Rotations

Rotations

Rotations 3D ont 3 degrés de libertés, Représentation non unique

Matrice

$$R = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix}$$

$R^T\, R = \mathrm{I}$

$\det(R) = 1$

Euler Angles

3 angles: (α, β, γ)

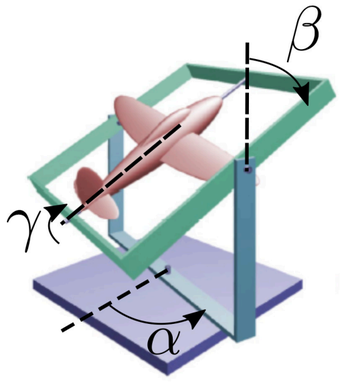
Composition de rotation autour d'axes définis

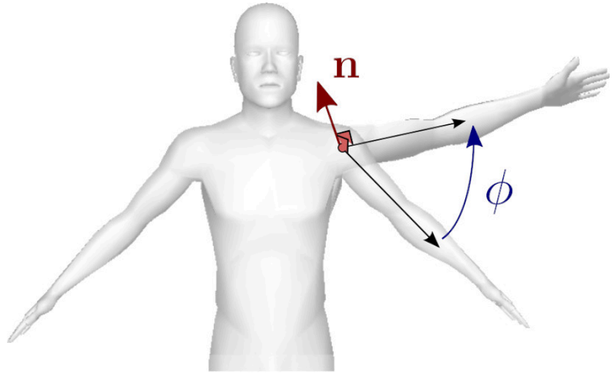
Axis angle

$(\mathbf{n}, \theta)$

Quaternion

$q = (x, y, z, w)$
$= \left(\mathbf{n} \sin\left(\frac{\theta}{2}\right), \cos\left(\frac{\theta}{2}\right)\right)$





Rotation: Focus angles d'Euler

Trois rotations consécutives autour d'axes fixes orthogonaux.

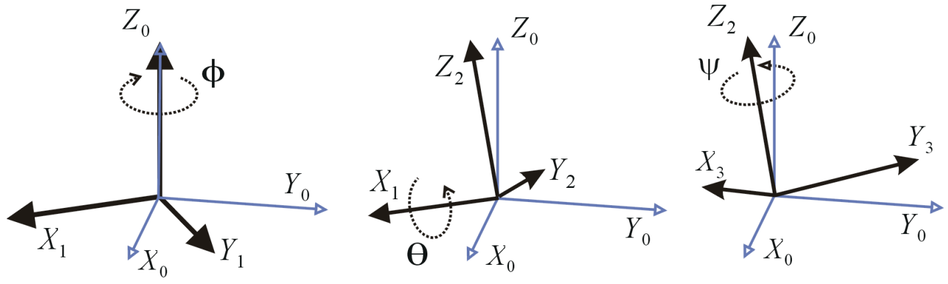
Représentation sous forme de multiplication de matrices

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin\alpha & \cos(\alpha) \end{pmatrix} \quad R_y = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \quad R_z = \begin{pmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Conventions multiples pour les angles d'Euler

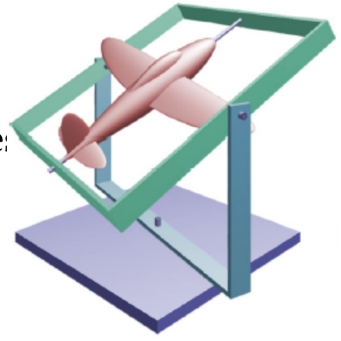
- *Proper Euler : z-x-z', x-y-x', y-z-y', ...*
- *Trait-Bryan : x-y-z, z-y-x, x-z-y, ...*

Attention lors de l'export/import/parsing entre différent logiciels



Pro

- Composition de rotations autour d'axes fixes
- Paramètres compréhensibles (3 dof)
- Les animateurs peuvent interagir avec les courbes angulaires
- *Largement utilisé en robotique*



Rotation: Focus angles d'Euler

Limitations des angles d'Euler

- *Gimbal Lock* lors de la composition de certains rotations

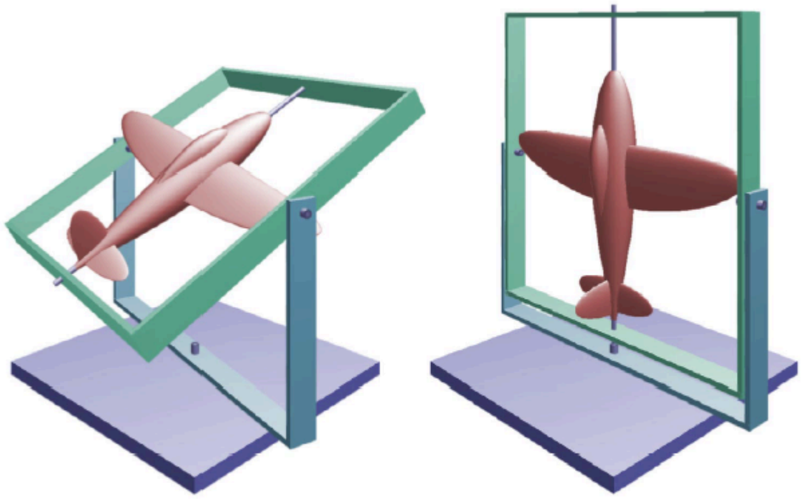
  Perte d'un degré de liberté de certaines configurations

ex. $R_x(\alpha)\,R_y(\pi/2)\,R_z(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}$

$$= \begin{pmatrix} 0 & 0 & 1 \\ \sin(\alpha)\cos(\gamma) + \cos(\alpha)\sin(\gamma) & -\sin(\alpha)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & 0 \\ -\cos(\alpha)\cos(\gamma) + \sin(\alpha)\sin(\gamma) & \cos(\alpha)\sin(\gamma) + \sin(\alpha)\cos(\gamma) & 0 \end{pmatrix}$$

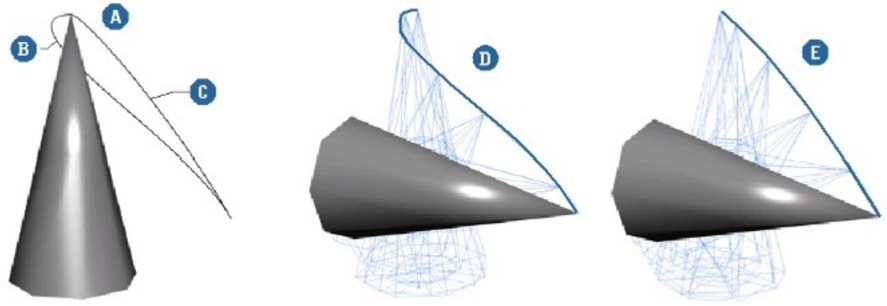
$$= \begin{pmatrix} 0 & 0 & 1 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 \\ -\cos(\alpha+\gamma) & \sin(\alpha+\gamma) & 0 \end{pmatrix}$$

Souhaite 2-dof, mais on a uniquement 1-dof



http://www.fho-emden.de/~hoffmann/gimbal09082002.pdf

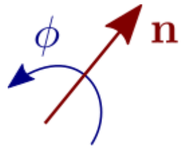
- Interpolation de 3 angles possibles, mais pas nécessairement avec la trajectoire la plus simple.



Rotation: Focus Axe Angle

Une rotation 3D quelconque peut être représentée par

- A axe unitaire n
- Un angle θ

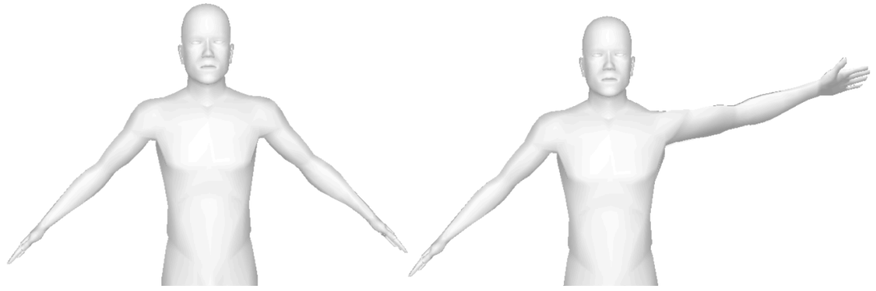


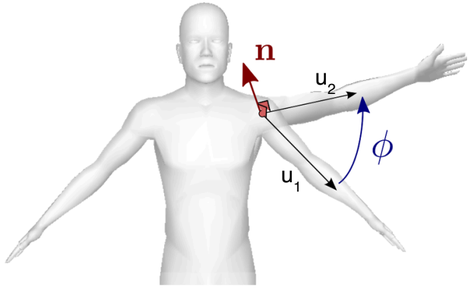
Pro.

- Représentation concise: 3 DOF
- Paramètres compréhensibles
- Décrit aisément la rotation entre deux vecteurs

Ex. Rotation entre u_1 and u_2

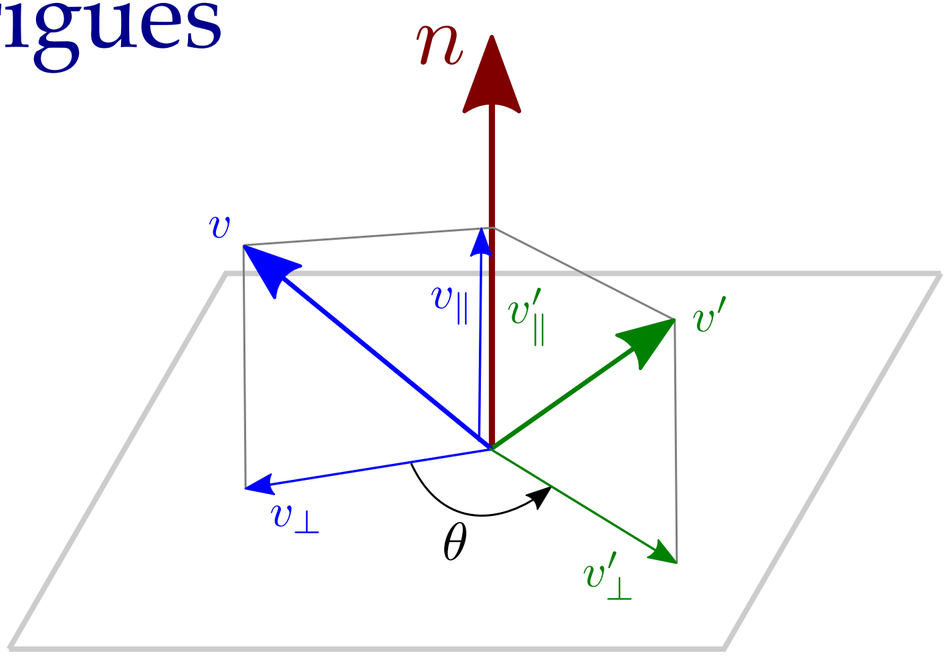
- Axe de rotation $n = (u_1 \times u_2)/\|u_1 \times u_2\|$
- Angle de rotation $\theta = \text{acos}(u_1 \cdot u_2)$
- Twist autour de u_2 peut être pris arbitrairement





Rotation: Focus Axe Angle - Rodrigues

Appliquer une rotation (n, θ) au vecteur v



$$v = v_{\parallel} + v_{\perp}$$
$$v' = v'_{\parallel} + v'_{\perp}$$
$$\Rightarrow v' = v_{\parallel} + (\cos(\theta)\, v_{\perp} + \sin(\theta)\, n \times v_{\perp})$$

$$v_{\parallel} = (v \cdot n)\, n$$
$$v_{\perp} = v - (v \cdot n)\, n$$

$$v' = (v \cdot n)\, n + \cos(\theta)(v - (v \cdot n)\, n) + \sin(\theta)\, n \times (v - (v \cdot n)\, n)$$

$$\Rightarrow v' = \cos(\theta)\, v + \sin(\theta)\, n \times v + (1 - \cos(\theta))\, (v \cdot n)\, n$$

Formule de **Rodrigues**

Rotation: Focus Axe Angle en Matrice

$$v' = \cos(\theta)\, v + \sin(\theta)\, n \times v + (1 - \cos(\theta))\, (v \cdot n)\, n = R(n, \theta)\, v$$

$$v' = \cos(\theta) v + \sin(\theta) \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \times v + (1 - \cos(\theta)) \begin{pmatrix} n_x & n_y & n_z \end{pmatrix} v \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

$$v' = \cos(\theta)\, v + \sin(\theta) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathrm{K}} v + (1 - \cos(\theta)) \underbrace{\begin{pmatrix} n_x^2 & n_x\, n_y & n_x\, n_z \\ n_x\, n_y & n_y^2 & n_y\, n_z \\ n_x\, n_z & n_y\, n_z & n_z^2 \end{pmatrix}}_{\mathrm{K}^2} v$$

$$v' = \begin{pmatrix} \cos(\theta) + n_x^2(1 - \cos(\theta)) & n_x n_y (1 - \cos(\theta)) - n_z \sin(\theta) & n_x n_z (1 - \cos(\theta)) + n_y \sin(\theta) \\ n_x n_y (1 - \cos(\theta)) + n_z \sin(\theta) & \cos(\theta) + n_y^2 (1 - \cos(\theta)) & n_y n_z (1 - \cos(\theta)) - n_x \sin(\theta) \\ n_x n_z (1 - \cos(\theta)) - n_y \sin(\theta) & n_y n_z (1 - \cos(\theta)) + n_x \sin(\theta) & \cos(\theta) + n_z^2 (1 - \cos(\theta)) \end{pmatrix} v$$

Rotation: Focus Axe Angle - Résumé

Etant donné une rotation (n, θ) - La matrice de rotation correspondante

$$R(n,\theta) = \cos(\theta)\,\mathrm{I} + \sin(\theta)\,\mathrm{K} + (1-\cos(\theta))\,\mathrm{K}^2$$

$$R(n,\theta) = \begin{pmatrix} \cos(\theta) + n_x^2(1-\cos(\theta)) & n_x n_y(1-\cos(\theta)) - n_z\sin(\theta) & n_x n_z(1-\cos(\theta)) + n_y\sin(\theta) \\ n_x n_y(1-\cos(\theta)) + n_z\sin(\theta) & \cos(\theta) + n_y^2(1-\cos(\theta)) & n_y n_z(1-\cos(\theta)) - n_x\sin(\theta) \\ n_x n_z(1-\cos(\theta)) - n_y\sin(\theta) & n_y n_z(1-\cos(\theta)) + n_x\sin(\theta) & \cos(\theta) + n_z^2(1-\cos(\theta)) \end{pmatrix}$$

Pro

- Représentation concise et générale
- Paramètres expressifs dans l'espace 3D (axe, angles)
- Rotation efficace et correspondance avec représentation matricielle

Cons

- Pas de composition simple entre deux rotations
- Pas d'interpolation directe entre deux rotations

(géré par la représentation en quaternion)

Rotation: Focus sur Quaternions

Quaternions: generalisation des nombres complexes.

$q = x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k} + w$ w real part, (x, y, z) imaginary (or pure quaternion) part.

Représentation courte $q = (x, y, z, w)$

(ne pas confondre avec les vecteurs 4D en coordonnées homogènes)

Propriétés algébriques des vecteurs de bases imaginaires

$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$
$\mathbf{ij} = -\mathbf{ji} = \mathbf{k}$
$\mathbf{jk} = -\mathbf{kj} = \mathbf{i}$
$\mathbf{ki} = -\mathbf{ik} = \mathbf{j}$
$\mathbf{ijk} = -1$

Opérations basiques sur les quaternions

- Conjugé $q^\star = (-x, -y, -z, w)$

- Norme $\|q\| = \sqrt{q\, q^\star} = \sqrt{x^2 + y^2 + z^2 + w^2}$. Unit quaternion satisfies $\|q\| = 1$.

- Produit de quaternions

$$q_1\ q_2 = (x_1\,\mathbf{i} + y_1\,\mathbf{j} + z_1\,\mathbf{k} + w_1)\ (x_2\,\mathbf{i} + y_2\,\mathbf{j} + z_2\,\mathbf{k} + w_2) = \ \ldots$$

$$q_1\ q_2 = \begin{pmatrix} x_1 w_2 + w_1 x_2 + y_1 z_2 - z_1 y_2 \\ y_1 w_2 + w_1 y_2 + z_1 x_2 - x_1 z_2 \\ z_1 w_2 + w_1 z_2 + x_1 y_2 - y_1 x_2 \\ w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \end{pmatrix}$$

Note: Produit de quaternion est

- associatif: $(q_1 q_2) q_3 = q_1 (q_2 q_3) = q_1\ q_2\ q_3$
- **non-commutatif**: $q_1\ q_2 \neq q_2\ q_1$

Similaire au produit matriciel

Parfois intéressant de séparer la *partie réelle* w de la *partie quaternion pure* $s = (x, y, z)$.

- Forme vectorielle abrégée $q = (s, w)$
- Produit quaternion en forme vectorielle $q_1\ q_2 = (s_1\, w_2 + s_2\, w_1 + s_1 \times s_2\ ,\ w_1\, w_2 - s_1 \cdot s_2)$

Relation entre les quaternions et les rotations

On considère

- un quaternion $q = (s, w)$ de norme unitaire $\|q\| = 1$.
- un vecteur $v = (v_x, v_y, v_z)$ assimilé à un quaternion pure $q_v = (v, 0) = (v_x, v_y, v_z, 0)$.

On a la propriété suivante

- $q_{v'} = \mathcal{R}_q(v) = q\, q_v\, q^\star$ est un quaternion pure $q_{v'} = (v'_x, v'_y, v'_z, 0)$
- Et $v' = (v'_x, v'_y, v'_z)$ est la rotation du vecteur v autour de l'axe $n = s/\|s\|$, avec l'angle $2\ acos(w)$.

Demonstration

$\mathcal{R}_q(v) = (s, w)\ (v, 0)\ (-s, w) = \ \ldots\ = ((w^2 - s^2)\, v + 2(s \cdot v)\, s + 2w\, (s \times v), 0)$

Comme $\|q\| = 1$, on peut écrire $q = (s, w) = (n \sin(\phi), \cos(\phi))$, where $\|n\| = 1$

Alors $\mathcal{R}_q(v) = (\underbrace{(\cos^2(\phi) - \sin^2(\phi))}_{\cos(2\phi)}\, v + \underbrace{2 \sin^2(\phi)}_{1-\cos(2\phi)}(n \cdot v)\, n + \underbrace{2\ \cos(\phi) \sin(\phi)}_{\sin(2\phi)}\, n \times v\ ,\ 0)$

$\Rightarrow$ formule de Rodrigues pour l'axe n et l'angle 2ϕ.

Le quaternion unitaire $q = (n\ \sin(\theta/2), \cos(\theta/2))$ représente la rotation d'angle θ autour de l'axe n.

Composition de rotations

Soit deux rotations (R_1, R_2) associées à leur quaternions unitaire (q_1, q_2).

Le produit $q_1\, q_2$ représente la composition $R_1 \circ R_2$.

Demonstration

On montre que $\mathcal{R}_{q_1\, q_2}(v) = \mathcal{R}_{q_1} \circ \mathcal{R}_{q_2}(v)$.

$\mathcal{R}_{q_1\, q_2}(v) = (q_1 q_2)\; v\; (q_1 q_2)^\star$

$\mathcal{R}_{q_1\, q_2}(v) = (q_1\, q_2)\; v\; (q_2^\star\, q_1^\star)$, as $(q_1 q_2)^\star = q_2^\star\, q_1^\star$

$\mathcal{R}_{q_1\, q_2}(v) = q_1\, (q_2\; v\; q_2^\star)\, q_1^\star$

$\mathcal{R}_{q_1\, q_2}(v) = q_1\, \mathcal{R}_{q_2}(v)\, q_1^\star = \mathcal{R}_{q_1} \circ \mathcal{R}_{q_2}(v)$

Correspondence quaternion et matrices de rotation

Le quaternion unitaire $q = (x, y, z, w)$ représente la matrice de rotation

$$R = \begin{pmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{pmatrix}$$

Demonstration

$$v' = \mathcal{R}_q(v) = q\, q_v\, q^\star = ((w^2 - s^2)\, v + 2(s \cdot v)\, s + 2w\, (s \times v), 0) \quad \text{with } s = (x, y, z)$$

$$v' = (w^2 - x^2 - y^2 - z^2)v + 2 \begin{pmatrix} x & y & z \end{pmatrix} v \begin{pmatrix} x & y & z \end{pmatrix}^T + 2w \begin{pmatrix} x & y & z \end{pmatrix} \times v$$

$$v' = \left((w^2 - x^2 - y^2 - z^2)\, \mathrm{I} + 2 \begin{pmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{pmatrix} + 2w \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \right) v$$

$v' = R\, v$, avec $x^2 + y^2 + z^2 + w^2 = 1$

Résumé - Correspondence quaternion / matrice-vector

Représentation et opérations

	3D space	Quaternion space
Vector	$v = (v_x, v_y, v_z)$	$q_v = (v, 0) = (v_x, v_y, v_z, 0)$
Rotation	R (3×3 matrix)	$q = (x, y, z, w)$, $\|q\| = 1$
Apply rotation to vector	Rv	$q\ q_v\ q^\star$
Rotation composition	$R_1\ R_2$	$q_1\ q_2$

Rotation vers quaternion

Rotation d'axe n et d'angle $\theta \Rightarrow q = (n \sin(\theta/2), \cos(\theta/2))$

Quaternion vers rotation

Quaternion unitaire $q = (x, y, z, w) \Rightarrow R = \begin{pmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{pmatrix}$

Warning: Ces correspondances ne sont valides que pour des quaternions unitaires.

Résumé pour les rotations

Rotations en 3D ont 3 degrés de libertés, Pas de représentation unique.

Matrice

$$R = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix}$$

$R^T\, R = \mathrm{I}$
$\det(R) = 1$

(+) Simplicité de calcul
(-) DDL non explicites, redondances

Angles d'Euler

3 angles: (α, β, γ)

Composition de rotation autour d'axes de bases

Convention multiples (x-y-z, y-z-x, x-y-x', x-z-x', ...)

(+) Paramètres compréhensibles
(-) Gimbal-lock

Axe angle

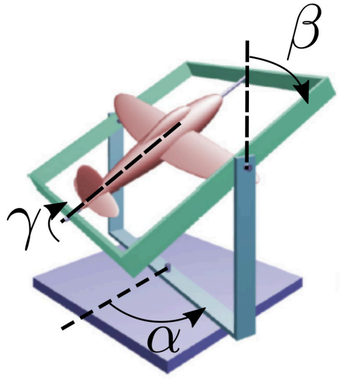
$(\mathbf{n}, \theta)$

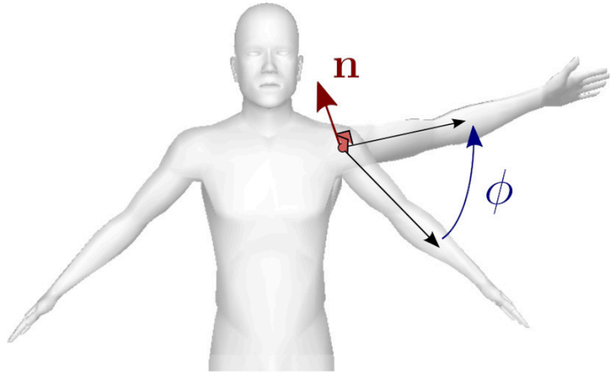
(+) Paramètres compréhensibles
(-) Pas de composition directe

Quaternion

$q = (x, y, z, w)$
$= \left(\mathbf{n} \sin\left(\frac{\theta}{2}\right), \cos\left(\frac{\theta}{2}\right)\right)$

(+) Composition et interpolation
(-) Composants moins intuitifs





Interpolation

Rotations

Interpolation de rotations

Probleme: Soit 2 rotations r_1, r_2

- Trouver la rotation $r(t)$ st $r(0) = r_1$, $r(1) = r_2$, qui varie de manière continue avec t

Représentation matricielle

Interpolation composante par composante inadaptée

ex. $t\,R_1 + (1-t)\,R_2$ n'est pas une rotation: introduction de scaling/shearing

La formulation correcte pour interpoler sur la variété serait $R1\,exp(t\log(R_1^T\,R_2))$

Calcul numérique complexe d'une exponentielle de matrice

Angle d'Euler

Peut être interpolé par interpolation linéaire sur les 3 angles

(+) Conduit à une rotation

(-) Ne suit pas nécessairement la trajectoire la plus simple

Axe-Angle

Interpolation non triviale avec axes différents

Quaternion

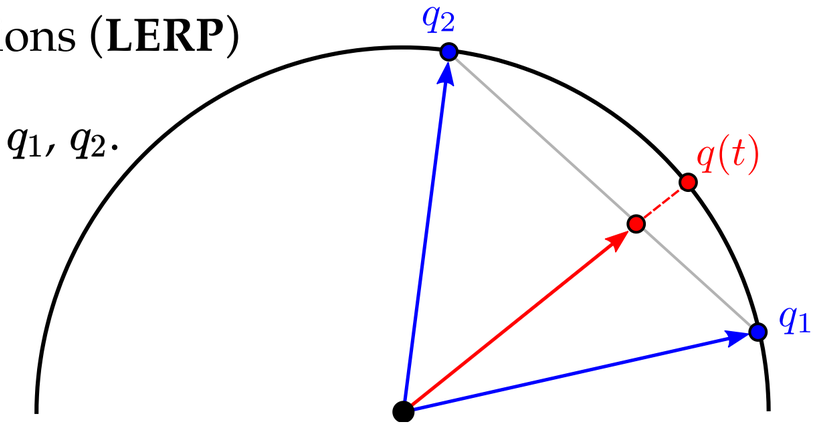
Bien adapté

Interpolation de Quaternions - LERP

Interpolation linéaire (avec normalisation) dans l'espace des quaternions (**LERP**)

- Considérons deux rotations avec les quaternions correspondants q_1, q_2.
- Le quaternion *interpolé linéairement* au paramètre $t \in [0, 1]$ est

$$q(t) = \frac{(1-t)\,q_1 + t\,q_2}{\|(1-t)\,q_1 + t\,q_2\|}$$



(+) Suit un chemin le plus court (grand cercle sur la sphère 4D)

(+) Peut être généralisé à des courbes paramétriques plus générales (Splines, etc.)

(-) La vitesse angulaire de l'orientation n'est pas constante

ex. cas extrême de deux quaternions opposés

Interpolation de Quaternions - SLERP

Spherical Linear Interpolation (**SLERP**)

- Considérons deux vecteurs unitaires (dimensions arbitraires) v_1, v_2.
- L'*interpolation SLERP* au paramètre $t \in [0, 1]$ est

$$v(t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)} v_1 + \frac{\sin(t\Omega)}{\sin(\Omega)} v_2 \quad \text{avec } \cos(\Omega) = v_1 \cdot v_2$$

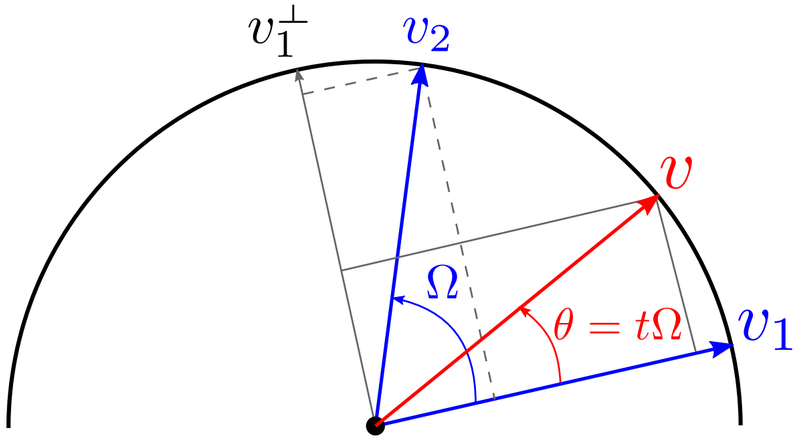
Démonstration

Considérons
- Deux vecteurs unitaires (dimensions arbitraires) v_1, v_2.
- Ω: angle entre v_1 et v_2
- Le vecteur interpolé v à l'angle $\theta = \Omega t$, $t \in [0, 1]$.

$v = v_1 \cos(\theta) + v_1^\perp \sin(\theta)$, et $v_1^\perp = \frac{v_2 - \cos(\Omega)\, v_1}{\sin(\Omega)}$

$\Rightarrow v = \left(\frac{\sin(\Omega)\, \cos(\theta) - \cos(\Omega)\, \sin(\theta)}{\sin(\Omega)} \right) v_1 + \frac{\sin(\theta)}{\sin(\Omega)} v_2$

$\Rightarrow v = \frac{\sin(\Omega - \theta)}{\sin(\Omega)} v_1 + \frac{\sin(\theta)}{\sin(\Omega)} v_2$

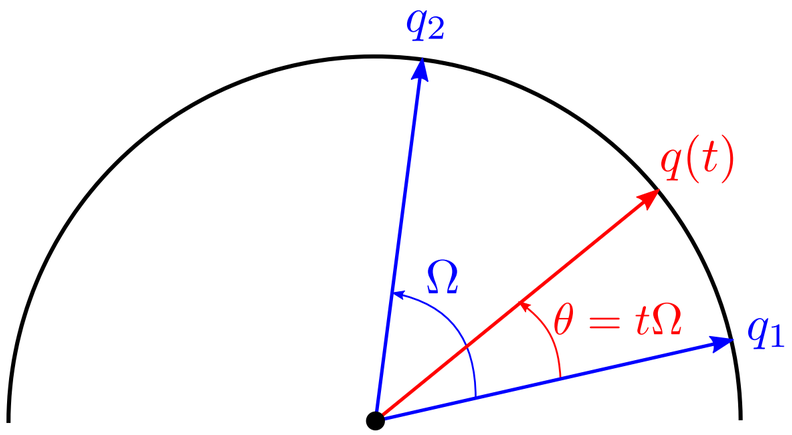


Interpolation de Quaternions - SLERP

Spherical Linear Interpolation (**SLERP**)

- Considérons deux vecteurs unitaires (dimensions arbitraires) v_1, v_2.
- L'*interpolation SLERP* au paramètre $t \in [0, 1]$ est

$$q(t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)} q_1 + \frac{\sin(t\Omega)}{\sin(\Omega)} q_2 \quad \text{avec } \cos(\Omega) = q_1 \cdot q_2$$



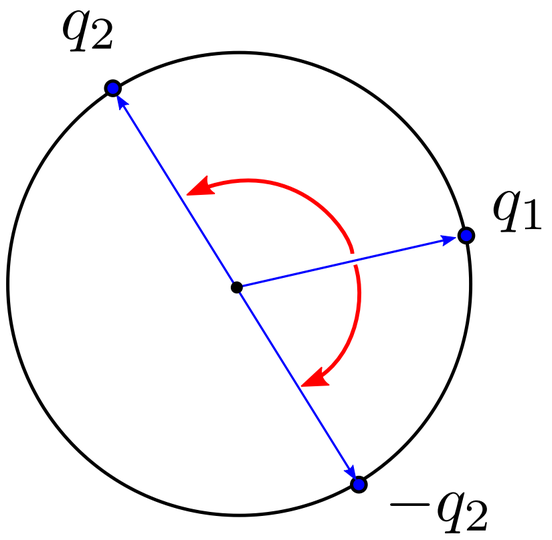
(+) Suit un chemin le plus court (grand cercle sur une sphère 4D)

(+) Vitesse angulaire constante

(-) Ne peut pas être généralisé à des courbes plus générales.

Ne peut pas interpoler entre plus de deux quaternions

Attention à l'opposé des quaternions



$+q$ et $-q$ correspondent à la même rotation ($n \to -n$, $\theta \to 2\pi - \theta$)

Attention $-q$ ***ne correspond pas*** *à la matrice de rotation* $-R$.

Mais à un **chemin différent** lorsqu'il est interpolé dans l'espace 4D des quaternions.

$\Rightarrow$ le chemin $q_1 \to -q_2$ est plus court que $q_1 \to q_2$ lorsque $q_1 \cdot q_2 < 0$.

En pratique, nous vérifions le chemin le plus court avant d'appliquer SLERI

Algorithme

```
if( dot(q1,q2)<0 )
    q2 = -q2

q(t) = SLERP(q1,q2,t)
```

Interpolation

Transformations affines

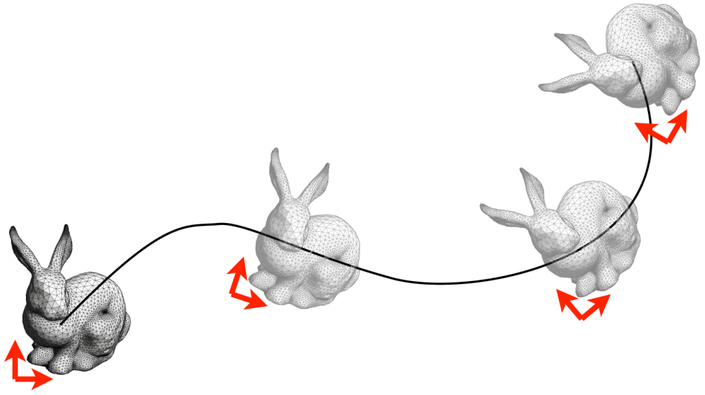
Interpolation de mouvement rigide

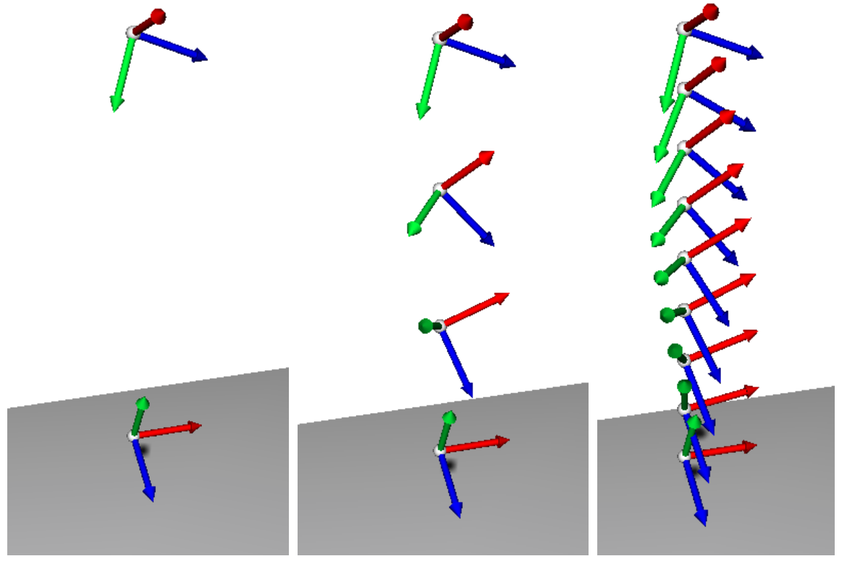
Les objets 3D ont une orientation r et une position p.

$\Rightarrow$ Besoin d'interpoler les deux, généralement traité séparément.

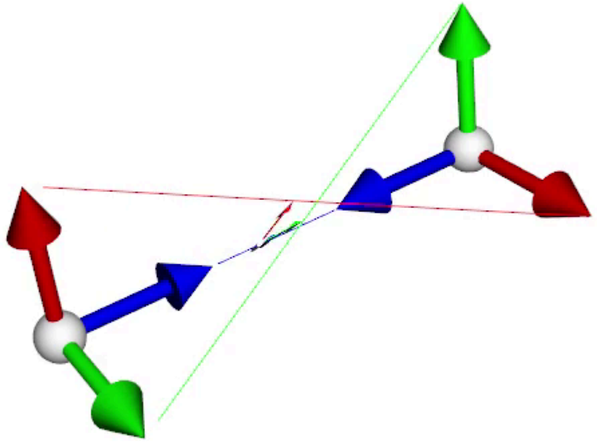
Étant donné deux positions clés (p_1, r_1) et (p_2, r_2), les *positions intermédiaires* sont calculées au temps $t \in [0, 1]$ comme suit :

- Interpolation linéaire des positions $p(t) = (1 - t)\, p_1 + t\, p_2$
- Interpolation de la rotation avec SLERP sur les quaternions
  - Convertir $(r_1, r_2) \rightarrow (q_1, q_2)$
  - Calculer $q(t) = SLERP(q_1, q_2, t)$
  - Convertir $q(t) \rightarrow r(t)$

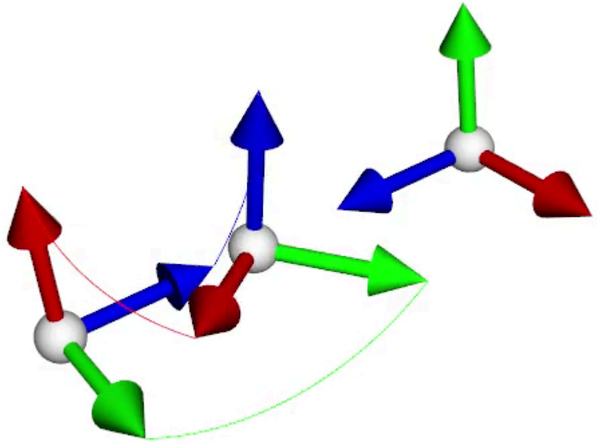




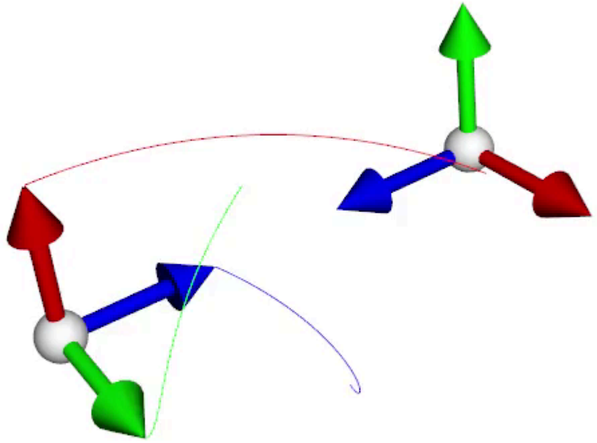
Interpolation de transformations rigides - Comparison



Matrix interpolation



Euler angle interpolation



Quaternion interpolation

Transformations affines : Décomposition polaire

Les transformations clés souvent données en matrices 4×4.
Comment interpoler entre des transformations affines ?

- Séparer la partie linéaire M et la translation est facile.
- *Problème* : interpoler la partie linéaire M
  mélange rotation, mise à l'échelle, cisaillement

$\Rightarrow$ Séparer M en : partie rotation, et mise à l'échelle/cisaillement.

- Interpoler la rotation avec quaternion (ex. SLERP/LERP)
- Mise à l'échelle/cisaillement en utilisant une interpolation composante par composante (ex. linéaire).

- Décomposition polaire : $\mathrm{M} = \mathrm{R}\,\mathrm{D}$
  - R : Matrice de rotation
  - D : Matrice semi-définie positive

[K. Shoemake et T. Duff, Animation de matrices et décomposition polaire. Graphics Interface 92.]

- La décomposition polaire est obtenue à partir de la SVD

$\mathrm{SVD}(\mathrm{M}) = W \Sigma V^T$ avec $\boxed{\mathrm{R} = \mathrm{W}\mathrm{V}^T}$, $\mathrm{D} = \mathrm{V}\Sigma\mathrm{V}^T$

- Ou numériquement, R peut être calculée en utilisant le schéma itératif suivant

$$\boxed{R_0 = \mathrm{M}\,,\ R_{i+1} = 0.5\,(R_i + (R_i^{-1})^T)}$$

Gestion des transformations affines

Algorithme pour interpoler entre deux matrices générales 4×4 M_1, M_2

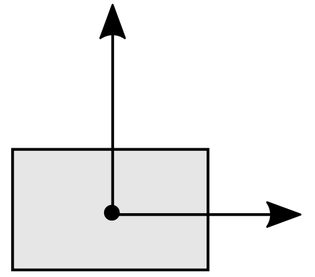
1. Extraire la translation p_1, p_2 de M_1, M_2.
2. Calculer R_1, R_2 (matrices de rotation 3×3) et D_1, D_2 (matrices de mise à l'échelle / déformation 3×3) à partir de M_1, M_2.
3. Interpoler linéairement la position et la mise à l'échelle / déformation $p(t) = (1-t)\,p_1 + t\,p_2$, $D(t) = (1-t)\,D_1 + t\,D_2$.
4. Calculer les quaternions q_1, q_2 à partir de R_1, R_2.

   Remarque $M \to q$ avec $q = \left(\frac{M_{zy}-M_{yz}}{2r}, \frac{M_{xz}-M_{zx}}{2r}, \frac{M_{yx}-M_{xy}}{2r}, \frac{r}{2}\right)$, $r = \sqrt{1 + M_{xx} + M_{yy} + M_{zz}}$.
5. Calculer $q(t)$ =SLERP(q_1, q_2, t).
6. Reconvertir en matrice $q(t) \to R(t)$.
7. Calculer la matrice finale $M(t) = R(t)\,D(t)$ avec la translation $p(t)$.

Précautions avec l'ordre des transformations

Attention, l'ordre des opérations est important !

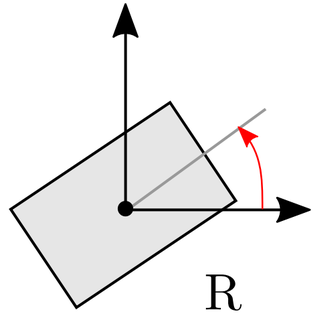
Rotation r, Translation t: $r \circ t \neq t \circ r \;\Rightarrow\; \mathrm{M}_1 = \mathrm{T}\,\mathrm{R} \neq \mathrm{R}\,\mathrm{T} = \mathrm{M}_2$

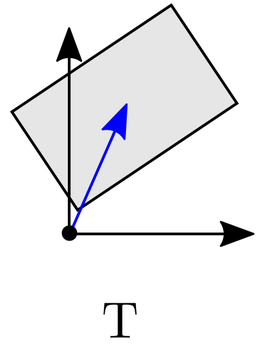


Attention (2) : les matrices de transformation sont appliquées aux coordonnées de droite à gauche.

$$\mathrm{M}_1 = \mathrm{TR} = \left(\begin{array}{c|c} 1 & t \\ \hline 0 & 1 \end{array}\right) \left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right)$$

D'abord rotation, puis translation

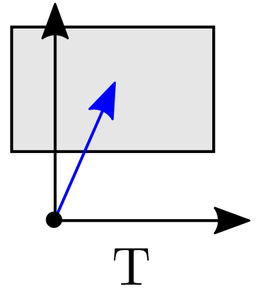


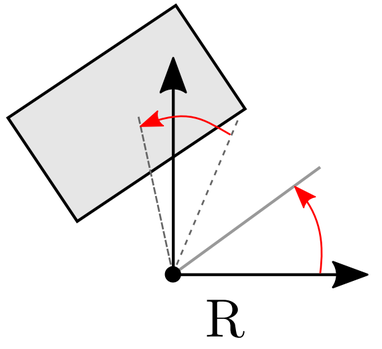


$$\mathrm{M}_2 = \mathrm{RT} = \left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right) \left(\begin{array}{c|c} 1 & t \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} R & \mathrm{R}\,t \\ \hline 0 & 1 \end{array}\right)$$

D'abord translation, puis rotation

La rotation se fait toujours autour de l'origine.



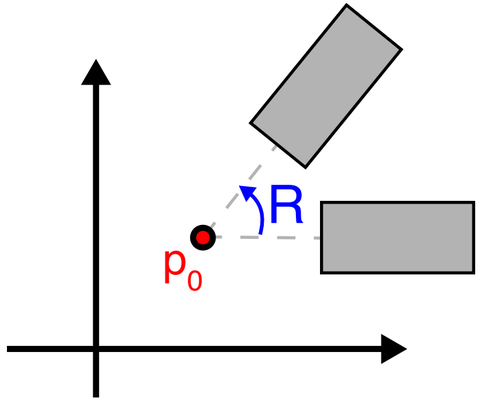


Attention : Certaines bibliothèques (ancien OpenGL, Three.js) appliquent les transformations de "gauche à droite" en utilisant une multiplication matricielle transposée

Exercices sur les Transformations Affines

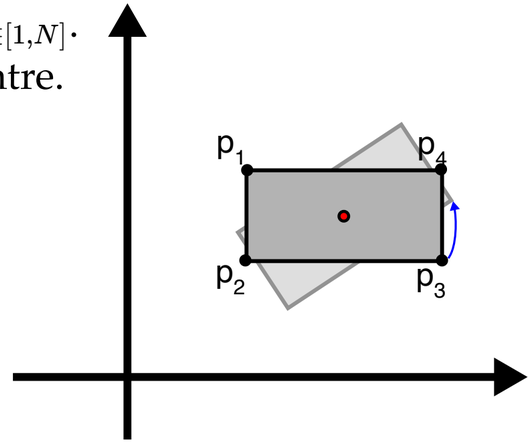
Q. Exprimez la transformation affine (sous forme d'une matrice bloc 4×4) correspondant à une rotation R appliquée autour d'une position arbitraire p_0 dans l'espace.

> $M = ...$



Q. Considérez une forme définie par un maillage triangulaire avec des positions de sommets $(p_i)_{i \in [1,N]}$.

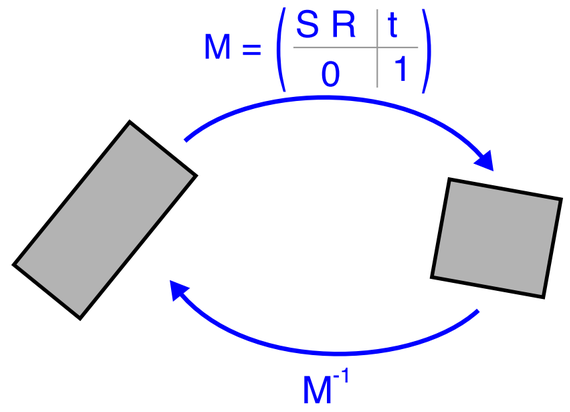
> Exprimez la transformation affine permettant de faire tourner la forme autour de son barycentre.



Transformations affines: exercices

Q. Considérez la transformation affine M paramétrée par un facteur d'échelle s, une rotation R, et une translation t.

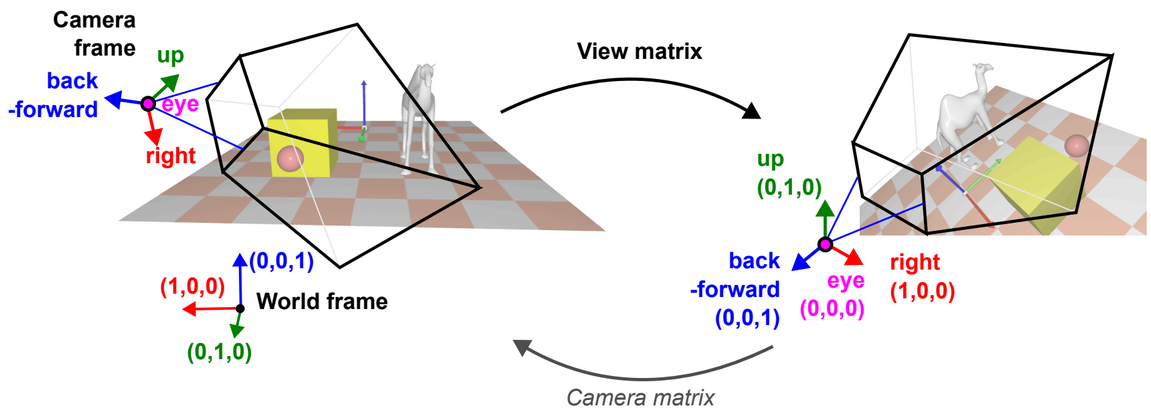
> Exprimez la matrice inverse M^{-1} en fonction de s, R et t



View matrix - Rappel

Matrice **View**:

Transforme *World space coords.* $\rightarrow$ *View space coords.*



Coordonnées de la caméra: Cam

$\text{Cam} \times (1,0,0,0) = right$

$\text{Cam} \times (0,1,0,0) = up$

$\text{Cam} \times (0,0,1,0) = back$

$\text{Cam} \times (0,0,0,1) = eye$

$$\text{Cam} = \begin{pmatrix} | & | & | & | \\ right & up & back & eye \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} O & eye \\ 0 & 1 \end{pmatrix}$$

View matrix: View

$\text{View} \times right = (1,0,0,0)$

$\text{View} \times up = (0,1,0,0)$

$\text{View} \times back = (0,0,1,0)$

$\text{View} \times eye = (0,0,0,1)$

$$\text{View} = \begin{pmatrix} (\cdots & right & \cdots) & -right \cdot eye \\ (\ldots & up & \cdots) & -up \cdot eye \\ (\cdots & -front & \cdots) & -back \cdot eye \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} O^T & -O^T\, eye \\ 0 & 1 \end{pmatrix}$$

$$\boxed{\text{View} = \text{Cam}^{-1}}$$

Interaction avec la caméra

L'orientation de la caméra a 3 degrés de liberté (+ 3 dof en position)

Comment gérer l'orientation d'une caméra avec une souris (2-dof) ?

En supposant une caméra librement orientée

Approches courantes

"Caméra sphérique"/coordonnées (θ, ϕ)

(+) Pratique pour "tourner autour" d'une structure orientée

(-) Manque un dof : Pas de "rotation" : la position sur la sphère unité impose l'*angle*

Pas adapté pour tourner autour d'objets non orientés/mal orientés.

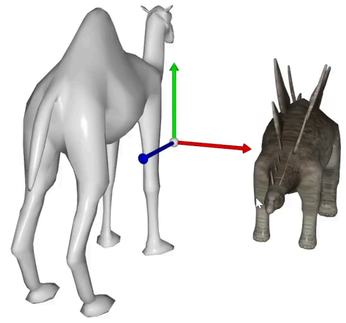
Coordonnées sphériques + rotation (ex. roulis, tangage, lacet)

(+) Contrôle précis et complet

(-) Nécessite une touche/un contrôleur supplémentaire pour 3 dof

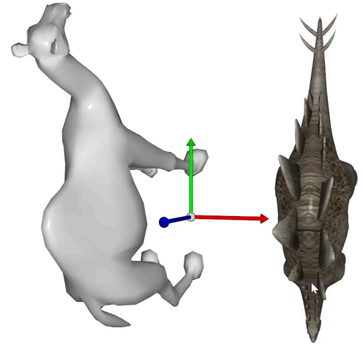
Exemple de "Caméra sphérique"

Orientation correcte



Mauvaise orientation de la forme

caméra sphérique "bloquée" en orientation



Interaction avec la caméra - Métaphore ArcBall/Trackball

Autre approche : **Comportement ArcBall/Trackball**

3 degrés de liberté : gauche/droite, haut/bas, mais aussi rotation

Idée : Utiliser la position du curseur 2D comme "point 3D sur le trackball"

Mouvement entre 2 positions du curseur = Rotation 3D appliquée à la sphère

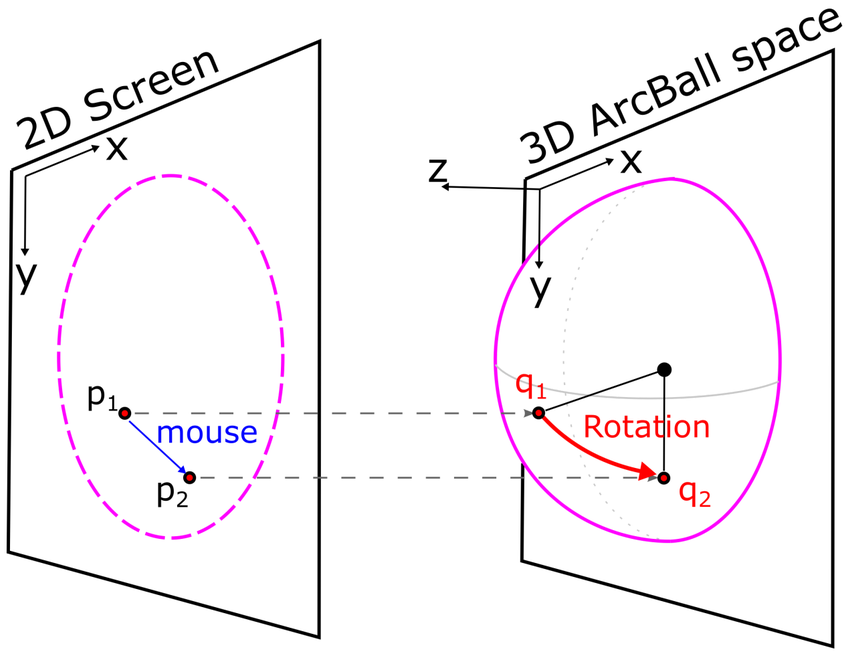
[K. Shoemake. Une interface utilisateur pour spécifier l'orientation tridimensionnelle à l'aide d'une souris. Graphics Interface, 1992.]

Algorithme :

Input $p_1 = (p_{1x}, p_{1y})$, $p_2 = (p_{2x}, p_{2y})$ in screen space.
$q_{1/2}$ = ProjectionArcBall($p_{1/2}$)
R= rotation between vectors (q_1, q_2)

ProjectionArcBall(p)
d =norm(p)
If(d<$1/\sqrt{2}$)
  $q = (p, \sqrt{1-d^2})$
Else / / hyperbole
  $q = (p, 1/(2d))$
return q

Note : l'hyperbole étend la sphère sur tout l'écran.



Interaction avec la caméra - Métaphore ArcBall/Trackball

Manière naturelle de tourner autour d'une forme

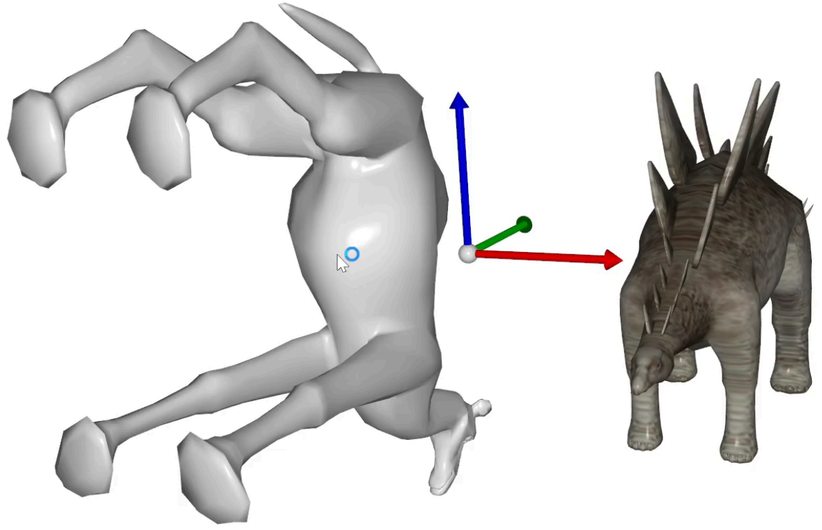
(+) Pas d'axe privilégié

Pas de configuration de référence, fonctionne avec des rotations incrémentales

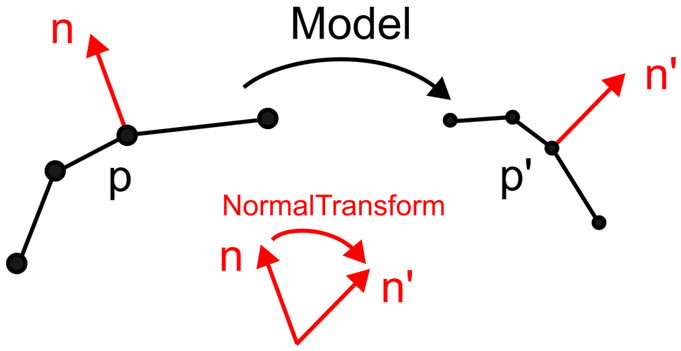
Le comportement est invariant à l'orientation

(-) Moins précis que le contrôle individuel des ddl

Dérive de twist lors des mouvements aller-retour



Normal Transform



Model transform $\Rightarrow$ matrice 4×4 de transformation sur un objet donné.

$p' = \mathrm{M}\, p$

Comment déformer les normales ? $n' = \mathrm{N}\, n$

Trivial sur translation, rotation, et scaling. Comment gérer le cas générique?

Soit t un vecteur tangent à la surface de l'objet en p.

- On a $n \cdot t = 0$
- Après transformation, on a $t' = \mathrm{M}\, t$, et $n' \cdot t' = 0$

$n' \cdot t' = 0$

$\Rightarrow (\mathrm{N}\, n) \cdot (\mathrm{M}\, t) = 0$

$\Rightarrow (\mathrm{N}\, n)^T\ (\mathrm{M}\, t) = 0$

$\Rightarrow n^T\ (\mathrm{N}^T \mathrm{M})\ t = 0\ (1)$

Comme $n^T\, t = 0$ par définition, la condition (1) est satisfaite si

$\mathrm{N}^T \mathrm{M} = \mathrm{Id}$

$\Rightarrow \mathrm{N} = (M^{-1})^T$

$$\mathrm{NormalTransform} = (\mathrm{inverse}(\mathrm{Model}))^T$$

```
mat4 normalTransform = transpose(inverse(Model));
vec4 normal = normalTransform * vec4(vertex_normal, 0.0);
```