République Algérienne démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Abou Bakr Belkaïd de Tlemcen
Faculté des Sciences de l'Ingénieur

Département d'Informatique

Projet de Fin d'Etude

Thème :

Développement d'un plugin de chargement des fichiers 3D Studio Max pour ALLEGRO

Présenté par :

Kaouadji Farah **&** *Merzouk Riad*

Soutenu en : **Juillet 2004, devant le jury**

Examinateurs : *M.Hadjila*
L.Benhemmadi

Encadreur : *M. MOUSSAOUI A.*

2003-2004

REMERCIEMENT

Ce travail a été réalisé grâce à la prise en charge de M.Moussaoui, qu'il trouve ici nos vifs et sincères remerciements pour sa disponibilité, compétence et son aide qui nous a permis la réalisation de ce projet.

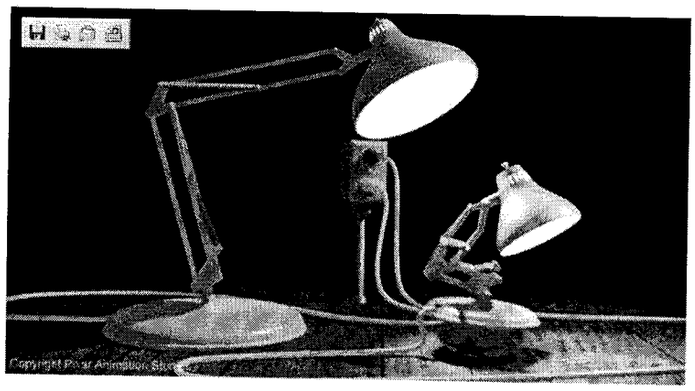
Nous remercions également Messieurs les membres du jury, monsieur Mourad Hadjila, et monsieur L.Benhemmadi pour avoir bien accepté de présider et examiner notre travail.

DEDICACES

A nos parents.

SOMMAIRE

CHAPITRE I. INTRODUCTION



La technologie de rendu 3d a réussi à s'immiscer dans la vie de millions de personnes à travers le monde. Que ce soit une console de jeux branchée à une télévision, un logiciel d'animation sur un poste de travail ou la dernière bombe cinématographique gonflée d'effets spéciaux , nous avons tous recours à l' expérience de la 3D et nous nous accrochons à son pouvoir sans pour autant nous soucier de la merveilleuse technologie qui se cache derrière.

L'objectif des premiers chapitres (I à VII) est de présenter les bases mathématiques aux profanes de la 3D. Les autres chapitres par contre traiteront de la réalisation du HEA3d.

CHAPITRE II. QUELQUE NOTION DE GEOMETRIE ANALYTIQUE

II.1. GENERALITE :

Dans notre traité, il sera nécessaire de savoir représenter et manipuler toute forme de courbe. Retenons que la meilleure façon de représenter une courbe par ordinateur est d'utiliser son équation mathématique.

Voici les différentes formes classiques d'équations d'une courbe :

FORME EXPLICITE :

y = f(x). Cette forme de représentation est univoque. Il n'est donc pas possible de représenter des courbes fermées ou qui se recoupent.

FORME IMPLICITE :

f(x,y) = 0. Cette forme permet la représentation de courbes fermées. Malheureusement tout point de la courbe doit se calculer en résolvant une équation.

Forme paramétrique :

$$\begin{cases} x = f(t) \\ y = f(t) \end{cases}$$ Tous les points de la courbe ont une coordonnée qui est une fonction d'un ou plusieurs paramètres. Cette forme la représentation des courbes fermées ou qui se recoupent.

FORME POLAIRE :

R = f (θ). Cette forme permet la représentation aisée des courbes fermées ou qui se recoupent.

DIVERSES EQUATIONS D'UN CERCLE :

On sait qu'un cercle est lieu des points situés à égale distance R d'un point fixe (X0, Y0) appelé centre.

L'équation cartésienne d'un cercle est :

$$(X-X+)2 +(Y-Y0)\ 2 = R2(1)$$

Si le cercle est centré à l'origine des axes alors l'équation devient :

X2-Y2 = R2 (2)

EQUATION CARTESIENNE EXPLICITE :

L' équation. 2 peut se décomposer en les deux équations suivantes :

$$Y = \pm\sqrt{R^2 - X^2}$$

EQUATIONS PARAMETRIQUES :

La trigonométrie élémentaire donne immédiatement :

$$\begin{cases} X = R.\cos\theta \\ Y = R.\sin\theta \end{cases}$$

EQUATION RECURSIVE :

Cette méthode permet de tracer un cercle en ne calculant qu'une seule fois deux nombres trigonométriques :

$$\begin{cases} X_{n+1} = X_n.\cos\theta - Y_n.\sin\theta \\ Y_{n+1} = Y_n.\cos\theta + X_n.\sin\theta \end{cases}$$

LES CONIQUES, ELLIPSE – PARABOLE – HYPERBOLE :

Les trois coniques sont des courbes fondamentales qui interviennent dans de nombreuses applications.

L'ELLIPSE :

Voici son équation cartésienne :

$$\frac{X^2}{A2} + \frac{Y^2}{B^2} = 1$$

Son équation paramétrique est :

$$\begin{cases} X = A.\cos\theta \\ Y = B.\sin\theta \end{cases}$$

LA PARABOLE :

Son équation cartésienne est :

$Y=X^2$

Son équation paramétrique est :

$$\begin{cases} X = 2p.\cot g^2\theta \\ Y = 2p.\cot g\theta \end{cases}$$

L'HYPERBOLE :

Equation cartésienne :

$$\frac{X^2}{A^2} - \frac{Y^2}{B^2} = 1$$

Équation paramétrique :

$$\begin{cases} X = A.\cos\theta \\ Y = B.tg\theta \end{cases}$$

CHAPITRE III. TRACES DE BASE

III.1. INTRODUCTION

Un ordinateur est capable de transmettre ses informations à une large variété de dispositifs capables de produire des sorties graphiques. Leurs principes sont de convertir des informations stockées dans la mémoire en une image visible.

Le dispositif périphérique le plus utilisé aujourd'hui est la visu à balayage de trame (CRT = cathode ray tube = tube a rayons cathodiques).

Le traçage de primitives 2D (pixel, segment, cercle, ellipse...) est une des fonctionnalités de base du dessin 2D qui est utilisée par la plupart des applications en synthèse d'images.

Les aspects fondamentaux des algorithmes correspondants sont la rapidité d'exécution et la précision de l'approximation discrète réalisée.

Les solutions pour résoudre le problème du tracer de primitives sont :

III.2. METHODES BRUTES (NAÏVE) :

Cette méthode qui permet la discrétisation de la primitive en n points, puis l'approximation au pixel le plus proche pour chacun des n points reste peu efficace et peu précise.

III.3. METHODES INCREMENTALES :

La position du point suivant est choisie de façon à minimiser l'erreur d'approximation. Méthode optimale pour le tracé de segments de droites.

III.4. LES PIXELS

Un pixel d'une visu ne représente pas un point mathématique. Il représente une région qui contient théoriquement une infinité de points.

Le pixel est une représentation graphique d'un point sur l'écran. On définit un pixel par sa position sur l'écran et sa couleur.

III.5.SEGMENTS DE DROITES

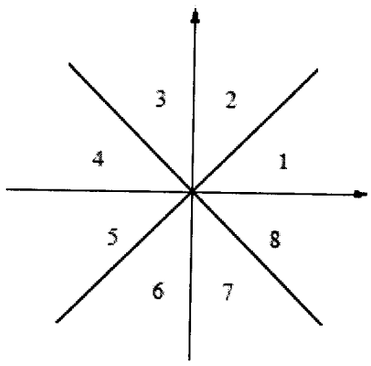


Figure 1: Les huit octants.

On réduit la complexité du problème en traitant tout d'abord le tracer de segment appartenants au premier octant.

Les autres se déduisant ensuite par symétries.

III.6.ALGORITHMES SIMPLES

Une droite est définie par l'équation :

$y = m*x + B$ avec $m = \Delta y / \Delta x$.

En partant du point le plus à gauche et en incrémentant la valeur de xi de 1 à chaque itération, on calcule :

$y_i = m*x_i + B$

Puis on affiche le pixel (xi, $trunc(y_i + 0.5)$), le pixel le plus proche en x_i du segment traité.

Cette méthode est inefficace en raison du nombre d'opérations nécessaires à chaque itération.

En remarquant que :

$y_{i+1} = m*x_{i+1} + B$

$= m*(x_i + \Delta x) + B$

$= y_i + m*\Delta x$

On peut donc à partir de la position (xi, yi) déterminer la position du point suivant de manière incrémentale :

$x_{i+1} = x_i + 1;$

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

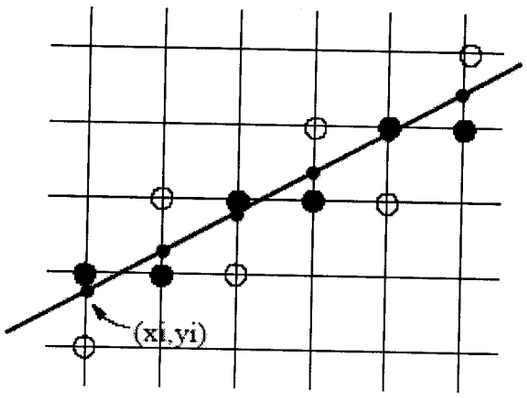


Figure 2 : Méthode simple

III.7. ALGORITHMES DE BRESENHAM (1965)

L'idée est de mesurer l'erreur à chaque itération (distance du pixel à la droite) et de choisir la meilleure approximation.

On teste pour cela le signe de d1 - d2.

$e = d2 - d1 \geq 0$ Nord-Est

$e < 0$ Est

Au premier point (x0 + 1; y0 + m):

$e1 = (d2 - d1)=2 = d2 - 0.5 = m - 0.5$

Ensuite le calcul de l'erreur se fait de manière incrémentale, à chaque itération:

Nord-Est : $e = e + m - 1;$

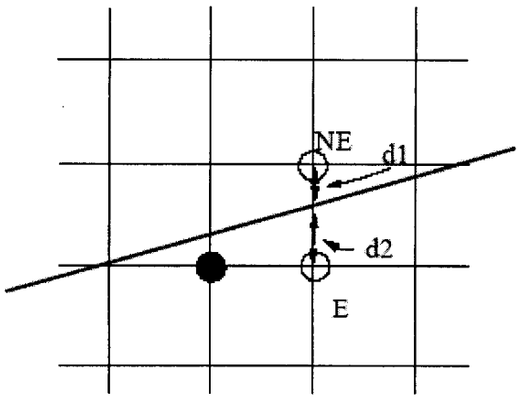


Figure 3: Algorithme de *Bresenham* : le choix se fait entre le point Est et le point

Est : e = e + m:

Nord-est à chaque itération en fonction de l'erreur d2 - d1.

Dans l'approche précédente seule l'erreur et le coefficient directeur de la droite sont réels. Une amélioration possible consiste à effectuer toutes les opérations sur des entiers. Pour cela, il suffit de remarquer que :

NE : $e = e + \Delta y / \Delta x - 1$

E : $e = e + \Delta y / \Delta x$

NE : $\Delta x * e = e + \Delta y - \Delta x$

E : $\Delta x * e = e + \Delta y$

NE : $e' = e' + \Delta y - \Delta x$

E : $e' = e' + \Delta y$

Avec : e'

$e'1 = e1 * \Delta x = \Delta y - \Delta x/2$

Bresenham a montré que cet algorithme correspond à la meilleure approximation de la droite (erreur minimum).

III.8. GENERALISATION AUX HUIT OCTANTS

La généralisation se fait en choisissant différentes valeurs d'incrémentation pour

x et y et en itérant sur x ou y en fonction de la valeur du coefficient directeur

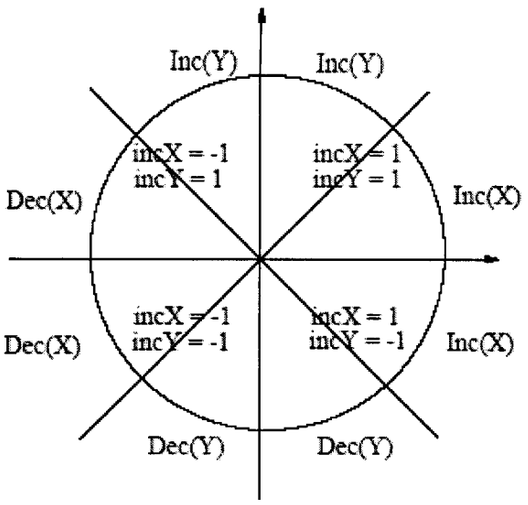


Figure 4: Généralisation de l'algorithme de **Bresenham** aux 8 octants.

(supérieur ou inférieur à 1).

III.9.CERCLES

Nous traitons cette fois-ci les pixels du deuxième octant. L'erreur est calculée en évaluant la valeur de la fonction F(x, y, R) = x2 + y2 - R2 au point milieu M entre deux pixels, cette valeur étant positive à l'extérieur du cercle et négative à l'intérieur.

Si le point milieu M est à l'extérieur du cercle, alors le pixel Sud-Est est le plus

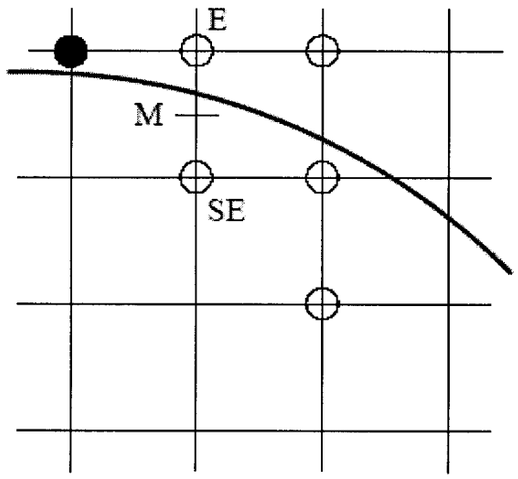


Figure 5: Tracer de cercle

proche du cercle. Dans le cas contraire, le pixel Est est le plus proche (voir la figure).

De la même manière que pour les segments de droites, le calcul de l'erreur se fait de manière incrémentale.

Au pixel affiché (xi, yi), la position du point M (pour le point suivant) dépend du signe de ei :

$e_i = F(x_i + 1, y_i - 1=2, R)$

Si $e_i > 0$ (M à l'extérieur) alors le pixel à afficher est Sud-Est est le prochain point milieu sera $(x_i + 2, y_i - 3/2)$.

La valeur ei+1 s'écrit :

$e_i+1 = F(x_{i+2}, y_i - 3/2, R) = ei + 2x_i - 2yi + 5$

Si ei < 0 (M à l'intérieur) alors le pixel à afficher est Est est le prochain point milieu sera (xi + 2, yi – 1/2).

La valeur ei+1 s'écrit :

ei+1 = F(xi + 2, yi – ½, R) = ei + 2xi + 3

Cette approche nécessite des opérations réelles pour le calcul de d.

En remarquant que le rayon a une valeur entière et que les incréments sur d ont aussi des valeurs entières, on peut alors substituer comme valeur initiale de d la valeur 1 - R et effectuer des opérations strictement entières.

CHAPITRE IV. LES TRANSFORMATIONS DE VISUALISATION :

IV.1. DEFINITIONS

IV.1.1. LES SYSTEMES DE COORDONNEES :

Pour repérer un pixel sur écran, on utilise généralement un système d'axe (repère) à deux dimensions, présenté comme suite :

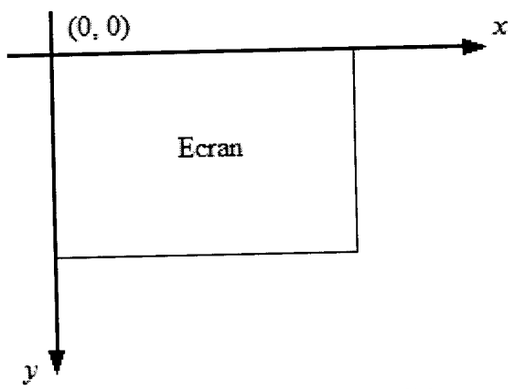


Figure 1 : repère associé à l'écran

IV.2. FENETRAGE ET CLOTURE

Une fenêtre (écran) spécifie la zone de données qui va être vue par l'ordinateur.

A l'intérieur de la fenêtre il est nécessaire de définir une clôture (*Viewport*), car nous n'avons pas toujours besoin de l'ensemble de la fenêtre.

La clôture spécifie la zone écran où sera projeté son contenu.

REMARQUE

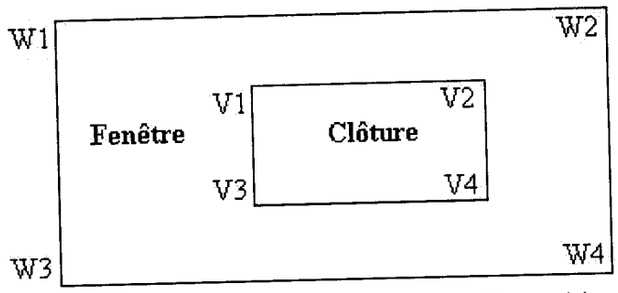


Figure 2 : Fenêtre et clôture (a)

Pour obtenir un dessin sans distorsions, il faut que les zones fenêtre et clôture soient homothétiques (voir figure).

B/H= B'/H'

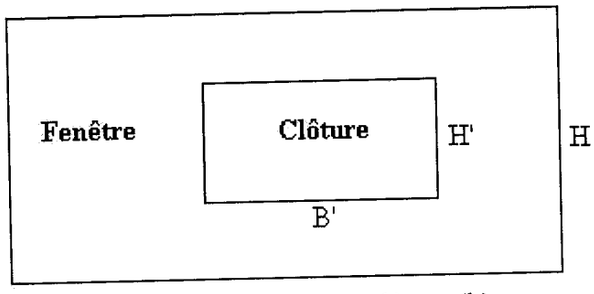


Figure 2 : Fenêtre et clôture (b)

IV.3. LE PROBLEME DU DECOUPAGE DANS LE PLAN (CLIPPING)

IV.3.1. INTRODUCTION

Lorsque des parties graphiques n'appartiennent pas à la fenêtre (voir fenêtrage), elles seront coupées. Il y aura donc un phénomène de découpage (*Clipping*).

Si on ne désire pas une grande précision dans le découpage, on peut se contenter de comparer les coordonnées des points à dessiner avec les frontières de la fenêtre.

Dans le cas où le point se trouve à l'extérieure de la fenêtre, il n'est pas dessiné. Cependant cette méthode n'est bonne que pour dessiner des points et donne de très mauvais résultats pour le tracé de segments.

Aussi allons-nous présenter à la suite de ce chapitre une autre méthode plus précise et plus efficace.

IV.3.2. METHODE DE DAN COHEN ET YVAN SUTHERLAND :

On élargit les bords de la fenêtre de telle façon à diviser le plan de l'utilisateur en neuf régions. On donne un code binaire de 4 bits à chacune de ces régions avec la signification suivante :

Bit de rang 1 : 1 si le point est à gauche du bord gauche de la fenêtre.

Bit de rang 2 : 1 si le point est à droite du bord droit de la fenêtre.

Bit de rang 3 : 1 si le point est en- dessous du bord inférieur de la fenêtre.

Bit de rang 4 : 1 si le point est au- dessus du bord supérieur de la fenêtre.

Dans tous les autres cas le bit concerné est 0.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Il n'existe que trois cas possibles :

- deux extrémités en dehors de la fenêtre. Cela signifie que l'on a deux codes différents de 0000.

- un segment complètement dans la fenêtre. On a deux codes de 0000.

- un segment dont une seule extrémité est dans la fenêtre. On a un code de 0000.

Le principe de l'élimination est le suivant :

si les deux codes binaires sont 0000 alors le segment est entièrement vu.

Sinon on calcule l'intersection logique (ET ou AND) des deux codes binaires :

Si le résultat est différent de 0000 alors le segment est entièrement invisible.

Dans le cas contraire le segment a une extrémité dans la fenêtre et l'autre en dehors. On cherche la coordonnée du point d'intersection de ce segment avec un bord de la fenêtre. Puis on relie ce dernier avec l'autre extrémité du segment (qui est dans la fenêtre).

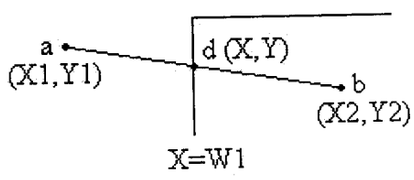
IV.3.2.1. Calcul d'une intersection avec la fenêtre :

Soit un segment ab qui coupe la fenêtre au point d.

Sachant que les coordonnées de a et b sont respectivement (X1, Y1) et (X2, Y2), on trouve la coordonnée du point d en cherchant la coordonnée du point d'intersection des droites X=w1 et ab :

$$ab = \frac{Y - Y1}{Y2 - Y1} = \frac{X - X1}{X2 - X1}$$

$$D'où \begin{cases} X = w1 \\ Y = Y1 + \dfrac{(Y2 - Y1)(w1 - X1)}{X2 - X1} \end{cases}$$



CHAPITRE V. BASES MATHEMATIQUES DU GRAPHISME A 2 DIMENSIONS

V.1. LES TRANSFORMATIONS DU PLAN :

Les principales transformations du plan sont les suivantes :

les translations, les changements d'échelles, les symétries, les rotations.

Pour cela nous aurons besoin des matrices et de l'utilisation du calcul matriciel pour résoudre ces problèmes.

En effet, soit (x, y) la coordonnée d'un point du plan cartésien.

La matrice générale $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ est une matrice de 2 lignes sur deux colonnes ; elle sera notée matrice (2x2).

Considérons le produit matriciel suivant et notons [x' y'] le résultat.

$$[x'y'] = [x\ y].\begin{bmatrix} A & B \\ C & D \end{bmatrix} = [A.x + C.y \quad B.x + D.y].$$

Donc, tout point du plan (x, y) multiplié par la matrice (2x2) a pour transformé un nouveau point du plan (x', y') tel que :

$$\begin{cases} x' = Ax + Dy \\ y' = Bx + Dy \end{cases}$$

La transformation obtenue dépendra des valeurs données aux variables A, B, C, D.

V.1.1. LES CHANGEMENTS D'ECHELLES :

Ces transformations sont contrôlées par la matrice $M = \begin{bmatrix} A & 0 \\ 0 & D \end{bmatrix}$

En effet : $[x\ y].\begin{bmatrix} A & 0 \\ 0 & D \end{bmatrix} = [A.x\ \ D.y]$ d'où $\begin{cases} x' = A.x \\ y' = D.y \end{cases}$

REMARQUE :

Comme la matrice identité est l'élément neutre pour la loi de multiplication des matrices, une multiplication par la matrice identité transforme tout point en lui-même.

V.1.2. LES SYMETRIES :

Les matrices qui contrôlent les symétries ne sont que des cas particuliers de la matrice changement d'échelle dans laquelle A et/ou D sont négatifs.

Exemple :

$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ Produira une symétrie par rapport à l'axe des y.

V.1.3. LES ROTATIONS :

Pour faire une rotation d'un point p dans un plan 2D autour de l'origine, on multiplie les coordonnées de ce point par R la matrice rotation :

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

Avec θ est l'angle de rotation.

V.1.4. LES TRANSLATIONS ET LES COORDONNEES HOMOGENES :

Pour faire une translation d'un point du plan il nous faut arriver à la forme suivante :

$$\begin{cases} x' = x + M \\ y' = y + N \end{cases}.$$

On peut remarquer qu'aucun produit par la matrice générale (2x2) ne permet de trouver ces relations. Ce qui nous oblige a introduire une troisième composante aux vecteurs [x y] et [x' y'] les faisant devenir [x y 1] et [x' y' 1].

La matrice de transformation doit obligatoirement devenir une matrice (3x2) car on sait qu'un produit matriciel n'est possible que si le nombre de lignes de la deuxième matrice égale le nombre de colonnes de la première.

Soit la matrice :

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ M & N \end{bmatrix}$$

Montrons qu'elle fournit bien une translation :

$$[\,x\ y\ 1\,].\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ M & N \end{bmatrix} = [x+M\ \ y+N] \quad \text{d'où} \begin{cases} x' = x + M \\ y' = y + N \end{cases}$$

Le coefficient M provoque une translation par rapport à l'axe X.

Le coefficient N provoque une translation par rapport à l'axe Y.

Mais une matrice (3x2) n'étant pas carrée, ne possède pas donc d'inverse.

C'est pourquoi nous allons rajouter une troisième colonne à la matrice (3x2) de façon à obtenir une matrice carrée (3x3)(inversible).

On aura pour résultante :

$$[x\ y\ 1].\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ M & N & 1 \end{bmatrix} = [x+M\ \ y+N\ \ 1] = [x'\ y'\ 1]$$

On dit que la représentation de la position d'un point du plan par un vecteur à 3 dimensions est une représentation en coordonnées homogènes.

V.2. COMPOSITION DE TRANSFORMATIONS

Le but de ce dernier est de construire une matrice, qui effectuera plusieurs transformations à la fois.

Exemple : une rotation et une translation. Pour la rotation nous avons besoin d'une matrice de rotation M et pour la translation une matrice de translation N. Afin d'avoir une seule matrice capable de faire les deux transformations, nous devons multiplier les deux matrices M et N, afin d'obtenir une troisième matrice T. Disons immédiatement que l'ordre des opérations à beaucoup d'importance car en général le produit matriciel n'est pas commutatif. En appliquant la matrice T sur l'objet considéré, ce dernier effectuera une rotation en même temps qu'une translation.

Ainsi on peut généraliser cette matrice sous la forme suivante :

$$\begin{bmatrix} A & B & 1 \\ C & D & 1 \\ M & N & E \end{bmatrix}$$

On peut la diviser en quatre parties qui auront chacune un rôle bien précis.

Comme nous l'avons vu, A, B, C et D produisent les changements et les rotations. Les termes M et N produisent les translations.

Le terme E produit un changement d'échelle général, c'est- à- dire uniforme selon les deux axes.

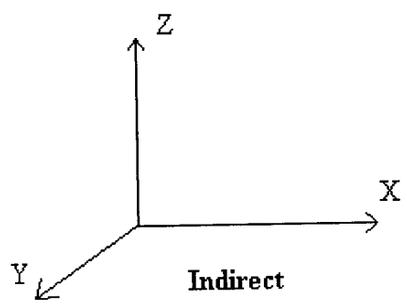
CHAPITRE VI. LES BASES MATHEMATIQUES DU GRAPHISME A 3 DIMENSIONS

VI.1. LES SYSTEMES DE COORDONNEES :

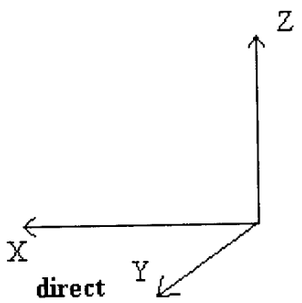
VI.1.1. LES COORDONNEES CARTESIENNES : P(X, Y, Z)

Le système direct est dit de la main droite, car si l'index pointe dans la direction de l'axe X et le majeur dans la direction de l'axe Y, le pouce pointe alors dans la direction de l'axe Z.

Le système indirect aura la direction de l'axe Z opposée. D'une autre manière, si l'on regarde le plan XY d'un système direct alors l'axe Z pointe vers l'œil de l'observateur.



Indirect



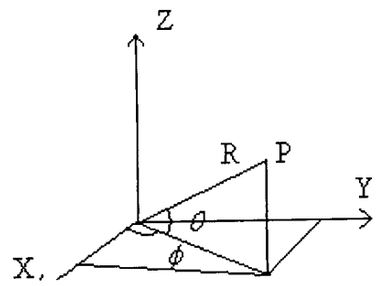
direct

REMARQUE :

Pour repérer un point p dans l'espace, on peut adopter le système direct.

VI.1.2. LES COORDONNEES SPHERIQUES :

$x = R.\cos\theta.\cos\phi$



$y = R.\sin\theta.\cos\phi$

$z = R.\sin\phi$

$R = X2+Y2+Z2$

Tout point de l'espace cartésien a une coordonnée du type (x, y, z) et une coordonnée homogène (x, y, z, 1).

Il faut donc que toutes les transformations que nous allons étudier soient représentées par des matrices (4x4).

La quatrième colonne ajoutée à la matrice est nécessaire pour obtenir une matrice carrée, ce qui est indispensable pour qu'elle aie une inverse.

VI.2. LES CHANGEMENTS D'ECHELLES :

L'opérateur changement d'échelle non uniforme (*A selon l'axe des x, B selon l'axe des y et C selon l'axe des z*) pour les points 3d est défini par la matrice suivante :

$$\begin{bmatrix} A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & C & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Les termes de la diagonale de la matrice (4x4) produisent les changements d'échelles selon un axe bien déterminé.

REMARQUE :

Dans le cas où A=B=C, le changement d'échelle est dit uniforme.

VI.3. LES ROTATIONS :

Les matrices Rx, Ry, Rz représentent, respectivement les matrices de rotation autour des trois axes x, y, z.

Dans tous les cas la rotation est faite dans le sens positif (le sens trigonométrique) lorsqu'on regarde l'origine d'un point situé sur l'axe positif concerné.

$$Rx = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Ry = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Rz = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ -\cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

VI.4. LES TRANSLATIONS :

La matrice suivante représente une translation d'un point dans l'espace M selon l'axe des x, N selon l'axe des y, et P selon l'axe des z.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ M & N & P & 1 \end{bmatrix}$$

VI.5. LES SYMETRIES :

Pour trouver le symétrique d'un objet par rapport au plan XY, il suffit de changer le signe de la cote Z de tous les points de cet objet. Le raisonnement est identique pour les symétries par rapport aux plans XZ et YZ.

$$SXY = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad SXZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad SYZ = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

VI.6. INVERSE D'UNE MATRICE :

Exemple : si l'on a une matrice T qui effectue une translation d'un point qui se trouve, à l'origine, au point (1, 2, 3) la matrice inverse de T (T^{-1}) fera une translation au point (-1,-2,-3).

REMARQUE :

Cela signifie qu'une matrice de transformation et son inverse ont un effet contraire.

L'inverse d'une matrice de translation s'obtient en remplaçant les termes –M,-N et -P.

L'inverse d'une matrice de changement d'échelle s'obtient en remplaçant les termes A, B, C par leur inverse 1/A, 1/B, 1/C.

L'inverse d'une matrice de rotation s'obtient en remplaçant l'angle θ par $-\theta$.

VI.7. TRANSFORMATION D'UN SYSTEME D'AXE :

Nous avons transformé jusqu'alors des points rapportés à un système d'axes. Il sera très utile aussi de transformer un système d'axes en un autre système en faisant subir une transformation aux axes eux- mêmes.

La matrice nécessaire à la rotation d'un système d'axes est la matrice inverse de celle nécessaire à la rotation d'un point.

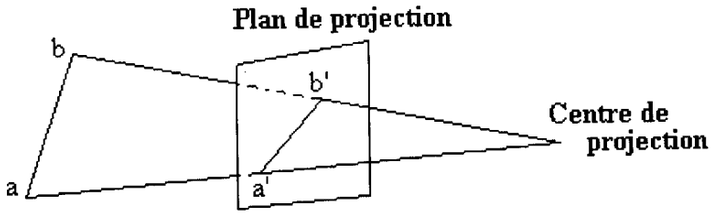
VI.8. LES DIFFERENTS TYPES DE PROJECTION SUR UN PLAN :

La catégorie de projection qui nous intéresse est connue sous le nom de ***projections géométriques planes*** parce que la projection se fait sur un plan plutôt que sur une surface courbée.

Les projections géométriques planes se divisent en deux grandes classes :

VI.8.1. LES PROJECTIONS PERSPECTIVES :

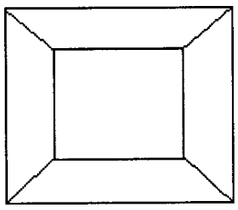
Dans ce premier type, le centre de projection est à distance finie du plan de projection.



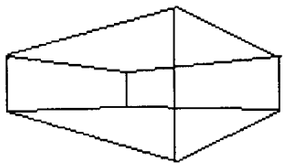
Ce type de projection crée un effet visuel semblable à celui perçu par le système de vision humain. La taille de l'objet est inversement proportionnelle à la distance séparant l'objet du centre de projection.

On peut classer les projections perspectives selon le nombre de leurs *points de fuite* : un point de fuite étant le point de rencontre des parallèles de la projection.

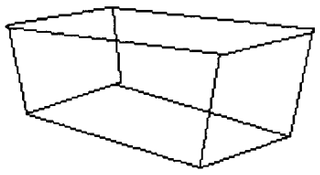
Voici trois schémas illustrant la notion de point de fuite :



1 Point de fuite



2 Point de fuite



3 point de fuite

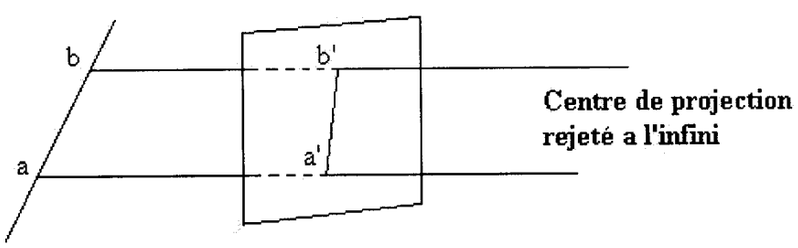
La projection perspective à un point de fuite a lieu lorsqu'une des faces d'un parallélépipède est parallèle au plan de projection.

La projection perspective à deux points de fuite a lieu lorsqu'un ensemble d'arêtes sont parallèles au plan de projection sans qu'aucune face ne lui soit parallèle.

La projection perspective à trois points de fuite si aucune arête n'est parallèle au plan de projection.

VI.8.2. LES PROJECTIONS PARALLELES :

Dans ce type de projection, le centre de projection est à l'infini.



Ce type de projection est moins réaliste que la projection perspective. Le parallélisme est conservé.

Les distances incluses dans une face parallèle au plan de projection restent invariables. Les projections parallèles sont à leur tour divisées en deux classes :

VI.8.2.1. Les projections orthogonales

On obtient trois vues classiques : évaluation, face et profil. Dans ce cas, le plan de projection est perpendiculaire à la direction de la projection et à l'un des axes.

Si le plan de projection n'est pas perpendiculaire à la direction de la projection on a une projection dite axonométrique. De ce fait plusieurs faces sont montrées simultanément.

VI.8.2.2. La projection axonométrique

La projection axonométrique diffère cependant de la perspective car le raccourcissement des distances est uniforme.

Les projections axonométriques les plus usuelles sont :

TRIMETRIQUE :

Les trois axes sont raccourcis de façon inégale.

DIMETRIQUE :

Deux des trois axes sont raccourcis de façon égale.

ISOMETRIQUE :

Le plan de projection fait des angles égaux avec les axes. Les trois axes sont raccourcis de façon égale permettant ainsi des mesures avec la même échelle.

VI.8.2.3. Les projections obliques :

Dans ce type de projection, le plan de projection est perpendiculaire à l'un des axes (mais plus perpendiculaire à la direction de la projection). Les plus connues sont la projection cavalière et la projection cabinet.

Dans la première, la direction de projection fait un angle de 45° avec le plan de projection.

Dans la seconde, on raccourcit les lignes fuyantes dans le rapport ½. De ce fait la projection cabinet est plus réaliste que la projection cavalière.

VI.9. LA PROJECTION EN PERSPECTIVE

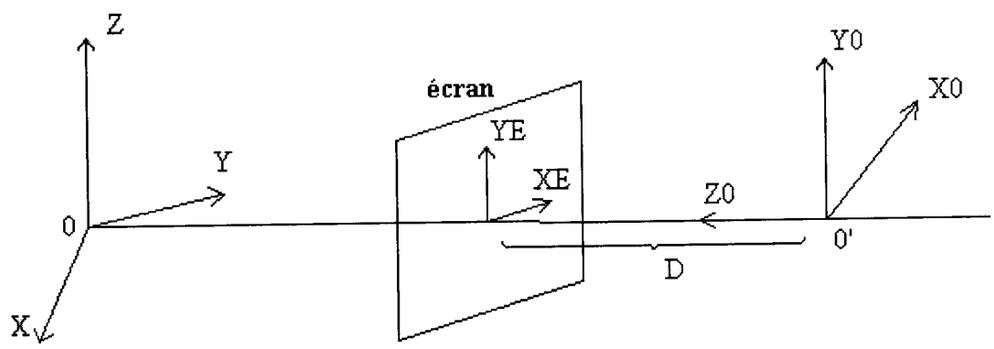
Afin de projeter une image d'un objet sur le plan de l'écran, il est nécessaire d'associer à toute coordonnée d'un point de l'espace une coordonnée écran. Nous postulerons toujours que :

- L'écran est un plan perpendiculaire à la droite qui joint l'œil à l'origine du système d'axes auquel l'objet est rapporté.

- L'écran est situé à une distance D de l'observateur.

- Le système de coordonnées de l'observateur a son origine placée en son œil et l'axe Z0 pointe dans la direction de l'origine 0.

C'est donc un système indirect.



Une vue en perspective de l'objet sera générée simplement en projetant chaque point de l'objet sur le plan de l'écran.

Voici la relation simple liant la coordonnée écran (xe,ye) à la coordonnée (xo,yo,zo) d'un point quelconque p de l'objet considéré dans le système de l'observateur :

$$xe = \frac{D.xo}{zo} \qquad \text{Et} \qquad ye = \frac{D.yo}{zo}$$

VI.10. VISUALISATION D'UN OBJET A TROIS DIMENSIONS

Dans ce chapitre nous essayerons d'étudier la manière de traiter les données d'un objet dans un espace à trois dimensions, dans le but de produire des images réalistes à deux dimensions de différentes vues de cet objet.

Nous venons de voir comment la projection perspective pouvait être utilisée pour créer une image bi- dimensionnelle d'un objet à trois dimensions. Mais pour le percevoir sous n'importe quel angle nous avons le choix entre deux possibilités :

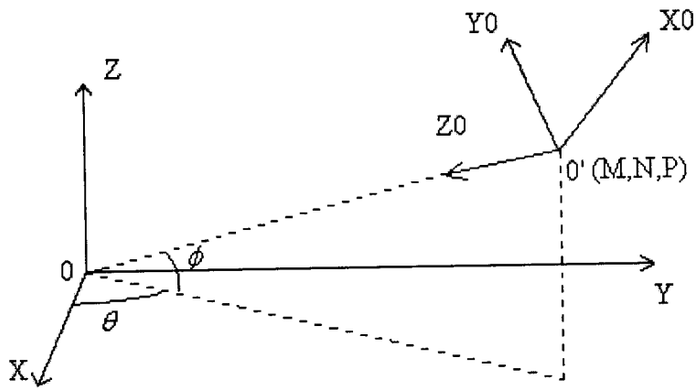
-Le point de vu reste fixe et l'objet subit toutes les transformations souhaitées.

-L'objet reste fixe et le point de vue est positionné correctement.

La deuxième possibilité est plus proche de la réalité. Comme la manipulation d'une caméra, les objets restent fixes et c'est la caméra qui change de point de vue.

Nous allons prendre pour exemple un objet repéré dans le système de coordonnées(X, Y, Z) et centré au point 0.

L'œil de l'observateur a une coordonnée (M, N, P) dans ce même système. Sa coordonnée sphérique (R, θ, ϕ).

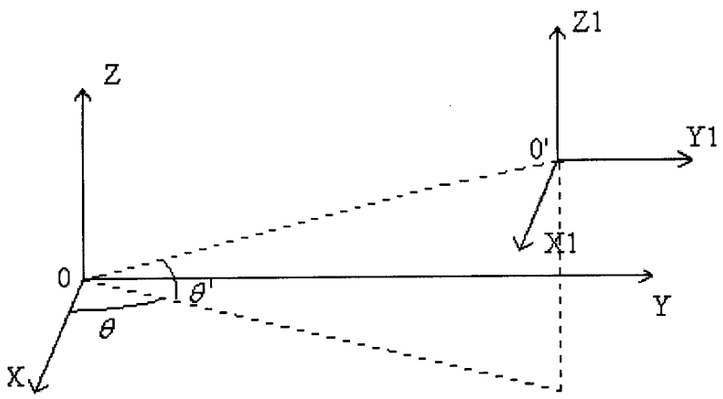


Nous allons voir quelles sont les transformations que le système (X, Y, Z) doit subir pour coïncider avec le système (X0, Y0, Z0).

ETAPE1 :

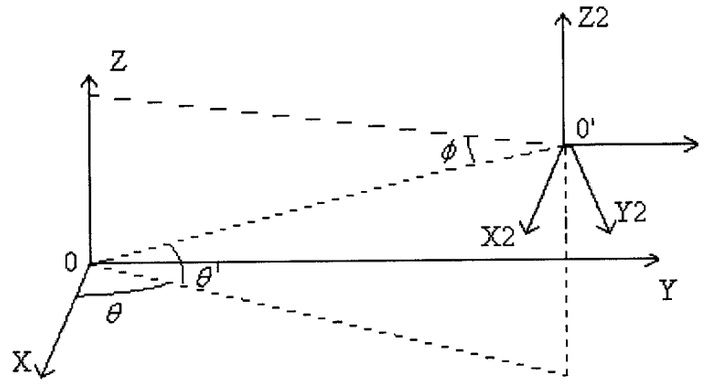
Translation de l'origine 0 au point 0' :

Celà se fera grâce à la matrice de translation A.



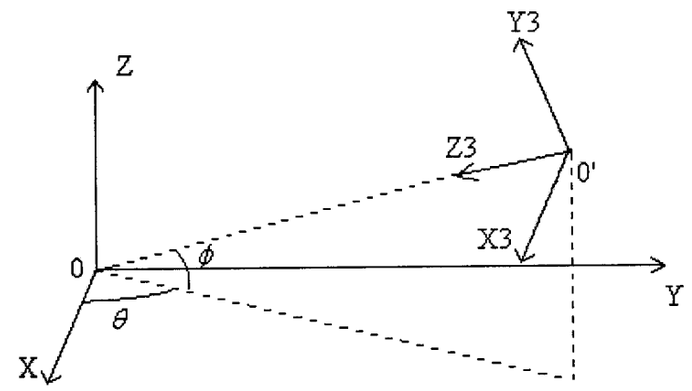
ETAPE 2 :

Rotation du système (X1, Y1, Z1) de $-\theta$ autour de l'axe Z1 grâce a la matrice de rotation B.



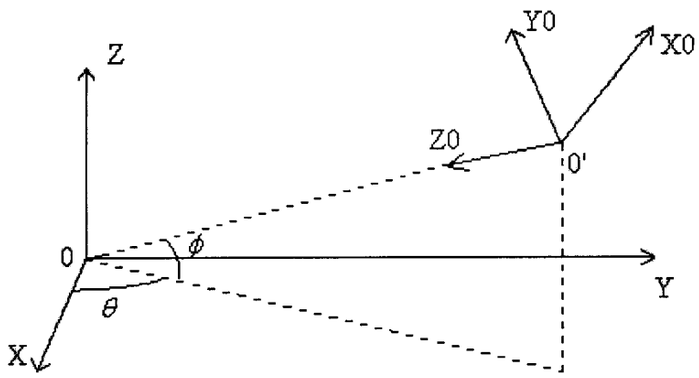
ETAPE 3 :

Rotation du système (X2, Y2, Z2) de $90°+\phi$ autour de l'axe X2 grâce à la matrice C.



ETAPE 4 :

Conversion du système (X3, Y3, Z3) grâce à la matrice D.



Nous pouvons constater que nous avons atteint notre but : Le système (X, Y, Z) a été transformé en système (X0, Y0, Z0).

CONCLUSION

Tout point dans l'espace objet aura dans le système observateur une coordonnée obtenue comme suit :

[xo yo zo 1] = [x y z 1]. A.B.C.D.

Si nous voulons obtenir les coordonnées écran, nous utiliserons ainsi les deux formules suivantes :

$$xe = \frac{D.xo}{zo} \qquad \text{Et} \qquad ye = \frac{D.yo}{zo}$$

CHAPITRE VII. LE PROBLEME DES SURFACES CACHEES

VII.1. INTRODUCTION

Chaque fois qu'une scène comporte des objets ou des surfaces opaques, ceux qui sont plus prés de l'observateur occultent la vue des objets plus éloignés.

Les surfaces ainsi occultées doivent être effacées si on veut obtenir une vue d'écran réaliste. Il existe beaucoup d'algorithmes performants, chacun a ses caractéristiques propres qui font qu'il n y a pas un seul qui soit meilleur que l'autre.

Dans notre travail nous avons étudié un algorithme classique qui a l'avantage de ne nécessiter qu'un faible appareil mathématique.

VII.2. LA METHODE DU TEST DE VISIBILITE D'UNE SURFACE :

Cette méthode permet de déterminer quelles surfaces d'un objet convexe font face à l'observateur. Les notions simples de produit scalaire et de produit vectoriel de deux vecteurs vont nous aider puissamment dans cette tâche.

Considérons deux vecteurs p= (p1, p2, p3) et q = (q1, q2, q3) de l'espace.

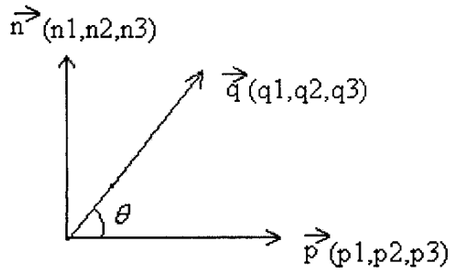
Ces deux vecteurs sont rapportés à un système d'axe direct.

Par définition le produit vectoriel de deux vecteurs est un troisième vecteur perpendiculaire à un plan.

La norme de ce troisième vecteur est égale au produit des normes des deux vecteurs donnés que multiplie le sinus de l'angle formé par ces deux vecteurs.

$n1 = p2.q3 - q2.p3$

$n2 = p3.q1 - q3.p1$



n3 = p1.q2 – q1.p2

$\|\vec{n}\| = \|\vec{p}\|.\|\vec{q}\|.\sin\theta$

REMARQUE

Le produit vectoriel n'est pas commutatif. L'orientation du vecteur résultant suit la rège de la main droite.

VII.2.1. LE TEST DE VISIBILITE :

Ayant bien saisi la nature géométrique du produit vectoriel, nous pouvons déterminer, si une face est une face avant, donc *visible*, ou arrière, donc *invisible*. Mais d'abord rappelons la définition du produit scalaire.

Le produit scalaire de deux vecteurs v et n est un nombre réel:

$\vec{v}.\vec{n} = \|v\|.\|n\|.\cos\theta$

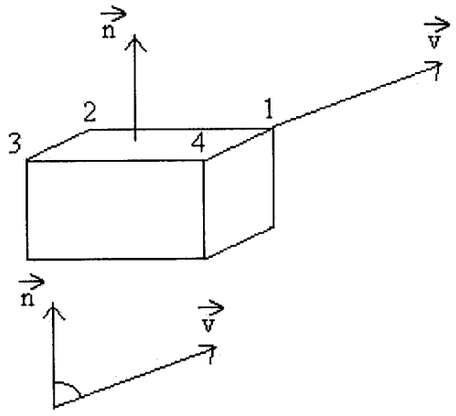
$\vec{v}.\vec{n}$= v1.n1+v2.n2+v3.n3

Puisqu'une norme est toujours positive, le signe du produit scalaire ne dépend que du facteur $\cos\theta$.

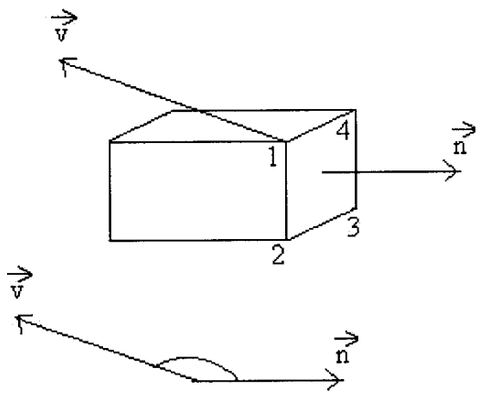
Si $0°<\theta<90°$	alors	$\cos\theta>0$	et	$\vec{v}.\vec{n}>0$
Si $90°<\theta<180°$	alors	$\cos\theta<0$	et	$\vec{v}.\vec{n}<0$

Considérons les deux figures suivantes. Pour chaque face testée nous associons au vecteur normal n un second vecteur v qui est le vecteur de vision.

C'est tout simplement un vecteur joignant n'importe quel sommet de la face regardée à l'œil de l'observateur. Si l'on ramène les deux vecteurs à une même origine, on a :



Face visible $\cos\theta>0$



Face invisible $\cos\theta<0$

CONCLUSION :

Pour déterminer les faces cachées d'un polyèdre convexe, on procède comme suit pour chacune des faces :

-On détermine le vecteur normal $\vec{n}$.

-On détermine le vecteur de vision $\vec{v}$.

-On calcule le produit scalaire $\vec{v} \cdot \vec{n}$. Si ce produit est négatif la face est alors invisible ; sinon elle est visible.

Cette méthode fonctionne bien pour les objets convexes, mais non pas pour les objets concaves.

Voici un autre algorithme qui résout bien ce problème : L'algorithme du peintre (***Z- Sorting***) :

VII.3. Z- SORTING

Pour chaque facette de la géométrie :

- calculer le centre de gravité.

- trier les facettes par rapport à la composante z du centre

- pour chaque facette de la plus lointaine à la plus proche : tracer la facette.

L'opération consiste à trier les facettes de la plus lointaine à la plus proche et ensuite de les dessiner une à une ; ainsi les facettes lointaines seront recouvertes par les plus proches.

CHAPITRE VIII.HEA3D

VIII.1. OUTILS UTILISES

Pour la réalisation de ce projet, nous avons utilisé comme langage de programmation, le langage C, et comme compilateur, nous avons préféré l'utilisation du GCC (GNU C Compiler).

VIII.1.1. LE LANGAGE C

Le C n'est pas orienté vers un domaine d'applications spéciales, comme par exemple le FORTRAN (applications scientifiques et techniques) ou le COBOL (applications commerciales ou traitant de grandes quantités de données). Le langage C utilise des expressions et des opérateurs qui sont très proches du langage machine. Il est possible de développer des programmes efficients et rapides.

VIII.1.2. GCC

GCC est le nom générique de la suite d'outils de développement, basée sur le compilateur C/C++ GNU développé par la Free Software Foundation.

Ce compilateur est très souple et peut être utilisé pour les différentes phases de production des binaires : preprocessing des fichiers sources, compilation, assemblage et édition de lien. Il dispose d'un grand nombre d'options, et peut être paramétré à l'aide d'un fichier de configuration.

VIII.1.3. ALLEGRO

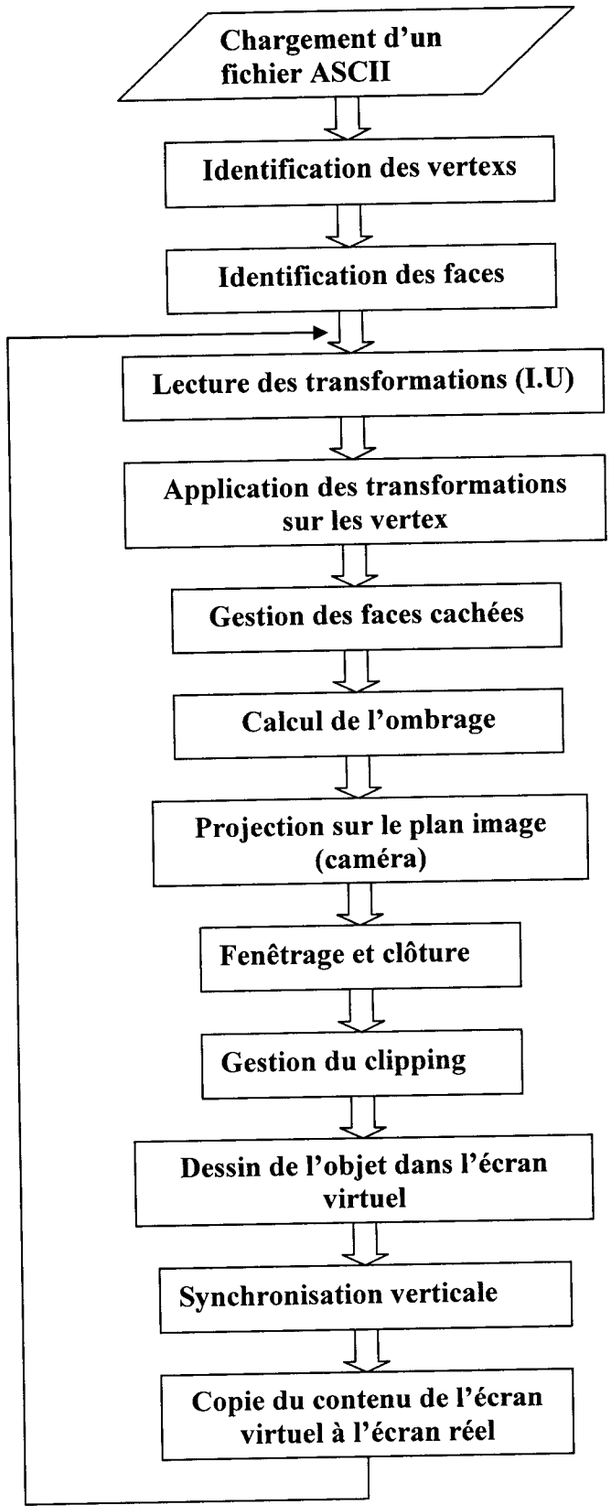
Notre travail étant en grande partie une programmation graphique, nous avons choisi d'utiliser une bibliothèque graphique ALLEGRO, cette librairie de développement supportant les plates-formes DOS, Unix (Linux, FreeBSD, Irix, Solaris), Windows, QNX et BeOS, le système MacOS.

Elle fournit de nombreuses fonctions graphiques et sonores, gère le clavier, la souris, le joystick et des timers haute résolution.

Elle dispose également de fonctions mathématiques 3d et en point fixe, de fonctions de gestion de fichiers de données compressées, et des fonctions pour la gestion des interfaces graphiques.

VIII.2. PRINCIPE DU HEA3D

Nous illustrons l'organigramme du HEA3d de la manière suivante :


Chargement d'un fichier ASCII
Identification des vertexs
Identification des faces
Lecture des transformations (I.U)
Application des transformations sur les vertex
Gestion des faces cachées
Calcul de l'ombrage
Projection sur le plan image (caméra)
Fenêtrage et clôture
Gestion du clipping
Dessin de l'objet dans l'écran virtuel
Synchronisation verticale
Copie du contenu de l'écran virtuel à l'écran réel

VIII.3. CHARGER UN FICHIER ASCII (3DSTUDIO MAX)

3D studio MAX est un logiciel de visualisation, de modélisation et d'animation d'objets tridimensionnels.

Il permet de créer en un temps restreint des objets 3D professionnels. Ce qui nous intéresse cependant est la possibilité qu'il nous offre d'exporter les objets en fichier standard ASCII (American Standard Code International Interchange).

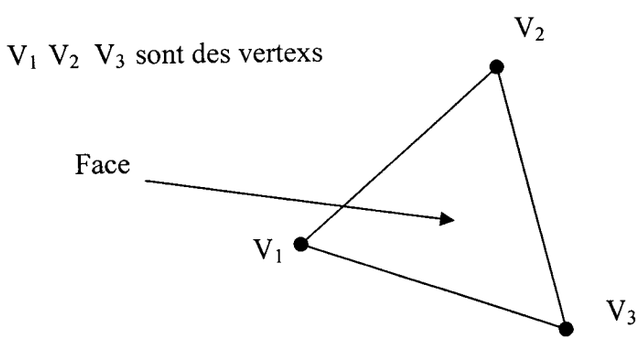
Ces fichiers sous l'extension (.asc) sont standards et peuvent être exportés dans d'autres Moteurs ou logiciels 3d.

Pour la conception du HEA3d, la lecture d'un fichier ASCII demeure primordiale. Pour cela nous avons développé une procédure permetant d'accomplir cette tâche.

Avant d'expliquer son fonctionnement voici un exemple sur la structure d'un fichier ASCII :

VIII.3.1. EXEMPLE D'UN FICHIER ASCII

```
Named Object: "Box"
Tri-mesh, Vertices: 8     Faces: 12
Vertex list:
Vertex 0: X: -10,000000     Y: -10,000000     Z: -10,000000
Vertex 1: X: 10,000000     Y: -10,000000     Z: -10,000000
Vertex 2: X: -10,000000     Y: 10,000000     Z: -10,000000
Vertex 3: X: 10,000000     Y: 10,000000     Z: -10,000000
Vertex 4: X: -10,000000     Y: -10,000000     Z: 10,000000
Vertex 5: X: 10,000000     Y: -10,000000     Z: 10,000000
Vertex 6: X: -10,000000     Y: 10,000000     Z: 10,000000
Vertex 7: X: 10,000000     Y: 10,000000     Z: 10,000000
Face list:
Face 0:    A:0 B:2 C:3 AB:1 BC:1 CA:0
Smoothing: 2
Face 1:    A:3 B:1 C:0 AB:1 BC:1 CA:0
Smoothing: 2
Face 2:    A:4 B:5 C:7 AB:1 BC:1 CA:0
Smoothing: 3
Face 3:    A:7 B:6 C:4 AB:1 BC:1 CA:0
Smoothing: 3
Face 4:    A:0 B:1 C:5 AB:1 BC:1 CA:0
Smoothing: 6
Face 5:    A:5 B:4 C:0 AB:1 BC:1 CA:0
Smoothing: 6
Face 6:    A:1 B:3 C:7 AB:1 BC:1 CA:0
Smoothing: 5
Face 7:    A:7 B:5 C:1 AB:1 BC:1 CA:0
Smoothing: 5
Face 8:    A:3 B:2 C:6 AB:1 BC:1 CA:0
Smoothing: 6
Face 9:    A:6 B:7 C:3 AB:1 BC:1 CA:0
Smoothing: 6
Face 10:    A:2 B:0 C:4 AB:1 BC:1 CA:0
Smoothing: 7
Face 11:    A:4 B:6 C:2 AB:1 BC:1 CA:0
Smoothing: 7
```



Un objet est défini par ses faces et chaque face est déterminée par trois vertexs.

Dans ce fichier nous remarquons la présence de plusieurs données nécessaires à la reconstitution de l'objet en question :

- Le nom de l'objet.

- le nombre de vertexs et de faces.

- Pour les vertexs : trois donnés X, Y et Z. Ce sont les coordonnées du vertex dans l'espace.

- Pour les faces: trois données A, B et C. Elles représentent les vertex qui délimitent ces faces.

Cette procédure stocke toutes ces données dans des variables et structures suivantes :

```
int NV ;                                    // Nombre de vertex
int NP ;                                    // Nombre de polygone

typedef struct VTX
{
 fixed x,y,z ;                        // Structure pour les vertex.
}VTX ;

typedes struct POLY
{
 VTX *vtxliste ;                      // Pointeur sur les VTX.
 int v1,v2,v3 ;                       // Structure pour les faces.
}

VTX *inpoints ; /* Pointeur sur les VTX, pour stocker les
coordonnées de tous les */
```

VIII.4. LECTURES DES TRANSFORMATIONS :

Le Hea3d propose plusieurs types de transformation : *les changements d'échelle, les translations, ou les rotations.*

ALLEGRO propose plusieurs fonctions pour la construction des matrices de transformations. Exemple :

Pour une matrice de rotation on appliquera :

Get_rotation_matrix(&m,rx,ry,rz) ;

- *m* est une variable de type *MATRIX,* ce type est pré- défini dans la bibliothèque *Allegro.h*

- *rx, ry* et *rz* : sont de type *Fixed* , *rx* est l'angle de rotation sur l'axe des x, *ry* sur l'axe des y et *rz* sur l'axe des z .

VIII.5. APPLICATION DES TRANSFORMATIONS SUR LES VERTEXS

Après avoir construit la matrice nécessaire à la transformation, nous devrons multiplier toutes les coordonnées des vertex par la matrice.

Avant d'utiliser la procédure qui effectue cette opération, nous déclarerons deux nouveaux pointeurs afin stocker les coordonnées de ces vertex après transformation :

```
VTX  *outpoints ;
POLY *outpoly ;

Cette procédure est pré- déclaré dans allegro.h :
Apply_matrix(&m,inx,iny,inz,&outx,&outy,&outz) ;
```

La procédure appliquera la matrice *m* sur le point (inx, iny, inz) et stockera le résultat dans le point (outx, outy, outz).

Afin d'appliquer cette matrice sur tous les points nous avons procéder de la manière suivantes :

```
For (i=0 ;i<NV ;i++)  {
   Apply_matrix(&m,inpoints[i].x,  inpoints[i].y,  inpoints[i].z,&outpoints.x,
&outpoints.y,

   &outpoints.z) ;
   outpoly[i].vtxlist = outpoints ;
                }
```

VIII.6. GESTION DES FACES CACHÉES

Cette procédure est divisée en deux étapes:

VIII.6.1. TEST DE VISIBILITÉ:

Nous devons calculer la normale de chaque face qui est représentée par ses trois vertexs (v1, v2, v3). Pour celà nous utiliserons une fonction pré-déclarée dans allegro.h, qui reçoit en parametre trois vecteurs vt1, vt2 et vt3. La normale de vt1 et vt2 sera affectée à vt3.

cross_product(vt1.x, vt1.y, vt1.z, vt2.x, vt2.y, vt2.z, &vt3.x, &vt3.y, &vt3.z);

sachant que: vt1 = v1- v2

vt2 = v3 -v2

Pour le test de visibilité de cette face : Si Vt3.z < = 0 la face n'est pas visible;

Sinon elle est visible;

VIII.6.2. TRI DES FACES SELON Z:

On calcule le centre de gravité de chaque face selon la composante z. On les triera ensuite par ordre décroissant.

Une fois ces deux étapes accomplies, on pourra dessiner les faces.

Cette procédure sera définie comme suit:

```
int *ord;              //pour stocker l'ordre des faces
fixed *grv;                    //Pour stocker le centre de gravité des faces selon z


void tripoly()
{
 int i,j,k;
 fixed g1,g2,g3,fi;

 ord=calloc(NP,sizeof(int));
 grv=calloc(NP,sizeof(int));

 for (i=0;i<NP;i++)
 {
    outpoly[i].vtxlist = outpoints;
    g1 = outpoly[i].vtxlist[outpoly[i].v1].z;
    g2 = outpoly[i].vtxlist[outpoly[i].v2].z;
    g3 = outpoly[i].vtxlist[outpoly[i].v3].z;
    grv[i] =itofix( fixtoi(g1+g2+g3)/3);

  }
    for (i=0;i<NP;i++) ord[i]=i;

    for (i=0;i<NP-1;i++)
     for (j=i+1;j<NP;j++)
        if (grv[i]<grv[j]) {
                                    k = ord[i];
                                    ord[i] = ord[j];
                                    ord[j] = k;

                                    fi = grv[i];
                                    grv[i]=grv[j];
                                    grv[j]=fi;

                                }
}
```

VIII.7. CALCUL DE L'EFFET D'OMBRAGE

Aprés le calcul de l'amplitude du vecteur perpendiculaire à la face, on divise la composante z sur l'amplitude, afin d'obtenir une valeur entre 0 et 1. Cette valeur indique le degré d'inclinaison, et la densité de la couleur.

```
void triangl(BITMAP *b, VTX *v1, VTX *v2, VTX *v3)
    static int xc = 0;
    char s[50];
    int col,v;
    float amp,f,xf,yf,zf;
    fixed x, y, z;
    fixed an;

    V3D vtx1 = { v1->x, v1->y, v1->z, 0, 0, 0};
    V3D vtx2 = { v2->x, v2->y, v2->z, 0, 0, 0};
    V3D vtx3 = { v3->x, v3->y, v3->z, 0, 0, 0};

    cross_product(
        v1->x - v2->x,      v1->y - v2->y,v1->z - v2->z,
        v3->x - v2->x,      v3->y - v2->y,      v3->z - v2->z,
        &x, &y, &z );

    if ( (fixtoi(z)<0) )
      return;

    xf = fixtof( x );
    yf = fixtof( y );
    zf = fixtof( z );

    amp = sqrt( xf*xf + yf*yf + zf*zf );        //calcule de l'amplitude
    f = fixtof( z ) / amp;                      // calcul du degré d'inclinaison
    v = 255 * f;
    col=makecol(v,v/3,v/3);

    vtx1.c = col;
    vtx2.c = col;
    vtx3.c = col;

    xc++;

    triangle3d(b, NULL, NULL, &vtx1, &vtx2, &vtx3);          //dessiner
la face.
  }
```

VIII.8. PROJECTION SUR L'IMAGE DE LA CAMERA

Avant d'afficher les points sur l'écran, nous devons prendre en consideration le point de vu de la camera,

ALLEGRO nous offre plusieurs fonctions destinées à la construction de la matrice, afin de translater tous les points de l'objet sur cette matrice, et de stocker le résultat dans deux nouveaux pointeurs :

```
VTX  *ocampoints;
POLY *ocampoly;
void camera()
{
 int d;
 MATRIX mc;                    //La matrice camera

 get_camera_matrix(&mc,pcx,pcy,pcz,xf,yf,zf,xup,yup,zup,champv,rapport);
    //fonction qui construit la matrice

 for (d=0;d<NV; d++)
 {
    apply_matrix(&mc,
      outpoints[d].x,      outpoints[d].y,      outpoints[d].z,
      &ocampoints[d].x,    &ocampoints[d].y,    &ocampoints[d].z);
    //appliquer la matrice sur les points

    persp_project(ocampoints[d].x, ocampoints[d].y, ocampoints[d].z,
    //projection des points 3 dimention
                          &ocampoints[d].x,&ocampoints[d].y);
 }
 for (d=0; d<NP ;d++)
 {
          ocampoly[d] = outpoly[d];
          ocampoly[d].vtxlist = ocampoints;
 }
```

VIII.9. FENETRAGE ET CLOTURE :

Nous devons spécifier la zone écran où sera dessiné notre objet (clôture). L'utilisation de la fonction **set_projection_viewport**(*int x, int y, int w, int h*) permettra de définir l'échelle du viewport .

x et y sont les coordonnées du point haut- gauche de la clôture.

w est la longueur de la clôture et h est sa largeur.

VIII.10. DESSINER L'OBJET SUR L'ECRAN VERTUEL

Dans la procédure *void* **triangl***()*, nous avons appelé en dernier lieu la fonction "triangle3d(b, NULL, NULL, &vtx1, &vtx2, &vtx3);"

Le premier argument de cette procédure est le bitmap où sera dessiné l'objet. b sera une zone mémoire dans la R.A.M qui contiendra une image de type bitmap.

VIII.11. SYNCHRONISATION VERTICALE

Cette synchronisation, permet d'éliminer le problème de déchirement ; nous utiliserons pour ce faire une fonction pré- déclarée dans ALLEGRO .

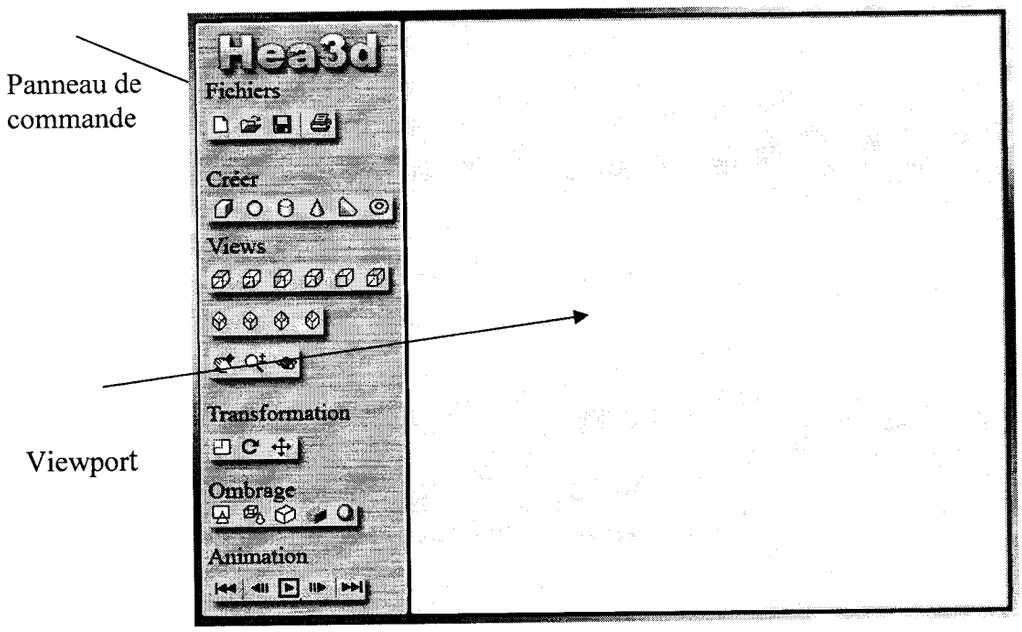
VIII.12. COPIE DU CONTENU DE L'ECRAN VIRTUEL A L'ECRAN REEL

Cette étape finale est réalisée grâce à la fonction **blit***(BITMAP *source, BITMAP *dest, int source_x, int source_y, int dest_x, int dest_y, int width, int height);* cette fonction réalise une copie rectangulaire de la zone source vers la zone de destination. Les paramètres *source_x* et *source_y* indiquent le coin haut gauche dans le bitmap ; les paramètres *dest_x* et *dest_y* indique quant à eux celui de destination. La taille de la zone est la largeur width * la hauteur height. Cette fonction réalise le clipping automatiquement.

VIII.13. DESCRIPTION DE L'INTERFACE UTILISATEUR

L'interface du HEA3D est divisé en deux parties: Le panneau de commande et le viewport.

VIII.13.1.1. Le panneau de commande



Le panneau de commande est constitué de plusieurs rubriques. Chaque rubrique regroupe un ensemble de boutons (fonctions).

LA RUBRIQUE FICHIERS

Cette rubrique contient des fonctions destinées à la gestion des fichiers.

Le bouton Nouveau : Cette fonction vide le contenu du viewport

Le bouton Ouvrir : Permet de sélectionner un fichier (.asc), et de le charger dans le viewport.

Le bouton Enregistrer : Enregistre la vu courante du viewport dans un fichier bitmap.

Le bouton imprimé : imprime la vu courante du viewport

LA RUBRIQUE CREER

Les boutons de cette rubrique permettent d'insérer des primitives standard ; cube, sphère, cylindre, cône, pyramide, ou torus.

LA RUBRIQUE VIEW

Ses boutons Sont destiné la gestion de la camera afin de choisir un point de vu sur l'objet. (Vu de Haut, Vu de bas, vu de gauche, vu de droite, vu de face, vu arrière, vu sud- ouest, vu sud- est ,vu nord- est et vu nord- ouest).

Le bouton de position (x, y) : change la position de la camera.

LA RUBRIQUE TRANSFORMATIONS

Chaque bouton fait subir une transformation sur l'objet subit.

Le bouton changement d'échelle: Agrandi ou diminue la taille de l'objet.

Le bouton rotation : Fait subir des rotations sur un l'un des axes (x, y, z).

LA RUBRIQUE OMBRAGE

Cette rubrique nous permet de sélectionner un type d'ombrage, fil de fer ou flate-shading.

La rubrique Animation

Permet de générer automatiquement des mouvement de rotation de la camera autour du centre de la scène.

CHAPITRE IX. CONCLUSION :

Le HEA3d permet ainsi de charger des fichiers standard (ASCII) importés par exemple de 3D studio MAX. Il permet également d'effectuer des tests de visibilité (grâce notamment à l'algorithme *Z-sorting)* et d'inclure des effets d'ombrages.

La finalité du programme est de projeter les objets importés sur écran. Ces objets peuvent être visualisés sous différents angles, grâce à la manipulation d'une camera par l'interface utilisateur.

Ce projet nous a permis particulièrement de renforcer nos connaissances sur la 3D. Nous espérons pour notre part qu'il pourra apporter quelques réponses aux programmeurs passionnés.

Ce projet n'est en fait que le début d'une longue et profonde exploration d'un domaine qui ne cesse de s'imposer. Aussi souhaitons-nous beaucoup de courage et d'abnégation à tous ceux qui désireraient s'initier à la magie de la 3D.

BILIOGRAOHIE :

[1] R.DONY. « *Graphisme scientifique 3ème édition*». MASSON 1988.

[2] M. ABRASH. « Zen d la programmation graphique 2ème *édition*». THAMSON Publishing1997.

[3] R. A. Plastock/ G.Kalley « Infographie Série Schaum». McGraw- Hill 1988.