGeometry
// l'apparence est crée dans la méthode createAppearance
public VisualObject() {

voGeometry = createGeometry();
voAppearance = createAppearance();
this.setGeometry(voGeometry);
this.setAppearance(voAppearance);
}

private Geometry createGeometry() {
// code pour créer une géométrie par défaut de l'objet visuel
}
```

```
private Appearance createAppearance () {
// code pour créer une apparence par défaut de l'objet visuel
}

} // fin de la classe VisualObject
```

Code 5.5 Squelette de code pour une classe VisualObject.

Prendre le Shape3D comme une base pour créer des classes d'objets visuels les rend facile d'utilisation dans un programme Java 3D. La classe VisualObject peut être utilisée aussi facilement que la classe ColorCube dans les exemples HelloJava3D du Chapitre 1. Le constructeur peut être appelé et le nouvel objet crée inséré comme un enfant d'un quelconque Group par une seule ligne de code. Dans les exemples suivants de ligne de code, objRoot est une instance de Group. Ce code crée un VisualObject et l'ajoute comme un enfant de l'objRoot au graphe scénique :

```
objRoot.addChild(new VisualObject());
```

Le constructeur du VisualObject fabrique le VisualObject par la création d'un objet Shape3D qui référence les composants de noeud crées par les méthodes createGeometry()et createAppearance(). La méthode createGeometry() crée un composant de noeud de Geometry qui sera utilisé par l'objet visuel. La méthode createAppearance() est responsable de la création du composant de noeud qui définira l'Appearance de l'objet visuel.

5.6.4 Les classes Geometry

En graphisme 3D sur ordinateur tout, depuis le simple triangle vers un jumbo jet (avion géant) le plus Compliqué, est modelé et rendu en prenant comme données de base des sommets. Avec Java 3D, chaque objet Shape3D se doit d'appeler sa méthode setGeometry() pour faire référence à un et un seul objet Geometry.

Pour être plus précis, Geometry est une super-classe abstract, de cette manière l'objet référencé est une instance d'une sous-classe de Geometry.

Les sous-classes de Geometry tombent dans trois vastes catégories :

- Geometry de base-sommet non-indexé (chaque fois qu'un objet visuel est rendu, ces sommets seront utilisés une seule fois)
- Geometry de base-sommet indexé (chaque fois qu'un objet visuel est rendu, ces sommets seront réutilisés).
- Autres objets visuels (les classes Raster, Text3D, et CompressedGeometry).

On n'a utilisé que la première catégorie mentionnée ci-dessus. La hiérarchie de classe pour les classes et les sous-classes de Geometry sont décrites par la figure 5.10 Hiérarchie de classe Geometry.

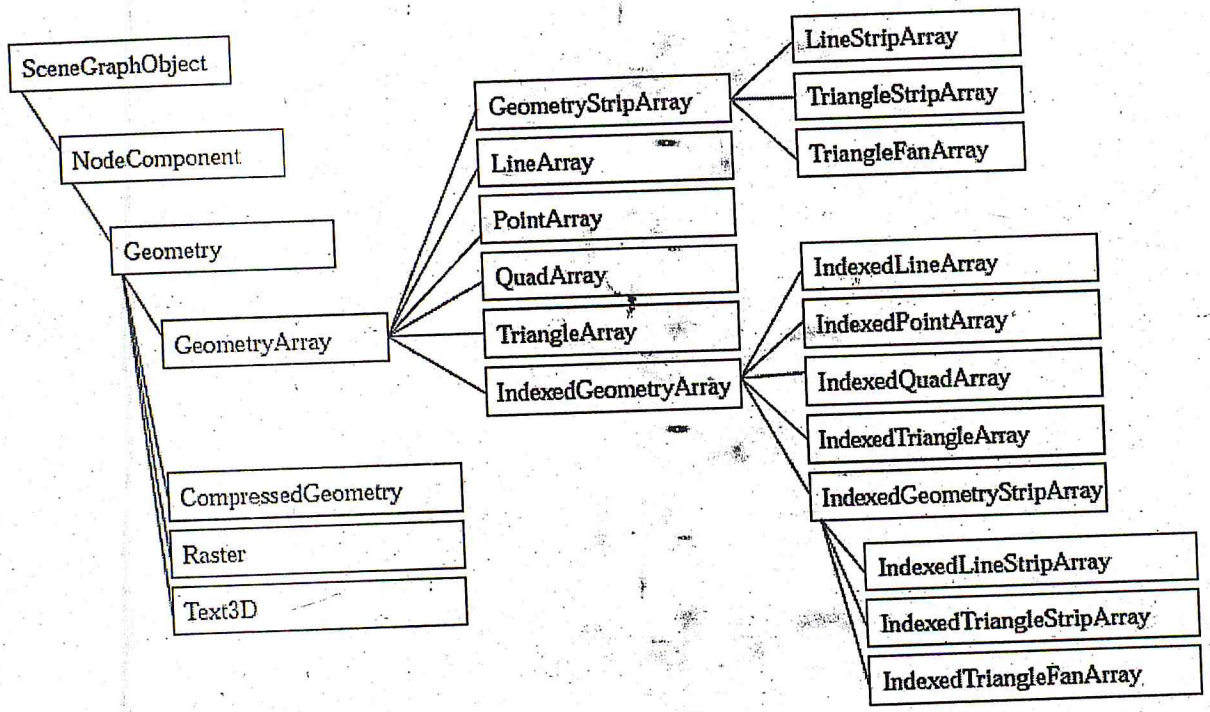


Figure 5.10 Hiérarchie de classe GeometryArray.

5.6.4.1 La classe GeometryArray

Comme vous devez en déduire par le nom de la classe, les sous-classes de Geometry peuvent être utilisées pour déterminer des points, des lignes, et des polygones pleins (triangles et quadrilatères). Les primitives de base-sommet sont des sous-classes de la classe abstract GeometryArray, laquelle indique que chacune d'entre-elles possède des tableaux qui conservent les données par somme

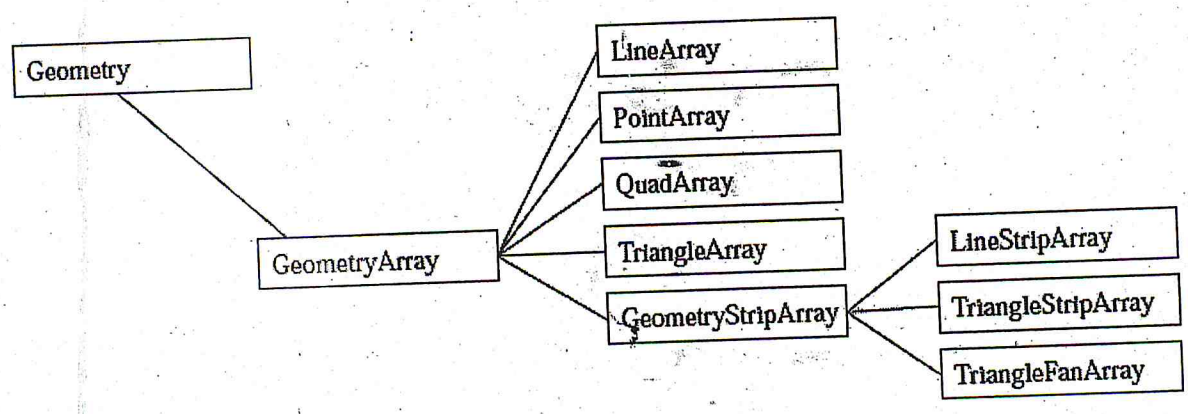


Figure 5.11 Hiérarchie de classe Geometry.

Les méthodes de GeometryArray (liste partielle) :

GeometryArray est la super-classe pour PointArray, LineArray, TriangleArray, QuadArray, GeometryStripArray, et IndexedGeometryArray.

void setCoordinate(int index, float[] coordinate)

void setCoordinate(int index, double[] coordinate)

void setCoordinate(int index, Point* coordinate)

Les sous-classes de GeometryArray :

Comme on a dit dans la Partie précédente, la classe GeometryArray est abstraite pour bien d'autres sous-classes utilitaires, comme LineArray. La figure 5.11 montre la hiérarchie de la classe GeometryArray et d'une partie de ces sous-classes. La principale distinction entre ces sous-classes est la façon dont le rendu Java 3D décide de rendre leurs sommets.

La figure 5.12 montre des exemples des quatre sous-classes GeometryArray : PointArray, LineArray,

TriangleArray, et QuadArray (les seules qui ne sont pas aussi des sous-classes de GeometryStripArray). Dans cette figure, les trois jeux de sommets les plus à gauche montrent les même six points sommets rendant six points, trois lignes, ou deux triangles. Le quatrième dessin montre quatre sommets définissant un quadrilatère. Notez qu'aucun des sommets n'est partagé : chaque ligne ou polygone plein est rendu indépendamment des autres.

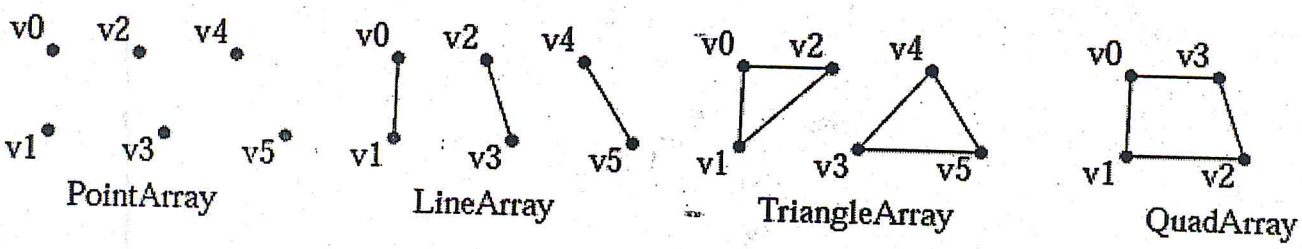


Figure 5.12 Sous-classes de GeometryArray

Les Segment dans notre logiciel sont représentés par des LineArray.

5.6.5 Classes utilitaires de géométrie :

Cette partie couvre les classes utilitaires pour la création de primitives géométriques boite, cône, cylindre, et sphère. Les primitives géométriques fournissent une seconde solution pratique pour créer du volume dans un univers virtuel.

Les classes utilitaires Box, Cone, Cylinder et Sphere sont définies dans le package com.sun.j3d.utils.geometry. La partie de la hiérarchie du package com.sun.j3d.utils.geometry qui contient les classes de primitive est montrée dans la figure 5.13

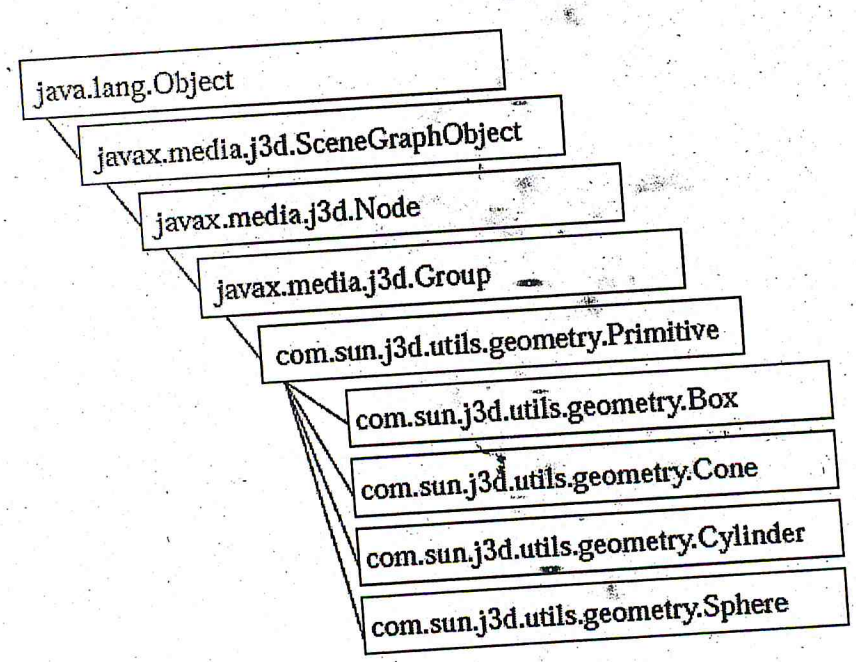


Figure 5.13 Hiérarchie de la classe d'utilitaires des primitives géométriques : Box, Cone, Cylinder, et Sphere.

5.6.5.1 Sphere :

La classe Sphere crée un objet sphérique centré à l'origine. Le rayon par défaut est de 1.0.

-Constructeurs de Sphere (liste partielle)

Package: com.sun.j3d.utils.geometry

Sphere étends Primitive, une autre classe du package com.sun.j3d.utils.geometry.

Sphere()

Construit une Sphere par défaut de rayon 1.0.

Sphere(float radius)

Construit une Sphere par défaut de rayon donné.

Sphere(float radius, Appearance appearance)

Construit une Sphere de rayon et d'apparence donné.

Les Joints dans notre projet sont représentés par des Sphères.

5.6.6 Apparence et attributs :

Les objets Shape3D peuvent faire référence à la fois à des objets Geometry et ou Appearence. Comme il est abordé dans la Partie 2.5, un objet Geometry détermine les informations par-sommet d'un objet visuel. Les informations par-sommet dans un objet Geometry peuvent spécifier la couleur des objets visuels. Les données dans un objet Geometry sont souvent insuffisantes pour décrire entièrement l'aspect d'un objet. Dans tous les cas, un objet Appearance est nécessaire. l'information qui décrit l'apparence d'une primitive géométrique est dite stockée à l'intérieur d'un « ensemble d'apparence », comme décrit à la Figure 5.14.

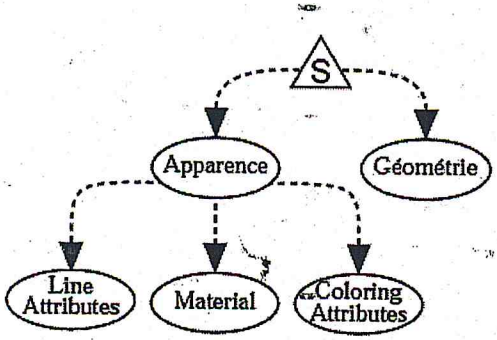


Figure 5.14 Un ensemble d'Appearence (appearence bundle).

Un objet Appearance peut se référer à plusieurs sous-classes de NodeComponent appelées les objets d'attributs d'apparence, incluant :

- PointAttributes
- LineAttributes
- PolygonAttributes

- ColoringAttributes
- TransparencyAttributes
- RenderingAttributes
- Material
- TextureAttributes
- Texture
- TexCoordGeneration

Un objet Appearance avec les objets attributs auxquels il se réfère est nommé appearance bundle (paquet d'apparence). Pour référencer n'importe lequel de ces composants de noeud, un objet Appearance possède une méthode avec un nom évident. Par exemple, pour un objet Appearance se référant à un objet ColoringAttributes, il utilise la méthode Appearance.setColoringAttributes().

```
ColoringAttributes ca = new ColoringAttributes();
ca.setColor (1.0, 1.0, 0.0);
Appearance app = new Appearance();
app.setColoringAttributes(ca);
Shape3D s3d = new Shape3D();
s3d.setAppearance (app);
s3d.setGeometry (someGeomObject);
```

Code 5.6 fragment de code pour l'Usage des objets Node Component Appearance et ColoringAttributes.

Le graphe scénique qui résulte de ce code se trouve dans la Figure 5.15.

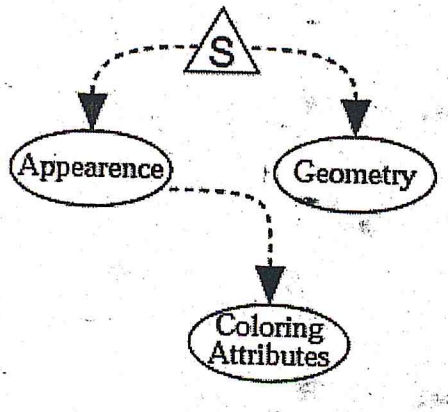


Figure 5.15 Le paquet d'apparence crée par le Fragment de code 5.2.

5.6.7 Implémentation :

5.6.7.1 Dessin des axes X, Y, et Z :

On utilise plusieurs objets LineArray pour dessiner des traits afin de représenter les axes x, y, et z.

Le fragment de code suivant montre l'instanciation des trois classe LineArrays.

```
// construit l'objet représentant l'axe X
 LineArray axisXLines= new LineArray (2, LineArray.COORDINATES);


 // construit l'objet représentant l'axe Y
LineArray axisYLines = new LineArray(2, LineArray.COORDINATES
| LineArray.COLOR_3);


// construit l'objet représentant l'axe Z
LineArray axisZLines = new LineArray(10, LineArray.COORDINATES
| LineArray.COLOR_3);
```

Code 5.3 Les constructeurs de GeometryArray

Le fragment suivant illustre une portion du code montrant l'usage d'une méthode permettant de stocker les valeurs de coordonnées.

```
axisZLines.setCoordinate(0, new Point3f(-1f, 0.1f, 0.9f));
axisZLines.setCoordinate(1, new Point3f( 0.9f, 0.1f, 0.1f));
axisZLines.setCoordinate(2, new Point3f( 0.9f, -0.1f, 0.1f));
axisZLines.setCoordinate(3, new Point3f(-0.9f, 0.1f, -0.1f));
```

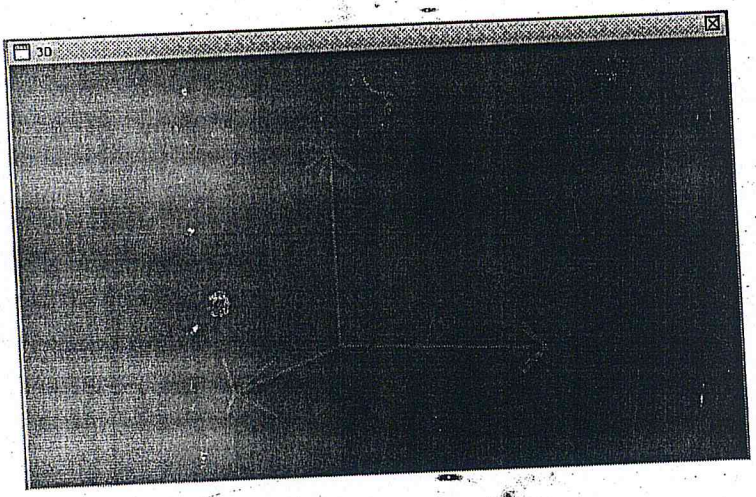


Figure 5.16 les Axes X, Y et Z

5.6.7.2 Dessin des noms des axes :

Pour ajouter du texte dans l'univers on a utilisé la classe Text2D .Les objets Text2D sont des polygones rectangulaires avec le texte appliqué sous la forme d'une texture. Text2d est une classe utilitaire qui étend Shape3D

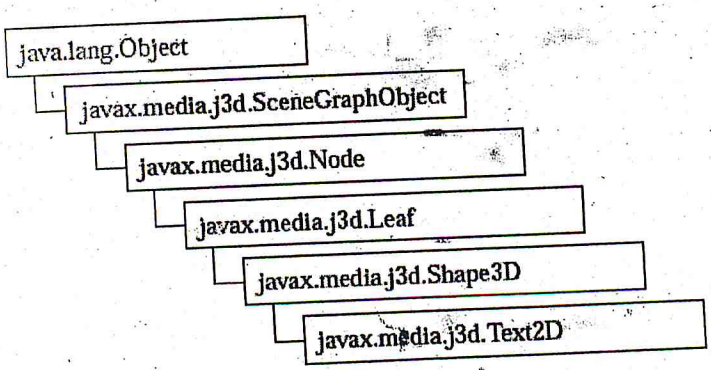


Figure 5.17 Hiérarchie de la classe Text2D.

Voici un fragment de code l'utilisant.

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.*;
import java.awt.Font;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.Text2D;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;

// Text2DApp rend un simple objet Text2D.

public class Text2DApp extends Applet {

public BranchGroup createSceneGraph() {
// Crée la racine de la branche du graphe.
BranchGroup objRoot = new BranchGroup();

// Crée un noeud de terminaison Text2D, l'ajoute au graphe scénique.
Text2D text2D = new Text2D("2D text is a textured polygon",
new Color3f(0.9f, 1.0f, 1.0f),
"Helvetica", 18, Font.ITALIC);
objRoot.addChild(text2D);
```

Code 5.4 Création d'un objet Text2D

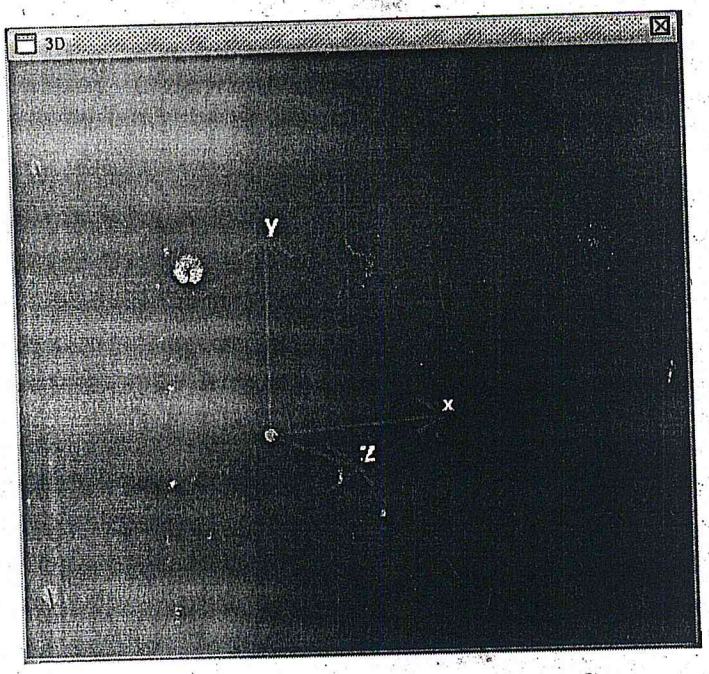


Figure 5.18 un imprimé écran montrant les axes surnommé

5.6.7.3 Dessin de la forme d'une structure Frames 3D, Frames 2D et Trusses 2D :

La classe Frames3D crée par notre system possède le constructeur suivant :

Frames3D(**int** nb_x,**int** nb_y,**int** nb_z,**double** taill_x[],**double** taill_y[],**double** taill_z[],Appearance AppLines,Appearance AppPoints)

Paramètres

int nb_x : nombre d'étages

int nb_y : nombre de travées suivant l'axe des x

int nb_y : nombre de travées suivant l'axe des y

double taill_x[] : largeur des travées suivant l'axe des x

double taill_y[] : largeur des travées suivant l'axe des y

double taill_z[] : largeur des travées suivant l'axe des z

Appearance AppLines : Apparence des LineArray par défaut

Appearance AppPoints : Apparence des PointArray par défaut

La méthode TransformGroup RetournerTransform() retourne un TransformGroup Contenant tout les Sphères et LineAttributes en place. Cette méthode Parcours un ensemble de Points (Point3d en java3d), qui forment des sommets qui formeront a leur tours des segments, tout en lui attribuant des noms (des entiers), elle leur associé aussi des Apparance(Couleurs).

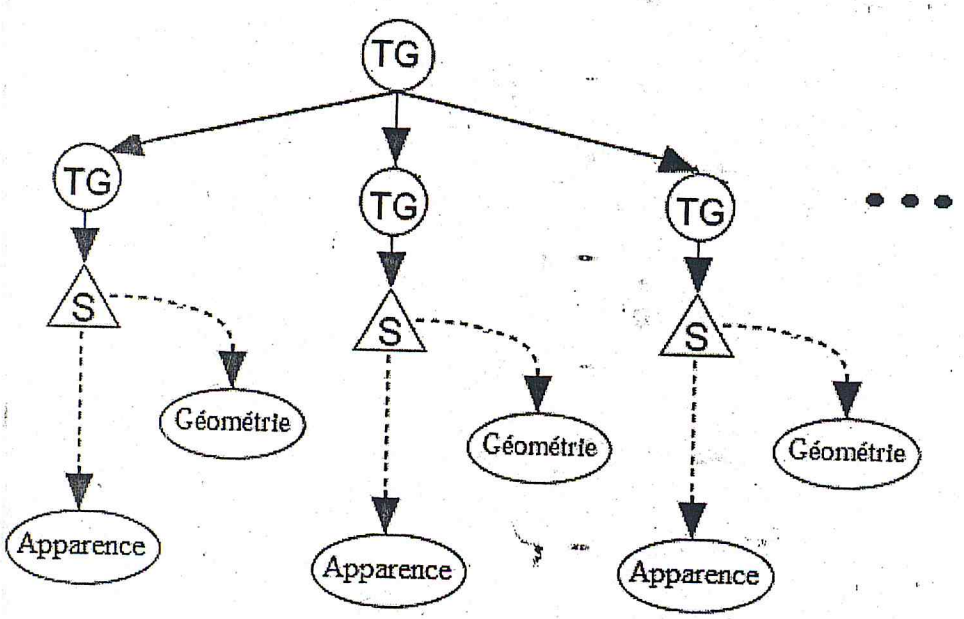


Figure 5.19 sous-Arbre générer par la méthode RetournerTransform()

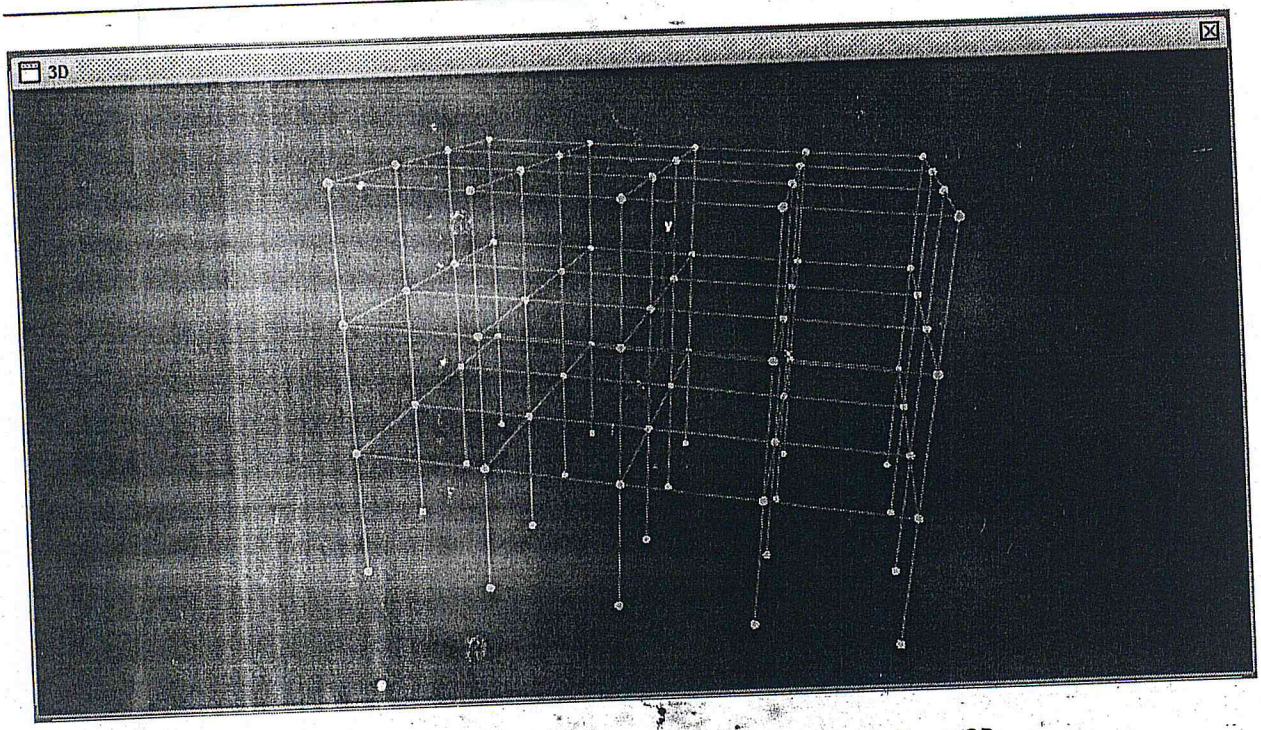


Figure 5.20 un imprimé écran montrant un exemple de Frame3D

La même procédure est appliquée, sauf le parcours des points, pour d'autre construction (Trusses2D, Frames2D).avec les constructeur de classe :

```
Frames2D(int nombre_d_etage,int nombre_de_trave,double[]
hauteurs_des_etage,double[] largeurs_des_trave){
```

```
Trusses2D(int nombre_de_division,double hauteur,double[]
longueur_des_divisions){
```

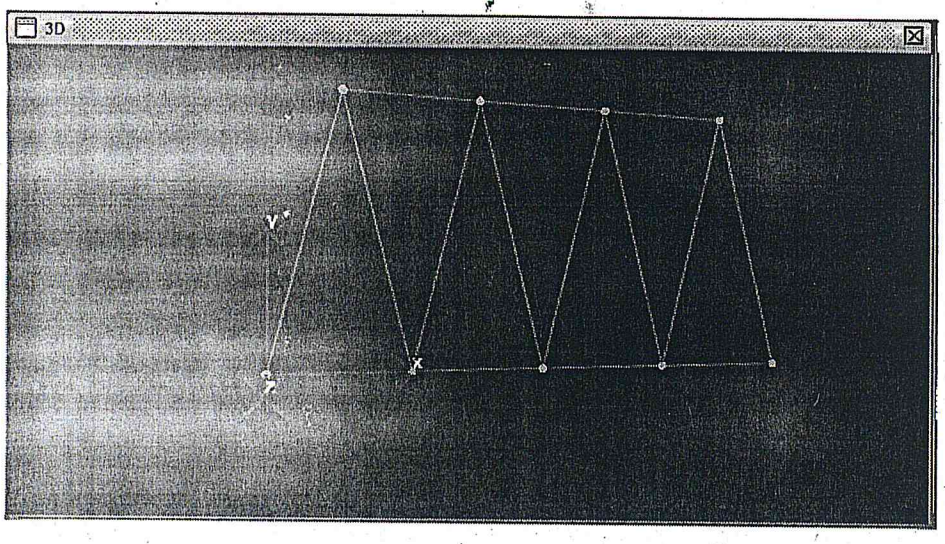


Figure 5.21 un imprimé écran montrant un exemple de Trusses2D.

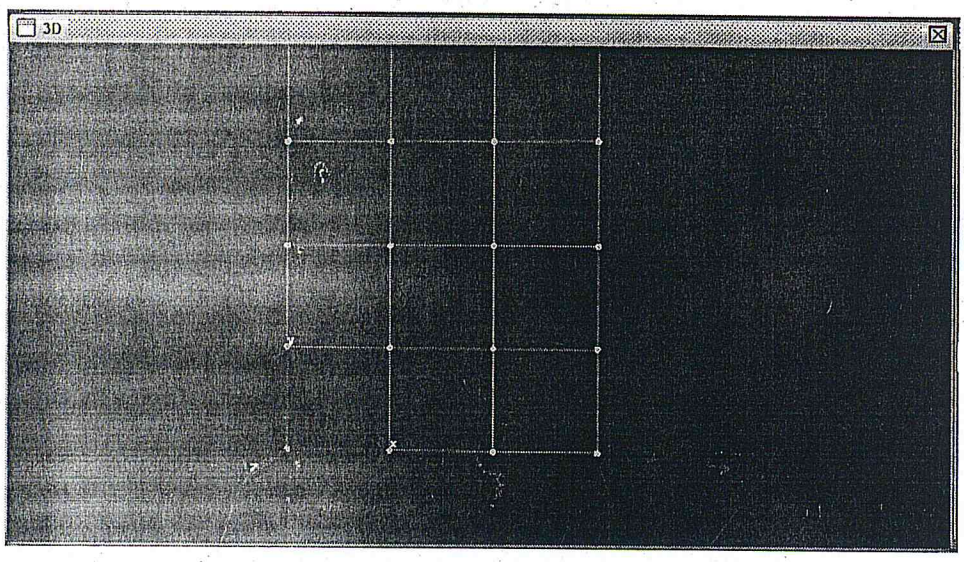


Figure 5.22 un imprimé écran montrant un exemple de Frame2D.

5.6.7.4 Interaction :

La classe Behavior (comportement) : La base des interactions et animations :

La classe Behavior est une classe abstraite qui fournit le mécanisme permettant d'inclure du code pour changer le graphique de scène, ou n'importe quel autre objet de la scène ,si l'objet cible le permet car y a ce qu'on appel des aptitudes pour chaque nœud déterminant si ou non ses propriété peuvent être modifier. Des erreurs seront provoquées si une mal conception était fait.

Le but d'un objet Behavior dans un graphique scénique est de changer le graphique de scène, ou des objets dans le graphique scénique, en réponse à un certain stimulus. Un stimulus peut être la presse d'une touche clavier, d'un mouvement de souris, de la collision des objets, du passage du temps, d'un autre événement, ou d'une combinaison de ces derniers. Les changements produits incluent ajouter des objets au graphique de scène, enlevant objecte du graphique de scène, changeant des attributs des objets dans le graphique de scène, réarrangeant objecte dans le graphique de scène, ou une combinaison de ces derniers. Les possibilités sont seulement limitées par les possibilités des objets de graphique de scène.

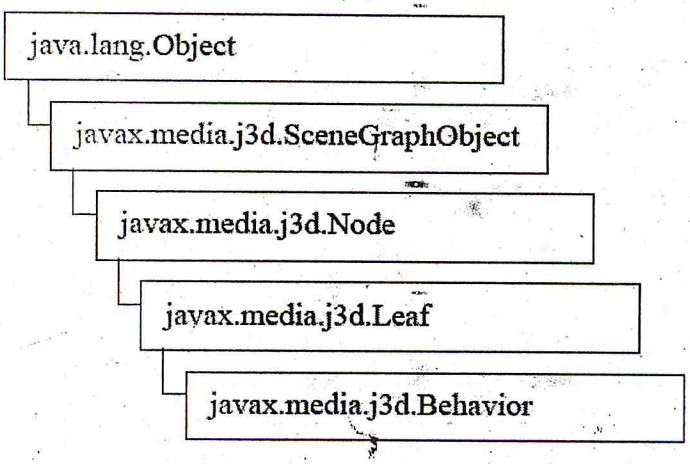


Figure 5.23 hierarchie pour Behavior dans l'API Java 3D.

-Application de Behavior

Puisqu'un Behavior est un lien entre un stimulus et une action, Considérer toutes les combinaisons des stimulus possibles et des actions possibles est de considérer les nombreuses applications des objets de Behavior. La table suivante examine le royaume des possibilités avec Behavior, énumérant les stimuli possibles en bas de la colonne gauche et les changements possibles à travers le dessus.

Le tableau ne présente pas toutes les applications possibles du Behavior, seulement le simple (des résultats d'un stimulus dans un changement). Celles-ci sont énumérées comme « spécifiques à l'application ». En outre, les combinaisons du stimulus et les combinaisons des actions sont possibles.

stimulus (reason for change)	object of change			
	TransformGroup (visual objects change orientation or location)	Geometry (visual objects change shape or color)	Scene Graph (adding, removing, or switching objects)	View (change viewing location or direction)
user	interaction	application specific	application specific	navigation
collisions	visual objects change orientation or location	visual objects change appearance in collision	visual objects disappear in collision	View changes with collision
time	animation	animation	animation	animation
View location	billboard	level of detail (LOD)	application specific	application specific

Figure 5.24 Applications du Behavior classées par catégorie par Stimulus et objet de changement

Dans le tableau, l'objet qu'un Behavior agit au moment est mentionné par l'objet du changement (object of change). Il est par cet objet, ou objets, que le Behavior affecte le monde virtuel.

Le mécanisme du Behavior est résumé en une recette pour l'implémentation

1. écrire (au moins un) constructeur

Enregistrer la référence de L'objet de changement (object of change)

2. redéfinir la méthode public void initialization()

Spécifier le réveille initial (wakeup criteria (trigger))

3. redéfinir la méthode public void processStimulus()

Décoder la condition "trigger"

Action accordé

Réaffecter des triggers approprié

Code 5.5 Recette pour l'écriture d'une classe Behavior simple

Les deux méthodes mentionné sont des méthodes de la classe abstraite Behavior. Dans l'ordre décrit dans la recette, la première est exécuter pour déclaré les trigger.

this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));

Tout un ensemble de classe est définit par l'API java 3D pour ces trigger. On ne peut pas détaillé.les noms de ces classe peuvent être vu dans l'appendice. La deuxième méthode est appeler après chaque réveille.

La recette du code montre les étapes de base pour la création d'une classe Behavior. Des Behaviors complexe requièrent beaucoup de pratique de programmation.

-Utilisation de Behavior

1. préparer le graphe scénique (par l'ajout d'un TransformGroup ou d'autre objet nécessaire)
2. insérer l'objet behavior dans le graphe scénique, référençant l'objet de changement
3. spécifier le "scheduling bounds", c'est une limitation d'utilisation du behavior
4. mètre a jour les aptitude (d'écriture et de lecture) de l'objet de changement

Code 5.6 recette pour l'utilisation d'une classe Behavior

Il existe des Behavior prédéfinit dans l'API , on peut cité MouseMove, MouseRotate, MouseTranslate ... en voici d'autre

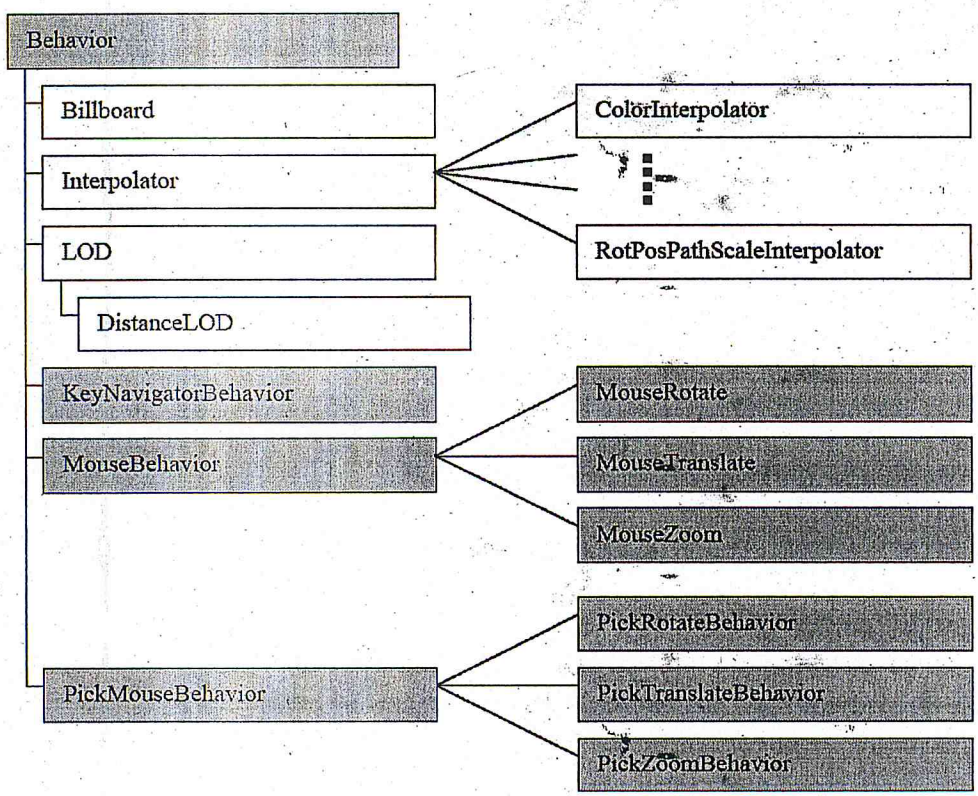


Figure **5.25** hiérarchie de la sous classe de Behavior

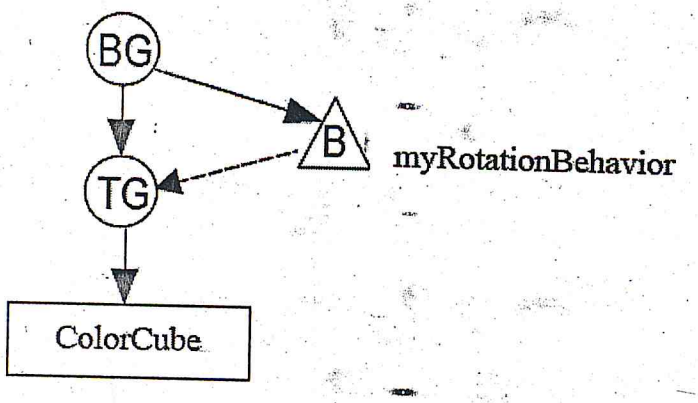


Figure 5.26 exemple d'un placement de behavior dans un graphe scénique

Dans la figure ci dessus, l'objet de changement est l'objet ColorCube en rectangle. Le Behavior touche la rotation de cet objet visuel.

Utilisation de Behavior dans notre projet :

Navigation

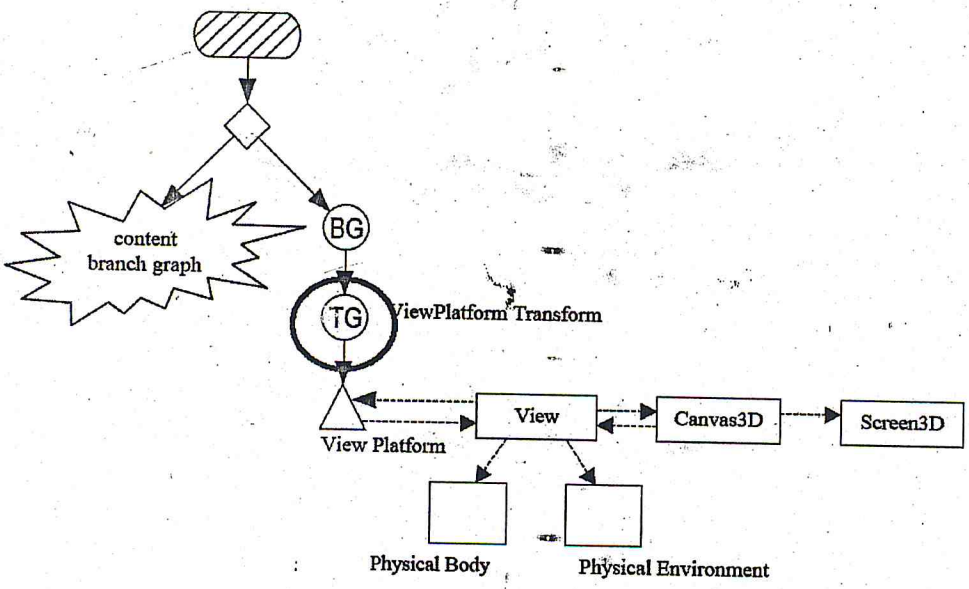


Figure 5.27 branche visuel

Le transformGroup encercler en rouge permet de contrôler la vue, il dispose de l'information sur la position de la « caméra », si on le modifie (modifier le Transform3d correspondant, ce qui revient a modifié la matrice associé au Transform3d) on verra que la « caméra » bougera (subi une translation), tourne autour d'elle, ou autour d'un axe, ou d'un point (subi une rotation).

KeyNavigatorBehavior, montré dans la figure 5.25, est un Behavior qui est assez intéressant. On a suivi la recette d'utilisation de Behavior pour l'utilisé. c'est ce TransformGroup(montré dans la figure 5.27) qui sera l'objet de changement et c'est avec les touches claviers (figure 5.28) qu'on pourra naviguer dans l'univers.

Key	movement	Alt-key movement
←	rotate left	lateral translate left
→	rotate right	lateral translate right
↑	move forward	
↓	move backward	
PgUp	rotate up	translation up
PgDn	rotate down	translation down
+	restore back clip distance (and return to the origin)	
-	reduce back clip distance	
=	return to center of universe	

Figure 5.28 mouvements de KeyNavigatorBehavior

Sélection d'objets (Picking)

--Selection simple

Le Picking (sélection) est implémenté par un behavior déclenché par l'événement « click de souris » .Dans la sélection d'objet visuel, l'utilisateur place la souri derrière l'objet de son chois, et click sur le bouton de la souris. le behavior est déclenché par ce click souris et lance l'opération de sélection d'objet (picking opération).un rayon est projeté dans l'univers virtuel depuis la position pointu par la souris et parallèle avec la projection. L'intersection de ce rayon avec les objets visuel du monde virtuel est calculée. L'objet visuel intersecté le plus proche de l'image plate est selectionné.la figure 5.29 montre un rayon projeté dans l'univers virtuel.

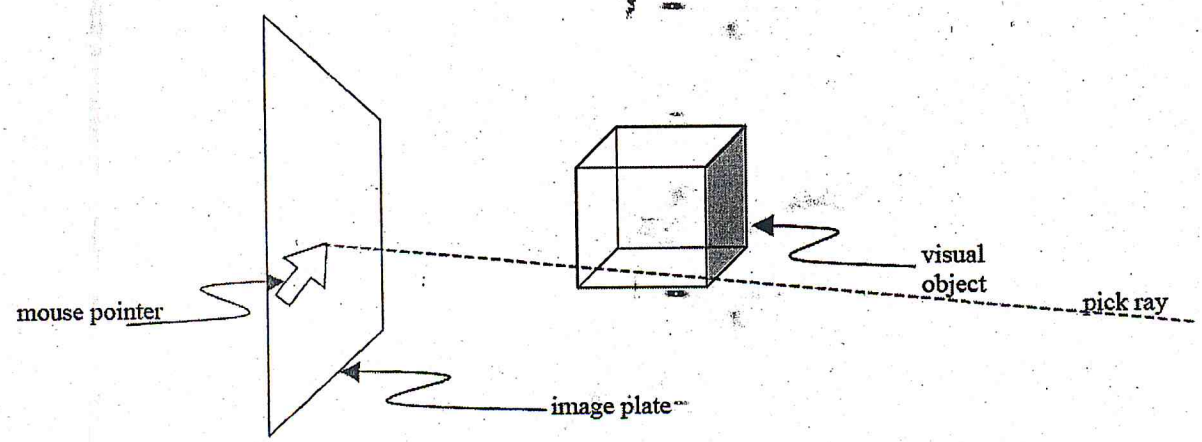


Figure 5.29 Projection de PickRay dans l'univers virtuel

Constructeur de PickRay

PickRay(Point3d origine, Vector3d direction)

Les paramètres sont très logiques, un vecteur et un point feront l'affaire.

L'intersection est très couteuse en temps de calcul, surtout si la scène est complexe. Java3D fournis beaucoup de solution pour en réduire cette contrainte .Une voie importante est de bien définir les objets pouvant être sélectionnés. un node avec setPickable() met à false ne peut être sélectionnée.

--Selection rectangulaire

La classe **PickShape est une classe abstraite disposant de plusieur sous classe, on peut cité** PickBounds, PickRay, PickSegment, Pick .avec PickBounds on peut définir des limites pour la définition d'une région,ces régions sont définit par l'union, intersection de volumes définit généralement par des équations.et après cela Java 3D effectuera l'intersection avec le monde virtuel.

La sélection rectangulaire dans un univers 3D semble simple. Un non connaissons pense directement qu'il s'agie d'intersecté les objets visuels du monde virtuel, avec un parallélogramme, définit par les 4 coints du rectangle crée par la souris. Alors que c'est faut .il faut plutôt penser a une pyramide.

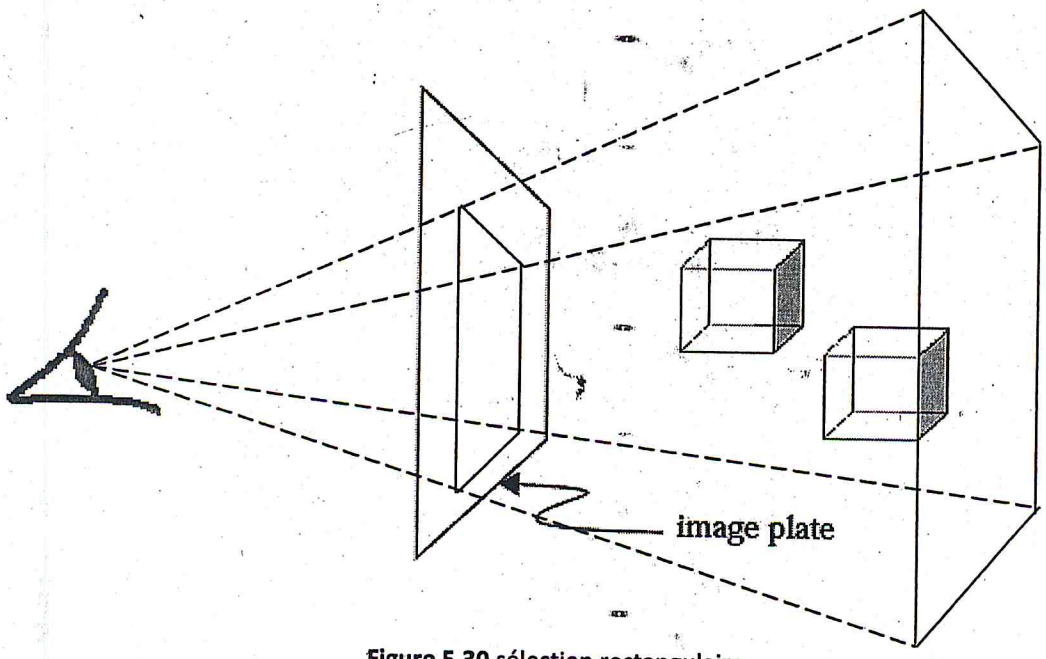


Figure 5.30 sélection rectangulaire

Un BoundingPolytope est un Bounds (une Limitation) définissant un volume par l'intersection de plusieurs volumes, ces volumes sont des demi-espaces

Un BoundingPolytope est un Bounds (une Limitation).Un BoundingPolytope définit une région polyèdre utilisant l'intersection de quatre demi-espaces ou plus. La région définie par un BoundingPolytope est toujours convexe.

Chaque plan dans le BoundingPolytope spécifie un demi-espace défini par l'équation :

$$Ax + By + Cz + D \leq 0$$

Où A, B, C, D sont les paramètres qui spécifient le plan dont l'équation est

$$Ax + By + Cz + D = 0$$

On a définit dans notre système la méthode

Vector4d plan(Point3d a,Point3d b,Point3d c)

Retournant les Paramètre A, B, C et D d'un plan passant par trois points. le Vector4d , comme son nom l'indique est un vecteur de quatre double, qui sont respectivement A,B,C et D. plus de détail sur cette méthode voir Annexe C.

La sélection rectangulaire consiste à définir en premier lieu un rectangle, définit en un plan de projection (dans une vue). Les points définissant le rectangle serons convertie par la suite en coordonnées dans l'univers virtuel en fonction du vue où le rectangle a été crée. Apre, un objet BoundingPolytope est instancié prenant en paramètre quatre Vector3d (quatre demi-espaces). Ces vecteurs sont crée par la méthode plan(Point3d a,Point3d b,Point3d c),de manière a ce qu'on obtient ce qu'on veut, voir figure 5.30

Remarque : la position de l'œil entre dans le calcul

5.7 Démonstration de manipulation du logiciel:

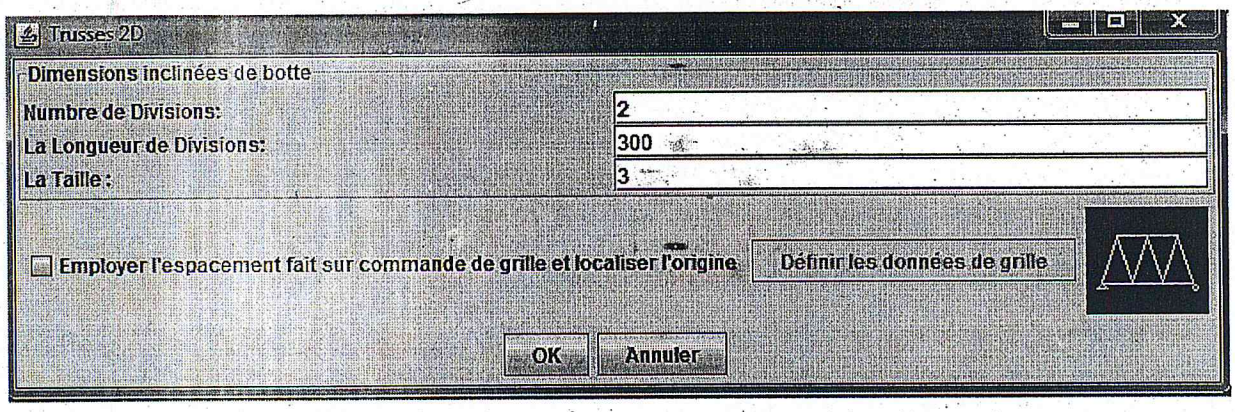


Figure 5.31 Fenêtre pour la configuration des paramètres Trusses 2D.

Apre confirmation Le système affichera ceci

En bas c'est indiquer :

- Les coordonnées du curseur dans le monde virtuel
- Le nombre d'éléments (joints + segments) sélectionnée
- Le dernier élément sélectionné

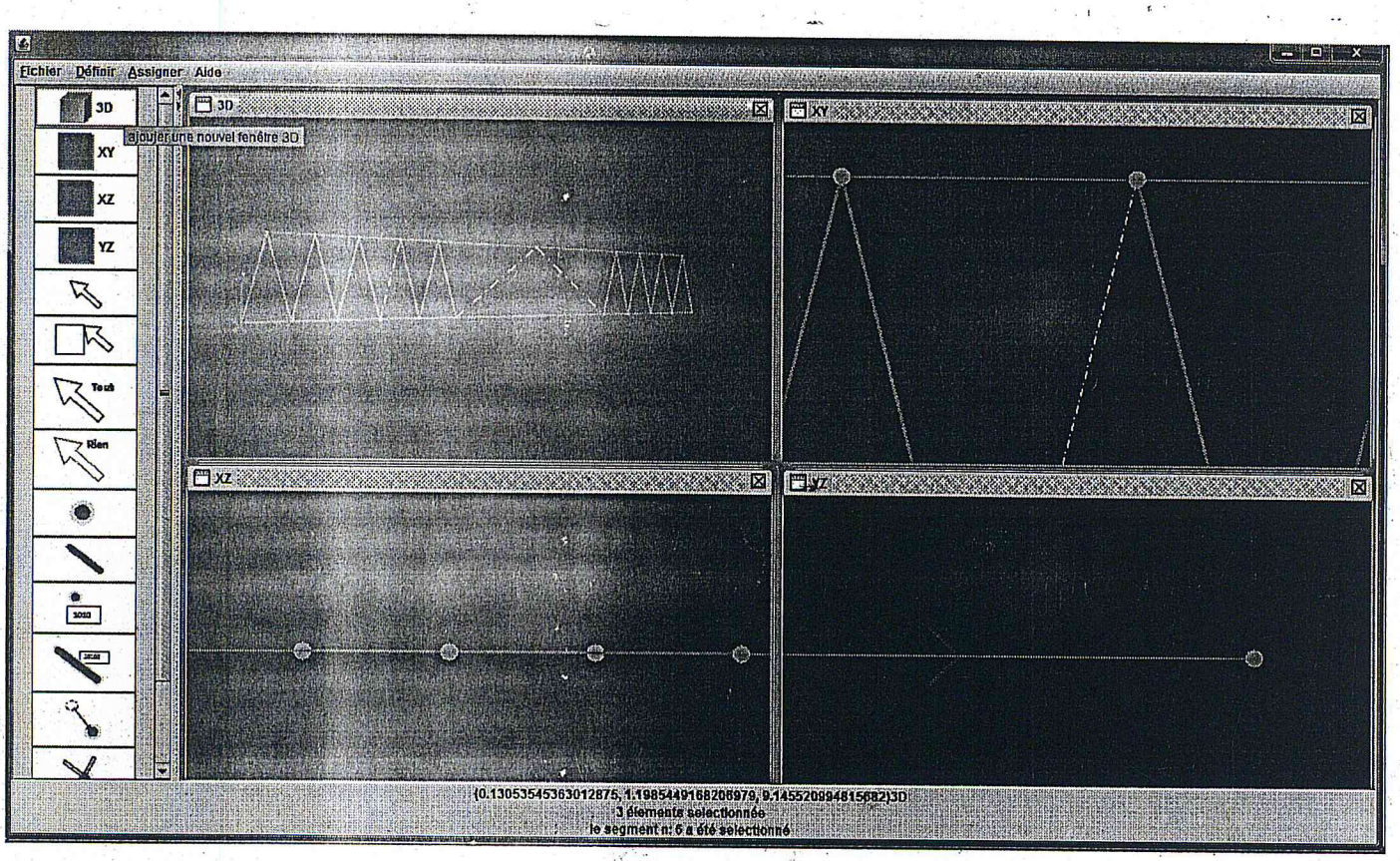


Figure 5.32 Imprimé écran du logiciel pour un projet Trusses 2D.

Dans le menu à gauche on a respectivement :

- ajouter une nouvelle vue 3D
- ajouter une nouvelle vue en plan XY
- ajouter une nouvelle vue en plan XZ
- ajouter une nouvelle vue en plan YZ
- Sélection simple (la nouvelle sélection=la sélection précédente U le nouveau élément sélectionnée)
- Sélection rectangulaire
- Sélectionner tout les éléments
- Désélectionner tout les éléments
- Création de joints
- Création de segments
- Création de joints par formulaire

- Création de segments par formulaire
- Déplacer les joints sélectionnés
- Supprimer les joints sélectionnés
- Supprimer les segments sélectionnés

Avec ces outils notre système peut être alimenter, modifier..

On peut crée de nouveau joints et de nouveau segments

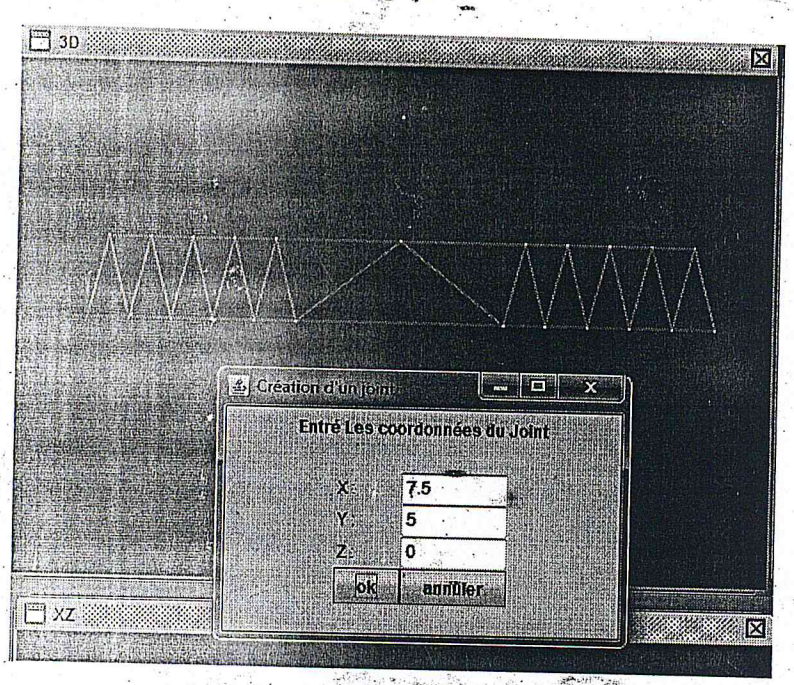


Figure 5.33 Création d'un joint par formulaire.

Le joint est affiché en un petit point en haut.

Avec ce joint on peut crée de nouveaux segments

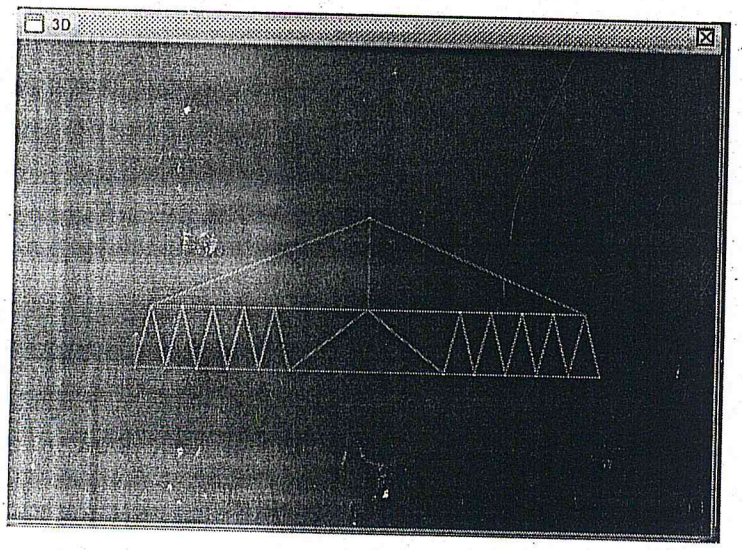


Figure 5.34 Exemple d'ajout de segment

Remarque : les segments et joints crée sont automatiquement nommer.

Pour modifier la structure, et pour ne pas supprimer et recréer. La palette à gauche comprend l'outil de déplacement  .cet outil permet de translater des joint, en spécifiant un vecteur de translation.

Le déplacement du joint influence sur la position des segments qui sont relié avec lui. Voici un exemple

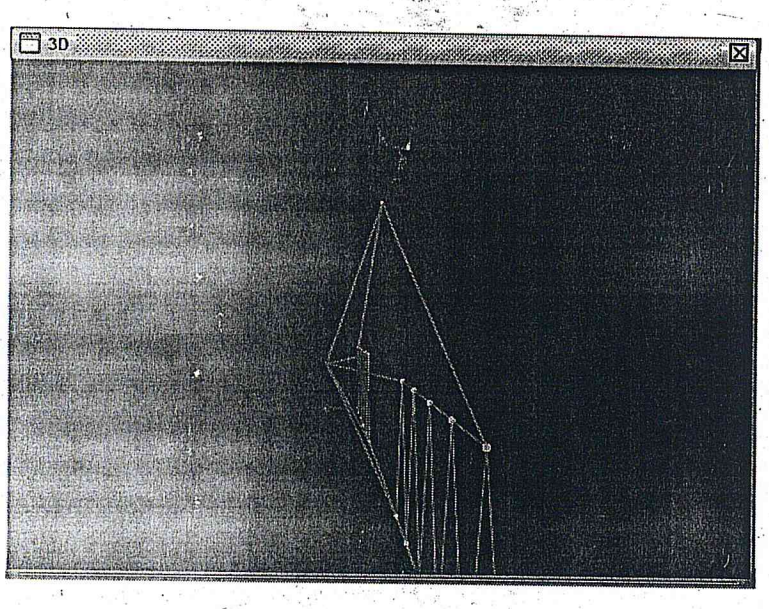


Figure 5.35 Influence du déplacement d'un joint sur les segments.

Un effet semblable survient lors de la suppression d'un joint. Tous les segments qui étaient relié avec lui seront automatiquement supprimer. Voici un exemple de suppression de joint qui était au milieu.

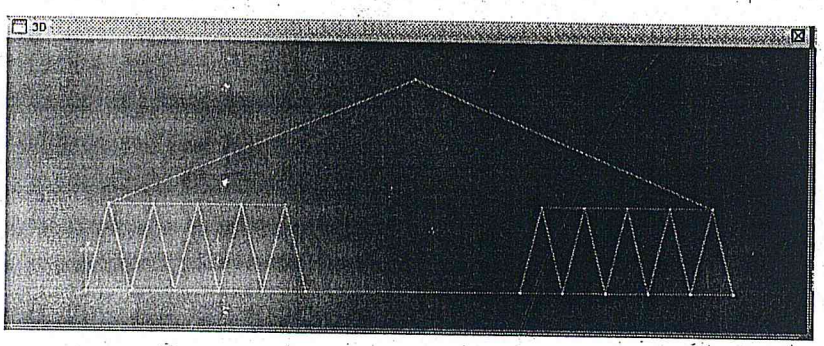


Figure 5.36 Influence de la suppression d'un joint sur les segments.

Remarque : cette suppression a été faite juste pour montrer un exemple.

Ces outils permettent de bien définir la géométrie de la structure, mais serrent aussi comme plateforme pour définir les propriétés des joints et des segments.

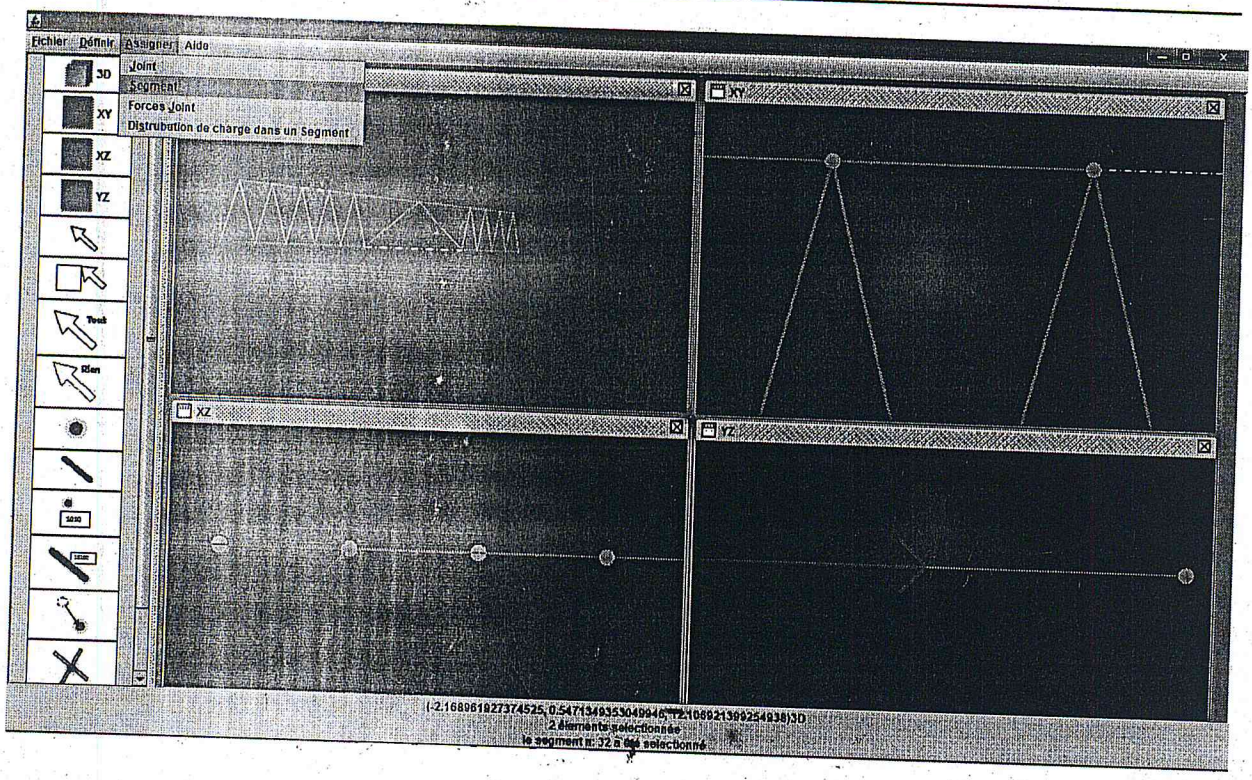


Figure 5.37 imprimé écran de manipulation de l'application.

5.8 Conclusion :

Suite à la modélisation qu'on a élaborée dans le chapitre de conception, nous avons présentée dans ce chapitre une implémentation qui met en valeur notre application. Nous soulignons que nous avons eu à manipuler un grand nombre de composant Java. Nous avons découvert beaucoup de formalités offertes par ce langage surtout dans le domaine de la 3D.

Conclusion générale

Conclusion générale:

L'objet de ce projet était de développer une application orienté vers le domaine de Génie Civil, afin de fournir des informations nécessaires qui aide l'ingénieur à faire l'analyse structurale. Après une étude bibliographique des méthodes et des nouvelles technologies dans le domaine de logiciel, notre projet s'est inspiré de plusieurs projets informatiques destinés au Génie Civil.

Nous avons choisi comme modèle le logiciel SAP2000 qui est le fruit de 50 années de recherche, avec la différence, que nous ne sommes pas intéressés au calcul de structures, chose que SAP2000 traite, mais nous avons développé une interface graphique avec une visualisation en 3D et des projections en plan et généré une base de données qui contient toutes les informations nécessaires d'aide à la décision pour l'ingénieur en génie civil.

Notre travail sera exploité par un binôme en génie civil. Il développera ce qu'on a appelé précédemment le system B. On a collaboré avec ce binôme et leur avons donné une copie de notre structure de base de données. Avec le système complet, une analyse de structure est possible.

Les applications utilisant la 3D ne manquent pas. La puissance de calcul des ordinateurs évoluant constamment, et l'implication de l'informatique augmentant chaque jour dans la vie quotidienne de chacun d'entre nous, l'utilisation de la 3D va encore s'amplifier, et évoluer. Les outils qui sont au cœur de ce développement évoluent également. Le développement des moteurs 3D et des outils qui y contribuent, est donc en perpétuelle évolutions, guidés par les innovations des cartes vidéo, et poussant toujours plus loin le réalisme des images.

L'activité de conception reste bien évidemment l'affaire de l'homme de métier, qui seul dispose de facultés créatrices s'appuyant sur ses connaissances et ses expériences professionnelles. C'est cette interaction entre l'outil informatique et l'être humain qui fait des miracles architecturaux.

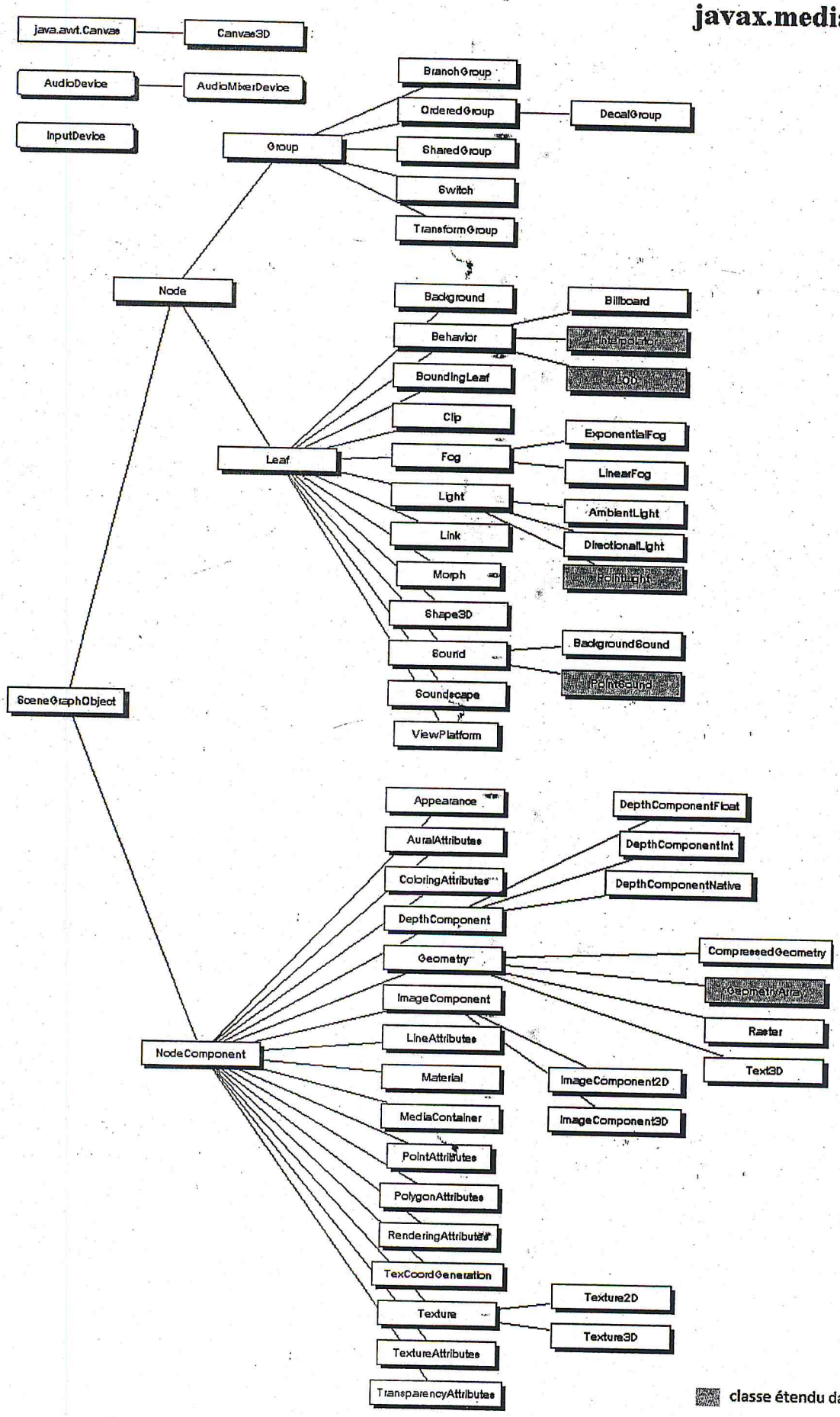
Après cette étude, on a pu localiser l'ensemble des outils entrant dans la réalisation du projet. Un tableau de bord offrant ces outils est disponible dans notre système, mettant en place tout ce qu'un projet de génie civil a besoin pour, premièrement la conception, et bien sûr le paramétrage des objets, pour que notre système englobe tous les éléments entrant dans le calcul de structures.

Pour notre part, ce projet nous a permis d'améliorer nos compétences concernant le développement du logiciel, puisque il s'agissait de programmer des scripts en langage Java. Par ailleurs, nous nous somme familiarisés avec beaucoup de connaissances du domaine de génie civil.

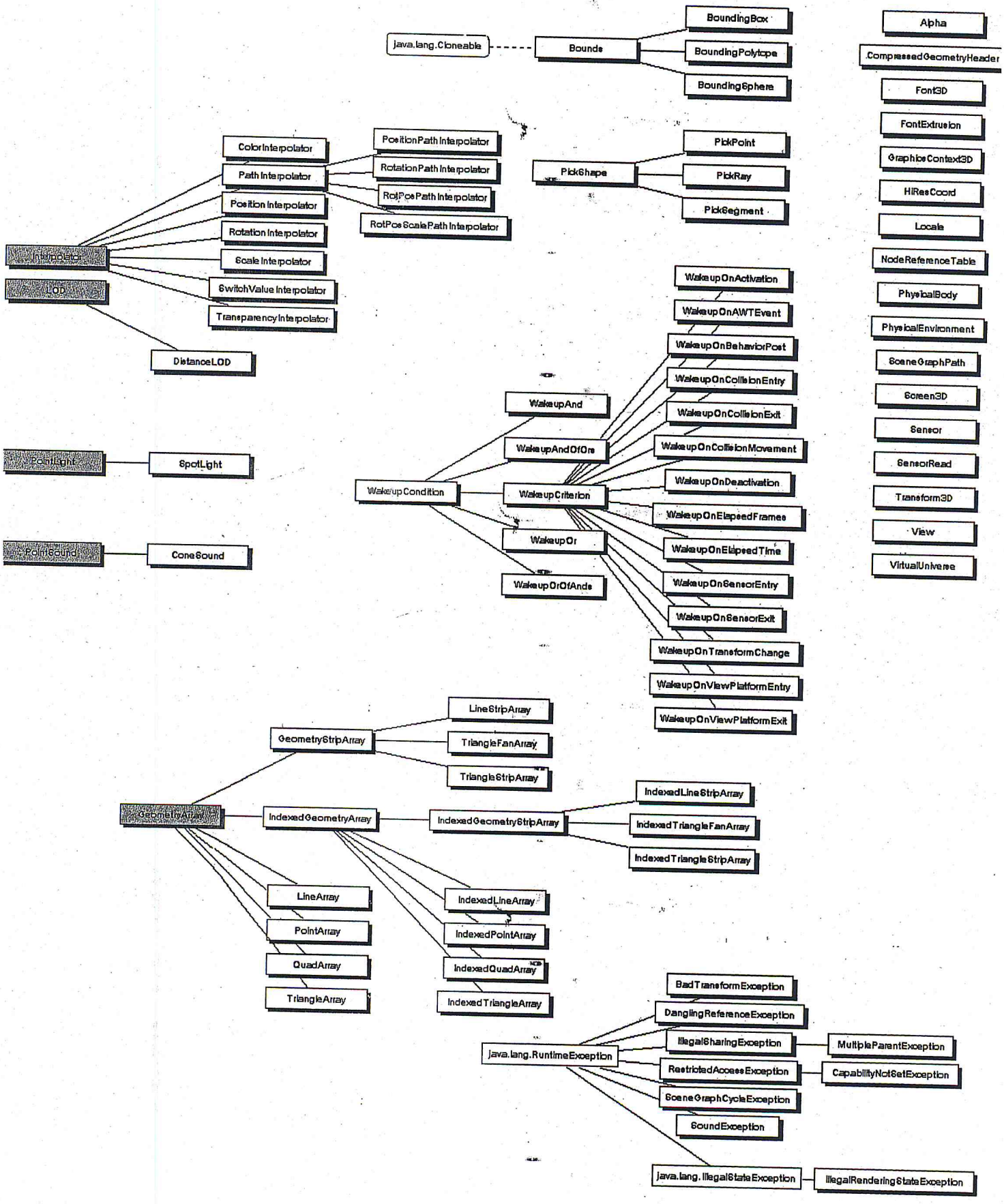
Annexe

A

javax.media.j3d



classe étendu dans la page suivante


java.lang.Cloneable
Bounds
BoundingBox
BoundingPolytope
BoundingSphere
Alpha
CompressedGeometryHeader
Font3D
FontExtrusion
GraphicsContext3D
HiResCoord
Locale
NodeReferenceTable
PhysicalBody
PhysicalEnvironment
SceneGraphPath
Screen3D
Sensor
SensorRead
Transform3D
View
VirtualUniverse
PickShape
PickPoint
PickRay
PickSegment
Interpolator
ColorInterpolator
PathInterpolator
PositionInterpolator
RotationInterpolator
ScaleInterpolator
SwitchValueInterpolator
TransparencyInterpolator
PositionPathInterpolator
RotationPathInterpolator
RotPosPathInterpolator
RotPosScalePathInterpolator
LOD
DistanceLOD
PointLight
SpotLight
PointSound
ConeSound
WakeupCondition
WakeupAnd
WakeupAndOfOrs
WakeupCriterion
WakeupOr
WakeupOrOfAnds
WakeupOnActivation
WakeupOnAWTEvent
WakeupOnBehaviorPost
WakeupOnCollisionEntry
WakeupOnCollisionExit
WakeupOnCollisionMovement
WakeupOnDeactivation
WakeupOnElapsedFrames
WakeupOnElapsedTime
WakeupOnSensorEntry
WakeupOnSensorExit
WakeupOnTransformChange
WakeupOnViewPlatformEntry
WakeupOnViewPlatformExit
GeometryArray
GeometryStripArray
LineStripArray
TriangleFanArray
TriangleStripArray
IndexedGeometryArray
IndexedGeometryStripArray
IndexedLineStripArray
IndexedTriangleFanArray
IndexedTriangleStripArray
LineArray
PointArray
QuadArray
TriangleArray
IndexedLineArray
IndexedPointArray
IndexedQuadArray
IndexedTriangleArray
java.lang.RuntimeException
BadTransformException
DanglingReferenceException
IllegalSharingException
MultipleParentException
RestrictedAccessException
CapabilityNotSetException
SceneGraphCycleException
SoundException
java.lang.IllegalStateException
IllegalRenderingStateException

Annexe

B

Les classes Java 3D utilisées dans HelloJava3D

Pour mieux comprendre l'API Java 3D et l'exemple HelloJava3D, un résumé de chaque classe utilisée dans le programme d'exemple HelloJava3D est présenté ici.

La classe BranchGroup :

Les objets de cette classe sont utilisés pour produire les graphes scéniques. Les instances de BranchGroup sont les racines des points de branchements. Les objets BranchGroup sont les seuls autorisés à être des enfants d'objets Locale. Les objets BranchGroup peuvent avoir de multiples enfants. L'enfant d'un objet BranchGroup peut être d'autres objets Group ou Leaf (terminaison).

Constructeur par défaut du BranchGroup

Branchgroup ()

La classe Canvas3D :

La classe Canvas3D est dérivée de la classe Canvas de l'Abstract Windowing Toolkit (AWT). Au moins un objet Canvas3D doit être référencé dans la branche visuelle du graphe scénique.

Constructeur du Canvas3D

Canvas3D(GraphicsConfiguration graphicsconfiguration)

Construit et initialise un nouvel objet Canvas3D que Java 3D peut rendre en donnant un objet GraphicsConfiguration valide.

La classe Transform3D :

Les objets Transform3D représentent les transformations de la géométrie 3D comme la translation et la rotation. Ces objets sont typiquement utilisés dans la création d'objets TransformGroup. Premièrement, on construit l'objet Transform3D, combinaison possible d'objets Transform3D. On construit ensuite l'objet TransformGroup en utilisant l'objet Transform3D.

Un objet de transformation est généralement représenté intérieurement par une matrice 4x4 de double précision à virgule flottante. C'est une représentation mathématique de haut-niveau. L'objet Transform3D n'est pas utilisé dans le graphe. Il est utilisé pour déterminer les transformations de l'objet TransformGroup.

Le Constructeur par défaut de Transform3D

Transform3D()

Construit un objet Transform3D qui représente la matrice identifiée (pas de transformation). Un objet Transform3D peut représenter une translation, une rotation, un changement d'échelle, ou une combinaison de ces transformations. Pour déterminer une rotation, l'angle est exprimé en radians.

Les Méthodes de Transform3D (liste partielle)

void rotX(double angle)

Règle la valeur cette transformation de rotation autour de l'axe x dans le sens des aiguilles d'une montre.

void rotY(double angle)

Règle la valeur cette transformation de rotation autour de l'axe y dans le sens horaire.

void rotZ(double angle)

Règle la valeur cette transformation de rotation autour de l'axe z dans le sens horaire.

void set(Vector3f translate)

Règle la valeur de translation de la matrice par le paramètre Vector3f.

La classe TransformGroup :

Comme une sous-classe de la classe Group, les instances de TransformGroup sont utilisées dans la création des graphes scéniques et possèdent une collection d'objets nœuds enfants. Les objets TransformGroup conservent les transformations géométriques comme la translation et la rotation. La transformation est crée de façon typique dans un objet Transform3D, celui-ci n'étant pas un objet du graphe scénique.

Les Constructeurs de TransformGroup

TransformGroup()

Construit et initialise un TransformGroup en utilisant une transformation désignée.

TransformGroup(Transform3D t1)

Construit et initialise un TransformGroup depuis l'objet Transform3D qui lui est transmis.

Paramètres:

t1 - l'objet transform3D

La classe Vector3f :

Vector3f est une classe mathématique située dans le package javax.vecmath pour déterminer un vecteur utilisant trois valeurs à virgule flottante. Les objets Vector sont souvent utilisés pour designer des translations de la géométrie. Les objets Vector3f ne sont pas utilisés directement dans la construction du graphe scénique. Ils sont utilisés pour déterminer des translations, des surfaces, des lignes, ou d'autres choses.

Les constructeurs de Vector3f

Un vecteur de trois éléments est représenté par un simple point précis à virgule flottante de coordonnées x, y et z.

Vector3f()

Construit et initialise un Vector3f à (0,0,0).

Vector3f(float x, float y, float z)

Construit et initialise un Vector3f depuis les coordonnées spécifiées x,y,z..

La classe ColorCube :

ColorCube est une classe utilitaire du package com.sun.j3d.utils.geometry qui définit la géométrie et les couleurs d'un cube placé à l'origine avec des couleurs différentes sur chaque face. ColorCube étend la classe Shape3D ; c'est donc un noeud de terminaison (Leaf). Le ColorCube est simple d'utilisation quand on le pose dans un univers virtuel. Les dimensions par défaut de l'objet ColorCube sont de 2 mètres de hauteur, largeur et profondeur. Si on place un cube sans lui attribuer de rotation (comme dans HelloJava3D), la face rouge est visible dans le cadre de visualisation.
Les autres couleurs sont bleu, magenta, jaune, vert, et cyan.

Les Constructeurs du ColorCube

ColorCube()

Construit un cube de couleur de taille par défaut. Par défaut, un coin est placé à 1 mètre de chaque axes depuis l'origine, donnant un cube centré à l'origine et dont tous les cotés font 2 mètres.

ColorCube(double scale)

Construit un cube de couleur mis à l'échelle par la valeur spécifiée. La taille par défaut étant de 2 mètres pour chaque coin. Le ColorCube qui en résulte place les coins aux positions (scale, scale, scale) et (-scale,-scale, -scale).

Annexe

C

Equation du plan passant par trois points

Soit donnée 3 points a, b et c dans l'espace disjoint et ne forment pas une ligne.

Apre quelque calcul, on c'est aboutit à l'équation du plan passant par ces points.

$$Ax + By + Cz + D = 0$$

Tel que

$$A = (y_b - y_a) * (z_c - z_a) - (y_c - y_a) * (z_b - z_a);$$
$$B = -(x_b - x_a) * (z_c - z_a) - (x_c - x_a) * (z_b - z_a);$$
$$C = (x_b - x_a) * (y_c - y_a) - (x_c - x_a) * (y_b - y_a);$$
$$D = -x_a * A - y_a * B - z_a * R.z;$$

Voici un programme java retournant ce résultat dans un `Vector4d`

```
Vector4d plan(Point3d a,Point3d b,Point3d c){
      Vector4d R=new Vector4d();

R.x=(b.y-a.y)*(c.z-a.z)-(c.y-a.y)*(b.z-a.z);
R.y=-(b.x-a.x)*(c.z-a.z)-(c.x-a.x)*(b.z-a.z);
R.z=(b.x-a.x)*(c.y-a.y)-(c.x-a.x)*(b.y-a.y);
R.w=-a.x*R.x-      a.y*R.y      -a.z*R.z;

      return R;
}
```

Annexe

D

Les forces appliquées sur les éléments d'une structure

L'extension (ou traction):

On dit qu'une poutre travaille en extension simple lorsqu'elle est soumise à des actions mécaniques dont les torseurs associés se réduisent, au centre de surface des sections extrêmes, à deux résultantes directement opposées et qui tendent à l'allonger.

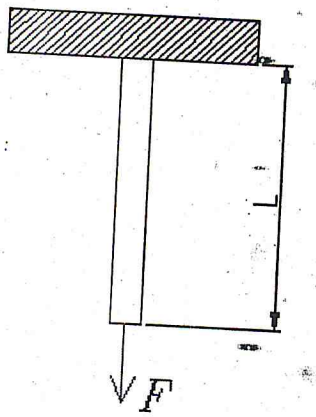


Figure 1.25 La traction.

Une barre de section S est fixée au plafond, on considère la fixation rigide. Sous l'action de la force F, la barre s'allonge. On applique une contrainte dans cette barre.

- s : la contrainte en N/mm²
- F : la force en N
- S : la section en mm².

On a :

$$s = F/S$$

Pour chaque matériau, on a plusieurs valeurs fondamentales :

- R : la résistance à la rupture par extension (en Mpa) 1 Mpa = 1 N/mm²
- Re : la limite élastique
- Rp : la résistance pratique à l'extension.
- E : module d'élasticité longitudinale en N/mm² (Pour l'acier, E= 200 000 à 300 000 N/mm²).
- G : module d'élasticité transversale en N/mm² (On considère G = 2/5 E).
- Rp est la valeur que l'on utilise dans les calculs.
- Cs est un coefficient de sécurité (on prend pour Cs une valeur de 4 à 10).

On a :

$$Rp = R / Cs$$

La compression :

On dit qu'une poutre travaille en compression simple lorsqu'elle est soumise à des actions mécaniques dont les torseurs se réduisent, au centre de surface des sections extrêmes, à deux résultantes directement opposées et qui tendent à la raccourcir.

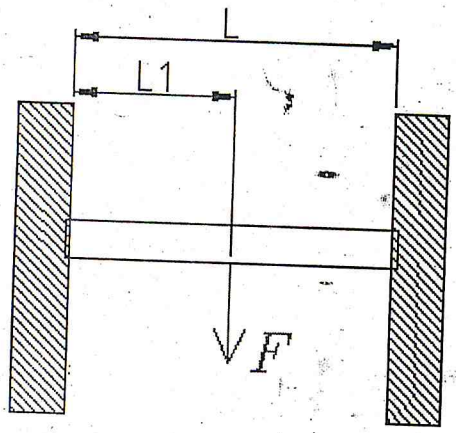


Figure 1.26 La compression.

C'est la même chose que l'extension, mais au lieu de tirer, on pousse (on comprime) la barre. Dans ce cas, on a aussi : $s = F/S$

Le flambage :

Le flambage ou flambement est un phénomène d'instabilité d'une structure, qui soumise à un effort normal de compression, a tendance à fléchir et se déformer dans une direction perpendiculaire à l'axe de compression (passage d'un état de compression à un état de flexion).

Figure 1.27 Le flambage.

Cette flexion latérale est appelée voilement ou flambage.

La torsion :

On appelle torsion le fait d'appliquer à un objet un couple (Force x distance)

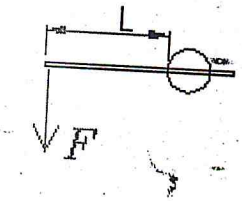


Figure 1.28 La torsion.

On a :

$$a = M / (G \times I0)$$

Avec :

- M : moment du couple de torsion en N.mm.
- G : module d'élasticité transversale.
- I0 : Moment d'inertie polaire en mm4.
- a angle de torsion par unité de longueur (en radian par mm)

La flexion :

La flexion est la déformation d'un objet qui se traduit par une courbure. Dans le cas d'une poutre, elle tend à rapprocher les deux extrémités de la poutre. Dans le cas d'une plaque, elle tend à rapprocher deux points diamétralement opposés sous l'action.

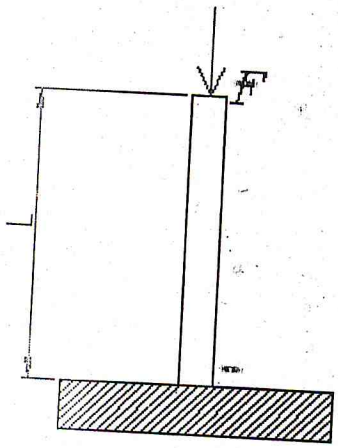


Figure 1.29 La flexion.

Lorsqu'une poutre fléchit, sa partie supérieure est soumise à une compression et sa partie inférieure est soumise à une extension.

On a :

$$h = M/(Iab/v)$$

Avec :

- h contrainte en N/mm².
- M : moment fléchissant en N.mm.
- Iab/v : module d'inertie en mm3 (Pour une section rectangulaire : Iab/v = bh²/6 et pour une section circulaire : Iab/v I = 0.1 d3).

Suivant la configuration le moment fléchissant est différent (position et répartition de la charge, poutre posée ou encastrée).

Dans le cas de mon schéma (poutre encastrée, charge au milieu), nous avons :

$$f = (1/192) \times ((PL3)/E \times Iab).$$

Avec :

- f : flèche en mm.
- P : poids en N.
- L : longueur en mm.
- E : module d'élasticité longitudinale en N/mm².
- I : ici en mm4.
- Puisque la section rectangulaire alors : Iab = bh3/12.

Unité de mesure

Définition :

Les unités sont des étalons pour la mesure de grandeurs physiques qui ont besoin de définitions précises pour être utiles. Les systèmes d'unité, définis en cherchant le plus large accord dans le domaine considéré, sont rendus nécessaires par la méthode scientifique dont un des fondements est la reproductibilité des expériences (et donc des mesures), ainsi que par le développement des échanges d'informations commerciales ou industrielles.

Par convention, les noms d'unité sont des noms communs, ils s'écrivent en minuscule (« kelvin » et non « Kelvin », « ampère » et non « Ampère » malgré MM. Kelvin et Ampère) et prennent la marque du pluriel (un volt, deux volts). Par contre, l'abréviation utilise une majuscule si le nom dérive de celui d'une personne (V pour volt / Volta, A pour ampère / Ampère), sinon une minuscule (m pour mètre).

Unités de base du système international :

Le système international comprend sept unités de base indépendantes (ou unités fondamentales) à partir desquelles sont obtenues par analyse dimensionnelle toutes les autres unités, ou unités dérivées qui sont les unités de masse, longueur, temps, courant électrique, température, intensité lumineuse et quantité de matière.

Dans notre projet nous allons utiliser seulement les unités suivantes :

Unité de longueur(L) : Le mètre (symbole m, du grec metron) est l'unité de base de longueur du Système international (SI). Il est défini, depuis 1983, comme la distance parcourue par la lumière dans le vide en 1/299 792 458 seconde et la taille d'un pied est d'environ 0,30 m.

Unité de masse (M): Le kilogramme (symbole kg) est l'unité de masse dans le Système international d'unités (SI). Le kilogramme est actuellement défini comme la masse de ce prototype au pavillon de Breteuil, un cylindre en platine iridié (90 % platine et 10 % iridium) de 39,17 mm de diamètre et 39,17 mm de haut4 déclaré unité SI de masse depuis 1889 par le Bureau international des poids et mesures.

Comme l'unité de base « kilogramme » comporte déjà un préfixe, les préfixes SI sont ajoutés par exception au mot « gramme » ou à son symbole g, bien que le gramme ne soit qu'un sous-multiple du kilogramme (1 g = 10-3 kg).

Unité de température thermodynamique: Le kelvin (symbole K, du nom de Lord Kelvin), unité de température thermodynamique, est la fraction 1/273,16 de la température thermodynamique du point triple de l'eau (H2O), et une variation de température de 1 K est équivalente à une variation de 1 °C1. Toutefois, à la différence du degré Celsius, le kelvin est une mesure absolue de la température qui a été introduite grâce au troisième principe de la thermodynamique. La température de 0 K est égale à -273,15 °C et correspond au zéro absolu (le point triple de l'eau est donc à la température 0,01 °C).

Le degré Fahrenheit (°F, caractère Unicode U+2109) est aussi une unité de mesure de la température, qui doit son nom au physicien allemand Daniel Gabriel Fahrenheit, qui la proposa en 1724. Dans l'échelle des températures en Fahrenheit, le point de solidification de l'eau est de 32 degrés, et son point d'ébullition est de 212 degrés. On en déduit qu'une différence d'un degré Fahrenheit équivaut à une différence de 5/9 de kelvin ou de degré Celsius.

Unité de force : Le newton (symbole : N) est l'unité SI de force nommée ainsi en l'honneur d'Isaac Newton pour ses travaux en mécanique classique. Un newton est la force capable de communiquer à une masse de 1 kilogramme une accélération de 1 m/s². Il faut donc un newton pour augmenter la vitesse d'une masse de 1 kilogramme de 1 mètre par seconde chaque seconde (ou encore de 3,6 kilomètres à l'heure chaque seconde Note 1). Cette unité dérivée du système international s'exprime en unités de base ainsi :

$$1N=1\frac{Kg\,.\,m}{s\,.\,s}$$

Références bibliographiques

Références bibliographiques

[QCRM] Dobrescu C.Alexandru, 1992 , Quelques chapitres de résistance des matériaux, Office Des Publication Universitaires.

[CM] Manfred A.hirt, 1994, Construction Métallique, l'école polytechnique fédérale de Lausanne.

[ADS] Mohamed osman Zakaaria, 1989, Analyse Des Structure, Office Des Publication. Universitaires.

[MAT] Nourredine BOUAOUADJA, 1991, Matériaux, Office Des Publication Universitaires.

[SIA3D] R.Malgouyres, 2002, Algorithmes pour la synthèse d'image et l'animation 3D,Dunod.

[J3dp] Daniel Selman , 2003, Java 3D programming, Manning.

[Gsj] Dennis J Bouvier, 2004, Getting started with the java3D API, K Computing (SUN).

[OPDLM3D] Copigneaux Bertrand et Dalmasso Nicolas, 2004, outils pour le développement logiciel de moteur 3D, rapport de licence en informatique (université de Nice).