Computación Gráfica

Ing. Gabriel Ávila, MSc.

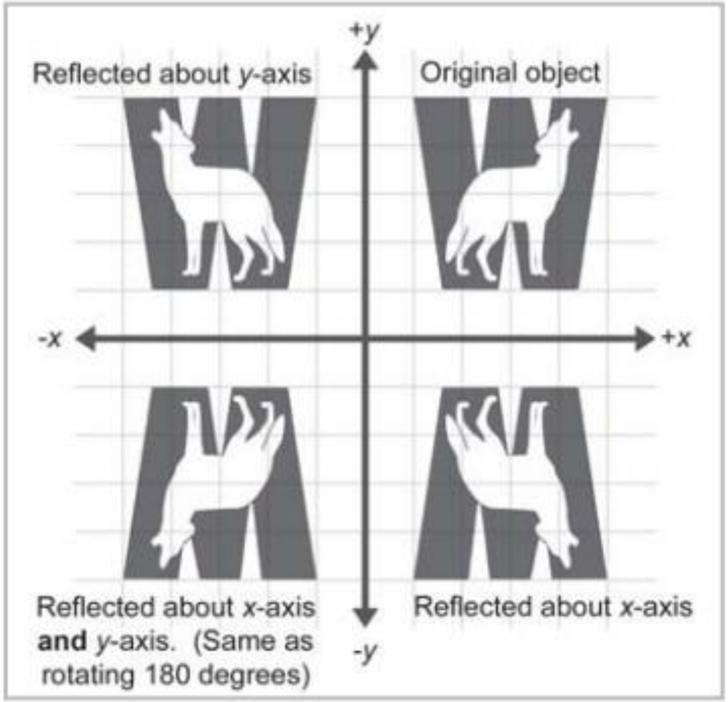
Otras transformaciones

Reflection

También llamada transformación de espejo, refleja el elemento con respecto a una línea (en 2D) o a un plano (en 3D). Dado un vector n:

En 2D, con respecto a un eje, que pasa por el origen y es perpendicular a n:

$$R(n) = \begin{bmatrix} 1 - 2{n_x}^2 & -2n_x n_y \\ -2n_x n_y & 1 - 2{n_y}^2 \end{bmatrix}$$

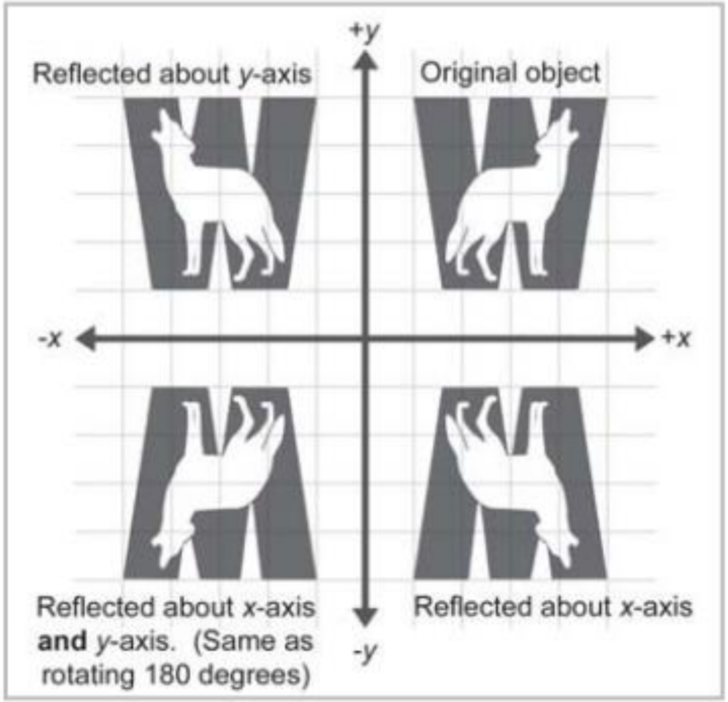


Reflection

También llamada transformación de espejo, refleja el elemento con respecto a una línea (en 2D) o a un plano (en 3D). Dado un vector n:

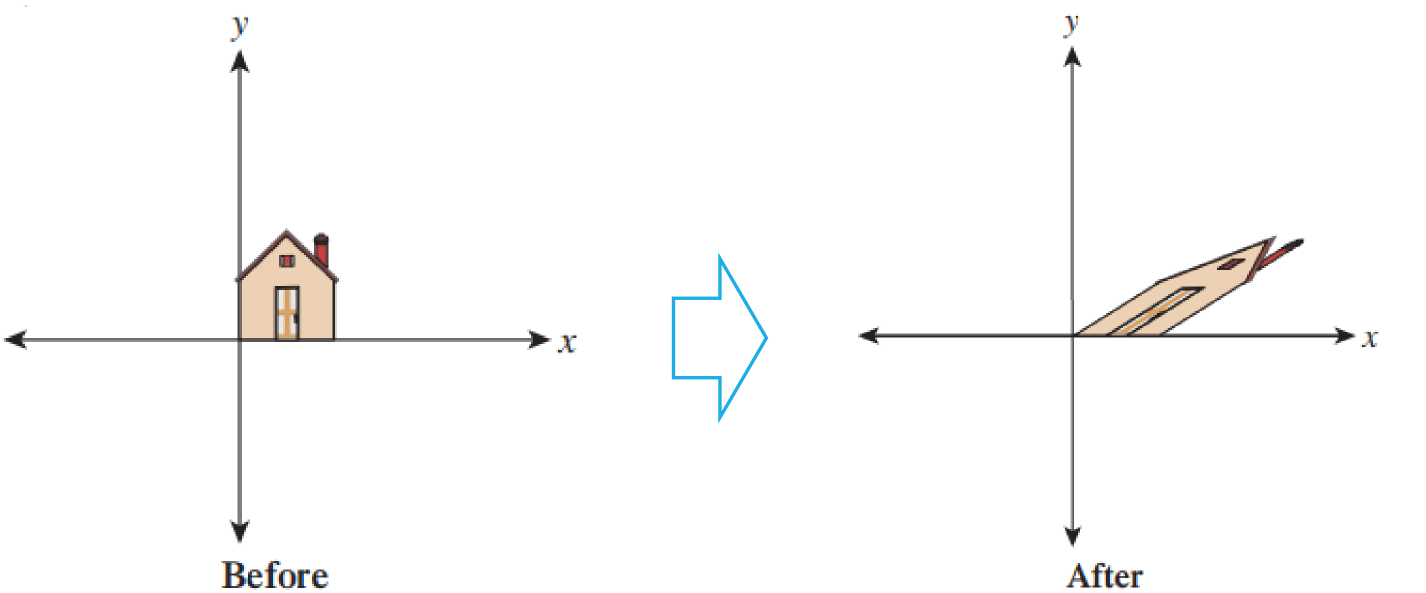
En 3D, con respecto a un plano de reflexión (en el origen y perpendicular a n):

$$R(n) = \begin{bmatrix} 1 - 2{n_x}^2 & -2n_x n_y & -2n_x n_z \\ -2n_x n_y & 1 - 2{n_y}^2 & -2n_y n_z \\ -2n_x n_z & -2n_y n_z & 1 - 2{n_z}^2 \end{bmatrix}$$



Shear - *Skew*
Transformación de corte

Permite deformar el elemento en 3D de manera no uniforme. No se preservan los ángulos, sin embargo, los volúmenes y áreas sí.



Shear - *Skew* Transformación de corte

La idea básica es agregar un múltiplo de una coordenada a la otra. Por ejemplo, en 2D, se podría tomar un múltiplo de y para agregárselo a x, de tal forma que:

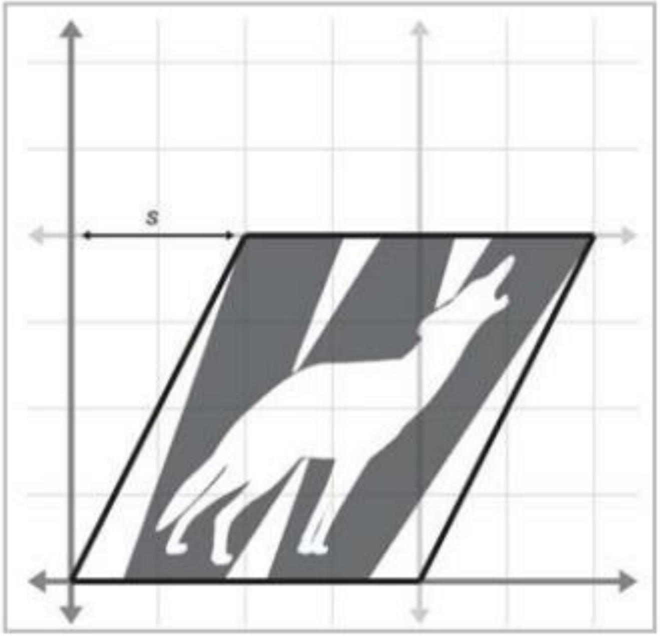
$$x' = x + sy$$

La matriz que permite hacer esto es:

$$H_x(s) = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}$$

En la coordenada y, la matriz sería:

$$H_y(s) = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}$$



Shear - *Skew* Transformación de corte

En 3D, esta deformación puede darse en 6 direcciones posibles:

- Corte de X en Y: ***Sxy***
- Corte de X en Z: ***Sxz***
- Corte de Y en X: ***Syx***
- Corte de Y en Z: ***Syz***
- Corte de Z en X: ***Szx***
- Corte de Z en Y: ***Szy***

$$T_{x,y,z} = \begin{bmatrix} 1 & S_{yx} & S_{zx} & 0 \\ S_{xy} & 1 & S_{zy} & 0 \\ S_{xz} & S_{yz} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotaciones

Rotaciones de Euler

Secuencia de tres rotaciones, una después de la otra, en un orden determinado.

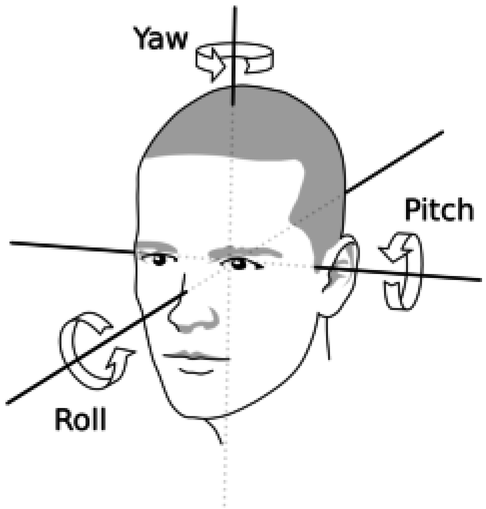
$$R_x \to R_y \to R_z$$

Haciendo las rotaciones de esta forma, es posible rotar un objeto en cualquier eje.

Rotaciones de Euler

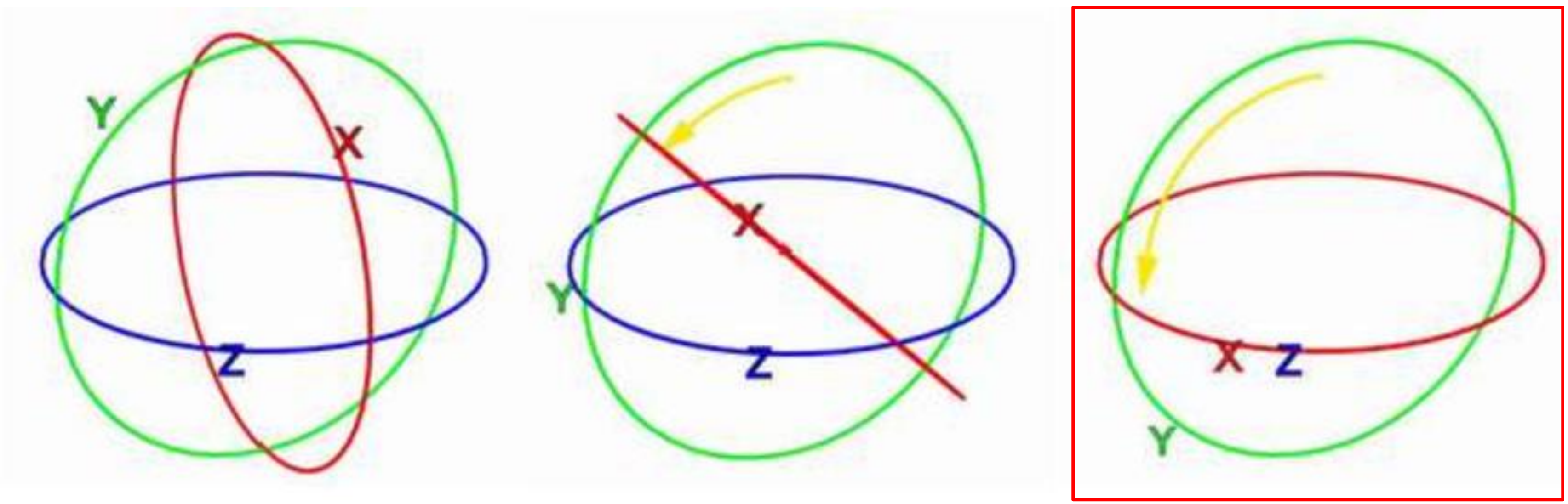
Cada rotación recibe un nombre en particular:

- **Yaw** (*Guiño*):
  Rotar alrededor del eje Y.
- **Pitch** (*Cabeceo*):
  Rotar alrededor del eje X (Ya rotado).
- **Roll** (*Alabeo*):
  Rotar alrededor del eje Z (Ya rotado).



Problema: *Gimbal lock*

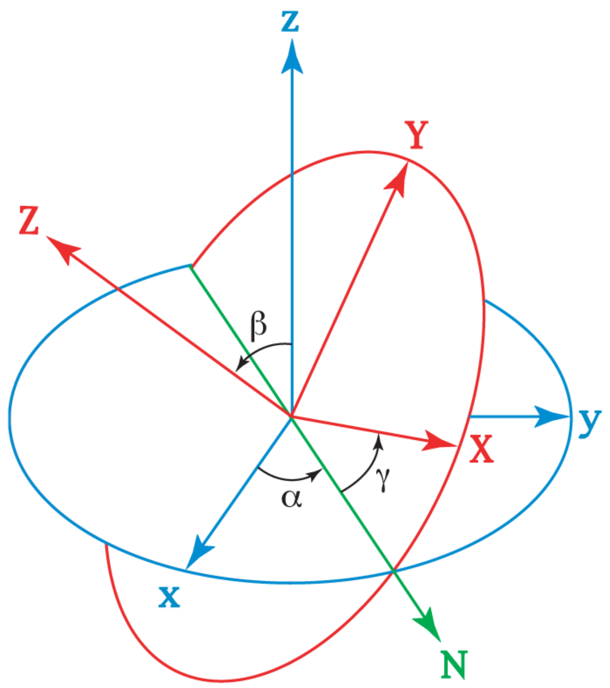
Cuando dos de los ejes de rotación están alineados, se pierde un grado de libertad en la rotación.



Ángulos de Euler

Dados dos sistemas de coordenadas XYZ y xyz, existe una línea que representa la intersección entre dos planos de dichos sistemas (XZ y xz por ejemplo) llamada línea de nodos (N).

En la imagen, los ángulos de Euler son:
α: Ángulo entre el eje x y N.
β: Ángulo entre el eje z y el eje Z.
γ: Ángulo entre N y el eje X.

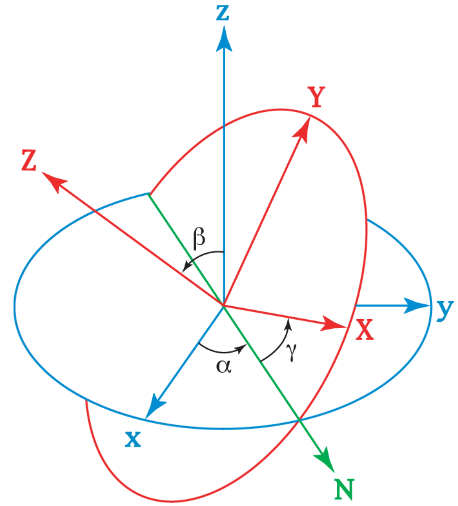


Tomado de: https://es.wikipedia.org/wiki/%C3%81ngulos_de_Euler

Problema: *Ambigüedades*

Las rotaciones de Euler pueden ser de dos tipos, intrínsecas o extrínsecas. Es necesario definir cuál tipo se va a usar, y comprender sus diferencias:

Intrínsecas (con respecto a los ejes del sistema que se está rotando). Dados dos planos base (XY y xy por ejemplo), se debe obtener la línea de nodos (N, línea que representa la intersección entre los planos), y se rota con respecto a dicha línea.



Problema: *Ambigüedades*

Las rotaciones de Euler pueden ser de dos tipos, extrínsecas o intrínsecas. Es necesario definir cuál tipo se va a usar, y comprender sus diferencias:

Extrínsecas (con respecto a un marco de referencia que no se mueve). Dados dos marcos de referencia XYZ (a rotar) y xyz de base, se hace:

- Rotación alrededor del eje z un ángulo α. El sistema xyz no se mueve.

- Rotación alrededor del eje x un ángulo β.

- Rotación alrededor del eje z un ángulo γ.

Problema: *Ambigüedades*

Las rotaciones de Euler pueden ser ***extrínsecas*** (con respecto a un marco de referencia que no se mueve), o ***intrínsecas*** (con respecto a los ejes del sistema que se está rotando).

Existen 12 posibilidades para generar las rotaciones:

Ángulos de Euler (rotación intrínseca): ZXZ, XYX, YZY, ZYZ, XZX, YXY.

Ángulos de Tait-Bryan (rotación extrínseca): XYZ, YZX, ZXY, XZY, ZYX, YXZ.

Ejemplo: Rotación XYZ

1. Rotar alrededor de X un ángulo θ_1
2. Rotar alrededor de Y un ángulo θ_2
3. Rotar alrededor de Z un ángulo θ_3

$$R_{x,y,z} = \underbrace{\begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Rz}} \underbrace{\begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 \\ 0 & 1 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix}}_{\text{Ry}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 \end{bmatrix}}_{\text{Rx}}$$

Cuaterniones

De manera general, se consideran como la mejor opción. ***Extienden el concepto de rotación en 3 dimensiones a rotación en 4 dimensiones.***

Un Cuaternión se define usando cuatro valores | w x y z | los cuales se calculan a partir de la combinación de las 3 coordenadas del eje de rotación y el ángulo de rotación.

Utilizándolos, es posible evitar el problema del Gimbal-lock, generando rotaciones suaves y continuas.

Cuaterniones

Generalización de números complejos usando tres números imaginarios:

$$i, j, k$$

Tal que:

$$i^2 = -1, j^2 = -1, k^2 = -1, ijk = -1$$

$$\begin{matrix} ij = k & jk = i & ki = j \\ ji = -k & kj = -i & ik = -j \end{matrix}$$

Es posible representar un cuaternión como:

$$q = s + xi + yj + zk$$

Cuaterniones

Las rotaciones se representan mediante cuaterniones unitarios:

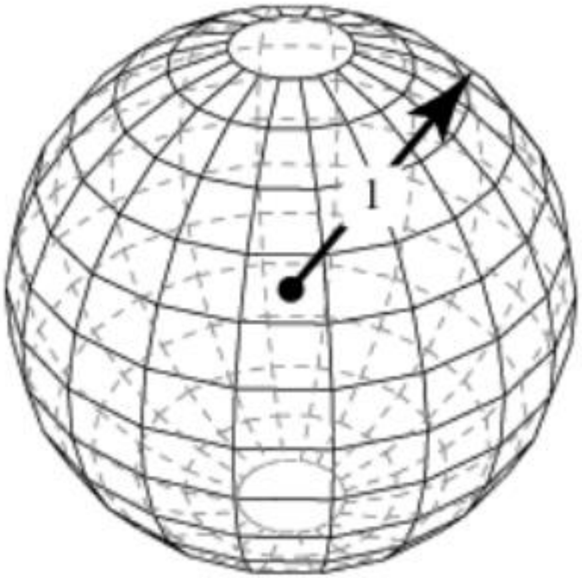
$$q = s + xi + yj + zk$$

Donde:

$$s^2 + x^2 + y^2 + z^2 = 1$$

Esto forma una esfera en 4 dimensiones.

Se conoce como la esfera de cuaternión unitario.



Rotación con cuaterniones

A partir de un vector y un ángulo, es posible generar un cuaternión unitario que permita realizarla rotación.

Algunas librerías matemáticas permiten realizar este procedimiento y el método se ha estandarizado.

```
var quaternion = new THREE.Quaternion();

quaternion.setFromAxisAngle( newTHREE.Vector3( 0, 1, 0 ), Math.PI/2 );

var vector = new THREE.Vector3( 1, 0, 0 );

vector.applyQuaternion( quaternion );
```

Bibliografía

[1] Dunn, F. y Parberry, I. (2002). 3D Math Primer for graphics and game development. Wordware Publishing, Inc.

[2] Tutorial 3: Matrices. Tomado de: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

[3] The Matrix and Quaternions FAQ. (2002). http://www.opengl-tutorial.org/assets/faq_quaternions/index.html

[4] Rueda, A. (2015). Transformaciones en 3D. Universidad Javeriana.