

Formation sur les mathématiques liés à la 3D
SARL David Lanier 3D © Copyright 2007, tous droits réservés
Ce document ne peut être utilisé à des fins commerciales ou non, ni publié, ni modifié sans
l'autorisation de son auteur

Support de formation sur les mathématiques liés à la 3D

Par David Lanier de SARL David Lanier 3D

www.dl3d.com

Programmation informatique graphique : plug-ins, shaders

<u>Support de formation sur les mathématiques liés à la 3D.....</u>	<u>1</u>
<u>1.Introduction et notation.....</u>	<u>3</u>
<u>2.Vecteurs en dimension 3.....</u>	<u>3</u>
<u> 2.1.Addition de vecteurs.....</u>	<u>3</u>
<u> 2.2.Multiplication de vecteurs.....</u>	<u>3</u>
<u> 2.3.Norme d'un vecteur.....</u>	<u>3</u>
<u> 2.4.Produit scalaire.....</u>	<u>4</u>
<u> 2.5.Produit vectoriel.....</u>	<u>5</u>
<u>3.Les matrices.....</u>	<u>5</u>
<u>4.Les représentations dans IR 3.....</u>	<u>6</u>
<u> 4.1.Représentation d'un plan.....</u>	<u>6</u>
<u> 4.2.Représentation paramétrique d'un segment de droite et d'une droite.....</u>	<u>7</u>
<u>5.Représentation des rotations.....</u>	<u>8</u>
<u> 5.1.Matrices 3x3.....</u>	<u>8</u>
<u> 5.2.Angles d'Euler.....</u>	<u>10</u>
<u> 5.3.Axe et angle.....</u>	<u>11</u>
<u> 5.4.Quaternions.....</u>	<u>11</u>
<u> 5.4.1.Interpolation des quaternions : SLERP.....</u>	<u>12</u>
<u> 5.4.2.Interpolation des quaternions : SQUAD.....</u>	<u>13</u>
<u> 5.5.Vecteur angulaire.....</u>	<u>13</u>
<u>6.Changement de repère.....</u>	<u>15</u>
<u> 6.1.Repère local d'un objet.....</u>	<u>15</u>
<u> 6.2.Matrices 4x4.....</u>	<u>17</u>
<u>7.Exercices pratiques résolus / tips.....</u>	<u>17</u>
<u> 7.1.Projection d'un point sur plan.....</u>	<u>17</u>
<u> 7.2.Intersection segment / plan.....</u>	<u>18</u>
<u> 7.3.Un exemple 2D : Changement de repère.....</u>	<u>20</u>
<u> 7.4.Un exemple 2D : Intersection segment / segment.....</u>	<u>21</u>
<u> 7.5.Construire une matrice de rotation à partir d'un vecteur.....</u>	<u>22</u>
<u> 7.6.Faire un billboard.....</u>	<u>23</u>
<u> 7.7.Collisions Rayons / Faces.....</u>	<u>24</u>
<u> 7.8.Quelques notions de balistique (dynamique du point) - Particules.....</u>	<u>24</u>
<u>8.Remerciements.....</u>	<u>25</u>
<u>9.Références.....</u>	<u>25</u>

1. Introduction et notation

Le but de ce cours d'initiation aux maths 3D est de donner les bases de la 3D. Comme il serait possible d'écrire des dizaines de livres sur ce sujet, nous nous concentrerons ici sur les connaissances nécessaires à la résolution des problèmes pratiques que l'on rencontre le plus souvent dans les jeux vidéo.

Donc nous allons passer en revue beaucoup de domaines de la 3D sans réellement en approfondir un particulièrement, mais nous aurons de ce fait une vision de ce qui existe et à quoi cela sert. Si le lecteur veut approfondir certains de ces domaines, des références de sites web sont données à la fin de ce support.

Dans tout ce qui suit, on notera « SQRT » la fonction racine carré.

2. Vecteurs en dimension 3

On appelle vecteur v l'ensemble $\{x,y,z\}$ avec $x,y,z \in \mathbb{R}$. On notera $\{0,0,0\}$ le vecteur nul origine du repère. Les axes X , Y et Z représentant une base canonique de \mathbb{R}^3 et sont $X = \{1,0,0\}$, $Y = \{0,1,0\}$, $Z = \{0,0,1\}$.

Pour accéder à l'un des composants de ce vecteur on utilisera la notation « . » exemple, la première composante de v sera $v.x$.

Un vecteur est en fait toujours un point de l'espace auquel on soustrait l'origine (le vecteur nul). On fait donc souvent l'amalgame entre un point dans l'espace 3D et un vecteur dans ce même espace. En fait, le point 3D « v » peut être vu comme le vecteur v représenté par $v = v - \{0,0,0\}$

2.1. Addition de vecteurs

On définit l'addition de 2 vecteurs, notée $+$ par l'application qui a 2 vecteurs associe un vecteur ($\mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$). Donc le résultat de l'addition de vecteurs est un vecteur.

Soit $\forall u$ et $v \in \mathbb{R}^3$

$$u + v = w \text{ avec } w = \{u.x + v.x, u.y + v.y, u.z + v.z\}$$

2.2. Multiplication de vecteurs

On définit la multiplication de 2 vecteurs, notée $*$ par l'application qui a 2 vecteurs associe un vecteur ($\mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$). Donc le résultat de l'addition de vecteurs est un vecteur.

Soit $\forall u$ et $v \in \mathbb{R}^3$

$$u * v = w \text{ avec } w = \{u.x * v.x, u.y * v.y, u.z * v.z\}$$

Remarque : on peut aussi multiplier le vecteur par un réel (le « scalar ») ce qui nous donne pour $a \in \mathbb{R}$:

$$a*u = \{a * u.x, a * u.y, a * u.z\}$$

2.3. Norme d'un vecteur

On définit la norme d'un vecteur, notée $\| \|$ par l'application qui a un vecteur associe un réel $\| \| : \mathbb{R}^3 \rightarrow \mathbb{R}$

Soit $\forall v \in \mathbb{R}^3$

$$\|v\| = \text{SQRT} (v.x^2 + v.y^2 + v.z^2)$$

Remarque : Cette formule vient de la formule de Pythagore en décomposant le vecteur sur chaque axe.

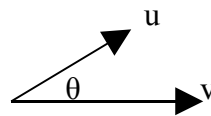
Normaliser un vecteur consiste à calculer sa norme puis à diviser tous les composants par la norme du vecteur. Dans la pratique : on utilise souvent la norme au carré d'un vecteur pour éviter le calcul de la racine carré.

2.4. Produit scalaire

On définit le produit scalaire de 2 vecteurs, noté « . » par l'application qui a 2 vecteurs associe un réel ($\mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$). Donc le résultat du produit scalaire de 2 vecteurs est un réel.

Soit $\forall u$ et $v \in \mathbb{R}^3$

$$u \cdot v = u.x * v.x + u.y * v.y + u.z * v.z$$



Remarque : le produit scalaire est symétrique : $u \cdot v = v \cdot u$, et $u \cdot u = \|u\|^2$.

De manière trigonométrique, en appelant θ l'angle formé par u et v , on a aussi la formule suivante :

$$u \cdot v = \|u\| * \|v\| * \cos \theta$$

Remarque : Grâce à cette formule, on peut déterminer l'angle modulo 2π entre 2 vecteurs.

On peut aussi déterminer si les vecteurs pointent dans le même sens où s'ils sont orthogonaux etc.. S'ils pointent dans le même sens, l'angle sera compris entre $[-\pi/2, +\pi/2]$ (soit $[-90^\circ, +90^\circ]$) donc le cosinus sera positif. En résumé :

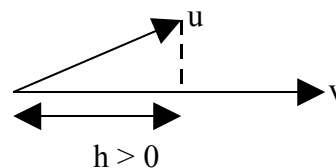
Si $u \cdot v > 0$ les vecteurs « pointent dans la même direction » à 90° près.

Si $u \cdot v = 0$ les vecteurs sont orthogonaux ($\cos \theta = 0 \Leftrightarrow \theta = \pi/2 + k\pi, k \in \mathbb{Z}$ (entiers))

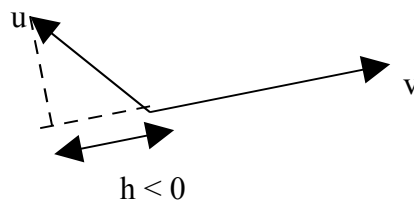
Si $u \cdot v < 0$ les vecteurs ne « regardent » pas dans la même direction.

De manière analytique, le produit scalaire représente la projection d'un vecteur sur un autre orthogonalement, à condition que l'un des 2 vecteurs soit normalisé.

exemple avec $\|v\| = 1$, si on définit $h \in \mathbb{R}$ par $h = u \cdot v$. On a la représentation suivante :



Dans le cas précédent $h > 0$, dans le suivant $h < 0$.



Remarque : Autre vision du vecteur $v = \{x,y,z\}$ on peut en fait décomposer v dans la base de \mathbb{R}^3 (X,Y,Z) de la manière suivante (X, Y et Z sont les axes classiques de \mathbb{R}^3) :

$$v = \{ v \cdot X, v \cdot Y, v \cdot Z \}$$

2.5. Produit vectoriel

On définit le produit vectoriel de 2 vecteurs, noté « \wedge » par l'application qui à 2 vecteurs associe un vecteur orthogonal ($\mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$). Donc le résultat d'un produit vectoriel est un vecteur orthogonal aux 2 vecteurs dont on a fait le produit.

$$u \wedge v = w \text{ avec } w = \{ u.y * v.z - u.z * v.y, u.z * v.x - u.x * v.z, u.x * v.y - u.y * v.x \}$$

Remarque : le produit vectoriel n'est pas symétrique. On a $u \wedge v = -v \wedge u$

De manière trigonométrique, en appelant θ l'angle formé par u et v , on a aussi la formule suivante en posant $w = u \wedge v$:

$$\|w\| = \|u\| * \|v\| * |\sin \theta|$$

Attention, comme w est un vecteur, c'est ici sa norme qui permet de retrouver le sin θ en valeur absolue. Donc cela n'est jamais suffisant pour déterminer l'angle θ , contrairement au produit scalaire.

Comme résultat, on a un vecteur w qui est orthogonal à la fois à u et à v (même si u et v ne sont pas orthogonaux entre eux).

3. Les matrices

On ne va pas étudier ici toutes les généralités sur les matrices, mais se concentrer sur ce qui nous servira le plus souvent.

On appelle matrice un tableau constitué de m lignes et n colonnes

$$A = (a_{ij})_{m \times n} = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

où les a_{ij} sont des réels est appelé matrice $m \times n$.

On peut additionner des matrices les multiplier etc... On ne détaillera pas ce genre de choses ici.

Voici la matrice identité, elle n'a que des 0 sauf sur la diagonale où se trouvent des 1 :

$$E = E_{n \times n} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$$

Transposé de matrices : Pour transposer une matrice on fait $a_{ij} = a_{ji}$ pour tout i et j . Donc seule la diagonale reste inchangée, c'est une sorte de symétrie par rapport à la diagonale.

Quelques propriétés de la multiplication de matrices :

- i) Le produit de matrice n'est pas commutatif. En général $AB \neq BA$

- ii) Loi associative: $(AB)C = A(BC)$
- iii) Loi distributive: $A(B+C) = AB + AC$
- iv) $AE = EA = A$ avec E matrice identité.
- v) $(AB)^T = B^T A^T$

Matrice Inverse :

La matrice carrée $n \times n$ A est dite inversible, s'il existe une matrice B avec la propriété
 $AB = BA = E$.

Une telle matrice B est unique. On appelle B l'inverse de A et on la note A^{-1} . La matrice A^{-1} est appelée l'inverse de A .

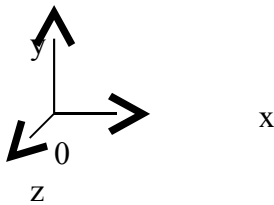
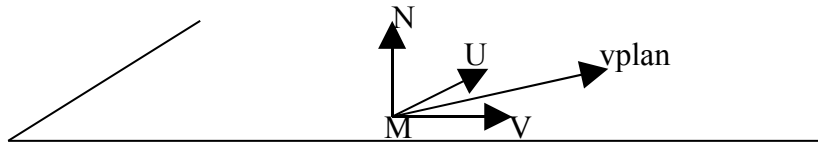
4. Les représentations dans \mathbb{R}^3

Pour représenter un point on considère son vecteur.

4.1.Représentation d'un plan

Un plan dans \mathbb{R}^3 est un espace de dimension 2 infini, il est donc engendré par 2 vecteurs non colinéaires à partir d'un point précis (cela permet de le fixer).

- On peut donc le représenter par 1 point 3D dit « origine » M et 2 vecteurs dit générateurs U et V . Soient 3 vecteurs 3D en tout.
- Mais de manière plus simple, on utilise souvent la représentation faisant uniquement intervenir 2 vecteurs 3D, soient un point 3D origine M et un vecteur qui représente la normale au plan N .



On a alors : $\forall v = \{x,y,z\} \in \mathbb{R}^3$ v est un point 3D appartenant au plan :
 On note v_{plan} le vecteur du plan $v - M$. On a alors :

$$v_{\text{plan}} \cdot N = 0$$

et aussi

$$v_{\text{plan}} = a * U + b * V \text{ avec } a \text{ et } b \in \mathbb{R}$$

Ce qui veut dire que v_{plan} s'exprime en fonction des 2 vecteurs générateurs du plan. Plus précisément, il est une combinaison linéaire de U et V .

L'équation d'un plan dans \mathbb{R}^3 se traduisant de manière analytique par :

Tout vecteur $v = \{x,y,z\} \in \mathbb{R}^3$ étant dans dans le plan doit vérifier

Or on a

$$v_{\text{plan}} = v - M = \{ x - M.x, y - M.y, z - M.z \}$$

Et en utilisant la formule $v_{\text{plan}} \cdot N = 0$

On a l'équivalence suivante :

$$v_{\text{plan}} \cdot N = 0 \Leftrightarrow (x - M.x) * N.x + (y - M.y) * N.y + (z - M.z) * N.z = 0$$

Cette équation nous permet de déterminer a, b, c et d avec :

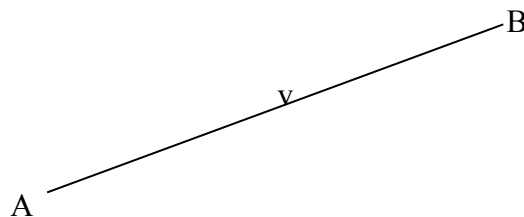
$$a = N.x, b = N.y, c = N.z \text{ et } d = -(M \cdot N)$$

Donc de manière générale l'équation du plan à partir de sa normale et d'un point s'écrit analytiquement :

$$N.x * x + N.y * y + N.z * z = -(M \cdot N) \text{ avec } \{x,y,z\} \text{ formant un vecteur dans le plan}$$

4.2.Représentation paramétrique d'un segment de droite et d'une droite

On représente un segment de droite noté v par 2 points qui sont ses extrémités:

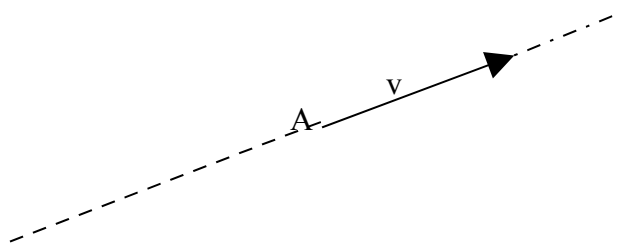


On utilise une représentation paramétrique du segment sous cette forme :

$$v(x) = A + x * (B - A) \text{ avec } x \text{ variant de } 0 \text{ à } 1.$$

$$\text{Pour } x = 0, v(0) = A, \text{ pour } x = 1 v(1) = B.$$

Pour représenter une droite, il faut 1 point 3D noté A et un vecteur qui donne la direction noté v :



De la même manière que le segment de droite, la droite de manière paramétrée s'écrit :

$$V(x) = A + x * v \text{ avec } x \in \mathbb{R}.$$

L'intérêt de ces représentations est qu'elles sont valables aussi bien pour des points 3D que 2D.

5. Représentation des rotations

Les rotations peuvent être représentées par différents moyens dans \mathbb{R}^3 , une matrice 3x3, un quaternion, un axe et un angle, des angles d'Euler, un vecteur angulaire. Nous allons étudier chacun de ces moyens avec leur défaut et leurs avantages. On peut passer de l'un à l'autre, le détail de ces transformations ne sera pas donné, voir les références pour le détail.

On reconnaît une rotation au fait que lorsque l'on l'applique sur un vecteur, la norme du vecteur reste inchangé.

En général, on utilise dans une interface utilisateur les angles d'Euler car ils sont les plus intuitifs puis on passe à une représentation différente en interne.

5.1. Matrices 3x3

C'est le moyen le plus connu de représenter une rotation. On a survolé les matrices précédemment, ici on va se concentrer sur les matrices 3x3.

Une matrice 3x3 c'est un tableau $A = (a_{ij})$ avec $0 \leq i, j \leq 2$

On note les vecteurs colonnes $v_0 = \{ a_{00}, a_{10}, a_{20} \}$ $v_1 = \{ a_{01}, a_{11}, a_{21} \}$ et $v_2 = \{ a_{02}, a_{12}, a_{22} \}$. Ce sont les colonnes de cette matrice de haut en bas.

$$A = \begin{matrix} & \begin{matrix} v_0 & v_1 & v_2 \end{matrix} \\ \begin{matrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{matrix} \end{matrix}$$

Est-ce toujours une rotation ?

Non. Les matrices 3x3 ne représentent des rotations que si elles sont orthonormales, c'est-à-dire que l'on a la propriété suivante :

$$A A^T = Id \quad \text{avec Id matrice identité.}$$

On appelle cela aussi une matrice homogène.

C'est à dire que l'inverse de cette matrice est la transposée de celle-ci.

Ceci se traduit sur les vecteurs colonnes v_0 , v_1 et v_2 de la matrice par la propriété suivante : Tous ces vecteurs sont normalisés (ont une norme de 1) et sont tous orthogonaux entre eux, càd :

$$v_0 \cdot v_1 = v_0 \cdot v_2 = v_1 \cdot v_2 = 0$$

Pour effectuer ceci, en pratique on fait par exemple :

```
void OrthonormalizeOrientation( matrix_3x3 &Orientation )
{
    vector_3 X(Orientation(0,0),Orientation(1,0),Orientation(2,0));
    vector_3 Y(Orientation(0,1),Orientation(1,1),Orientation(2,1));
    vector_3 Z;

    X.Normalize();
    Z = CrossProduct(X,Y).Normalize();
    Y = CrossProduct(Z,X).Normalize();
}
```

Orientation(0,0) = X(0); Orientation(0,1) = Y(0); Orientation(0,2) = Z(0);
Orientation(1,0) = X(1); Orientation(1,1) = Y(1); Orientation(1,2) = Z(1);
Orientation(2,0) = X(2); Orientation(2,1) = Y(2); Orientation(2,2) = Z(2);
 }

Propriétés :

- Pour inverser une rotation, il suffira de transposer la matrice de rotation de l'objet.
- Pour combiner 2 rotations représentées par des matrices 3x3 A et B, il suffit de les multiplier, la multiplication conserve l'orthonormalisation.

Remarque sur les matrices, avec les notations précédentes :

$$A = \begin{matrix} & \begin{matrix} v0 & v1 & v2 \end{matrix} \\ \begin{matrix} \overline{a_{00}} & \overline{a_{01}} & \overline{a_{02}} \\ \overline{a_{10}} & \overline{a_{11}} & \overline{a_{12}} \\ \overline{a_{20}} & \overline{a_{21}} & \overline{a_{22}} \end{matrix} \end{matrix}$$

En fait le vecteur v0 représente la transformation du vecteur X = {1,0,0} par cette matrice, en effectuant le calcul :

$$A X = v0$$

De la même manière on a v1 = AY et v2 = AZ. Ceci sert en pratique pour la construction de matrices spécifiques.

Par exemple :

Je veux construire une matrice de rotation qui à mon axe X fait correspondre Y et à Y fait correspondre -X tout en laissant Z inchangé (rotation autour de l'axe Z de 90°). Cela me donne :

v0 = AX et je veux que cela me donne Y donc v0 = Y, de même v1 = -X et v2 = Z (inchangé) ce qui me donnera la matrice suivante :

$$A = \begin{matrix} \overline{0} & \overline{-1} & \overline{0} \\ \overline{1} & \overline{0} & \overline{0} \\ \overline{0} & \overline{0} & \overline{1} \end{matrix}$$

Attention dans certains cas les matrices sont transposées par rapport à celles-ci (Direct X par exemple).

Avantages :

- Pas de problèmes de gimbal lock (bloqué dans une orientation sans pouvoir en bouger, voir les angles d'Euler).
- Les matrices de rotation simples comme l'exemple précédent sont facilement reconnaissables. (Je sais ça fait maigre comme avantage ☺)

Inconvénients par rapport aux autres représentations :

- Plus coûteux en mémoire.
- Orthonormalisation plus coûteuse en temps CPU.

Ce document ne peut être utilisé à des fins commerciales ou non, ni publié, ni modifié sans l'autorisation de son auteur

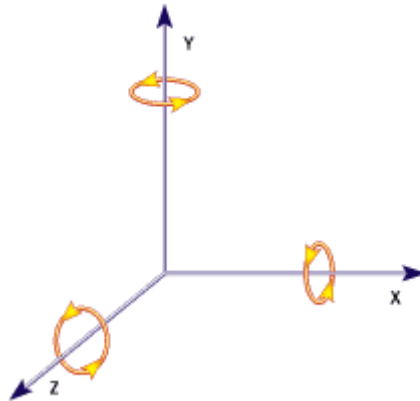
- Erreurs d'arrondis peuvent engendrer une matrice non orthonormale lors de combinaisons de matrices et donc translater ou scaler l'objet au lieu de seulement le tourner.
- L'interpolation de 2 rotations de manière smooth est coûteuse (calcul de vecteurs propres) et n'apparaît pas aussi smooth qu'avec un quaternion.

5.2. Angles d'Euler

Les angles d'Euler sont des angles qui représentent des rotations autour des axes X, Y et Z dans un ordre précis.

Soit un angle d'Euler $E = \{60, 89, 120\}$, c'est une rotation qui veut dire que l'on a tourné d'un angle de 60° autour de X puis 89° autour de Y puis 120° autour de Z. Ces angles peuvent aussi être exprimés en radians.

GD,9801, Fig1



Attention, l'ordre de rotation autour des axes est important, **car quand on tourne autour d'un axe, tous les autres axes tournent avec !**

On doit définir si l'on tourne en premier sur X puis ensuite sur Y et enfin sur Z si on change cet ordre cela ne donnera pas la même orientation finale dans le cas général. Il y a 12 cas d'ordre possibles de rotation.

Dans Maya par exemple, on peut définir l'ordre dans lequel appliquer les rotations.

Il n'existe pas une manière unique de représenter une rotation donnée avec les angles d'Euler.

Les angles d'Euler peuvent être convertis en matrices 3x3 à l'aide des matrices de rotation autour des axes X, Y et Z. Les 3 matrices suivantes représentent de gauche à droite :

Rotation d'un angle θ autour de X	idem autour de Y	idem autour de Z.
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin -\theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$	$\begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ \sin -\theta & 0 & \cos \theta \end{bmatrix}$	$\begin{bmatrix} \cos \theta & \sin -\theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Avantages :

- 3 variables seulement sont utilisées pour représenter les 3 degrés de liberté.
- Il n'y a pas à appliquer une quelconque transformation pour en faire une rotation, c'est toujours une rotation.
- Représentation assez intuitive de la rotation.

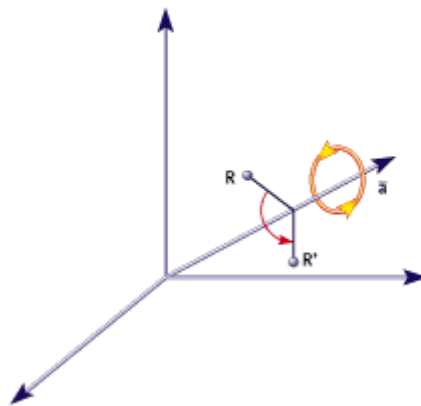
Inconvénients :

- Pas de combinaison des rotations sous cette forme.
- On peut tomber dans un cas où on se retrouve bloqué dans un plan (appelé le Gimbal Lock problem) si on utilise les rotations autour des axes, X, Y et Z.
- L'interpolation smooth fait intervenir de l'intégration numérique ce qui peut être coûteux en CPU.

5.3.Axe et angle

C'est une représentation assez intuitive aussi.

GD,9801, Fig2



On définit la rotation appliquée sur le vecteur R par une rotation d'un certain angle autour d'un axe arbitraire.

Avantages :

- Il n'y a pas à appliquer une quelconque transformation pour en faire une rotation, c'est toujours une rotation.
- Représentation assez intuitive de la rotation.

Inconvénients :

- Pas de combinaison de rotation sous cette forme.
- L'interpolation smooth fait intervenir de l'intégration numérique ce qui peut être coûteux en CPU.
- Gimbal lock problem.

5.4.Quaternions

C'est une des représentations les plus utilisées. Un quaternion c'est un vecteur de \mathbb{R}^4 . Il est donc composé de 4 réels :

$$q = \{ x, y, z, w \} \text{ avec } x, y, z, w \text{ réels.}$$

q s'écrit aussi :

$$q = \{ v, w \} \text{ avec } v \text{ vecteur tel que } v = \{x, y, z\} \text{ et } w \text{ réel.}$$

Les quaternions sont à l'origine une extension des nombres complexes. Nous n'aborderons pas les détails de cela dans ce support de cours.

Est-ce qu'un quaternion est toujours une rotation ?

Non.

De la même manière que les matrices doivent être orthonormales pour représenter une rotation, les quaternions doivent être unitaires.

On définit la norme d'un quaternion par la norme d'un vecteur de \mathbb{R}^4 :

$$\|q\| = \text{SQRT} (x^2 + y^2 + z^2 + w^2)$$

Et on appelle quaternion unitaire un quaternion q tel que $\|q\| = 1$. C'est la représentation d'une rotation.

On ne va pas détailler ici toutes les opérations que l'on peut faire sur les quaternions et comment elles se font, la littérature à ce sujet est déjà dense.

Le quaternion peut aussi être vu à partir d'un axe et d'un angle de cette manière :

Si on a une rotation représentée par un axe v et un angle θ , le quaternion correspondant s'écrit :

$$q = \{ \sin (\theta/2) * v, \cos (\theta/2) \}$$

soit

$$q = \{ x*\sin (\theta/2) , y*\sin (\theta/2) , z*\sin (\theta/2) , \cos (\theta/2) \}$$

Pour composer des rotations sous forme de quaternion, on les multiplie par l'opérateur de multiplication des quaternions (non détaillée ici). Attention de manière générale la multiplication des quaternions n'est pas symétrique : $q_1 * q_2 \neq q_2 * q_1$.

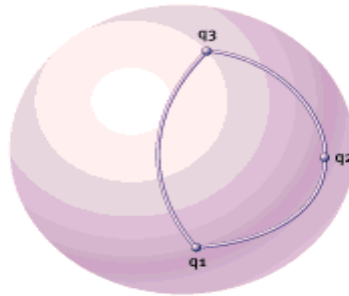
Pour inverser une rotation, on inverse le quaternion en multipliant par -1 chaque composante de l'axe.

5.4.1. Interpolation des quaternions : SLERP

La méthode la plus connue est le SLERP (Spherical Linear intERPolation).

$$\text{SLERP}(p, q, t) = \frac{p \sin((1-t)\theta) + q \sin(t\theta)}{\sin(\theta)}$$

Cette méthode permet de passer du quaternion p au quaternion q de manière linéaire (sur une sphère). En gros on trace une droite que l'on projette sur la sphère.



5.4.2. Interpolation des quaternions : SQUAD

L'autre méthode qui est la plus utilisée est le SQUAD (Spherical Cubic Interpolation). Elle est mathématiquement identique à une interpolation de Bezier ou d'Hermite cubique (spline). On l'utilise par exemple dans les courbes d'animation avec une interpolation de type TCB pour les rotations.

Pour « a » quaternion de départ et « b » quaternion d'arrivée, on utilise les tangentes en a notée p et en b notée q (dans le cas des rotations, les tangentes d'une clé d'animation sont des rotations aussi la tangente représentant une « direction à suivre ») :

$$\text{squad}(t, a, p, q, b) = \text{slerp}(2t(1-t), \text{slerp}(t, a, b), \text{slerp}(t, p, q)) \text{ avec } t \in [0,1]$$

On a la possibilité de contrôler la vitesse de départ et d'arrivée du quaternion grâce aux tangentes. Comme on le voit le squad est coûteux en temps CPU et aussi en mémoire si l'on veut stocker les tangentes.

En résumé, pour les quaternions :

Avantages :

- L'interpolation donne un très bel effet.
- Pas de Gimbal lock problem.
- On peut combiner les rotations en multipliant les quats.

Inconvénients :

- Pas de représentation intuitive.

5.5. Vecteur angulaire

La représentation d'une rotation se fait sous la forme d'un vecteur de \mathbb{R}^3 , $v = \{x,y,z\}$ x,y et z réels. Cette représentation est similaire à celle avec un axe et un angle.

- v représente l'axe autour duquel s'effectue la rotation.
- l'angle autour de l'axe v est placé dans la norme du vecteur v.

Pour combiner les rotations, il suffit d'ajouter les vecteurs angulaires.

Avantages :

Ce document ne peut être utilisé à des fins commerciales ou non, ni publié, ni modifié sans l'autorisation de son auteur

- Cette représentation est utile pour contraindre des rotations suivant un axe et un angle, il suffit dans ce cas, de minorer/majorer les composantes du vecteur.
- Ca sert aussi dans certains cas à différencier une matrice de rotation pour avoir la vitesse angulaire (c'est le vecteur angulaire). (Voir <http://www.martinb.com> et la partie Rotation)

Inconvénients :

?

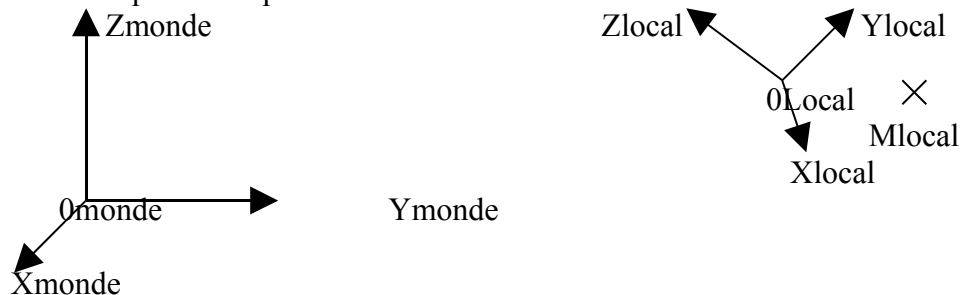
6. Changement de repère

6.1. Repère local d'un objet

Lorsqu'un objet noté OBJ est placé dans un monde 3D. Il existe au moins 2 repères. Un repère de référence appelé le repère du monde et le repère local de l'objet OBJ. On notera Mlocal un point exprimé par un vecteur en local dans l'objet. Quand on dit que ce vecteur est exprimé en local cela veut dire que Mlocal est exprimé en fonction des vecteurs Xlocal, Ylocal et Zlocal :

$M_{local} = x_l * X_{local} + y_l * Y_{local} + z_l * Z_{local}$ ou x_l, y_l et z_l sont des réels.

Ce vecteur ne « connaît pas » le repère du monde.



Il existe entre les 2 repères une translation, un scale et une rotation. Cet ensemble noté transformation permet de changer de repère.

Chaque objet possède une transformation, cela permet de passer du repère de l'objet dans le monde.

Par exemple le point Mlocal dans le repère du monde s'écrit :

$$M_{local \text{ exprimé dans le monde}} = \text{TransfodeOBJ} * M_{local}$$

On notera SOBJ le vecteur scale de l'objet OBJ, TOBJ le vecteur translation

(= $0_{Local} \text{ exprimé dans le monde} - 0_{monde}$) de OBJ et ROBJ sa rotation.

Soit :

$$M_{local \text{ exprimé dans le monde}} = ROBJ * (M_{local} * SOBJ) + TOBJ$$

On applique d'abord le scale puis la rotation et on ajoute ensuite la translation.

Comme on l'a vu précédemment, ROBJ peut-être une matrice, un quaternion etc... Dans ce cas l'opération * entre ROBJ et le vecteur $M_{local} * SOBJ$ utilise l'opérateur * des matrices ou des quaternions...

Inversement, si on a un point Mmonde exprimé dans le repère du monde, pour l'exprimer en fonction du repère local de OBJ, il suffit d'inverser la transformation OBJ et de l'appliquer

sur Mmonde, le vecteur résultant sera $M_{monde \text{ exprimé dans le repère OBJ}}$.

Pour inverser une transfo on fait :

Si Scale = {sx, sy, sz} et que sx, sy et sz sont non nuls, on a ScaleInverse = {1/sx, 1/sy, 1/sz}

L'inverse de la translation T est -T. Et pour inverser la rotation, utiliser l'opérateur inverse de la représentation choisie, dans le case des matrices 3x3 il suffit de transposer la matrice.

Remarque : Pour passer une normale définie dans un repère quelconque dans un autre repère, on ne fait qu'appliquer des rotations, il ne faut pas appliquer de translation ou de scale, sinon ce n'est plus une normale.

Exemple : Nlocal est la normale d'une face exprimée dans le repère local de l'objet, je veux l'exprimer en fonction du monde, je dois faire

$$N_{\text{local}}_{\text{exprimée dans le monde}} = R_{\text{local}} * N_{\text{local}}$$

Tips (En pratique) :

Quand on a des algorithmes du genre :

```
for (int i=0 ;i<NumNormalsMesh1 ;i++)
{
    const Vec3f& localnormalMesh1 = mesh1.GetNormal(i) ;
    for (int j=0 ;j<NumNormalsMesh2 ;j++)
    {
        const Vec3f& localnormalMesh2 = mesh2.GetNormal(j) ;
        CompareNormals(localnormalMesh1, localnormalMesh2) ;
    }
}
```

Bien sûr ceci est faux, car il faudrait passer les normales dans le même repère pour les comparer. Pour ceci on a le choix de passer chaque normale dans le monde ce qui revient à faire :

```
Transfo Transfomesh1 = mesh1.GetTransfo() ;
Transfo Transfomesh2 = mesh2.GetTransfo() ;
Rotation Rmesh1 = Transfomesh1.GetRotation();
Rotation Rmesh2 = Transfomesh2.GetRotation();

for (int i=0 ;i<NumNormalsMesh1 ;i++)
{
    Vec3f localnormalMesh1 = mesh1.GetNormal(i);
    //Passage dans le monde de la normale du mesh 1
    localnormalMesh1= Rmesh1 * localnormalMesh1;

    for (int j=0 ;j<NumNormalsMesh2 ;j++)
    {
        Vec3f localnormalMesh2 = mesh2.GetNormal(j) ;

        //Passage dans le monde de la normale du mesh 2
        localnormalMesh2 = Rmesh2 * localnormalMesh2;

        CompareNormals(localnormalMesh1, localnormalMesh2) ;
    }
}
```

On remarque qu'à chaque passage dans la boucle sur le mesh 2 on est obligé de passer la normale par la rotation du mesh 2 pour l'exprimer dans le monde.

Et ça va consommer du temps CPU !

Alors que la solution idéale consiste à exprimer les normales du mesh 1 dans le repère du mesh 2 comme suit :

```
Transfo Transfomesh1 = mesh1.GetTransfo() ;
Transfo Transfomesh2 = mesh2.GetTransfo() ;
Rotation Rmesh1 = Transfomesh1.GetRotation();
Rotation Rmesh2 = Transfomesh2.GetRotation();
```

Rotation FromMesh1ToMesh2 = Rmesh1 * (Rmesh2.Inverse())

```
for (int i=0 ;i<NumNormalsMesh1 ;i++)
{
    Vec3f normalMesh1InMesh2Coords = mesh1.GetNormal(i);

    //Passage de la normale du mesh 1 dans le repère du mesh 2
    normalMesh1InMesh2Coords = Rotation FromMesh1ToMesh2 * localnormalMesh1;

    for (int j=0 ;j<NumNormalsMesh2 ;j++)
    {
        Vec3f localnormalMesh2 = mesh2.GetNormal(j) ;

        //Plus besoin de transformer localnormalMesh2 car tout est dans le repère de
        //Mesh 2
        CompareNormals(normalMesh1InMesh2Coords, localnormalMesh2) ;
    }
}
```

Ceci est évidemment similiaire s'il s'agissait de vertices, on ne prendrait pas que la rotation mais toute la transfo que l'on appliquerait de la même manière pour passer d'un repère à l'autre.

6.2. Matrices 4x4

Pour simplifier les calculs de transformation des points, on utilise plutôt en pratique des matrices 4x4. Ces matrices contiennent la translation, la rotation et le scale à la fois. Toute les opérations telles que translation, rotation et scale d'un vecteur peuvent être représentées sous forme de matrices 4x4 et ensuite assemblées (multipliées) sous forme d'une seule matrice 4x4 représentant la transformation complète.

$$R = \begin{bmatrix} r_{00} & r_{01} & r_{02} & 0 \\ r_{10} & r_{11} & r_{12} & 0 \\ r_{20} & r_{21} & r_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad S = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Et

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix}$$

Une fois la matrice composée de ces 3 matrices, pour l'appliquer sur un vecteur de dimension 3, on construit le vecteur de dimension 4 en lui rajoutant 1 comme dernière composante.

Ce qui nous donne un vecteur de dimension 4 : [x y z 1]

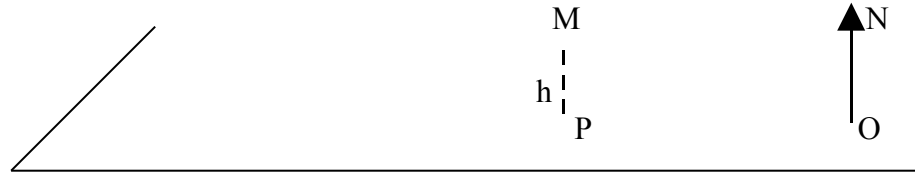
7. Exercices pratiques résolus / tips

7.1. Projection d'un point sur plan

Soit un plan, on a vu précédemment que ce plan peut s'exprimer sous forme d'un vecteur 3D noté O origine du plan et d'un vecteur normal au plan noté N :

On note M un point 3D quelconque, on cherche le point P projection orthogonale de M sur le plan.

Exprimer P en fonction de M, N et O.



Une solution :

On a vu précédemment que le produit scalaire d'un vecteur v et d'un vecteur unitaire noté N nous donne la projection de ce vecteur sur N.

En prenant $v = M - O$ et N la normale du plan comme vecteur unitaire, on obtient

$$h = v \cdot N \text{ avec } h \text{ réel.}$$

$$h = (M - O) \cdot N$$

h représente la distance orthogonale entre le point M et le plan, grâce à cette distance, on peut projeter le point M sur le plan :

$$P = M - h \cdot N$$

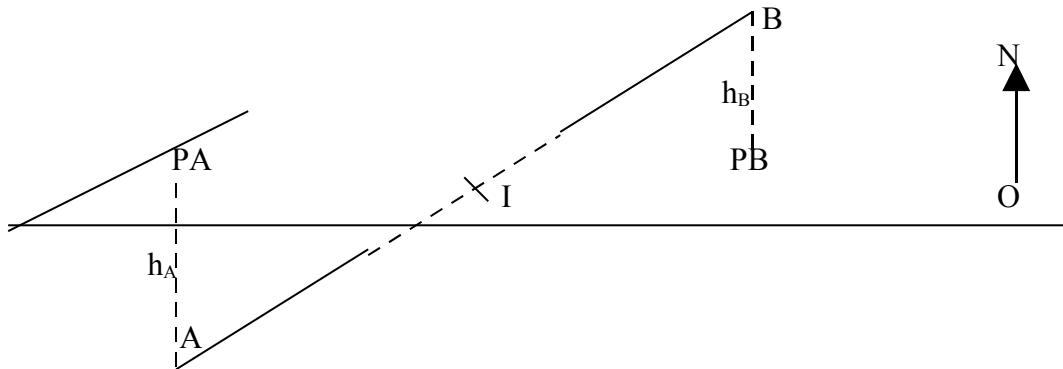
Soit exprimé en fonction de M, N et O uniquement :

$$P = M - ((M - O) \cdot N) \cdot N$$

7.2. Intersection segment / plan

On a un plan défini comme précédemment par les 2 vecteurs $O = \{x_0, y_0, z_0\}$ et N. Soient les 2 points $A = \{x_A, y_A, z_A\}$ et $B = \{x_B, y_B, z_B\}$. Ils forment un segment noté [AB] qui coupe le plan au point $I = \{x_i, y_i, z_i\}$.

Trouver les coordonnées de I en fonction de O, N, A et B.



Une solution :

Exprimons l'équation du plan :

Si un vecteur $M = \{x, y, z\}$ est dans le plan, il vérifie l'équation du plan vue précédemment qui est :

$$N \cdot x (x - x_0) + N \cdot y (y - y_0) + N \cdot z (z - z_0) = 0$$

Donc I doit vérifier cette équation. Soit :

$$N \cdot x (x_i - x_0) + N \cdot y (y_i - y_0) + N \cdot z (z_i - z_0) = 0$$

Or I est un point du segment [AB], on peut donc l'exprimer sous forme paramétrique comme on l'a vu précédemment :

$I = A + t (B - A)$ avec un t fixé compris entre 0 et 1, en fait trouvé les coordonnées de I revient à trouver le paramètre t.

Soit sous forme décomposée sur chaque variable, I s'écrit :

$$x_i = x_A + t (x_B - x_A)$$

$$y_i = y_A + t (y_B - y_A)$$

$$z_i = x_A + t (z_B - z_A)$$

On pose le vecteur $\Delta = (B - A)$ pour simplifier. On a donc :

$$x_i = x_A + t \Delta.x$$

$$y_i = y_A + t \Delta.y$$

$$z_i = x_A + t \Delta.z$$

En remplaçant maintenant x_i , y_i et z_i dans l'équation du plan on a :

$$N.x (x_A + t \Delta.x - x_0) + N.y (y_A + t \Delta.y - y_0) + N.z (z_A + t \Delta.z - z_0) = 0$$

En extrayant t de l'équation, cela nous donne :

$$t (\Delta.x \cdot N.x + \Delta.y \cdot N.y + \Delta.z \cdot N.z) = - (N.x (x_A - x_0) + N.y (y_A - y_0) + N.z (z_A - z_0))$$

D'où :

$$t = (N.x (x_0 - x_A) + N.y (y_0 - y_A) + N.z (z_0 - z_A)) / (\Delta.x \cdot N.x + \Delta.y \cdot N.y + \Delta.z \cdot N.z)$$

En remarquant que le numérateur est le produit scalaire de N par le vecteur (O-A), c'est l'opposé de la distance orthogonale du plan au point A notée h_A . Et en remarquant que le dénominateur est le produit scalaire de B-A et de N, en notant la distance de B au plan h_B On a au final, on obtient un résultat très simple :

$$t = h_A / (h_A - h_B)$$

Comme vérification, on doit avoir $t \in [0,1]$ et h_A et h_B de signe contraires.

Remarque : si h_A et h_B sont de même signe, il n'y a pas d'intersection avec le plan, ils sont tous les 2 au-dessus ou en dessous.

7.3. Un exemple 2D : Changement de repère

Avec un repère 2D (O, i, j) , on cherche à passer dans le repère (O, u, v) , c'est à dire changer de base. Ceci est utilisé pour la mapping.

Trouver les coordonnées du point $M(x,y)$ x, y réels, du repère (O,i,j) dans le repère (O,u,v)



Une solution :

On a dans le repère (O,i,j) : $OM = x*i + y*j$.

On a aussi $u = u_i * i + u_j * j$ et $v = v_i * i + v_j * j$

Il nous reste à trouver i et j en fonction de u et v , donc résoudre le système d'équations linéaires d'ordre 2 :

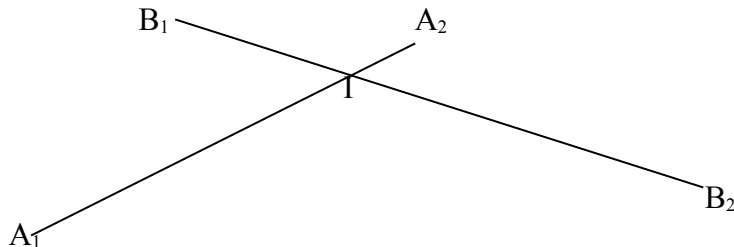
$$u = u_i * i + u_j * j$$

$$v = v_i * i + v_j * j$$

Une fois trouvé i et j en fonction de u et de v , on remplace les valeurs de i et j dans l'équation $OM = x * i + y * j$.

7.4. Un exemple 2D : Intersection segment / segment

Ce n'est pas un problème 3D, mais il est intéressant car assez similaire aux problèmes 3D que vous pouvez rencontrer. Soient 2 segments 2D $[A_1A_2]$ et $[B_1B_2]$.



Trouver l'intersection I de ces 2 segments en fonction de A_1, A_2, B_1, B_2 .

Une solution :

Toujours sous forme paramétrique :

I est sur $[B_1B_2]$ il s'exprime donc en paramétrique par :

$$I = B_1 + t_B (B_2 - B_1)$$

I est aussi sur $[A_1A_2]$ donc:

$$I = A_1 + t_A (A_2 - A_1)$$

Notez que le paramètre t est différent sur chaque segment. Dans le cas général, $t_A \neq t_B$.

On pose les vecteurs $V_A = A_2 - A_1$ et $V_B = B_2 - B_1$

En remplaçant ceci dans les 2 équations précédentes, on obtient :

$$I = A_1 + t_A V_A = B_1 + t_B V_B$$

On a donc une équation à 2 inconnues t_A et t_B . En projetant cette équation sur l'axe X et sur l'axe Y, on obtient un système linéaires de 2 équations à 2 inconnues :

$$A_1.x + t_A V_{A.x} = B_1.x + t_B V_{B.x}$$

$$A_1.y + t_A V_{A.y} = B_1.y + t_B V_{B.y}$$

Soit :

$$t_A V_{A.x} - t_B V_{B.x} = B_1.x - A_1.x$$

$$t_A V_{A.y} - t_B V_{B.y} = B_1.y - A_1.y$$

Ou sous forme matricielle, en posant le vecteur $T = \{ t_A, t_B \}$ le vecteur $R = B_1 - A_1$

Et la matrice 2x2

$$M = \begin{vmatrix} V_{A.x} & -V_{B.x} \\ V_{A.y} & -V_{B.y} \end{vmatrix}$$

On obtient le système linéaire matriciel suivant :

$$MT = R$$

Si le déterminant 2x2 de M est non nul, l'équation a une unique solution qui se trouve en inversant la matrice M, l'inverse de M étant noté M^{-1}

On a

$$T = M^{-1} R$$

Remarque, seule la connaissance d'une des 2 valeurs T_A ou T_B nous suffit pour trouver l'intersection I.

En notant « det » le déterminant de 2 vecteurs 2D.

Nous obtenons donc pour t_A par exemple :

$$t_A = \det((B1 - A1) , v_B) / \det (v_A, v_B)$$

7.5. Construire une matrice de rotation à partir d'un vecteur

Soit un vecteur V exprimé dans le repère du monde. On cherche à orienter un objet dans la même direction que ce vecteur pour que l'objet « regarde » dans cette direction. L'objet a pour repère les 3 vecteurs orthonormés X, Y, Z .



Construire une matrice 3x3 de rotation notée R pour orienter XYZ à partir de V .
 On note X' Y' et Z' les vecteurs X Y et Z après leur avoir appliqué la rotation cherchée.
 Soient

$$\begin{aligned} X' &= R * X \\ Y' &= R * Y \\ Z' &= R * Z \end{aligned}$$

On doit avoir au final :



Une solution :

On cherche donc la matrice 3x3 R . En premier si V n'est pas normalisé, on le normalise.
 On doit avoir l'axe $X = \{1, 0, 0\}$ de l'objet orienté dans la même direction que V , on a vu que cela revient à écrire que :

$$X' = V = R * X$$

Ce qui est équivalent à dire que la première colonne de R est le vecteur V (voir remarque sur les matrices de rotation)

Pour trouver Y' , on sait que Y' est un vecteur orthogonal à X' , on prend donc arbitrairement $Y' = \{-X'.y, X'.x, 0\}$ (attention si $X'.x = X'.y = 0$, on pose $X'.x = 1$). Puis on fait $Y' = Y' / \|Y'\|$ pour avoir un vecteur normalisé.

On peut vérifier que le produit scalaire de X' et Y' est nul.

Puis pour déterminer Z' on utilise le produit vectoriel entre X' et Y' .

Soit

$$Z' = X' \wedge Y'$$

Par conséquent notre matrice de rotation se déduit de X' , Y' et Z' qui sont respectivement la première colonne, la deuxième et la troisième de R .

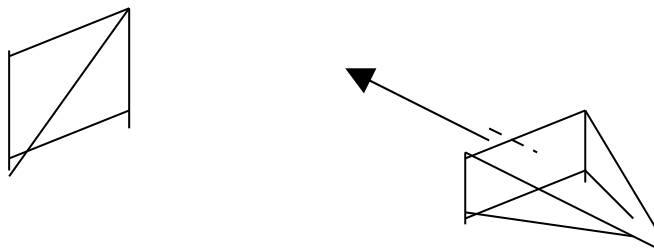
Soit

$$R = \begin{bmatrix} V.x & Y'.x & (X' \wedge Y').x \\ V.y & Y'.y & (X' \wedge Y').y \\ V.z & 0 & (X' \wedge Y').z \end{bmatrix} \begin{matrix} = X' \\ = Y' \\ = Z' \end{matrix}$$

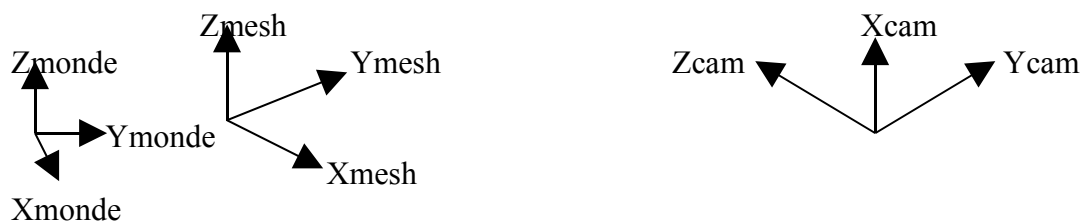
7.6. Faire un billboard

Il s'agit ici d'orienter un objet 3D en fonction de l'orientation d'une caméra. L'objet doit toujours être face à la caméra dans toutes les situations.

L'objet à gauche est un mesh formé de 2 triangles. Celui de droite est une caméra avec le vecteur indiquant la direction de la vue.



Ceci peut se représenter sous forme de repères avec la Transfo du mesh et celle de la caméra :



Orienter le mesh en fonction de l'orientation de la caméra.

Une solution :

Il s'agit ici de ne modifier que l'orientation du mesh, par conséquent trouver une matrice de rotation en fonction de la vue de la caméra sera suffisant.

Il nous faut déterminer le vecteur direction de la caméra dans le monde (Zcam sur le schéma), il indique dans quelle direction « regarde » la caméra.

Le scale et la translation de la caméra nous importent peu, seule sa rotation sous forme de matrice notée Rcam nous intéresse.

Calculons le vecteur Zcam dans le monde. Dans ce cas, il est utile d'avoir représenté Rcam sous forme d'une matrice 3x3.

Il est facile de vérifier qu'avec $Zcam = \{0,0,1\}$ dans le repère de la caméra, si on calcule le vecteur :

$$Zcam_{\text{ dans le repère du monde }} = Rcam * \{0,0,1\}$$

on trouve la dernière colonne de la matrice (ou ligne dans la notation Américaine des matrices)

Comme on veut que l'objet soit orienté vers la caméra et non dans le même sens, il nous faut prendre comme vecteur d'orientation du mesh l'opposé de Zcam dans le monde :

soit $-Z_{cam}$ dans le repère du monde.

Grâce à ce vecteur, il ne nous reste plus qu'à construire une matrice de rotation pour orienter notre mesh. Et en utilisant l'exemple précédent, on calcule cette matrice.

7.7. Collisions Rayons / Faces

Décrire l'algorithme utilisé pour faire des collisions rayon / face.

Une solution :

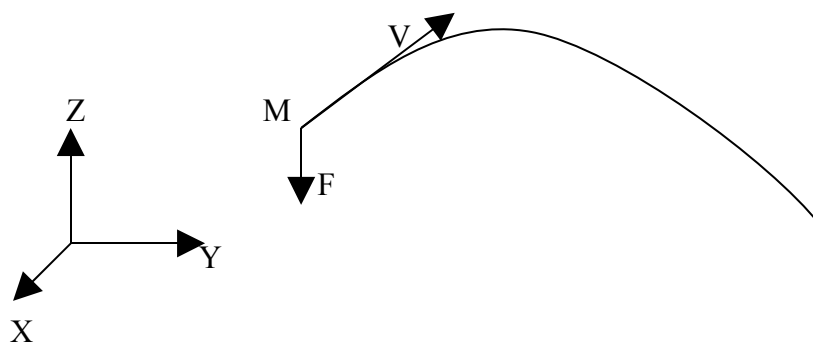
Le rayon est représenté par un segment de droite en paramétrique. Il suffit avec ce qui précède de faire l'intersection entre le segment et le plan dans lequel se trouve la face. Pour définir le plan à partir d'une face, on prend la normale de la face comme normale du plan et un des vertex de cette face comme origine du plan.

Si le rayon intersecte le plan, il reste à déterminer si l'intersection se trouve à l'intérieur de la face, pour cela il y a plusieurs méthodes possibles.

Par exemple :

- Calculer dans le plan, les vecteurs normaux aux edges de la face : le produit scalaire entre l'origine du edge et la normale à l'edge doit être négatif si le point est à l'intérieur. (l'intérêt de cette méthode est que l'on peut stocker les normales aux edges)
- On peut aussi utiliser le produit vectoriel entre un edge et le vecteur formé par le point moins l'origine du edge, le vecteur résultant devrait se trouver du même côté de la normale pour tous les edges s'il est à l'intérieur.

7.8. Quelques notions de balistique (dynamique du point) - Particules



On considère un objet de masse m dans un le repère du monde, il peut être décrit comme un point 3D noté M si l'on ne considère pas son orientation. On cherche à lancer cet objet dans une direction V_0 (c'est sa vitesse initiale). Cet objet est soumis à une force F (généralement la gravité). Trouver en fonction du temps comment faire évoluer la position de cet objet, en fait il s'agit ici de décrire l'algorithme utilisé en pratique pour faire bouger l'objet.

Une solution :

Avec le principe fondamental de la dynamique, on a

$$\sum F = m a$$

ou F représente les forces appliquées sur M, m est la masse de l'objet et a est son accélération. Ceci nous permet de déterminer a, par intégration, on a la vitesse v :

$$v(t) = \int a(t) dt$$

Et enfin la position en intégrant la vitesse :

$$p(t) = \int v(t) dt$$

Comment se traduit ceci en pratique :

Comme on ne peut intégrer ces équations sur tout l'intervalle de temps à la fois pour avoir une formule générale, on casse le temps en petits intervalles et on considère que :

Si les forces représentées par F, ne varient pas au cours du temps, avec la formule $\sum F = m a$ on a une accélération constante au cours du temps :

$$a = \sum F / m$$

Pour trouver la vitesse en fonction du temps, on utilise la formule d'intégration d'Euler :

$$v(t + \Delta t) = v(t) + \Delta t * (dv(t + \Delta t) / dt)$$

Or $dv(t + \Delta t) / dt = a(t + \Delta t) = a$ (constant par rapport à t)

Donc la vitesse s'écrit :

$$v(t + \Delta t) = v(t) + \Delta t * a$$

En pratique, on peut prendre $\Delta t = 1 / 30$ secondes.

Ensuite le même manière, pour la position on a :

$$p(t + \Delta t) = p(t) + \Delta t * (dp(t + \Delta t) / dt)$$

Or $dp(t + \Delta t) / dt = v(t + \Delta t)$ que l'on a déjà calculé, d'où :

$$p(t + \Delta t) = p(t) + \Delta t * v(t + \Delta t)$$

Remarque :

On est toujours obligé de stocker l'état précédent de l'objet (car pour calculer la nouvelle vitesse et position il nous faut la vitesse et la position précédente).

Cette méthode d'Euler est peu précise mais elle est rapide et convient parfaitement dans ce cas tant que l'on ne veut pas calculer de manière plus exacte la trajectoire. Sinon il faudra utiliser une autre méthode d'intégration comme Runge Kutta d'ordre 4 (RK4) par exemple. On ne rentrera pas dans les détails de ces autres méthodes.

8. Remerciements

Merci à André Bertrand, Jean-François Lopez et Nicolas Dalstein de Kalisto pour leurs remarques et corrections.

9. Références

- Approfondir vos connaissances les quaternions (opérations, interpolation etc...) :
http://www.gamasutra.com/features/19980703/quaternions_01.htm ou
http://freefall.freehosting.net/articles/quaternions_1.html ou
<http://www.gamedev.net/reference/programming/features/whyquats/>

Ce document ne peut être utilisé à des fins commerciales ou non, ni publié, ni modifié sans l'autorisation de son auteur

- La physique selon Chris Hecker (avec exemples de code fournis dont celui sur l'orthonormalisation des matrices 3x3) : <http://www.d6.com/users/checker/>
- Approfondir les angles d'Euler : http://vered.rose.utoronto.ca/people/david_dir/GEMS/GEMS.html
- Quickmaths permet de calculer sur le web par exemple des déterminants de matrices 3x3, son but est de résoudre des problèmes simples de maths sur le web : <http://www.quickmath.com> aussi sur <http://www.calc101.com/>
- Des tas de formules, de leçons sur l'algèbre : <http://www.algebrahelp.com/>
- Leçons de maths : <http://www.sosmath.com/index.html>
- Archives de maths : <http://archives.math.utk.edu/calculus/crol.html>
- Distance entre ligne et point http://geometryalgorithms.com/Archive/algorithm_0102/algorithm_0102.htm
- **Du code pour tout ce qui est géométrie dont certains sont applicables aux jeux : <http://www.magic-software.com>**
- Initiation aux maths 3D : <http://www.geocities.com/pcgpe/graphics.html> ou ici avec des petits tests de questions : <http://chortle.ccsu.ctstateu.edu/vectorLessons/tutorialIndex.html>
- **Tout le process d'un jeu du point de vue programmeur : <http://www.nihilistic.com/GDC2000/GDC2000Tutorial/index.htm>**
- 3D, physique, cinématique etc... <http://www.martinb.com>
- A propos des splines (TCB, Hermite, Bezier...) <http://www.cubic.org/~submissive/sourcerer/hermite.htm>
- Interpolation de matrices polynômiales (article dur) http://www.nd.edu/~pantsakl/elecpubs/IJC_whole.pdf
- Matrices et quaternions faq : <http://skal.planet-d.net/demo/matrixfaq.htm> (Faites gaffe y a des conneries dedans, sur le déterminant au moins)
- Numerical recipes in C : <http://lib-www.lanl.gov/numerical/bookcpdf.html> (Il est aussi dans Tools à Kalisto)
- Méthodes de calcul numérique avec programmes en C : <http://www.univ-lille1.fr/eudil/jbeuneu/index.html>