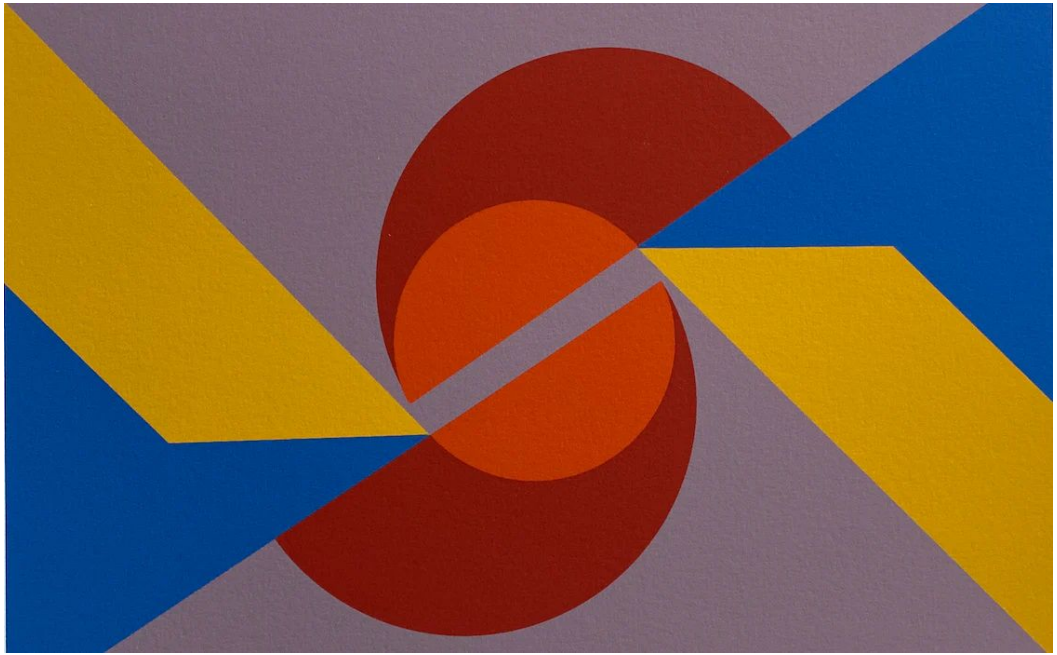
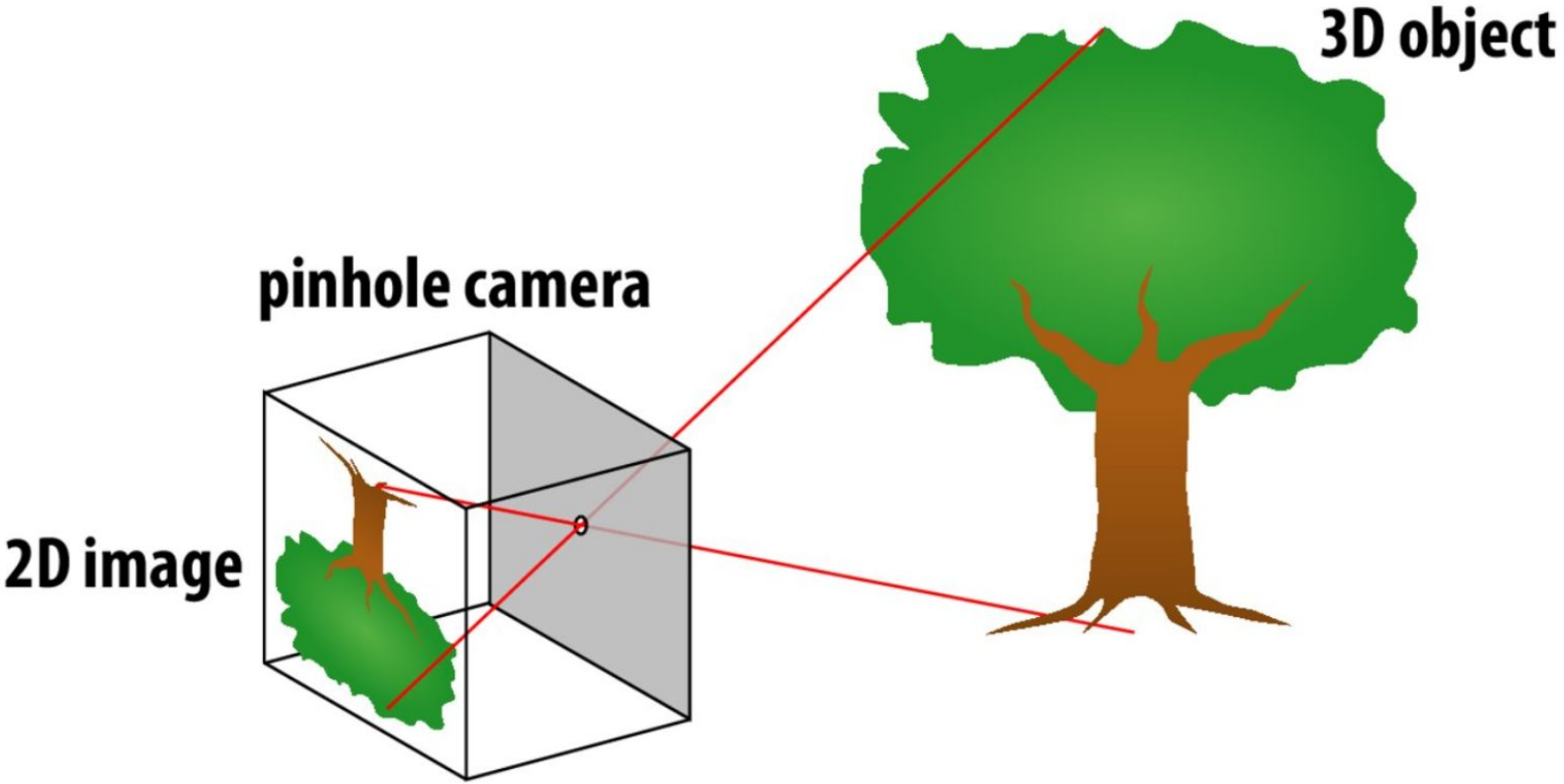


Germinación del núcleo, Elsa Bolívar (2017)

CC3501 Transformaciones (Parte 2)

Eduardo Graells-Garrido

Otoño 2023



Miremos el mundo a través del lente de una cámara puntual. Esta metáfora nos ayudará a entrar al mundo de las **coordenadas homogéneas**, llamado **espacio proyectivo**.

Coordenadas Homogéneas

Surgieron al querer estudiar **perspectivas**

Introducidas por **Moebius** como una manera natural de asignar coordenadas a líneas

Surgen de manera natural en muchos aspectos de computación gráfica:

Transformaciones 3D, proyección en perspectiva, mapas de sombras, geometría discreta conformal, recorte, luces direccionales...

... es un concepto que usaremos de manera recurrente durante el curso.

Espacio Proyectivo

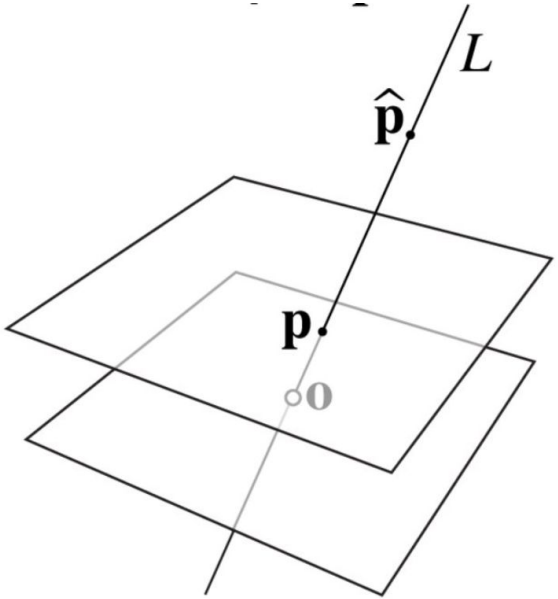
Consideren cualquier plano 2D que no contenga al origen en 3D. Por ejemplo, el **plano de la imagen**.

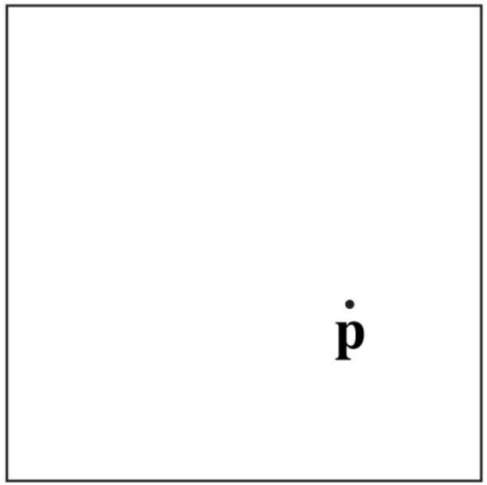
Cualquier línea L que pase a través del origen se intersecta con ese plano.

Para representar a la línea podemos usar el punto p en que intersecta al plano. En este espacio, **cualquier punto que esté en la línea L puede ser usado para representar al punto p**. Así

p = (ax, ay, a) para cualquier valor de a != 0

En el espacio **cartesiano**, p = (x/a, y/a) (para a != 0)





Definición

p = (xi, yi)

El plano de proyección será el mismo de nuestra imagen, y se encontrará en un valor z = f (usualmente f = 1)

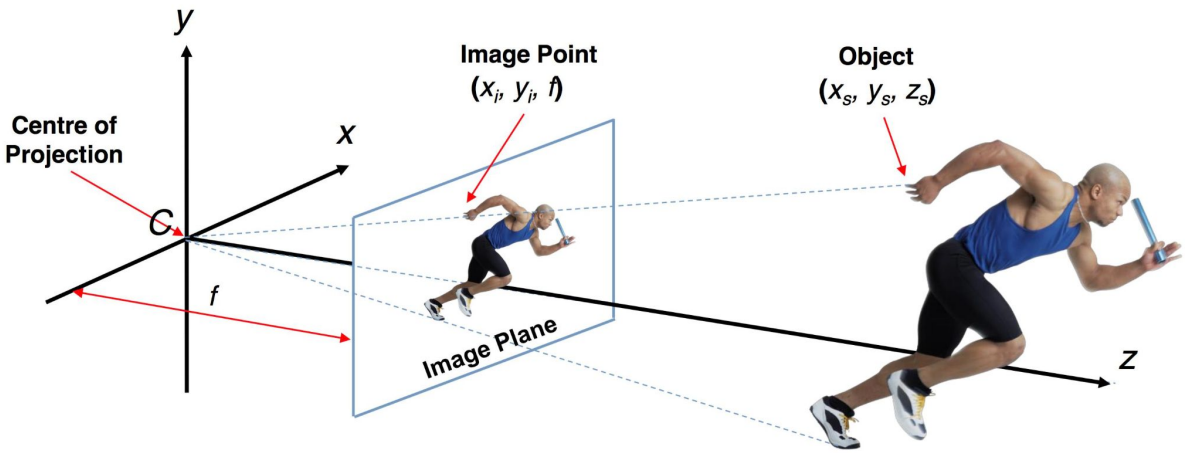
p^ = (xs, ys, zs) tal que (xs/zs, ys/zs) = (xi,yi)

Se dice que **p^** es una coordenada homogénea de **p**

Ejemplo: (x, y, 1)

En general, (cx, cy, c) para c != 0

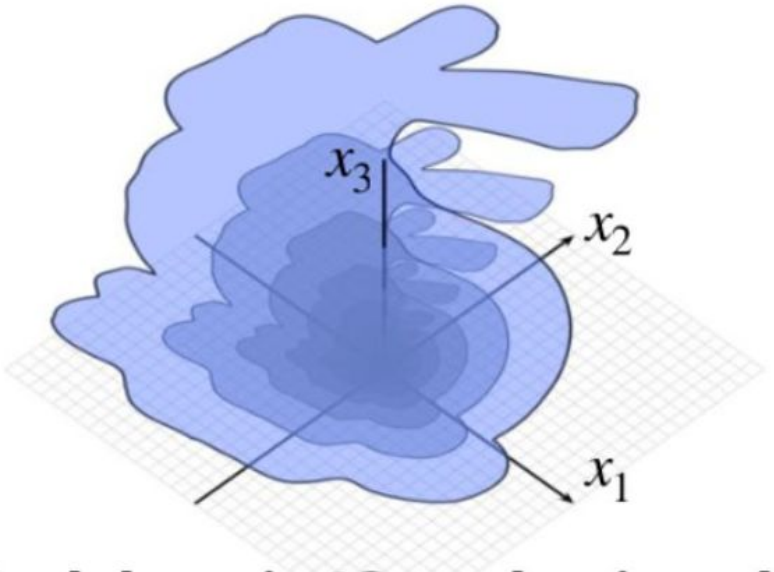
¿Por qué sirve esto para las transformaciones?



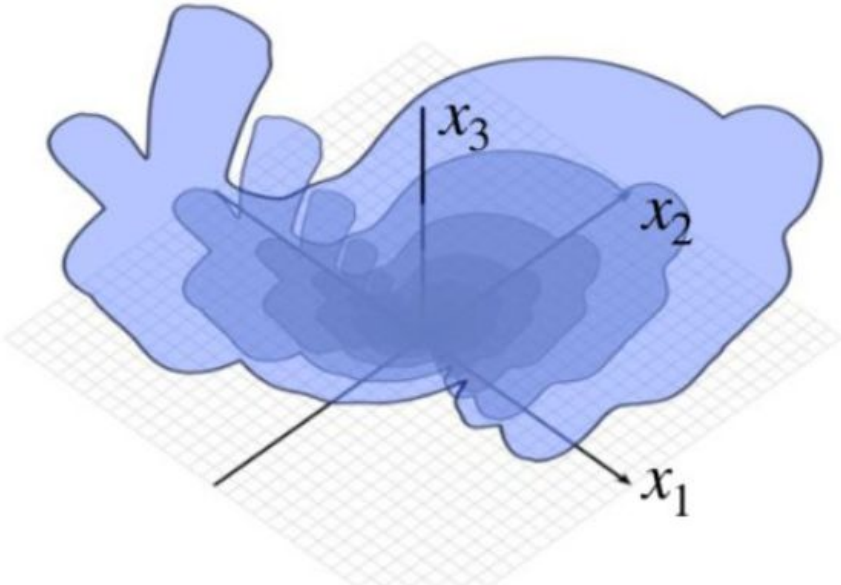
$$x_i = f\frac{x_s}{z_s}, \quad y_i = f\frac{y_s}{z_s}$$

En Coordenadas Homogéneas...

La forma original en 2D se puede interpretar como si tuviese múltiples copias, cada una escalada de manera uniforme por X3 (para distintos valores de c)



La rotación 2D es una rotación 3D alrededor del eje X3



¿Y shearing?

Recordemos la definición de shear:

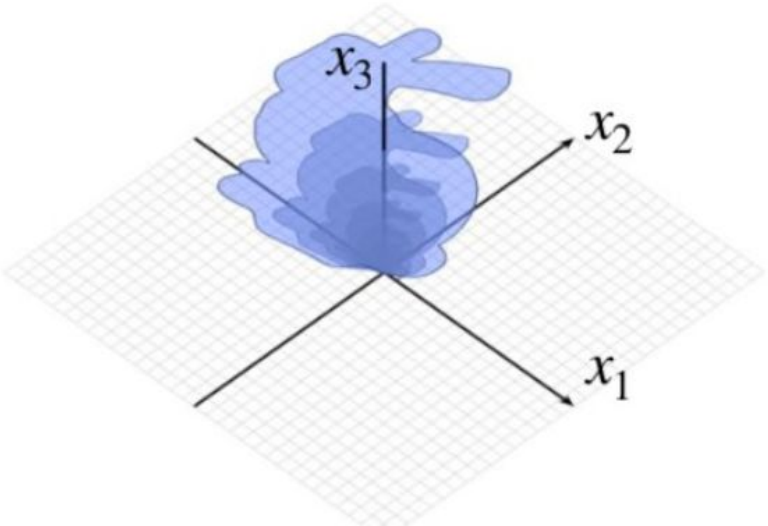
$$f_{\mathbf{u},\mathbf{v}}(\mathbf{x}) = \left(I + \mathbf{u}\mathbf{v}^{\mathsf{T}}\right)\mathbf{x}$$

Si v = (0, 0, 1), obtenemos

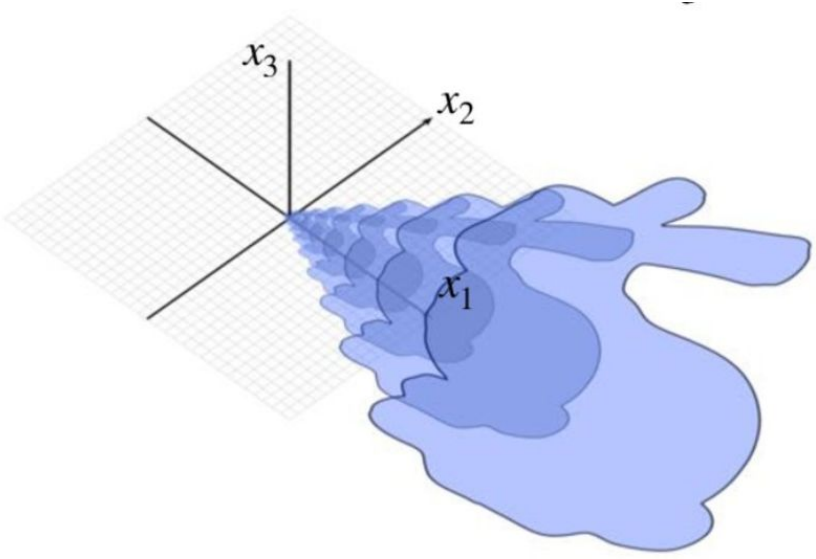
$$\begin{bmatrix} 1 & 0 & u_1 \\ 0 & 1 & u_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cp_1 \\ cp_2 \\ c \end{bmatrix} = \begin{bmatrix} c(p_1 + u_1) \\ c(p_2 + u_2) \\ c \end{bmatrix} \xrightarrow{1/c} \begin{bmatrix} p_1 + u_1 \\ p_2 + u_2 \end{bmatrix}$$

En Coordenadas Homogéneas...

La escala 2D es una escala no uniforme, en X1 y X2, preservando X3.



Traslación 2D es el shear 3D.



Transformaciones 3D en Coordenadas Homogéneas

Ahora es fácil componer transformaciones, porque todas se expresan como matrices 4D.

A las transformaciones lineales en 3D se les agrega una columna, y la traslación es shear en 4D.

point in 3D

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

rotate (x, y, z) around y by θ

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

shear (x, y) by z in (s, t) direction

$$\begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & t & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

scale x, y, z by a, b, c

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translate (x, y, z) by (u, v, w)

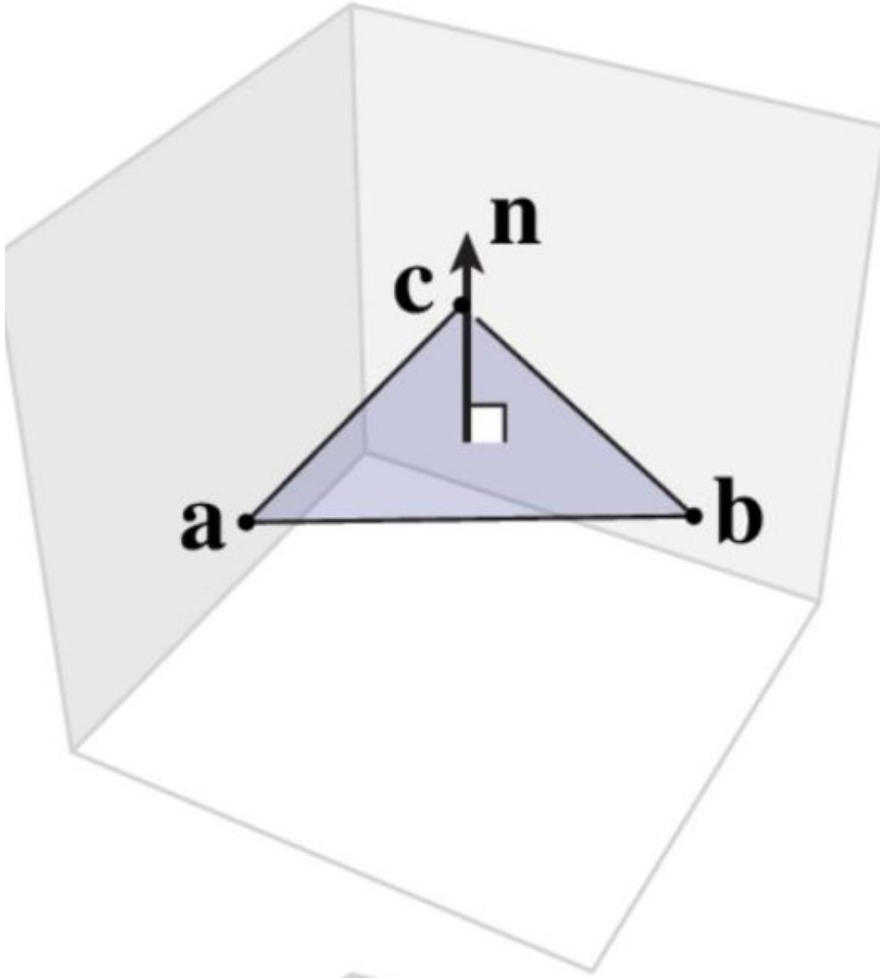
$$\begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Puntos vs Vectores

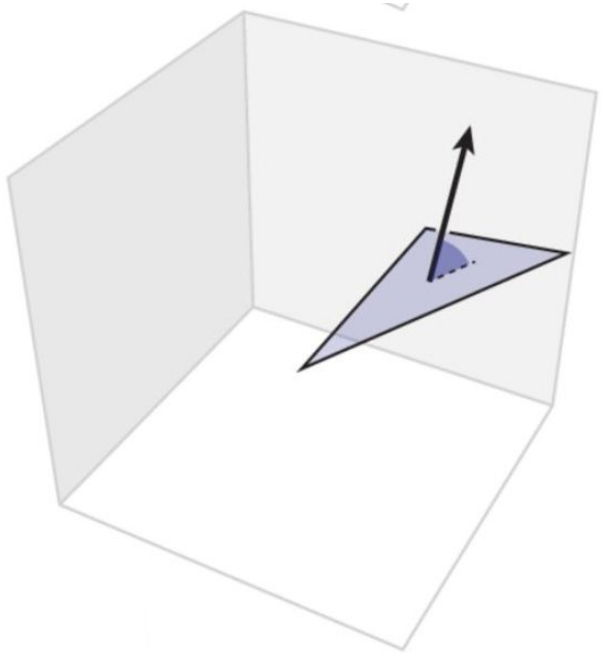
Otra utilidad de las coordenadas homogéneas es la distinción entre **puntos** y **vectores**.

¿Cuál es la diferencia?

¿Qué pasa si transformo este triángulo y su normal?



$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & u \\ 0 & 1 & 0 & v \\ -\sin\theta & 0 & \cos\theta & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Al aplicar la misma transformación a la normal, nos damos cuenta que su normal ya no es ortogonal a la superficie.

Cuando transformamos una normal (un vector), solo queremos que rote, no que se traslade.

La solución es usar la coordenada homogénea 0.

Eso logra que la traslación se ignore y por tanto solo queda la rotación.

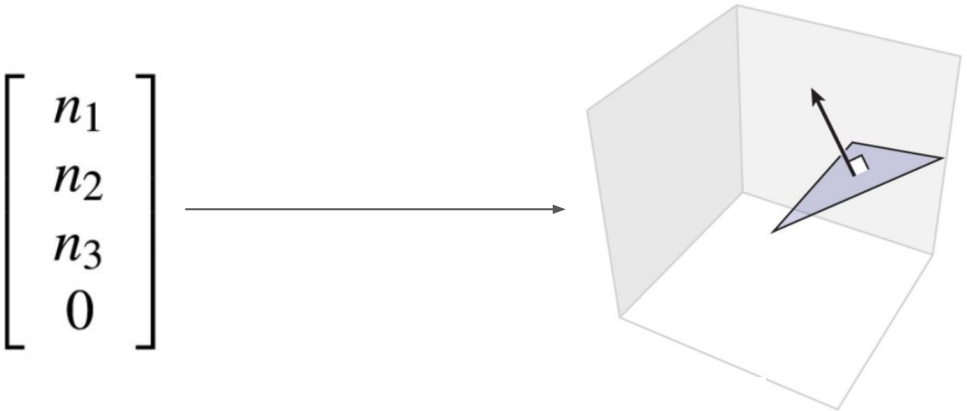
Propuesto: ¿cuál es la implicación matemática de usar c=0?

rotate normal around y by θ

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & u \\ 0 & 1 & 0 & v \\ -\sin\theta & 0 & \cos\theta & w \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ 1 \end{bmatrix}$$

translate normal by (u, v, w)

$$\begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ 0 \end{bmatrix}$$



Transformaciones 3D en Coordenadas Homogéneas

Ahora es fácil componer transformaciones, porque todas se expresan como matrices 4D.

A las transformaciones lineales en 3D se les agrega una columna, y la traslación es shear en 4D.

point in 3D

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

rotate (x, y, z) around y by θ

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

shear (x, y) by z in (s, t) direction

$$\begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & t & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

scale x, y, z by a, b, c

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translate (x, y, z) by (u, v, w)

$$\begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Cuando transformamos una normal (un vector), solo queremos que rote, no que se traslade.

La solución es usar la coordenada homogénea 0.

Eso logra que la traslación se ignore y por tanto solo queda la rotación.

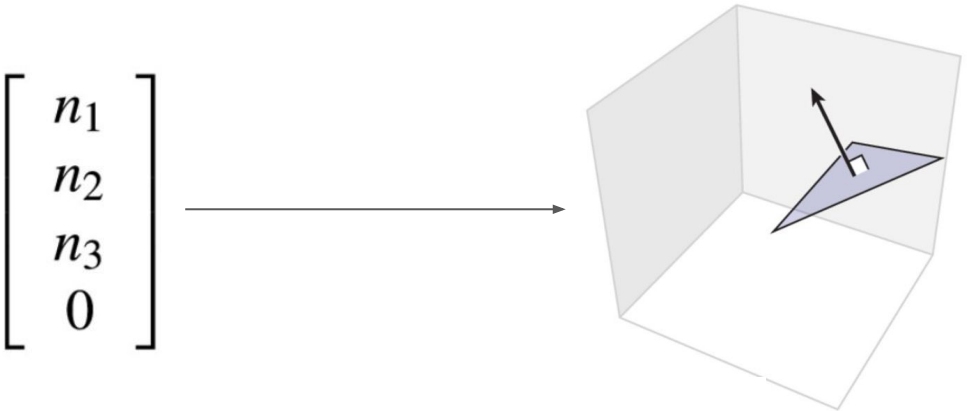
Propuesto: ¿cuál es la implicación matemática de usar c=0?

rotate normal around y by θ

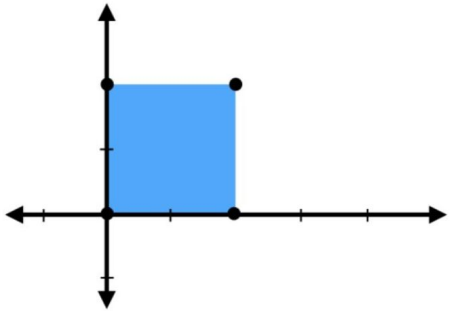
$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & u \\ 0 & 1 & 0 & v \\ -\sin\theta & 0 & \cos\theta & w \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ 1 \end{bmatrix}$$

translate normal by (u, v, w)

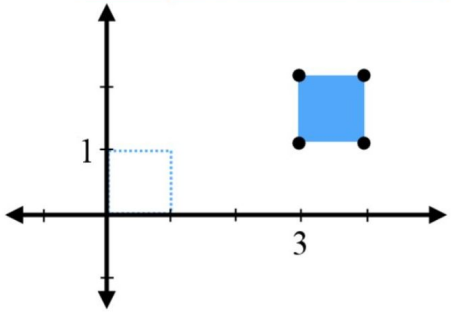
$$\begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ 0 \end{bmatrix}$$



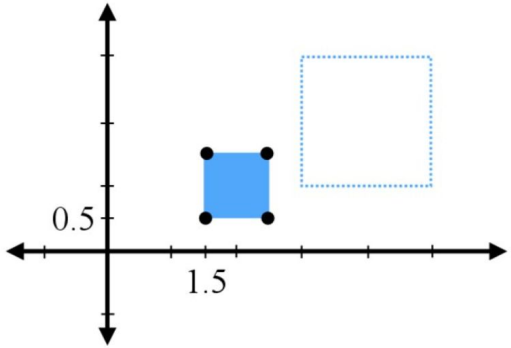
El orden importa



scale by 1/2, then translate by (3,1)



translate by (3,1), then scale by 1/2



Rotar respecto a un centro de gravedad

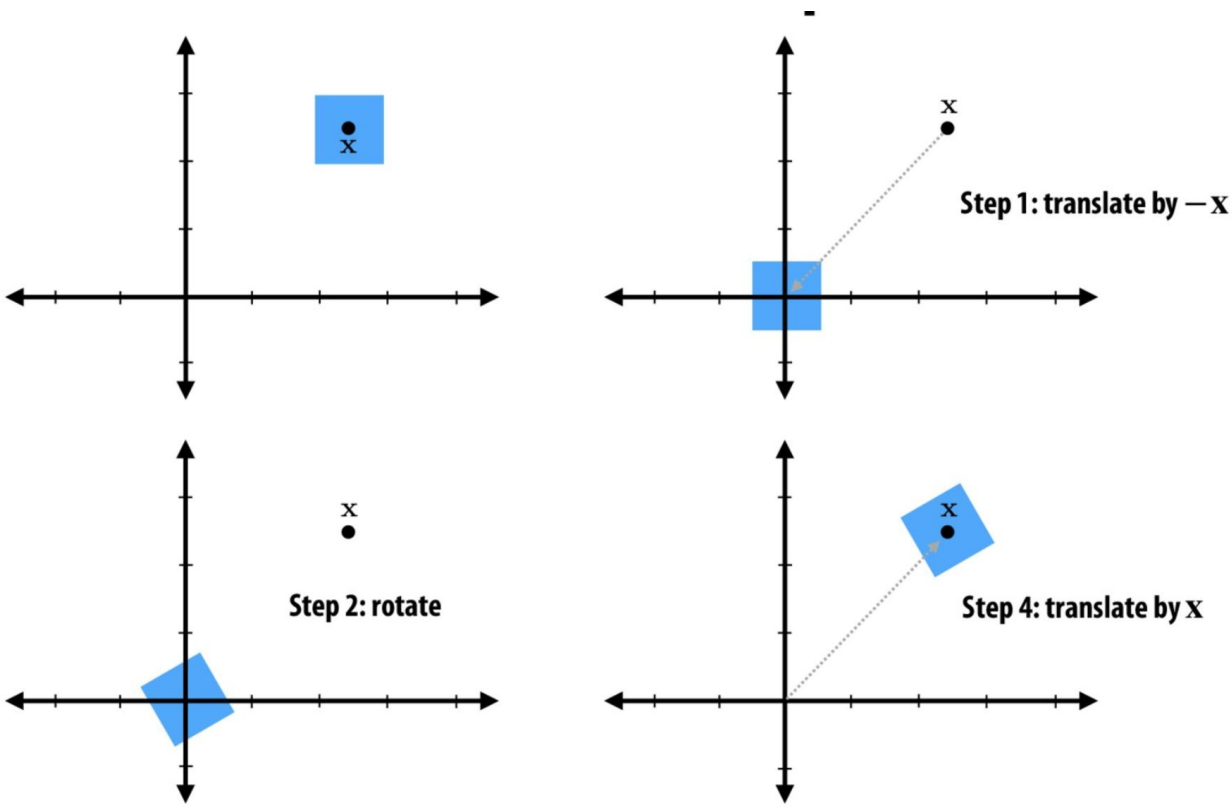
El orden importa.

Las rotaciones preservan el origen.

Usualmente no pensamos en las rotaciones respecto al origen, sino al centro de gravedad de un objeto.

Por tanto, **para rotar respecto a ese centro de gravedad, necesitamos usar traslaciones para llevar ese centro al origen**.

Luego de rotar, restauramos el origen con una nueva traslación.



¿Cómo identificar una rotación?

Las distancias se preservan (no hay deformación)

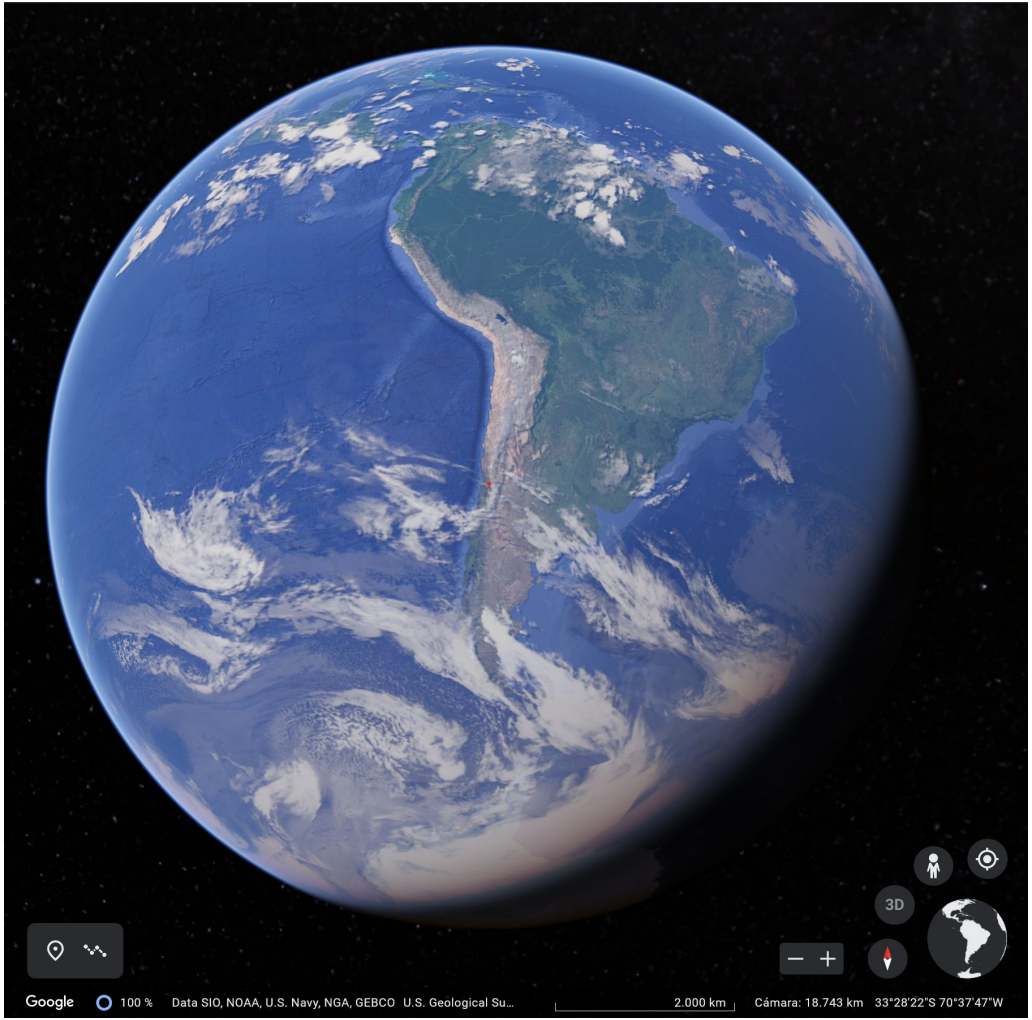
La orientación se mantiene (los vértices mantienen su coherencia)

El origen se mantiene.

Pero, ¿cómo especificar una rotación en 3D?

En el planeta un punto se especifica como latitud y longitud.

https://www.google.com/earth/index.html



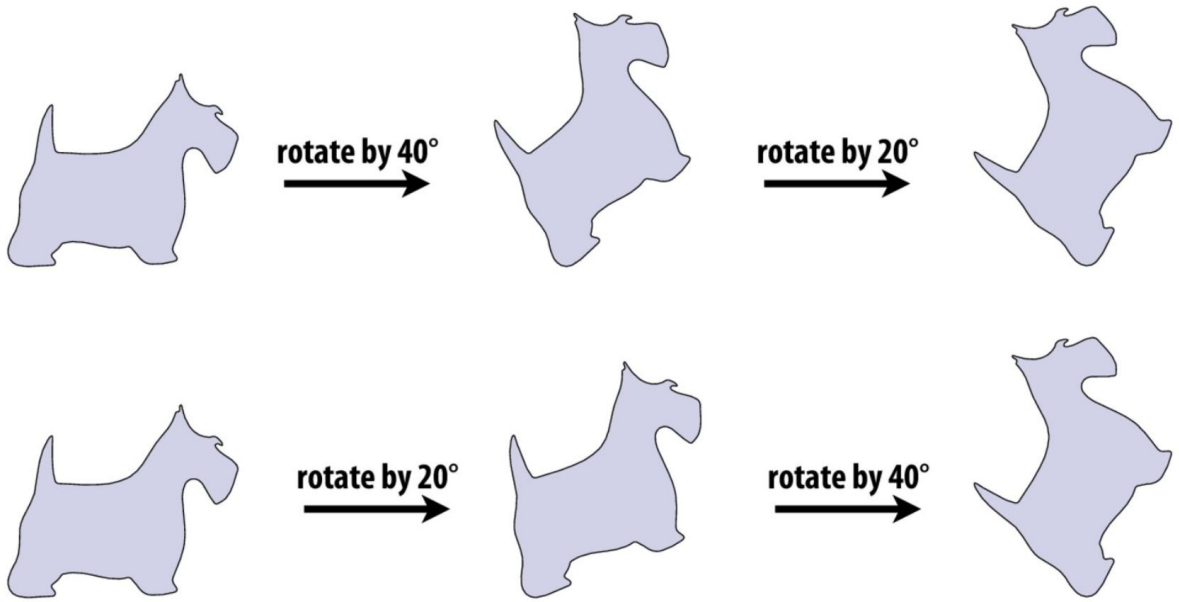
¿Cómo se especifica esta transformación?

Hay muchas maneras de lograr esa rotación.

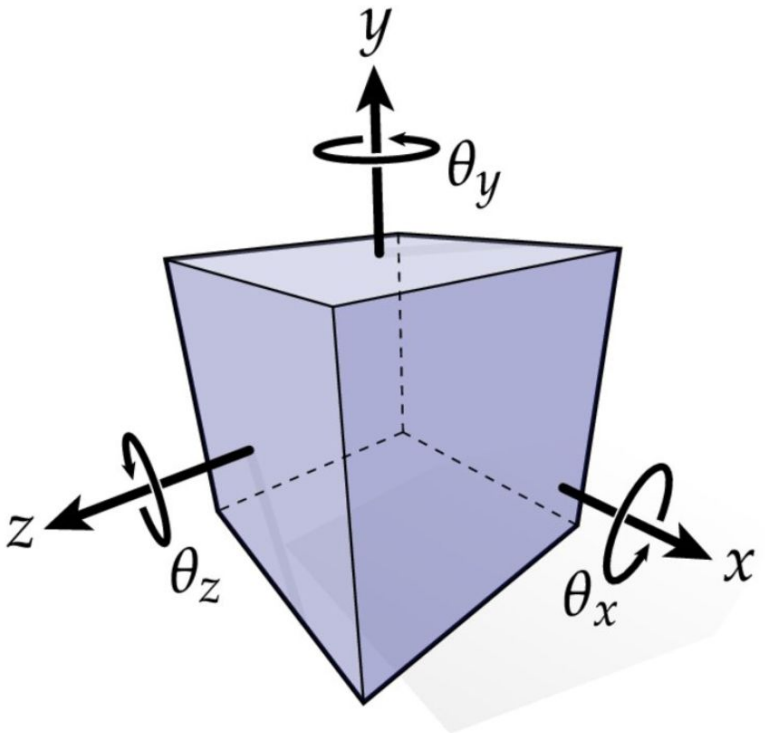
Sin embargo, debemos considerar que la rotación no es conmutativa en 3D.

Como es conmutativa en 2D, podemos confundirnos.

¡Pero en 3D no es así!



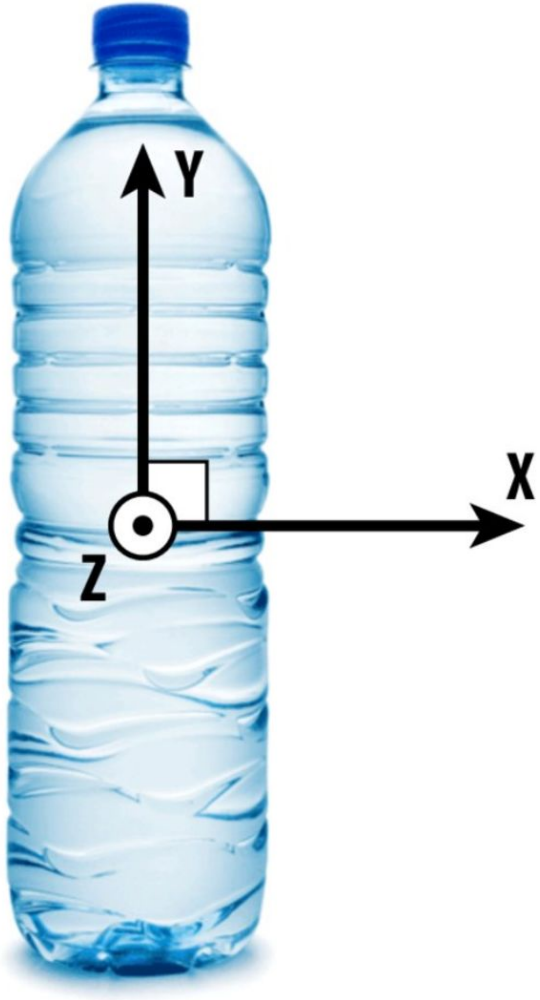
¿Por qué en 3D la rotación no es conmutativa?



Un pequeño ejercicio

1) Rotar 90 alrededor de Y, luego 90 alrededor de Z, luego 90 alrededor de X
2) Rotar 90 alrededor de Z, 90 alrededor de Y, 90 alrededor de X

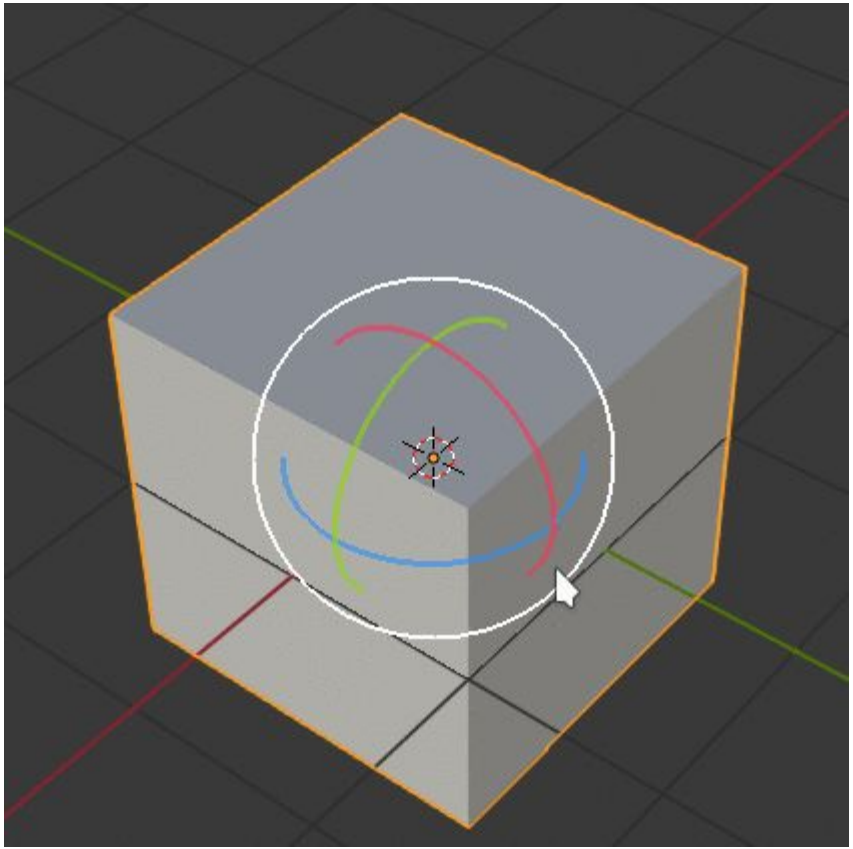
¿Cuál es la diferencia?

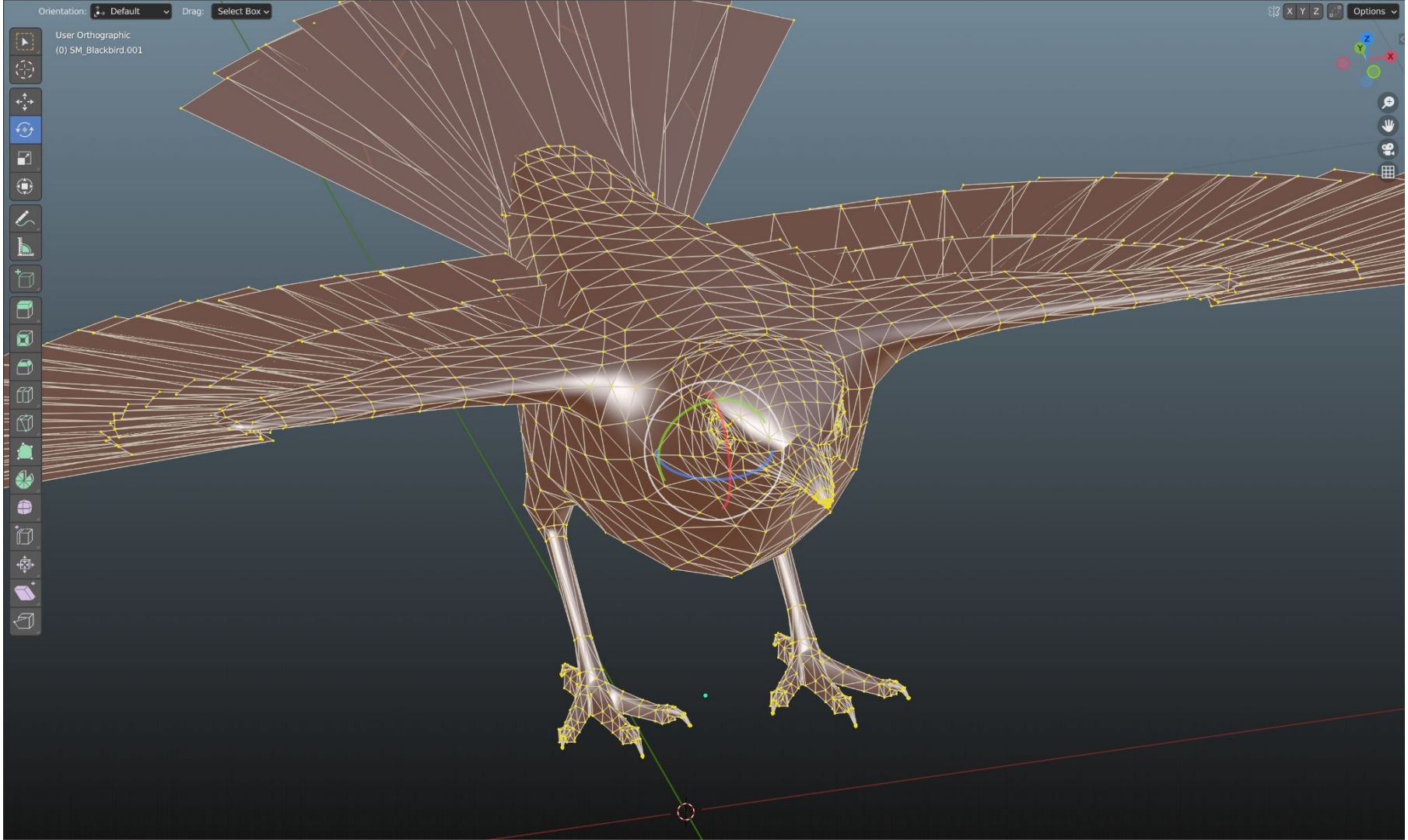


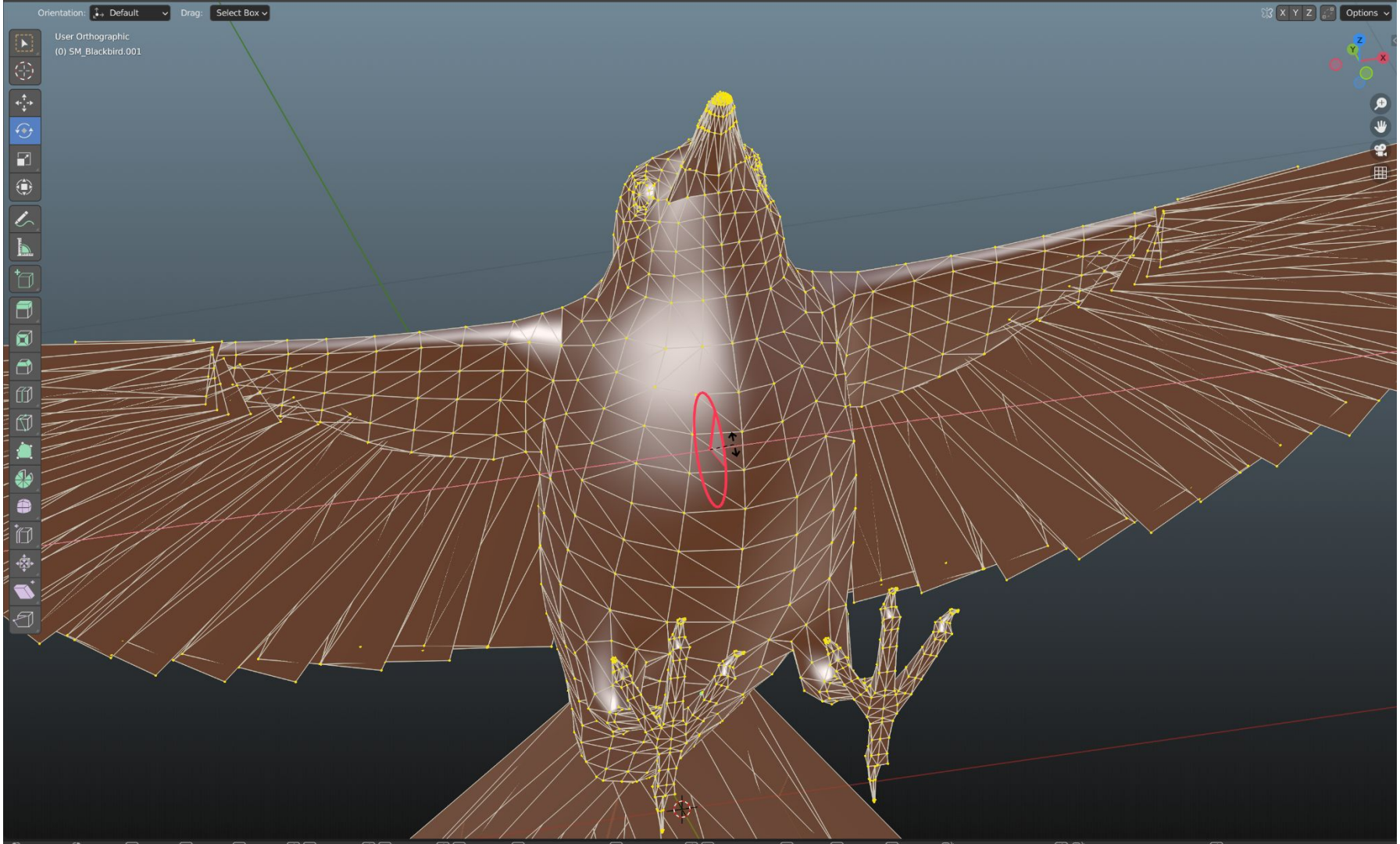
Representación típica: Ángulos de Euler

Una rotación por eje cartesiano

Es una manera frecuente de operar en programas de modelado 3D




Orientation:
Default
Drag:
Select Box
User Orthographic
(0) SM_Blackbird.001
X Y Z
Options


Orientation: Default
Drag: Select Box
User Orthographic
(0) SM_Blackbird.001
Options

Hay un problema llamado *Gimbal Lock*. ¿Imaginan cuál es?

Gimbal Lock

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \quad R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \quad R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

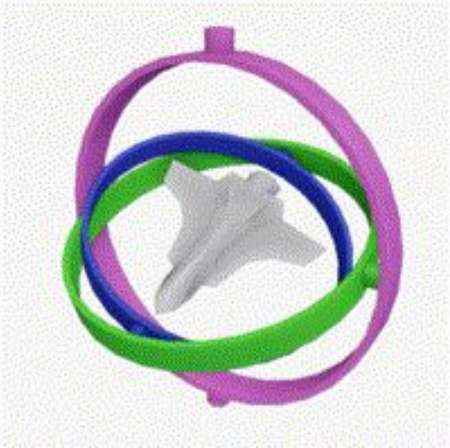
$$R_x R_y R_z = \begin{bmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_y \sin\theta_z & \sin\theta_y \\ \cos\theta_z \sin\theta_x \sin\theta_y + \cos\theta_x \sin\theta_z & \cos\theta_x \cos\theta_z - \sin\theta_x \sin\theta_y \sin\theta_z & -\cos\theta_y \sin\theta_x \\ -\cos\theta_x \cos\theta_z \sin\theta_y + \sin\theta_x \sin\theta_z & \cos\theta_z \sin\theta_x + \cos\theta_x \sin\theta_y \sin\theta_z & \cos\theta_x \cos\theta_y \end{bmatrix}$$

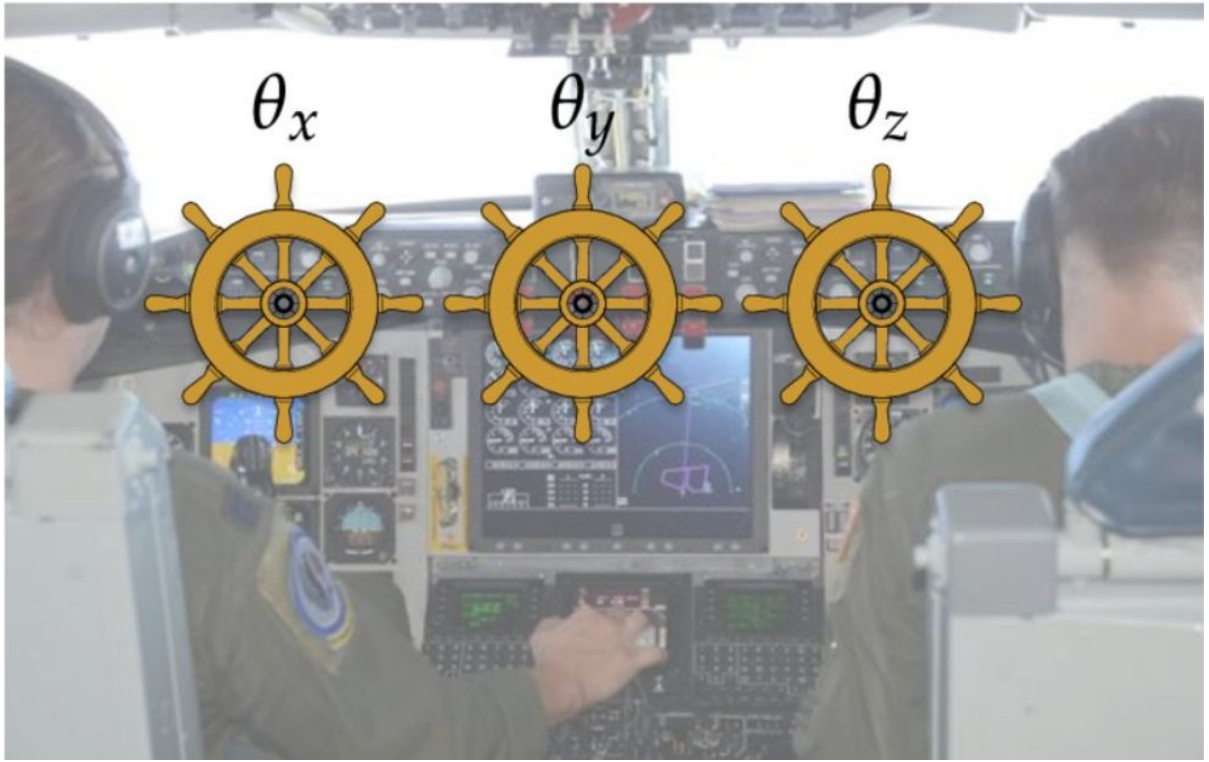
$$\theta_y = \pi/2 \text{ (so, } \cos\theta_y = 0, \sin\theta_y = 1)$$

$$\begin{bmatrix} 0 & 0 & 1 \\ \cos\theta_z \sin\theta_x + \cos\theta_x \sin\theta_z & \cos\theta_x \cos\theta_z - \sin\theta_x \sin\theta_z & 0 \\ -\cos\theta_x \cos\theta_z + \sin\theta_x \sin\theta_z & \cos\theta_z \sin\theta_x + \cos\theta_x \sin\theta_z & 0 \end{bmatrix}$$

Sin importar como ajustemos los ángulos, perdimos un eje de libertad.

¡Eso puede ser crítico en algunas ocasiones!





Propuesta: rotaciones 2D expresadas como operaciones en números complejos.

Pasemos a un tema relacionado: Almacenamiento de Objetos 3D


Orientation: Default
Drag: Select Box
User Orthographic
(0) SM_Blackbird.001
Options
¿Cómo trabajar con este pajarito en 3D?

Formato OBJ

El formato OBJ es uno de los más simples para almacenar información de objetos 3D (superficies).

Define:

coordenadas de vértices **v**

coordenadas de texturas (lo veremos más adelante) **vt**

vectores normales **vt**

triángulos **f**

cube.obj

```
# Blender v2.76 (sub 0) OBJ File: ''
# www.blender.org
mtllib cube.mtl
o Cube
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -0.999999
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vt 1.000000 0.333333
vt 1.000000 0.666667
vt 0.666667 0.666667
vt 0.666667 0.333333
vt 0.666667 0.000000
vt 0.000000 0.333333
vt 0.000000 0.000000
vt 0.333333 0.000000
vt 0.333333 1.000000
vt 0.000000 1.000000
vt 0.000000 0.666667
vt 0.333333 0.333333
vt 0.333333 0.666667
vt 1.000000 0.000000
vn 0.000000 -1.000000 0.000000
vn 0.000000 1.000000 0.000000
vn 1.000000 0.000000 0.000000
vn -0.000000 0.000000 1.000000
vn -1.000000 -0.000000 -0.000000
vn 0.000000 0.000000 -1.000000
usemtl Material
s off
f 2/1/1 3/2/1 4/3/1
f 8/1/2 7/4/2 6/5/2
f 5/6/3 6/7/3 2/8/3
f 6/8/4 7/5/4 3/4/4
f 3/9/5 7/10/5 8/11/5
f 1/12/6 4/13/6 8/11/6
f 1/4/1 2/1/1 4/3/1
f 5/14/2 8/1/2 6/5/2
f 1/12/3 5/6/3 2/8/3
f 2/12/4 6/8/4 3/4/4
f 4/13/5 3/9/5 8/11/5
f 5/6/6 1/12/6 8/11/6
```

pip install trimesh

¡**trimesh** es la biblioteca que nos ayudará a cargar objetos 3D!

ej: ex_pajarito.py

¿Cómo lo hace?

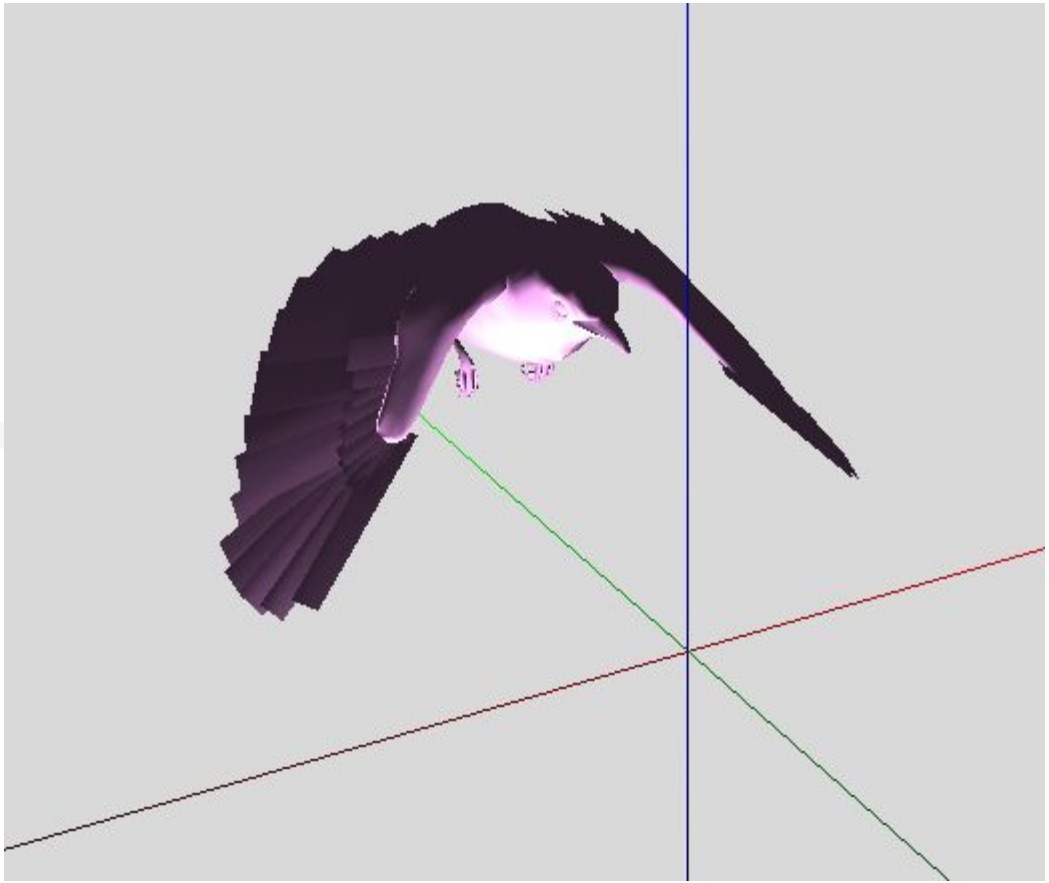
Ver https://github.com/mikedh/trimesh/blob/main/trimesh/viewer/windowed.py

```
# Drawing shapes
glUseProgram(pipeline.shaderProgram)
glUniform3f(glGetUniformLocation(pipeline.shaderProgram, "viewPosition"), viewPos
[0], viewPos[1], viewPos[2])
glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "view"), 1,
GL_TRUE, view)

bird_rotation_matrix = tr.matmul([tr.translate(*(centroid*5)), tr.rotationX
(bird_theta), tr.translate(*(-centroid*5))])

glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "transform"), 1,
GL_TRUE, bird_rotation_matrix)


glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "model"), 1,
GL_TRUE, tr.uniformScale(5))
pipeline.drawCall(gpuSuzanne)
```



¿Preguntas?

Propuesto: leer
https://math.stackexchange.com/questions/1570016/why-are-homogenous-coordinates-needed-in-image-projection