MIG-004-113-1

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique.

Mémoire pour l'obtention
D'un diplôme d'ingénieur d'état en informatique.

Option : IA

Sujet :

ÉTUDE ET RÉALISATION D'UN FRAMEWORK POUR LA MODÉLISATION 3D SOUS J2ME

Présenté par

Messaoudi Mounir

Promoteur : Mohamed Mahieddine

Organisme d'accueil : LRDSI, université de Blida

2005-2006

REMERCIEMENTS

Tout d'abord, je tiens à remercier Dieu tout puissant pour m'avoir préservé, donné la santé et guidé vers la connaissance et le savoir.

Je tiens à exprimer toute ma reconnaissance à Monsieur Mohammed Mahieddine docteur à l'université de Blida et membre dans le laboratoire de recherche et de développement des systèmes d'information(LRDSI) pour m'avoir encadré et guidé tout au long de ce mémoire et pour les précieux conseils qu'il ma prodigués. Il a fait preuve d'une grande disponibilité à l'aboutissement de ce mémoire.

J'adresse mes remerciements à tout les les membres du jury qui m'ont fait l'honneur d'accepter de juger mon travail.

Je remercie tous les enseignants de la faculté des sciences et surtout mes enseingants du département informatique.

Je remercie toutes les personnes qui m'ont aidé par leur soutien et leur encouragement à accomplir ce travail.

Je tiens enfin à présenter mes remerciements les plus chaleureux à ma famille et en particulier à mes parents. Je veux leur témoigner toute gratitude et ma reconnaissance pour leur soutien constant.

Dédicace

Je dédie ce mémoire

A mes très chers parents pour leur soutiens et leur patients durant tout ma carrière d'étude,

Pour leurs bienveillance, leur efforts et l'encouragements dans mes études,

A tout mes frères et mes sœurs,

A tout ma familles de proche et de loin

A mes très chères amies : Amine, Bilel, Mehdi, Redha, Farouk , les frères Taleb, Krimo, Omar(Smina), Abd el hak(Nouni), Riadh ,Ramzi ,...

A toute ma promotion,

A toute personne qui me connaît

RÉSUMÉ

Comparé à un ordinateur individuel, les terminaux mobiles ont typiquement une capacité de traitement plus faible, moins de capacité mémoire et une résolution inférieure d'affichage. Tandis que les deux premiers facteurs sont clairement des inconvénients pour les applications de rendu 3D sur les terminaux mobiles, le facteur d'affichage est un avantage pour le rendu 3D. Cependant le processus de visualisation 3D ne peut pas profiter de l'affichage plus petit parce qu'il nécessite un temps d'exécution qui dépend du nombre de faces a rendus. En revanche, le temps d'exécution des méthodes de rendu 3D dépend principalement de la résolution d'affichage.

Les patterns fournissant un ensemble de techniques qui facilitent l'analyse, la conception et la réalisation de système informatiques. Les frameworks sont des sous systèmes prêt à être utiliser avec des possibilités bien définies d'adaptation et d'extension.

L'objectif de notre travail est d'étudier les patterns et les frameworks afin de réaliser un framework pour le graphisme 3D sous la plateforme J2ME. Dans le but de tester notre framework nous avons implémenté un moteur de rendu 3D destiné à des téléphones portables de la plateforme J2ME (CLDC 1.1, MIDP 2.0).

Mots clés :

Graphisme 3D, Moteur 3D, Patterns, Frameworks, J2ME, MIDP, CLDC, Midlet, Java, terminaux mobiles, UML.

Abstract

Compared with a personal computer, the mobile terminals have typically a lower processing capacity, less memory size and a lower resolution of posting.While the first two factors are disadvantages for the applications of 3d rendering on the mobile terminals, the factor of posting is an advantage for 3D render. the process of 3D visualization cannot benefit from smaller posting because it requires an execution time which depends on the number of faces rendered.

The patterns providing a whole of techniques that facilite the analysis, the design and the realization of information system.The frameworks are under systems loan with being to use with well defined possibilities of adaptation and extension.

The aim of this work is studing the patterns and the frameworks in order to develop a framework for 3d graphics under J2ME platform. In order to test testing our framework we have implemented an 3D engine for portable telephones of platform J2ME (CLDC 1.1, MIDP 2.0).

Key words:

Graphics 3D, 3D Engine, Patterns, Frameworks, J2ME, MIDP, CLDC, mobile terminals, Midlet, Java, UML.

Table des Matières

Remerciments **1**

Dédicace **2**

Résumé **3**

Liste des Figures **8**

Liste des Tables **11**

Introduction **12**

CHAPITRE 1. J2ME et la 3D dans les terminaux mobiles **15**

1.1. Introduction 15

1.2. La plateforme J2ME (Java 2 Micro Edition) 15

1.2.1. Présentation de J2ME 15

1.2.2. Les configurations et les profils 17

1.2.3. La machine virtuelle 18

1.2.4. La configuration CLDC 18

1.2.5. MIDP (Mobile Information Device Profile) 19

1.3. La 3D dans les terminaux mobiles 25

1.3.1. Vue sur l'ensemble du marché 25

1.3.2. Types de l'implémentation 3D 26

1.3.3. Les Logiciels dans les mobiles (Système d'exploitation, langages et APIs) 26

1.4. Conclusion 28

CHAPITRE 2. Le graphisme 3D **29**

2.1. Introduction 29

2.2. Généralités sur la modélisation 3D 29

2.2.1. Définition 29
2.2.2. Les différents types de modélisations 30
2.2.3. Les bases de la 3D temps réel 34
2.2.4. Optimisation possible du pipeline 3D 40
2.2.5. Elimination des parites cachées 41
2.3. La lumière 48
2.3.1. Les modèles d'illumination 48
2.3.2. Atténuation de la lumière 51
2.3.3. Les modèles d'ombrage 51
2.4. Conculsion 53

CHAPITRE 3. LES PATTERNS ET LES FRAMEWORKS **54**

3.1. Introduction 54
3.2. Les patterns 54
3.2.1. Définition 54
3.2.2. Classification des patterns de conception: 55
3.2.3. Exemples des patterns de conception (Design patterns) 57
3.3. Les frameworks 61
3.3.1. Définition d'un framework 61
3.3.2. Pourquoi utiliser un framework ? 61
3.3.3. Structure d'un Framework 62
3.3.4. Types de frameworks 64
3.3.5. Cycle de vie d'un framework 65
3.3.6. Utilisation de frameworks 67
3.4. Conclusion 68

CHAPITRE 4. DÉVELOPPEMENT DE L'APPLICATION **69**

4.1. Introduction 69
4.2. Analyse 70
4.2.1. Analyse des besoins 70
4.2.2. Sélection des patterns 72

4.3. Conception et implémentation du framework 73
4.3.1. L'architecture du Framework 74
4.3.2. La classification 76
4.3.3. Les modules mathématiques 77
4.3.4. Les classes de bases pour les images et le graphisme 2D 79
4.3.5. Le module du moteur 3D 80
4.3.6. La classe Observable 81
4.3.7. Le module des caméras 83
4.3.8. La modélisation des modèles 3D simples 84
4.3.9. La modélisation de la scène 3D 85
4.4. Implémentation et testes 88
4.4.1. Implémentation du moteur 3D 89
4.4.2. Implémentation des classes de graphique et les images 90
4.4.3. L'implémentation de la caméra 91
4.4.4. L'implémentation des modèles d'illuminations 92
4.4.5. Les testes de rendu 94
4.5. Conclusion 101

Conclusion générale **102**

Annexe **104**

Annexe A **104**

Bibliographie **109**

Liste des Figures

Figure 1.1. L'architecture de la plateforme J2ME 16
Figure 1.2. Les utilisations possibles des profils 17
Figure 1.3. Cycle de vie d'une Midlet 20
Figure 1.4. Les interfaces d'utilisateurs de MIDP 22
Figure 2.1. Ambiguïté du modèle en fil de fer 30
Figure 2.2. Modèle avec des facettes 32
Figure 2.3. Modèle surfacique 33
Figure 2.4. Les différentes étapes du pipeline 3D. 35
Figure 2.5. Les deux repères utilisés en 3D. 36
Figure 2.6. Les transformations 3D 37
Figure 2.7. Déterminations du vecteur normal à une face et test de visibilité 38
Figure 2.8. Calcul de la projection. 39
Figure 2.9. Modèle de coloriage Gouraud et placage de texture 40
Figure 2.10. Données utilisées à chaque étape du pipeline 3D et culling . . . 41
Figure 2.11. Les parties cachées dans un cube 42
Figure 2.12. Le tampen de profondeur selon l'axe Z 44
Figure 2.13. Présentation 3D et 2D avec l'algorithme Z-Buffer 44
Figure 2.14. L'ordre de visibilité des faces de la scène à modéliser 45
Figure 2.15. Test de visibilité d'une face 47
Figure 2.16. le modèle de réflexion diffuse 49
Figure 2.17. l'angle entre le rayon de lumière et la normal d'une surface . . . 49
Figure 2.18. Le modèle de réflexion spéculaire 50
Figure 3.1. L'architecture MVC 57
Figure 3.2. Diagramme de classe du pattern Observer 58
Figure 3.3. Diagramme de classe du pattern Strategy 59
Figure 3.4. Diagramme de classe du pattern Composite 60
Figure 3.5. Diagramme de classe du pattern AbstractFactory 60

FIGURE 3.6. La structure et l'évaluation d'un framework typique 63
FIGURE 3.7. Activités rencontrées lors du développement d'une application et d'un framework 66
FIGURE 4.1. L'architecture du notre framework dans un système 71
FIGURE 4.2. Diagramme de classe de l'architecture du Framework 75
FIGURE 4.3. Diagramme de classe des triplets 77
FIGURE 4.4. Diagramme de classe des matrices de transformations 78
FIGURE 4.5. Diagramme de classe des opérations trigonométrques 79
FIGURE 4.6. Diagrammes de classe pour les modules graphisme 2D et image 79
FIGURE 4.7. Diagramme de classe des interfaces I3DAPI et IRenderer 80
FIGURE 4.8. La structure de la liste chainée 82
FIGURE 4.9. Diagramme de classe du pattern Observer 83
FIGURE 4.10. Diagramme de classe de l'interface ICamera 84
FIGURE 4.11. Diagramme de classe de la structure d'arbre 85
FIGURE 4.12. Diagramme de classe de la structure d'une scène 3D 86
FIGURE 4.13. Diagramme de séquence de rendu d'une scène 3D 87
FIGURE 4.14. Diagramme de séquence des intéractions entre les le moteur 3D et la file d'attente 87
FIGURE 4.15. Diagramme de classe de gestionnaire d'API 3D 90
FIGURE 4.16. Diagramme de classe des outils de graphique 2D 91
FIGURE 4.17. Diagramme de classe de module Camèra 92
FIGURE 4.18. Diagramme de classe du programme de test 94
FIGURE 4.19. Modélisation filière d'un avion 96
FIGURE 4.20. Modélisation filière d'un plan 96
FIGURE 4.21. Modélisation polygonale des objets 3D simples 97
FIGURE 4.22. Rendu d'une scène(2 cubes et un plan)avec le modèle d'ombrage plat 98
FIGURE 4.23. Rendu d'une scène 3D (2 cubes et une sphère)en mode filière . 98
FIGURE 4.24. Résultat d'affichage d'un cube avec le modèle d'ombrage gouraud 99

FIGURE 4.25. Tests de placage de texture 100

Liste des Tables

Table 1.1. L'architecture des CPU graphiques pour les terminaux mobiles . 26
Table 1.2. Performances des CPU et les Chips 3D Texas Instruments 26
Table 2.1. Résumé des catégories de modélisations 3D 33
Table 2.2. Les caractéristiques de l'éspace d'objet et l'éspace d'image 43

Introduction

Introduction

Le logiciel envahit notre vie. On le trouve dans nos téléphones, nos voitures, nos domiciles,...etc, et ce ne sont pas les évolutions du matériel qui vont ralentir ce phénomène. C'est là qu'intervient J2ME (Java 2 Micro Edition)[DEL02], elle fournit un langage reconnu : Java, et des outils permettant de concevoir et réaliser des applications adaptables le plus facilement possible sur l'ensemble des ces équipement, qu'il soient mobiles ou embarqués.

Les dispositifs mobiles, par exemple: PDA, les téléphones cellulaires ont plus de possibilités d'apprécier un nombre de plus en plus important d'utilisateurs au cours de ces dernières années. Avec la croissance de nombre d'utilisateurs, le nombre d'applications qui n'appartiennent pas aux tâches classiques de ces dispositifs augmente aussi. Certains d'entre eux pourrait exigé la disponibilité des solutions interactives de graphiques dans la 2D ou la 3D.

La diffusion d'applications 3D se fait selon plusieurs axes. Tout d'abord, par des applications exécutables. Pour celles-ci, le programmeur peut utiliser des bibliothèques de fonctions 3D (API 3D) disponibles sous la forme de librairies. La seconde manière de diffuser un contenu 3D est d'utiliser un ensemble modeleur et moteur de rendu 3D. Ceux-ci doivent être capables d'échanger leurs données par un format de fichier commun.

Le processus de création des images se décompose en deux étapes : la modélisation et le rendu. La modélisation consiste à définir la scène que l'on souhaite visualiser en données interprétables. Ces données, telles que la position des surfaces et leur nature ou l'éclairement de la scène, sont utilisées lors de l'étape de rendu pour obtenir des images.

Les patterns de conception (*design patterns*)[SHE03],[BOI04] fournissent un ensemble de techniques facilitant l'analyse, la conception et la réalisation des systèmes informatiques et ils réduisent l'immobilité d'un système en permettant la création des modules réutilisables. Les frameworks (*cadre d'application*)[PRE00] sont des sous systèmes prêt à être utiliser avec des possibilités bien définies d'adaptation et

d'extension.

Objectifs

Dans ce travail, nous utilisons les patterns dans le développement d'un framework pour le graphisme 3D sous la plateforme J2ME, nous implémentons un moteur 3D pour la visualisation 3D dans les terminaux mobiles.

Cette étape exige un temps d'acquisition et d'élaboration assez long. Bien que ce ne soit pas le but de ce travail, mais nous devrons indiquer comment employer les patterns et les frameworks dans une méthodologie de conception de logiciel graphique avec la pris en charge des limitations des terminaux mobiles.

Pour tester le moteur 3D nous utilisons plusieurs émulateurs pour les terminaux mobiles (J2ME Wireless Toolkit 2.2 et Nokia Developers Suite for J2ME 3.0).

Nous examinons le temps de rendu pour des arrangements de qualité d'affichage.

Nous considérons l'ajout des objets 3D étape par étape à la scène et nous examinons le rendu résultant.

Plan de mémoire

Le chapitre un, présente la technologie J2ME, son architecture et leurs applications sur les terminaux mobiles. Nous présentons le profil MIDP et la configuration CLDC que nous avons choisi comme environnement de travail dans la réalisation pratique du moteur 3D. Nous décrivons ensuite les différentes technologies existantes dans le domaine de la 3D pour les systèmes embarqués et les terminaux mobiles.

Dans le chapitre deux, nous introduisons des généralités sur la 3D et nous décrivons les différentes techniques de rendu. Nous présentons le processus (pipeline) de la visualisation 3D et les différentes méthodes d'illumination dans une scène 3D. Ces dernières sont utilisées dans le développement du moteur 3D.

Le chapitre trois présente les concepts et les notions fondamentaux sur les patterns et les frameworks. Nous présentons les concepts fondamentaux sur les patterns et les frameworks, leurs types et quelques exemples de patterns de conception que nous avons utilisés dans la modélisation de notre framework.

Le quatrième chapitre est consacré à la conception et l'implémentation d'une application qui est constituée de deux partie : le développement d'un framework pour

la modélisation 3D et l'implémentation d'un moteur 3D pour tester notre framework. Ces deux parties constituent les résultats d'une étude théorique faite sur les patterns et les frameworks, ainsi que l' étude théorique et pratique sur les interfaces 3D et leurs outils (moteurs 3D).

Nous terminons notre travail par une conclusion générale.

CHAPITRE I

J2ME et la 3D dans les terminaux mobiles

CHAPITRE 1

J2ME ET LA 3D DANS LES TERMINAUX MOBILES

1.1 Introduction

Les téléphones portables ont été limités à des affichages simples, quelques lignes de texte avec un fond gris et quatre couleurs. L'évolution technologique fait qu'aujourd'hui des téléphones plus puissants, avec plus de mémoire et un meilleur affichage. On voit maintenant des appareils qui intègrent un appareil photo, d'autres un lecteur mp3 ou la radio, l'évolution de la téléphonie est loin de s'essouffler. Cependant, les téléphones évoluent mais sur des standards qui diffèrent selon les fabricants et les modèles.

Avec Java, nous pouvons cependant entrevoir une solution. L'entreprise Sun Microsystems nous propose une version allégée de J2SE, adaptée aux appareils de faible puissance appelée J2ME (Java 2 Micro Edition).[DEL02]

Nous commençons ce chapitre par un aperçu sur Java 2 Micro Edition, Ceci inclut les informations sur l'architecture de J2ME, ces configurations et leurs profils. Nous présentons la configuration CLDC (Connected Limited Device Configuration) et le profil MIDP(Mobile Information Device Profile) puisqu' ils sont utilisés dans le développement de notre application.

Dans la deuxième partie de ce chapitre nous présentons les différentes technologies de la 3D dans les terminaux mobiles et leurs domaines d'applications.

1.2 La plateforme J2ME (Java 2 Micro Edition)

1.2.1 Présentation de J2ME

L'entreprise Sun Microsystems a proposé plusieurs plateformes pour le développement d'applications sur des machines possédant des ressources réduites.

On peut citer quelque plateformes :

• JavaCard : pour le développement sur des cartes à puces

• EmbeddedJava : pour le développement sur des machines a système embarqué Java.

• PersonnalJava : pour le développement sur des machines possédant au moins 2Mo de mémoire.

En 1999, Sun Microsystems propose de mieux structurer ces différentes plateformes sous l'appellation J2ME (Java 2 Micro Edition). Seule le plateforme JavaCard n'est pas incluse dans J2ME et reste à part.

J2ME propose donc une architecture modulaire. Chaque configuration peut être utilisée avec un ensemble de packages optionnels qui permet d'utiliser des technologies particulières (Bluetooth, services web, lecteur de codes barres, ...etc.). Ces packages sont le plus souvent dépendant du matériel.

Il ne faut cependant pas oublier que les types de machines cibles de J2ME sont tellement différents (de téléphone mobile au set top box), qu'il est sûrement impossible de trouver un dénominateur commun. Ceci associé à l'explosion du marché des machines mobiles explique les nombreuses évolutions en cours de la plateforme.[DOU04]

La figure suivante représente l'architecture de la plateforme J2ME et les autres plateformes de Java.

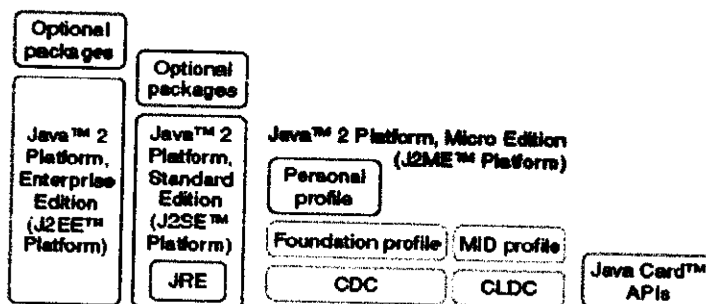


FIGURE 1.1. L'architecture de la plateforme J2ME

1.2.2 Les configurations et les profils

Les appareils mobiles sont de natures différentes, J2ME définit alors deux types de spécifications fonctionnant simultanément, les configurations et les profils.

Une configuration est une machine virtuelle et un ensemble minimal de classes de base et d'API .Elle spécifie un environnement d'exécution généralisé pour les terminaux embarqués et agit comme plateforme Java sur le terminal.

Un profil est une spécification des API Java définie par l'industrie, utilisée par les fabricants et les développeurs à destination de terminaux spécifiques.

1.2.2.1 Les configurations

Deux configurations sont essentiellement utilisées : la configuration CDC (Connected Device Configuration) et la configuration CLDC (Connected Limited Device Configuration) :

- La configuration CDC est plus adaptée aux terminaux relativement puissants comme les PDA.
- La configuration CLDC est par contre dédiée aux appareils avec de faibles capacités de 160ko à 512ko de mémoire comme les téléphones portables.

1.2.2.2 Les profils

Sun MicroSystems propose deux profils de référence J2ME : le profil Foundation et MIDP(Mobile Information Device Profile).

- Le profil Foundation est destiné à la configuration CDC.
- Le profil MIDP est destiné à la configuration CLDC.

La figure 1.2 résume les différentes utilisations possibles des profils :

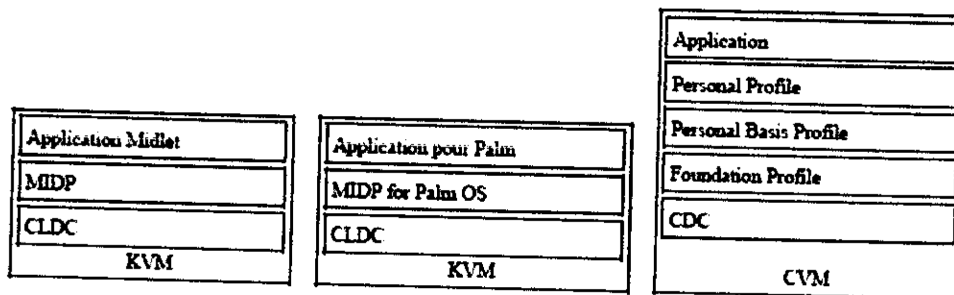


FIGURE 1.2. Les utilisations possibles des profils

D'autres exemples de profils proposés par JCP (Java community Process), dans [DEL02] : PDAP JSR 75, Foundation Profile JSR 46, Personal Profile JSR 62, RMI JSR 66.

1.2.3 La machine virtuelle

En fonction de la cible, la machine virtuelle pourra être allégée afin de consommer plus ou moins de ressources , les deux type de machines virtuelles utilisées dans J2ME sont KVM et CVM [DEL02] :

1.2.3.1 CVM (Compact Virtual Machine):

C'est une machine virtuelle conçue pour les terminaux ayant besoins de l'ensemble de fonctionnalités de JVM (Java Virtual Machine) mais avec des capacités plus réduites. Elle est conçue pour répondre aux besoins du marché émergent des terminaux embarqués de nouvelle génération. Les terminaux utilisant CVM sont généralement des terminaux compacts et connectés, orientés consommateur.

1.2.3.2 KVM (Kilo Virtual Machine):

La machine virtuelle KVM est une nouvelle machine virtuelle de Java fortement optimisée pour les ressources et les contraints du dispositif. Convenablement, la machine virtuelle K est très petite dans la taille (de 40 – 80 Ko), et tout le soin a été pris pour la rendre appropriée aux terminaux tels que les Palm, les téléphones mobile, et les PDA. Comme n'importe quelle autre machine virtuelle de Java, elle permet de télécharger et exécuter des applications.

La KVM peut fonctionner sur n'importe quel système qui a un processeur 16-bit/32-bit et environ 160-512Ko de mémoire totale. La KVM n'est pas implémentée par Sun Microsystems et à déjà être porté sur plusieurs dispositifs par des constructeurs agrée par Sun [DRE02].

1.2.4 La configuration CLDC

L'API du CLDC se compose de quatre packages :

- java.io : classes pour la gestion des entrées/sorties par flux,
- java.lang : classes de base du langage java,

• java.util : classes utilitaires notamment pour gérer les collections, la date et l'heure, etc.

• javax.microedition.io : classes pour gérer des connections génériques.

Ces packages ont des fonctionnalités semblables à ceux proposés par J2SE avec quelques restrictions, notamment il n'y a pas de gestion des nombres flottants dans la configuration CLDC 1.0.

Remarque 1.1. *La version courant de CLDC 1.1 comprend des nouvelles classes pour définir les nombres réels de 32 bits et 64 bits :*

Nom	Rôle
java.lang.Double.java	Classe qui encapsule une valeur de type Double
java.lang.Float.java	Classe qui encapsule une valeur de type Float

1.2.5 MIDP (Mobile Information Device Profile)

C'est le premier profil qui a été développé dont l'objectif principal est le développement d'application sur des machines aux ressources faibles et à l'interface limitées tel qu'un téléphone cellulaire. Ce profil peut aussi être utilisé pour développer des applications sur des PDA de type Palm [DOU04].

L'API du MIDP se compose des API du CDLC et de trois packages :

javax.microedition.midlet : cycle de vie de l'application

javax.microedition.lcdui : interface homme machine

javax.microedition.rms : persistance des données

Il existe deux versions du MIDP :

1.0 : la dernière révision est la 1.0.3 dont les spécifications sont issues de la JSR 37

2.0 : c'est la version la plus récente dont les spécifications sont issues de la JSR 118

1.2.5.1 La Midlet et son cycle de vie

Les applications créées avec MIDP sont des Midlets : ce sont des classes qui héritent de la classe abstraite javax.microedition.midlet.Midlet. Cette classe permet

le dialogue entre le système et l'application. Elle possède trois méthodes qui permettent de gérer le cycle de vie de l'application en fonction des trois états possibles (active, suspendue ou détruite) :

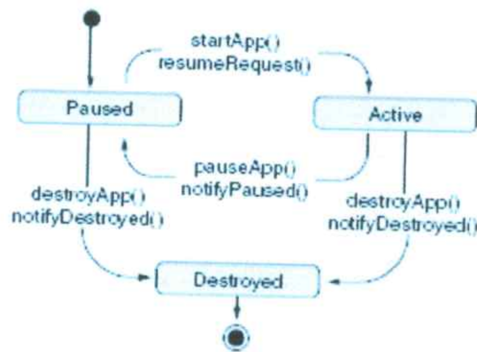


FIGURE 1.3. Cycle de vie d'une Midlet

startApp() : cette méthode est appelée à chaque démarrage ou redémarrage de l'application

pauseApp() : cette méthode est appelée lors de la mise en pause de l'application

destroyApp() : cette méthode est appelée lors de la destruction de l'application.

Ces trois méthodes doivent obligatoirement être redéfinies.

Voici un exemple de base pour la création d'une midlet:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Test extends MIDlet {
public Test() {
}
public void startApp() {
}
public void pauseApp() {
}
public void destroyApp(boolean unconditional) {}
}
```

1.2.5.2 L'interface utilisateurs

MIDP est conçu pour tourner sur de nombreux types de terminaux : téléphones, Palm Pilot, etc. Or la plupart de ces terminaux sans fil sont utilisés dans la main, disposent d'un petit écran et tous ne possèdent pas de système de pointage comme un stylo. Tout en respectant ces contraintes, les applications MIDP doivent intégrer toujours les mêmes fonctionnalités quelque soit le terminal. La solution a été de décomposer l'interface utilisateur en deux couches : l'API de haut niveau et celle de bas niveau. La première favorise la portabilité, la second l'exploitation de toutes les fonctionnalités du terminal. Le concepteur doit donc faire un compromis entre portabilité et bénéfice des particularités du terminal.

L'API de bas niveau donne accès direct à l'écran du terminal et aux événements associés aux touches et système de pointage. Aucun composant d'interface utilisateur n'est disponible : vous devez explicitement dessiner chaque composant, y compris les commandes.

L'API de haut niveau fournit quant à elle des composants d'interface utilisateur simples. Mais aucun accès direct à l'écran ou aux événements de saisie n'est permis. C'est l'implémentation MIDP qui décide de la manière de représenter les composants et du mécanisme de gestion des saisies de l'utilisateur.

Il est possible d'utiliser l'API de haut niveau et l'API de bas niveau dans un même Midlet mais pas simultanément. Par exemple, les jeux qui utilisent l'API de bas niveau pour contrôler l'écran peuvent aussi utiliser l'API de haut niveau pour afficher les meilleurs scores. L'API de bas niveau peut aussi être utilisée pour tracer des graphes.

La figure suivante représente les interfaces d'utilisateurs de profil MIDP:

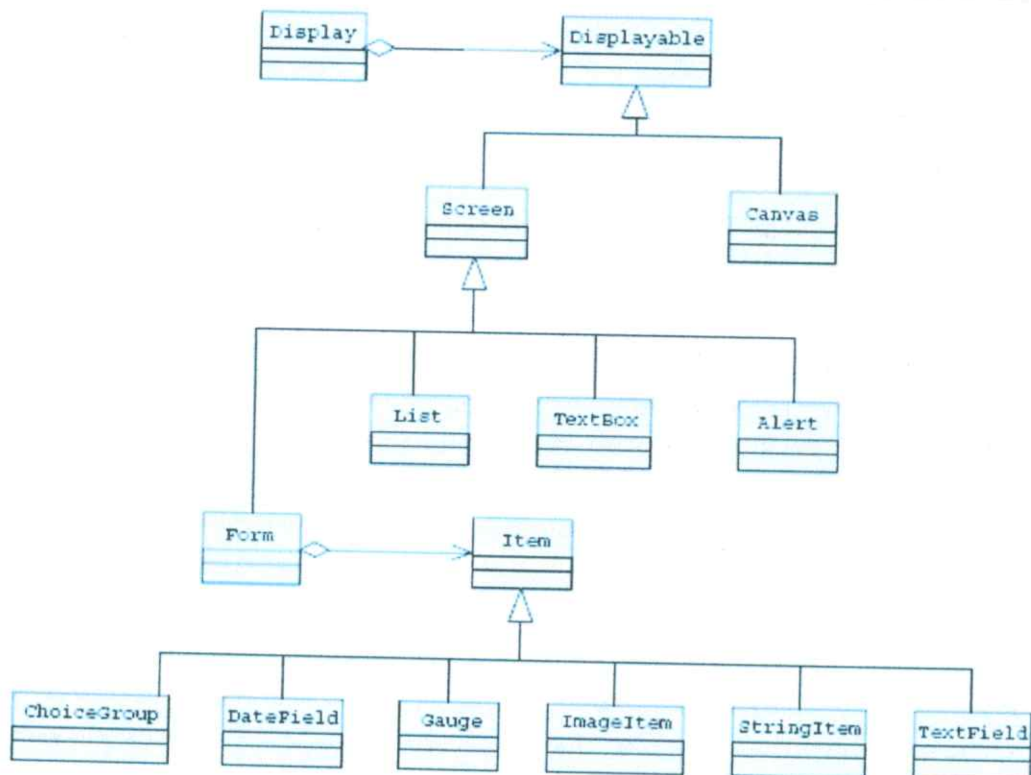


FIGURE 1.4. Les interfaces d'utilisateurs de MIDP

1.2.5.2.1 La classe Display

Pour pouvoir utiliser les éléments graphiques, il faut obtenir un objet qui encapsule l'écran. C'est un objet de la classe Display. Cette classe possède des méthodes pour afficher les éléments graphiques. La méthode statique getDisplay() renvoie une instance de la classe Display qui encapsule l'écran associé à la Midlet fournie en paramètre de la méthode.

Voici un exemple simple d'utilisation de la classe Display:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Hello extends MIDlet {
private Display display;
public Hello() {
display = Display.getDisplay(this);
```

```
public void startApp() {
}
public void pauseApp() {
}
public void destroyApp(boolean unconditional){
}
}
```

Les éléments de l'interface graphique appartiennent à une hiérarchie d'objets : tous les éléments qui peuvent être affichés héritent de la classe abstraite Displayable.

La classe Screen est la classe mère des éléments graphiques de haut niveau.

La classe Canvas est la classe mère des éléments graphiques de bas niveau.

Il n'est pas possible d'ajouter directement un élément graphique dans un Display sans qu'il soit inclus dans un objet héritant de Displayable.

Un seul objet de type Displayable peut être affiché à la fois. La classe Display possède la méthode getCurrent() pour connaître l'objet courant affiché et la méthode setCurrent() pour afficher l'objet fourni en paramètre.

La classe TextBox : Ce composant permet de saisir du texte.

La classe List : Ce composant permet la sélection d'un ou plusieurs éléments dans une liste d'éléments.

La classe Alert : Cette classe permet d'afficher une boite de dialogue pendant un temps déterminé.

La classe Form : La classe Form permet d'insérer dans l'élément graphique qu'elle représente d'autres éléments graphiques : cette classe sert de conteneurs. Les éléments insérés sont des objets qui héritent de la classe abstraite Item.

La classe Item : La classe javax.microedition.lcdui.Item est la classe mère de tous les composants graphiques qui peuvent être insérés dans un objet de type Form. Cette classe définit seulement deux méthodes, getLabel() et setLabel() qui sont le getter et le setter pour la propriété label.

1.2.5.2.2 La gestion des événements

Les interactions entre l'utilisateur et l'application se réalisent par le traitement

d'événements particuliers pour chaque action.

MIDP définit l'interface de type Listener pour la gestion des événements :

Interface	Rôle
CommandListener	Listener pour une activation d'une commande
ItemStateListener	Listener pour un changement d'état d'un composant (modification du texte d'une zone de texte, ...)

1.2.5.2.3 Le Stockage et la gestion des données(RMS)

Avec MIDP, le mécanisme pour la persistance des données est appelé RMS (Record Management System). Il permet le stockage de données et leur accès ultérieur.

RMS propose un accès standardisé au système de stockage de la machine dans lequel s'exécute le programme. Il n'impose pas aux constructeurs la façon dont les données doivent être stockées physiquement.

Du fait de la simplicité des mécanismes utilisés, RMS ne définit qu'une seule classe nommée RecordStore. Cette classe ainsi que les interfaces et les exceptions qui composent RMS sont regroupées dans le package javax.microedition.rms.

Les données sont stockées dans un ensemble d'enregistrements (records). Un enregistrement est un tableau d'octets. Chaque enregistrement possède un identifiant unique nommé recordId qui permet de retrouver un enregistrement particulier.

A chaque fois qu'un ensemble de données est modifié (ajout, modification ou suppression d'un enregistrement), son numéro de version est incrémenté. Un ensemble de données est associé à un unique ensemble composé d'une ou plusieurs Midlets (Midlet Suite) et possède un nom composé de 32 caractères maximum.

La classe RecordStore

L'accès aux données se fait obligatoirement en utilisant un objet de type RecordStore. Pour pouvoir utiliser un ensemble d'enregistrements, il faut utiliser la méthode statique openRecordStore() en fournissant le nom de l'ensemble et un booléen qui précise si l'ensemble doit être créé au cas ou celui ci n'existe pas. Elle renvoie un objet RecordStore qui encapsule l'ensemble d'enregistrements.La méthode closeRecordStore() permet de fermer un ensemble précédemment ouvert.

1.3 La 3D dans les terminaux mobiles

Alors que la vidéo fait ses premiers pas sur les mobiles (palmtop, téléphones mobiles, PDA, Pocket PC, etc), la 3D pointe apparaisse et devrait rapidement devenir indispensable.[WWW2]

Actuellement, on assiste à une multitude d'annonces dans ce domaine : il faut dire que le marché est particulièrement prometteur.

Les activités professionnelles qui profiteront directement de la 3D ne seront certainement pas nombreuses dans un premier temps.

Par contre, le secteur du grand public devrait être particulièrement influencé par la "vague 3D". Le jeu vidéo vient en tête avec déjà de nombreux jeux 3D sur Pocket PC 2002. La plateforme mobile de Microsoft devrait bientôt devenir une véritable console de jeux portable([WWW3], [WWW4]).

1.3.1 Vue sur l'ensemble du marché

D'autres marques de téléphones mobiles qui utilisent la 3D

- Consoles des jeux portatives
- PDAs
- Les mobiles

1.3.1.1 Le matériel

L'architecture des Processeurs (CPU) :

Il y a trois révisions principales de l'architecture ARM qui sont utilisées dans les téléphones mobiles.

Aucun support de calcul de la virgule flottante dans le matériel.[PAT05]

Architecture ARM	Noyau ARM
ARMv4	StrongARM, et ARM7
ARMv5TEJ	ARM9, ARM10
ARMv6	ARM11

TABLE 1.1. L'architecture des CPU graphiques pour les terminaux mobiles

1.3.2 Types de l'implémentation 3D

Rendu logiciel

- Les instructions visées par la 3D peuvent fournir quelque amélioration.

Rendu matériel

- dans un chips DSPs qui accélèrent le calcul des triangles.
- dans un chips ou des processeurs graphiques externes.
- la mémoire Frame-Buffer peut être disponible dans le chips.

Performances des Processeurs et les Chips 3D:

Le tableau suivant illustre les performances des processeurs et des Chips 3D:

TABLE 1.2. Performances des CPU et les Chips 3D Texas Instruments

Chip	Noyau ARM	Fréquence (MHz)	Matériel 3D	KTri/s	MTex/s
OMAP1510 jusqu'a OMAP1710	ARM9	plus de 220	non	logiciel <50	logiciel <1
OMAP2420	ARM11	330	complet	2000	7

1.3.3 Les Logiciels dans les mobiles (Système d'exploitation, langages et APIs)

Les systèmes d'exploitations :

Il y a plusieurs marques de système d'éxploitation pour les termianux mobiles et les PDA :

- BREW (Binary Runtime Environment for Wireless) Symbian OS
- Windows Mobile
- Linux for Mobile
- SavaJE OS
- Beaucoup de systèmes d'exploitation développés par les fabricants des téléphones mobiles.

Java et C++ :

Il y a deux plateformes pour le développement des applications pour les terminaux mobiles :

- BREW

- J2ME (pour tous les systèmes d'exploitations)

Les APIs :

- Pour java, on a besoin des APIs 3D

- Pour C++, on a besoin des APIs pour utiliser le matériel 3D

OpenGL ES :

OpenGL ES (OpenGL for Embedded Systems) est un ensemble d'API de bas niveau et léger pour les systèmes embarqués qui utilise des profils bien définis de sous-ensemble d'OpenGL. Il fournit une interface de programmation d'applications de bas niveau (API) entre les applications logiciels et les moteurs de rendu matériel ou logiciel.[PAT05]

JSR-184 (Mobile 3D Graphics for J2ME) :

Cette proposition spécifié une bibliothèque léger et interactive qui s'ajout a J2ME et MIDP comme un package optionnel.

L'API sera assez flexible pour les applications, y compris les jeux, les messages animés, les écrans de veilles,...etc. Il ne sera limité à aucun type particulier d'application. Il sera également assez simple d'utiliser pour rendre le développement rapide de applications 3D faisable.[PAT05]

L'API est visé aux machines de classe CLDC qui ont typiquement la capacité de traitement et la mémoire très petites, et à aucun soutien de matériel des graphiques 3D ou de calcul de virgule flottante. Elle utilise :

- Haut et base niveau d'affichage
- Format de fichier bien spécifique

Voici quelque implémentation de la JSR-184:

- La version 4 de Hi Corp's Mascot Capsule Engine Micro 3D Edition
- Superscape's Swerve Client
- Hybrid Graphics's Hybrid Mobile Framework

1.4 Conclusion

Nous avons présenté dans ce chapitre la plateforme J2ME, c'est une architecture technique dont le but est de fournir un support de développement aux applications embarqué.

Ce qu'il faut retenir de l'architecture J2ME c'est qu'elle se compose d'une configuration contenant la machine virtuelle et les fonctionnalités minimales nécessaire à accéder au matériel et au réseau ,d'un profil qui contient le reste des fonctionnalités exploitables sur une famille d'appareils aux caractéristiques semblables. et des bibliothèques optionnelles que les constructeurs sont libres d'implémenter.

Les applications écrites à partir du profil MIDP se nomment les Midlets. Ces applications doivent tenir compte de la configuration limitée des dispositifs mobiles. Ce ne sont en effet pas des stations de calcul et ont des éléments de pointage très simples ou en sont carrément dépourvus. Ainsi, une Midlet ne doit pas, si possible, demander plus d'une interaction à la fois de la part de l'utilisateur.

Malgré que le plus grand nombre des Midlets actuelles dans le cadre des jeux ou des petits utilitaires, les activités professionnelles qui profitent directement de la 3D dans ce profil ne sont pas nombreuses, pour cela il faut créer des libraires "haut niveau" pour implémenter au niveau logiciel le rendu ou de s'affranchir du matériel. Ces librairies offrent aux programmeurs des services pour afficher des primitives simples (lignes, triangles,...etc) et gérer tout une partie de l'affichage .

Pour développer ce genre d'application nous avons besoin des connaissances et des notions dans le domaine de la modélisation 3D et l'infographie, que nous présontons dans le chapitre suivant.

CHAPITRE II

Le graphisme 3D

CHAPITRE 2

Le graphisme 3D

2.1 Introduction

Les progrès effectués en infographie ont révolutionné le monde de l'image et ils ont amélioré diverses techniques graphiques telles le rendu d'images de synthèse, la création de graphismes 3D que l'on retrouve dans les films d'animations, les publicités et les jeux vidéo.

Derrière ces graphismes se trouvent des techniques et des procédés que l'on n'imagine pas en regardant ces images. Le processus de création d'une image 3D se fait en plusieurs étapes (transformations, test de visibilté, calcul de lumières, ...etc) ayant chacune un rôle important.

Nous présentons dans ce chapitre les notions fondamentaux sur la modélisation 3D (la modélisation géométrique et radiométrique , le processus de visualisation , ...etc) et les différents techniques utilisées pour le rendu.

2.2 Généralités sur la modélisation 3D

2.2.1 Définition

On peut dire qu'il s'agit de la représentation du réel. Précisément, c'est la première étape à effectuer pour représenter un objet réaliste. On pourrait comparer la modélisation au travail du sculpteur ou du potier :

Il a besoin d'un matériau de base, et à partir de celui-ci, il travaille une forme en lui donnant l'aspect désiré. Pour l'infographiste, le matériau de base n'existe pas au départ (ou alors, c'est l'imagination); c'est à lui de le créer en représentant en perspective des formes virtuelles. Celles-ci seront travaillées afin d'obtenir l'objet désiré.

Plusieurs étapes s'effectuent entre le départ du traitement et l'obtention d'un résultat. Nous verrons ici les différentes modélisations qu'utilisent les infographistes 3D.[WWW5]

2.2.2 Les différents types de modélisations

Modélisation filière

C'est la première représentation d'objets tridimensionnels. Compte tenu de la puissance des ordinateurs du début des années 70, et de leurs capacités mémoires, le modèle en fil de fer était tout à fait représentatif du niveau de technologie de l'époque.

Dans ce type de modélisation, seules les coordonnées des points et les segments reliant ces points sont stockées en mémoire, donc peu d'espace mémoire encombré et peu de puissance de calcul (juste afficher des points et des traits) pour cette méthode. Au delà de ces deux minces avantages, on note pas mal d'inconvénients:

- la confusion, entre le vide et le non vide, ou encore le sens physique de l'objet modélisé (voir figure 2.1);

- la difficulté de résoudre le problème de suppression des parties cachées;

- l'impossibilité d'effectuer certains calculs comme celui de masse.

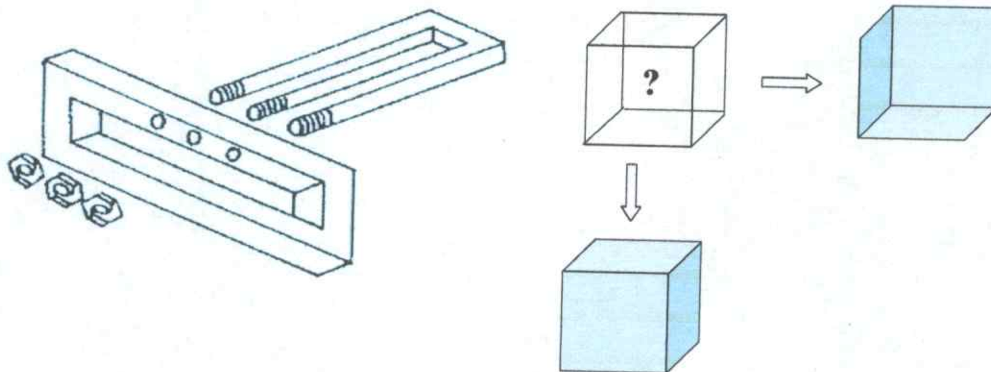


FIGURE 2.1. Ambiguïté du modèle en fil de fer

On notera quand même qu'on l'utilise pour les mises au point de certaines séquences d'animation.

Modélisation polygonale

C'est la modélisation utilisée par les logiciels actuels. Il est vrai que celle-ci offre un champ d'action immense. A partir d'une "simple" figure plane, on peut modéliser pratiquement n'importe quelle forme. (voir figure 2.2)

Tout d'abord, un petit rappel sur ce qu'est un polygone: c'est une figure plane définie par une liste de points et délimitée par des segments de droite reliant les points entre eux. Un polygone est dit convexe si ses angles internes sont inférieurs à 180°, et concave si au moins un de ses angles internes est supérieurs à 180°.

La modélisation polygonale offre une souplesse permettant de former presque n'importe quelle forme. Certes, les formes "carrées" des cubes ou autres rectangles sont faciles à modéliser dans ce cas comme dans la modélisation fil de fer, mais ici, la modélisation de formes sphériques est plus que possible. En multipliant les nombres de polygones, on affine le rendu final de l'objet. Ce maillage massif de polygones permettant d'afficher des formes sphériques a un prix: la consommation mémoire proportionnellement accrue. Avec la puissance des machines d'aujourd'hui, ceci ne représente pas vraiment un inconvénient. Ce qui explique aussi la grande pratique de cette représentation polygonale est qu'elle offre une infinité de fonctions de modélisations. Un bon modeleur polygonal doit posséder les fonctions connues telles que:

- la génération de surface par symétrie circulaire à partir du profil 2D de l'objet (bouteille, sphère, pion d'échec, ...etc), on parle aussi de génération de surfaces de révolution.

- les fonctions rotation, dimensionnement de la taille des objets

- des fonctions permettant d'agir sur un point, un segment, une face, ou l'objet entièrement,...etc.

- des fonctions fort sympathiques qui inversent, dupliquent,...etc.

Il est clair que cette modélisation est la plus malléable, cependant on peut noter un gaspillage de mémoire au niveau de la représentation de formes sphériques nécessitant beaucoup de polygones. En fait, aucun modèle n'est parfait, il y a toujours un cas qui sera plus facile à traiter avec une autre méthode, nous verrons ceci dans le modèle volumique.

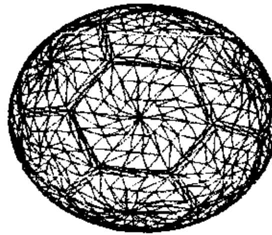


FIGURE 2.2. Modèle avec des facettes

Modélisation volumique (solid modeling)

Cette modélisation est très différente des deux autres, au niveau même de sa conception. Les objets 3D ne sont plus considérés graphiquement mais à partir de formules régissant leurs structures (on parle de primitives 3D). Ainsi, la représentation d'une sphère ne sera définie que part son centre et son rayon, un cube par les équations des plans que constituent ses faces, un cylindre par son diamètre et sa longueur ; par exemple, un cône ne sera qu'un cylindre possédant deux diamètres différents, etc..

L'avantage de cette modélisation est sa faible consommation de mémoire qui permet un stockage des données moins lourd et un rendu plus rapide à traiter. Elle a aussi l'avantage de produire des objets parfaitement arrondis, ce qui est théoriquement impossible avec une modélisation polygonale. Cependant, la modélisation volumique manque de souplesse car on ne peut pas travailler point par point. De plus il est certain que pour représenter des formes gauches (irrégulières et complexes) dont la recherche d'une formule serait très difficile.

Modélisation surfacique algébrique

Nous venons de voir que la modélisation volumique n'était pas très capable de représenter une surface gauche. Il s'agit justement du point fort de la modélisation surfacique. En fait, ce modèle travaille par une représentation algébrique, en traitant des équations de courbes. On distingue trois types de surfaces courbes en imagerie 3D : les surfaces de révolution, les surfaces paramétriques définies par des fonctions de deux valeurs et les surfaces à formes libres, celles qui dévoilent la puissance de ce modèle surfacique.

Pour les surfaces courbes à formes libres, les équations sont des polynômes de degré supérieur ou égal à 3. Ce qu'il faut retenir c'est que les représentations sont très souples, on peut modifier une courbure au point près en sachant que des points de contrôles sont répartis le long de la courbe. Ce type de modélisation convient particulièrement à ceux qui conçoivent des pièces mécaniques et qui ont des machines puissantes (les fonctions des courbes peuvent être gourmandes en ressource système).

FIGURE 2.3. Modèle surfacique

En résumé, pour un usage plus général, il vaut mieux préférer la modélisation polygonale. Ainsi, les différents modèles présentent des avantages et des inconvénients. Ces modèles peuvent aboutir à des objets fort bien rendus, d'ou la possibilité de pouvoir utiliser différents modèles dans un même modeleur. On peut résumer les catégories de modélisations dans le tableau suivant :

TABLE 2.1. Résumé des catégories de modélisations 3D

MODELISATION	ENTITES DE BASE
Fil de fer	Points, arêtes
Polygonale	Facette
Volumique	Primitives, solides
Surfacique algébrique	Courbes

2.2.3 Les bases de la 3D temps réel

Avant de présenter les différentes manières de créer et d'afficher une scène 3D, nous rappelons les fondements de la 3D temps-réel. Ils sont rassemblés dans la chaîne de production d'une image ou pipeline graphique (ou encore pipeline 3D), ainsi que les différentes étapes de celui-ci [TOP01],[PER04].

Le pipeline 3D est l'ensemble des opérations nécessaires pour afficher une scène constituée d'objets 3D regardés depuis une position et avec une orientation données. A chaque fois que l'état de la scène change (c'est-à-dire à chaque mouvement de la caméra, à chaque déplacement d'un objet, etc.), elle doit être redessinée. Pour cela, la description en mémoire vive des objets de la scène doit être traduite en points 2D à l'écran. Ce processus est le travail du pipeline 3D (ou moteur 3D). Quel que soit le moteur 3D utilisé, plugin ou API, un pipeline 3D effectue les différents calculs nécessaires à l'affichage d'une scène 3D sur un écran.

Les éléments d'entrée de ce pipeline sont des triangles plus pratiques que les quadrilatères ou autres polygones pour les calculs car ils ne peuvent être ni difformés (dont les sommets ne sont pas coplanaires) ni concaves donc plus faciles à tracer. Cependant, afin de nous autoriser à créer des polygones complexes, la plupart des moteurs 3D effectuent une triangulation des différentes faces avant de les envoyer au pipeline graphique. Ainsi, ils nous permettent de réfléchir en terme de polygones et non uniquement en terme de triangles mais évitent néanmoins le problème de remplissage des faces concaves.

Le schéma suivant illustre les différentes étapes de calculs menant à l'affichage d'un triangle.

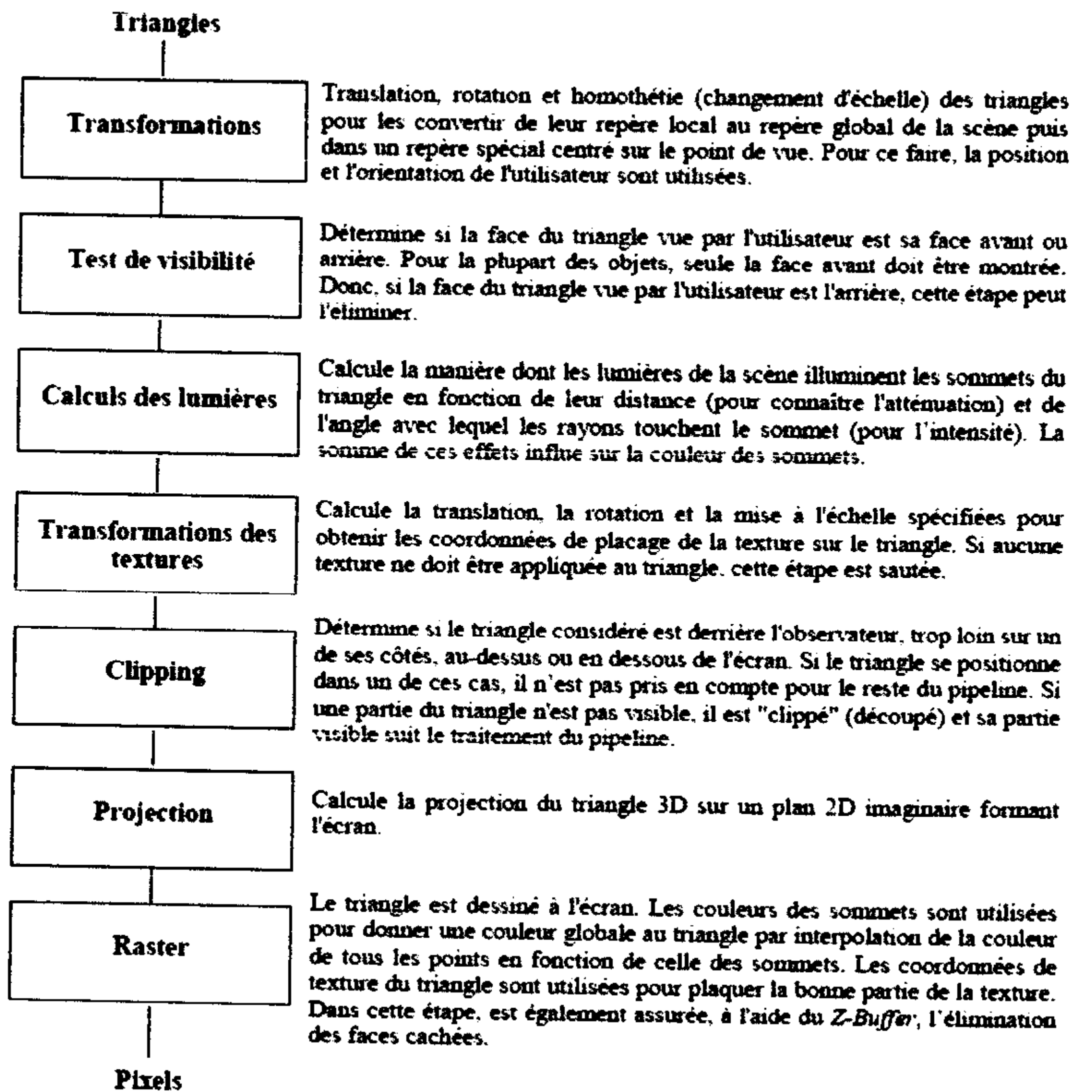


FIGURE 2.4. Les différentes étapes du pipeline 3D.

Les pipelines graphiques calculent la scène le plus rapidement possible sans prendre en compte le temps écoulé entre l'affichage de deux trames successives.

Avec une scène de petite taille, ou optimisée, le pipeline calcule le rendu efficacement et effectue un rafraîchissement rapide. On obtient par conséquent une animation fluide. Au contraire, avec une scène comportant de nombreux objets ou pour une scène non optimisée, le pipeline risque de bloquer dans une, voire plusieurs étapes. Il existe cependant des pipelines graphiques garantissant 25 images par seconde quelque soit la scène ; ceci en arrêtant leurs traitements au-delà du 25ème de

seconde si cela est nécessaire. D'autres techniques, moins radicales, existent également.

Nous expliquons ici succinctement les calculs effectués à chacune des étapes du pipeline graphique afin que le lecteur non habitué aux techniques sous-jacentes à la 3D puisse comprendre le coût du calcul d'une image.

Avant tout, il convient de parler du mode de représentation utilisé pour placer un point dans l'espace. Les calculs nécessaires pour la transformation et la projection d'un point dépendent de cette représentation.

Le système de coordonnées que l'on utilise pour la 3D est cartésien. La façon la plus naturelle de le représenté est d'utiliser ses deux mains (d'où les noms « main droite » et « main gauche ») comme cela est schématisée dans la figure 2.5. Pour chacune des mains, il suffit de faire pointer le pouce (donnant l'axe X) vers la droite, l'index (l'axe Y) vers le haut. Le majeur, perpendiculaire aux deux doigts précédents (et à la paume de la main) nous donne automatiquement la direction positive de l'axe Z.

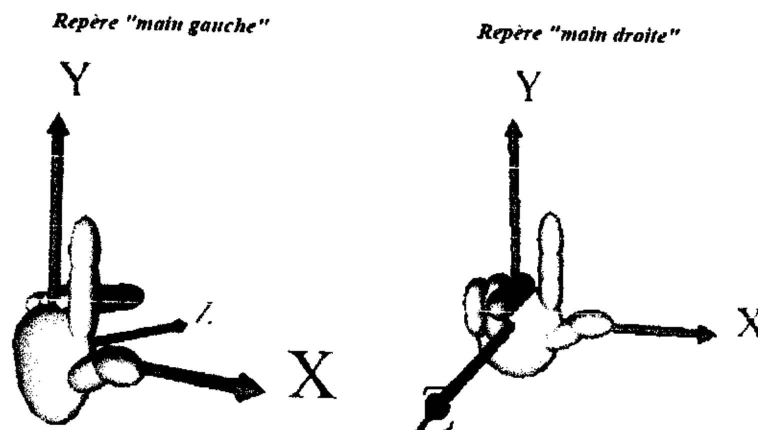


FIGURE 2.5. Les deux repères utilisés en 3D.

Toute primitive graphique complexe est d'abord transformée en un ensemble de triangles. L'apparence obtenue par approximation en un nombre plus ou moins de triangles permet d'avoir soit un rendu plus rapide soit un rendu de meilleure qualité. Chacun des triangles subit alors les différentes étapes du pipeline graphique dont

nous donnons pour les plus importantes le détail des calculs :

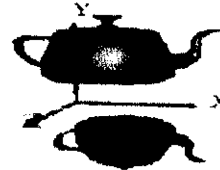
- **Transformations** : les opérations matricielles donnant le transformé d'un point par une ou plusieurs des matrices suivantes :

- Translation selon un vecteur (t_x, t_y, t_z) :

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

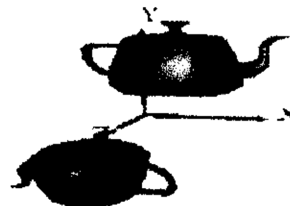
- Rotations par rapport aux axes X, Y et Z données ici pour un repère « main droite » :

$$R_X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$R_Y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_Z = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- Homothétie d'un facteur h_x, h_y et h_z sur chacune des coordonnées :

$$H = \begin{bmatrix} h_x & 0 & 0 & 0 \\ 0 & h_y & 0 & 0 \\ 0 & 0 & h_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

FIGURE 2.6. Les transformations 3D

- **Test de visibilité** : le calcul de visibilité d'un triangle se fait grâce à son vecteur normal donnant sa face avant. Si l'angle formé entre le vecteur normal et le vecteur de vision (allant de la face à l'oeil) est aigu alors la face est visible

sinon elle est invisible. Ce calcul se fait en utilisant le produit scalaire. Si celui-ci est négatif alors cela signifie que le cosinus de l'angle entre les deux vecteurs est négatif et donc que l'angle est obtus. Si nécessaire, le vecteur normal est calculé en faisant le produit vectoriel entre le vecteur formé par les deux premiers points et celui formé par les deuxième et troisième points du polygone. Par convention, tous les moteurs 3D attendent des polygones dont les sommets sont donnés dans l'ordre trigonométrique comme dans la figure suivante. Par conséquent le vecteur normal résultant du produit vectoriel est bien extérieur à la face.

- **Détermination du vecteur normal** :

$$\vec{N} = \vec{AB} \wedge \vec{BC} = \|\vec{AB}\|\|\vec{BC}\| \sin\left\langle \vec{AB}, \vec{BC} \right\rangle$$

- **Détermination si face visible** :

$$\vec{N}.\vec{V} = \|\vec{N}\|\|\vec{V}\| \cos\left\langle \vec{N}, \vec{V} \right\rangle = x_N.x_V + y_N.y_V + z_N.z_V$$

Si $\vec{N}.\vec{V} < 0$ *alors* $\left\langle \vec{N}, \vec{V} \right\rangle$ *obtus*

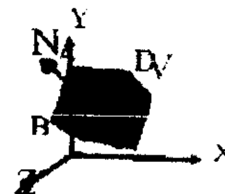


FIGURE 2.7. Déterminations du vecteur normal à une face et test de visibilité

- **Calculs des lumières** : l'utilisation de lois physiques simplifiées (pour des raisons de performance) comme la loi de *Lambert* donnant l'intensité de la lumière réfléchie en fonction du matériel d'un objet.

La somme des intensités des différentes lumières frappant la face donne un coefficient global d'éclaircissement de la face. Cette valeur est alors utilisée lors du coloriage. La loi de Descartes donnant les angles de réflexion et de réfraction en fonction du matériel de l'objet n'est pas utilisée comme dans la technique du lancé de rayon car les lumières réfléchies ne sont pas prises en compte.

- **Transformations des textures** : Cette étape permet de transformer les textures avant qu'elles ne soient appliquées au triangle (dans l'étape de Rasterization que nous définissons plus tard). Ce sont des transformations 2D sur les images qui sont un cas simplifié des transformations 3D précédentes.

- **Clipping** : c'est l'étape d'élimination des triangles ne faisant pas partie du volume de vue et découpage de ceux en partie visibles selon leurs intersections avec le volume de vue.

- **Projection** : la projection d'une scène 3D sur l'écran 2D comprend en fait deux étapes différentes : la conversion dans le repère de l'observateur et la projection sur l'écran (plan de projection). Dans la figure 2.8 nous donnons les opérations pour ces deux étapes et un schéma explicatif. Dans le schéma ainsi que dans les calculs nous utilisons un repère « main droite » dont l'origine est O. La position de l'observateur est C (pour caméra) et il regarde le long de l'axe Z négatif (vers O). Son repère local est donc « main gauche » comme nous pouvons également le constater sur le schéma. Le plan de projection est perpendiculaire à OC à une distance D de l'observateur.

- Changement de référentiel vers le repère de l'observateur :
  1. translation de l'origine O en O' donne un le repère (X_1, Y_1, Z_1).
  2. rotation de (X_1, Y_1, Z_1) autour de Z_1 pour amener X_1 sur X_0 donne le repère (X_2, Y_2, Z_2).
  3. rotation de (X_2, Y_2, Z_2) autour de X_2 pour amener Y_2 sur Y_0.
  4. conversion en un repère « main gauche ».

$$\begin{cases} x' = -x.\sin\theta + y.\cos\theta \\ y' = -x.\cos\theta.\sin\varphi - y.\sin\theta.\sin\varphi + z.\cos\theta \\ z' = -x.\cos\theta.\cos\varphi - y.\sin\theta.\cos\varphi - z.\sin\theta + R \end{cases}$$

- Projection sur l'écran :
  - perspective :

$$X = D.\frac{x'}{z'} \ et \ Y = D.\frac{y'}{z'}$$

  - parallèle :

$$X = x' \ et \ Y = y'$$

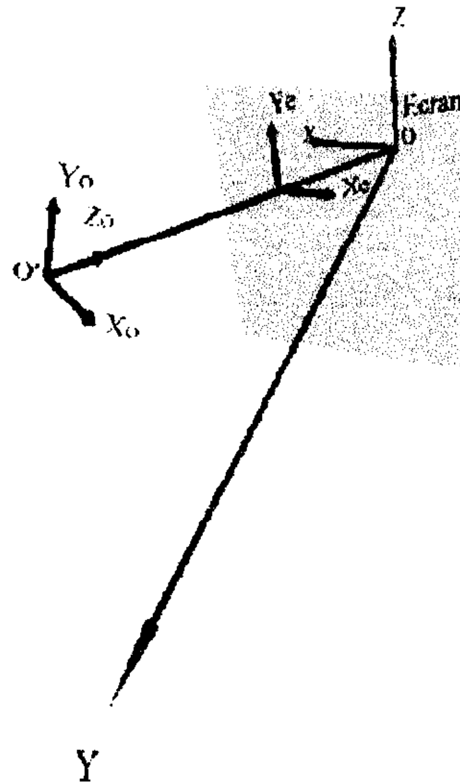


FIGURE 2.8. Calcul de la projection.

- **Rasterization** : c'est l'étape qui transforme les formes géométriques 3D en des pixels sur l'écran(voir figure 2.9).

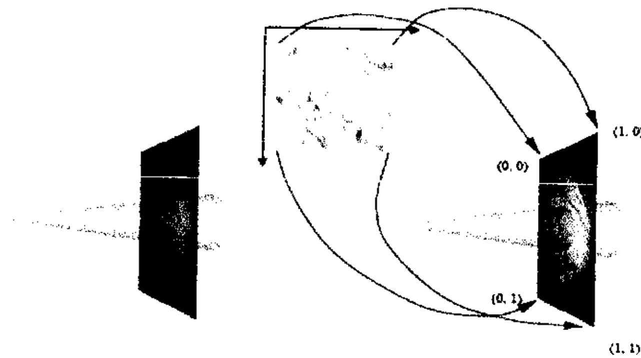


FIGURE 2.9. Modèle de coloriage Gouraud et placage de texture

2.2.4 Optimisation possible du pipeline 3D

Avoir un taux de rafraîchissement supérieur ou égal aux 25 images par seconde (le maximum que l'oeil humain puisse discerner) est d'une extrême importance pour percevoir une navigation de manière fluide. Lorsque le nombre d'images par seconde n'est pas suffisant, il existe quelques techniques pour réduire le temps d'affichage des trames. Ces techniques visent à optimiser soit la scène pour que son contenu soit affiché plus rapidement, soit le code de l'application afin qu'il effectue des optimisations sur toutes les scènes.

Chaque étape du pipeline 3D peut être caractérisée par le type de données sur lequel elle passe le plus de temps à faire ses traitements. Par exemple, l'étape de transformation, qui calcule la nouvelle position de chacun des sommets des triangles, manipule les coordonnées des sommets des triangles ainsi que les données de transformation que sont la translation, la rotation et l'homothétie.

La figure 2.10 reprend chacune des étapes du pipeline 3D avec les données qu'elle manipule. Lorsque l'on connaît les données manipulées par le pipeline, nous pouvons alors déterminer pourquoi elles peuvent le saturer.

Quand le moteur 3D dessine trop lentement la scène, cela signifie que le pipeline a trop de données à traiter pour pouvoir calculer une image en un temps acceptable. Pour optimiser une scène, il convient donc de trouver le type de données qui ralentit le pipeline 3D et cela passe par certaines manipulations. La suppression d'un certain

nombre de ces données pourra accélérer le pipeline et donc la fluidité de la navigation et la réponse aux interactions dans la scène.

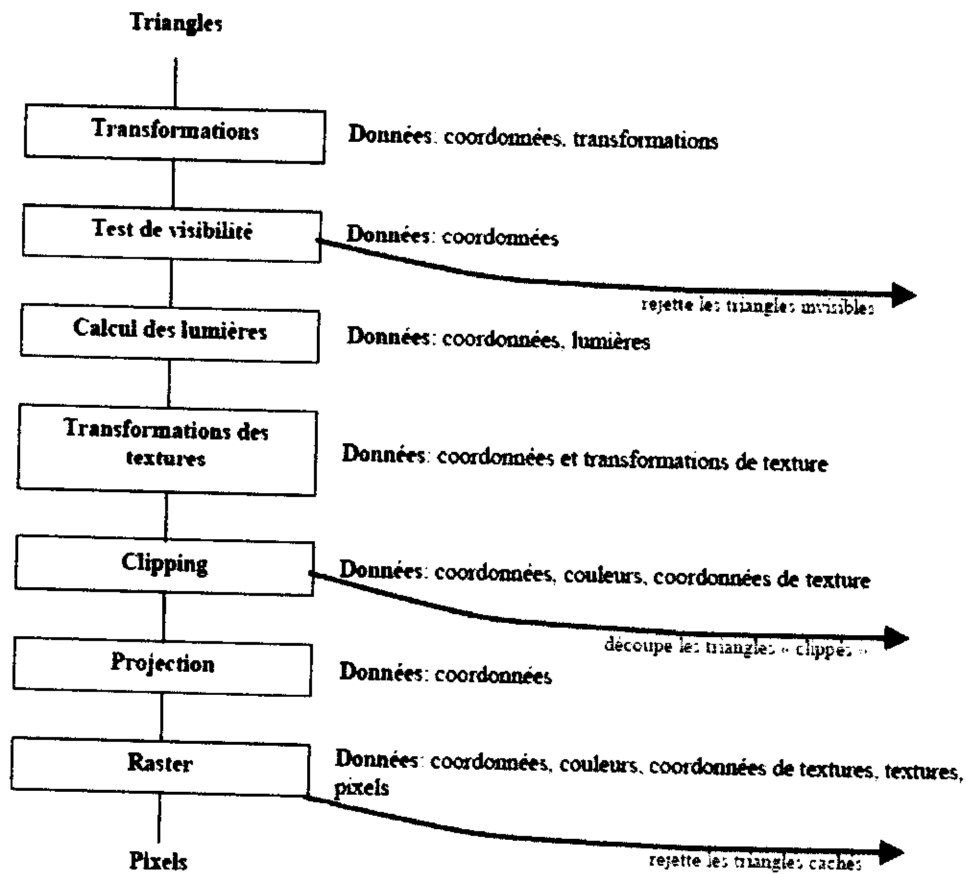


FIGURE 2.10. Données utilisées à chaque étape du pipeline 3D et culling

2.2.5 Elimination des parites cachées

Le problème de la suppression des parties cachées est l'un des plus difficiles de la modélisation 3D. Pour le résoudre, des algorithmes ont été développés afin de déterminer quelles parties d'un ou plusieurs objets (arêtes, côtés, surfaces, volumes) doivent pouvoir être discernées ou non par un observateur situé en un point donné de l'espace. Certains algorithmes favorisent plutôt la vitesse d'exécution. Ils sont en général utilisés pour les simulations en temps réel (comme les simulateurs de vol), étant capables de fournir des solutions à la fréquence vidéo (30 images par seconde).

Tandis que d'autres, plus lents, favorisent la qualité du rendu pour un résultat final plus réaliste, plus détaillé, avec des ombres, des effets de transparence et de textures, etc.. On les utilisera plutôt pour la création d'images de synthèses[WWW5].

2.2.5.1 Principe général des algorithmes de suppression

La plupart des algorithmes de suppression sont basés sur des tris, c'est à dire qu'il va falloir déterminer quelles sont les parties cachées à supprimer d'un objet suivant les coordonnées spatiales de l'observateur (voir figure 2.11).

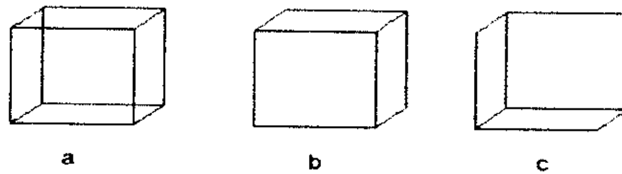


FIGURE 2.11. Les parties cachées dans un cube

Ici, on peut visualiser ce cube sous deux vues, une supérieure droite (b) et une inférieure gauche (c).Il faut donc trier les parties à supprimer suivant la position de l'observateur.

L'une des principales fonctions de ces tris est de déterminer la distance géométrique entre un volume, une surface, un côté ou un point, et le point d'observation. Ils reposent sur l'hypothèse que plus un objet est éloigné du point de vue et plus il y a de chances qu'il soit en partie ou totalement caché par un autre objet, plus proche du point de vue. Il parait donc clair que l'efficacité de ces algorithmes dépend de l'efficacité du processus de tri. On peut classer ces algorithmes en deux catégories suivant le système de coordonnées dans lequel ils sont intégrés

- la première est composée des algorithmes dits de "l'espace objet". Ceux ci sont implantés dans le système de coordonnées physiques (le nôtre). Les calculs effectués fournissent des résultats très précis, à la précision de la machine.

- la deuxième est composée des algorithmes dits de "l'espace image", implantés dans le système de coordonnées de l'écran sur lequel les objets sont visualisés. Pour

cette catégorie, la précision obtenue est plus faible puisque les calculs ne sont effectués qu'à la précision de l'écran.

Voici les caractéristiques pour les deux espaces:

TABLE 2.2. Les caractéristiques de l'éspace d'objet et l'éspace d'image

ESPACE	PRECISION	VOLUME DE CALCUL
Objet	Coordonnées physiques	Fonction ou carré du nombre d'objet
Image	Coordonnées de l'écran	Fonction du nombre d'objets*nombre de pixels

Nous allons maintenant passer à l'étude de quelques algorithmes de ces deux espaces.

2.2.5.2 Les algorithmes de suppression des parties cachées

Algorithme du z-buffer : Le z-buffer (ou mémoire de profondeur) est l'un des algorithmes de suppression des parties cachées les plus simple, c'est un algorithme de l'espace image. Il nécessite une mémoire d'image qui sert à stocker les attributs de chaque pixel (intensité, couleur, etc.) et un tampon des profondeurs séparé pour stocker la coordonnée z, ou profondeur, de chaque pixel visible à l'écran. En bref, pour chaque pixel, on compare la valeur du z d'un nouveau pixel à écrire dans la mémoire d'image à la valeur du z déjà stockée pour ce pixel dans le buffer. Si le nouveau z se trouve devant celui stocké, il est écrit dans la mémoire d'image et sa valeur remplace l'ancienne dans le z-buffer (voir figures 2.12 et 2.13).

La figure (a) présente l'ajout d'un polygone à un écran vide, la (b) présente l'ajout d'un autre, coupant le premier. (Les nombres correspondent à la valeur de z pour un polygone).

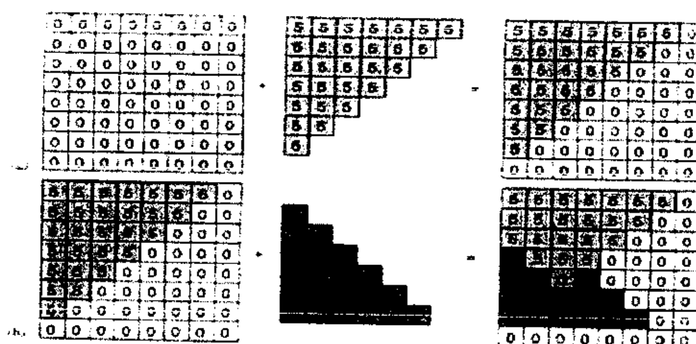


FIGURE 2.12. Le tampen de profondeur selon l'axe Z

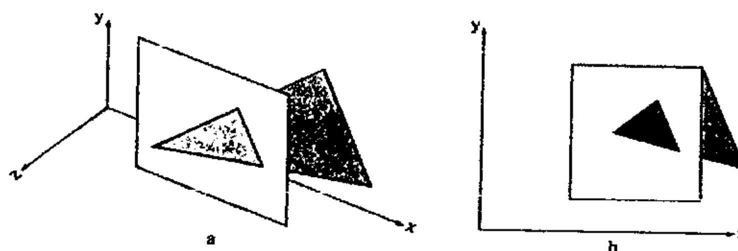


FIGURE 2.13. Présentation 3D et 2D avec l'algorithme Z-Buffer

Nous allons maintenant présenter la méthode algorithmique du z-buffer :

- Mettre la mémoire d'image à l'intensité ou à la couleur du fond.
- Mettre le z-buffer à la valeur de z maximum.
- pour chaque facette de la géométrie
  - calculer le triangle projeté
  - pour chaque pixel du triangle :
    - calculer le point correspondant dans l'espace 3D
    - si le point.Z > zbuffer[x][y] alors

      zbuffer[x][y] = point.Z

      dessiner le pixel de la couleur de la facette

  fsi

Un des principaux avantages de l'utilisation de cet algorithme réside dans sa grande simplicité, il traite aisément le problème de suppression des parties cachées.

Un autre est que pour le z-buffer, aucun tri n'est nécessaire puisque l'opération se résume à une recherche du z le plus grand pour chaque pixel, ce qui réduit le temps de calcul.

Néanmoins, il présente un inconvénient majeur qui est le volume mémoire nécessaire pour son application, augmentant avec la résolution de l'écran (il faut en effet stocker pour chaque pixel ses attributs dans la mémoire image et sa plus grande coordonnée z dans le z-buffer). En général, le z-buffer est réalisé sur 16 bits, ce qui est suffisant pour obtenir un résultat convenable sauf lorsque la scène se compose d'objets de l'ordre du millimètre et assez éloignés les uns de autres.

Algorithme du peintre : Pour cet autre algorithme de suppression des parties cachées, nous ferons une description moins détaillée que la précédente. L'algorithme du peintre diffère du z-buffer d'une part parce que c'est un algorithme dit à "listes de priorités", et d'autre part parce qu'il évolue dans l'espace objet (il est donc plus précis) .Quand on parle de listes de priorités, il s'agit en fait de déterminer un ordre de visibilité de toutes les faces de la scène à modéliser. Cet algorithme travaille comme un peintre le ferait (d'où son nom), c'est à dire qu'il va afficher la scène en fonction de la profondeur des objets qui la composent, partant du premier plan (le plus lointain du point de vue) et allant jusqu'au dernier. On se retrouve finalement avec le dernier plan ainsi que les objets des précédents qui n'ont pas été cachés (voir figure 2.14).

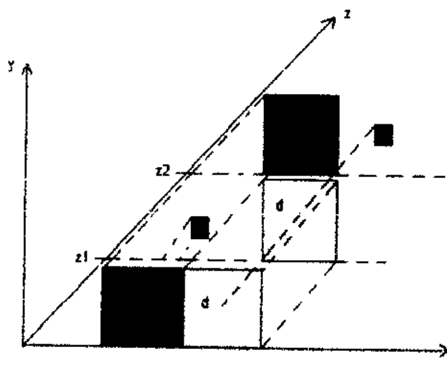


FIGURE 2.14. L'ordre de visibilité des faces de la scène à modéliser

Ici, on affiche d'abord le dernier plan (de profondeur z2), puis le premier plan (de profondeur z1). On obtient donc le résultat affiché sur l'axe x.

Pour ce faire, les trois étapes suivantes sont nécessaires:

- il faut trier les faces des objets en fonction de leur coordonnée z, afin de déterminer sur quel plan elles se trouvent;

- il faut lever l'ambiguïté, c'est à dire qu'en cas de situations complexes (chevauchements, intersections, etc., de certains polygones), il faut diviser ces polygones;

- on affiche les polygones du plus proche au plus éloigné;

Néanmoins, cet algorithme n'est pas très efficace. En effet, pour chaque nouvelle image, on doit refaire les deux premières étapes du processus précédent, ce qui rend le temps de calcul extrêmement long. Mais mise à part la vitesse, cet algorithme est très bien adapté aux situations simples.

Les faces arrière (backfaceculling) : Si on considère une face seule. Une face est décrite par 3 sommets. On décrit donc un morceau de plan. Or un plan n'est qu'une abstration mathématique. Il ne peut pas correspondre à un objet réel. En effet, un plan n'a pas de volume. En d'autres termes, il est infiniment plat. Si on veut décrire des objets réels, il faut impérativement avoir un volume. Pour cela il faut que l'objet soit défini par un ensemble de faces qui décrivent un espace fermé. A l'intérieur de cet espace, on considère qu'il s'agit de la matière de l'objet.

Fort de cette constatation, on se rend compte que certaine faces ne peuvent pas être visibles pour une position d'observation donnée. Cela est vrai si on prend pour hypothèse que la lumière n'est jamais transmise dans la matière. Bref, qu'il n'y a pas d'objet transparent. En effet, si entre l'observateur et la face il y a la matière de l'objet, cela veut dire qu'il y a des faces devant notre face qui la cache pour pouvoir fermer le volume. Dans notre test, on va juste regarder où se trouve la matière par rapport à la face. En fonction de sa position, on pourra se passer ou pas de la face pour construire notre image.

Ce test ne nous permet pas de savoir s'il y a de la matière plus loin que celle qui est directement "accrochée" à la face. Ce qui veut dire que si on a à faire à des objets concaves ou s'il y a plusieurs objets, ce test ne nous permettra pas d'éliminer toutes les faces cachées mais une partie seulement.

Pour effectuer ce test, on va se servir de la normale au plan. On considère que notre normale au plan est orientée vers la direction opposée à la matière et cela tout le temps. Si on est capable d'obtenir cette information, alors il suffit de tester l'angle *theta* entre la normale au plan et le point de vue pour savoir si la face est visible ou pas. Si l'angle theta est compris entre PI/2 et -PI/2 alors la face est visible, sinon elle est cachée par la matière proche (voir figure 2.15).

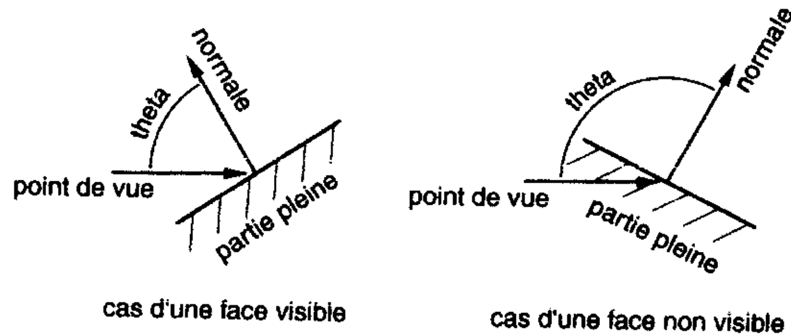


FIGURE 2.15. Test de visibilité d'une face

Algorithmes Hybrides : Ceux-ci consistent à travailler séquentiellement dans les deux espaces, ainsi on optimise la 2eme passe qui parcourt les facettes dans l'ordre optimal.

Algorithme S-buffer : Approximation du Zbuffer, le traitement se fait ligne par ligne et non plus pixel par pixel, l'accélération est considérable.

Algorithme BSP Tree : L'algorithme BSP (Binary Space Partitionning) est un algorithme de l'espace objet. Il consiste à transformer un problème complexe en un ensemble de sous problèmes plus simples. Plus exactement, sa méthode est de diviser la scène avec des plans arbitrairement choisis, ce qui va entrainer la formation d'un arbre binaire. Le BSPtree (Binary space Partitioning tree) est une bonne technique mais son implémentation est beaucoup plus rude. L'idée est de décomposer les facettes et leur intersection dans un arbre binaire.

2.3 La lumière

2.3.1 Les modèles d'illumination

La lumière ambiante

La lumière ambiante correspond au modèle le plus simple. On considère qu'il existe une source lumineuse présente partout et qui éclaire de manière égale dans toutes les directions.

Ce modèle de lumière correspond au niveau minimum d'éclairage qui sera appliqué sur les objets. En terme physique cela correspond un peu au soleil réfléchi par tout l'environnement qui donne une sorte de lumière présente partout.[WWW5]

On définie l'intensité de cette lumière sur une surface comme suit :

$$Ip = pa * Ia$$

Cette intensité lumineuse est constante sur toute la surface. Ia désigne l'intensité de la lumière, pa est le coefficient de réflexion de la lumière ambiante par la surface $(0 <= pa <= 1)$. Ip correspond à l'intensité de la lumière résultant de la réflexion sur la surface.

Prenons un objet fait d'une matière unique. pa reste constant sur la totalité de l'objet. Dans ce cas, l'objet apparait d'une seule et même couleur. Ce modèle d'illumination ne met pas en valeur le volume d'un objet .

La réflexion diffuse

On prend comme hypothèse que la source de lumière est ponctuelle et qu'elle émet de manière constante dans toutes les directions de l'espace.

Dans le modèle de réflexion diffuse, l'intensité en un point d'une surface dépend de l'angle formé entre le rayon de lumière qui touche le point de la surface et la normale à la surface. Plus l'angle formé entre le rayon de lumière et la normale au plan est faible, plus l'intensité lumineuse réfléchie visible par l'observateur est forte.

Le principe physique qui se cache derrière ce modèle est simple. Notre source lumineuse émet une certaine énergie au mètre carré. Suivant l'incidence des rayons lumineux, cette énergie sera répartie sur une plus ou moins importante surface de

l'objet. Si les rayons et la surface sont perpendiculaires, alors l'énergie lumineuse sera répartie sur la plus petite surface possible et donc l'énergie par unité de surface sera maximale (voir figure 2.16).

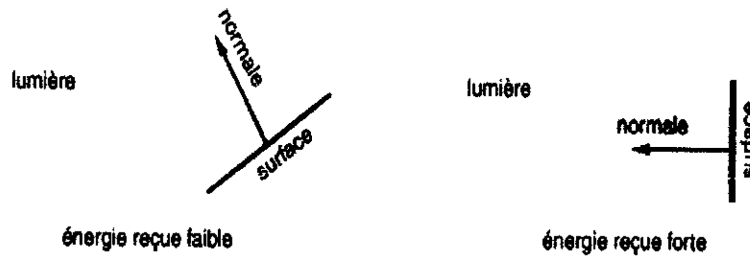


FIGURE 2.16. le modèle de réflexion diffuse

Maintenant, on sait pourquoi l'angle formé entre la normale au plan et les rayons de lumière est important. Par contre, on ne connait toujours pas la lumière qui sera réflechie en direction de l'observateur. Pour cela on se base sur les propriétés de la surface. Si on considère que la surface est une surface Lambertienne (surface mate) comme le papier ou la neige, alors, on peut considérer que la lumière qui arrive sur la surface est réfléchie dans toutes les directions de manière égale. Dans ce cas, la position de l'observateur ne compte pas. La lumière émise en direction de l'observateur dépend donc de l'intensité de la source lumineuse Il, de l'angle theta formé par le rayon de lumière et la normale au plan et du coefficient de réflexion pd de la lumière diffuse par la surface ($0 <= pd <= 1$).On obtient la formule :

$$Ip = pd * Il * cos(theta)$$

Pour la figure :

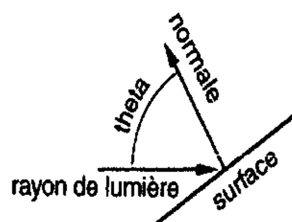


FIGURE 2.17. l'angle entre le rayon de lumière et la normal d'une surface

On notera au passage que si theta est supérieur à PI/2 alors c'est que la face n'est pas du tout éclairée par la source lumineuse car elle lui tourne le dos. Dans ce cas, l'intensité lumineuse est 0.

La réflexion spéculaire

Le modèle de réflexion spéculaire se différencie du modèle de diffusion en faisant intervenir le point d'observation. Dans ce modèle les rayons de lumière sont réfléchis par symétrie par rapport à la normale à la surface. Ce modèle correspond aux propriétés de "mirroir" des objets.

Il faut calculer quel est le rayon réfléchi sur la face. Ensuite, l'intensité de la lumière observée dépend de *theta* qui correspond à l'angle entre le rayon réfléchi et le point d'observation, de l'intensité *Il* de la source de lumière et du coefficient de réflexion *ps* de la lumière spéculaire par la surface ($0 <= ps <= 1$). Si on se prend le rayon réfléchi dans l'oeil alors on a l'intensité maximum. Autour de cela, on peut quand même voir quelque chose d'un peu atténué. Cela veut dire que la surface ne réfléchit pas directement la rayon mais qu'il y a une certaine "diffusion" autour du rayon réfléchi. La fonction cosinus joue bien son rôle ici mais comme on veut pouvoir régler la diffusion autour du rayon réfléchi, on introduit le coefficient n. On obtient la formule :

جامعة البليدة
المكتبة
كلية العلوم

$$Is = ps * Il * cos(theta)\hat{\ }n$$

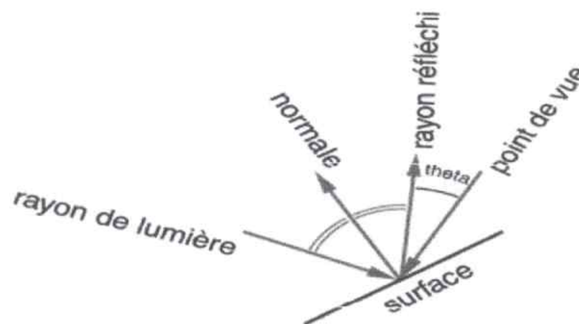


FIGURE 2.18. Le modèle de réflexion spéculaire

2.3.2 Atténuation de la lumière

Jusqu'à présent, on n'a pas pris en compte la distance entre la source lumineuse et la surface. On peut faire intervenir un coefficient d'atténuation de la lumière dans les modèles de réflexion diffuse et de réflexion spéculaire pour pondérer l'intensité de la lumière en fonction de la distance.

Voici un exemple de coefficient de pondération :

$$fd = min(1/(c1 + c2 * dist + c3 * dist\hat{}2), 1)$$

On prend le minimum entre 1 et la valeur calculée pour que l'atténuation ne devienne pas une amplification.

Cette formule fait intervenir trois nouveau coefficient $c1$,$c2$ et $c3$ sur lesquels on peut jouer.

Pour utiliser cette formule dans les modèles de réflexion vue ci-dessus, il suffit de calculer la distance entre la source et le point pour lequel on calcul l'intensité de la lumière. Ensuite, il faut calculer la lumière avec le modèle choisi ainsi que le coefficient fd. Le résultat de l'intensité correspond à la multiplication de l'intensité calculée par le coefficient de pondération fd.

2.3.3 Les modèles d'ombrage

On sait maintenant calculer l'intensité de la lumière en un point. Seulement pour des raisons de performance la plupart du temps, on ne calcule pas la lumière en chaque point. De plus, la plupart des logiciels 3D modélisent les objets avec des polygônes. Seulement l'objet final souhaité n'est pas un objet avec un ensemble de faces plates. On ne peut pas mettre une infinitée de faces plates pour définir des objets courbes. Dans ce cas, il est quand même souhaitable d'avoir un aspect courbé.

Il existe plusieurs façons de procéder pour approximer la lumière en un point. On va voir les plus courantes.

L'ombrage plat

On commence par la méthode la plus simple. On calcule la lumière pour un seul point de la surface que l'on veut représenter et on utilise la même intensité pour

toute la surface. Cette méthode à tendance a beaucoup faire ressortir les polygônes qui représentent un objet.

L'ombrage de Gouraud

Le principe de l'ombrage de Gouraud est de calculer l'intensité de la lumière pour les sommets du polygône et d'interpoler linéairement l'intensité des sommets pour déterminer l'intensité en un point de la face. L'interpolation linéaire se fait sur le polygône projeté.

Cette technique permet d'avoir un lissage qui "gomme" les frontières entre les polygônes que l'on obtient avec un ombrage plat.

Cette technique d'ombrage ne permet pas de voir un point lumineux qui serait au centre d'une face par exemple car on ne calcule la lumière qu'aux sommets.

L'ombrage de Phong

L'ombrage de Phong est assez similaire à l'ombrage de Gouraud à la différence près que ce n'est pas l'intensité lumineuse des sommets que l'on interpole linéairement sur le polygône 3D mais ce sont les normales des sommets.

On a déjà vu que pour calculer l'intensité lumineuse en un point, on a besoin de la normale à la surface en ce point pour les modèles comme la diffusion et la spécularitée. Avec l'ombrage de Phong, on interpole linéairement les normales des sommets pour déterminer une normale en chaque point et on se sert de cette normale pour recalculer en chaque point l'intensité de la lumière. On se rend bien compte que cette méthode est bien plus coûteuse qu'un ombrage de Gouraud met elle permet un meilleur rendu du modèle d'illumination spéculaire avec un plus petit nombre de polygône. En effet, la "tâche" de lumière peu être petite et même entièrement comprise dans un polygône. Dans ce cas, si on calcule l'intensité aux sommets, elle ne sera pas importante alors qu'elle est importante au centre. Avec Gouraud, on ne la verra pas alors qu'avec Phong elle apparaîtra car la lumière est recalculée en chaque point (même si c'est avec une normale interpolée).

2.4 Conculsion

Nous avons présenté dans ce chapitre les notions sur la trois dimensions (3D). Nous avons décrit comment sont obtenues les images sur un écran 2D à partir d'une scène décrite en cordonnées 3D (pipeline de visualisation). Ansi, nous avons décrit les différentes techniques et méthodes de modélisation et de rendu.

Dans le but de développer notre application (voir chapitre IV), on a besoin d'étudier les patterns et les frameworks. Ils sont présentés dans le chapitre suivant (voir chapitre III).

CHAPITRE III

Les Patterns et les frameworks

CHAPITRE 3

Les Patterns et les frameworks

3.1 Introduction

Nous présentons dans ce chapitre les concepts et les notions sur les frameworks et les patterns, nous invitons le lecteur de ce référé aux livres [SHE03], [BOI04], [RAP02] pour plus d'information sur les frameworks et les patterns.

La première référence au terme "pattern" est apparue dans le domaine de l'architecture quand en 1977 et 1979 Christopher Alexander publia ses deux livres traitant des meilleures méthodes, pratiques et manuels d'architecture.[CHR77],[CHR79]

Le concept de patterns remonte au début des années 80, en ce temps là, la programmation structurée était la plus utilisée et les langages orientés objet n'étaient qu'à leurs début (Smalltalk était le langage orienté objet le plus couramment utilisé). L'idée la plus répandue était celle des Frameworks ou squelettes d'application, avec le développement de cette dernière on a vu l'émergence des patterns.

Les frameworks constituent avec les patterns de conception (design patterns)des techniques de modélisation et de réutilisation du logiciel.

Afin de mieux comprendre les principes de chacune de ces deux techniques, nous avons étudié ces deux dernières et nous exposons les points importants et essentiels de cette étude.

3.2 Les patterns

3.2.1 Définition

En général, un pattern décrit un problème qui se produit fréquemment dans la conception et l'implémentation des logiciels, et puis décrit la solution à ce problème de telle manière qu'il puisse être réutilisé. [SHE03]

Selon Bruce Eckel on peut penser qu'un pattern est une manière intelligente de résolution d'une classe particulière de problèmes.

Les patterns sont introduits pratiquement pour documenter une bonne conception ; ils sont considérés comme des véhicules du transfert de la connaissance et d'expérience à partir des experts au débutant. Pour cela, des travaux ont été motivés pour documenter et découvrir des nouveaux patterns dans des divers domaines. Les patterns peuvent être classifiés, selon la phase de développement où ils sont employés , les patterns d'analyse[FOW97], les patterns d'architecture [BUS96] , les patterns de conception [GAM95], et les idiomes.

Le terme "design patterns" est souvent utilisé. Cependant il ne faudrait pas en déduire que l'approche patterns intervient seulement dans la partie conception (design). Cela veut dire qu'un pattern diffère de l'approche traditionnelle qui est une phase d'analyse puis de conception et enfin d'implémentation. Un pattern est une idée sans programme qui peut apparaître dans la phase d'analyse ou à un haut niveau de conception, il peut être directement implémenté par un langage de programmation.

3.2.2 Classification des patterns de conception:

Les patterns de conception (Design patterns) sont classifiés selon trois objectifs suivants:[GAM95]

Créateur (Creational): La meilleure façon de créer des instances d'objets est l'objectif de cette classe de patterns, cela rend le programme indépendant de la manière avec la quelle les objets sont crées.

Parmi les patterns créateurs, on peut citer:

- **Abstract Factory** : interface pour la création de familles d'objets sans spécifier les classes concrètes.

- **Builder** : séparation de la construction d'objets complexes de leur représentation afin qu'un même processus de construction puisse créer différentes représentations.

- **Factory Method** : définition d'une interface pour la création d'objets associés dans une classe dérivée.
- **Prototype** : spécification des types d'objet à créer en utilisant une instance prototype.
- **Singleton** : comment assurer l'unicité de l'instance d'une classe.

Structurel (Structural): Les patterns structurels décrivent la manière avec la quelle la classe et les objets peuvent être combinés pour former des structures plus larges.

On peut citer parmi les patterns structurels:

- **Adapter** : traducteur adaptant l'interface d'une classe en une autre interface convenant aux attentes des classes clientes.
- **Bridge** : découplage de l'abstraction de l'implémentation pour faire varier les deux indépendamment.
- **Composite** : structure pour la construction d'agrégations récursives.
- **Decorator** : extension d'un objet de manière transparente.
- **Facade** : unification de plusieurs interfaces de soussystèmes.
- **Flyweight** : partage efficace de plusieurs objets.
- **Proxy** : approximation d'un objet par un autre.

Comportemental (Behavioral): Ce type de patterns traitent de la communication entre objets. Parmi les patterns Comportementaux on peut citer:

- **Observer** : mise à jour automatique des dépendants d'un objet.
- **State** : permettre à un objet de modifier son comportement lorsque son état interne change.
- **Strategy** : abstraction pour sélectionner un algorithme parmi plusieurs.
- **Template method** : définition d'un squelette d'algorithme dont certaines étapes sont fournies par une classe dérivée.
- **Visitor** : représentation d'opérations devant être appliquées à des éléments d'une structure hétérogène d'objets.

• **Command**: encapsulation de requêtes par des objets afin de permettre à un objet de traiter plusieurs types de requêtes.

• **Interpreter** : étant donné un langage, représentation de la grammaire le définissant pour l'interpréter.

• **Iterator** : parcours séquentiel de collections.

• **Mediator** : coordination d'interactions entre des objets associés.

• **Memento** : capture et restauration d'états d'objets.

3.2.3 Exemples des patterns de conception (Design patterns)

Nous présentons dans cette section quelque patterns de conception concrète,ceux que nous avons utiliser dans la conception et la description à un niveau plus abstraite les composants de notre framework.[SHE03]

3.2.3.1 L'architecture Model\View\Controller (MVC)

L'architecture MVC a pour but de faciliter la programmation dans le cas des systèmes ou on a besoin de présenter une même donnée de manière synchrone et multiple.

Le modèle MVC sépare un composant logiciel en trois parties : un modèle, une vue, et un contrôler (voir la figure 3.1).

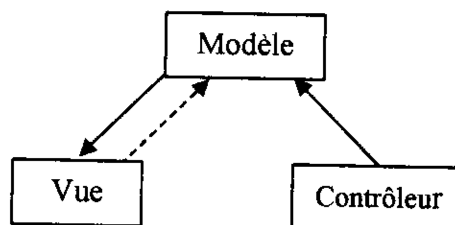


FIGURE 3.1. L'architecture MVC

• Le modèle(*Model*) est la partie qui représente le comportement et l'état du composant logiciel.

• La Vue (*View*) est la partie qui permet la visualisation de l'état représentant le modèle.

• Le contrôleur (*controller*) est la partie qui gère l'interaction de l'utilisateur avec le model, c'est à dire permet de changer l'état du modèle.

Il est important de noter que :

• Le modèle n'as aucune connaissance particulière sur ces contrôleurs, c'est le système qui avertit les vues lors d'un changement d'état dans le modèle.

• Un modèle peut avoir plusieurs vues et plusieurs contrôleurs.

3.2.3.2 Le pattern L'observer

Le pattern Observer a deux interfaces qui laissent coordonner le sujet observé avec ses observateurs. Ces interfaces sont l'opération notify() dans le sujet et l'opération d'update() dans l'observateur.

Deux classes sont alors utilisées pour implémenter le pattern observer, la classe abstraite observer et la classe observable. Tous les objets qui désirent être avertis lors d'un changement dans un autre objet doivent implémenter la classe observer. Les objets qui notifient d'autres objets d'un changement qu'ils ont subis doivent quant a eux étendre (hériter) de la classe observable.

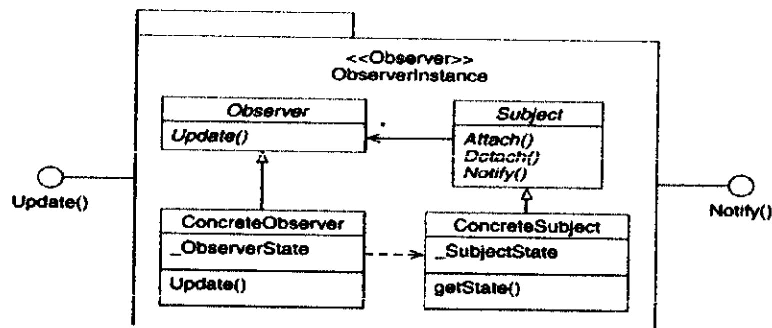


FIGURE 3.2. Diagramme de classe du pattern Observer

Le principe de ce pattern est très simple :

• L'observable (*model*) subit un changement qui peut intéresser d'autres objets (de type observer), il avertit alors ces observateurs de ce changement.

• L'observer(*view*)ayant reçu un avertissement effectue les opérations nécessaires ou prévues lors d'un changement dans l'observable émetteur de cet avertissement.

3.2.3.3 Le pattern Strategy

Le pattern strategy a une classe context car l'interface encapsule la stratégie de contrôle. On pourrait également employer l'opération ContextInterface() comme une interface d'opération pour le pattern strategy.

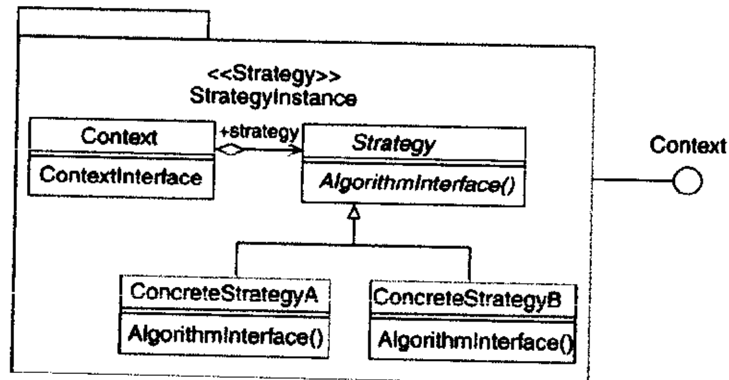


FIGURE 3.3. Diagramme de classe du pattern Strategy

3.2.3.4 Le pattern Singleton

Le pattern singleton s'applique aux nombreuses situations dans les quelles il faut créer une seule instance d'une classe, un seul objet. Il est souvent laissé jusqu'au programmeur pour s'assurer qu'une considération importante en mettant en application ce pattern est comment rendre cette instance facilement accessible par d'autres objets.

3.2.3.5 Le pattern Composite

L'interface d'un pattern composite est la classe Component par laquelle ses éléments se constituent, simple (un fils feuille) ou composé (parent), interface à d'autres composants.

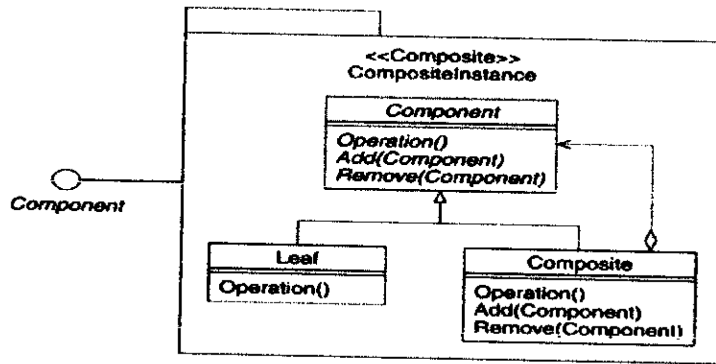


FIGURE 3.4. Diagramme de classe du pattern Composite

3.2.3.6 Le pattern AbstractFactory

Le pattern AbstractFactory a deux interfaces. La première interface est pour un objet factory(objet d'usine), l' interface est AbstractFactory, qui fournit une interface pour créer de divers types de produits. La deuxième interface est pour le produit créé par l'usine, la classe interface AbstractProduct fournit une interface pour tous les divers produits créés par le l'usine(factory).

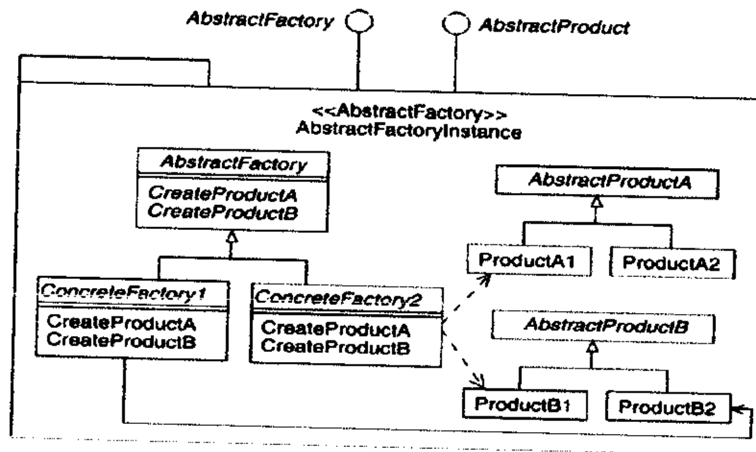


FIGURE 3.5. Diagramme de classe du pattern AbstractFactory

3.3 Les frameworks

3.3.1 Définition d'un framework

Afin de définir un framework, nous préférons citer quelques définitions plutôt que d'en créer une nouvelle. Ainsi, D'après Ralph Johnson [JOH97] : « a framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact ». Celle-ci peut être complétée de la définition suivante: « a framework is the skeleton of an application that can becustomized by an application developer » [JOH93] .Ces définitions sont les plus utilisées, cependant certaines personnes omettent la dimension liée au domaine d'application, et d'autres vont jusqu'à affirmer qu'un framework est « A design pattern implementation » [LOR93] quand il ne s'agit pas d'une simple « classe abstraite» [PRE00] .

Un framework a pour but de faciliter la construction d'une application en fournissant un petit ensemble de composants (classes) connus et éprouvés à composer ou à étendre. Il va au-delà de la simple réutilisation de code en favorisant la réutilisation d'analyse et de conception. Nous retiendrons principalement de cet ensemble de classes son caractère adaptable et extensible, et la représentation partielle incomplète qu'il propose d'une application.

Un framework a donc à la fois un aspect structurel (une architecture de classes) et un aspect comportemental (les fonctionnalités fournies par ces classes). Ils couvrent des domaines aussi variés que les interfaces graphiques au sens large (HotDraw, ET++, AWT, X11), la construction d'environnements de développement (Eclipse), ...etc.

3.3.2 Pourquoi utiliser un framework ?

Un framework est utilisé comme base à l'écriture d'une application, car il fournit des éléments à partir desquels il est plus facile de construire une application.

Les bénéfices liés à l'utilisation d'un framework sont:

- la cohérence entre les différents produits dérivés de celui-ci ;
- l'augmentation de la productivité;

- la réduction de la maintenance ;
- l'économie d'argent.

Ces bénéfices ne deviennent réels que si le framework est utilisé sur une longue période de temps. En effet, l'apprentissage d'un framework est plus long que l'apprentissage d'une simple bibliothèque, et peut parfois être aussi long que de concevoir une application. Cela est dû à la complexité du framework qui ne résout pas un problème particulier, mais une généralisation de ce problème. Il est donc conseillé de réutiliser le même framework plusieurs fois. Par ailleurs, la construction d'un framework est une activité encore plus coûteuse. Elle requiert une très grande expertise du domaine à modéliser de manière à fournir une flexibilité suffisante aux développeurs et la validation d'un framework n'est faite qu'une fois la mise en oeuvre d'applications suffisamment différentes effectuée .[JOH93]

3.3.3 Structure d'un Framework

Les frameworks sont typiquement implémentés en utilisant des langages orientés objets comme Java, et ils profitent des techniques fournissent par ces langages : les classes abstraits, polymorphisme, l'héritages et la composition des objets.

3.3.3.1 Les couches d'un framework

Les objets dans les frameworks sont la plupart du temps décrits avec des classes abstraites. Pour des applications ils sont des templates afin de créer des sous-classes, et pour le framework ils sont des conceptions de ces composants. Cette conception inclut l'interface et souvent un noyau pour leur implémentation.

Par exemple une classe abstraite peut définir un squelette pour un algorithme, de même les templates dans les patterns.

Chaque étape dans l'algorithme est définie comme un appel à une méthode abstraite (dans la même classe ou dans une autre classe).

Un framework peut indiquer une implémentation par défaut pour ces méthodes ou les laisser abstraites (non implémenté).

Une application dérivée du framework peut directement utiliser les implémentations existantes par défaut ou spécifier des nouvelles classes concrètes qui héritent

de la classe abstraite et appliquent ses méthodes abstraites.

Le langage Java, sépare les classes et les interfaces. Cependant, Les interfaces indiquent seulement des aspects statiques, parce qu'un framework est également le modèle de collaboration ou le modèle de l'interaction d'objet. Les frameworks écrits contient une interface et une classe abstraite pour leurs composants.

Il est préférable de séparer les interfaces de framework et l'implémentation abstraite (partielles).Ceci est le résultat d'un framework avec une structure claire. Les couches les plus élevées (plus abstraites) sont indépendantes des couches inférieures (plus concrètes), ainsi que les couches inférieures peuvent être remplacées sans que les plus élevées seront changées.

Nous classons ici les couches de framework comme couche d'interface, couche d'implémentation de noyau du framework, et couche des composantes par défaut (voir la figure 3.6). La couche d'implémentation de noyau du framework, par exemple, peut contenir les classes abstraites qui prolongent d'autres classes abstraites.

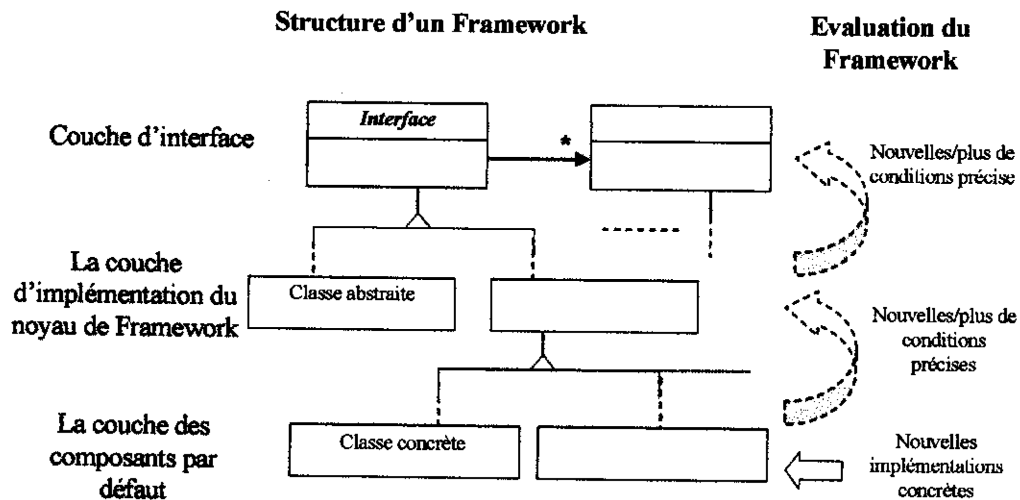


FIGURE 3.6. La structure et l'évaluation d'un framework typique

- La couche d'interface comprend des interfaces (entièrement abstraites) qui définissent les objets de base du framework, leurs services, et leurs rapports en termes de signatures de méthode. La couche d'interface est entièrement indépendante des couches inférieures (les implémentations des interfaces).

- La couche de noyau d'implémentation du framework fournit le comportement par défaut du framework en mettant partiellement une implémentation de la couche d'interface en classes abstraites. L'implémentation abstraite est prévue pour être employée dans la plupart des cas de développement d'application.

- La couche composante par défaut contient la totalité des implémentations par défauts (classes concrètes). Les frameworks possèdent généralement ce genre de bibliothèques de composantes qui sont leurs compagnons de base, mais elles ne sont pas leurs essences. L'essence est le modèle de l'interaction et le contrôle de l'information entre les objets du framework.

3.3.4 Types de frameworks

Bien qu'on puisse caractériser les frameworks par leur domaine d'application, leur taille, leur langage d'implémentation,le fait qu'ils possèdent ou non leur boucle de contrôle,...etc. Nous définissons leur type d'utilisation et leurs mécanismes d'adaptation.

On désigne deux types de frameworks : les frameworks blancs et les frameworks noirs. La couleur ne représente pas une dichotomie stricte, mais décrit un style d'utilisation d'un framework qui n'est jamais ni complètement blanc, ni complètement noir.

3.3.4.1 Frameworks blancs

Le terme *framework blanc* désigne à la fois une forme de framework et sa technique de réutilisation. Ainsi l'utilisateur d'un framework blanc a accès au code source et doit généralement l'étudier avant de pouvoir l'étendre.

L'utilisation d'un tel framework, principalement basée sur l'héritage, est très liée à des fonctionnalités du langage d'implémentation (héritage, liaison dynamique, polymorphisme), et se fait en ajoutant de nouvelles classes, sous-classes, surchargeant des méthodes, etc. Bien qu'avec un tel framework toutes les modifications soient possibles, elles se produisent dans des endroits identifiés où l'adaptation a été prévue. On identifie ces endroits par les termes de « hotspots » [PRE94] ou « axis of variation » . Nous traduirons ces termes par point de variation. Un point de variation identifie

une classe ou un ensemble de classes à modifier afin de particulariser le framework. Ces points de variation sont décrits dans la documentation qui est associée au framework. Par exemple : HotDraw et JUnit sont des représentants de cette catégorie de frameworks.

3.3.4.2 Frameworks noirs

Comme le terme framework blanc, le terme *framework noir* représente à la fois une catégorie de framework et une technique de réutilisation. Un framework noir est caractérisé par l'impossibilité d'accéder au code source, mais surtout par la technique d'utilisation qui lui est liée. Ainsi l'utilisation d'un framework noir se fait par paramétrisation de classes et par l'assemblage de composants. Tout comme dans les frameworks blancs, ces modifications sont apportées aux points de variation. Le framework COM est un représentant de cette catégorie.

Cette distinction entre framework noir et blanc résulte de la dualité entre la facilité de la conception et la facilité d'utilisation. Un framework noir est plus difficile à développer mais plus simple à utiliser, alors qu'un framework blanc est plus simple à développer, mais plus complexe à utiliser.

3.3.5 Cycle de vie d'un framework

Que ce soit dans un cycle en V, cascade, spirale ou autre, développer une application (à partir ou non d'un framework) ou un framework requiert les mêmes activités : analyse, conception, tests unitaire, tests d'intégration, validation et packaging. Cependant si l'on n'y regarde de plus près la complexité, l'importance et la motivation de chacune de ses activités n'est pas la même selon les cas.

Ainsi la flexibilité requise par un framework ne peut être atteinte qu'au prix d'une analyse et conception cherchant explicitement cette flexibilité et réutilisabilité, ce qui n'est pas le cas dans le développement d'une application.

Au-delà de ce point, la principale différence se trouve lors des tests d'intégration et de la validation. Alors que dans le contexte du développement d'une application ces activités consistent à vérifier le fonctionnement des classes écrites et à vérifier l'adéquation de l'application avec les besoins du client, cela est beaucoup

plus complexe dans le cas du framework. En effet, les besoins ne sont pas aussi clairement établis et les tests d'intégration et la validation doivent donc permettre de s'assurer que le framework offre bien la flexibilité nécessaire au développement d'applications(voir figure 3.7).

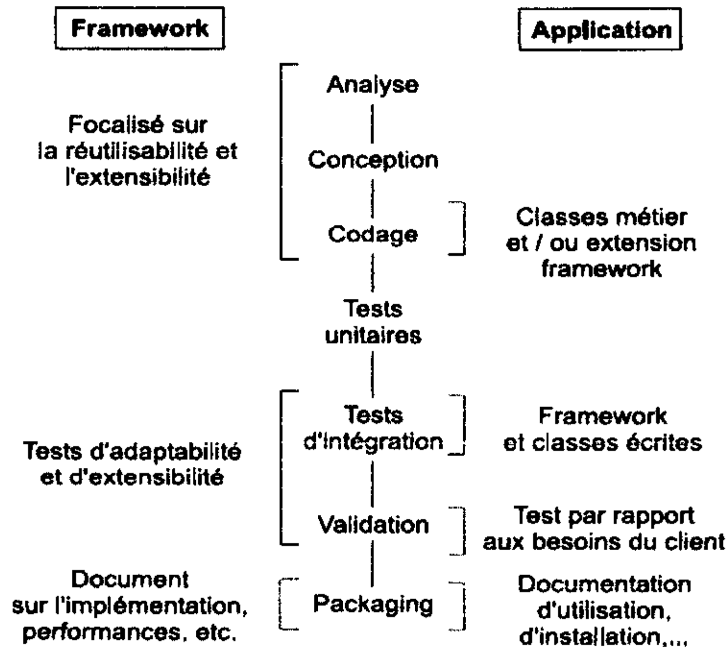


FIGURE 3.7. Activités rencontrées lors du développement d'une application et d'un framework

L'autre différence notable entre le développement d'une application et d'un framework, qui est une conséquence du public visé, se trouve dans le packaging (la mise en forme prête à l'emploi). Dans le cas d'une application la documentation doit expliquer comment l'utiliser, accéder à une fonctionnalité ou montrer des captures d'écran, alors que pour un framework il est nécessaire de fournir le domaine visé, les possibilités d'extension, les performances, des exemples d'utilisation, etc. Ces informations sont d'autant plus importantes qu'elles permettront à l'utilisateur de tirer l'avantage du framework.

3.3.6 Utilisation de frameworks

Nous présentons maintenant les problèmes posés par l'utilisation des frameworks.

De par les nombreuses possibilités qu'offre un framework, le terme utilisation prend plusieurs significations.

2.3.6.1 Personnalisation

Dans un framework blanc le terme utilisation a souvent la signification d'extension (création de nouvelles classes, adaptation de la structure du framework...). C'est ce que nous appelons personnalisation. Les problèmes rencontrés lors de la personnalisation d'un framework sont essentiellement dus aux dépendances inséparables à la structure du framework, aux éléments constituant l'application voire à la combinaison des deux. Bien qu'ayant un aspect comportemental, ces dépendances sont principalement structurelles, et leur respect est primordial pour assurer le bon fonctionnement du framework. En particulier leur non considération peut entraîner des erreurs à la compilation ou à l'exécution. Cependant ces dépendances ne font pas l'objet d'une attention particulière lors de la conception du framework. Elles sont considérées comme faisant partie de l'implémentation, et ne sont donc prises en compte que tardivement lors de la création de documentation.

Parfois la modification aux points de variation ne suffit plus pour étendre le framework. Bien que cela ne soit pas conseillé pour des aspects de portabilité, l'utilisateur a recours à la modification de classes centrales au framework. Pour cela il peut soit modifier le code source du framework, ce qui pose des problèmes liés à la traçabilité des modifications, soit créer de nouvelles sous-classes, ce qui pose des problèmes liés à l'instanciation et au référencement de cette nouvelle sous-classe.

2.3.6.2 Instanciation

La seconde signification du terme utilisation décrit l'instanciation d'un framework pour la réalisation d'une application. Ceci consiste à la création et la connexion d'instances soit des classes du framework, soit des classes obtenues par personnalisation. Ainsi, pour un framework blanc elle se produit après la personnalisation, alors que pour un framework noir elle consiste plus généralement à sélectionner les composants intéressants et à les connecter. La difficulté de cette utilisation réside dans

la compréhension des dépendances comportementales existant entre les instances à l'exécution, et le protocole d'utilisation de chacune de ces instances. En effet, de par sa complexité, un framework implique certainement des dépendances comportementales entre les éléments qui le composent. Ainsi certaines classes doivent être initialisées avant d'être appelées, ou les appels de méthodes de plusieurs composants doivent être enchevêtrés dans le temps. Les erreurs générées par la mauvaise utilisation du protocole du framework sont d'autant plus difficiles à corriger que, pour les reproduire, il faut parfois rejouer de longues séquences de tests.

2.3.6.3 Difficultés liées à la documentation

L'une des causes des problèmes rencontrés lors de la personnalisation et de l'instanc-iation est la mauvaise documentation des frameworks. Cela est dû à la perte des informations de conception. On compte parmi ces informations les schémas de conception utilisés, le comportement des composants, les dépendances entre tous les éléments, la localisation des points d'extension, etc. Cette mauvaise documentation a pour cause : l'inadéquation entre l'information à représenter et les supports disponibles rendant délicat l'accès à l'information à travers des outils d'aide à l'utilisation, la mauvaise intégration de cette documentation, le coût de production de la documentation...etc.

3.4 Conclusion

Nous avons présenté dans ce chapitre les notions essentielles sur les patterns et les frameworks. Nous avons décrit aussi la structures et l'architecture des frameworks pour réaliser et poser les premières étape afin de commencer la réalisation de notre framework présenté dans le chapitre suivant.

CHAPITRE IV

Développement de l'application

CHAPITRE 4

DÉVELOPPEMENT DE L'APPLICATION

4.1 Introduction

Dans ce chapitre nous présentons le framework que nous avons réalisé,les étapes que nous avons suivit pour son développement et les méthodes utilisées dans la modélisation et la conception. Le framework est implémenté en java et pour tester l'utilisation de ce framework dans une application, nous avons implémenté un moteur de rendu 3D sous la plateforme J2ME (MIDP2.0, CLDC 1.1).

Notre framework est fondu sur des patterns de conceptions. Ils nous aiderons à réaliser une conception universel et flexible. Ce framework est fondé sur l'héritage et la composition, qui nous permet de brancher d'autres classes et composants.

Comme n'importe quelle méthodologie de développement d'un logiciel, le développement d'un framework commence par analyser les besoins d'application. Le processus d'analyse tend à produire les objets de bases qui sont les plus appropriés à la phase de conception et au reste du procédé de développement. Après, nous passons à l'étape de conception et l'implémentation. A la fin, l'étape de test qui signifie l'écriture des applications utilisant le framework.

Pour modéliser et concevoir notre framework, on a utilisé quelques digrammes de langage UML :

- Le diagramme de classe, pour représenter l'architecture du framework (vue statique) et le moteur 3D.

- Le diagramme de séquence, pour représenter les interactions entres les composants du framework.

4.2 Analyse

La phase d'analyse que nous avons suivit contient un ensemble d'activités. Les activités principales dans cette phase sont :

- L'analyse de besoin, pour identifier les composants logiques du système et les problèmes à résolus dans l'application.

- La récupération des patterns pour choisir un ensemble de patterns candidats.

- La sélection des patterns à partir de l'ensemble de patterns candidats pour qu'ils seront inclus dans la phase de conception.

4.2.1 Analyse des besoins

Dans la partie d'analyse des besoins, nous analysons le domaine d'application du framework. Le framework que nous avons développé définit un squelette de base pour un moteur de rendu (ensemble d'APIs 3D) et utilise les techniques de programmation graphique de trois dimensions qui permettent :

- la modélisation géométrique des objets 3D simples (cube, sphère, ...etc.) ou composés,

- la modélisation radiométrique (illumination et calcul d'intensité),

- la modélisation des scènes 3D,

- l'animation des modèles 3D.

- le rendu dans l'écran.

Le framework peut être utilisé pour développer des moteurs de rendu 3D, il est considéré comme une interface entre le langage Java et la machine (les terminaux mobiles, PDA aux autres machines qui possèdent une plateforme J2ME).

Pour développer le framework, nous avons besoins de décomposer l'architecture du moteur 3D et savoir les composants principaux de ce dernier.

Après les études que nous avons effectué sur la modélisation 3D et le développement des moteurs 3D, nous pouvons décomposé le système en composants basés sur des fonctionnalités indépendantes :

- Le noyau du moteur, qui définit les APIs nécessaires pour le rendu des objets 3D simples ou des scènes 3D et qui peut être lié à un moyen d'affichage. Il sera la

cible de rendu (rendering target) ,

- Le module de rendu, qui définit les différents fonctions de rendu 2D (pixels ,ligne, triangle, triangle texturé, ...etc). L'implémentation de ce module dépend des bibliothèques graphiques utilisées (par exemple dans J2ME, on utilise les fonctions de la classe Graphics.java) ,

- La camera est un objet 3D qui possède une position (x,y,z) et une orientation de vue.

- Les modèles 3D (simples ou composés), qui sont des objets possédant une liste de points et de faces.

- La scène 3D, qui définit la structure de la scène et ces composants (objets 3D, Camera,...etc).

- Les composants et les outils qui assurent les opérations et les calculs mathématiques (sinus, cosinus, ...etc) et les transformations géométriques (vecteurs, sommets, matrices, transformations sur les matrices),

- Les composants de modélisation radiométrique indiquent les valeurs des paramètres intervenant dans le modèle d'illumination et qui déterminent l'éclairement des objets 3D en cohérence avec la position des sources de lumière et de l'observateur (Camera).

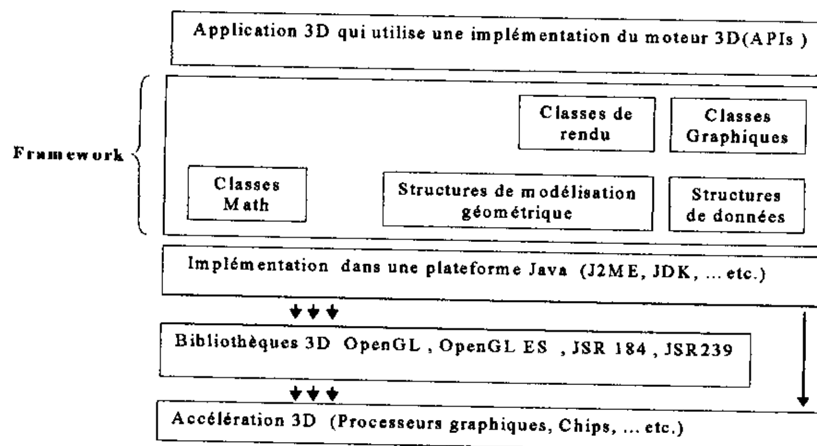


FIGURE 4.1. L'architecture du notre framework dans un système

4.2.2 Sélection des patterns

Dans l'étape de sélection des patterns, nous analysons les responsabilités et les fonctionnalités de chaque composant afin d'identifier les patterns qui peuvent fournir une solution de conception pour chaque composant, à savoir :

• Le moteur 3D est basé sur le pattern de conception MVC (voir chapitre III-section III.2.3). La vue est le responsable de l'affichage des informations dans une interface utilisateur qui est dans notre cas l'image rendue. Le contrôleur traite l'interaction de l'utilisateur sur la fonctionnalité de l'application. Le modèle relie la vue et le contrôleur, il comprend la logique et les données de l'application. La vue est avertie des changements des données. L'utilisateur doit fournir sa propre vu (image, canvas, ...etc) pour représenter graphiquement les objets 3D du modèle. Le gestionnaire des API 3D devient un modèle. Le contrôleur dans cette architecture est tout objet qui peu modifier la vue (l'image ou le graphique du rendu), nous pouvons remarqué que :

- Le La gestionnaire des API 3D est un modèle et en partie un contrôleur, il influe sur la vue par la modification de mode du rendu, la résolution de la fenêtre d'affichage, l'arrière plan...etc.

- La camera est un contrôleur, la transformation de la camera engendre la modification de matrice de projection se qui modifie la scène à afficher et la vue indirectement.

- Tout objet transformable et qui est dans la scène 3D (modèles 3D, lumières, les groupes de transformations...etc).

• On peut utiliser le pattern Abstract Factory (voir chapitre III-section III.2.3) dans plusieurs cas, par exemple dans la modélisation des différents types de lumières (lumière ambiante, lumière directionnelle, ...etc) et les matrices des transformations.

• L'utilisation de plusieurs techniques (ZBuffer, algorithme de peintre ,...etc) dans le rendu nous a conduit d'utiliser le pattern Strategy dans la modélisation du rendu (par exemples il y plusieurs stratégies et algorithmes pour éliminer les parties cachées, trie des facettes,...etc).

• Le MVC est introduit aussi dans la modélisation de la camera. Elle peut être

vu comme un modèle et un contrôleur pour gérer ces transformations (rotations, déplacements,...etc.). La camera peut gérer plusieurs vue qui sont vus comme des observateurs.

• L'architecture MVC est réalisée à l'aide du pattern Observer.

• Le gestionnaire des APIs 3D du moteur 3D utilise plusieurs fonctions et ressources de données pour réalisé le rendu 3D (calcul de lumière, transformation des facettes, trie des facettes, projection...etc.) et d'autre ressources de stockage (cordonnés des objets 3D, les couleurs, les normales des facettes . . . etc). Pour cela on doit utiliser le pattern singleton (voir chapitre III-section III.2.3) pour assurer l'unicité de créer un seul objet de cette classe dans un programme.

• La structure de la scène peut être modélisée sous forme hiérarchique d'arbre, où la racine et les nœuds sont des composants de transformations (rotation, translation, changement d'échelle) et les feuilles sont des composants qui peuvent être rendus telle que les objets 3D et la lumière. Cette structure hiérarchique de classe a été présentée dans le pattern de conception Composite (voir chapitre III-section III.2.3).

• Concernant le composant responsable de graphisme 2D (pixel, ligne, triangle,. . . tec.), il y a plusieurs stratégies pour réaliser ces tâches : soit on utilise les fonctions prédéfinis dans le système, soit on redéfinit ces fonctions et dans ce cas il y a plusieurs algorithmes qui peuvent être implémentés. Le pattern Startegy est une solution pour résoudre ce problème (voir chapitre III-section III.2.3), il donne la flexibilité d'ajouter d'autres composants et d'augmenter la réutilisation de l'algorithme. La solution est de définir une classe abstraite qui contient toutes les fonctions de graphisme 2D et les différentes implémentations sont encapsulés dans des composants externes.

4.3 Conception et implémentation du framework

L'étape de conception que nous avons suivit commence par une conception dite "globale" qui décrit l'architecture du framework, elle est suivie par l'étape de conception détaillée pour décrire les composants du système.

4.3.1 L'architecture du Framework

Le résultat de cette étape est une description générale de l'architecture du framework permettant d'identifier les différents composants et leurs liens.

Dans cette étape, nous utilisons le diagramme de classe pour identifier à partir de l'étape de spécification des besoins, les principaux composants du framework à construire.

Le digramme de classe nous permet de modéliser, d'une manière statique, les relations qui existent entre l'ensemble des classes. Il développe d'une part la structure des entités du framework et d'autre part celle d'un code orienté objet.

Afin de permettre au lecteur de bien comprendre notre framework, nous présentons dans ce qui suit une vue d'ensemble sur son l'architecture (figure 4.2). Ensuite nous allons œuvrer à décomposer et à détaller chaque composant à part.

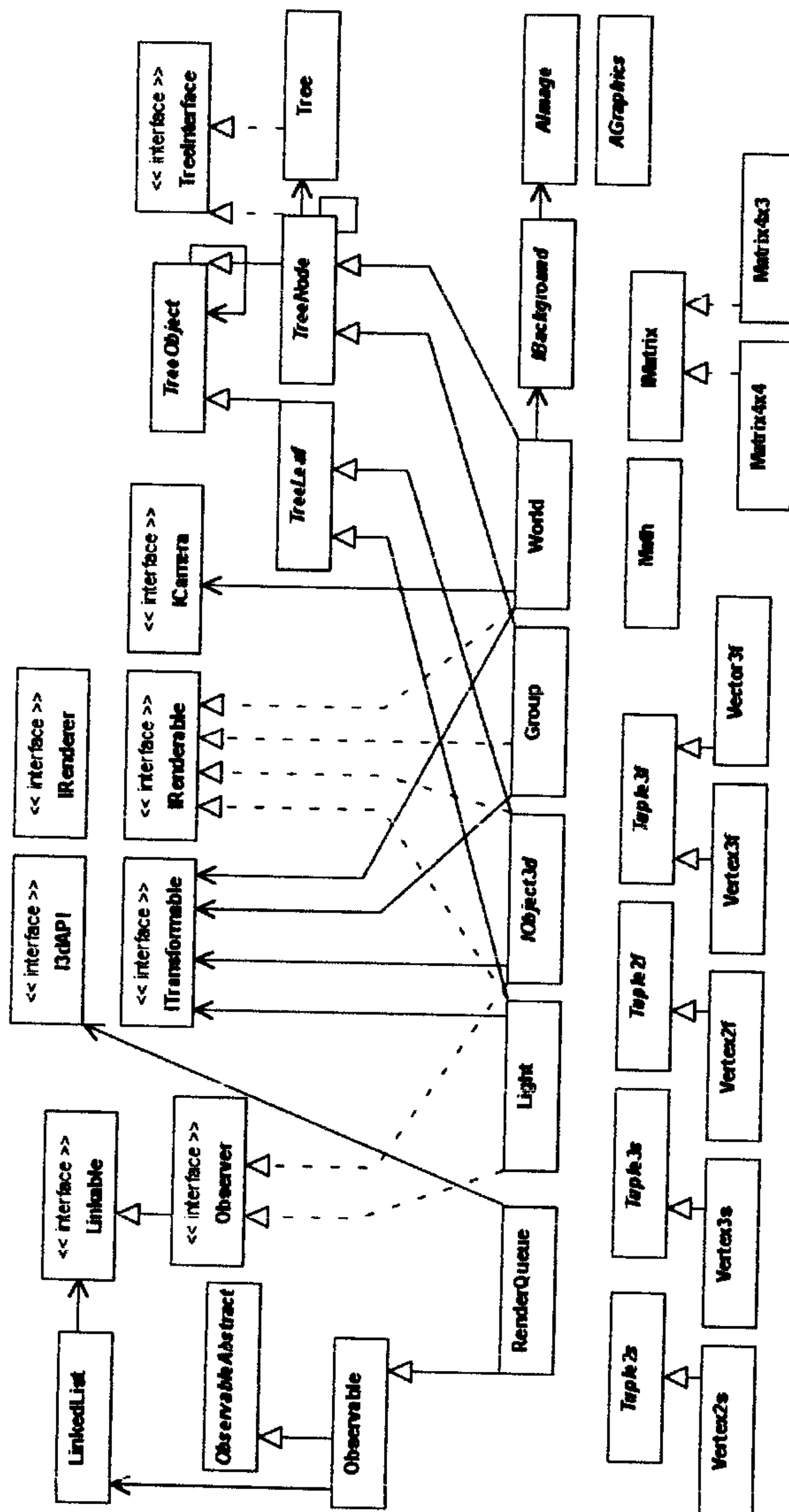


FIGURE 4.2. Diagramme de classe de l'architecture du Framework

4.3.2 La classification

Une des tâches importantes pendant les phases d'analyse et conception est la classification. Pendant cette phase, on cherche l'ordre entre les différentes abstractions. L'identification des ordres mène par la suite à identifier des classes.

L'intérêt des concepteurs orienté objet est d'identifier la hiérarchie des classes pendant que ces classes sont identifiées. Ceci mène à séparer l'abstraction commune à une classe basse et à employer la transmission pour définir les deux (ou plus) classes dérivées. La collection de la structure parent-fils définit une hiérarchie de classe. Un exemple de hiérarchies de classe est donné par la suite.

Les points et les vecteurs sont des triplets

Supposer que nous voulons définir une classe des points dans l'espace 3D, appelée Vertex3f .La classe Vertex3f stockerait les coordonnées (x, y, z) de chaque instance de la classe, elle fournirait également un constructeur qui prend les coordonnées comme paramètre pour initialiser l'objet et des méthodes pour lire et modifier x, y et z.

Supposons que nous voudrions également définir une classe des vecteurs dans l'espace 3D, appelé Vector3f. La classe Vector3f doit stocker les coordonnées (x, y, z) de chaque vecteur et fournir un constructeur, des méthodes pour rapporter les coordonnées et d'autres fonctions appliquées sur les vecteurs (produit scalaire, produit vectoriel, normalisation ... etc).

Pour profiter de l'avantage de la ressemblance entre les deux classes, nous définissons une classe abstraite appelée Tuple3f qui stocke un triplet de coordonnées réels (x,y,z) et fournit des méthodes pour lire et modifier les trois coordonnées, nous définissons alors les classes Vertex3f et Vector3f par dérivation de la classe Tuple3f .Nous ajoutons également les méthodes nécessaires aux deux classes (voir figure 4.3).

De la même manière, nous définissons les classes Tuple2f pour une paire de cordonnée (x,y) de type réel, Tuple3s et Tuple2s pour les cordonnées de type entier naturel.

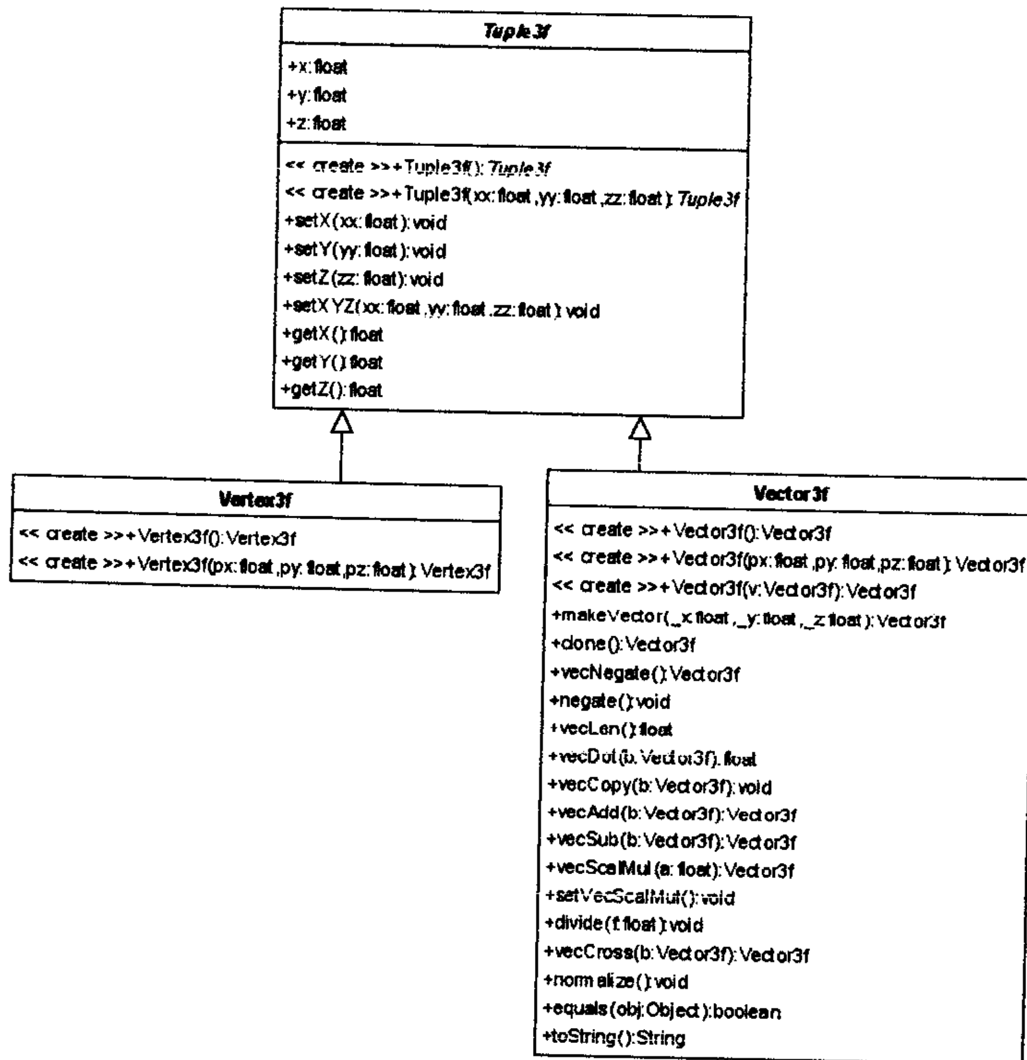


FIGURE 4.3. Diagramme de classe des triplets

4.3.3 Les modules mathématiques

Pour des raisons de simplicité dans les manipulations, on se sert souvent de la notation des matrices pour modéliser les différentes transformations et notamment les transformations des sommets d'un objet.

Nous avons définit une interface IMatrix pour modéliser une matrice de transformation qui contient les signatures des fonctions à appliquer sur une matrice de transformation (translation, rotation, produit matriciel...etc) et on définit les différents types de matrices de transformation (matrice homogène, matrice de pro-

jection, ...etc.) par une implémentation de l'interface IMatrix. L'utilisateur peut aussi implémenter les matrices de transformation à sa manière, qui est un moyen de réutilisation important.

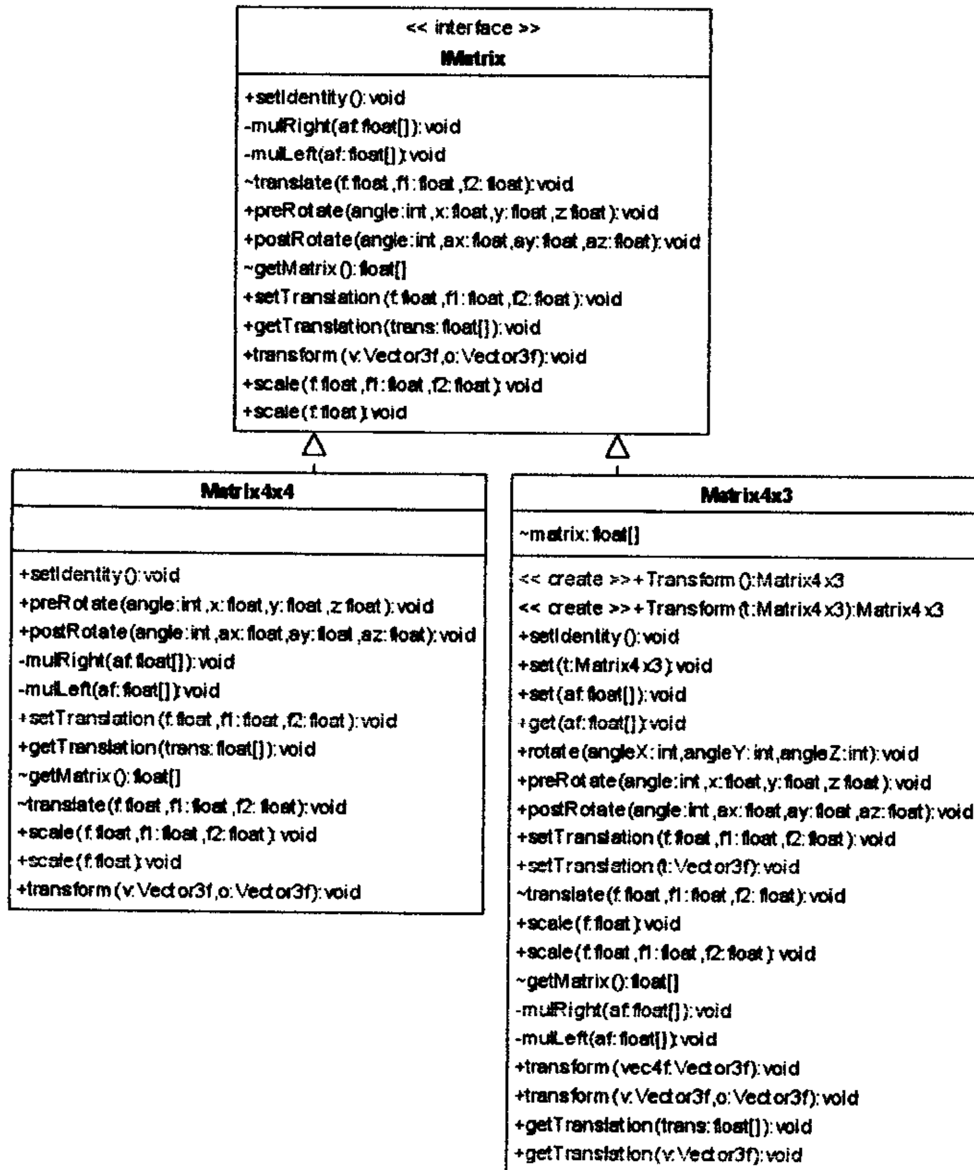


FIGURE 4.4. Diagramme de classe des matrices de transformations

Les rotations sont réalisées en partie grâce aux opérations trigonométriques (sin et cos), ce qui en terme de rapidité a un coût (plus de 20 cycles). Donc on peut y remédier en pré calculant une table de sinus et cosinus pour un angle de 0 à 360

degrés avec une résolution appropriée dans la classe Math

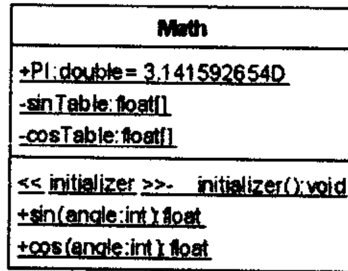


FIGURE 4.5. Diagramme de classe des opérations trigonométrques

4.3.4 Les classes de bases pour les images et le graphisme 2D

Les classes AImage et AGraphics sont des classes abstraites pour définir une image et un objet permettant de produire des graphiques en 2D respectivement.

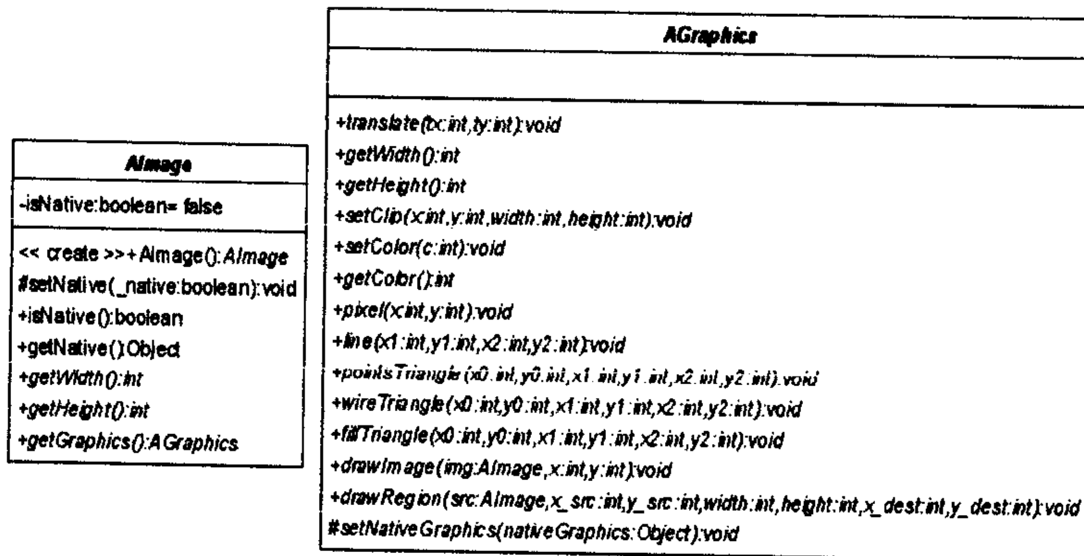


FIGURE 4.6. Diagrammes de classe pour les modules graphisme 2D et image

4.3.5 Le module du moteur 3D

Nous avons définit deux interfaces pour un noyau du moteur 3D.

L'interface I3DAPI contient toutes les signatures des fonctions nécessaires pour construire des APIs 3D, la classe qui implémente cette interface doit utiliser le pattern singleton (voir chapitre III-section III.2.3) pour assurer l'unicité de créer une seule instance de cette classe et elle doit utiliser un objet qui implémente l'interface IRenderer pour lancer le rendu dans la fenêtre d'affichage.

L'interface IRenderer contient toutes les signatures des fonctions nécessaires pour le rendu 2D (pixel, droite, triangle, texturation, ...etc.).

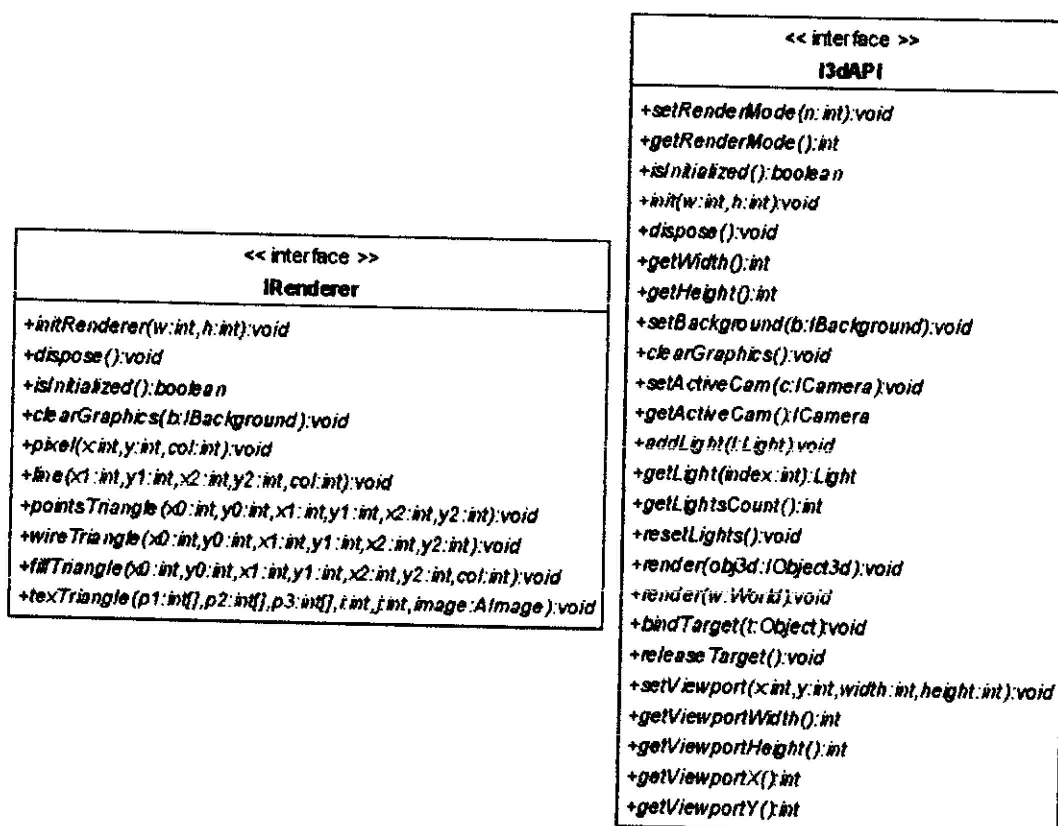


FIGURE 4.7. Diagramme de classe des interfaces I3DAPI et IRenderer

Le noyau du moteur doit être initialiser à l'aide de la fonction init(int w , int h) où w et h est la largeur et la longueur de la fenêtre d'affichage respectivement .

Le moteur peut être configurer pour qu'il change le mode rendu (point, fil de fer, facette, ...etc) à l'aide de la méthode setRenderMode(int mode).

Le moteur doit connecter la cible de rendu (objet où l'affichage est effectué) qui peut être un objet de type AGraphics, AImage ou d'autres objets implémentés par l'utilisateur du framework, la méthode bindTarget(Object target) effectue cette tâche.

Pour effacer l'écran d'affichage par une image arrière plan (couleur ou image coloré) l'utilisation de la méthode setBackground(Background b) est indispensable.

Pour le rendu il y deux possibilités :

- un rendu en mode immédiat : l'objet 3D à rendu est donné au moteur à l'aide de la fonction render(IObject3d obj) , la camera et l'arrière plan utilisées sont ajouter au moteur par les deux méthodes setBackground(IBackground b)et setActivCam(ICamera c).

- un rendu de la scène : les objets 3D à rendu sont dans une scène type World, le rendu est effectué par la méthode render(World w), la camera et l'arrière plan sont inclues dans la scène.

Pour libérer la cible de rendu, on utilise la fonction releaseTarget(), l'image finale sera dessiner dans la cible.

4.3.6 La classe Observable

L'utilisation du pattern de conception Observer (voir chapitre III-section III.2.3) nous a permit de réaliser l'architecture MVC. Pour cela, nous avons besoins de concevoir et implémenter la structure de ce pattern en ce basant sur ces principes.

La classe Observable doit avoir une référence sur une liste d'observateurs, lors d'une notification cette dernière est parcourue pour lancer la mis à jours de chaque observateur par la fonction update().

Dans notre cas nous avons implémenté notre propre liste qui est présentée dans la figure 4.8.

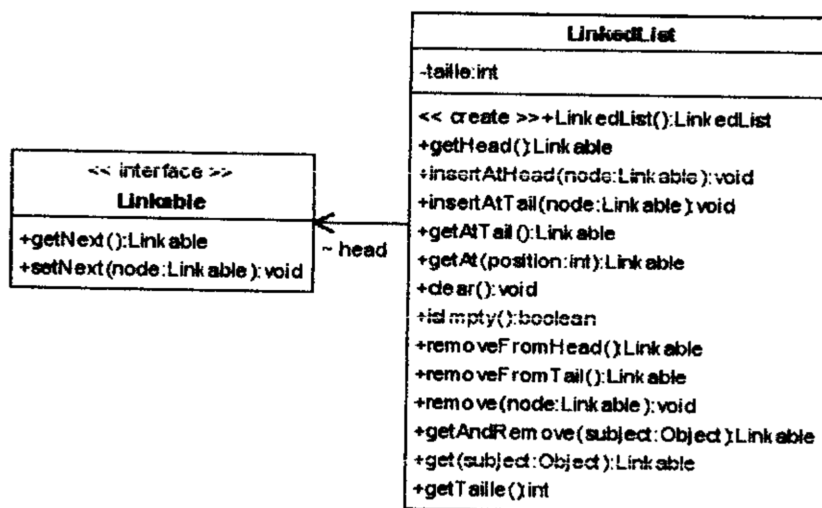


FIGURE 4.8. La structure de la liste chainée

La liste est définit dans la classe LinketList, elle possède une référence sur le premier élément de la liste (head) qui est un objet Linkable.

Un élément de la liste doit implémenter l'interface Linkable pour qu'il puisse être utilisé dans la liste linkedList.

La classe Observable hérite de la classe abstraite ObservableAbstract (voir figure 4.9) et utilise une liste Linkedlist pour définir la liste des observateurs, l'interface Observer hérite de l'interface Linkable, donc tout objet qui veut être un observateur doit implémenter les deux interfaces Observer et Linkable.

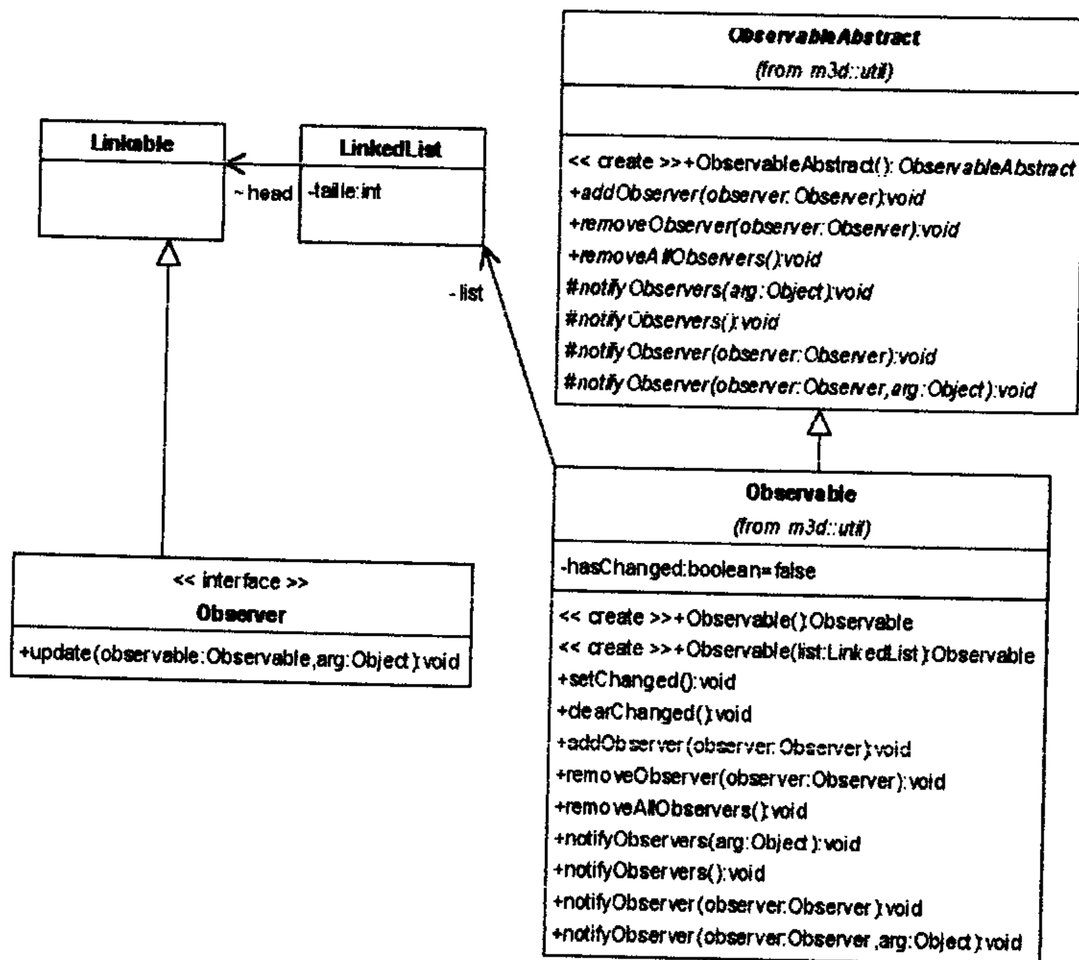


FIGURE 4.9. Diagramme de classe du pattern Observer

L'utilisation de la classe abstraite ObsrevableAbstract permet aux utilisateurs d'implémenter à leur manière la classe Observable, ils peuvent par exemple ajouter d'autre méthodes ou fonctionnalités.

4.3.7 Le module des caméras

Pour Concevoir le module de caméra, on propose d'utiliser le pattern Strategy (voir chapitre III-section III.2.3) qui nous permet de remplacer facilement la caméra utilisée par une autre, pour cela on utilise une interface Startegy qui contient tous les algorithmes et les méthodes nécessaires pour une caméra (voir figure 4.10), puis la classe Camera doit implémenter cette dernière. Lorsque on désire changer la classe

Camera, on change uniquement au niveau de l'initialisation par une autre classe qui implémente ICamera et le reste du programme reste inchangé.

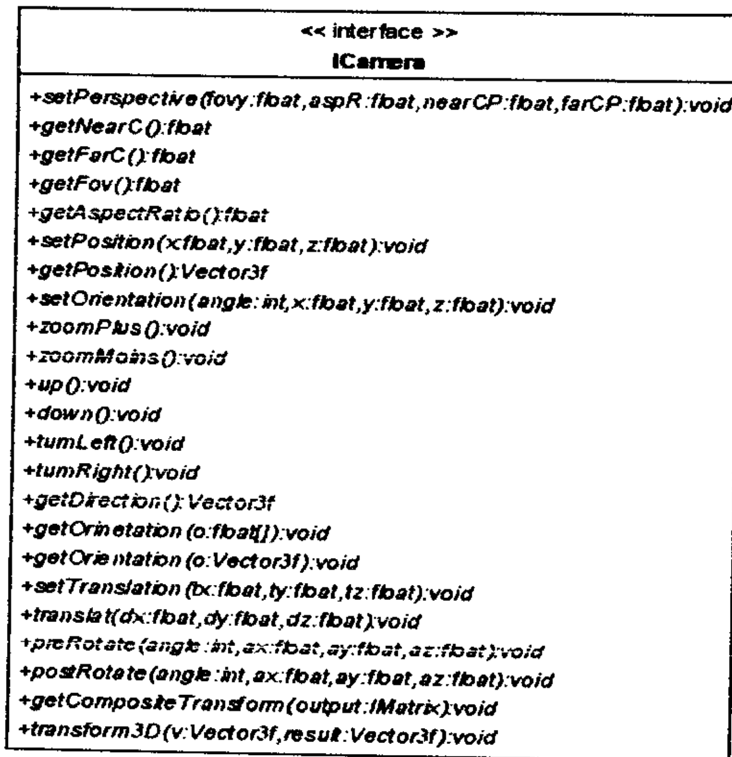


FIGURE 4.10. Diagramme de classe de l'interface ICamera

La caméra peut être utilisé dans une scène 3D ou dans un mode de rendu immédiat, dans les deux cas le noyau du moteur 3D doit avoir un référence sur la camera en cours d'utilisation pour calculer la matrice de projection.

4.3.8 La modélisation des modèles 3D simples

Un modèle 3D simple est un objet transformable et qui peut être rendu dans une image, la classe abstraite IObject3d hérite de la classe TreeLeaf (dans le cas où l'objet 3D est utilisé dans une scène 3D) est de l'interface IRenderable, elle utilise une instance qui implémente l'interface ITransformable pour définit les transformations géométriques (translation, rotation, changement d'échelle,... etc).

La définition des structures de stockages des points et des les faces de l'objet 3D

n'est pas présenter dans la classe abstrait IObject3d, l'utilisateur du framework doit implémenter ces structure à sa manières (utilisation des structures tableaux,vecteurs ou listes pour le stockage des cordonnés, faces, normales des faces,. . . etc).

4.3.9 La modélisation de la scène 3D

La question qui se poser est comment représenter la structure de la scène.

La solution qu'on a adoptée consiste à représenter la scène 3D par une structure hiérarchique d'arbre.

Cette arborescence commence par un objet de type World qui est la racine de l'arbre. Les nœuds de l'arborescence sont des objets de type Group qui utilisant un objet de types ITransformable, ces nœuds sont utilisés pour les transformations 3D (rotations, translations, changement d'échelle), ces transformations sont appliquées sur les fils d'un nœud Group.

Les feuilles de l'arbre sont des objets qui héritent de la classe TreeLeaf, ils sont des modèles 3D simple de type IObject3d ou des objets de type Light qui définissent une lumière dans la scène (voir figure 4.12).

Nous avons implémenté la structure d'arbre de manière qu'il soit réutilisable, elle est composé d'un ensemble d'interfaces et des classe qui implémentent ces interfaces (voir figure 4.11).

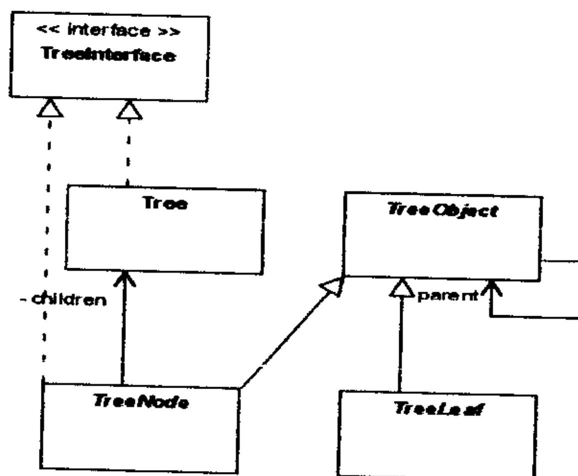


FIGURE 4.11. Diagramme de classe de la structure d'arbre

L'interface TreeInterface, définit un sommet d'un arbre.

La classe TreeObject, c'est une classe abstraite qui définit un nœud de l'arbre.

La classe Tree, l'implémentation de la structure d'arbre.

La classe TreeNode, c'est l'implémentation d'un sommet d'un arbre.

La classe TreeLeaf, c'est l'implémentation d'une feuille d'un arbre.

Pour modéliser une scène 3D, nous avons utilisé les deux patterns Composite et l'Observer (voir chapitre III-section III.2.3).

Le pattern Composite nous permet de définir la hiérarchie des classes qui compose la scène et il facilite la tâche d'ajouter des nouveaux composants dans la collection.

Le pattern Observer nous permet de gérer le rendu des composants feuilles de la hiérarchie de la scène.

On remarque dans la figure 4.12 que toutes les classes qui héritent de la classe TreeLeaf (une feuille de l'arbre) implémentent les interfaces Observer.java et IRenderable.java.

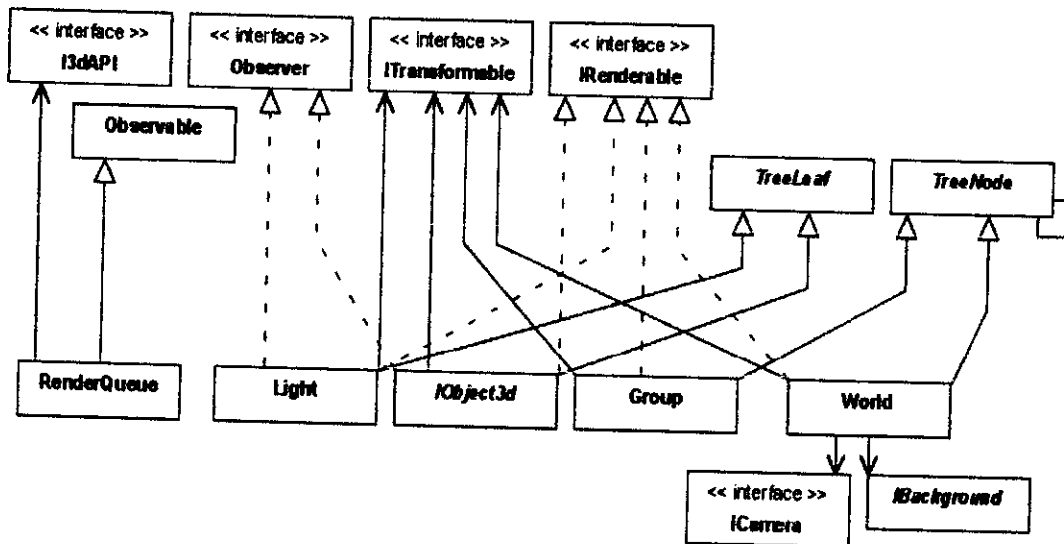


FIGURE 4.12. Diagramme de classe de la structure d'une scène 3D

Le rendu d'une scène 3D se fait par l'appel de la fonction render(World w) de la classe I3DAPI.java.

Dans la figure 4.13 nous remarquons l'utilisation du pattern Observer et le pattern Composite dans le rendu de la scène:

- l'appel de la fonction setupRender dans la racine ou dans un noeud de la scène implique l'appel de la même méthode pour touts les fils de ce nœud.

- si le fils est une feuille de l'arbre (TreeLeaf) alors il va s'ajouter à la file d'attente des objets rendables (liste des observateurs).

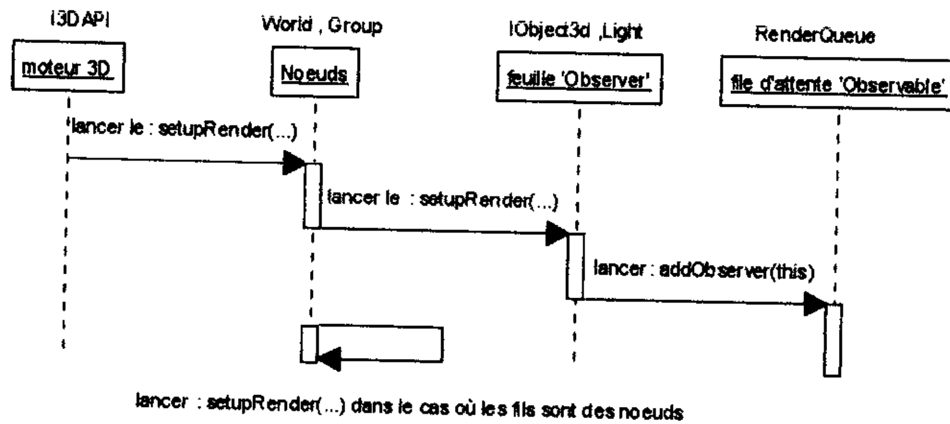


FIGURE 4.13. Diagramme de séquence de rendu d'une scène 3D

L'appel de la fonction commit() dans la file d'attente RenderQueue (Observable) lance le rendu de tout les objets insérés dans la file(Observateurs),à la fin la file sera vidée.

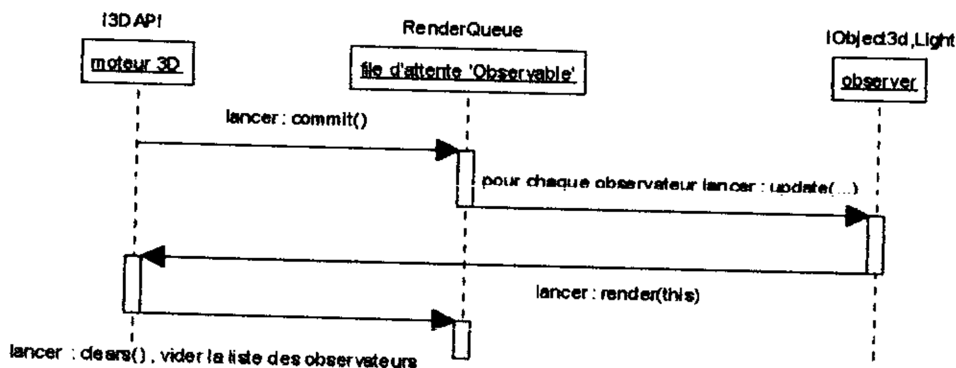


FIGURE 4.14. Diagramme de séquence des intéractions entre le moteur 3D et la file d'attente

D'après ces deux digrammes (voir figures 4.13, 4.14), on remarque que grâce au pattern MVC et Observer on optimise le calcul des matrices et on gère correctement la liste des objets de la scène 3D.

4.4 Implémentation et testes

Notre développement consiste à :

1. Mettre en place, c'est-à-dire :

- A choisir,
- A acquérir, et
- Installert

tout environnement de travail nécessaire
2. a développé deux applications

- Le framework, constitué d'un ensemble de classes écrites en langage Java.
- L'application pour tester l'utilisation du framework, qui est un moteur 3D écrit entièrement en Java sous la plateforme J2ME (CLDC 1.1, MIDP 2.0) et un ensemble de Midlets.

Environnement de travail :

- L'ensemble de travail a été réalisé, sous Windows XP.

- Les Midlets et les programmes java nécessitent la mise en place d'un environnement de développement Java (Java 2 SDK, Standard Edition, version 1.5) disponible sur le site de Sun Microsystems.

L'IDE JCreator a été installé comme éditeur de texte, ainsi que l'émulateur de téléphone portable J2ME Wirless ToolKit 2.2 pour la compilation et les testes d'exécutions.

D'autres émulateurs (Nokia, sonyErricson,... etc) sont utilisés pour tester les performances du moteurs 3D.

Réalisation de l'application :

L'implémentation du framework : qui est écrit totalement en java et il est constituer d'un ensemble de classe regroupé dans un package nommé framework. Ce dernier contient trois sous packages :

- framework.gfx3d.* : qui contient toutes les classes nécessaires pour réaliser un moteur de rendu 3D,

- framework.Tree.* : qui contient toutes les classes pour la structure d'arbre,

- framework.util.* : contient toutes les classes d'aides (LinkedList, Observer.java, Vertex3f.java...etc.) pour l'implémentation d'un squelette du moteur 3D.

Dans cette partie, nous présentons les résultats et les testes réalisés sur le moteur 3D que nous avons implémenté.

L'implémentation du moteur 3D est constituer d'un ensemble de classes Java regroupées dans un package nommé m3d.

Les deux packages framework et m3d sont compilés en utilisant l'outil J2ME wirless ToolKit 2.2 qui permet de développer des applications Java pour les terminaux compatible MIDP[WWW6], il comprend trois composants:

- un environnement de développement d'application MIDP,
- une interface utilisateur qui permet de créer, de compiler et d'exécuter une application java avec un émulateur,
- un émulateur permettant de faire tourner l'application sur différents émulés.

4.4.1 Implémentation du moteur 3D

Le noyau du moteur 3D est implémenté dans la classe Java M3dCore.java qui contient les différentes APIs pour le rendu 3D et la classe SWRenderer.java qui implémente l'interface IRenderer.java (du framework) qui contient les fonctions de rendu et d'affichage 2D. La figure suivante représente l'utilisation des interfaces du framework (I3dAPI et IRenderer) dans l'implémentation du moteur 3D.

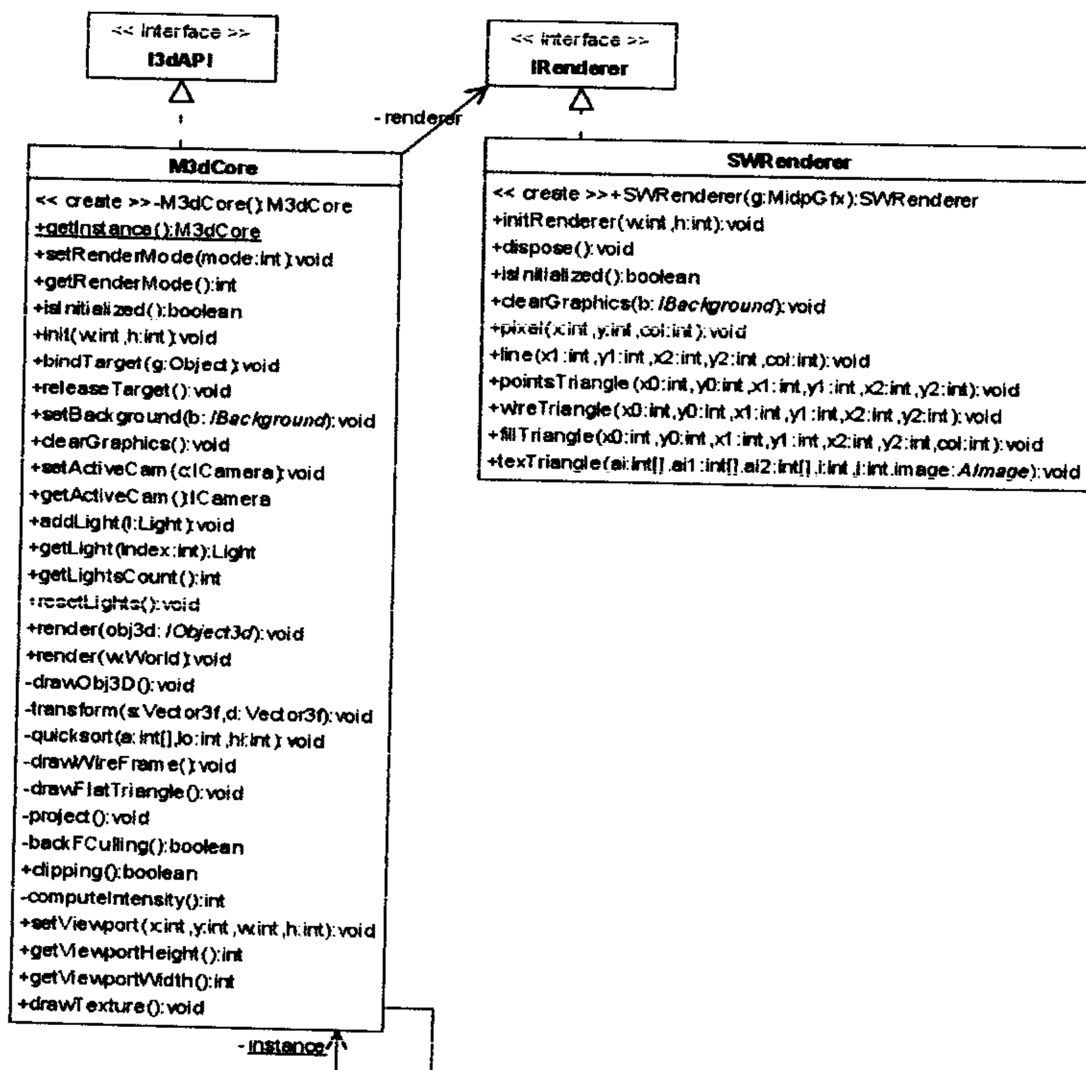


FIGURE 4.15. Diagramme de classe de gestionnaire d'API 3D

4.4.2 Implémentation des classes de graphique et les images

Pour implémenter les classes nécessaires pour le rendu (image et graphique) nous avons définit les classe abstraits du framework et utiliser les classes existantes dans le profil MIDP 2.0.

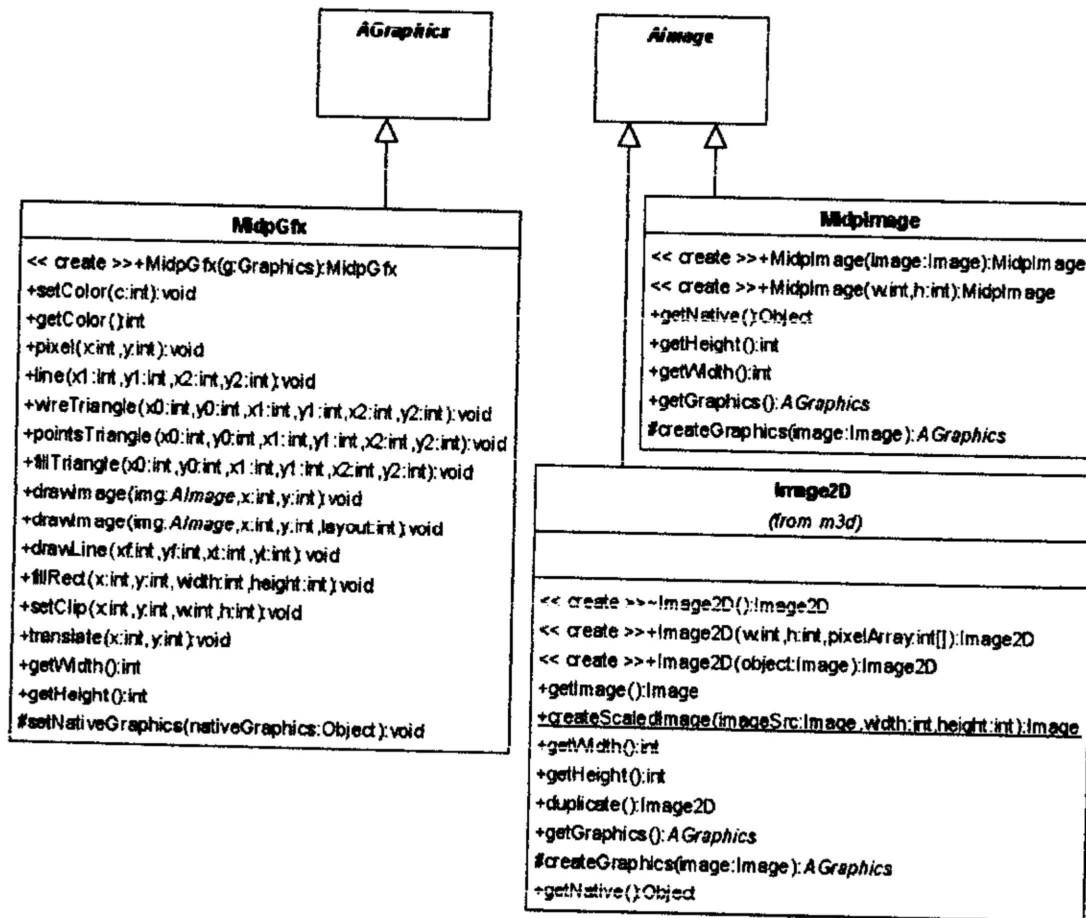


FIGURE 4.16. Diagramme de classe des outils de graphique 2D

La classe MidpImage.java définit une image 2D.

La classe Image2D.java définit une image qui peut être utilisée comme texture ou arrière plan, elle contient des fonctions pour la création d'une image à partir d'une autre avec une dimension bien précise.

La classe MidpGfx.java utilise les fonction de la classe Graphics.java du profil MIDP pour redéfinir les fonctions de base du dessin 2D.

4.4.3 L'implémentation de la caméra

La caméra est définit dans la classe Camera.java, elle implémente l'interface ICamera.java du framework et elle utilise un objet ITransformable(une instance de la classe CamTransf.java)pour définir les opérations de rotations et translations.

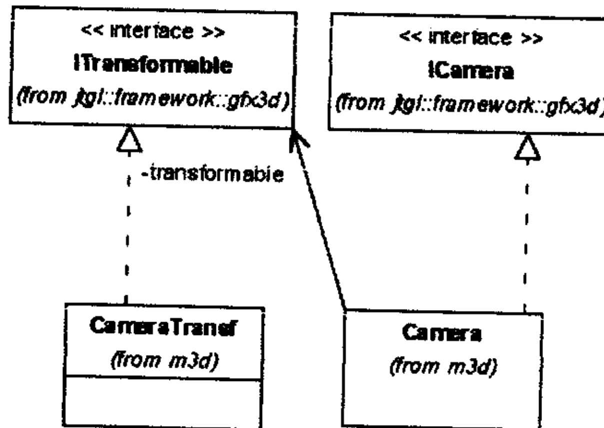


FIGURE 4.17. Diagramme de classe de module Camèra

4.4.4 L'implémentation des modèles d'illuminations

Le moteur 3D peut utilisé trois(3) types de lumières:

- la lumière ambiant : qui est implémentée dans la classe AmbientLight.java
- la lumière Omnidirectionnelles : qui est implémentée dans la classe OmniLight.java
- la lumière directionnelle : qui est implémentée dans la classe DirectLight.java

Ces trois classes utilisent la classe abstraite Light.java du framewrok et implémentent les interfaces IRenderable.java.

L'exemple suivant représente le code source de la classe AmbientLight.java.

```
package m3d;
import jtgl.framework.gfx3d.Light;
import jtgl.framework.gfx3d.Vector3f;
public class AmbientLight extends Light{
/** constructeur sans paramètres*/
public AmbientLight(){
  lighttype = Light.AMBIENT;
  color=0xFFFFFF;


}
```

```
/** constructeur , c la couleur de la lumière ambiante*/
public AmbientLight(int c){
    super(c);
    lighttype = Light.AMBIENT;
}
/**retourner la couleur de la lumière ambiante*/
public int reflectAmbient(){
 return color;
}
/**reflection de la lumière ambiante*/
public int reflectAmbient(Material material ){
    float ka = material.getKa();
    int matAmbCol =material.getColAmbient() ;
    byte r1=(byte)( Color.getRed(color)&Color.getRed(matAmbCol) );
     byte r2=(byte)( Color.getGreen(color)& Color.getGreen(matAmbCol) );
     byte r3=(byte)( Color.getBlue(color)&Color.getBlue(matAmbCol) );
     int col=Color.getColor(r1,r2,r3);
    r1 = (byte)( Color.getRed(col)*ka );
    r2 = (byte)( Color.getGreen(col)*ka );
    r3 = (byte)( Color.getBlue(col)*ka );
    col=Color.getColor(r1,r2,r3);
    return col ;
}
}
```

la classe LightFactory.java utilise le pattern AbstractFactory pour récupérer une instance de type Light avec un paramètre qui désigne le type de la lumière.

```
package m3d;
import jtgl.framework.gfx3d.Light;
public class LightFactory{
public Light getLight(int type){
    if(type == Light.AMBIENT )return new AmbientLight ();
```

```
        if(type == Light.OMNI )return new OmniLight ();
        if(type == Light.DIRECTIONAL )return new DirectLight ();
        return new AmbientLight();
  }
  }
```

4.4.5 Les testes de rendu

Dans cette partie nous présentons les tests et les résultats d'affichage du moteur 3D en utilisant les émulateurs de J2ME wirless toolKit 2.2 et Nokia Developer tools 3.0. [WWW6], [WWW7]

Le diagramme suivant résume une partie des classes implémentées pour tester le moteur 3D.

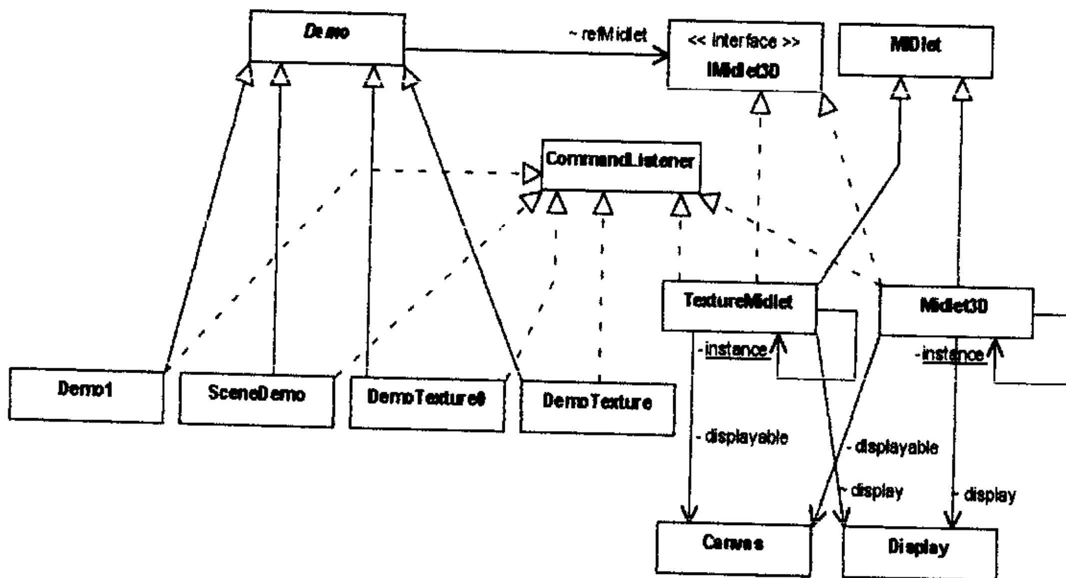


FIGURE 4.18. Diagramme de classe du programme de test

- La classe Demo.java définie une classe abstraite d'une démonstration (afficher la fenêtre, retourner au menu principal,...etc), elle a une référence sur la Midlet principale. Les classes qui héritent de cette classe définissent chaque une est un test sur une partie ou d'un composant du moteur 3D.

- Les classes TetxureMidlet.java et Midlet3D.java sont deux Midlets utilisées pourlancer les tests pour le rendu avec et sans texture des objets 3D.

Le code source de la classe Demo1.java est présenté dans l'annexe A.

Test de rendu filière :

Les figures 4.19,4.20 montrent le rendu en mode filière(*line*), la caméra est positionnée dans l'espace 3D (0,0,100).

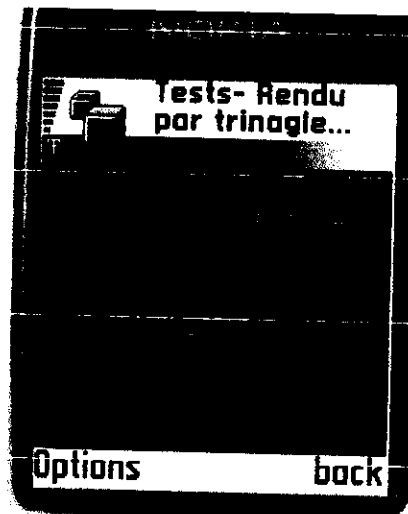


FIGURE 4.19. Modélisation filière d'un avion

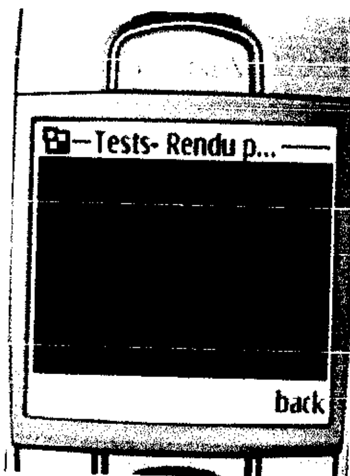


FIGURE 4.20. Modélisation filière d'un plan

Tests de rendu polygonale :

Les figures suivantes montrent le rendu en mode polygonale pour différentes formes géométriques. On a utilisé le modèle d'ombrage plat pour calculer l'intensité de la lumière.

La caméra est positionnée dans l'espace 3D (0,0,100) et dériger vers le centre (0,0,0) où les modéles 3D sont placés.

Une lumière omnidirectionnelle est placée dans l'espace 3D (0,0,50).

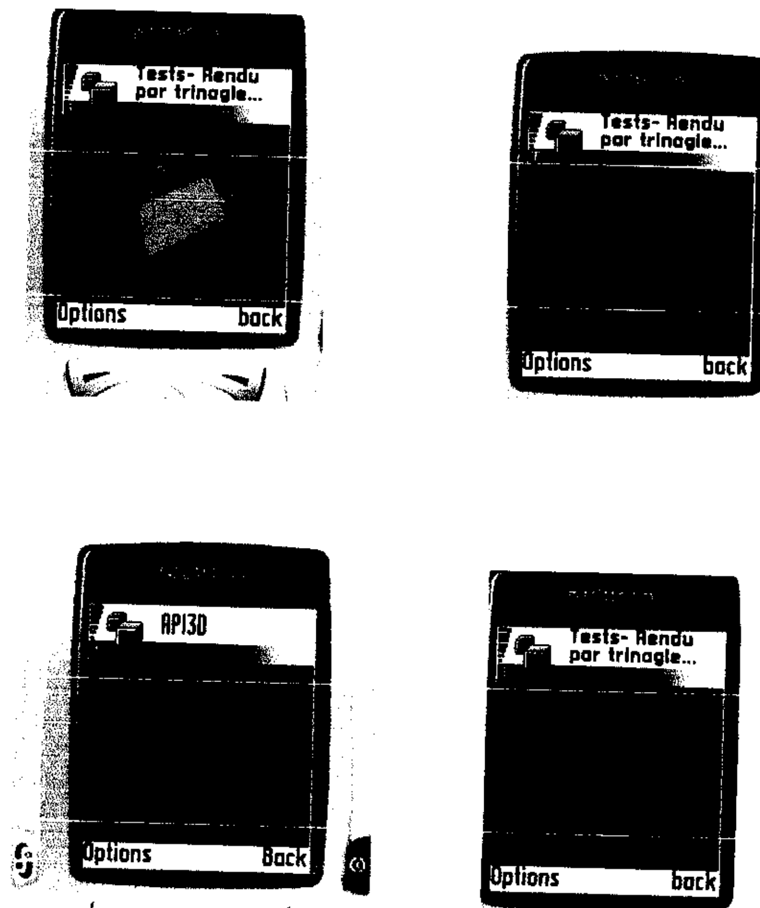


Figure 4.21. Modélisation polygonale des objets 3D simples

Tests de rendu d'une scène 3D :

Les figures suivantes représente le rendu d'une scène 3D constituée d'un ensemble de modèles 3D pour différentes formes géométriques.

La caméra est positionnée dans l'espace 3D (0,0,100) et dériger vers le centre (0,0,0) où les modéles 3D sont placés.

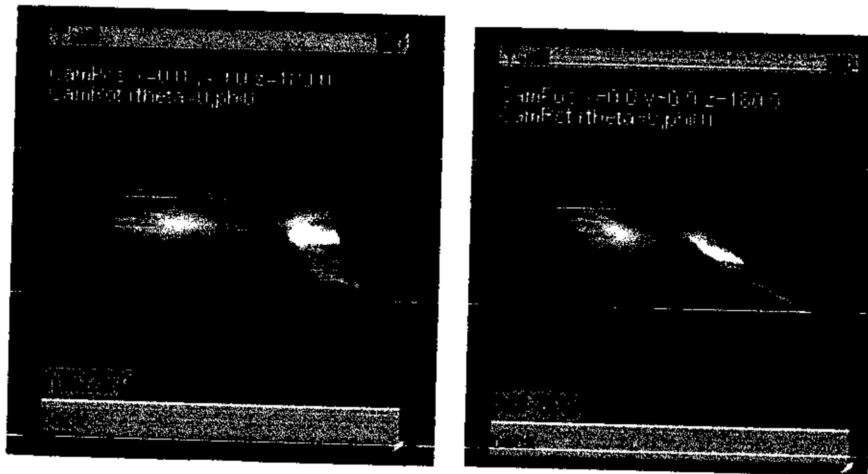
Une lumière omnidirectionnelle est placée dans l'espace 3D (0,0,50).

FIGURE 4.22. Rendu d'une scène(2 cubes et un plan)avec le modèle d'ombrage plat

FIGURE 4.23. Rendu d'une scène 3D (2 cubes et une sphère)en mode filière

Tests d'utilisation des textures :

Les deux exemples suivantes, montrent l'utilisation des textures dans le rendu, la figure a) montre un plan texturé avec une image et le résultat de rendu après une rotation. La figure b) représente un cube texturé avec une image 2D.



a).Texturation d'un plan à 4 points et rotation de plan

b). Texturation(UV) d'un cube avec une image 2D et rotation de cube

Figure 4.25. Tests de placage de texture

4.5 Conclusion

Nous avons présenté dans ce chapitre les étapes de développement d'un framework, ainsi que le moteur 3D et les résultats de rendu de quelques modèles 3D avec diverses techniques de modélisations 3D.

Dans notre travail, on constate que les patterns de conception jouent un rôle très important dans la modélisation et le développement du framework.

Les résultats de rendu sont très satisfaisants et la vitesse du moteur est également améliorée.

Conclusion générale

Conclusion générale

Le travail, que nous avons mené au cours de ce mémoire a eu pour but d'utiliser les patterns de conception et les frameworks dans la réalisation d'un moteur de rendu 3D sous J2ME. L'utilisation des patterns de conception permet la réduction de la rigidité, de la flexibilité et de la réutilisation de notre framework.

L'étude sur les patterns et les frameworks, nous a aidé à modéliser, concevoir et implémenter un framework pour les applications de graphisme 3D (moteurs 3D). L'avantage de l'utilisation du framework peuvent être décrit comme suit :

La modularité : Le framework augmente la modularité en encapsulant les détails d'exécution dans des interfaces stables. La modularité du framework améliore la qualité de logiciel en localisant l'impact des changements de conception et d'exécution. Cette localisation réduit l'effort exigé pour la compréhension et la maintenance du logiciel.

La réutilisation : Les interfaces stables fournissent par le framework augmentent la réutilisation en définissant des composants génériques qui peuvent être réutilisés pour créer de nouvelles applications.

La structure en liste chaînée que nous avons développée, nous a permis d'implémenter et utiliser facilement le pattern Observer.

L'utilisation de pattern de conception Strategy, nous a permis de tester et réaliser les différentes interfaces (IMarix , ITransformable, ICamera, ...etc).

La structure hiérarchique d'arbre de la scène nous a conduit à utiliser le pattern Composite. Il nous a permis de bien gérer la scène 3D et de communiquer facilement entre les différents nœuds lors de changement des matrices de transformation et dans le rendu de la scène 3D.

L'étude des différentes étapes du processus de visualisation 3D et la programmation graphique, nous a aidée d'implémenter les différents modules du moteur 3D (caméra, modèles 3D, scène 3D, illuminations, ...etc).

Le moteur 3D que nous avons implémenté est un moteur software (qui n'utiliser pas une accélération 3D). Il est composé d'un ensemble de classes assurant le stockage, la manipulation et le rendu de l'univers 3D à l'écran. La vitesse du moteur a

également été améliorée pour qu'il soit utilisé pour le rendu des scènes plus complexes.

L'étude de la plateforme J2ME nous a permis d'étendu nos connaissances dans le domaine de développement des applications pour les terminaux mobiles, de mieux s'adapter aux capacités de la machine,essayer d'optimiser le plus possible le code (programme) et l'allocation des ressources. Cette étude permet de mettre en pratique nos connaissances dans les divers domaines (programmation avec Java, programmation 3D, implémentation des patterns,...etc).

Perspectives

L'objectif de notre travail est atteint, on a pu : étudier et réaliser un framework pour la modélisation 3D, appliquer les concepts des patterns de conceptions dans la réalisation du framework et de tester le framework par l'implémentation d'un moteur 3D sous J2ME.

Les perspectives que nous envisageons dans le prolongement de ce travail s'articulent autour des points suivants :

- implémenter un chargeur (Loader) des scènes 3D à partir d'un fichier de format bien spécifie ou d'une base de données,
- concevoir et implémenter un module de base de données légère (RMS) afin de charger la géométrie et les propriétés du moteur 3D,
- implémenter le rendu par la technique de lancer de rayons (ray tracing),
- implémenter d'autres module d'illuminations,
- ajouter d'autres effets (L'antialiasing , SkyBox , La réflexion, Ombre... etc.),
- implémenter les formes de base qui peuvent être formées mathématiquement à partir des courbes de Bézier ou courbes splines (NURBS).
- utiliser l'accélération du matériel dans le rendu 3D (processeurs graphiques),
- utiliser les bibliothèques graphiques 3D existantes (OpenGL-ES / JSR-184) pour accélérer le calcul et le rendu,
- intégrer de nouveaux objets 3D dans la scène,
- utiliser les technologies de réseau sans fil (par exemple WLAN, GPRS et BlueTooth) pour transférer les données (fichiers contenant des informations d'une scène 3D ou des images 2D) du serveur à distance vers le dispositif mobile.

Annexe

Annexe A

Voici une parite du code de la classe Demo1.java, qui réalise le rendu 3D d'un cube :

```
import m3d.*;
import jtgl.framework.gfx3d.*;
import javax.microedition.lcdui.*;
public class Demo1 extends Demo implements CommandListener {

private Command backCommand =new Command("back", Command.BACK,0);
private boolean drawStats;
private I3dAPI engine;
private boolean stop;
public static final int rAngle = 3;
private int zTranslat;
private Object3D geo;
private Transform iTransform;
private Camera camera;
private int dxRot;
private int dyRot;
private int zoom;
private Background background;
private Light iLight;
/** nombre de sommets de cube*/
public final int geo_cube_nv =8;
/**les cordonnées (x,y,z)*/
public final short geo_cube_vv[]={
        -10 , -10, -10 ,
         10 , -10, -10 ,
        -10 , 10, -10 ,
         10 , 10, -10 ,
```

```
-10 , -10, 10 ,
 10 , -10, 10 ,
-10 , 10, 10 ,
 10 , 10 ,10 ,
};
/** les triangles (face)qui constituent le cube(p1,p2,p3) */
public final short geo_cube_vf [][]={
        { 0 ,1 ,3},{ 0 ,3 ,2},
        { 4 ,7 ,5},{ 7 ,4, 6},
        { 5 ,1, 0},{ 5 ,0, 4},
        { 1 ,7, 3},{ 1 ,5, 7},
        { 3 ,6, 2},{ 3 ,7, 6},
        { 2 ,4, 0},{ 2 ,6, 4},
};
/**constructeur*/
public Demo1(Midlet3D mid){
super(mid);
drawStats = true;
engine = M3dCore.getInstance();
if (!engine.isInitialized())engine.init(this.getWidth(),this.getHeight());
stop = false;
zTranslat = 0;
iTransform = new Transform();
dxRot = 0;
dyRot = 0;
zoom = 0;
initbackground();
initgeometry();
initCamera();
initLights();
engine.setActiveCam(camera);
```

```
engine.setRenderMode(M3dCore.FLAT);
 engine.setBackground(background);
 addCommand(backCommand);
 setCommandListener(this);
 }
 /** initialiser l'arrière plan (noir)*/
 public void initbackground(){
 background = new Background();
 background.setColor(0);
 }
 /** initialiser la camera */
 public void initCamera(){
 camera = new Camera();
 camera.setPosition(0.0f, 0.0f, 100.0f);
 camera.confCamera();
 }
 /** initialiser la géometrie à rendue*/
 public void initgeometry() {
 VArray coord = new VArray(8, 3);
 coord.set(0, geo_cube_vv, 8, 3);
 TrianglesArray triaArray = new TrianglesArray(geo_cube_vf);
 geo = new Object3D(coord, triaArray);
 geo.computeNormal();
 }
 /** initialiser la lumière */
 public void initLights(){
   iLight = new DirectLight(0x1100ff, 0.0f,0.0f,-50.0f);
   iLightAmbient.setColor(0x7700ff);
 }
```

```
/**dessiner le cube*/
public void paint(Graphics g){
long now = System.currentTimeMillis();
updateCamPosition();
engine.bindTarget(g);
engine.clearGraphics();
geo.preRotate(3, 1.0F, 1.0F,1.0F);
geo.setScale(1,1,1);
geo.setTranslation(0,0,-80);
engine.render(geo);
engine.releaseTarget();
}
/**mis a jours de la postion/Direction de la camera*/
public void updateCamPosition() {
if( dxRot == -1 )camera.turnLeft();
if( dxRot == 1 )camera.turnRight();
if( dyRot == 1 )camera.up();
if( dyRot == -1 )camera.down();
if( zoom==1 )camera.zoomPlus();
if( zoom==-1 )camera.zoomMoins();
camera.confCamera();
}
/*Traitement des touches pressées*/
protected void keyPressed(int keyCode) {
int keyAction = getGameAction(keyCode);
if(keyAction == 5)
dxRot = 1;
else if(keyAction == 2)
dxRot = -1;
else if(keyAction == 6)
dyRot = -1;
```

```
else
if(keyAction == 1)
dyRot = 1;
else if(keyCode ==Canvas.KEY_NUM1){
zoom=1;
} else  if(keyCode == Canvas.KEY_NUM3)
zoom=-1;
}
/*Traitement des touches relachées*/
protected void keyReleased(int keyCode)
{
int keyAction = getGameAction(keyCode);
if(keyAction == 1 || keyAction == 6) dyRot = 0;
if(keyAction == 2 || keyAction == 5) dxRot = 0;
if(keyCode == Canvas.KEY_NUM1 || keyCode == Canvas.KEY_NUM3)zoom=0;
if(keyAction == 8) drawStats = !drawStats;
}
/*traitement des commands ajoutées au menu*/
public void commandAction(Command c, Displayable s) {
if (c.getCommandType() == Command.BACK ){
    refMidlet.mainMenu(); //retourner au menu principal
    stop();
    }
}
```

RÉFÉRENCES

[BOI04] O. Boissie, "Design patterns ",SMA/G2I/ENS Mines Saint-Etienne, Olivier.Boissier@emse.fr, Avril 2004.

[BUS96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, & M. Stal. " Pattern-Oriented Software Architecture: A Pattern System ", 1996.

[CHR77] Christopher Alexander. "A Pattern Language", Oxford University Press, New York,1977.

[CHR79] Christopher Alexander. "The timeless way of building", Oxford University Press, New York, 1979.

[DOU04] Jean Michel Doudoux , "Développons en Java", http://perso.wanadoo.fr/jm.doudoux/java/, 2004.

[DEL02] Bruno Delb, "J2ME, Application java pour terminaux mobiles", Edition Eyrolles, ISBN 2-212-11084-7, 2002.

[DRE02] Dreamtech Software India, (www.dreamtechsoftware.com) "Wireless programming with Java", Hungry Minds, Inc, ISBN 0-7645-4885-9, 2002.

[FOW97] M. Fowler. "Analysis Patterns: Reusable Object Models ", Addison Wesley, Boston, 1997.

[GAM95] E. Gamma, R. Helm, R. Johnson, & J. Vlissides. "Design Patterns: Elements of Object-Oriented Software", Addison Wesley, Massachusetts, 1995.

[JOH93] Ralph Johnson. "How to design frameworks", October 1993.

[JOH97] Ralph E. Johnson. "Components, frameworks, patterns", ACM SIGSOFT, 1997.

[LOR93] Mark Lorenz. "Object-Oriented Software Development ", Englewood Cliffs, 1993.

[PER04] Benoît Piranda. "Synthèse d'images en temps réel ,Programmation OpenGL en C et C++", 2004.

[PRE94] Wolfgang Pree. "Meta Patterns - A Means to Capturing the Essentials of Reusable Object Oriented Design", European Conf. on Object Oriented Programming (ECOOP), 1994.

[PRE00] Wolgang Pree. A l'occasion du "workshop on methods and tools for objectorientedframework development" - oopsla. Communication personnelle, 2000.

[RAP02] Pascal Rapicault, " Modèles et techniques pour spécifier, développer et utiliser un framework : Une approche par méta modélisation ", THÈSE DE DOCTORAT, École doctorale « Sciences et Technologies de l'Information et de la Communication » de Nice Sophia-Antipolis 2002.

[SHE03] Sherif M. Yacoub, Hany H. Ammar, " Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems ",Addison Wesley, ISBN : 0-201-77640-5 , 2003.

[TOP01] Alexandre TOPOL , "VRML : étude, mise en oeuvre et applications", MÉMOIRE DIPLÔME D'INGÉNIEUR C.N.A.M, CONSERVATOIRE NATIONAL DES ARTS ET METIERS PARIS, 2001.

[PAT05] Patrick Prémont ,JAMDAT Mobile (Canada) , "3D on Cell Phones", Montreal Game Summit, November 2005.

[WWW1] www. Developpez.com

[WWW2] www.gomid.com

[WWW3] http://jcp.org/en/jsr/detail?id=184

[WWW4] http://www.khronos.org/opengles/spec.html

[WWW5] http://www.supinfo-projects.com/

[WWW6] http://www.java.sun.com/products/j2metoolkit

[WWW7] http://ww.forum.nokia.com