République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Mouloud MAMMERI de TIZI-OUZOU
Faculté de Génie Electrique et d'Informatique
Département d'Informatique

MEMOIRE

De fin d'études

En vue de l'obtention du diplôme de master en Informatique option système informatique

Thème

Réalité augmentée
Cas : interaction entre marqueurs

★★★★★★★★★★

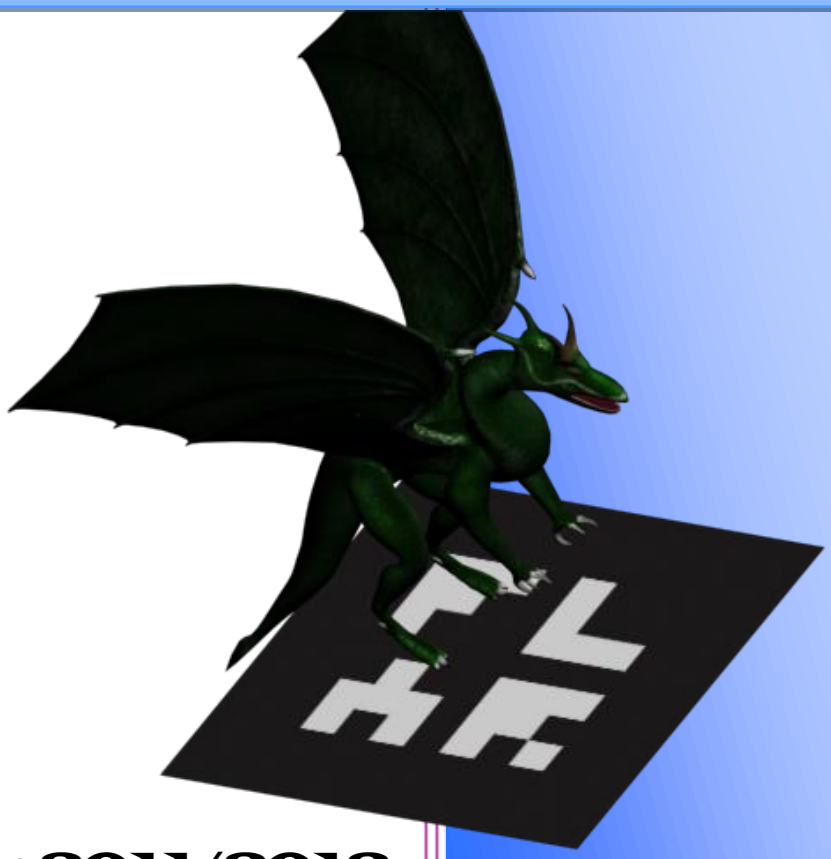
Dirigé par :

Mr KHEMLICHE S.

Réalisé par :

Mr OUAREZKI Koceïla

★★★★★★★★★★



Promotion : 2011/2012

Remerciements

Je tiens à présenter mes sincères remerciements à mon promoteur Mr KHEMLICHE pour m'avoir encadrés et guidés tout au long de mon projet et pour tous les conseils judicieux qu'il m'a prodigué.

Aussi je tiens à lui reconnaitre le temps précieux qu'il m'a consacré.

Que les membres du jury trouvent ici mes très vifs remerciements pour avoir accepté d'honorer par leur jugement mon travail, ainsi que pour le temps qu'ils ont consacré pour me donner leurs avis, corrections.

Je leurs exprimons ici mes sincères sentiments.

Enfin, merci à tous ceux qui de près ou de loin ont contribué à la réalisation de ce projet : mes enseignants, ma chères familles et mes amis(es).

Koceïla

Sommaire

Visualisation

Rendu

FLARToolKit

Papervision3D

Interaction entre marqueur

Présentation de l'application

Liste des figures

Liste des tableaux

INTRODUCTION GÉNÉRALE

La réalité augmentée (RA) vise à ajouter des éléments virtuels au monde qui nous entoure, en offrant à l'utilisateur la possibilité d'être immergé dans cet environnement mixte. Elle est loin d'être une technologie récente, mais qui aurait cru qu'elle avait déjà plus de 40 ans!

C'est en effet en 1968 qu'Ivan Sutherland[1] crée la première machine de réalité augmentée. Evidemment limitée par les technologies de l'époque, elle permet cependant six degrés de liberté, et met en œuvre un casque à vision transparente.

La première machine de réalité augmentée.

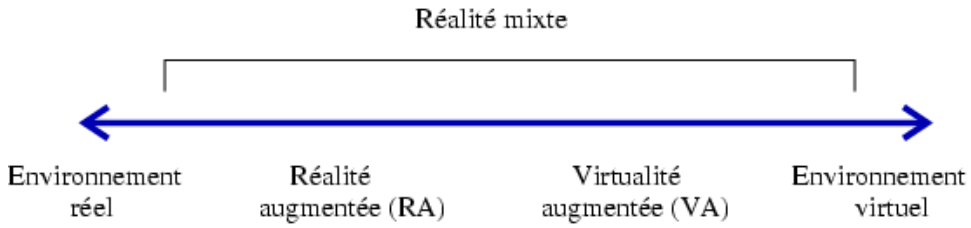
Elle fût à l'époque exclusivement réservées aux militaires et nécessitait de très importants budgets de recherche. Elle intéressa alors principalement les USA et plus particulièrement l'armée américaine. Plus récemment, dans les années 1990, des travaux de plusieurs chercheurs étendirent son champ d'application au domaine médical. Cette extension permit de faciliter les interventions chirurgicales à risque. Le but était alors de contribuer à sauver des vies et d'augmenter l'efficacité des soins prodigués.

Application militaire de la réalité augmentée.

A cette époque, le terme "réalité augmentée" n'existait même pas encore, puisqu'il a fallu attendre 1992 pour qu'il soit proposé par Tom Caudell et David Mizell, avant que sa définition précise, qui fait encore aujourd'hui référence, soit exposée par Paul Milgram et Fumio Kishino en 1994.

[1] **Ivan Edward Sutherland**, né le 16 mai 1938 à Hastings dans le Nebraska, est un ingénieur en informatique américain et un pionnier de l'internet et de l'infographie.

En effet, leurs travaux ont défini précisément les limites des différentes sous-parties de la "réalité mixte", depuis la pure réalité jusqu'à la pure virtualité, en passant par la virtualité augmentée et, bien sûr, la réalité augmentée.

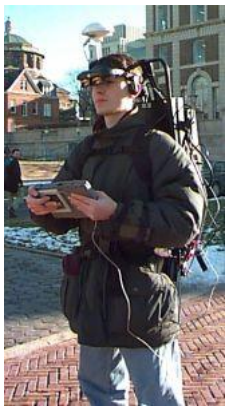


Le continuum réel / virtuel. Milgram et Kishino, université de Toronto.

Selon Ronald Azuma, chercheur à l'université de Caroline du Nord, la réalité augmentée doit respecter trois règles fondamentales : combiner le réel et le virtuel, de manière interactive (en temps réel) et en respectant l'homogénéité perspectiviste. Cette définition exclut donc les simples collages 2D qui ne respectent pas la cohérence 3D, ainsi que la composition en postproduction qui n'est pas temps réel. En revanche, elle n'impose aucune contrainte sur le réalisme photométrique des compositions (par exemple, les objets virtuels peuvent être affichés en mode filaire). La visualisation peut se faire à l'aide de lunettes stéréoscopiques particulières, qui permettent de voir la réalité en même temps que les objets synthétiques représentés en relief.

Un pas important a été franchi en 1996, avec l'introduction par Jun Rekimoto[2] des marqueurs 2D, permettant la visualisation d'objets virtuels avec six degrés de liberté. Cette technologie sera largement diffusée trois ans plus tard par Hirokazu Kato[3] et Mark Billinghurst[4], au travers de leur plateforme de développement en "logiciel libre" ARToolkit, et sert encore aujourd'hui de base à de nombreuses réalisations de réalité augmentée.

C'est en 1997 qu'a été présentée la première application de réalité augmentée en extérieur, préfigurant les applications actuelles sur smartphone, du type Layar… avec un ordinateur dans un sac à dos!



Première réalité augmentée en extérieur

[2] **Jun Rekimoto** professeur à l'université de Tokyo, Japon.
[3] **Hirokazu Kato** professeur à l'université de Nagoya, Japon.
[4] **Mark Billinghurst** professeur à l'université de Canterbury, Nouvelle-Zélande.

INTRODUCTION GÉNÉRALE

Enfin, dans les années 2000, la réalité augmentée se démocratise fortement et se fait connaître du grand public. Le développement de logiciels performants élaborés par différentes sociétés permet de l'utiliser à moindre coûts. A la fin des années 2010, la réalité augmentée a investi un très grand nombre de domaines. Elle est utilisée par des attractions comme le Futuroscope, des magazines en font la promotion et on assiste à une explosion des applications iPhones permettant de l'utiliser. Ce développement exponentiel semble se poursuivre et devrait continuer dans les années, voire dans les décennies à venir.

Dans ce mémoire je me suis intéressé à l'interaction entre marqueurs, Cette idée m'est venue en faisant des recherches sur la réalité augmentée, tout ce que j'ai trouvé c'était des tutoriaux et des vidéos sur comment afficher des modèles sur des marqueurs, pas d'interactions, pas d'animations, c'est à ce moment que je me suis posé la question « est-t-il possible de créer une interaction entre ces différents marqueurs ? ».

Afin de rependre moi-même à cette question je me suis lancé dans l'étude des différentes bibliothèques de réalité augmentée disponible sur le Net, après un certain temps de recherche j'ai décidé d'en utiliser deux, FLARToolKit et FLARManager, tout deux écrites en AS3.

J'ai divisé mon travail en trois parties :

- *Partie 1* : qui se compose des chapitres 1, 2, 3 et 4. J'explique de façon théorique ce qu'est la réalité augmentée et une scène 3D.
- *Partie 2* : qui se compose des chapitres 5, 6 et 7. J'explique les algorithmes de FLARToolkit, le moteur graphique que j'ai utilisé et le résultat de mes recherches.
- *Partie 3* : qui se compose du chapitre 8 uniquement. Je montre les deux types d'interactions que j'ai crée.

1) Introduction :

La réalité augmentée (RA) vise à mélanger le réel et le virtuel en temps réel, avec la possibilité pour l'utilisateur de s'immerger dans la scène augmentée. Je montre dans ce chapitre comment les techniques de positionnements peuvent être adaptées à la contrainte temps réel. Je fais aussi un rapide tour d'horizon des différents dispositifs de visualisations spécifiques à la RA.

2) Dispositifs de visualisation : [ROB 06]

Pour visualiser une scène augmentée en temps réel et en 3D, il est nécessaire d'utiliser un matériel dédié. Les HMD sont le plus souvent utilisés, mais l'inconfort visuel qu'ils suscitent ont incité les chercheurs à se tourner vers d'autres dispositifs.

2.1 Head mounted displays :

Les *head mounted displays* (HMD) sont des lunettes stéréoscopiques permettant de restituer le relief de la scène augmentée, en projetant deux images décalées devant l'œil gauche et l'œil droit de l'utilisateur. Il existe deux types de HMD: les HMD vidéo (Fig. 1.1) et les HMD optique (Fig. 1.2). Les HMD vidéo sont complètement opaques et utilisent deux caméras placées au niveau des yeux pour acquérir la scène réelle. Les images capturées par les caméras sont transmises à l'ordinateur qui ajoute les objets virtuels en temps réel et restitue les images composées devant chaque œil de l'utilisateur. Les HMD optiques sont munis de lentilles semi-réfléchissantes et semi-transparentes, permettant à l'utilisateur de voir réellement la scène à travers les lentilles (comme s'il était muni de lunettes classiques), en même temps que les objets virtuels dont l'image est projetés sur les lentilles.

Les HMD sont très utilisés en RA, mais possèdent plusieurs défauts qui les rendent difficilement supportables lorsque la durée d'utilisation est supérieure à quelques minutes :

- La petite taille des écrans impose une faible résolution ; pour les HMD optique, ce défaut ne concerne que la partie virtuelle de la composition puisque la partie réelle est directement observée ;
- Le champ de vision est réduit en raison des limitations du système optique ;
- La profondeur de l'image est constante, ce qui génère un inconfort visuel notamment pour les casque optique qui ne permettent pas à l'œil de se focaliser simultanément sur la scène réelle et l'image virtuelle, qui ont des profondeurs différentes ;
- Pour les HDM optique, il y a un léger délai entre l'observation du monde (instantanée) et l'apparition des images virtuelles correspondantes (émise par l'ordinateur après un certain temps de traitement) ; pour les HMD vidéo, les images réelles et virtuelles se correspondent (puisqu'elles sont envoyées simultanément par l'ordinateur), mas sont en léger décalage par rapport aux mouvements réels de l'utilisateur, ce qui provoque un mal du simulateur bien connu en RA ;
- Les HMD optiques ne permettent pas d'occulter le réel par un objet virtuel, car l'image projetée sur la lentille se superpose à l'image réelle, mais ne le remplace pas

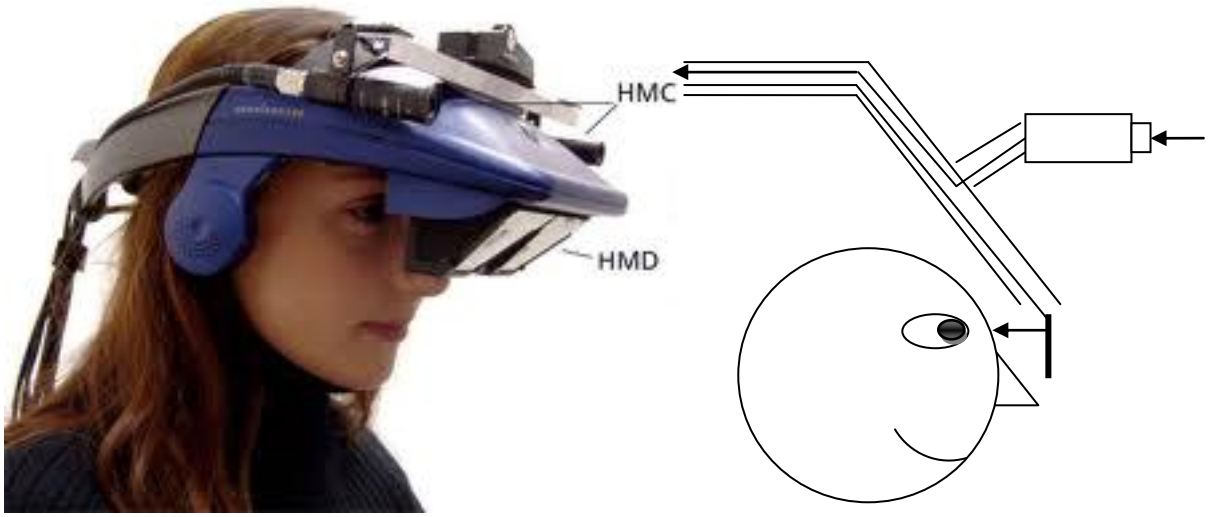


Fig. 1.1 - Schémas de principe et exemple de HMD vidéo.

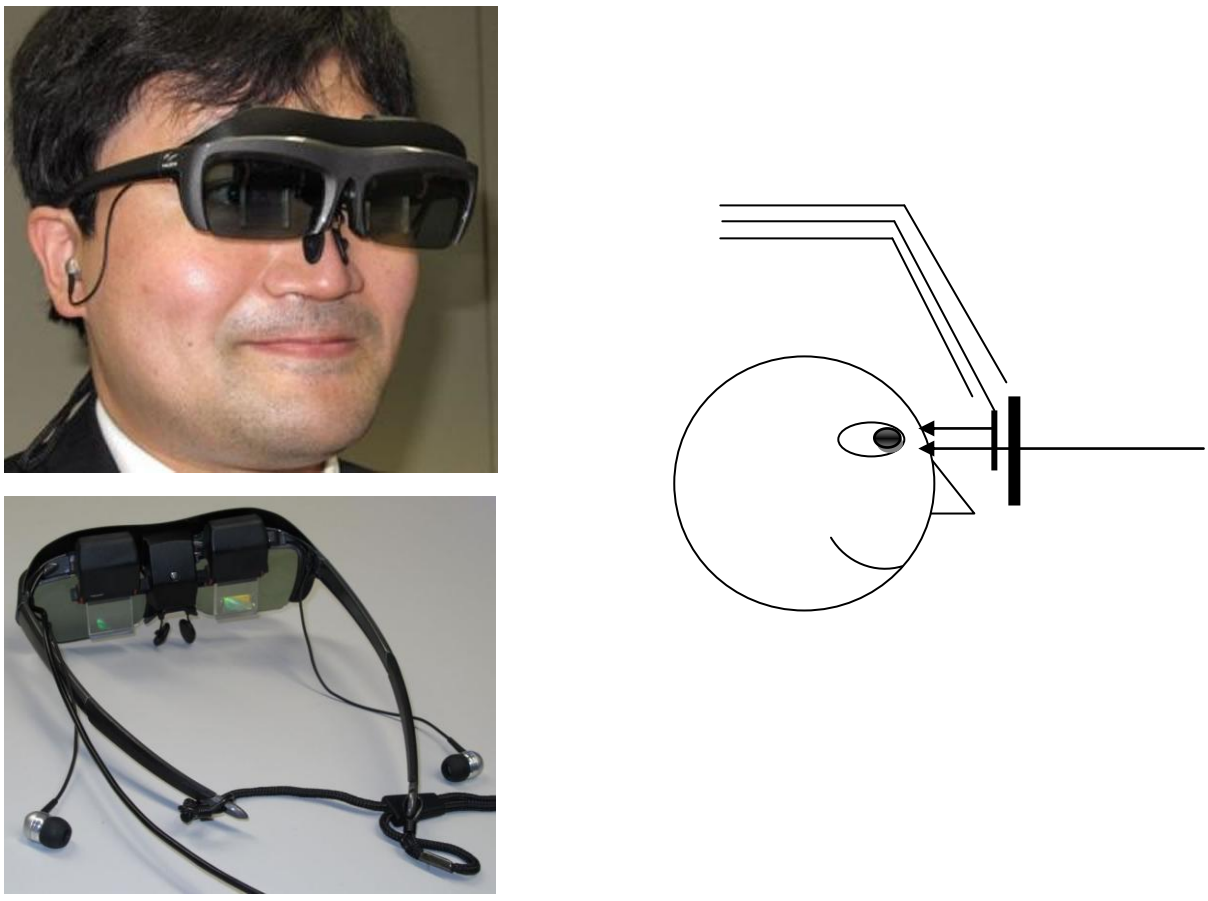


Fig. 1.2 - Schémas de principe et exemple de HMD optique.

Pour toutes ces raisons, des chercheurs ont tenté d'explorer de nouvelles possibilités pour la visualisation en temps réel d'image hybrides. Ainsi de nouveaux procédés sont apparus, dont certains sont en bonne place pour remplacer les HDM dans un avenir plus ou moins proche.

2.2 Autre dispositifs :

Bimber[1] et Raskar[2] ont proposé une classification des dispositifs de visualisation utilisable pour la RA, en fonction de la distance qui les sépare de l'observateur (Fig. 1.3). Ainsi, les auteurs ont recensé cinq types de dispositif différent, allant de la projection d'un laser sur la rétine de l'observateur à la projection de rayons lumineux directement sur les objets réels de la scène.

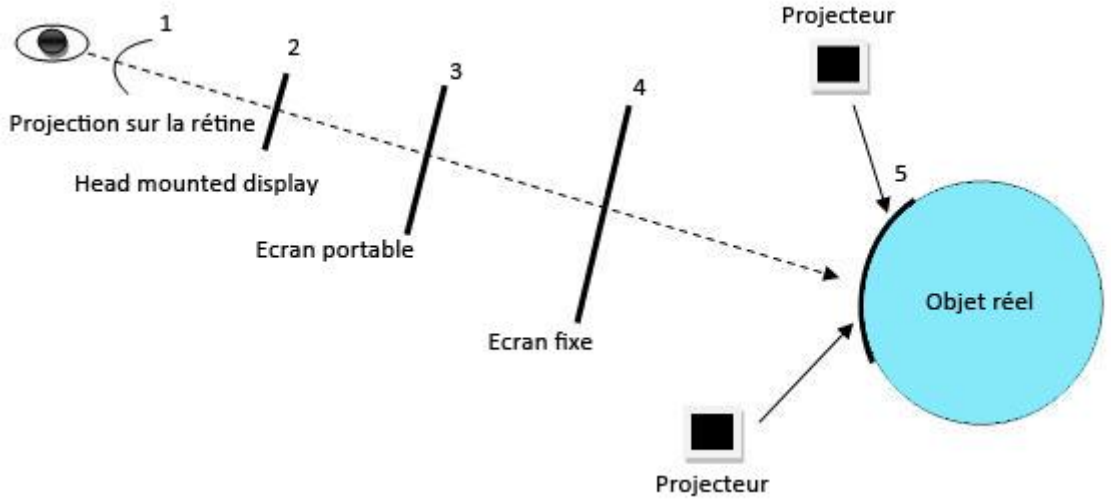


Fig. 1.3 – Différents dispositifs de visualisation pour la RA.

2.2.1 Projection sur la rétine :

Les écrans rétiniens utilisent des lasers à faible puissance pour projeter l'image virtuelle directement sur la rétine de l'œil humain. Cela permet d'obtenir des images bien plus brillantes qu'avec un HMD, ainsi qu'une résolution d'image plus élevée. Cependant, seuls les lasers monochromes (rouges) sont disponibles pour le moment, et il n'existe pas encore de version stéréoscopique pour ce type d'affichage.

2.2.2 Head mounted displays :

Ces dispositifs sont posés sur le nez de l'utilisateur, et possèdent les défauts que nous avons identifiés plus haut.

2.2.3 Écrans portables :

Cette catégorie concerne les appareils mobiles qui ont fait leur apparition ces dernières années, et qui tiennent dans la main de l'utilisateur : PDA, téléphones cellulaires, etc. Aujourd'hui, ces appareils possèdent un écran, un processeur et une mémoire qui sont suffisamment puissante pour calculer des images en basse résolution et afficher des vidéos. Munis d'une petite caméra, ils peuvent donc être utilisés pour la RA, selon le même principe que les HMD vidéo. L'avantage de cette technologie est son extrême mobilité due à la petite taille et à la légèreté du matériel. En contrepartie, la main utilisée pour tenir l'appareil ne peut pas interagir avec les éléments (réels ou virtuels) de la scène observée. D'autre part, la faible

[1] **Dr. Oliver Bimber**, directeur de l'institut d'informatique Graphique de l'Université de Linz.
[2] **Professeur Ramesh Raskar**, chercheur du MIT Media Lab aux Etats-Unis.

résolution de l'écran et la relative lenteur du processeur ne permettent pas d'envisager des compositions très réalistes. Enfin, ce type d'écran ne permet pas l'affichage en stéréo.

2.2.4 Écrans fixes :

L'écran d'ordinateur traditionnel reste finalement un support intéressant pour la RA : très répandu, pouvant être regardé pas plusieurs utilisateurs à la fois et offrant de bonne résolution d'images, il permet en outre aux observateurs d'être libres de leurs mouvements. Bien sûr, l'immersion est relative dans la mesure où la visualisation est monoscopique, et où le point de vue représenté n'est pas celui de l'observateur (en général, la caméra est fixée à l'arrière de l'écran ou déplacée à la main par l'opérateur). Toutefois, la vue subjective n'est pas forcément nécessaire pour nombre d'application de RA. D'autre part, il existe aussi des écrans optiques qui fonctionnent sur le même principe que les HMD optique (semi-transparents pour voir la scène réelle et semi-réfléchissantes ou à cristaux liquides pour afficher les objets virtuels), tout en étant monoscopiques. Les mouvements de la tête de l'observateur par rapport au système écran / scène réelle peuvent aussi être détectés, permettant d'adapter l'affichage des objets virtuels sur l'écran optique de manière à ce qu'ils se superposent bien aux objets réels du point de vue de l'utilisateur. Ce système est par exemple apprécié par les médecins, qui gardent ainsi un contrôle total sur le patient et l'environnement, sans avoir à supporter un casque inconfortable pendant toute la durée de l'intervention. Le principal défaut de ce type d'affichage est qu'il n'est pas adapté à la RA mobile. D'autre part, un seul utilisateur peut bénéficier de la vue subjective, et le problème des occultations du réel par le virtuel se pose toujours.

2.2.5 Projections :

Cette dernière technique, un peu atypique, consiste à projeter directement les éléments virtuels à insérer sur les objets solides, réellement présents dans la scène, en utilisant un ou plusieurs vidéoprojecteurs. Bien sûr, cette technique permet d'ajouter des éléments virtuels de type indication, textures, lumières, etc., sur la surface des objets présents, mais elle ne permet pas d'ajouter de nouveaux objets 3d dans la scène. Toutefois, en suivent la tête de l'observateur, on peut adapter l'image projeter à ses mouvement, on projette un modèle d'intérieur (dont les murs on été virtuellement décorées) par-dessus les murs réels de la pièce. Le revêtement mural ainsi visible ne dépend pas du point de vue de l'observateur, mais la projection du paysage situé derrière les fenêtres de la pièce doit par contre être adaptée à ses déplacements. L'image de synthèse vidéo-projetée est donc recalculée à chaque instant, en fonction des déplacements de l'observateur.

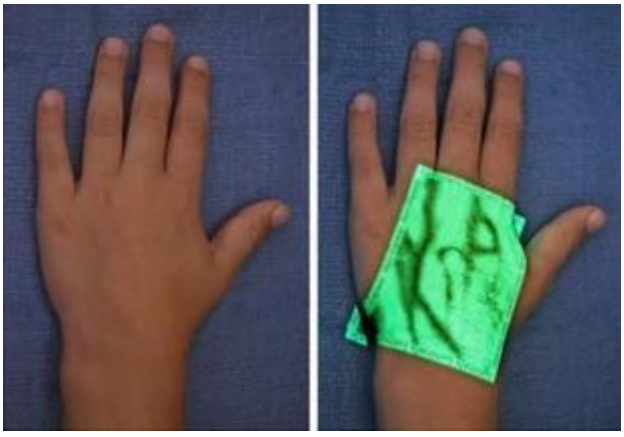


Fig. 1.4 - Projection d'élément virtuel sur une main pour faciliter une opération.

3) Positionnement temps réel :

Le premier problème à résoudre pour la RA est ce lui du positionnement, permettent d'obtenir des insertions cohérentes vis-à-vis de la perspective observée. Dans le cadre de la RA, il ne s'agit pas de retrouver le point de vue d'une camera, mais celui de l'observateur (ou plutôt de ses yeux). Pour cela, on utilise le fait que les yeux suivent le même mouvement que la tête, et on retrouve le point de vue à l'aide d'un dispositif matériel (capteur ou caméra) placé sur la tête de l'observateur. Il existe différente sore de capteurs, qui sont plutôt adapté aux applications nécessitant pas de recalage extrêmement précis ou pouvant être utilisé dons un cadre bien contrôlé. Pour les autres applications, il est préférable d'utiliser une caméra, qui permette de retrouver le point de vue à l'aide de techniques basées sur l'analyse d'image. Des méthodes hybride couplant données image et données capteurs sont aussi de plus en plus utilisées, car elles permettent d'obtenir des résulta précis tous en étant robuste. Ces différents procédés sont détaillés à présent.

3.1 Solutions basées capteurs : [ROB 06]

Diverses solution basées capteur existent pour le positionnement temps réel de l'observateur. Chaque solution possède ses avantages et ses inconvénients en termes de liberté de mouvement de l'utilisateur et de précision des points de vue obtenus.

3.1.1 Capteur magnétiques :

Ce type de positionnement fonctionne selon le principe de l'émission-réception : trois bobines orthogonales émettent un champ magnétique, capturé par trois bobines orthogonales réceptrice place sur l'objet à suivre : cela permet de récupéré à la fois la position et l'orientation du récepteur par rapport à l'émetteur. L'intérêt de cette méthode est qu'elle offre une assez bonne précision (jusqu'à 0,2 mm pour la position et 0,1 degrés pour la rotation), mais elle est très sensible aux perturbations du champ magnétique causées par les objets métalliques éventuellement présents dans la scène. D'autre par, le volume dans lequel elle opère est limité (de l'ordre de 3^3 mètres). Elle n'est donc applicable que dans des environnements restreints et parfaitement contrôlés.

3.1.2 Capteur acoustique :

Les capteurs acoustiques (ultrasonores) sont une alternative aux capteurs magnétiques, non sensible aux distorsions causées par les objets métalliques et de précision équivalente, mais tout autant limités en termes d'espace couvert. Le principe est le suivent : lorsqu'un haut parleur émet un son au temps *0*, un microphone placer a une distance *d* du haut-parleur va enregistrer le son au temps *t*, tel que *d* est égal à *t* multiplier par la vitesse du son. Connaissent le temps *t* (on est capable de détecter très précisément à quel moment le son est enregistré par le microphone), on peut donc en déduire la distance *d* entre le microphone et le haut-parleur. Ceci ne nous donne pas l'emplacement du haut parleur par à pore au microphone, mais nous indique que le haut-parleur est sur une sphère de rayon *d*, centrée autour du microphone. Pour connaître la position exacte du haut-parleur, il faut utiliser trois microphones, et intersecter les trois sphères obtenues.

3.1.3 Capteur optique :

Les capteurs optiques ne sont autres que des caméras stéréo, à l'aide desquelles les positions 3D de la cible particulières peuvent être calculées (en utilisent la technique de la triangulation). La principale difficulté réside dans la détection des cibles, qui peuvent être passive ou actives. Les cibles passives sont simplement des sphères de couleur uniforme,

souvent blanche, qui sont détectées par des techniques d'analyse d'image. Les cible actives sont aussi détectées par analyse d'image, mais elles émettent une lumière qui peut être contrôlé par la caméra, ce qui facilite l'identification de a cible. Ces capteurs sont très précis, mais encor une fois limité à un volume restreint, du même ordre que les capteurs précédents. D'autre par ils sont sensible aux occultations de cible pouvant ce produire ponctuellement, par exemple lorsque l'utilisateur passe involontairement la main devant une cible.

3.1.4 Capteur de positionnement global :

Le GPS est un système de navigation par satellite, imaginé par les militaires américains du DoD (Department of Defence), qui contrôle et finance entièrement le projet. Le principe et identique à celui des capteurs acoustique : un récepteur GPS mesure l'intervalle de temps séparant les émissions des satellites, et la réception des signaux satellites. Ces mesures sont représentatives de la distance du porteur à chaque satellite, et une intersection des sphères permet d'obtenir la position exacte du récepteur. La précision du GPS n'est malheureusement que d'une dizaine de mètre, ce qui le rend difficilement exploitable par la RA. Cependant, il peut être utilisé pour certaine application en environnement extérieur, où les éléments virtuels sont insérés à des positions très éloignée de la caméra.

Fig. 1.5 - Tripwolf application GPS en RA pour iPhone.

3.2 Solution basées vision : [ROB 06]

En résumé, les principaux défauts des capteurs physiques de mouvement sont leur manque de précision et/ou leur faible portée. À l'inverse, les méthodes de positionnement basées vision peuvent être utilisées dans des environnements quelconques, pour peut que ces environnements soient suffisamment texturés (des point d'intérêt doive être détectés). Ces méthodes sont aussi généralement plus précise que les méthodes basées capteur, car elles reposent sur une analyse directe des images à augmenter. Ainsi, en permettent une projection fidèle des points 3D correspondant aux points d'intérêt suivis dans les images, elles garantissent en principe une projection cohérente des points 3D virtuels situés dans le voisinage des points réels.

Les méthodes que nous présentons sont optimales pour une visualisation de type vidéo, c'est-à-dire pour laquelle l'image augmentée est exactement celle à partir de la quelle on calcule le point de vue. Pour les systèmes optiques, une caméra peut être utilisée pour le positionnement, mais la réalité est observée directement : la précision du résultat dépend alors aussi de la précision du positionnement, mais la réalité est observé directement : la précision du résulta dépend alors aussi de la précision du positionnement de l'œil par rapport à la caméra.

3.2.1 Marqueur artificiel :

Un moyen sûr de repérer des points 3D dans une image est d'utiliser des marqueurs artificiels. Les marqueurs sont des feuilles de papier blanc sur lesquelles sont imprimés des motifs uniques, facilement repérable à l'aide de diverses techniques d'analyse d'image pouvant être exécutées en temps réel.

Une bibliothèque *open source* nommée ARToolkit, disponible sur internet, permet aux utilisateurs familiarisés avec la programmation en C, de créer rapidement des programmes reconnaissant certains types de marqueurs, une version flash existe elle ce nomme flARToolkit. Les marqueur sont conçus par l'utilisateur et appris par le système à l'aide d'un logiciel distribué avec ARToolkit. La bibliothèque offre aussi des fonctions permettent de calculer le point de vue de la caméra à partir des marqueurs détectées, et d'insérer des objets virtuel en temps réel. Cette bibliothèque est utilisée dans nombre d'application de RA, et un workshop international lui est même entièrement consacré.

Cette solution possède néanmoins quelque handicap :

- Les marqueurs sont constamment visible, ce qui dénature la scène originale ;
- Lorsque plusieurs marqueur sont utilisés, il faut connaître leur position relative, ce qui implique un travail fastidieux de prises de mesure dans la scène ;
- Une conséquence des deux points précédents est qu'un nombre réduit d'indices est généralement disponible, ce qui ne permet pas d'obtenir des points de vue très précis ;
- Les marqueurs doivent toujours être visible, ce qui limite le champ d'action de l'utilisateur et rend le système vulnérable aux occultations ponctuelles des marqueurs.

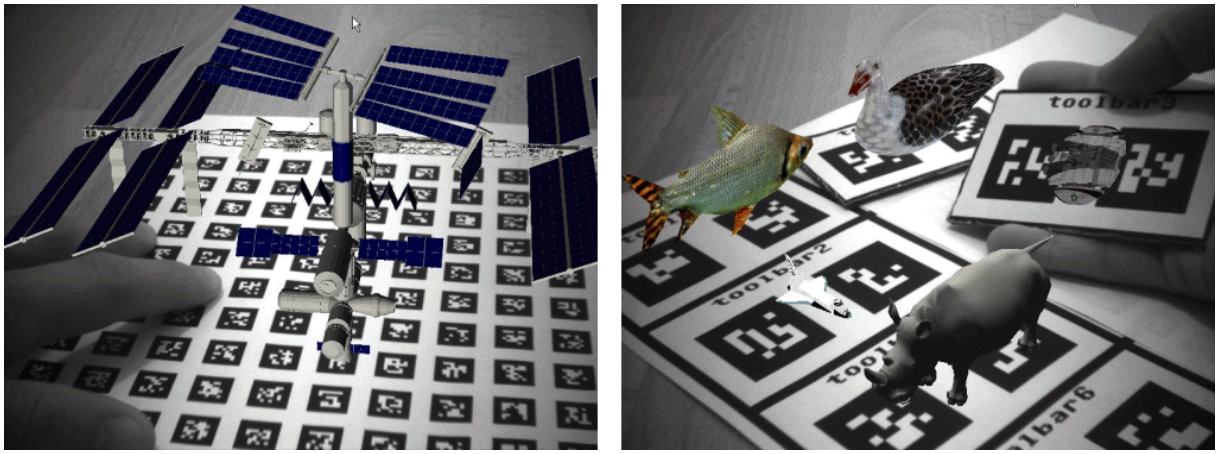


Fig. 1.6 – Exemple marqueur artificiel.

3.2.2 Indices naturels :

Pour pallier ces problèmes, les systèmes reposant sur les indices naturels de la scène (point d'encrage naturellement présents dans la scène : par exemple sûr la Fig. 1.6a où les point d'encrage sont affiché par des point) semblent tout indiqués. Malheureusement, ce type d'indices est difficile à suivre de façon entièrement automatique, des point peuvent disparaître du champ de vision, être occultés ou changer d'apparence au cours du temps.

Ce domaine est donc encore à l'étude dans les laboratoires d'informatique. Un suivie plus fiable peut être obtenue en tenant compte de contrainte particulières sur les point suivis. Par exemple sur des scènes comprenant un ou plusieurs plans texturés. Ce type de scène est relativement fréquent en environnement intérieur ou urbain.

Fig. 1.7a – Affichage des indices naturels

Fig. 1.7b – Affichage en RA après traitement des indices naturels

3.3 Solution hybrides : [ROB 06]

Finalement, les concepteurs d'application de RA se tournent de plus en plus vers des solutions hybrides, qui consistent à fusionner des capteurs avec des données image. L'intérêt est de bénéficier d'une part de la robustesse des systèmes basés capteur, et d'autre part de la précision des systèmes basés image. La fusion ce fait à l'aide de filtres particuliers, ou en utilisent les données capteur pour prédire la position des indice dans l'image.

3.4 Interface mains libres : [WEB4]

Jusqu'à présent, les démonstrations de réalité augmentée se basaient systématiquement sur un marqueur (un code ou un pictogramme à imprimer sur une feuille de papier) reconnu par l'application pour faire apparaître en surimpression un objet virtuel.

Mais si on veut un jour pouvoir utiliser la réalité augmentée comme interface homme machine dans la rue ou même simplement pour manipuler aisément des objets virtuels sans contraintes, il va falloir pouvoir s'affranchir de ces marqueurs et permettre aux caméras de reconnaître les objets et de pouvoir se positionner directement dans l'environnement existant.

Cela peut paraitre anodin, mais le traitement en temps réel des images vidéos nécessaire pour accomplir cette prouesse ouvre la voie à de nombreuses autres innovations à venir !

C'est l'objet de la recherche menée par Taehee Lee[3] et Tobias Höllerer[4] du "Four Eyes Lab" de l'Université de Santa Barbara en Californie.

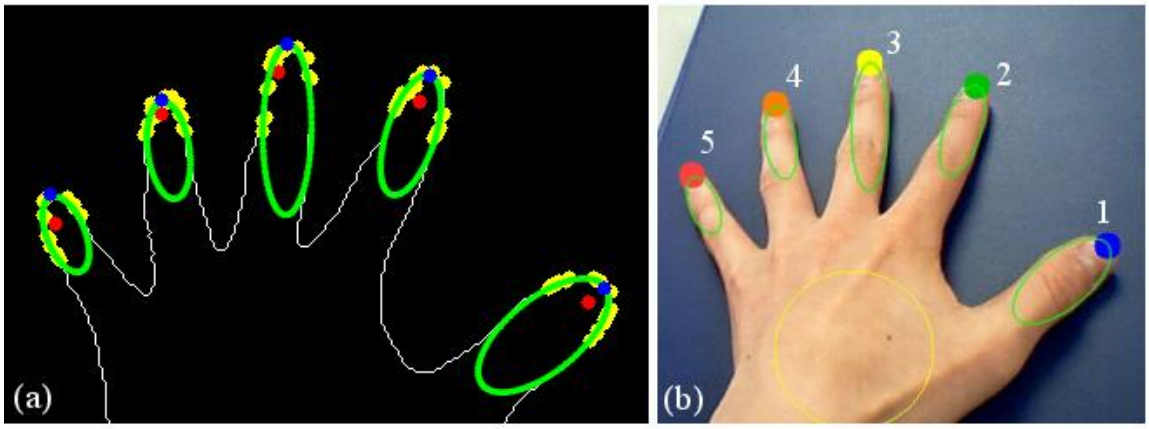


Fig. 1.8 – La détection se base sûr les courbures du contour de la main.

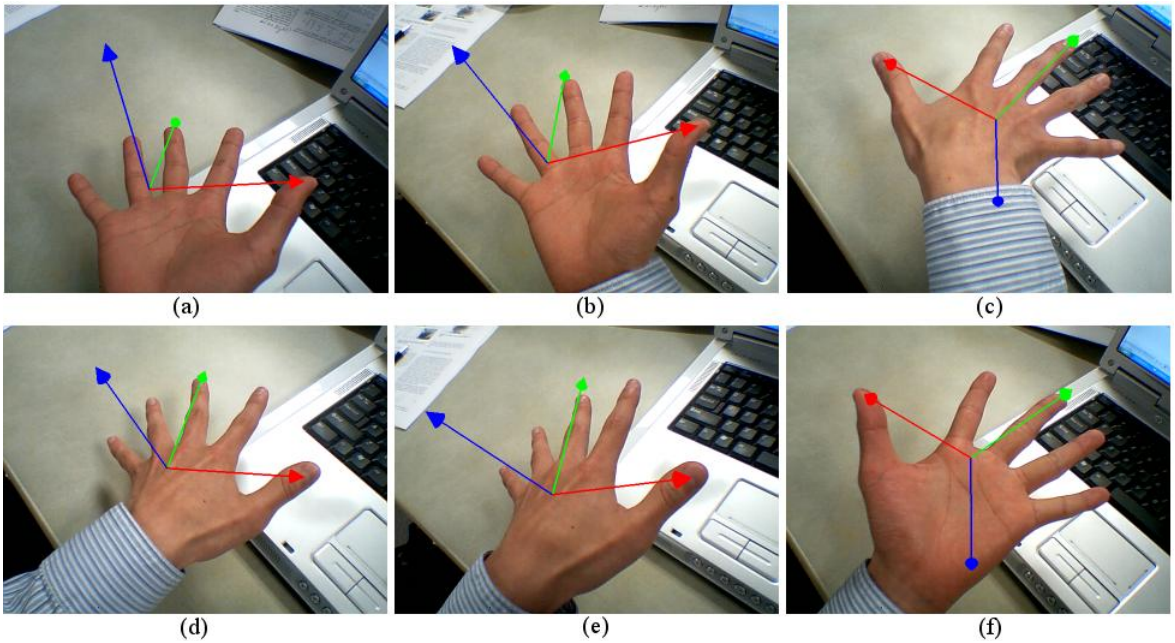


Fig. 1.9 – Exemple du suivi de la position de la main.

4) Conclusion :

La réalité augmentée est l'adaptation de l'intégration 3D à la contrainte temps réel. Différents dispositifs de visualisation existent, mais aucun n'est encore entièrement satisfaisant en terme de confort optique.

Quand on parle de réalité augmentée, on parle souvent d'objets 3D. C'est ce dont nous parlerons dans le prochain chapitre, les différentes techniques de modélisation.

[3] **Taehee Lee** est un candidat au doctorat en Vision Lab de Computer Science Department à l'UCLA (*Université de Los Angeles en Californie*). Auparavant, il a reçu sa maîtrise en sciences informatiques à l'UCSB conseillé par le professeur Tobias Höllerer, avec des projets sur la réalité augmentée.

[4] **Professeur Tobias Höllerer,** chercheur de l'UCSB (*Université de Santa Barbara en Californie*).

1) Introduction :

Dans le processus qui conduit à la réalisation d'une scène 3D, la première étape est la modélisation, elle consiste à créer des objets 3D, simples ou complexes, dans la mémoire de l'ordinateur. Ces données sont ensuite structurées pour permettre leur exploitation efficace par les algorithmes de visualisation et de rendu.

One peut assimiler le travail du modeleur au travail d'un artiste sculpteur.

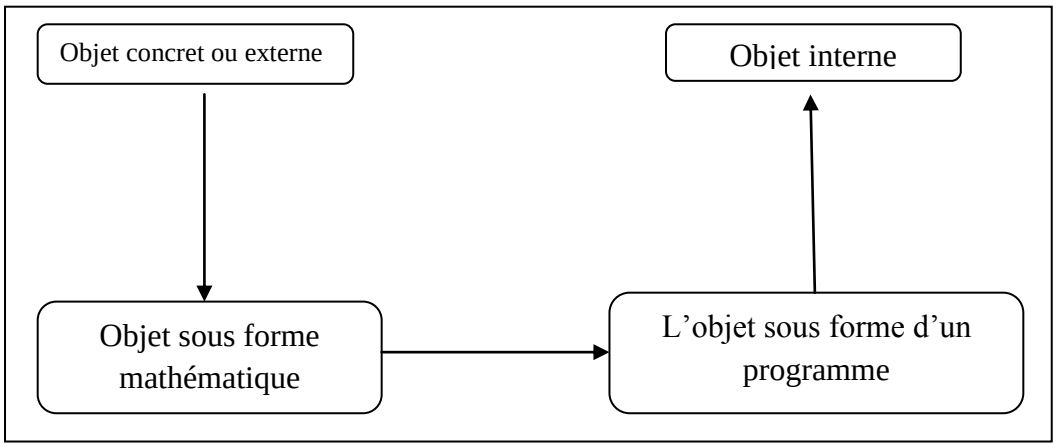


Fig. 2.1 - passage d'un objet concret à sa représentation interne

On distingue deux types de modélisation :

- La modélisation géométrique.
- La modélisation d'objets naturels.

2) La modélisation géométrique :

La modélisation géométrique est l'ensemble des outils mathématiques, numériques et informatiques qui combinés permettent de construire un modèle virtuel (ou modèle informatique) d'un objet réel. Cet objet peut être plus ou moins complexe, plus ou moins schématisé. Il peut être le fruit de l'imagination, d'une tendance ou plutôt une solution plus ou moins exacte d'un problème physique donné, voire un compromis entre les deux.

La modélisation géométrique sous-entend d'être en mesure de réaliser la construction et l'assemblage de formes élémentaires pour créer des objets de plus en plus complexes en respectant des contraintes topologiques.

La modélisation géométrique comporte différentes variantes, on cite :

2.1 La modélisation en fil de fer : [LOO 92]

C'est le plus ancien des modèles. Les premiers logiciels d'aide à la conception mécanique dans les années 70 chargeaient les tables de dessin, ainsi une pièce était représentée par la liste des segments qui permettaient de la dessiner au trait.

Cette modélisation conserve les coordonnées des sommets et des arêtes joignant ces sommets, donc l'objet est mémorisé sous la forme de segments (représentation filaire ou squelettique).

Points forts :

- Puissance de calcul minimal.
- Peu de place dans la mémoire de l'ordinateur

Points faibles :

- Impossible de distinguer les parties pleines de celles qui sont vides.
- Difficulté d'éliminer les parties cachées.
- Impossible de calculer la masse et le volume.

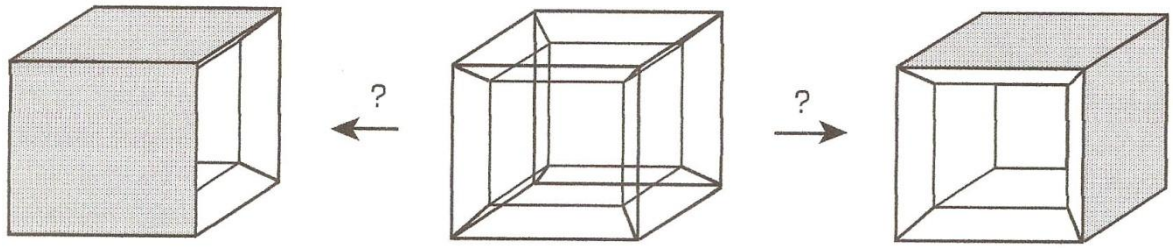


Fig. 2.2 - Ambiguïté du modèle filaire

2.2 La modélisation surfacique :

Les inconvénients du modèle fil de fer ont conduit à l'approche surfacique, l'idée de base de cette modélisation est que tout objet en volume peut être décrit par un ensemble de surfaces le constituant (son aspect externe). Ce modèle permet la définition des surfaces par fois très complexes, mais n'autorise pas la vérification automatique de la cohérence des objets créés car chaque surface de l'objet est décrite indépendamment, et aussi l'intérieur de l'objet représenté reste inconnu.

On trouve 2 types de représentation surfacique :

2.2.1 Représentation par des surfaces planes (polygonales) : [LOO 92]

C'est le modèle le plus populaire et il constitue la principale représentation interne utilisée par de nombreux logiciels.

L'idée est que chaque surface est bordée par un ensemble de segments de droites (fig. 1.2), et pour avoir une bonne approximation de l'objet à représenter, les surfaces le constituant doivent être découpé en facettes planes suffisamment petites, de ce fait nous pouvons approximer de façon très satisfaisante une sphère par un maillage d'un grand nombre de polygones (fig. 1.3).

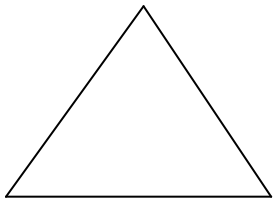


Fig. 2.3 - Un polygone

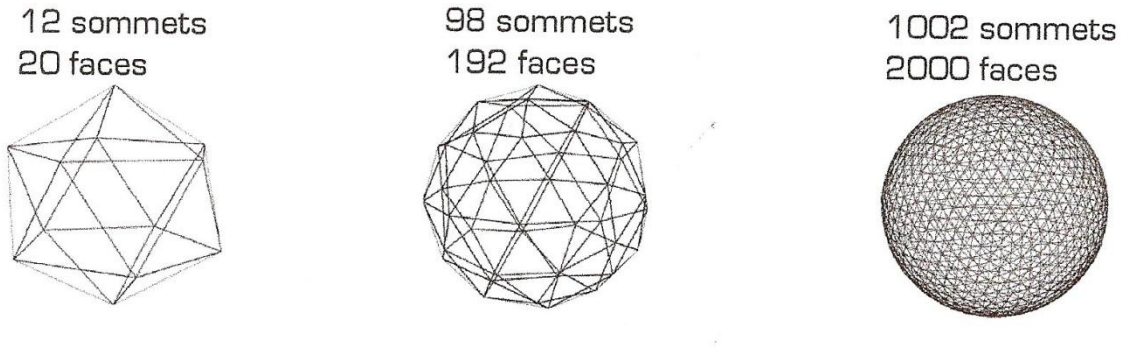


Fig. 2.4 - Approximation d'une sphère par maillage polygonale

Points forts :

- Possibilité de traiter les sommets un à un ce qui permet de sculpter les modèle et de leur donner la forme voulue.
- Intégration de fonctions de rotation, redimensionnement et de duplication qui peuvent agir sur l'arête, la face ou l'objet tout entier.

Point faible :

- L'approximation des objets arrondis coûte cher en termes d'espace mémoire sans pour autant donner des résultats satisfaisants.

2.2.2 Représentation par des surfaces courbes : [FOL 01]

Elle consiste à approcher les surfaces d'un objet par un ensemble de famille de surfaces connues. Cette représentation est utilisée pour modéliser des objets complexes à contours lisses.

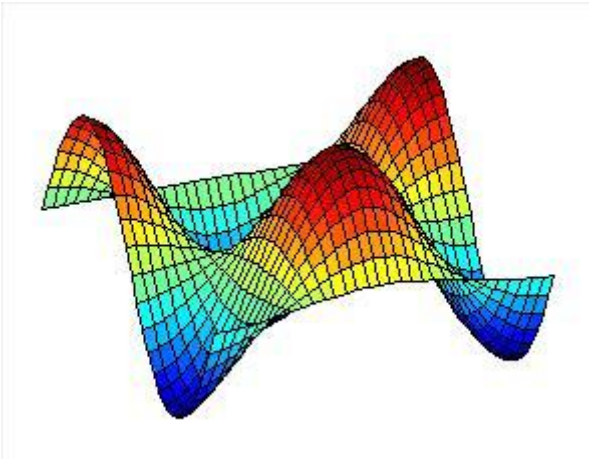


Fig. 2.5 - Surface à contours lisses

2.3 Modélisation volumique :

La modélisation volumique appelée parfois modélisation du solide utilise la notion de volume, l'objet est défini alors par son intérieur et son extérieur ce qui résout les problèmes soulevés dans la modélisation surfacique, et qui introduit aussi la possibilité de calculer les attributs physiques des objets 3D. Différents schémas de représentation des solides ont été élaborés dans cette modélisation :

2.3.1 Représentation générée par balayage : [PER 97]

Le principe est de faire un balayage d'un élément géométrique (arête/face) par une fonction d'animation (translation ou rotation) pour l'obtention d'un volume. C'est une modélisation non ambiguë et facile à créer.

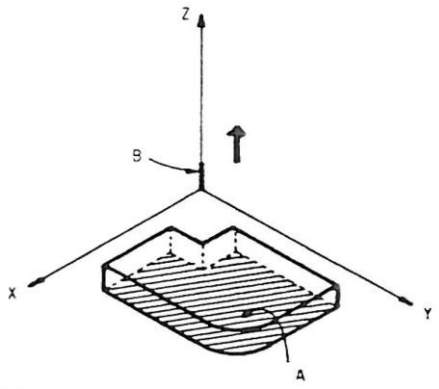


Fig. 2.6a - Génération par translation

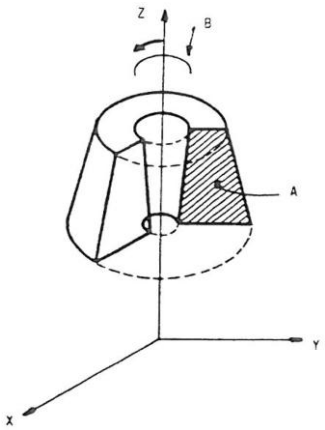


Fig. 2.6b - Génération par rotation

Point fort :

- Cette technique représente un moyen naturel et intuitif pour construire une variété d'objets : Elever des immeubles, donné de l'épaisseur aux lettres d'un logo…etc.

Point faible :

- La trajectoire et la forme de l'objet peuvent conduire à ce que l'objet se croise avec lui-même.

2.3.2 Représentation par frontière Brep : [PER 97]

Dans la représentation Brep (Boundary representation) l'objet est décrit par son contour : ses facettes, ses arcs ou arêtes, ses sommets et les diverses relations topologiques (incidence, contiguïté) entre ces éléments. Ce modèle garantit la cohérence topologique mais sous certaines conditions :

- Les objets doivent être fermés pour constituer des solides (pas de facettes pendantes).
- L'union de toutes les facettes doit impérativement constituer l'objet entier.
- L'intersection des objets n'est pas tolérée.

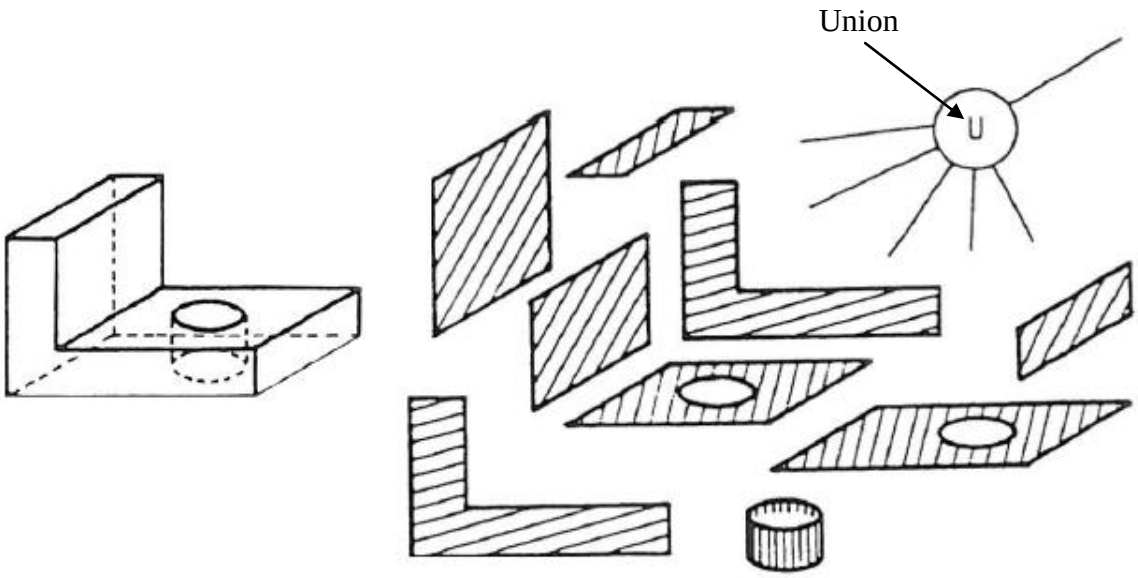


Fig. 2.7 - Exemple de solide représenté par sa frontière

Point fort :

- Les frontières sont explicitement connues et représentées ce qui procure une visualisation facile non ambiguë et rapide.

Points faibles :

- La représentation explicite des contours s'avère fort gourmande en espace mémoire.
- L'imprécision numérique compromet la fiabilité des Breps (les coordonnées d'un sommet ne vérifient pas les équations des surfaces incidentes)

2.3.3 Représentation constructive CSG: [PER 97]

Le modeleur CSG (Construct Solid Geometry) décrit l'objet par un arbre de construction dont les feuilles sont des primitives géométriques de base (sphère, cylindre, cube, tore, cône…) et les nœuds des opérations booléennes (and, or) ou ensemble (union, intersection) ou encore des transformations géométriques non linéaires (bruit, torsion, plissement…).

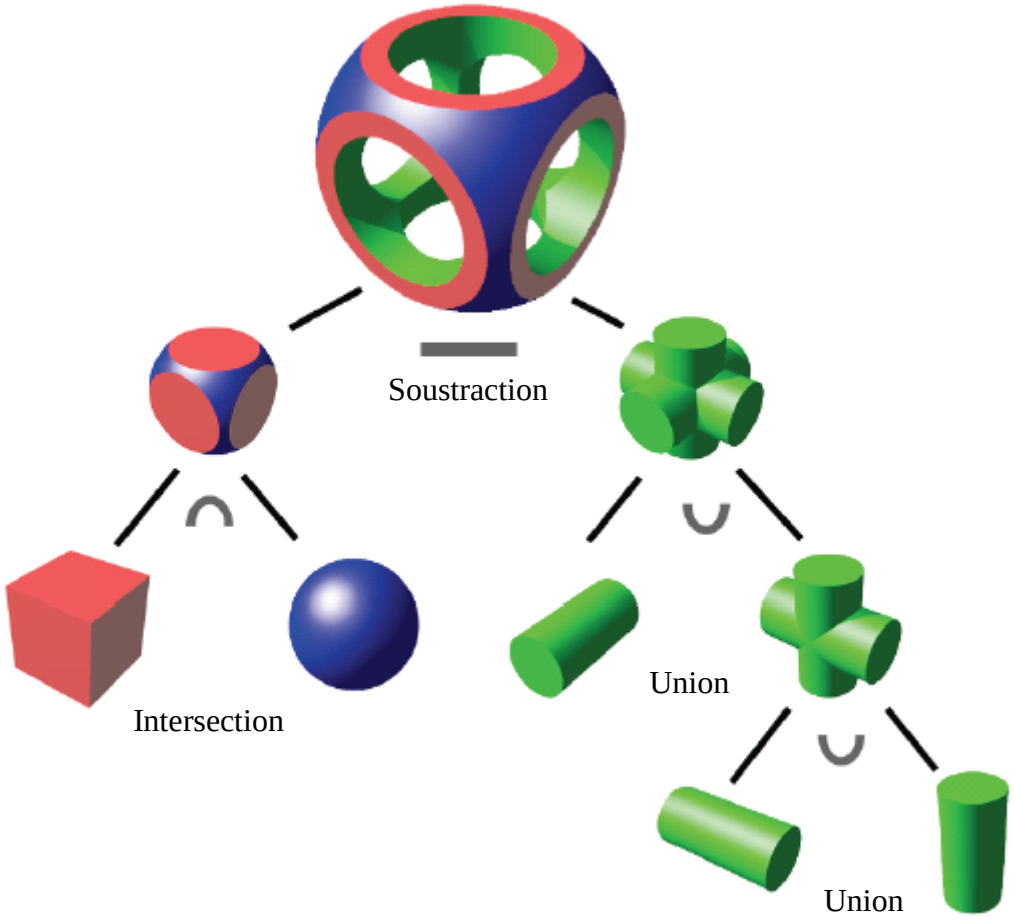


Fig. 2.8 - Exemple de construction CSG

Point forts :

- Très approprié pour les modèles géométriques où la précision est de rigueur (exp : pièce mécanique).
- L'histoire de la construction est mise on avant (on peut voir comment l'objet a été conçu).

Point faibles :

- Manque de souplesse ; contrairement au modèle polygonale il est impossible de travailler point à point sur l'objet 3D.
- Ce modèle permet l'union mais pas l'accolement.

2.3.4 Représentation spatiale : [PER 97]

Consiste à découper un objet tridimensionnel en plusieurs cellules identiques, nommées voxels (volume élément), dans une grille fixe et régulière. Divers types de cette représentation existent, la plus célèbre étant l'approche octrée.

– **L'approche Octrée :**

C'est la division successive d'un volume (objet) en huit sous volumes (partitions). Si chacune des partitions a la même valeur (vide ou pleine) alors le processus de décomposition s'arrête. Sinon on divise chacune en huit sous volume…etc. Jusqu'à atteindre le niveau de voxel.

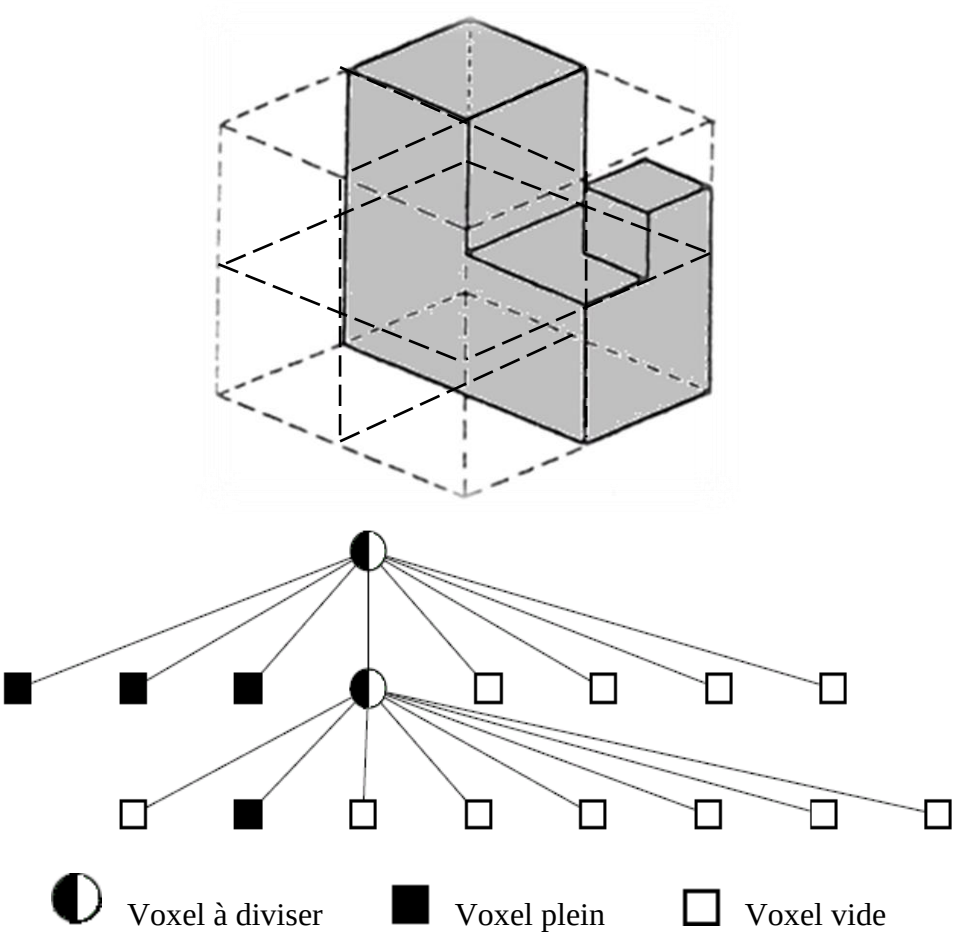


Fig. 2.9 - Représentation Octrée

Points forts :

- L'objet peut être codé par une liste unique et non ambigüe.
- Cette technique est souvent utilisée dans des applications biomédicales pour représenter des données volumétriques.

Points faibles :

- Les solides ne peuvent êtres qu'approximés.
- Coût en espace de stockage en mémoire élevé.

3) La modélisation d'objets naturels :

Les objets naturels sont caractérisés par leur aspect itératif et irrégulier d'où l'intérêt de les modéliser par des méthodes appropriées, les modèles récursifs permettent d'avoir des formes mouvementés comme un relief montagneux. Les modèles stochastiques offrent la possibilité de générer des effets atmosphériques comme le brouillard ou la fumée.

3.1 La modélisation récursive :

Introduit pour modéliser des objets n'ayant pas des formes géométriques telles que les arbres, montagnes…etc.

3.1.1 La modélisation par les fractales : [FOL 01]

Elle permet à partir d'un motif géométrique simple appelé motif de base (carré, triangle…) et d'un générateur de modéliser des objets naturels complexes tel que les paysage, les arbres et les montagnes. Les images en résultant sont spectaculaires. L'un des modèles les plus connue est la courbe Von Koch.

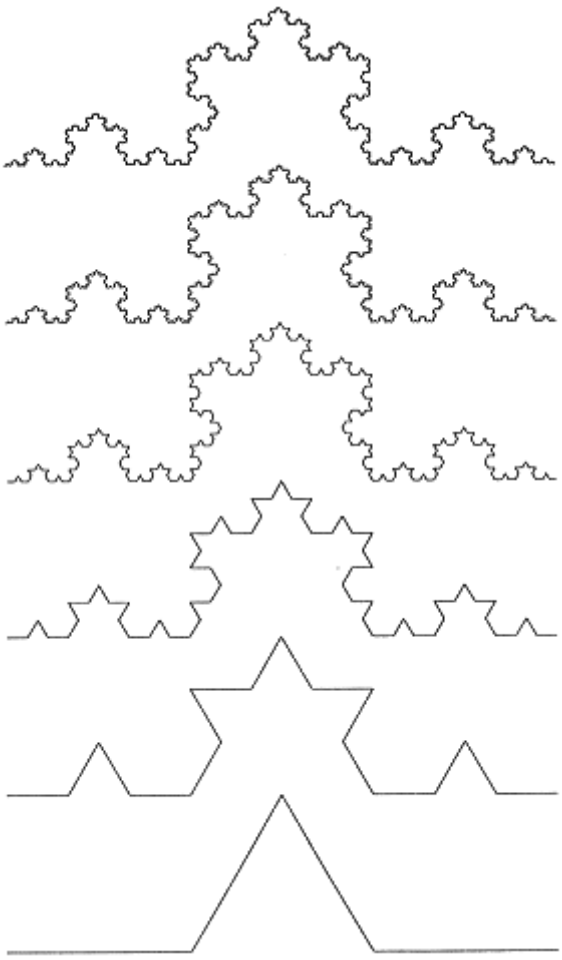


Fig. 2.10 - Courbes Von Koch

3.1.2 La modélisation basée sur les grammaires (Graftal) :

Smith a présenté un modèle basé sur les langages grammaticaux appelés *Graftals*, cette méthode est idéale pour représenter des plantes et des arbres. Chaque élément de l'alphabet représente une partie élémentaire d'un arbre.

3.2 La modélisation stochastique :

Elle est basée sur les lois de probabilités. Elle a été introduite pour modéliser les objets tridimensionnels ayant un aspect évolutif dans le temps à l'exemple de la fumée, les nuages…etc.

4) Comparaison :

Nous donnons le tableau de comparaison ci-dessous selon les critères suivants :

Unicité : il n'y a qu'une seule manière de représenter un objet.
Ambiguïté : un même schéma peut représenter plusieurs objets.
Création : les objets sont faciles à créer.
Concision : économique en espace mémoire.

	Fil de fer	Surfacique	CSG	Extrusion	Brep	Octrée
Unicité	Oui	Oui	Non	Non	Oui	Oui
Ambiguïté	Oui	Oui	Non	Non	Non	Non
Création	Oui	Oui	Oui	Oui	Non	Non
Concision	Oui	Oui	Oui	Oui	Non	Non

Tableau 1 : Tableau comparatif des différentes modélisations.

5) Conclusion :

Dans ce chapitre, nous avons abordé les principales théories de la modélisation, leur points forts et leur points faibles, nous avons appris que pour chaque objet il y avait une modélisation bien appropriée, mais pour afficher tous ces modèles il faut les visualiser et c'est ce que nous allons étudier dans le prochain chapitre.

1) Introduction :

Les objets que nous manipulons sont définis dans un espace 3D, alors comment faire pour les visualiser dans une fenêtre en 2D? C'est ce que nous allons étudier ici, les étapes à suivre pour remédier à ce problème.

2) Technique de visualisation :

Pour visualiser une scène 3D il faut :

- Déterminer les objets qui seront contenu dans la clôture (la surface d'affichage).
- Projeter les objets sur un plan 2D.
- Effectuer des transformations de visualisation.
- Afficher l'image sur écran.

2.1 Découpage :

On considère ici les parties de la scène contenues dans la clôture, qui seront affichées sur l'écran par la suite, souvent utilisé pour voir partiellement un objet correspondant à un détail agrandi.

2.2 Projection :

Comme son nom l'indique durant cette étape les objets tridimensionnels sont projetés sur un plan bidimensionnel, afin de les préparer à un affichage sur écran.

2.3 Transformation et visualisation :

Elle consiste à passer du système de coordonnées de l'utilisateur à celui du dispositif d'affichage de façon à ce que la clôture soit affichée sur écran, pour se faire nous devons effectuer une translation, un changement d'échelle et une seconde translation.

La première translation consiste à ramener la clôture à l'origine du repère utilisateur (fenêtre), elle est suivie d'un changement d'échelle, la seconde tient compte du décalage de la fenêtre par rapport à l'origine de l'écran.

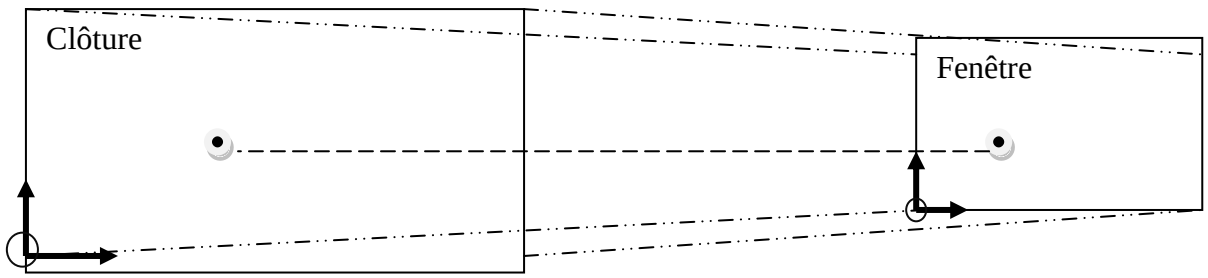


Fig. 3.1 - Passage d'un point de la clôture en un point de la fenêtre.

2.4 Affichage :

Consiste à convertir les segments, les polygones, les courbes…, en primitives graphiques (triangle, carré, cercle…etc.) dans l'espace écran.

3) Manipulation d'objets tridimensionnels :

La manipulation d'objets 3D est vitale pour créer une scène 3D correcte; changement de repère, translation, rotation, changement d'échelle. Ces différents concepts font appel à la géométrie analytique 3D.

3.1 Changement de repère :

Un repère est un système de coordonnées défini à partir d'un point O, appelé origine et de trois vecteurs unitaires perpendiculaires deux à deux : O_x, O_y, O_z. Le changement de repère est le passage d'un repère à un autre.

3.2 Translation :

Elle permet de déplacer au sein d'un même repère, chacun des points d'une surface ou d'un objet suivant une même direction et une même distance.

3.3 Rotation :

Les rotations des objets en 3D s'effectuent par rapport à un axe de rotation (l'axe X, Y, Z) et selon un angle bien défini.

3.4 Changement d'échelle (homothétie) :

Elle effectue des agrandissements ou des réductions des objets donnés. Pour cela il suffit de définir un facteur d'agrandissement selon les trois axes X, Y, Z.

4) Suppression des parties cachées :

Le problème de la suppression des parties cachées est l'un des plus difficiles de l'infographie. Comme nous le montre la Figure 3.2 sans cette suppression une scène 3D devient ambiguë, plusieurs algorithmes ont été créés pour remédier a ce problème, on distingue deux types d'algorithmes :

- Les algorithmes de l'espace objet.
- Les algorithmes de l'espace image.

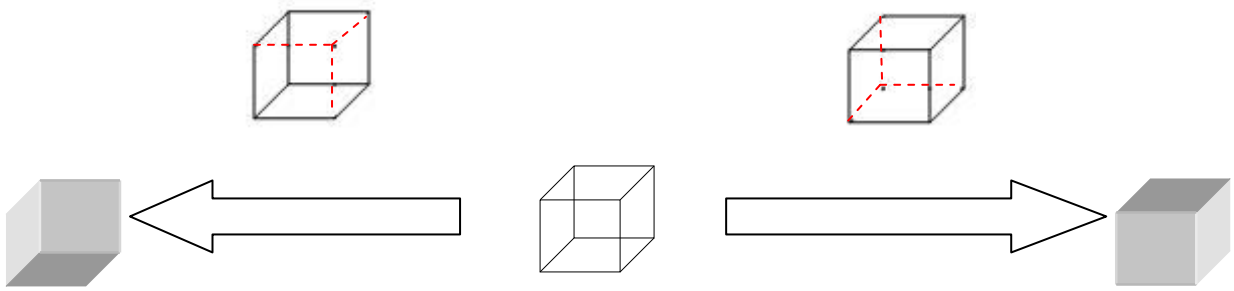


Fig. 3.2 - Exemple d'ambiguïté si on ne procède pas à l'élimination des parties cachées

4.1 Les algorithmes de l'espace objet :

Les algorithmes de l'espace objet sont implantés dans le système de coordonnées physique (espace infini) dans lequel les objets sont décrits. Ils sont particulièrement utiles dans les applications précises de l'ingénierie.

4.1.1 Algorithme de Roberts : [ROG 97]

Il a été la première solution connue au problème de la suppression des parties cachées. L'algorithme commence par éliminer les arêtes ou les plans de chaque volume (objet 3D) qui sont occultés par le volume lui-même. Ensuite, chaque arête restante de chaque volume est comparée à chacun des volumes restants pour déterminer la ou les portions éventuellement occultées.

4.1.2 Algorithme d'appel : [ROG 97]

Cette une amélioration de l'algorithme de robert. Il associe à chaque partie de segment d'un objet un degré d'invisibilité ; ce degré est indiqué par le nombre de face qui masque la partie. Donc les parties visibles sont celles dont le degré est nul.

4.2 Les algorithmes de l'espace image :

Les algorithmes de l'espace image sont implantés dans le système de coordonnées de l'écran (espace fini) sur lequel les objets sont visibles. Les calcules ne sont réalisés qu'à la précision de l'écran. Les scènes calculées dans l'espace image puis fortement agrandies ne donnent pas de résultats acceptables. C'est ainsi que les extrémités des segments peuvent ne pas coïncider.

4.2.1 Algorithme de l'horizon flottant : [ROG 97]

La technique de l'horizon flottant est la plus fréquemment utilisée pour la suppression des lignes cachées dans la représentation 3D des surfaces exprimées par une fonction de la forme F(x, y, z)=0.

L'idée fondamentale de cette technique est de convertir les problèmes 3D en 2D en coupant la surface par une série de plans sécants parallèles, à x, y ou z constante. La figure 2.3 montre des plans à *z* constante. La fonction F(x, y, z)=0 se trouve réduite à une courbe dans chacun de ces plan parallèles, c'est-à-dire réduite à

$$y = f(x, z) \text{ ou } x = g(y, z)$$

avec z constant pour chacun des plans parallèles.

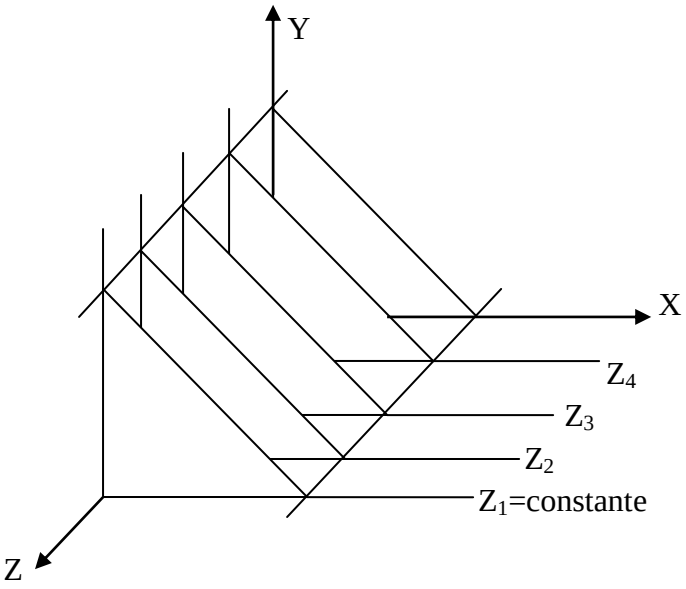


Fig. 3.3 - plans sécants à coordonnées constantes.

Cet algorithme trie tous d'abord les plans *Z* = constante dans l'ordre croissant des distances au point de vue. En commençant par le plan *Z* = constante le plus rapproché du point de vue, on génère la courbe de chaque plan ; c'est-à-dire que l'on calcule la valeur de *Y* pour chaque valeur de la coordonnée *X* de l'espace image

L'algorithme de suppression des lignes cachées est alors :

- Si pour une valeur donnée de *X* la valeur *Y* de la courbe est plus grande que la valeur *Y* de la courbe précédente pour cette même valeur de *X*, la courbe est visible. Sinon, elle est cachée.

L'algorithme fonctionne correctement sauf si l'une des courbes successives plonge au-dessous de la première courbe, comme sur la figure 3.4. Ces courbes sont normalement visibles au-dessous de la surface, mais l'algorithme les traite comme invisibles. La face inferieur qui flotte en s'enfonçant au cours du déroulement de l'algorithme. On implante pour cela un deuxième tableau de dimension égale à la résolution de l'espace image dans la direction des x et contenant la plus petite valeur de y pour chaque valeur de x.

L'algorithme devient alors :

- Si pour une valeur donnée de x la valeur de y de la courbe est plus grande que la plus grande valeur y, ou plus petite que la plus petite valeur de y de la courbe précédente pour cette même valeur de x, la courbe est visible. Sinon, elle est cachée

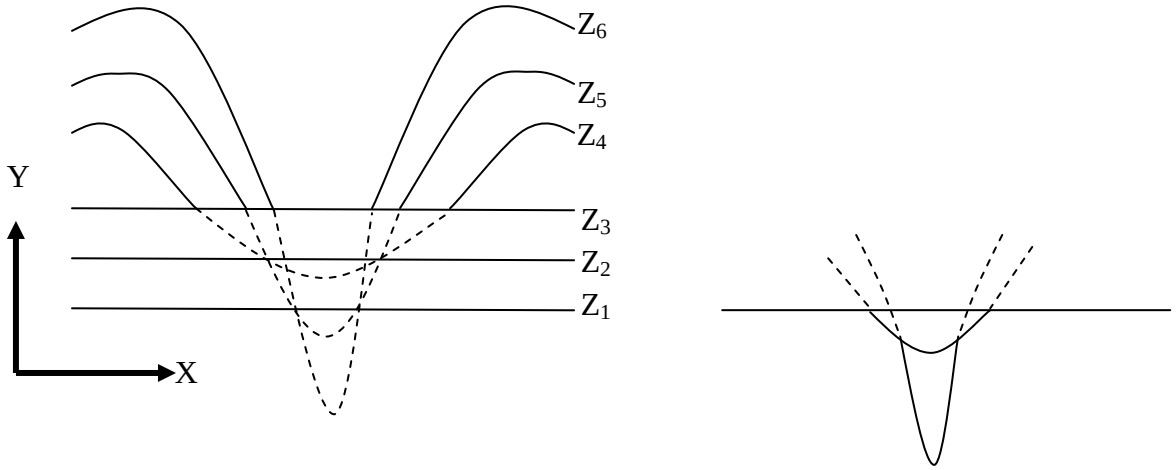


Fig. 3.4 - Prise en compte de la face inferieure de la surface.

4.2.2 Algorithme du Z-buffer : [ROG 97]

Le Z-buffer (le tampon des Z) est l'un des algorithmes les plus simples de la suppression des parties cachées. Le z-buffer est une simple extrapolation du concept de mémoire d'image. Une mémoire d'image sert à stocké les attribue (l'intensité) de chaque pixel de l'espace image. Le z-buffer est un tampon des profondeurs séparé qui sert à stocker la coordonnée *Z*, ou profondeur, de chaque pixel visible de l'espace image. En fonctionnement la valeur *Z* d'un nouveau pixel à écrire dans la mémoire d'image est comparée à la valeur de profondeur déjà stockée pour ce pixel. Si cette comparaison indique que le nouveau pixel est devant le pixel déjà stockée dans la mémoire d'image, ce nouveau pixel est écrit dans la mémoire d'image et le z-buffer mis à jour par la nouvelle valeur de *Z*. dans le cas contraire, aucune action n'est effectuée. Sur le plan conceptuel, l'algorithme consiste en une recherche de la plus grande valeur de Z(X, Y) sur X, Y.

Un exemple donne une idée plus précise de l'algorithme.

Considérons le rectangle dont les coordonnées des sommets sont P_1 (10, 5, 10), P_2 (10, 25, 10), P_3 (25, 25, 10), P_4 (25, 5, 10) et le triangle de sommets P_5 (15, 15, 15), P_6 (25, 25, 5), P_7 (30, 10, 5). Le triangle pénètre le rectangle par derrière, comme on le voit sur la figure 2.5a. Les polygones doivent être affichés à une résolution d'image de 32x32 à l'aide d'une mémoire d'image, l'intervalle du z-buffer va donc de zéro à 15. Le point de vue est à l'infini sur l'axe des z positifs, comme on peut le voir su la figure 3.5b.

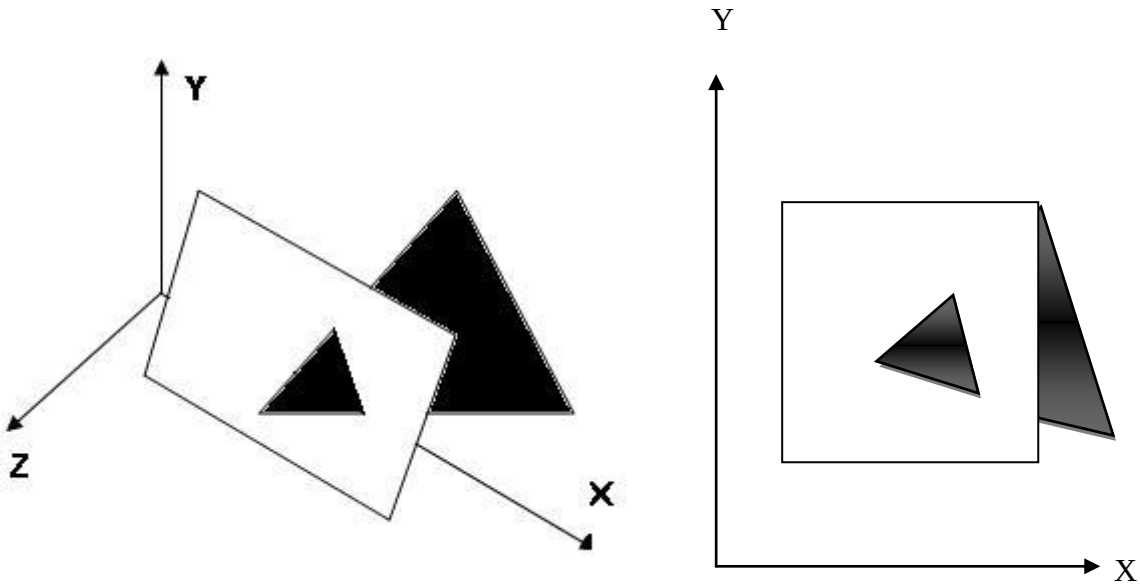


Fig. 3.5 - Exemple algorithme du z-buffer (triangle pénétrant).

5) Conclusion :

Nous venons de voir les différent moyens utilisés pour afficher des scènes 3D sur une fenêtre 2D et un certain nombre d'algorithmes qui résous les ambiguïtés du au fil de fer, bien que ces dernier ne soient pas les seuls disponibles. Il existe notamment des algorithmes qui peuvent travailler dans les deux espaces (objet et image) en même temps.

Maintenant nous allons passer au rendu qui va nous permettre d'obtenir des scènes 3D hautement réaliste.

1) Introduction :

A simplement parler, le rendu est le processus de production d'images réalistes pour y parvenir, on utilise les effets de lumières, ombres et textures.

Nous commencerons par définir plus exactement ce qu'est une image réaliste, et quelles sont les simplifications qui peuvent être réalisé au niveau des objets 3D ou des technique de rendu, de manière à réduire les temps de calcule tout en générant des images physiquement crédibles. Tenant compte de ces réflexions, nous présentons ensuite quelques techniques particulièrement adaptées au redu réaliste dans le contexte de la RA.

2) Aspects perceptifs : [ROB 06]

Qu'est-ce qu'une image réaliste ? Cette question n'est pas aussi simple qu'elle n'y paraît. D'abord, il est difficile de comparer une image avec la réalité dans la mesure où une image n'est qu'une projection en deux dimensions de la réalité (même une image stéréoscopique n'est qu'une reconstitution mentale d'une image 3D obtenu à partir de deux images 2D). Ainsi, une très grande majorité d'études sur ce domaine utilise comme référence absolue des photographies de la réalité, ce qui limite la portée de la question. Cependant, même dans ce cadre (une image réaliste est une image synthétique dont on croit qu'il s'agit d'une photographie), la question de savoir si une image est réaliste ou non reste complexe : peut on répandre par ou oui par non à cette question ? Si oui, sur quels critères se base-t-on ?

Des critères objectifs comme la comparaison deux à deux des pixels de photographie de référence avec les pixels de l'image synthétique n'apportent pas vraiment de réponse : à supposer que l'on puisse mettre en évidence le fait qu'une image A possède objectivement mois de différence avec une photographie de référence qu'une image B, quel conclusion pouvons-nous en tirer quant au réalisme de l'image A ? En fait aucune, car il n'existe pas de seuil discriminant objectif sur un tel critère.

Finalement la notion de réalisme d'une image est subjective. Si l'on présente une image de synthèse plus ou moins réaliste à un ensemble d'individus, certain penseront qu'il s'agit d'une photographie, d'autre détecteront immédiatement des éléments suspect qui leur permettront de reconnaître une image synthétique.

3) L'illumination(Shading) :

Dans le monde réel lorsque de l'énergie lumineuse tombe sur une surface, elle peut être absorbée, réfléchie ou transmise. Une petite quantité de l'énergie lumineuse incidente est absorbée par la surface et convertie en chaleur. Le reste est soit réfléchi, soit transmis. C'est la lumière réfléchie ou transmise qui rend un objet visible.

Dans le monde virtuel, l'illumination est établie par des modèles qui sont une représentation mathématique de la lumière. Et tout ça n'est qu'une approximation des phénomènes lumineux naturels.

3.1 Les sources de lumière :

Il existe plusieurs sources lumineuses en synthèse d'image, on cite entre autre :

- Les sources ponctuelles où les rayons lumineux viennent d'un seul point.
- Les sources directionnelles, où les rayons lumineux (parallèles) vont tous dans la même direction.
- Source de lumière ambiante ou globale dont l'intensité est la même en tout endroit de la scène.

3.2 La lumière renvoyée par les objets :

3.2.1 La lumière ambiante : [LOO 92]

Elle est définie comme une lumière diffuse provenant de toutes les directions avec la même intensité, de ce fait la luminosité d'une face éclairée par la lumière ambiante est indépendante de la direction de la face par rapport à l'observateur.

L'éclairage d'une scène avec la lumière ambiante aura pour effet d'éliminer toute sensation de profondeur, alors que s'en priver totalement produira une image sombre.

Un bon éclairement demandera donc à la fois l'action de sources lumineuses ponctuelles ou directionnelles et de la lumière ambiante.

3.2.2 Réflexion diffuse :

Considérons un objet éclairé par une source ponctuelle, la luminosité de l'objet variera d'un endroit à un autre en fonction de sa direction et de sa distance par rapport à la source lumineuse.

La réflexion diffuse correspond à l'absorbation, puits à la réémission diffuse de l'énergie lumineuse dans toutes les directions, c'est grâce à la lumière diffuse que l'on peut distinguer la couleur globale de l'objet éclairé. Elle est calculée à partir des normales aux facettes et de l'angle entre ces normales et la direction de chaque source de lumineuse. Le calcule est simple produit scalaire qui peut se faire très rapidement pour un grand nombre de polygones, ce qui est parfait pour la contrainte temps réel.

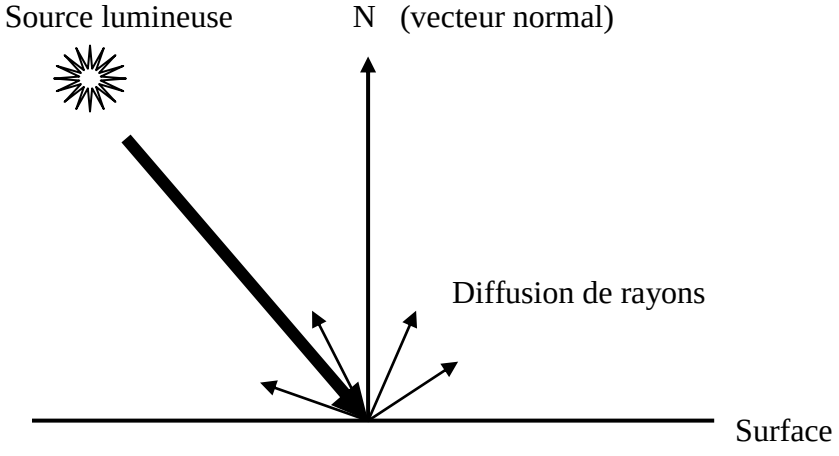


Fig. 4.1 - Réflexion diffuse

3.2.3 Réflexion spéculaire : [ROB 06]

La réflexion spéculaire est visible sur les surfaces brillantes ; elle correspond aux taches lumineuses qu'on peut trouver sur les objets brillants, celles-ci nous prouvent que les rayons captés par l'œil sont cette fois réfléchis dans la direction de vision de l'observateur.

La couleur du reflet spéculaire correspond exactement à la couleur de la source incidente.

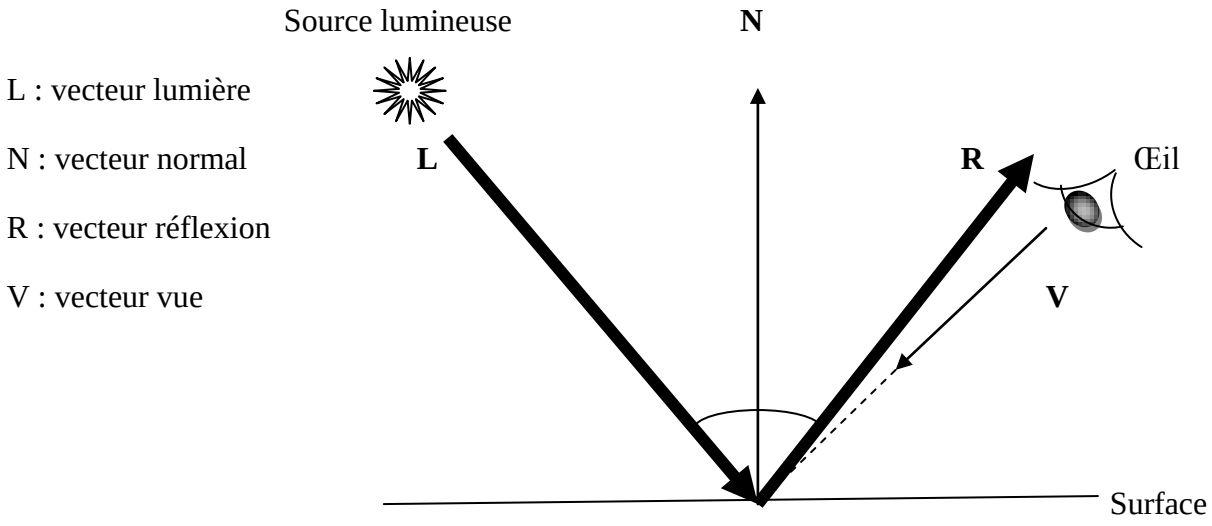
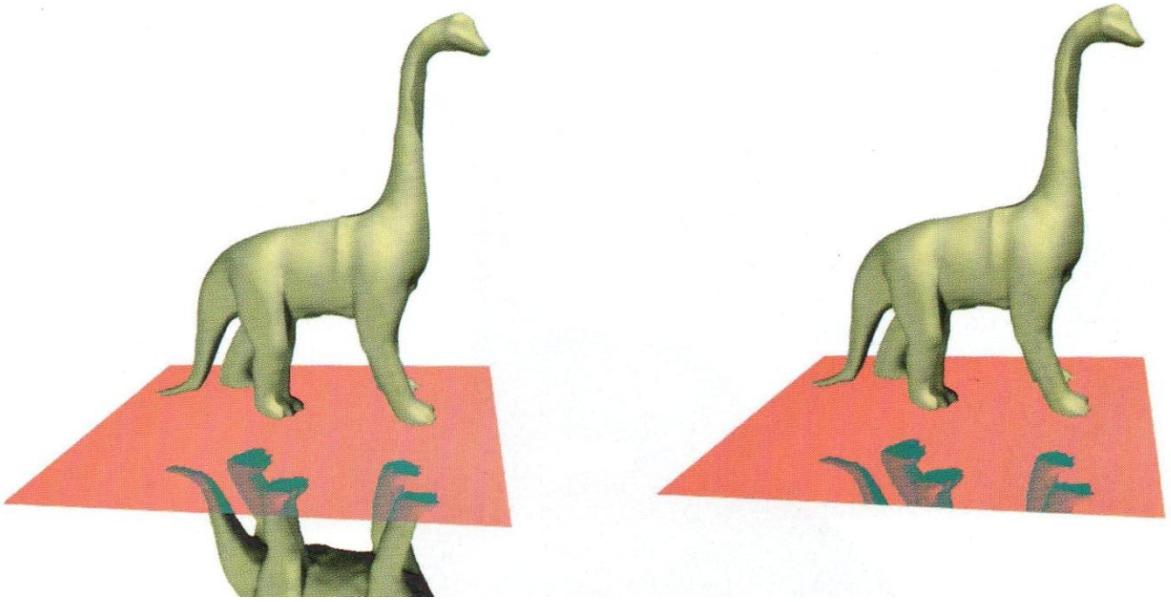


Fig. 4.2 - Réflexion spéculaire

Un rendu précis des réflexions spéculaire nécessite de mettre en œuvre un processus récursif extrêmement couteux en temps de calcule. Pour obtenir un rendue équivalent en temps réel, il est donc nécessaire d'utiliser certaines astuces. Une première astuce consiste à exploiter le fait que le reflet d'un objet sur une surface plane est la trace de cet objet à travers une symétrie. L'idée est donc de dessiner l'objet une deuxième fois, en l'inversent par rapport au plan (Fig. 4.3a). Bien sûr, un problème évident qui apparaît clairement en Fig. 3.3a est une le reflet ainsi généré peut dépasser de la surface plan. Pour palier ce problème on peut placer un pochoir (*stencil*) autour du plan concerné, de manière à ne dessiner que les facettes du reflet qui ce projettent sur le pochoir (Fig. 4.3b). La technique du pochoir fait partie des fonctionnalités de base de la bibliothèque OpenGL. Malheureusement, ce procédé ne permet pas de représenté des reflets sur des surface non planes, et peut être difficilement appliquait à des scènes complexes.

Une technique plus facilement exploitable et conduisant à des résultats très réaliste est la technique du mapping d'environnement. Le mapping sphérique étant pris en compte par OpenGL, ce procédé peut facilement être mis en œuvre dans un contexte temps réel.



(a) **(b)**

Fig. 4.3 – Réflexion spéculaire avec pochoir.

4) L'ombrage :

Lorsque la position de l'observateur coïncide avec celle de la source de lumière, on ne voit aucune ombre. L'ombre apparait dés que leur position diffère. Les ombres contribuent fortement au réalisme des scènes en améliorant la perception de la profondeur.

On observe qu'une ombre se compose en réalité de deux parties, l'ombre proprement dite (l'ombre pure), et la pénombre. La zone d'ombre centrale dense, noire et bien délimitée est l'ombre pure. La zone plus claire qui l'entoure est la pénombre. Mais il faut savoir que dans l'infographie les sources lumineuses ponctuelles ne génèrent que des ombres pures.

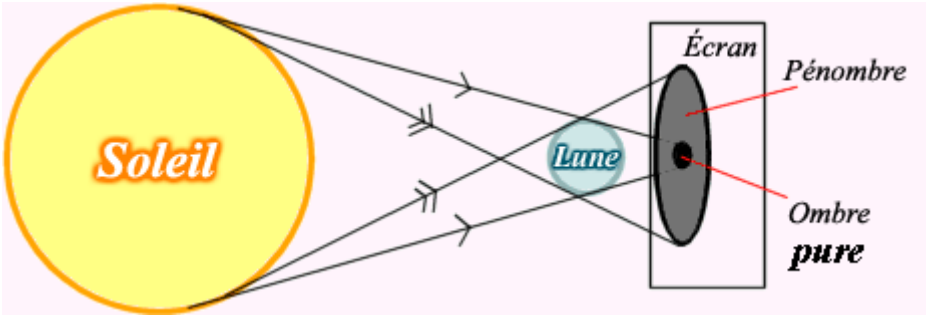


Fig. 4.4 - Exemple d'ombre pur et de pénombre

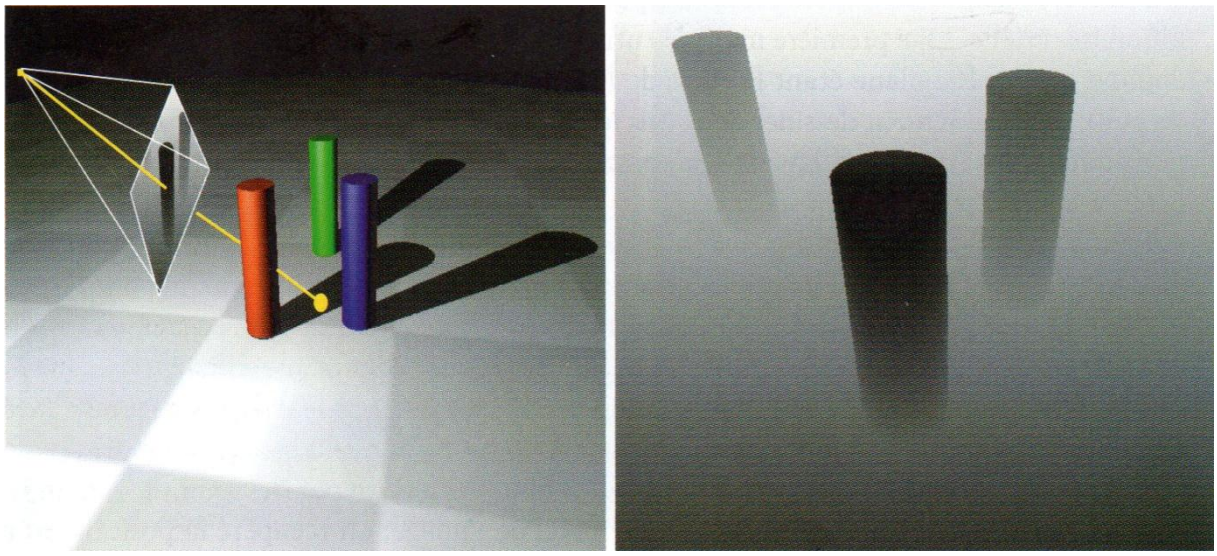
4.1 Représentation double : [ROB 06]

Plusieurs techniques ont été proposées pour représenter les ombres en tirant parti de l'accélération matérielle. Une première méthode utilise encore une fois le principe de la représentation double : une ombre plane étant la trace de l'objet porteur de l'ombre à travers une projection, on peut tous simplement dessiner deux fois le même objet, en utilisent deux matrice de projection différente (Fig. 4.5) Là encore, l'ombre qui dépasse du plan peut être supprimée grâce à la technique du pochoir.

Fig. 4.5 – Représentation double avec pochoir.

4.2 Mapping d'ombre : [ROB 06]

Pour des scènes plus complexes, la technique du mapping d'ombre (*shadow mapping*) est plus adaptée. Elle exploite le fait que les cartes graphiques sont particulièrement performantes en ce qui concerne les opérations de z-buffer. Le principe est de générer un z-buffer de la scène en la dessinant depuis le point de vue de la source lumineuse. Le z-buffer de la source lumineuse (appelé *shadow map*) partitionne le volume de la scène en deux région : la région ombragée (points dont la profondeur est supérieur à la profondeur stockée dans le z-buffer) et la région non ombrager (autre point) ; le dessin de la scène depuis le point de vue normale se fait alors de la façon suivante : pour chaque pixel de l'image, on récupère la position du point 3D vu à travers le pixel. Si la distance entre ce point et la source lumineuse est supérieur à la distance conservée dans la *shadow map*, alors le point est dans l'ombre, sinon il est éclairé. La Fig. 4.6 illustre cette méthode.

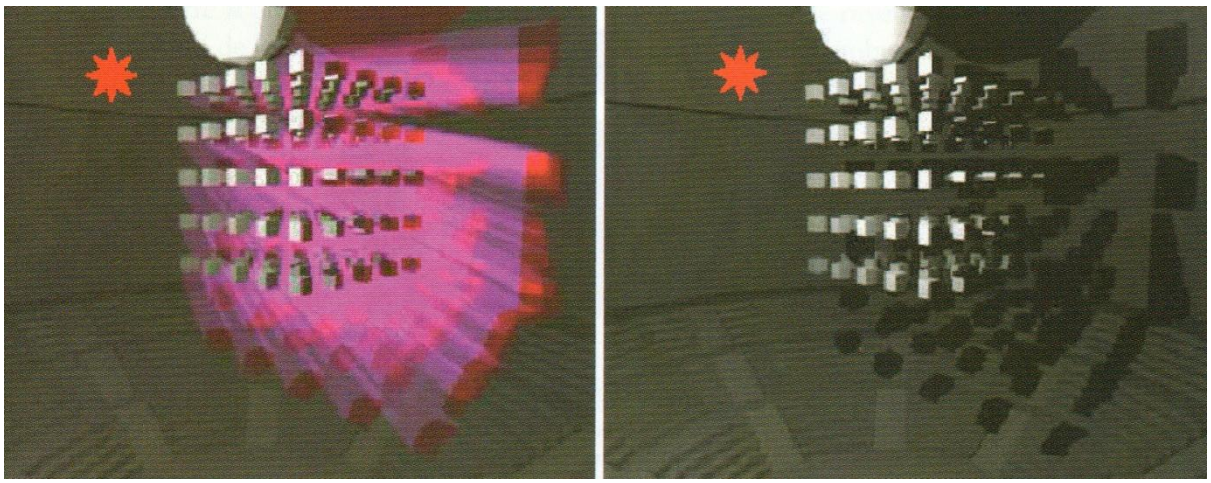


(a) **(b)**

Fig. 4.6 – Exemple de shadow map.

4.3 Volume d'ombre : [ROB 06]

Une autre technique assez répandue et purement géométrique est celle qui consiste à associer un volume d'ombre (shadow volume) à chaque objet de la scène, correspondant au volume de la scène occulté par l'objet, du point de vue de la source lumineuse concernée (Fig.4.7). Le test d'appartenance d'un pixel à une zone occultée se fait alors en comptant le nombre de fois que le rayon allant du point 3D visible à travers le pixel, entre ou sort d'un volume d'ombre. Si la différence entre le nombre d'entrées dans un volume et le nombre de sorties est strictement positif, alors le point représenté est dans la zone d'ombre.



(a) **(b)**

Fig. 4.7 – La technique du Shadow volume.
Les volumes d'ombres sont représentés en rose.

4.4 Ombre douce : [ROB 06]

Les technique qui viennent d'être représentées génèrent des ombres dites dures, c'st-à-dire des ombres de type binaire (un point de la scène est dans l'ombre ou ne l'est pas). Cependant, ce type d'ombre est obtenu en présence de sources lumineuses ponctuelles, qui n'existent pas en pratique : les images calculées avec ce procédé peuvent donc parfois sembler artificielles. Pour obtenir des images plus réalistes, il est préférable de dessiner des ombres dites douce : ce type d'ombres est composé d'une zone d'occultation pure et d'une zone de pénombre, où la lumière est partiellement reçue. De nombreuses techniques ont été proposées pour générer des ombres douces en temps réel. La plus simple d'entre elles consiste à calculer les ombres plusieurs fois de suite (en utilisent l'une des technique présentées précédemment), en déplaçant légèrement la source lumineuse à chaque nouveau calcul, puis à additionner les ombre obtenues (Fig. 4.8).

(a) **(b)** **(c)**

Fig. 4.8 – Exemple de l'ombre douce, (a) un, (b) quatre et (c) soixante-quatre calculs d'ombres ont été faits.

5) Texture : [ROG 97]

En infographie, le détail de la surface d'un objet est appelé sa texture. On considère généralement deux types de texture. Le premier consiste à ajouter à une surface lisse un motif élaboré séparément (texture 2D). La surface apparaît encore lisse après l'adjonction du motif. L'adjonction d'un motif à une surface lisse est fondamentalement une fonction de mappage, c'est-à-dire de topographie mémoire. L'autre type de texture consiste à donner à une surface l'apparence de rugosité (texture 3D). L'adjonction de rugosité est fondamentalement une fonction de perturbation.

Chapitre 4

Comme la base de l'adjonction de motifs de textures aux surfaces lisses est le mappage, le problème des textures se réduit à la transformation d'un système de coordonnées orthogonales (u, w), et la surface dans un second système de coordonnées orthogonales (θ,ϕ), alors l'adjonction du motif de texture à la surface implique de déterminer ou de spécifier une fonction de mappage entre les deux espace, c'est-à-dire

$$\theta = f(u, w) \qquad \phi = g(u, w)$$

Ou bien encor

$$u = r(\theta, \phi) \qquad w = s(\theta, \phi)$$

Bien que cela ne soit pas indispensable, la fonction de mappage est généralement supposée linéaire, c'est-à-dire

$$\theta = Au + B \qquad \theta = Cw + D$$

Les constantes A, B, C, D étant obtenues à partir de la relation entre deux points connus dans les deux systèmes de coordonnées. Un exemple simple va servir à illustrer cette technique.

Le motif de figure 4.9a doit être mappé sur l'élément de surface défini par l'octant de sphère représenté figure 4.9b. Le motif est une simple grille en deux dimensions de droites sécantes. La représentation paramétrique de l'octant de la sphère est donnée par

$$x = \sin\theta \sin\phi \qquad 0 \le \theta \le \pi/2$$
$$y = \cos\phi \qquad \pi/4 \le \phi \le \pi/2$$
$$z = \cos\theta \sin\phi$$

Supposant une fonction de mappage linéaire

$$\theta = Au + B \qquad \phi = Cw + D$$

Et supposant également que les coins du motif carré correspondent aux coins de l'élément de surface quadrilatéral, c'est-à-dire

$$u = 0, w = 0 \text{ à } \theta = 0, \quad \phi = \pi/2$$
$$u = 1, w = 0 \text{ à } \theta = \pi/2, \phi = \pi/2$$
$$u = 0, w = 1 \text{ à } \theta = 0, \quad \phi = \pi/4$$
$$u = 1, w = 1 \text{ à } \theta = \pi/2, \phi = \pi/4$$

On parvient à

$$A = \pi/2 \qquad B = 0 \qquad C = -\pi/4 \qquad D = \pi/2$$

1a fonction linéaire de mappage de l'espace uw dans l'espace $\theta\phi$ est

$$\theta = (\pi/2)\, u \qquad \phi = (\pi/2) - ((\pi/4)\, w)$$

Le mappage inverse de l'espace $\theta\phi$ dans l'espace uw est

$$u = \theta / (\pi/2) \qquad w = (\pi/2 - \phi) - (\pi/4)$$

Le résultat du mappage d'une simple droit de l'espace *uw* dans l'espace $\theta\phi$ et dans les coordonnées caractérisée *xyz* est visible au tableau ci-dessous. L'ensemble des résultats est illustré par la figure 4.9c.

u	*w*	θ	ϕ	*x*	*y*	*z*
¼	0	π/2	π/2	0,38	0	0,92
	¼		7/16π	0,38	0,20	0,91
	½		3/8π	0,35	0,38	0,85
	3/4		5/16π	0,32	0,56	0,77
	1		π/4	0,27	0,71	0,65

Tableau 2 : Résultat de la fonction de mappage.

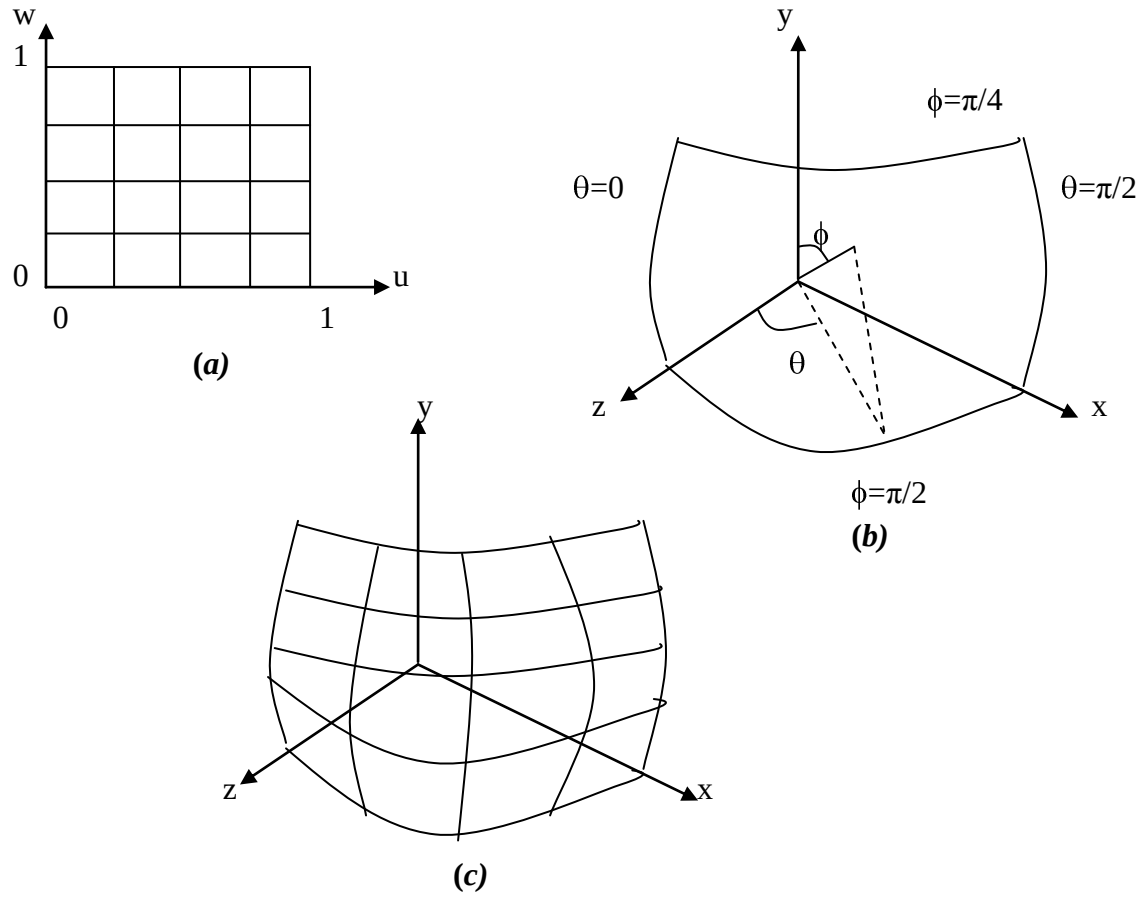


Fig. 4.9 - Mappage d'une texture 2D.

6) Conclusion :

Le redu réaliste en temps réel repose à la fois sur des considérations psychophysiologiques permettant de simplifier la scène à représenter et sur l'utilisation de certaines astuces de programmation, généralement implémentées dans les bibliothèques dédiées aux graphismes temps réel.

1) Introduction :

FLARToolKit est la version Flash Actionscript (v3) d'ARToolKit développée par Tomohiko Koyama[1], qui peut être utilisée pour développer rapidement des expériences basées sur le Web AR. C'est la bibliothèque la plus largement utilisée dans Flash AR, elle a le soutien d'une importante communauté de développeurs et de nombreux sites Web avec des exemples d'applications.

FLARToolKit reconnaît un repère visuel à partir d'une image d'entrée et calcule ensuite l'orientation de la caméra et sa position dans le monde 3D, ensuite elle superpose les modèles graphiques virtuels sur l'image vidéo en temps réel. FLAToolKit supporte tous les principaux moteurs graphiques 3D en flash (Papervision3D, Away3D, Alternativa3D).

La version open source de FLARToolKit peut être téléchargée à partir de:

http://www.libspark.org/wiki/saqoosha/FLARToolKit/en

2) Les principes de développement : [WEB7]

Le développement d'une application qui utilisent FLARToolKit ce divise en deux parties ; l'écriture de l'application et la création de marqueurs qui seront utilisés dans l'application afin de détecté l'emplacement des objets 3D dans le monde réel.

L'écriture d'une application avec FLARToolKit est très facile: plusieurs tutoriaux sont la pour aidée. De même, la phase de création des marqueurs est grandement simplifiée avec l'utilisation d'outil simple.

Les étapes suivantes doivent être prises en compte dans votre code:

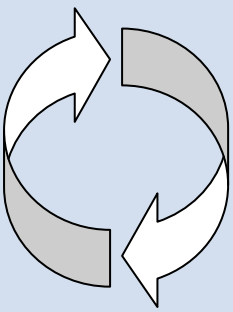
Initialisation	**1**. Initialisation de la capture vidéo et lecture des fichiers.
Boucle principale 	**2**. Prendre une trame d'entrée vidéo.
	3. Numériser l'image prise.
	4. étiquetage.
	5. Trouver des rectangles.
	6. Détecter les marqueurs reconnus dans la trame d'entrée vidéo.
	7. Calculer la matrice de transformation par rapport aux marqueurs détectés.
	8. Dessiner les objets virtuels sur les marqueurs détectés.
Fermeture	**9**. Fermer la vidéo.

Tableau 3 : Algorithme de FLARToolkit.

[1] Tomohiko Koyama est flash développeur japonais avec plus de 10 ans d'expérience, actuellement il travaille pour Katamari Inc

Les étapes 2 à 8 sont répétées continuellement jusqu'à ce que l'application se ferme, tandis que les étapes 1 et 9 sont tout simplement effectuées respectivement lors de l'initialisation et l'arrêt de l'application. En plus de ces étapes l'application peut avoir besoin d'événements souris et clavier.

Maintenant on va décrire en détail ces différentes étapes. Pour mieux comprendre en va prendre un exemple.

Fig. 5.1 : Exemple d'application, le modèle et son marqueur.

2.1 Initialisation de la capture vidéo et lecture des fichiers :

Ici on initialise le moteur graphique 3D. On charge le fichier de configuration de la camera et les différents fichiers en «.pat» des marqueurs, ainsi que les modèles utilisés dans la scène.

2.2 Prendre une trame d'entrée vidéo :

On capture une image avec une webcam qu'on stock ensuite dans une variable de type BitmapData.

Fig. 5.2 : exemple de capture d'image avec une webcam.

2.3 Numériser l'image prise :

Ici on transforme l'image en niveau de gris puits en noir et blanc pour appliquer un algorithme de détection de courbure.

Image capturée

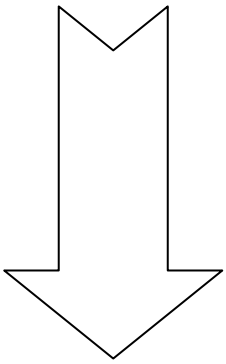


Image en niveau de gris

Fig. 5.3 : transformation de l'image en niveau de gris

Image en niveau de gris

Image en noir et blanc

Fig. 5.4 : Transformation de limage en noir et blanc

2.4 Etiquetage :

Ici on donne une couleur à chaque forme géométrique trouvée afin de les différencier.

Fig. 5.5 : Image étiquetée.

2.5 Trouver des rectangles :

Ici on cherche des formes rectangulaires avec l'image précédemment étiquetée.

Fig. 5.6 : résultat de la recherche des formes rectangulaires

2.6 Détecter les marqueurs reconnus dans la trame d'entrée vidéo :

Avant d'expliquer cette étape, parlons un peut des marqueurs.

- Il doit être de forme carrée.
- Seulement 50% de la superficie du centre est utilisée dans le processus d'appariement.
- Par défaut, le motif du Marker est de 16 x 16 points bitmap.
- La taille du motif peut être plus large, mais il faudra plus du temps pour la reconnaissance.

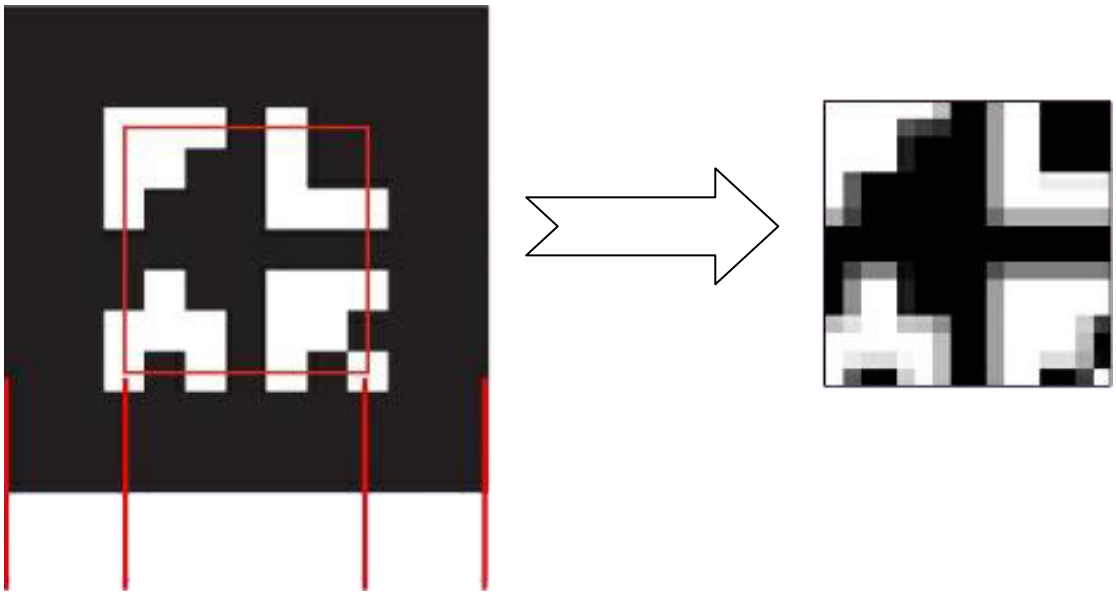


Fig. 5.7 : la superficie du marqueur utilisé.

On extrait les images à partir des rectangles détectés en utilisant la fonction homographique.

Fig. 5.8 : Extraction des images.

Chaque image extraite est comparée avec les marqueurs enregistrés.

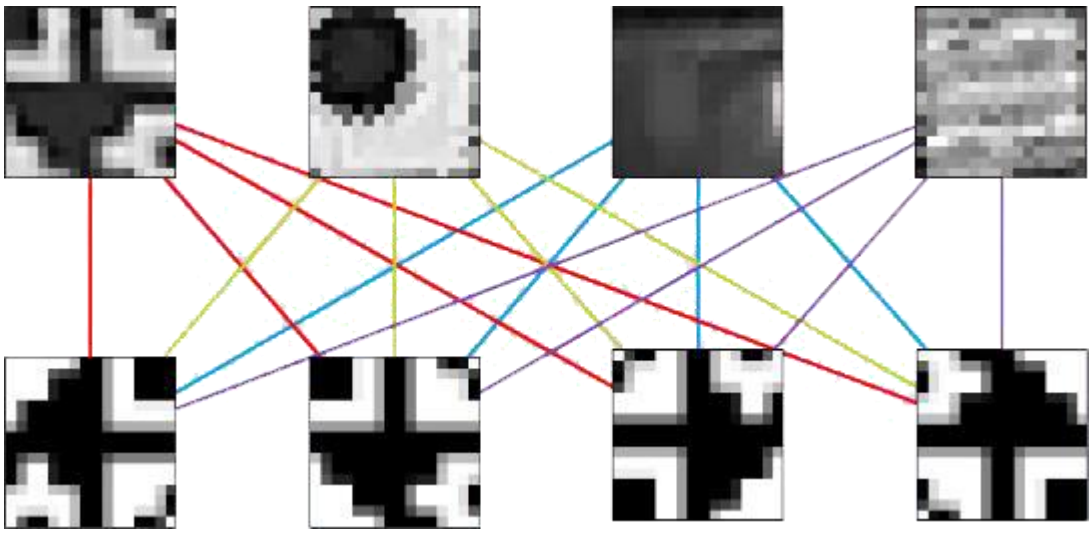


Fig. 5.9 : comparaison avec un marquer.

L'image qui reçoit la valeur la plus élevée est considérée comme le marqueur.

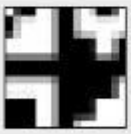
				
	0.38	0.29	-0.15	-0.11
	0.86	0.20	-0.01	-0.14
	0.27	-0.03	0.03	-0.14
	0.13	0.16	-0.08	-0.01

Tableau 4 : Résultat des comparaisons.

2.7 Calculer la matrice de transformation par rapport aux marqueurs détectés :

Tout d'abord c'est quoi une matrice de transformation ?

La matrice de transformation qu'utilise FLARToolKit est de 3x4 (3 lignes et 4 colonnes) elle contient toutes les données du marqueur, sa position, sa rotation, son échelle, et ses axe.

Avec la position du marqueur et son angle de rotation par rapport à la caméra, une matrice de transformation peut être calculée

Fig. 5.10 : l'angle et la position du marqueur su l'image détermine les données de la matrice de transformation.

2.8 Dessiner les objets virtuels sur les marqueurs détectés :

On utilise la matrice de transformation calculée à l'étape précédente sur le modèle pour qu'il soit parfaitement superposé sur le marqueur.

Fig. 5.11 : Le modèle est superposé sur le marqueur.

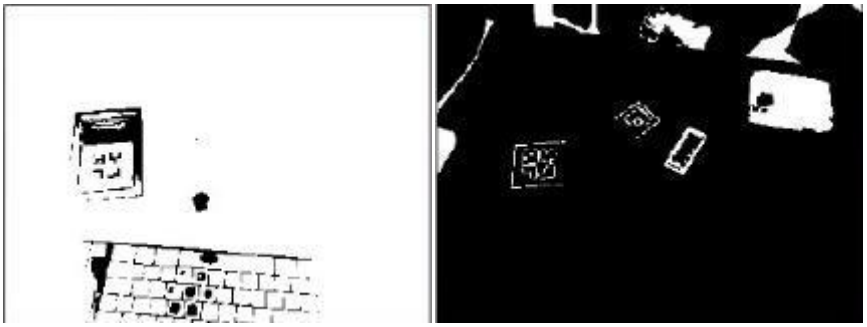
2.9 Fermer la vidéo :

C'est ici que l'application se termine et qu'on libère le flux de la vidéo.

3) L'intensité de la lumière : [WEB7]

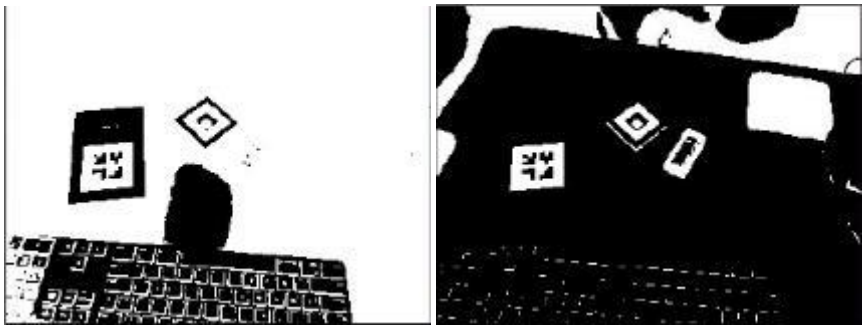
La réalité augmentée souffre d'un problème majeur et c'est celui de l'intensité de la lumière, La numérisation de l'image prise est l'étape la plus importante dans la reconnaissance des marqueurs, s'il y a trop ou pas assez de lumière la reconnaissance des rectangles ne peut avoir lieu, exemple en image.

Fig. 5.12 : exemple d'intensité de la lumière.



Threshold = 128
Fig. 5.13 : Après la numérisation les rectangles n'apparaissent plus.

Deux solutions se proposent à nous, soit régler le seuil de numérisation, le problème est que cette valeur est fixe. Donc dès que l'application est lancée elle ne peut plus être changée, s'il y a un brusque changement d'intensité de lumière dans la scène tous les marqueurs disparaissent.



Threshold = 50 **Threshold = 200**
Fig. 5.14 : exemple de réglage du niveau de numérisation.

Ou la seconde solution est d'utiliser FLARManager, c'est une bibliothèque qui ajoute des classes à FLARToolKit, elles permettent entre autres:

- De rendre le niveau de normalisation dynamique.
- Une meilleure prise en charge des multi-marqueurs.
- Une meilleure gestion des événements des différents marqueurs.

4) Image numérique : [ROB 06]

Précédemment j'ai parlé de numérisation d'image dons l'étape 3, mais c'est quoi une image numérique plus précisément ? C'est ce que je vais tenter d'expliqué ici.

Contrairement aux photographies argentiques ou aux dessins sur papier, les images créées par ordinateur sont numériques (représentées par une série d'éléments binaires). Une image numérique est une image dont la surface est divisée en élément de taille fixe appelés pixel, ayant chacun comme caractéristiques un niveau de gris ou une couleur (définis par l'utilisateur, prélèves dans une photographies numérique ou encore calculés à partir de la description d'une scène virtuelle à représenter). La numérisation d'une image est la conversion de celle-ci de son état analogique (distribution continue d'intensités lumineuses dans un plan) en une image numérique représentée par une matrice bidimensionnelle de valeurs numériques.

On distingue les images à niveau de gris (monochromes) des images couleurs. Le niveau de gris d'un pixel traduit l'intensité de la lumière au point physique correspondant. Il prend ses valeurs dans le domaine des entiers naturels compris entre 0 (noir) et 255 (blanc). Pour les images monochromes, chaque pixel est donc codé sur un octet. Fig. 5.15 donne un exemple de matrice d'octets, représentant une image à niveaux de gris de taille 15 x 15: il s'agit d'un « smiley » comportant trois niveaux de gris: le noir (0) pour le contour du visage, les yeux et la bouche, un gris clair (215) pour la couleur du visage et un gris plus foncé (192) pour le fond.

192	192	192	192	192	0	0	0	0	0	192	192	192	192	192
192	192	192	0	0	215	215	215	215	215	0	0	192	192	192
192	192	0	215	215	215	215	215	215	215	215	215	0	192	192
192	0	215	215	215	215	215	215	215	215	215	215	215	0	192
192	0	215	0	215	215	0	215	215	0	0	215	215	0	192
0	215	215	215	0	0	215	215	215	0	0	215	215	215	0
0	215	215	215	215	215	215	215	215	215	215	215	215	215	0
0	215	215	215	215	215	215	215	215	215	215	215	215	215	0
0	215	215	0	215	215	215	215	215	215	215	0	215	215	0
0	215	215	215	0	215	215	215	215	215	0	215	215	215	0
192	0	215	215	215	0	0	0	0	0	215	215	215	0	192
192	0	215	215	215	215	215	215	215	215	215	215	215	0	192
192	192	0	215	215	215	215	215	215	215	215	215	0	192	192
192	192	192	0	0	215	215	215	215	215	0	0	192	192	192
192	192	192	192	192	0	0	0	0	0	192	192	192	192	192

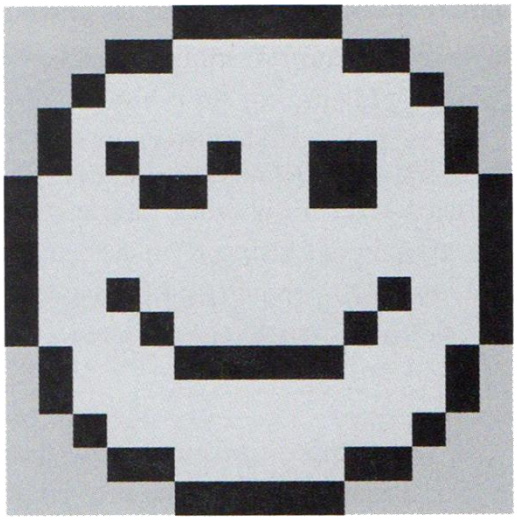


Fig. 5.15 : Image numérisé à niveaux de gris de taille 15 x 15.

La couleur d'un pixel peut être décrite de différentes manières. Les systèmes émettant de la lumière tels que les écrans d'ordinateurs, sont basés sur le principe de la synthèse additive: les

couleurs sont composées d'un mélange de rouge, vert et bleu (modèle RVB). Un pixel est alors codé sur trois octets, un octet par composante, ce qui permet de représenter plus de 16 millions de couleurs différentes. Afin de diminuer la mémoire nécessaire pour manipuler des images en 24 bits, on peut utiliser le mode de représentation en couleurs indexées: cela à limiter le nombre de couleurs différentes utilisées dans chaque image, puis à stocker ces couleurs (codées sur trois octets) dans une table, appelée *palette.* La couleur d'un pixel est alors représentée par son indice dans la palette, qui peut être codé sur un nombre de bits d'autant plus petit que le nombre des couleurs autorisées est faible.

Il existe de nombreuses alternatives au modèle RVB. L'une d'entre elle est le modèle TSL (teinte, saturation, luminance, HSL en anglais) qui est modèle proche de la représentation des couleurs par l'œil humain: la teinte (*hue*) correspond à la couleur proprement dite (rouge, orange etc.) la saturation à sa pureté (vif ou terne), la luminance à sa quantité de lumière (clair ou sombre). Ce modèle permet de définir une couleur de manière plus intuitive qu'avec le modèle RVB: un exemple d'interface permettant de définir une couleur dans l'espace TSL est donné en Fig. 5.16**.** L'axe horizontal de la zone rectangulaire permet de choisir la teinte, l'axe vertical la saturation; la barre verticale sur la droite de la fenêtre est utilisée pour définir la luminance. Comme le montre cette figure, une conversion systématique entre le modèle RVB et le modèle TSL existe.

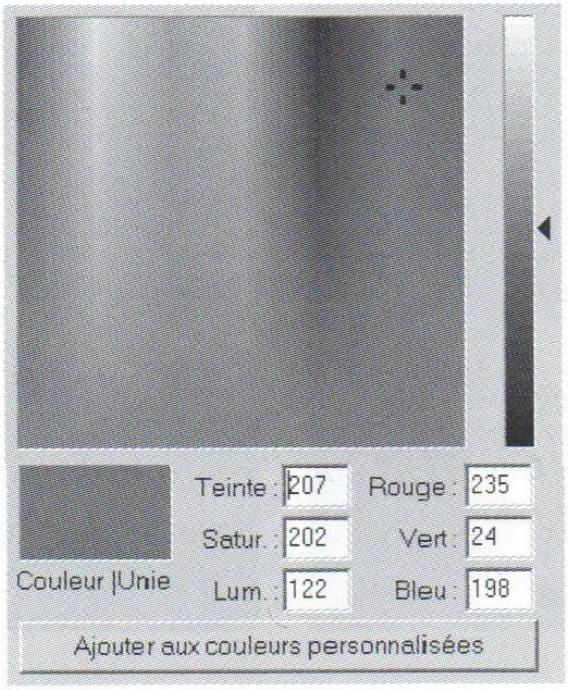


Fig. 5.16 : Exemple d'interface permettant de définir une couleur dans le modèle TSL.

5) Les marqueurs et les fichiers «.pat»:

FLARToolkit repose sur la détection des marqueurs. Les marqueurs ont deux composantes: le marqueur imprimé et son homologue virtuel, un fichier en «.pat» qui est utilisé par FLARToolkit pour reconnaître le marqueur. Nous allons voir ici ces deux composant en détail.

5.1 Les marqueurs :

Comme je l'ai expliqué précédemment les marqueurs sont la base même de la réalité augmentée, c'est grâce à eux que nous pouvons savoir où superposer nos modèles graphiques sur l'image réelle.

Pour créer un marqueur efficace qui sera facilement détecté par FLARToolKit il faut respecter certaines règles.

1. Il doit être de forme carrée.
2. Il doit être entouré par une bande noire, pour ne laisser qu'un carré blanc à l'intérieur.

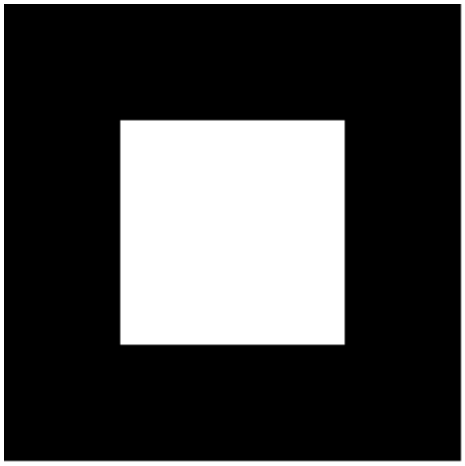


Fig. 5.17 : exemple d'un marqueur de base.

3. La forme dessinée à l'intérieur doit être simple, plus elle sera complexe plus la détection sera difficile.

4. Le marqueur doit être aux niveaux de noir et blanc, cela facilite grandement la détection.

Fig. 5.18 : exemple de marquer respectant les différentes conditions.

5.2 Les fichiers «.pat» :

Ce fichier contient les données du marquer numérisé, comme expliqué précédemment, une suite de chiffres compris entre 0 (noir) et 255 (blanc). Sous quatre angles différents, pour mieux comprendre la chose je vais prendre un exemple.

Tout d'abord comment générer un fichier «.pat», pour cela il existe différentes applications notamment *ARToolKit Marker Generator*, cette application utilise un algorithme de détection de coin pour détecter le marqueur et ainsi générer un fichier «.pat», elle peut être téléchargée à partir de:

http://saqoosha.net/lab/FLARToolKit/MarkerGenerator/MakerGenerator.air

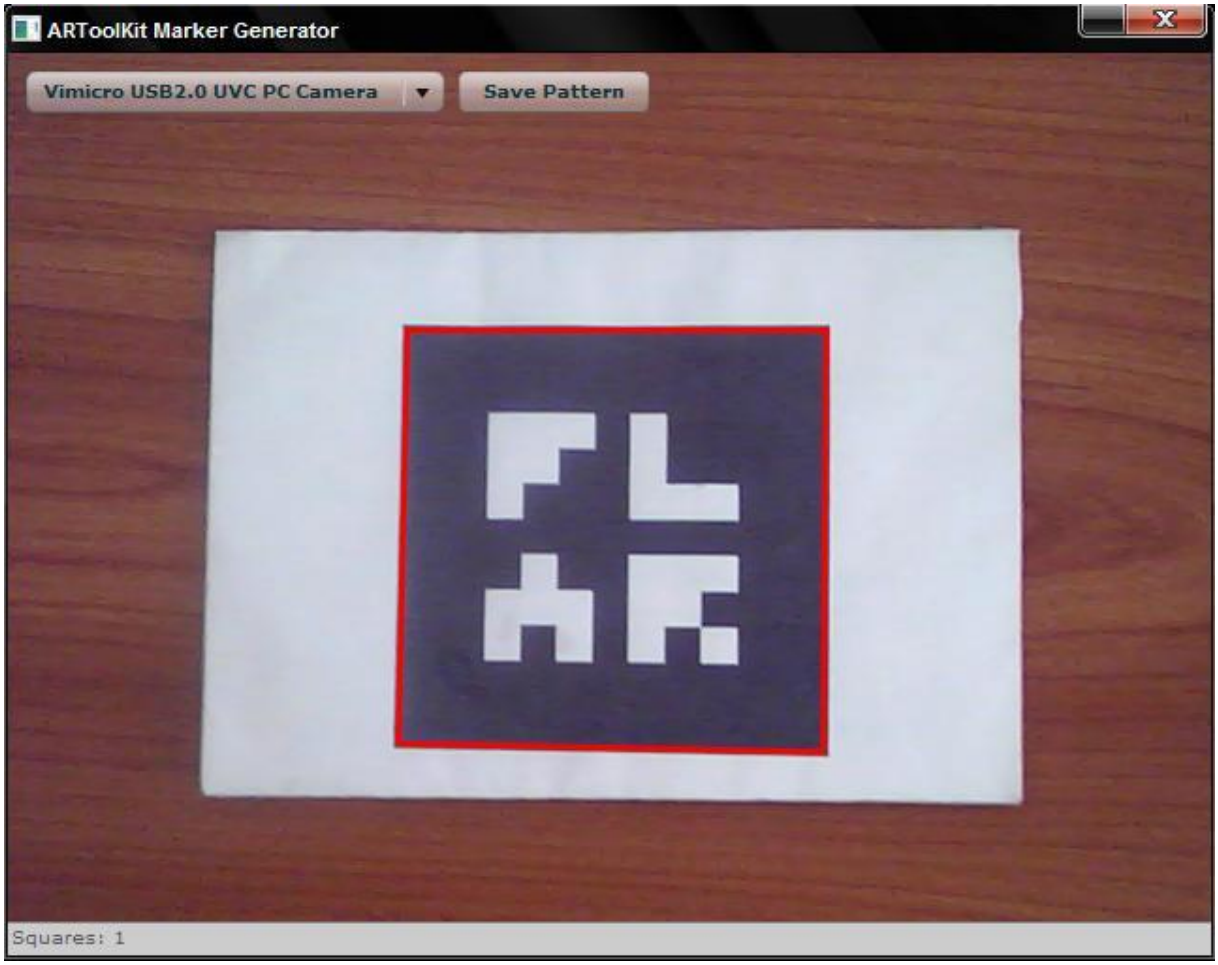


Fig. 5.19 : Exemple d'ARToolKit Marker Generator

Quand un marqueur est détecté un carré rouge apparait, il ne reste plus qu'à enregistrer le ficher pat.

Ce qu'on se demande ici c'est comment fait il pour détecter les marqueurs ? C'est la qu'intervient l'algorithme de détection de coin que j'ai mentionné plus haut.

L'algorithme de détection de coins que je présente dans cette partie a été proposé par Miroslav Trajkovic et Mark Hedley, dans la revue Image and Vision Computing (M. Trajkovic, M. Hedley. « Fast corner detection ». Image and Vision Computing, n° 16, 1998, pages 75-87).

Je l'ai choisi pour sa simplicité et sa rapidité (il est souvent utilisé pour des applications temps réel).

On suppose que l'image est représentée en niveau de gris. Considérons un pixel *p* arbitraire de cette image et un disque au tour de ce pixel. Le point central du disque *p* est un coin s'il est à la jonction de deux demi-droites séparant le disque en deux régions de niveau de gris uniforme (forme (c) ci-dessous). Si *p* est à l'intérieur de l'une des deux régions (forme (a)), ou sur une droite séparant les deux régions (forme (b)), il ne s'agit pas d'un coin.

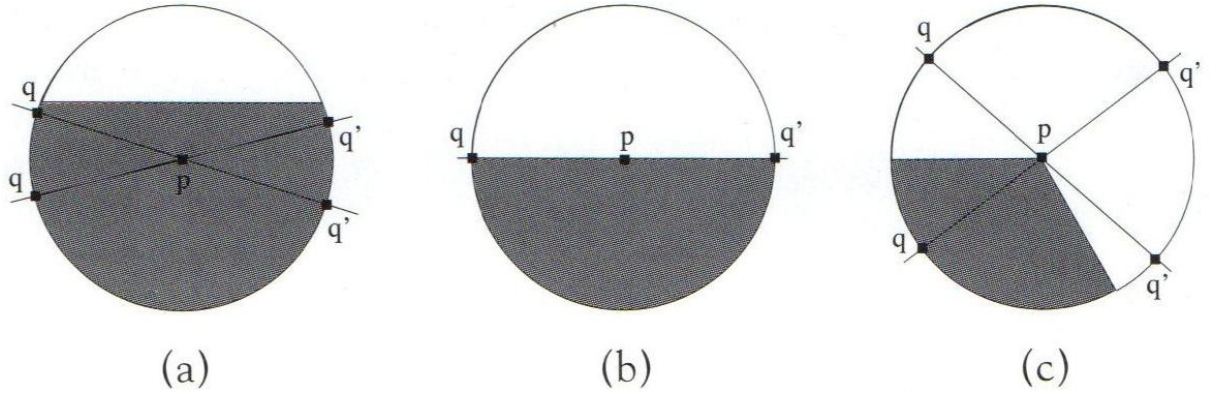


Fig. 5.20 : exemple de point pour la détection de coin.

Le principe de l'algorithme est de faire tourner une droite passant par *p* autour du disque et de mesurer les niveaux de gris I_q et $I_{q'}$ des pixels *q* et *q'* situés à l'intersection de la droite et du bord du disque (voir schéma ci-dessus). Le score du pixel *p* est calculé comme le minimum de la fonction suivante :

$$f(p,q,q') = (I_q - I_p)^2 + (I_{q'} - I_p)^2$$

Soit C le niveau de gris de la région claire, et *S* le niveau de gris de la région sombre. Supposons que le niveau de gris du pixel *p* soit celui de la région sombre ($I_p = S$). Trois cas de figure peuvent se produire :

Cas a: *p* est l'intérieur de la région sombre. Il existe plusieurs droites telles que *q* et *q'* soient à l'intérieur de la région sombre ($I_q = I_{q'} = I_p = S$). Le minimum de la fonction *f* est donc 0;

Cas b: *p* est sur la droite séparant les deux régions. Il existe exactement une droite telle que

$I_q = I_{q'} = I_p = S$ et le minimum de *f* est encore 0.

Cas c: *p* est un coin. Quelle que soit la droite considérée, au moins un des deux pixels *q* ou *q'* est dans la région claire ($I_q = I_{q'} = C$), *f* revoie 2 x$(C-S)^2$. Si l'un des deux pixels seulement est dans la région claire ([$I_q = C$ et $I_{q'} = S$] ou [$I_q = S$ et $I_{q'} = C$]) la fonction renvoie $(C-S)^2$. Le minimum de *f* est donc $(C-S)^2$, qui est d'autant plus grand que *C* est éloigné de *S*.

Nous voyons donc que le minimum de la fonction *f* est grand quand le pixel *p* est un coin, petit dans le cas contraire. C'est donc cette fonction qui est utilisée pour calculer les scores de chaque pixel. Les pixels ayant les scores les plus élevés sont considérés comme étant des coins de l'image.

Reprenant où on était avec le fichier pat, après sauvegarde nous nous retrouvons avec un fichier en «.pat», mais que contient-il en réalité ? La réponse en image.

Comme expliqué au début du chapitre FLARToolkit n'utilise que 50% du marqueur

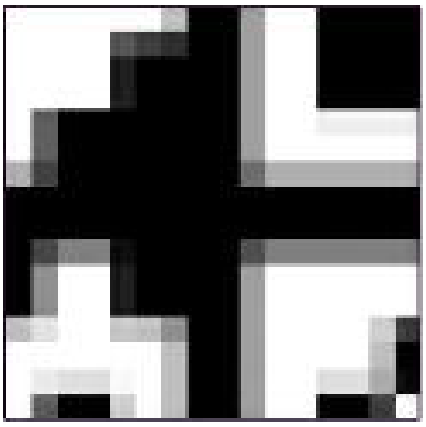


Fig. 5.21 : résultat après traitement de l'exemple pris plus haut.

Maintenant qu'on a notre marqueur on peut générer notre fichier pat qui contiendra ceci.

255 255 255 255 255 255 191 0 0 159 255 255 0 0 0 0'	0 0 0 0 239 255 175 0 0 127 255 255 63 0 15 255'
255 255 255 255 87 63 47 0 0 159 255 255 0 0 0 0'	0 0 0 0 239 255 175 0 0 127 255 255 207 191 171 63'
255 255 255 255 31 0 0 0 0 159 255 255 0 0 0 0'	0 0 0 0 239 255 175 0 0 127 255 255 255 255 223 0'
255 255 255 255 31 0 0 0 0 159 255 255 0 0 0 0'	0 0 0 0 239 255 175 0 0 127 255 255 255 255 223 0'
255 95 0 0 0 0 0 0 0 159 255 255 239 239 239 239'	255 255 255 255 255 255 175 0 0 127 255 255 255 255 255 255'
255 95 0 0 0 0 0 0 0 159 255 255 255 255 255 255'	255 255 255 255 255 255 175 0 0 127 255 255 255 255 255 255'
175 65 0 0 0 0 0 0 0 109 175 175 175 175 175 175'	159 159 159 159 159 159 109 0 0 79 159 159 159 159 159 159'
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0'	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0'
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0'	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0'
0 71 127 127 15 0 0 0 0 79 127 127 127 127 127 127'	191 47 0 0 0 0 0 0 0 0 0 0 143 191 191 191'
0 143 255 255 31 0 0 0 0 159 255 255 255 255 255 255'	255 63 0 0 0 0 0 0 0 0 0 0 191 255 255 255'
0 143 255 255 31 0 0 0 0 159 255 255 255 255 255 255'	255 87 31 31 0 0 0 0 0 15 31 31 199 255 249 207'
191 227 255 255 199 191 143 0 0 159 255 255 255 255 207 63'	255 255 255 255 0 0 0 0 0 127 255 255 255 255 223 0'
255 255 255 255 255 255 191 0 0 159 255 255 255 255 191 0'	255 255 255 255 0 0 0 0 0 127 255 255 255 255 223 0'
255 235 223 223 249 255 191 0 0 159 255 255 223 223 171 15'	255 255 255 255 95 95 65 0 0 71 143 143 227 255 235 95'
255 95 0 0 207 255 191 0 0 159 255 255 0 0 63 255'	255 255 255 255 255 255 175 0 0 0 0 0 191 255 255 255'
255 63 0 0 255 255 159 0 0 191 255 207 0 0 95 255'	255 255 255 191 0 0 0 0 0 175 255 255 255 255 255 255
15 171 223 223 255 255 159 0 0 191 255 249 223 223 235 255'	95 235 255 227 143 143 71 0 0 65 95 95 255 255 255 255
0 191 255 255 255 255 159 0 0 191 255 255 255 255 255 255'	0 223 255 255 255 255 127 0 0 0 0 0 255 255 255 255
63 207 255 255 255 255 159 0 0 143 191 199 255 255 227 191'	0 223 255 255 255 255 127 0 0 0 0 0 255 255 255 255
255 255 255 255 255 255 159 0 0 0 0 31 255 255 143 0'	207 249 255 199 31 31 15 0 0 0 0 0 31 31 87 255
255 255 255 255 255 255 159 0 0 0 0 31 255 255 143 0'	255 255 255 191 0 0 0 0 0 0 0 0 0 0 63 255
127 127 127 127 127 127 79 0 0 0 0 15 127 127 71 0'	191 191 191 143 0 0 0 0 0 0 0 0 0 0 47 191
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0'	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0'	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
175 175 175 175 175 175 109 0 0 0 0 0 0 0 65 175'	159 159 159 159 159 159 79 0 0 109 159 159 159 159 159 159
255 255 255 255 255 255 159 0 0 0 0 0 0 0 95 255'	255 255 255 255 255 255 127 0 0 175 255 255 255 255 255 255
239 239 239 239 255 255 159 0 0 0 0 0 0 0 95 255'	255 255 255 255 255 255 127 0 0 175 255 255 255 255 255 255
0 0 0 0 255 255 159 0 0 0 0 31 255 255 255 255'	0 223 255 255 255 255 127 0 0 175 255 239 0 0 0 0
0 0 0 0 255 255 159 0 0 0 0 31 255 255 255 255'	0 223 255 255 255 255 127 0 0 175 255 239 0 0 0 0
0 0 0 0 255 255 159 0 0 47 63 87 255 255 255 255'	63 171 191 207 255 255 127 0 0 175 255 239 0 0 0 0
0 0 0 0 255 255 159 0 0 191 255 255 255 255 255 255'	255 15 0 63 255 255 127 0 0 175 255 239 0 0 0 0

Fig. 5.22 : contenu du fichier pat.

Ainsi on se retrouve au final avec une représentation numérique du marqueur sous 4 angles différents, de taille 16x16 chacun.

Après chargement du pat dans FLARToolkit le marqueur peut être enfin détecté afin d'être utilisé pour superposer nos modèles dessus.

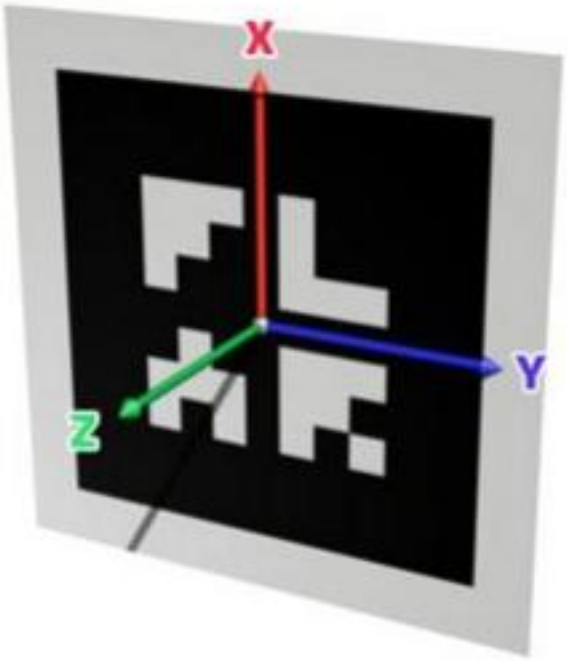


Fig. 5.23 : comment FLARToolKit utilise le marqueur pour superposer les modèles.

6) Conclusion :

Nous venons de comprendre le mécanisme de FLARToolKit, ainsi que l'utilité d'un marqueur, comment les créer et les utiliser, mais il nous reste à voire quelque chose d'indispensable à la réalité augmentée et c'est le moteur graphique, c'est lui qui s'occupe d'affiché nos modèles. Sans plus tarder nous allons voir ça au prochain chapitre.

1) Introduction :

Papervision3D ou PV3D, est un moteur 3D open-source pour Flash. Il permet de simuler un environnement en 3 dimensions à partir d'une animation Flash et de tout ce qu'elle comporte (clips, boutons, photos, vidéos, etc..). Il existe deux versions, l'une en ActionScript 3.0 et l'autre en ActionScript 2.0, mais il est fortement recommandé d'utiliser la version AS3 qui est plus stable et plus avancée. L'utilisation de PV3D dans vos animations les rend gourmandes en ressources et sachez que tous vos utilisateurs n'ont pas la chance d'avoir un ordinateur puissant.

La dernière version de PV3D peut être téléchargée ici :
http://papervision3d.googlecode.com/svn/trunk/

2) De l'ActionScript 2.0 à l'ActionScript 3.0 :

Supposons que vous partiez de très loin...

Rassurez-vous car l'ActionScript 3.0 n'est pas une créature méchante. Son truc à elle, c'est de posséder une manie très particulière qui va altérer vos humeurs durant votre adaptation. A savoir que si vous oubliez de lui fournir le plan d'une seule de ses pattes, elle vous pette dessus et génère une erreur. Autrement dit, n'oubliez pas de lui fournir l'ensemble des plans dont elle a besoin pour se constituer. Autrement dit, il est nécessaire de définir au début de chaque programme l'ensemble des classes dont il a besoin.

Il existe une classe par type d'objet et chacune d'elles contient au moins une fonction lui permettant de générer l'objet (appelons-la la fonction d'initialisation). Ainsi, pour créer un "clip animé", on appelle la fonction d'initialisationMovieClip() définie par la classe MovieClip. De même pour créer un champ de texte, on appelle la fonctionTextField() de la classe TextField (remarquez que les fonctions d'initialisation portent le même nom que leur classe).

Voici donc comment se définissent une classe et sa fonction d'initialisation :

```
public class Nom_de_la_classe
{
	public function Nom_de_la_classe():void
	{
		// Fonction d'initialisation
	}
}
```

Mais il nous faut pousser un peu plus la chose... Rendez-vous compte en vous faisant une idée de ce que représente en ActionScript 3.0 n'importe qu'elle animation faite dans votre passé : Ce sont des objets de la classe Sprite. Que dis-je ! Toute animation Flash est un objet de la classe Sprite. Nous avons donc la possibilité de créer une animation entière à partir d'un simple programme en ActionScript 3.0 et la classe Sprite. Bien sûr, il est nécessaire d'apporter des modifications à cette classe afin qu'elle génère votre animation. On dit alors que la classe Sprite est étendue et elle devient la classe du document.

Voici donc comment se définit la classe d'un document, ainsi que sa fonction d'initialisation :

```
package
{
	// D'abord, importez toutes les classes utiles à votre animation
	// Ici il s'agit simplement de la classe Sprite se trouvant dans le dossier "flash > display"
	// (Ces dossiers sont sur votre ordinateur à partir du moment où vous avez Flash)
	import flash.display.Sprite;
	// Puis, définissez la classe du document (les plans de votre animation)
	// Ici il s'agit de la classe nommée indifféremment "Classe_du_document"
// Classe créée sur la base de la classe Sprite, d'où la traduction littérale :
// je définie la classe "Classe_du_document" qui étend la classe Sprite
public class Classe_du_document extends Sprite
{
	// Enfin, définissez la fonction d'initialisation
	// Cette fonction correspond aux actions exécutées au démarrage de l'animation
	// Elle doit porter le même nom que la classe du document, soit "Classe_du_document"
	public function Classe_du_document():void
	{
		// Vos actions initiales ici...
	}
}
}
```

3) Les principales classes Papervision3D :

Papervision3D n'est autre qu'un ensemble de classes, des plans utiles à Flash pour représenter un environnement en 3 dimensions. Comme pour les classes Sprite et MovieClip de Flash, il suffit d'importer les classes Papervision3D dans vos animations (dans lesdites classe du document) afin de les utiliser.

Ainsi, vos animations nécessitent l'importation d'un certain nombre de classes Papervision3D :

```
package
{
    import flash.display.Sprite;
    import org.papervision3d.scenes.Scene3D;
    import org.papervision3d.cameras.Camera3D;
    import org.papervision3d.objects.Plane;
    import org.papervision3d.materials.ColorMaterial;
    ...
```

Chacune d'elles crée un objet utile à un environnement 3D.
Et comme un réalisateur qui définit les éléments essentiels de son film...

- la scène
- le caméraman
- les acteurs
- les costumes

...un environnement Papervision3D nécessite au moins une scène, un caméraman, un acteur et des costumes :

- la scène : les classes de type scenes
- le caméraman : les classes de type cameras
- les acteurs : les classes de type objects
- les costumes : les classes de type materials

Chacun de ces types contient une série de classes pour générer des objets :

- la scène : les classes de type scenes: Scene3D, MovieScene3D, InteractiveScene3D
- le caméraman : les classes de type cameras : Camera3D, FreeCamera3D
- les acteurs : les classes de type objects : Plane, Cube, Sphere
- les costumes : les classes de type materials : BitmapMaterial, MovieMaterial, ColorMaterial

3.1 La scène : [WEB8]

Une animation Flash classique peut être classifiée comme étant une animation en 2D, c'est-à-dire comme ayant des objets animés par rapport à deux axes, X et Y. L'axe X représentant l'horizontal et l'axe Y la verticale. Ainsi chacune de vos interpolations de mouvement font jouer les coordonnées de vos clips sur ces deux axes. Une animation 3D comporte aussi les axes X et Y, auxquels on ajoute un troisième axe Z pour représenter la 3ème dimension, ou la profondeur. C'est ainsi que l'on obtient un effet de perspective et de 3D :

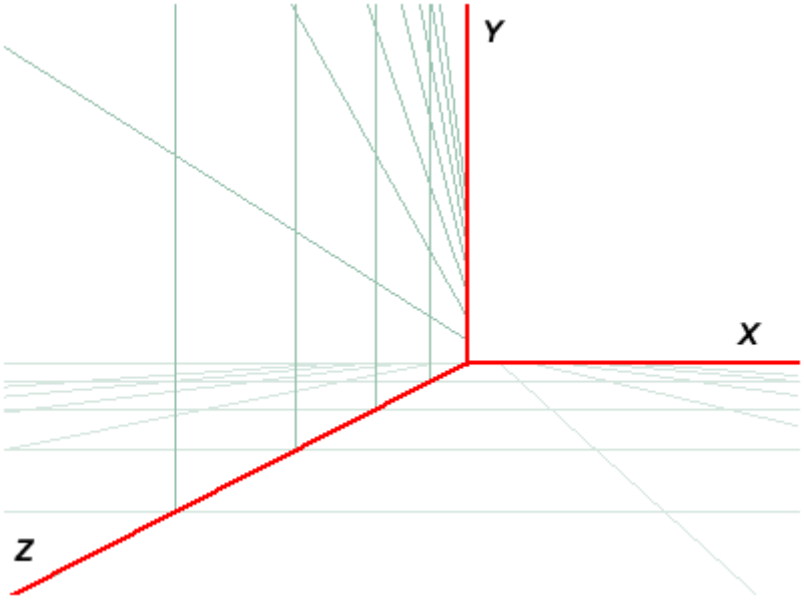


Fig. 6.1 : effet de perspective et de 3D.

Une scène *Papervision3D* agît comme un repère à trois dimensions. C'est un objet destiné à contenir d'autres objets (les acteurs) afin de les positionner sur trois axes X, Y, et Z. Mais pour fonctionner une scène a besoin d'un conteneur, un objet qui va lui permettre d'accueillir des acteurs. Cet objet doit être de la classe *Sprite*.

Voici donc comment se définissent une scène *Papervision3D* et son conteneur *Sprite* :

```
// Création d'un conteneur Sprite
    var container :Sprite;
    container = new Sprite();
// Ajout du conteneur sur la scène de l'animation Flash
    addChild( container );
    container.x = stage.stageWidth / 2;
    container.y = stage.stageHeight / 2;
// Création d'une scène Scene3D à partir du conteneur
    var scene :Scene3D;
    scene = new Scene3D( container );
```

3.2 Le caméraman : [WEB8]

Pour se représenter une scène, *Papervision3D* ou pas, vous devez définir un observateur, l'endroit où il se situe et dans quelle direction il regarde. Ce sont les objets de type *cameras* qui remplissent cette fonction. Il en existe deux différents, ***Camera3D*** et ***FreeCamera3D*** :

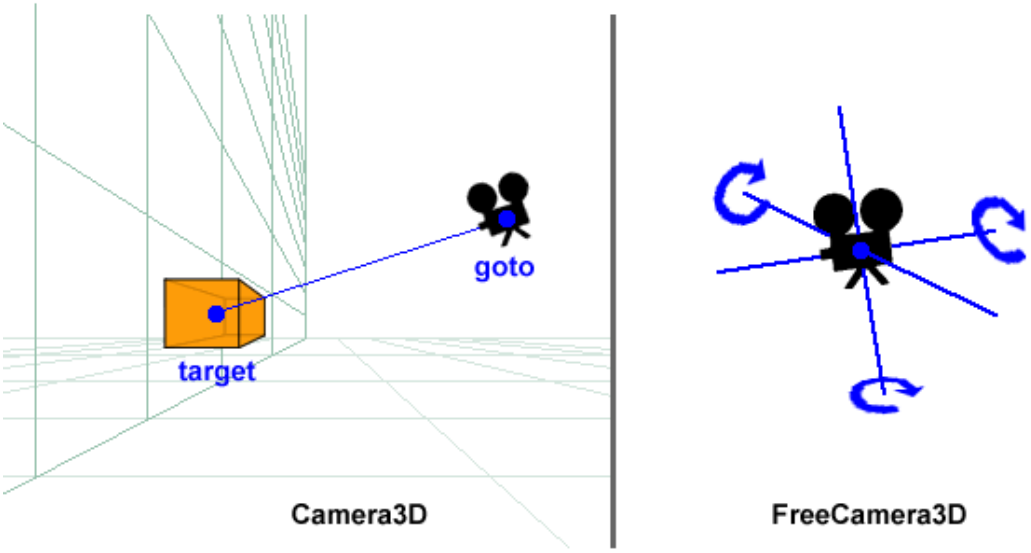


Fig. 6.2 : Schéma de *Camera3D* et *FreeCamera3D*.

L'objet *FreeCamera3D* se définit avec 6 propriétés principales: les coordonnées x, y et z de la caméra, ainsi que les rotations sur les axes X, Y et Z. Le paramétrage d'un tel objet est assez complexe lorsque l'on débute, c'est pourquoi nous utiliserons plutôt l'objet *Camera3D*.

L'objet *Camera3D* se définit aussi avec 6 propriétés principales: les coordonnées x, y et z des deux objets ***goto*** et ***target***. Ce sont des objets de la classe *DisplayObject3D* qui n'ont pour fonction que de représenter la position d'un point dans la scène 3D. En l'occurrence la position de la caméra et celle de sa cible. Les rotations de la caméra sont calculées automatiquement.

Voici donc comment se définit un objet Camera3D :

```
// Création d'un objet Camera3D
    var camera :Camera3D;
    camera = new Camera3D();


// Définition des objets goto et target
    var gotoObject :DisplayObject3D = new DisplayObject3D();
    gotoObject.x = 0;
    gotoObject.y = 0;
    gotoObject.z = 0;
    var targetObject :DisplayObject3D = new DisplayObject3D();
    targetObject.x = 200;
    targetObject.y = 200;
    targetObject.z = 200;
```

```
// Paramétrage de l'objet Camera3D
    camera.extra =
    {
    goPosition: gotoObject,
    goTarget: targetObject
    };
```

3.3 Les acteurs : [WEB8]

Voici venus les objets de type *objects* : **Plane**, **Cube** et **Sphere**. Ce sont les acteurs d'une scène *Papervision3D*. Comme leur nom l'indique, ils vous permettent de créer des surfaces, des cubes ou encore des sphères.

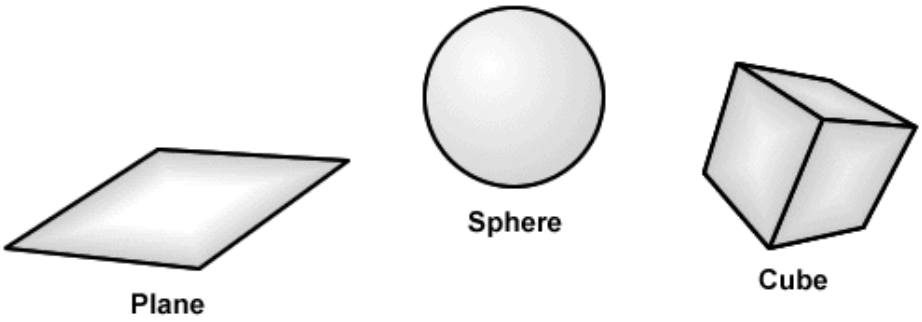


Fig. 6. 3 : exemple de primitive de type objects.

Pour les personnaliser et leur donner des textures, on les utilise conjointement avec des objets de typematerials, les costumes. Par exemple, pour donner de la couleur à une surface on utilise l'objet ColorMaterial. Ou encore pour afficher le contenu d'un clip MovieClip on utilise l'objet MovieMaterial.

Ainsi, une surface Plane de couleur 0x0000FF et de dimensions 480x320 px se définit comme suit :

```
// Création d'un objet de texture ColorMaterial
    var color :ColorMaterial;
    color = new ColorMaterial( 0x0000FF );

// Création de l'objet Plane
    var plane :Plane = new Plane( color, 480, 320 );

// Ajout de l'objet Plane dans une scène Papervision3D
    scene.addChild( plane, "myPlane" );
```

4) Fonctionnement d'un environnement Papervision3D :

Voyons comment mettre en relation les classes *Papervision3D* vues précédemment :

- une scène : Scene3D
- un caméraman : Camera3D
- un acteur : Plane, sphere, cube, Objet 3d (dea,3ds,md2).
- un costume : ColorMaterial, BitmapFileMaterial

En *Papervision3D*, une scène 3D se construit selon une arborescence bien précise, comme dans quasiment tous les moteurs graphiques que l'on appelle un graphe. Un graphe est constitué de nœuds (*node* en anglais) qui sont reliés entre eux par une relation parent-enfant (*parent-child*) et qui forment une structure appelée arbre.

Il y a deux sortes de nœuds : le nœud groupe (*DisplayObject3D*) qui peut avoir un ou plusieurs enfants mais seulement un parent, et les feuilles (plane, sphere, cube…etc) qui n'ont qu'un seul parent et pas d'enfants. C'est la méthode addChild(Objet nœud, String nom_du_ nœud) de la classe *DisplayObject3D* qui permet d'ajouter un enfant à un groupe.

On retiendra que les nœuds, quels qu'ils soient, ne peuvent avoir qu'un seul et unique parent.

Montrons maintenant la base d'une application papervision en utilisant toues ces classes.

```
package {
    import org.papervision3d.scenes.Scene3D;
    import org.papervision3d.view.Viewport3D;
    import org.papervision3d.cameras.Camera3D;
    import org.papervision3d.render.BasicRenderEngine;
    import org.papervision3d.objects.DisplayObject3D;
    import flash.display.*;
    import flash.events.*;
    import org.papervision3d.objects.parsers.DAE;

    public class Main extends Sprite {
            private var scene:Scene3D;
            private var viewport:Viewport3D;
            private var camera:Camera3D;
            private var renderer:BasicRenderEngine;
            private var deaObj : DisplayObject3D;
            private var dae:DAE;

            public function Main():void {
                    initialisationList();
            }
            private function initialisationList():void {
                    initPV3D();
                    initDae();
                    initListeners();
            }
```

```
        private function initPV3D():void {
                scene = new Scene3D();
                viewport = new Viewport3D();
                addChild( viewport );
                camera = new Camera3D();
                renderer = new BasicRenderEngine();
        }
        private function render(e:Event):void{
                dae.rotationY ++;
                 renderer.renderScene( scene, camera, viewport );
        }
        private function initDae():void{
                deaObj = new DisplayObject3D();
                var bitmapFileMaterial:BitmapFileMaterial = new
                BitmapFileMaterial("demonman_red.jpg",true);
                dae = new DAE();
                dae.load("model.dae", new MaterialsList({all:BitmapFileMaterial}));
                dae.scale = 35; //Redimensionne l'objet
                deaObj.addChild(dae);
                scene.addChild(deaObj);
        }
        private function initListeners():void{
                addEventListener(Event.ENTER_FRAME, render);
        }

    }
}
```

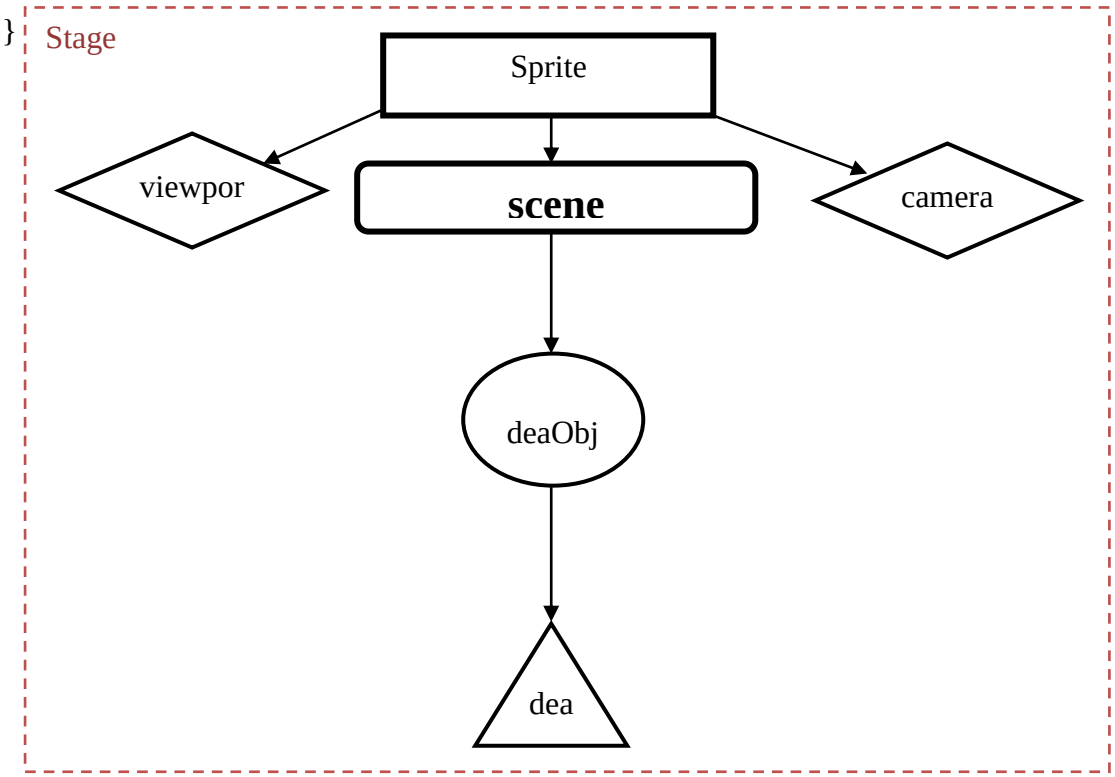


Fig. 6.4 : Arborescence du code ci-dessus sur PV3D.

Fig. 6.5 : résultat de la compilation du code ci-dessus.

- La classe ***Sprite*** est un bloc constitutif de base de la liste d'affichage : un nœud de liste d'affichage qui permet d'afficher des images et peut également contenir des enfants.
- La classe ***Stage*** représente la zone de dessin principale. Pour le contenu SWF s'exécutant dans le navigateur (dans Flash® Player), la scène représente la zone entière où le contenu Flash est affiché.
- ***Le viewport*** doit être ajouté à la scène via addChild() car c'est l'objet qui va s'occuper de rendre à l'écran le résultat de l'affichage de la scène.
- ***renderer*** est de type BasicRenderScene, c'est l'objet qui effectue les traitements nécessaires et les affichent dans le viewport.
- La variable ***scene*** quand à elle va accueillir les objets 3d crées via Papervision. Nous y ajouterons nos objets 3d via scene.addChild().
- Et enfin ***camera***, qui correspond à la caméra 3d, qui se charge de définir la zone de l'espace 3d qu'il faut rendre à l'écran.

5) Conclusion :

Nous venons de voir Papervision3D, un parmi plusieurs autres moteurs graphiques disponibles pour FLASH, mais à mon avis je pense que Papervision3D reste l'un des meilleur.

Maintenant qu'on a terminé avec FLARToolKit et Papervision3D, on peut enfin passer au cœur du mémoire qui consiste aux raisons qui m'on poussé à choisir ce thème et aussi pourquoi j'ai choisi ce cas d'étude. Bien sûr tout ceci sera expliqué au prochain chapitre.

1) Introduction :

J’ai choisi le thème de la réalité augmentée car je suis fasciné par l’univers du multimédia, sur tous tout ce qui a attrait à la 3D.

Quand au cas de l’interaction entre marqueurs, cette idée m'est venue en cherchant des informations sur la réalité augmentée sur internet, tous ce que j'ai trouvé c’était des vidéos et des images des personnes qui mettaient des modèle 3D sur leur marqueurs, pas d’animation ni d’interaction et c’est la que je me suis posé la question, était t’il possible d’interagir avec différents marqueurs en temps réel tout en animant nos modèles 3D ? Et donc je me suis lancé dans ce cas pour répondre à cette question.

2) Positionnement et rotation :

Le premier problème qui c’est posé est le positionnement, comment savoir où était les marqueurs les uns par rapport aux autres, rien de plus simple j’ai relié tous les nœuds marqueurs au même nœud parent ainsi la position des marqueurs était tous par rapport à un même axe parent.

La rotation c’était avérée une tache plus ardue que je ne l'est pensé, mais au final j’ai trouvé la solution en utilisant le théorème *d'Al-Kashi*, c’est un théorème de géométrie couramment utilisé en trigonométrie qui relie dans un triangle la longueur d'un côté à celles des deux autres et au cosinus de l'angle formé par ces deux côtés.

$$c^2 = a^2 + b^2 - 2ab\cos\gamma$$

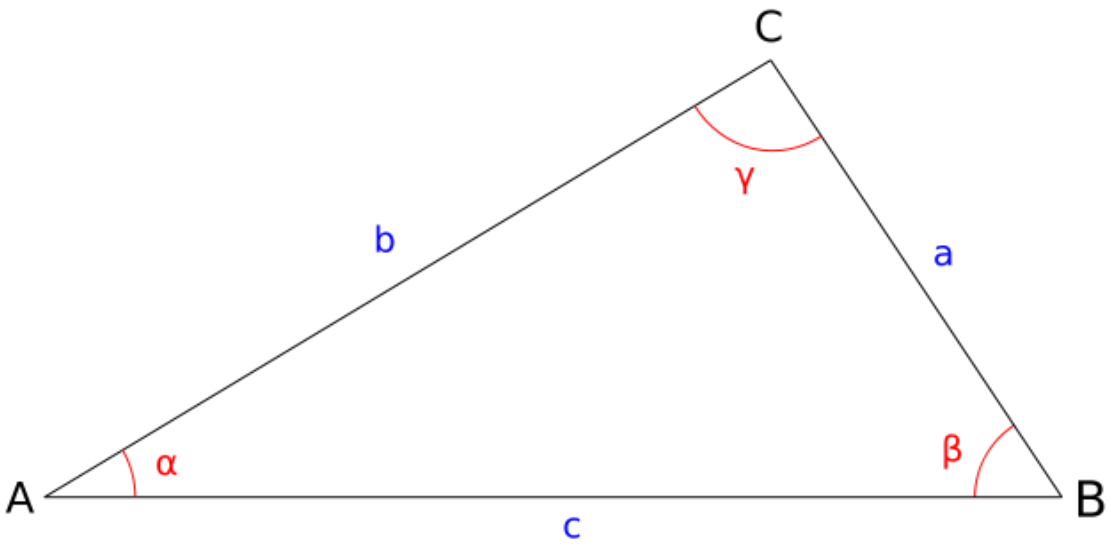


Fig. 7.1 : Notations usuelles dans un triangle quelconque.

Je vais vous expliquer comment j'ai utilisé ce théorème, tout d'abord j'ai relié mon personnage 3D à un cube qui se trouve en face de lui et qui symbolise la direction de sa vue avec l'objet lui-même et l'objet sur le marqueur. Nous nous retrouvons avec trois points qui forment un triangle. Ensuite vient un autre problème, comment connaitre la distance entre c'est trois points?. Il faut savoir que la position d'un objet est toujours par rapport à l'axe de sont nœud parent et comme mon personnage et l'objet sur le marqueur, sont dans deux nœud différent, il m'est impossible de trouver leur positions, chaque objet étant relié à un nœud marqueur qui est lui-même relié au nœud père.

Heureusement pour moi Papervision crée deux matrices de transformation pour chaque objet de la scène, une qui est reliée au nœud parent et une autre qui est reliée à l'univers, grâce à ça je connaissais le positionnement de tout mes points, il ne me restait plus qu'a connaitre la distance entre ces différent point pour utiliser le théorème *d'Al-Kashi*, chose facile, j'ai utilisé une simple formule mathématique.

Soit A(x, y) et B(x', y') deux points dans l'espace, on veut trouver la distance entre ces deux points.

$$Distance = \sqrt{(x - x')^2 + (y - y')^2}$$

Certains d'entre vous me dirons pour quoi ne pas avoir utilisé la matrice univers pour le positionnement?. Et bien je dirais que cette matrice ne peut être uniquement lue, c'est un problème que le créateur de Papervision n'a pas encore réglé.

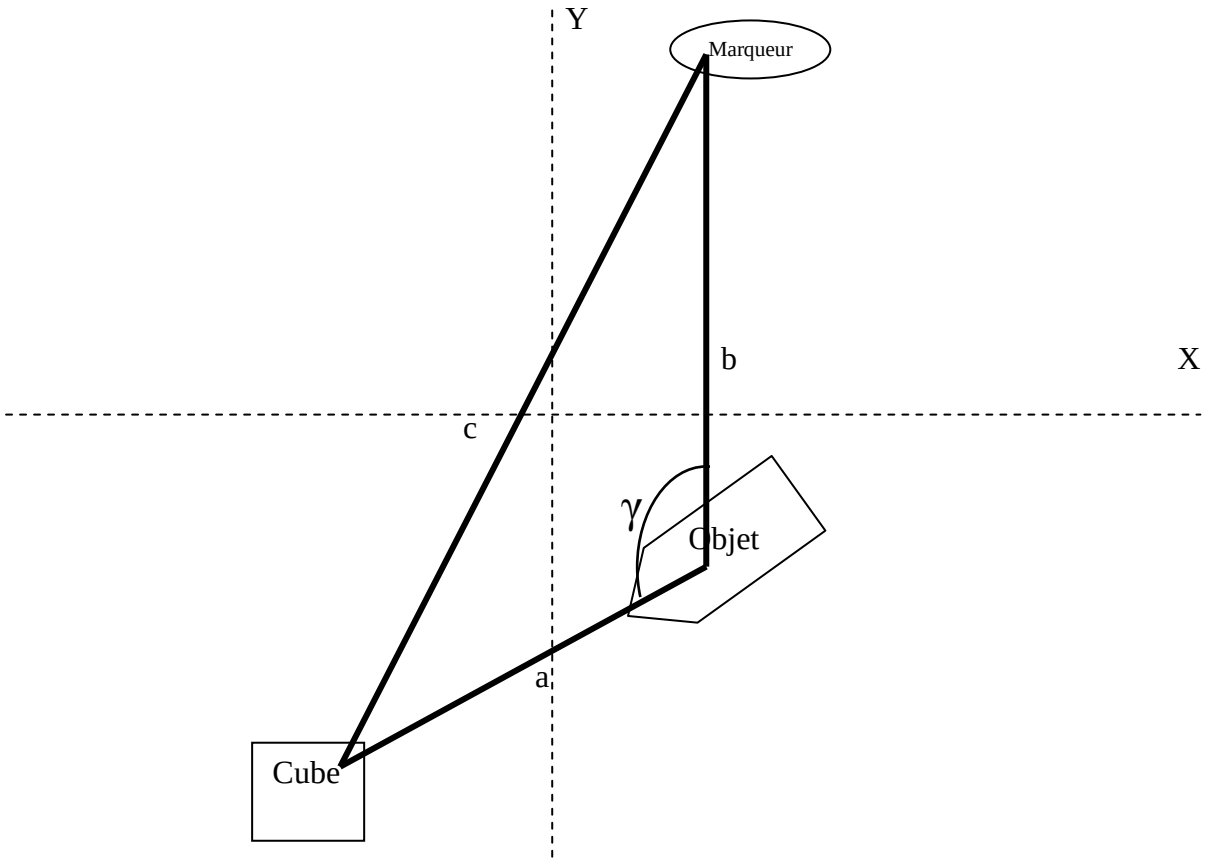


Fig. 7.2 : Exemple de positionnement d'objets sur la scène qui forme un triangle.

Maintenant il ne reste plus qu'a trouver l'angle de γ grâce à cette formule.

On a l'Objet J(x,y), Cube C(x',y'), Marquer M(x'',y'').

$$a = \sqrt{(x - x')^2 + (y - y')^2} \qquad b = \sqrt{(x - x'')^2 + (y - y'')^2} \qquad c = \sqrt{(x' - x'')^2 + (y' - y'')^2}$$

$$\gamma = \left(\arccos\frac{a^2 + b^2 - c^2}{2ab}\right)\frac{180}{\pi}$$

Mais n'oublions pas que nous somme dans un univers en 3D, il faut aussi compter avec l'axe Z et donc après rotation on obtient ceci.

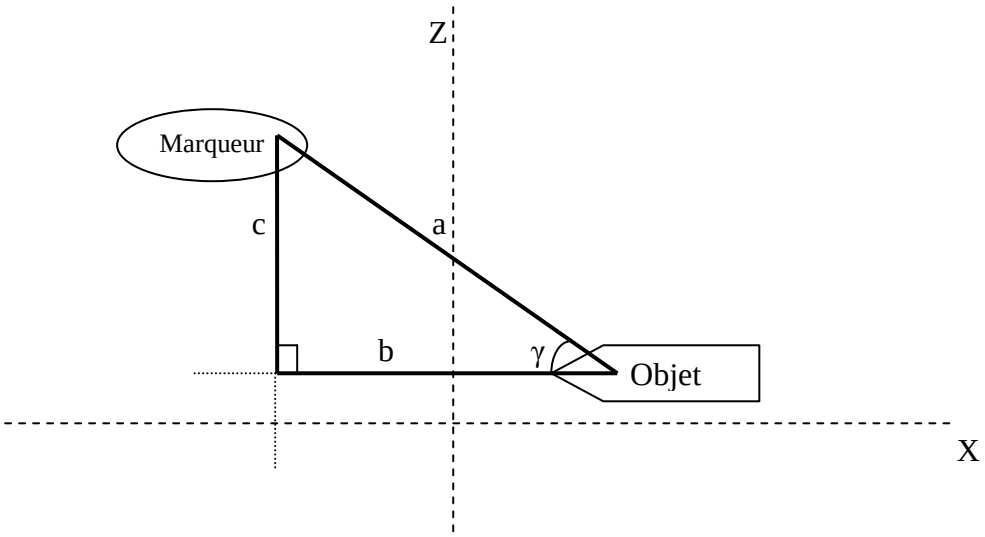


Fig. 7.3 : Exemple de positionnement d'objets sur la scène qui forme un triangle droit.

On refait la même chose, sauf que cette fois comme on est sûr que le triangle est droit grâce à l'intersection des *X*, le calcul sera plus simple, les distances a, b et c se calculent de la même manière et donc au final on aura cette formule.

$$\gamma = \left(\arccos\frac{b}{a}\right)\frac{180}{\pi}$$

3) L'animation :

Maintenant qu'on a la position et l'angle de rotation, il faut animer tous ça, mais tout d'abord on doit connaitre la *fps* (frame per second), le nombre d'images par second est un concept très important dans l'animation, c'est ce qui détermine si une animation est fluide ou pas, dans la majeur partie des applications la *fps* choisie est de 30 mais dans des applications multimédias où on a un univers en 3D la *fps* doit être de 60.

La *fps* a aussi un deuxième rôle à jouer, c'est celui de contrôler la vitesse de l'animation, il faut savoir que ce n'est pas tout le monde qui a un PC puissant, la même application peut tourner sur plusieurs ordinateur mais a des *fps* différentes et donc il faut faire en sorte de garder le même temps d'animation pour chaque ordinateur.

3.1 Calcul du *fps* d'une animation : [WEB5]

Parfois, avec Flash, il faut savoir réinventer la roue. Par exemple, lorsque l'on veut coder un moteur d'animation qui s'intègre au sein d'un système de rendu isométrique complexe, on aimerait à la foi pouvoir disposer d'une *fps* le plus élevé possible, tout en jouant les animations à la *fps* dans lequel elles ont été réalisées.

Pour se faire, il faut obligatoirement passer par l'indispensable case du calcul de la *fps* en cours. Il existe peut-être des mondes merveilleux ou la valeur que l'on a entrée dans le champ « Nombre d'images par secondes » de l'IDE Flash est tout le temps vraie, mais il ne s'agit pas du notre.

Voyons différentes approches pour calculer une *fps*.

3.1.1 La méthode couplée : Timer et ENTER_FRAME

On initialise un compteur et un timer

```
private var _framesCount : uint;
private var _fpsUpdater : Timer = new Timer(1000);
_fpsUpdater.addEventListener(TimerEvent.TIMER,onTimer);
private function onTimer(e : TimerEvent) : void
{
        _framesCount = 0;
}
```

On ajout un au conteur à chaque frame

```
stage.addEventListener(Event.ENTER_FRAME,onEnterFrame);
private function onEnterFrame(e : Event) : void
{
        _framesCount ++;
}
```

L'avantage de cette méthode, c'est qu'elle donne une valeur aussi réaliste que possible. Le désavantage, c'est qu'elle est couteuse en performances (puisque l'on utilise un ENTER_FRAME, qui est toujours un évènement qu'il vaut mieux éviter quand c'est possible – même si c'est rarement possible, surtout dans ce cas), et en plus, notre valeur est mise à jour une fois par seconde seulement et n'est pas disponible durant la première seconde d'écoulement de l'application.

3.1.2 Méthode de l'estimation en fonction de l'écart :

On lance l'évènement ENTER_FRAME

```
stage.addEventListener(Event.ENTER_FRAME,onEnterFrame);
private function onEnterFrame(e : Event) : void
{
        var now : uint = getTimer();
        if(_lastFrame != -1)
        _fps = 1000 / (now - _lastFrame);
        _lastFrame = now;
}
```

Cette méthode permet d'avoir un FPS mis à jour image par image, disponible dès la deuxième image de l'animation. Désavantage : on fait une opération relativement complexe sur un évènement ENTER_FRAME.

3.1.3 Méthode du calcul de la FPS global depuis le début de l'animation :

On lance l'évènement ENTER_FRAME

```
stage.addEventListener(Event.ENTER_FRAME,onEnterFrame);
private function onEnterFrame(e : Event) : void
{
        _framesCount ++;
      //On ne met à jour qu'une fois chaque 50 frames pour prendre moins de temps

        if(_framesCount % 50 == 0) {
                _fps = _framesCount /(getTimer() / 1000);
        }
}
```

Cette méthode est relativement peu couteuse (quoique dépendante du taux de rafraichissement désiré pour la valeur de FPS), mais n'a une précision que très limitée, puisque son temps d'adaptation est proportionnel au nombre d'images déjà jouées. Elle peut être intéressante pour des benchmarks très courts, tant que l'on garde cette méthode d'évaluation pour toutes les mesures, et que l'on fasse chaque mesure dans un temps prédéterminé égal.

3.1.4 La méthode de la mesure sur une période sans timer :

// Temps en miliseconde entre deux mises à jour du compteur

```
private const REFRESH_DELAY : uint = 5000;
private var _lastThreshold : uint = 0;
private var _framesCount : uint;
```

// On lance l'évènement ENTER_FRAME

```
stage.addEventListener(Event.ENTER_FRAME,onEnterFrame);
private function onEnterFrame(e : Event) : void
{
        _framesCount ++;
        var now : uint = getTimer();
        var delay : uint = now - _lastThreshold;
        if(delay > REFRESH_DELAY) {
                _fps = _framesCount / (delay /1000);
                _framesCount = 0;
                _lastThreshold = now;
        }
}
```

Cette méthode me parait être la moins gourmande, et garde une bonne précision. En plus, elle a l'indéniable avantage d'être personnalisable : plus le délai de rafraichissement est long, moins souvent l'on fait un calcul « complexe » (la relativité est une chose merveilleuse). C'est la méthode pour laquelle j'ai finalement opté.

3.2 Intégrer la *fps* dans une animation :

Il faut savoir que chaque bout d'une animation est joué dans une frame jusqu'a la complète lecture de l'animation, et le but est de connaitre la taille de ce bout d'animation pour que le temps d'animation soit le même d'un PC à un autre, c'est ce que je vais expliqué ici.

On prend par exemple un personnage *P* qui se déplace d'un point *A* vers un point *B* à une vitesse *V*, on voudrait savoir de combien il doit avancer à chaque frame pour avoir un même temps d'animation sur chaque machine.

Avant toute chose on calcule le temps d'animation, chose simple :

$$T = V/D$$

T : temps *V* : vitesse *D* : distance entre *A* et *B*

$$Dpf = D / (T * fps)$$

Dpf : déplacement par frame *fps* : nombre d'image par seconde

Voila nous avons trouvé de combien il fallait déplacer notre personnage à chaque frame pour avoir le même temps d'animation pour chaque PC.

4) Conclusion :

Nous venons de passer en revue tout ce que j'ai découvert en matière d'interaction entre marqueurs, positionnement, rotation et animation. Maintenant nous allons passer au prochain chapitre qui va traiter de l'application en elle-même.

1) Introduction :

Maintenant que j'ai terminé de présenter les différents problèmes rencontrés quand on veut réaliser une interaction entre marqueurs et la manière dont je les ai résolus, on peut passer à un exemple pratique, dans ce but j'ai crée deux application qui montre belle et bien de façon visuelle qu'une interaction entre marqueurs était tout à fait possible.

La première application montre une interaction entre plusieurs marqueurs et la seconde une interaction entre le modèle 3D et son propre marqueur, mais avant ça parlons un peut de l'environnement de développement.

2) L'environnement de développement :

2.1 Présentation de FlashDevelop :

FlashDevelop est un éditeur de texte, entièrement gratuit, pour environnement Microsoft Windows, basé sur le framework .NET 2.0. Il supporte les langages ActionScript 2 et 3, MXML et haXe. Il inclut des fonctionnalités de complètement et de coloration syntaxique.

FlashDevelop a été développé en 2005 afin de proposer une alternative gratuite à Adobe Flash. Ainsi beaucoup de gens ont pu débuter gratuitement en Flash. Depuis qu'il a été créé, FlashDevelop ne cesse d'être amélioré. De nouvelles versions sortent tous les 2-3 mois.

2.2 SDK Flex :

Flex est une solution de développement créée par Macromedia en 2004 puis reprise par Adobe en 2006, permettant de créer et de déployer des applications Internet riches (RIA) multiplate-formes grâce à la technologie Flash et particulièrement son lecteur. Son modèle de programmation fait appel à MXML (basé sur XML) et ActionScript 3.0, reposant sur ECMAScript.

La technologie Flex produit un fichier .swf intégré dans une page html. La richesse de l'interface graphique ainsi générée présente l'inconvénient, comme toute applet, de générer ici un fichier .swf sur le serveur un peu long à télécharger sur le poste client lors du chargement de la page.

Le 26 avril 2007, Adobe annonçait choisir la licence libre MPL 1.1 pour sa solution de développement Flex1. Adobe Flash Player, le lecteur multimédia sur lequel les applications Flex sont lues, et Adobe Flex Builder, l'IDE construit sur la plate-forme libre Eclipse utilisé pour développer des applications Flex, restent propriétaires.

Le 17 novembre 2011, Adobe place Flex sous l'égide de la Fondation Apache2.

3) Les outils utilisés dans le travail :

- **Adobe Photoshop CS5 :** logiciel de traitements d'images utile pour créer des texture.
- **Colladamax :** plugin pour 3ds Max 2009 perme d'exporté les modèles en forma COLLADA (.dae).
- **DAEMC2 :** lis les animations contenues dans un fichier COLLADA.

- **Autodesk 3ds Max 2009:** c'est un logiciel de modélisation et d'animation 3D, développé par la société Autodesk. Avec Maya, Softimage XSI, Lightwave, Houdini et Blender, il est l'un des logiciels de référence dans le domaine de l'infographie 3D.
  Le logiciel est issu du programme 3D Studio qui a tourné sous DOS jusqu'à sa version 4. Les programmeurs de Kinetix (une division d'Autodesk rebaptisée maintenant Autodesk Media and Entertainment) ont développé un logiciel entièrement nouveau et repensé.
  3dsmax est ainsi conçu sur une architecture modulaire et supporte des plug-ins (extensions), ainsi que les scripts écrits dans un langage propriétaire (maxscript). Le logiciel s'est développé rapidement, en étant utilisé principalement dans le cadre du jeu vidéo.

4) Description des applications :

La première application consiste à créer un jeu Pacman en réalité augmentée, le principe est de placer les différents marqueurs sur la scène, chaque marqueur représente un fruit et tous ces fruits sont reliés par des boulles blanches, à la fin on pose le marqueur qui représente Pacman et l'interaction est lancée. Pacman se déplace entre les différents marqueurs en mangeant les boulles blanches et les fruits puis revient à son marqueur d'origine.

La seconde application représente un dragon qui suit son marqueur, le principe est de déplacer le marqueur du dragon, puis le dragon doit repérer où est son marqueur sur la scène et de se déplacer vers ce dernier.

J'ai testé mes deux applications sur une machine ayant les performances suivantes :

- Processeur : Intel Pentium I3 2,53 GH_z
- RAM : 3 Go
- Carte graphique : Intel HD Graphics 1Go
- DirectX9

5) Quelques captures d'écran :

On commence par la première application réalisée avec FLARManager et on finit par la deuxième réalisée avec FLARToolKit.

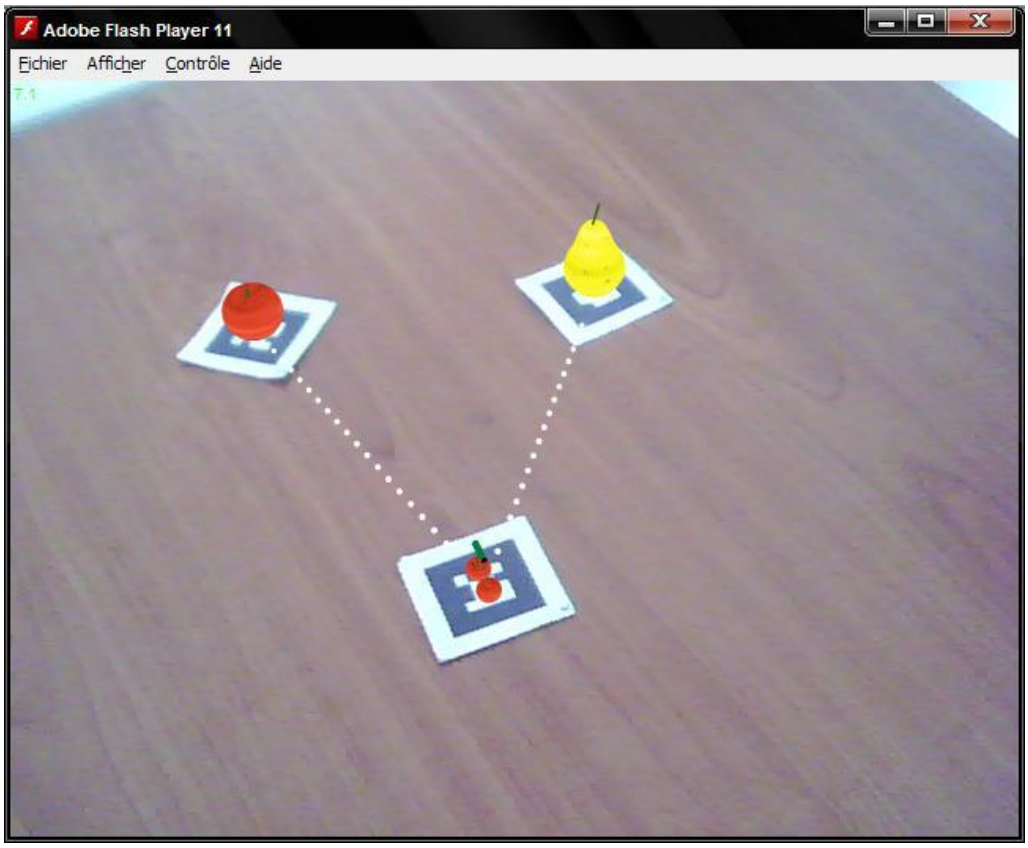


Fig. 8.1 : On pose les marqueurs pour faire un chemin.

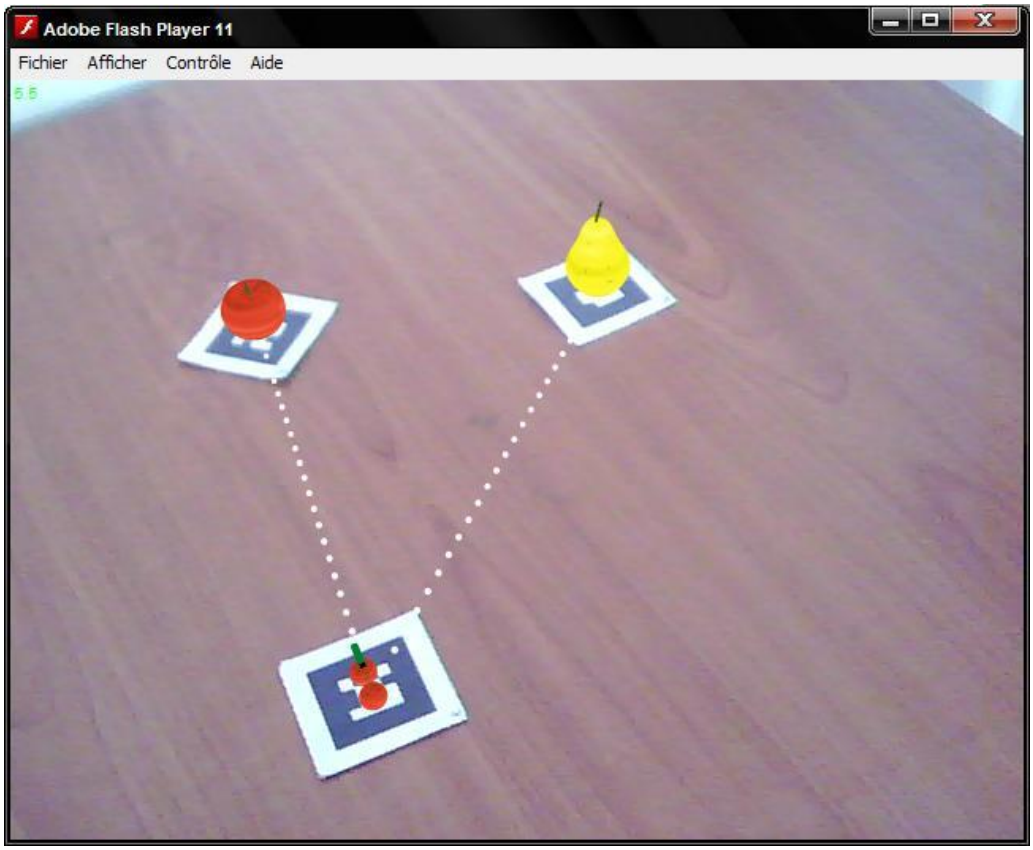


Fig. 8.2 : Faire déplacer un marqueur fait aussi déplacer les boulles blanches, ce qui démontre l'interaction en temps réel.

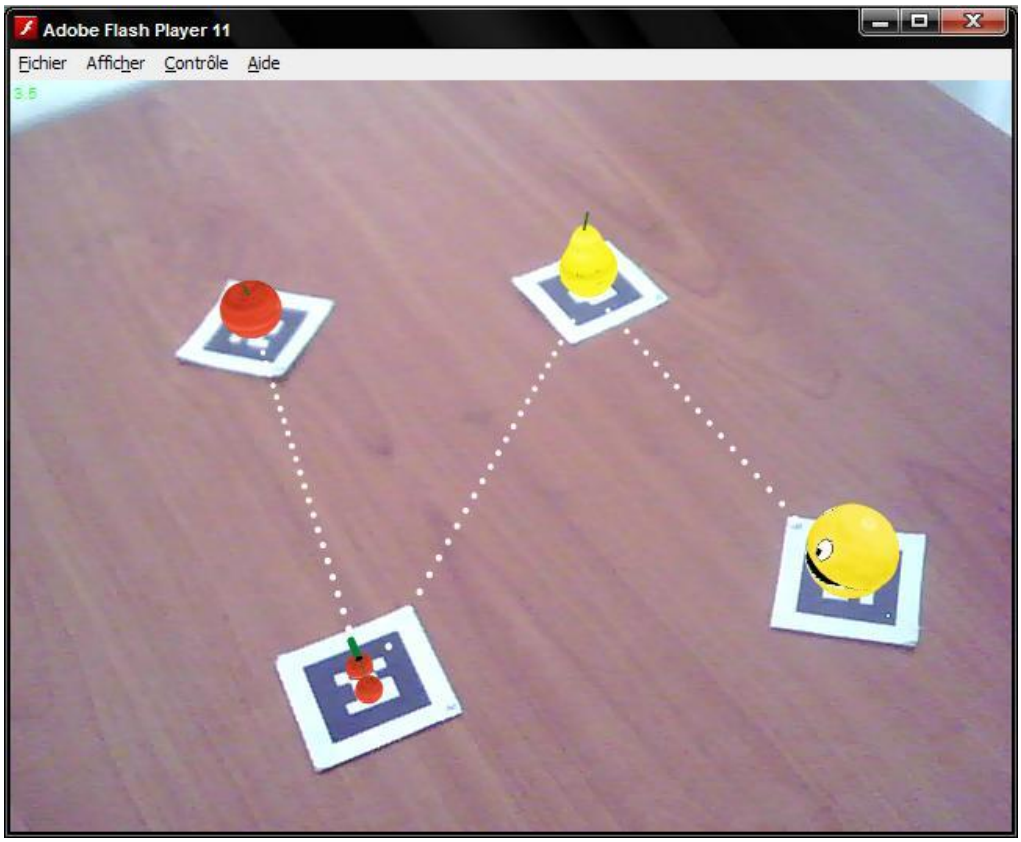


Fig. 8.2 : On pose le modèle 3D qui va subir l'interaction et l'animation peut commencer.

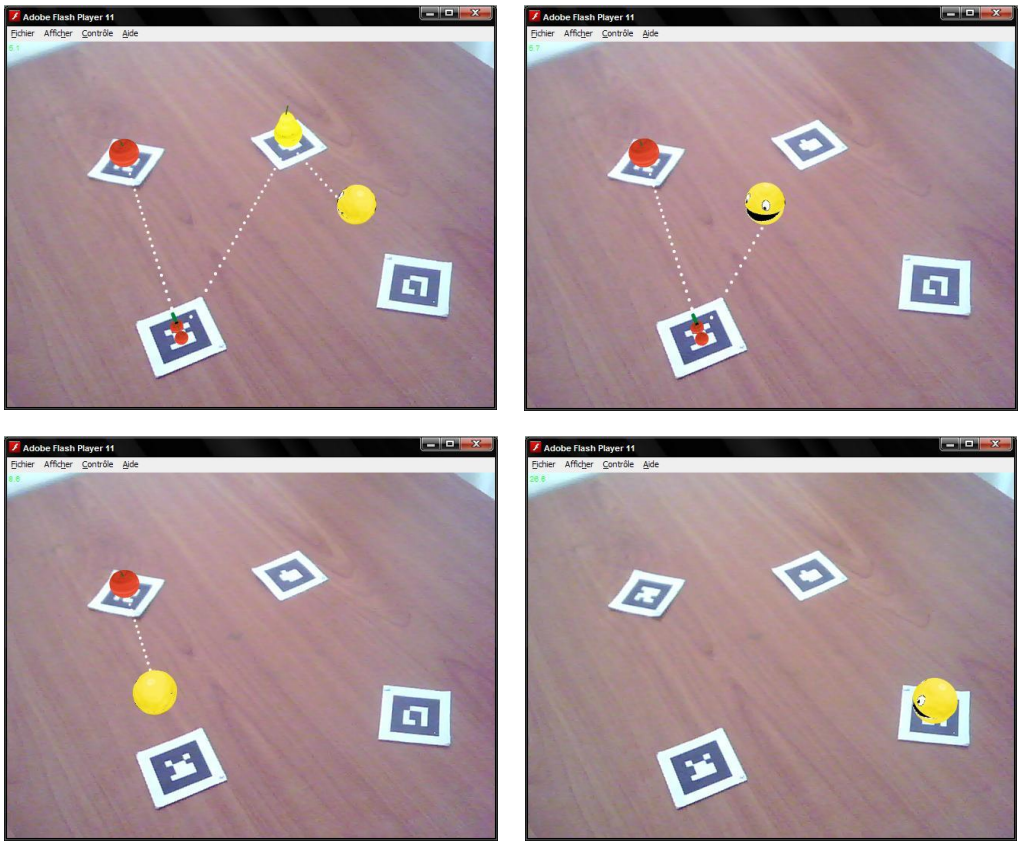


Fig. 8.3 : captures d'écran de l'animation.

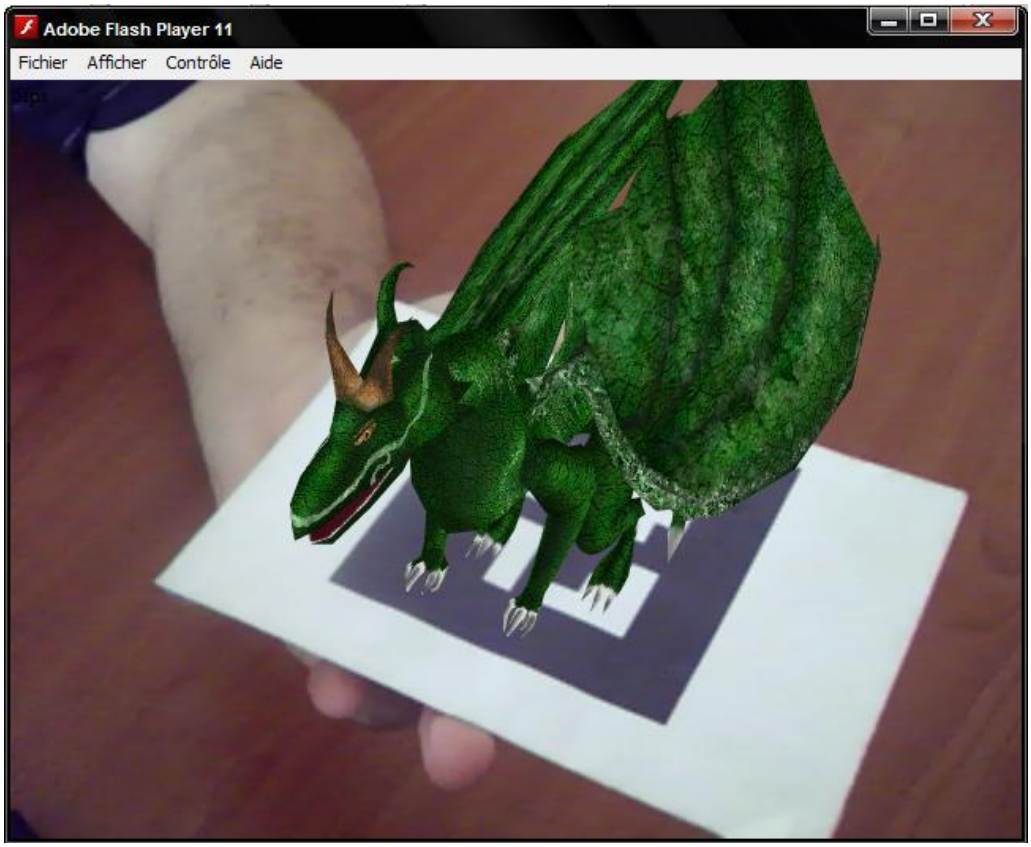


Fig. 8.4 : Présentation du modèle 3D d'un dragon de ma conception.

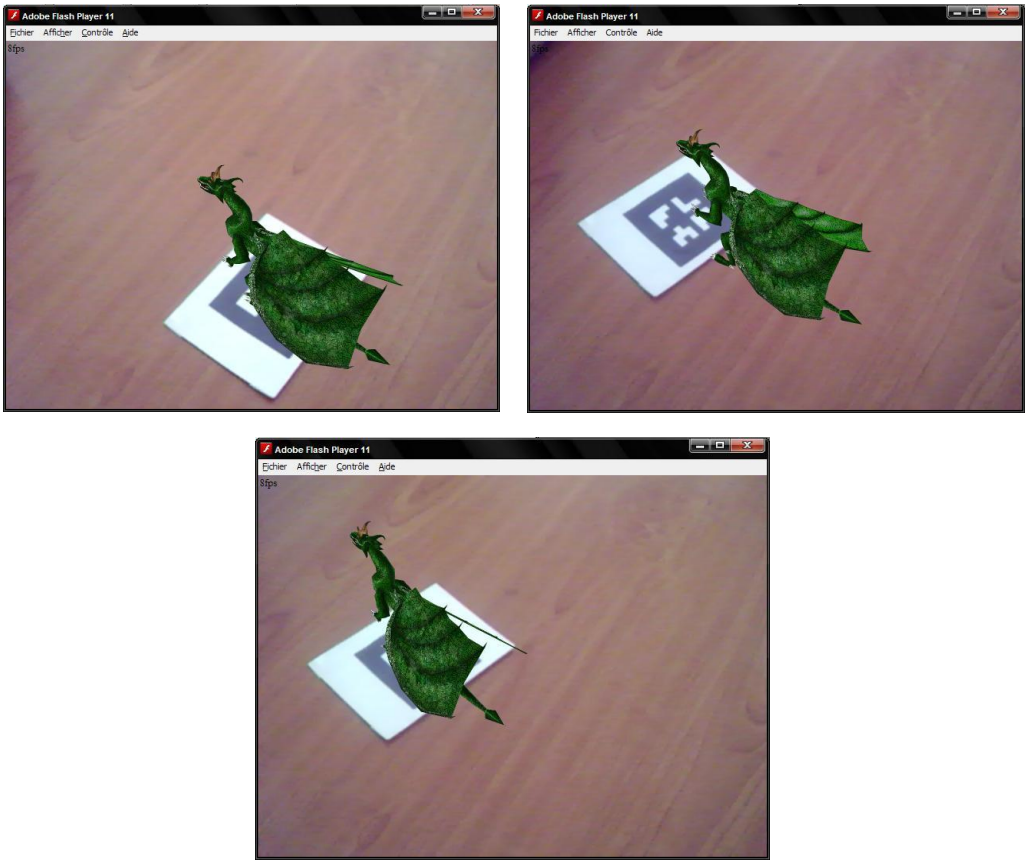


Fig. 8.4 : Interaction modèle marqueur avec une animation de marche si le marqueur est proche.

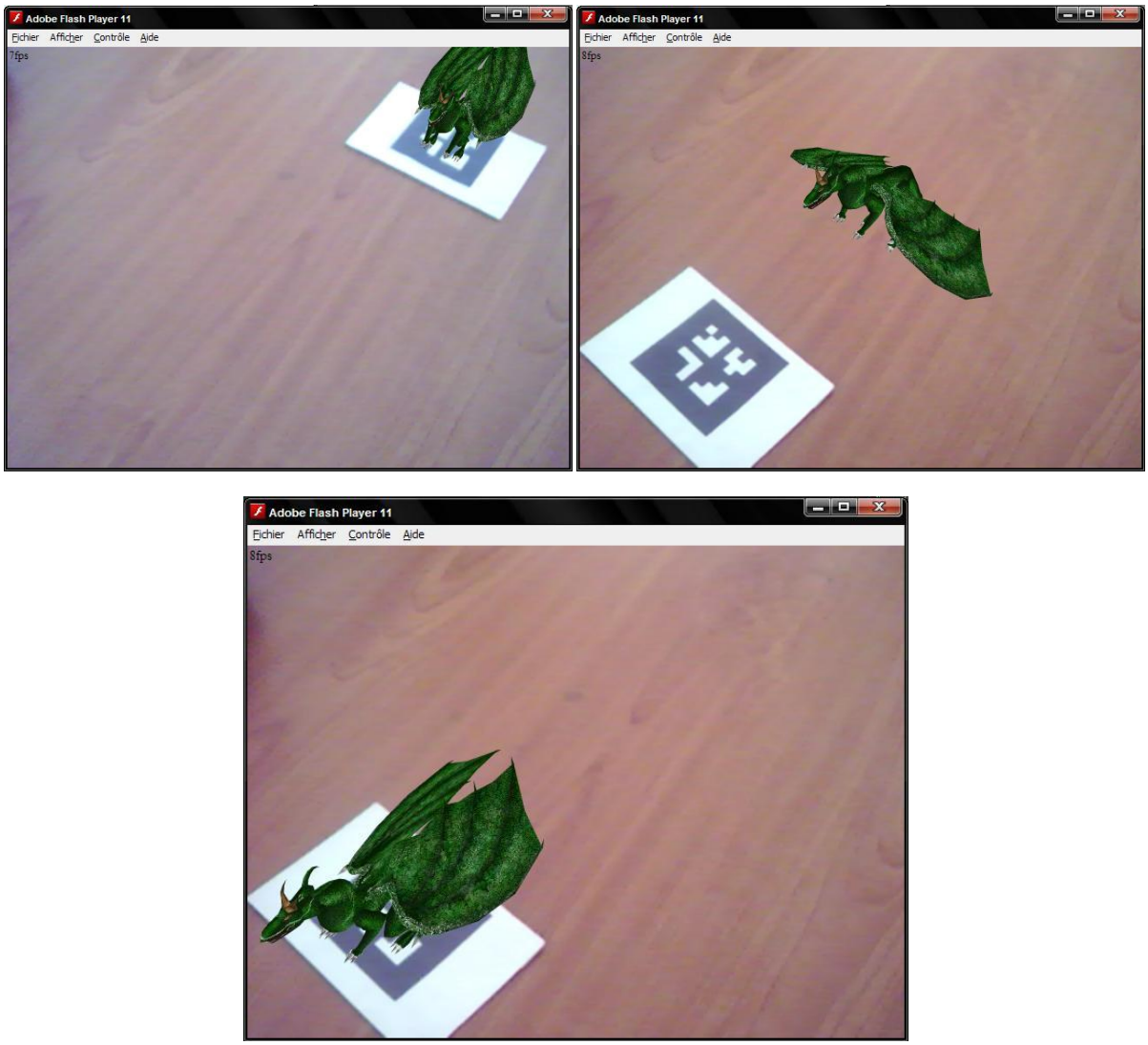


Fig. 8.5 : Interaction modèle marqueur avec une animation de vol si le marqueur est loin.

6) Conclusion :

Dans ce chapitre vous avais pu voire avec mes captures d'écran qu'il était tout à fait possible de faire interagir les marqueurs entre eux, ainsi que l'environnement de développement que j'ai choisi.

CONCLUSION GÉNÉRAL

La réalité augmentée est en perpétuelle évolution, elle représente le moyen audiovisuel du future, c'est pour ça que plusieurs entreprises du multimédia investissent dedans comme Appel et Microsoft, dans cette optique d'évolution continue j'ai voulu étudier s'il était possible de faire interagir différents marqueurs entre eux.

J'ai utilisé Flash AS3 dans ce but, car c'est un langage que j'affectionne et dont la documentation est riche et variée, entre autre il permet de créer des applications exécutables directement sous une simple fenêtre, ou bien des applets qui s'exécutent à travers des pages web.

Ce projet ma permis d'avoir des connaissances solides en AS3 surtout ce qui concerne la partie 3D temps réel ainsi qu'une bonne maîtrise des logiciels de modélisations 3D comme 3DSMax. Grâce à ça, je suis capable de créer n'importe quelle réalité augmentée et d'y faire interagir les différents marqueurs.

Cependant des perspectives d'améliorations restent envisageables telle que l'ajout d'un moteur physique pour rendre la scène plus dynamique, intégrer un algorithme d'indices naturels pour avoir des points d'ancrages autre que les marqueurs et enfin utiliser un algorithme qui détecte les courbures de la main pour que la main soit elle-même un marqueur.

Ceci conclut l'étude sur l'interaction entre les marqueurs, j'espère vous avoir fait apprécier la réalité augmentée à sa juste valeur, j'espère aussi que cette démarche ouvrira des perspectives pour la réalité augmentée à base du langage AS3 et donne lieu à d'autres applications du langage avec des études plus approfondies sur la bibliothèque de FLARToolKit afin de réaliser des applications RA plus réalistes et avec une plus grande facilité.

Bibliographie

[PER 97] B. PEROCHE, D. CHAZANFARPOUR, D. MICHELUCCI & M. ROELENS
« Informatique graphique, méthodes & modélisations » 2eme édition HERMES 1997.

[LOO 92] F. LOUGUET
« Synthèse d'image sur micro-ordinateur, Techniques, matériels, logiciels » DUNOD 1992.

[FOL 01] J. FOLEY, A. VAN DAM, S.K. FEINER
« Introduction à l'infographie » ADDISION WESLEY 2001.

[ROG 97] D.F ROGERS
« Algorithmes pour l'infographie » EDISCIENCE INTERNATIONAL 1997.

[ROB 06] L. ROBERT
« Intégrer images réelles et images 3D » DUNOD, PARIS, 2006.

Webographie

[WEB1] http://active.tutsplus.com/tutorials/3d/beginners-guide-to-augmented-reality/

[WEB2] http://www.augmented-reality.fr/about/

[WEB3] http://veille.epu-lyon1.fr/2011/realaugm/index.html

[WEB4] http://ilab.cs.ucsb.edu/projects/taehee/HandyAR/

[WEB5] http://tynambule.net/2007/12/13/41-calcul-du-framerate-reel-d-une-animation/

[WEB6] http://www.tutorgig.info/v/flartoolkit+basics

[WEB7] http://www.scribd.com/doc/27363030/Flartoolkit-intoduction

[WEB8] http://www.helioflash.com/adobe-flash/tutos-paper-vision-3d/francais/2/initiation-a-papervision-3d/