UNIVERSIDADE DA CORUÑA

epef

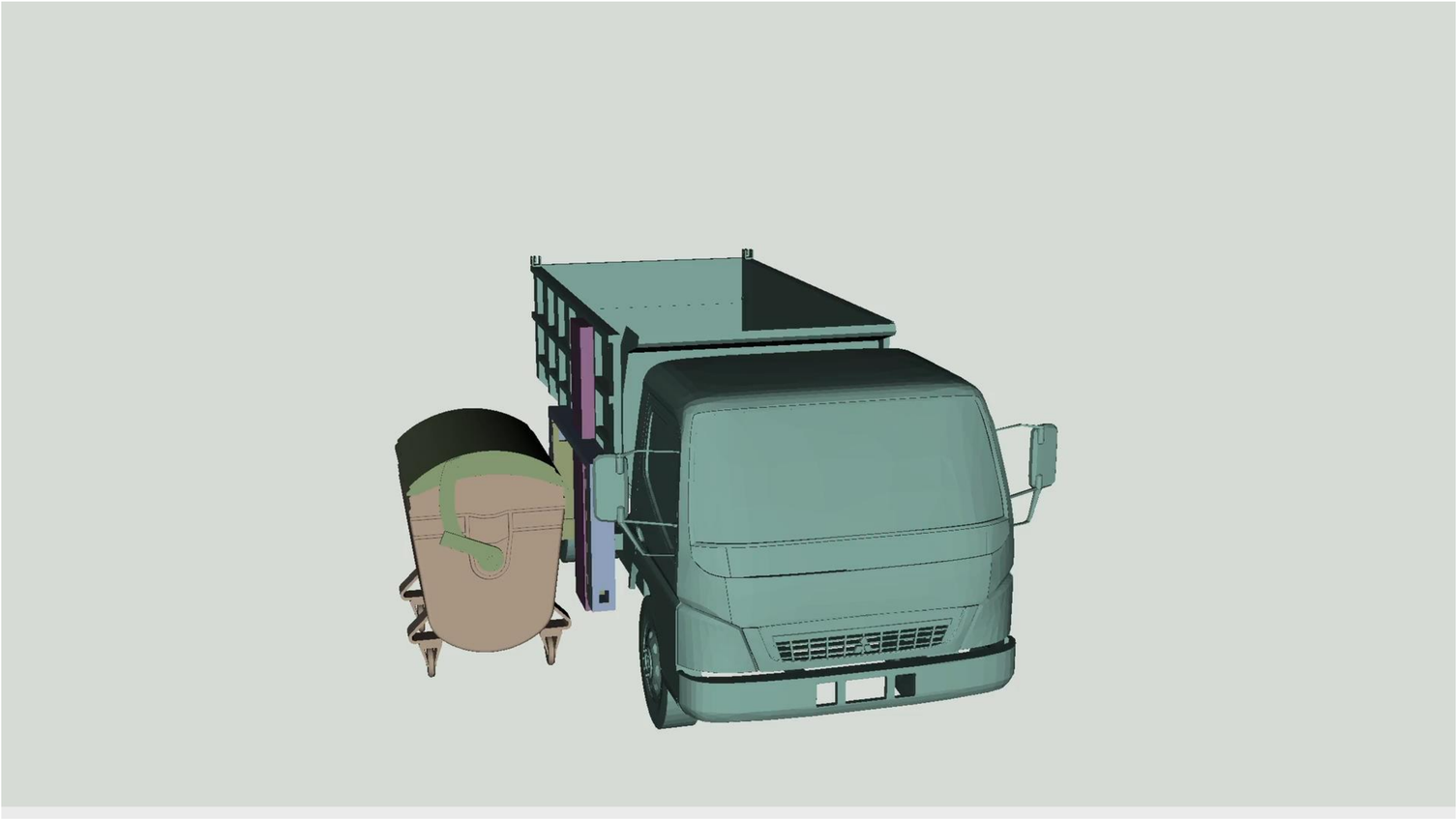
Práctica de programación

Informática

Grados en Ing. Mecánica e Ing. en Tecnologías Industriales - 1er curso

Resumen

- El programa debe generar una ***animación*** en la cual un camión de la basura vacía un contenedor en su interior.

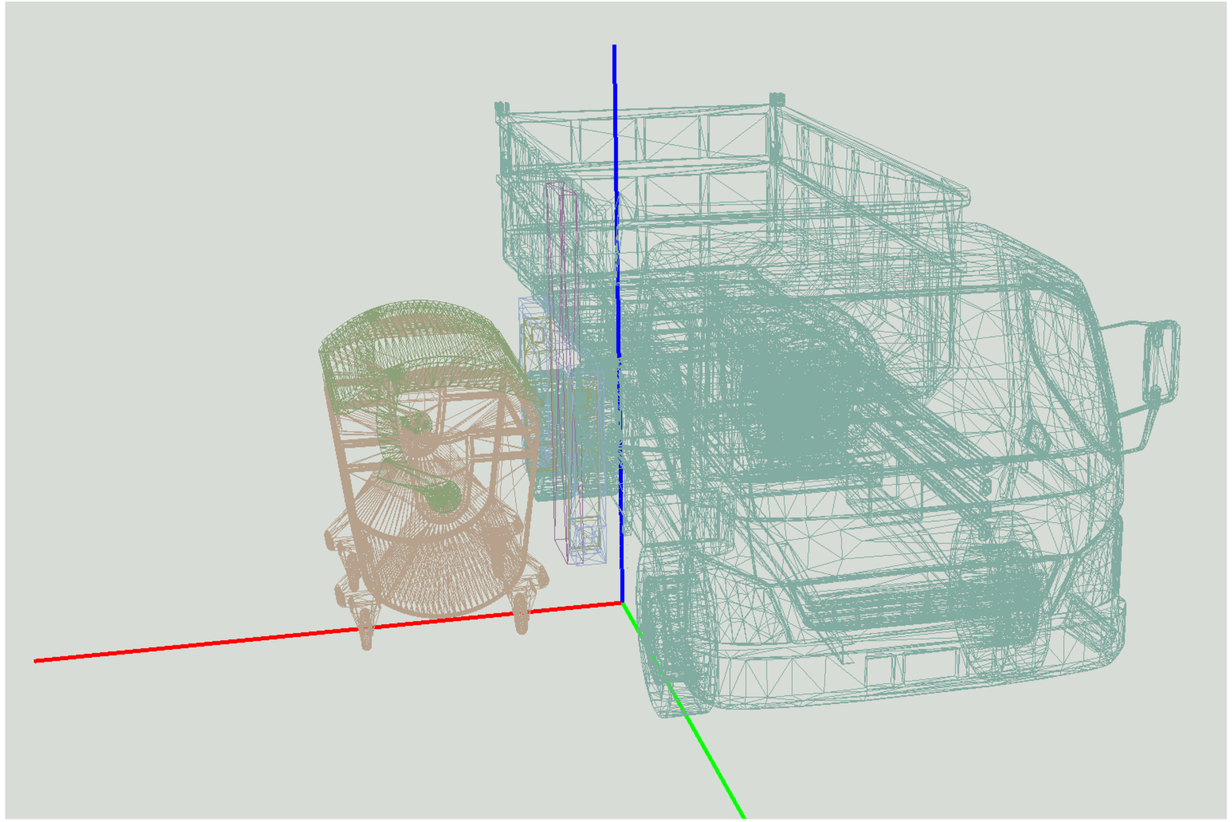


Resumen

- La ***animación*** se crea pintando las diferentes ***piezas*** que componen la máquina en su posición correcta, que irá cambiando con el paso del tiempo.

Resumen

- Las ***piezas*** son mallas compuestas por ***triángulos***, cada uno definido por sus 3 vértices, que tienen unas coordenadas en un sistema de coordenadas cuyo origen está donde se ubica el “empujador” del camión de la basura.



Resumen

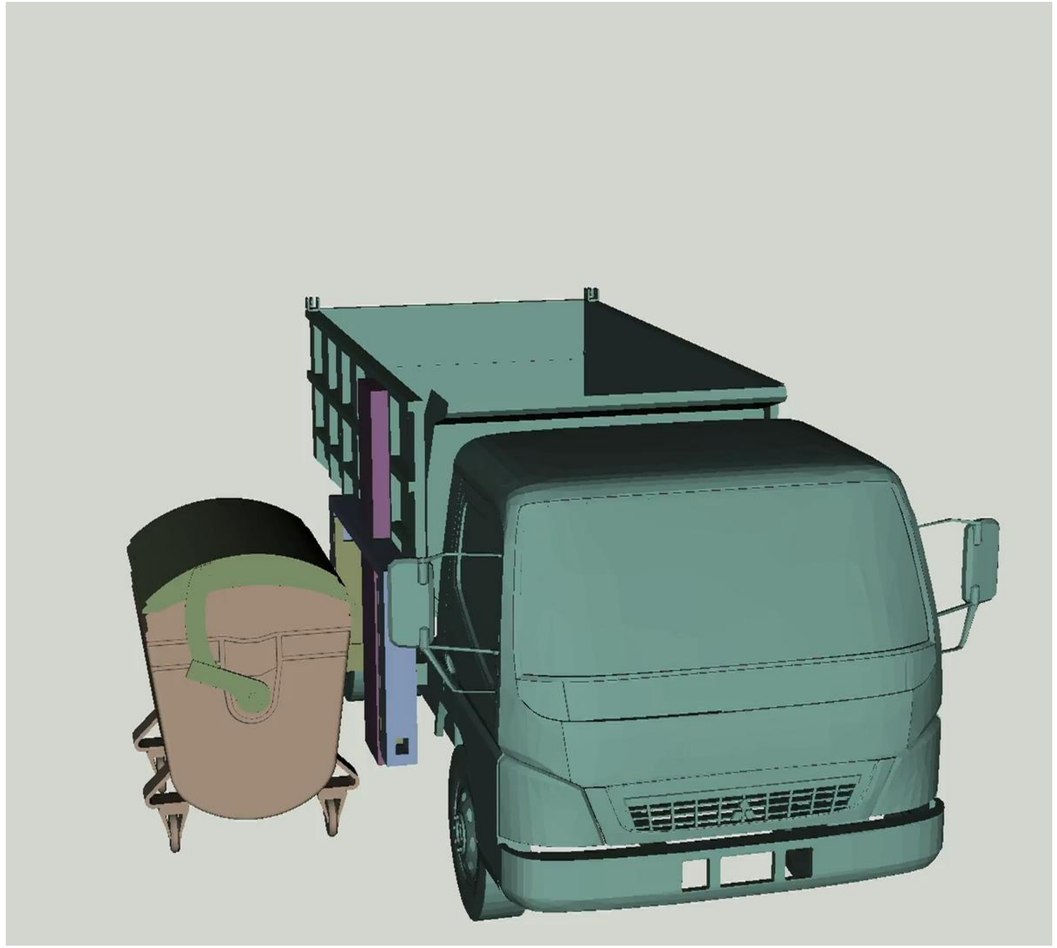
- Para pintar las piezas en su posición correcta en cada instante de tiempo tendremos que irlas rotando y trasladando, cambiando así las coordenadas de los ***triángulos*** que las forman, para lo cual usaremos matrices de transformación.

Resumen

- Una posible solución implicaría:
  - Extender el empujador que desplazará la deslizadera.
  - Elevar el brazo 1 de la horquilla hasta enganchar el contenedor.
  - Elevar la horquilla al mismo tiempo que el brazo 2 rota para enganchar la tapa.
  - Empieza a rotar el brazo 1, así como el brazo 2 para abrir la tapa.
  - Acaba de rotar el brazo 1 para vaciar el contenedor.
- Debemos de pintar el camión y el contenedor en esas ***posiciones clave***…
- … y para crear el efecto deseado de animación, pintarlo también en posiciones intermedias. Cuantas más posiciones intermedias, más fluida será la animación.

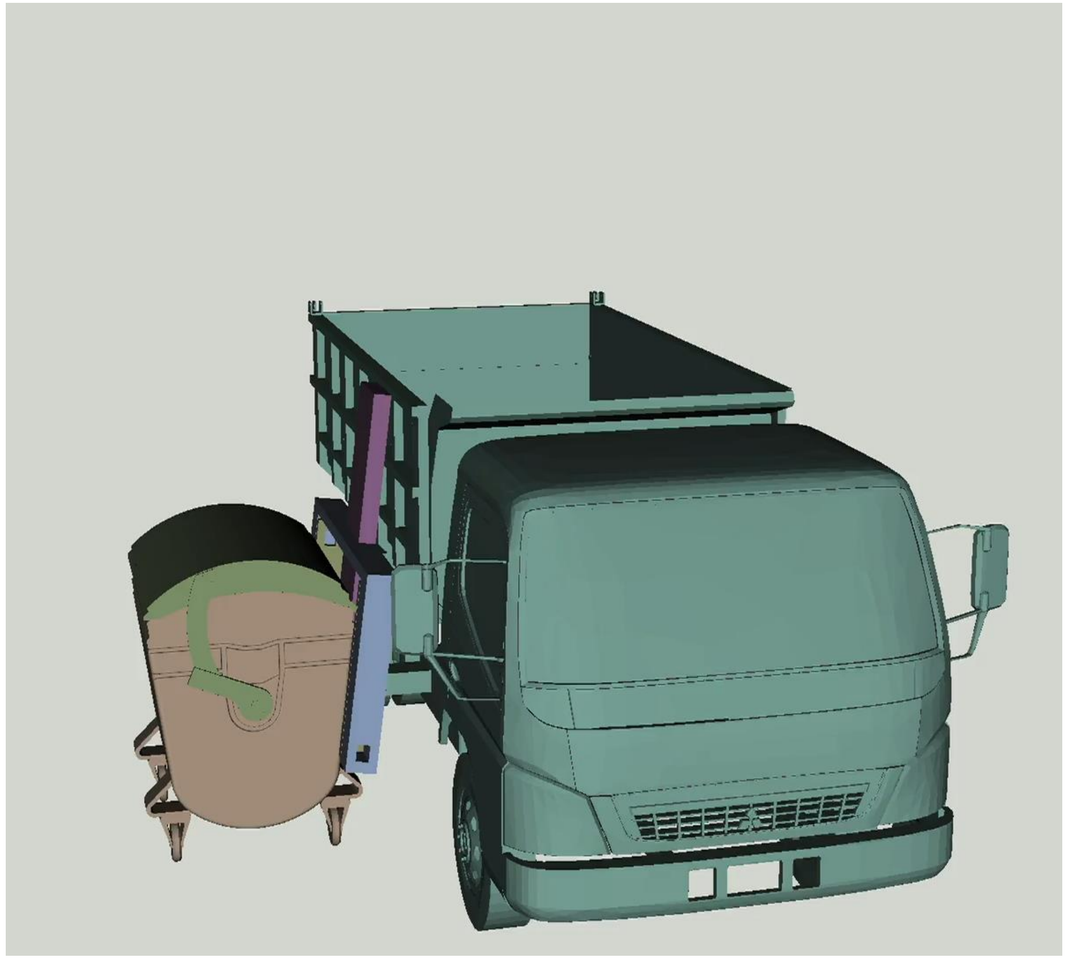
Ejemplo

- Extender el empujador que desplazará la deslizadera:



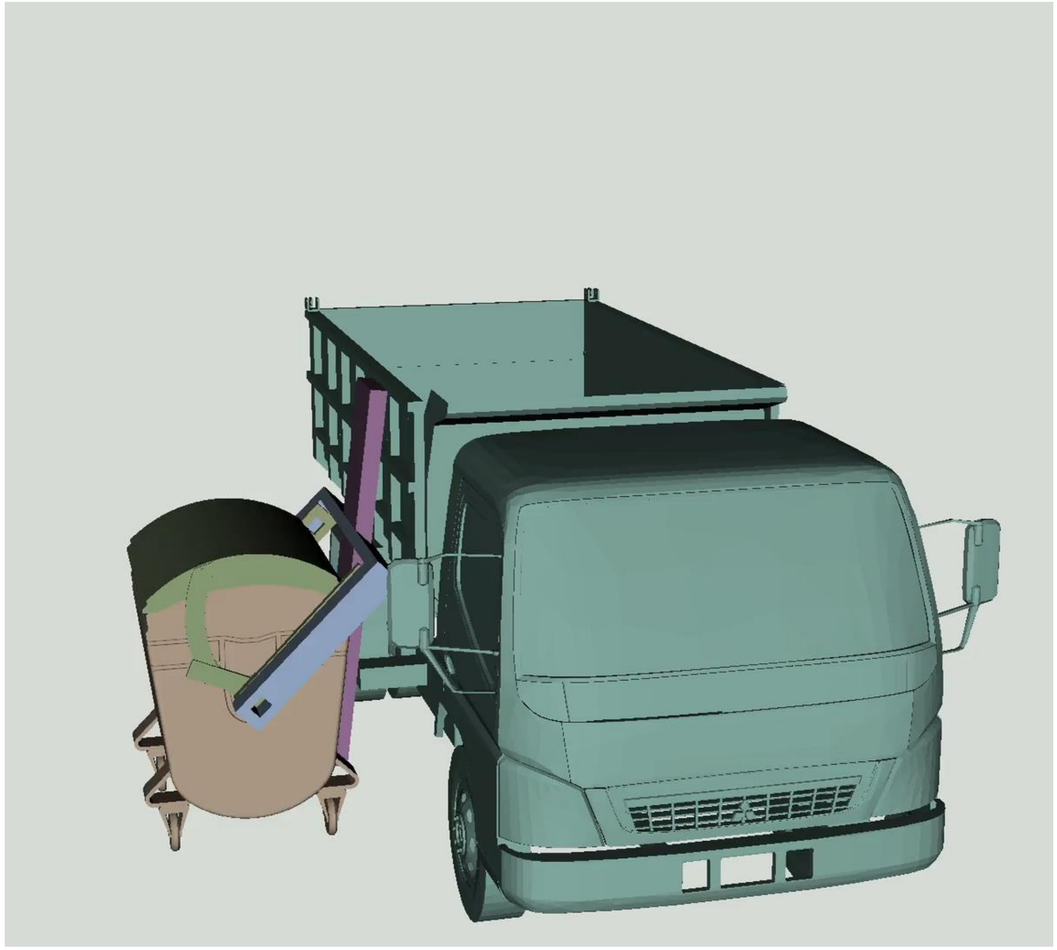
Ejemplo

- Elevar el brazo 1 de la horquilla hasta enganchar el contenedor.



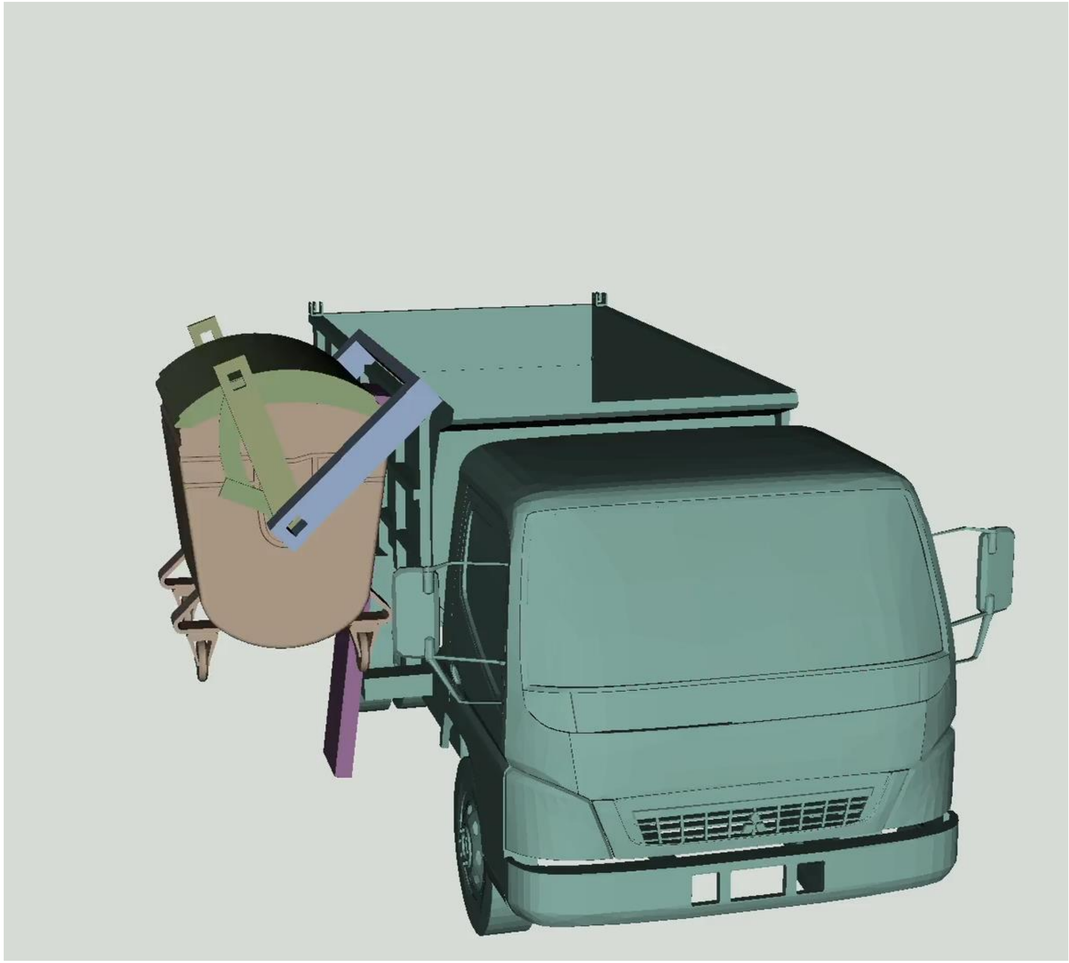
Ejemplo

- Elevar la horquilla al mismo tiempo que el brazo 2 rota para enganchar la tapa.



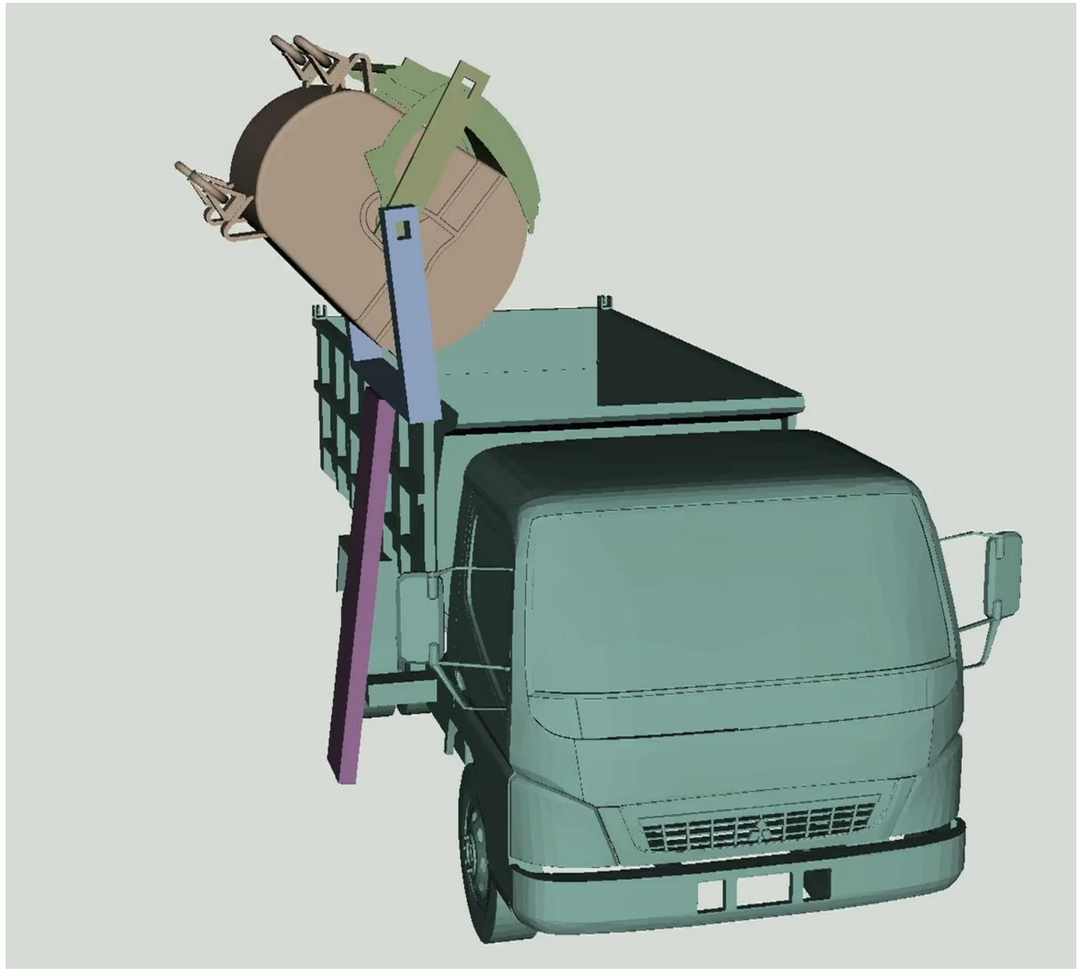
Ejemplo

- Empieza a rotar el brazo 1, así como el brazo 2 para abrir la tapa.



Ejemplo

- Acaba de rotar el brazo 1 para vaciar el contenedor.



Matrices de transformación

MATRICES DE ROTACIÓN EN X, Y, Z

MATRIZ DE TRASLACIÓN

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Tl(v) = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Una matriz de transformación permite definir una operación de traslación y / o rotación para un punto(s) y se calcula como la multiplicación de las matrices indicadas arriba:
  - T = Tl @ Rz @ Ry @ Rx
- Las nuevas coordenadas del punto(s) se pueden calcular mediante la multiplicación de dicha matriz por el vector(es) con las coordenadas del punto(s).
- Utilizaremos la librería **numpy-stl**, que permite aplicar fácilmente una matriz de transformación a una malla (con los puntos que definen un objeto) con el método **transform()**.

Matrices de transformación

- Podemos aplicar varias transformaciones simultáneamente a una pieza multiplicando sus correspondientes matrices de transformación.

- Estas transformaciones debemos de “leerlas” / visualizarlas como si las aplicásemos a la pieza secuencialmente de izquierda a derecha.

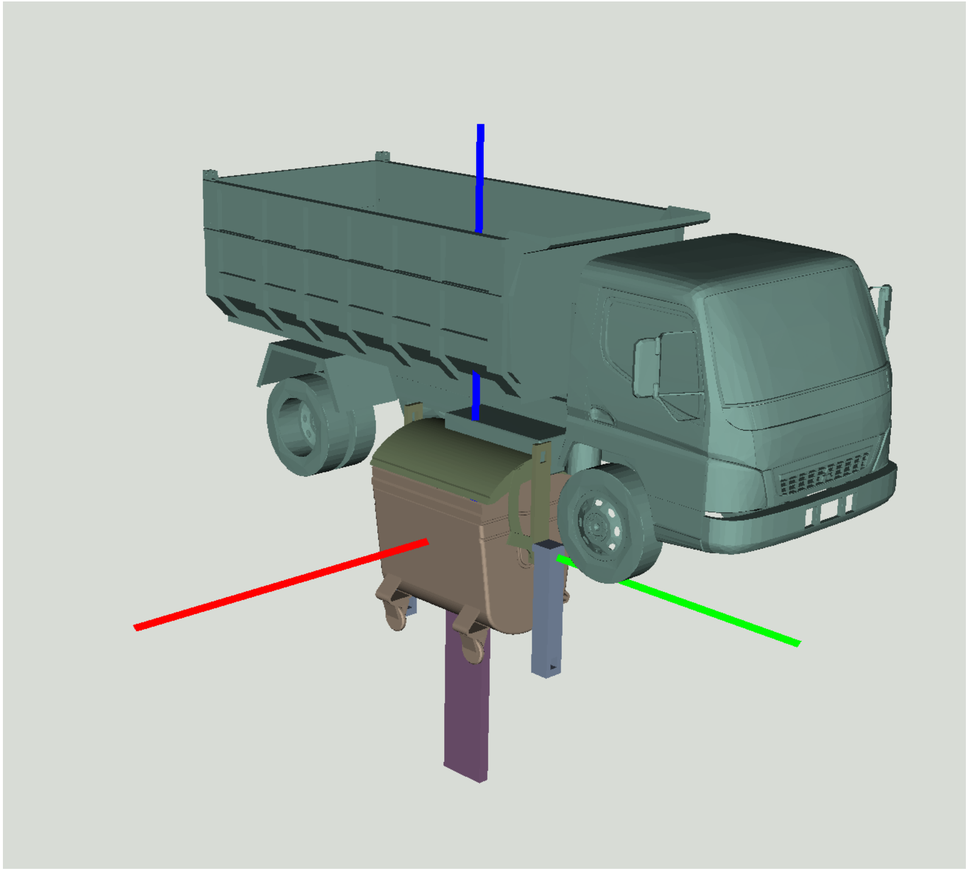
- Teniendo en cuenta lo anterior, si queremos mover una pieza (B) solidariamente con otra (A) y, además, que ejecute otro movimiento, solo tenemos que calcular la matriz de transformación de B como el producto de la matriz de transformación de A por otra matriz de transformación que refleje el movimiento adicional que queremos para B.

- A continuación, veremos varios ejemplos sobre esto.

Exportación de las piezas

- Los objetos aparecerán sobre el escenario en las coordenadas que tenían en SolidWorks cuando se exportaron.

- Por ello, se deben de exportar las piezas de forma que el punto sobre el que van a rotar esté en el origen de coordenadas, para que las rotaciones se visualicen correctamente.



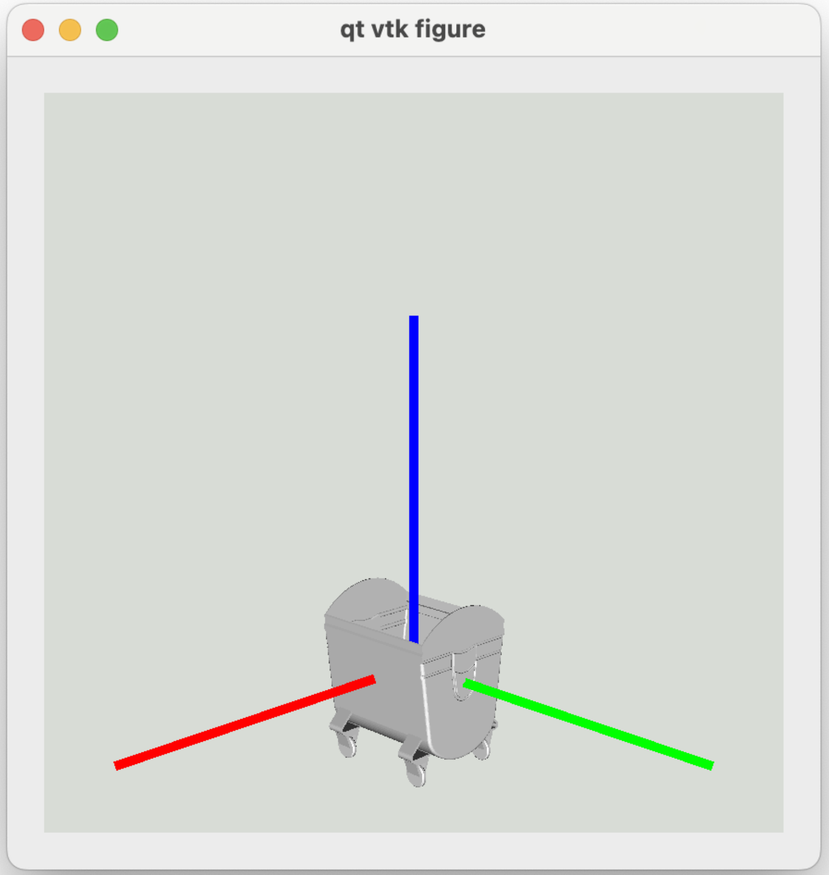
Ejemplos

- En los ejemplos que vienen a continuación solo aparecerán algunas de las piezas.

- Estos ejemplos están pensados para probar el código de forma interactiva, pero recordad que lo que se entregue tiene que ser un programa convenientemente estructurado en funciones.

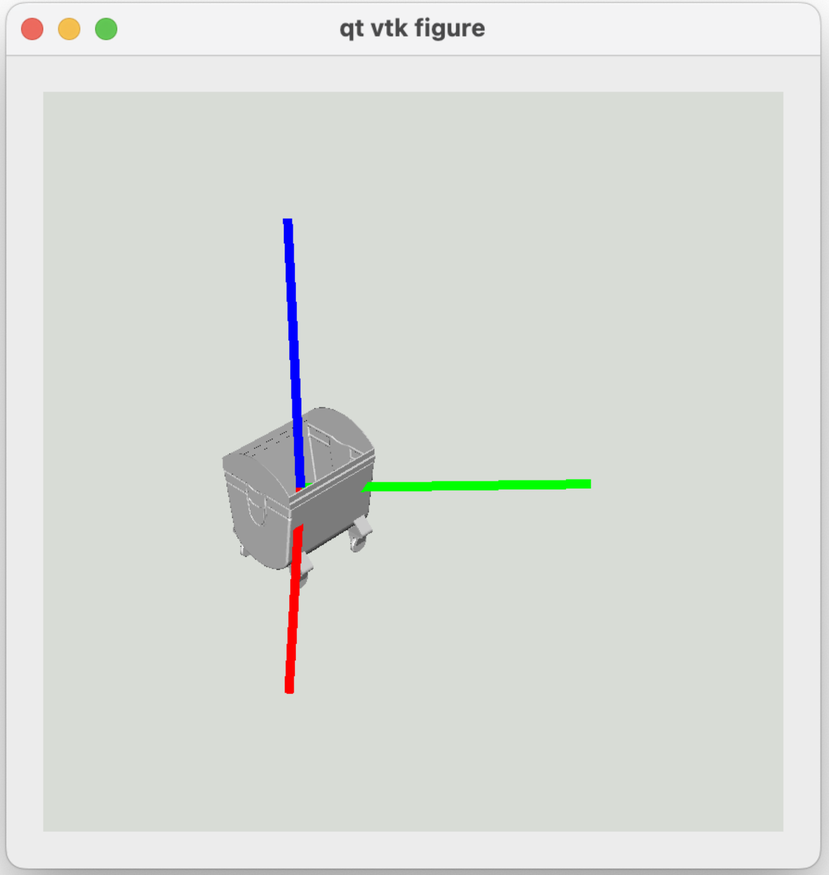
Carga de un modelo

```
import numpy
# Para mostrar la escena.
import vtkplotlib as vpl
# Para cargar los modelos 3D.
from stl import mesh


def pinta_ejes():
    vpl.plot(numpy.array([[0, 0, 0], [3, 0, 0]]), color=(0, 255, 0), line_width=10.0, label="X")
    vpl.plot(numpy.array([[0, 0, 0], [0, 3, 0]]), color=(0, 0, 255), line_width=10.0, label="Y")
    vpl.plot(numpy.array([[0, 0, 0], [0, 0, 3]]), color=(255, 0, 0), line_width=10.0, label="Z")


# Creamos la ventana de visualización.
vpl.QtFigure()
# Cargamos un modelo.
modelo = mesh.Mesh.from_file("contenedor.stl")
# Lo añadimos a la ventana.
vpl.mesh_plot(modelo)
# Pintamos los ejes (solo para ayudar a visualizar)
pinta_ejes()
# Ponemos la cámara donde nos interesa.
vpl.view(camera_position=(8.0, 4.0, 8.0), focal_point=(-1.0, 1.5, -1.0))
# Visualizamos la ventana (la figura se borra al cerrarla).
vpl.show()
```



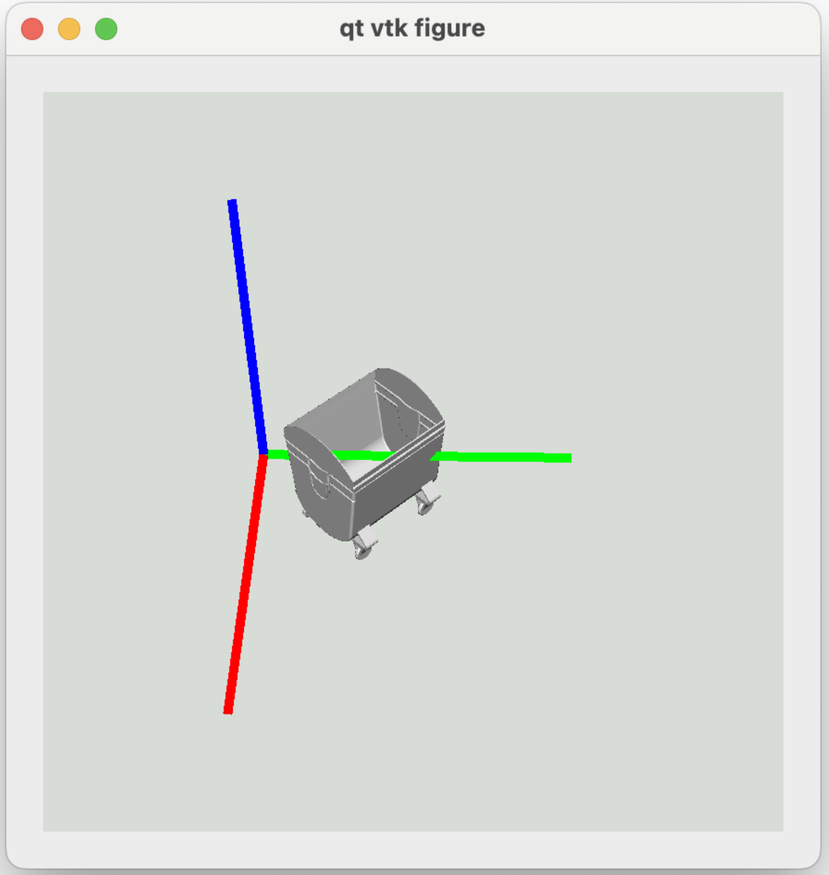
Rotación en Y

```
from copy import deepcopy

def rotate_y(angle):
    matrix = numpy.identity(4)
    matrix[0, 0:3] = [numpy.cos(angle), 0.0, numpy.sin(angle)]
    matrix[2, 0:3] = [-numpy.sin(angle), 0.0, numpy.cos(angle)]
    return matrix

vpl.QtFigure()
# Calculamos la matriz de transformación para rotar la pieza 45º en Y.
Rx = numpy.identity(4)
Ry = rotate_y(numpy.radians(45))
Rz = numpy.identity(4)
T = numpy.identity(4)
Mt = T @ Rz @ Ry @ Rx
# Copiamos el modelo para no modificar sus coordenadas originales.
copia = deepcopy(modelo)
# Aplicamos la matriz de transformación.
copia.transform(Mt)
vpl.mesh_plot(copia)
pinta_ejes()
vpl.show()
```



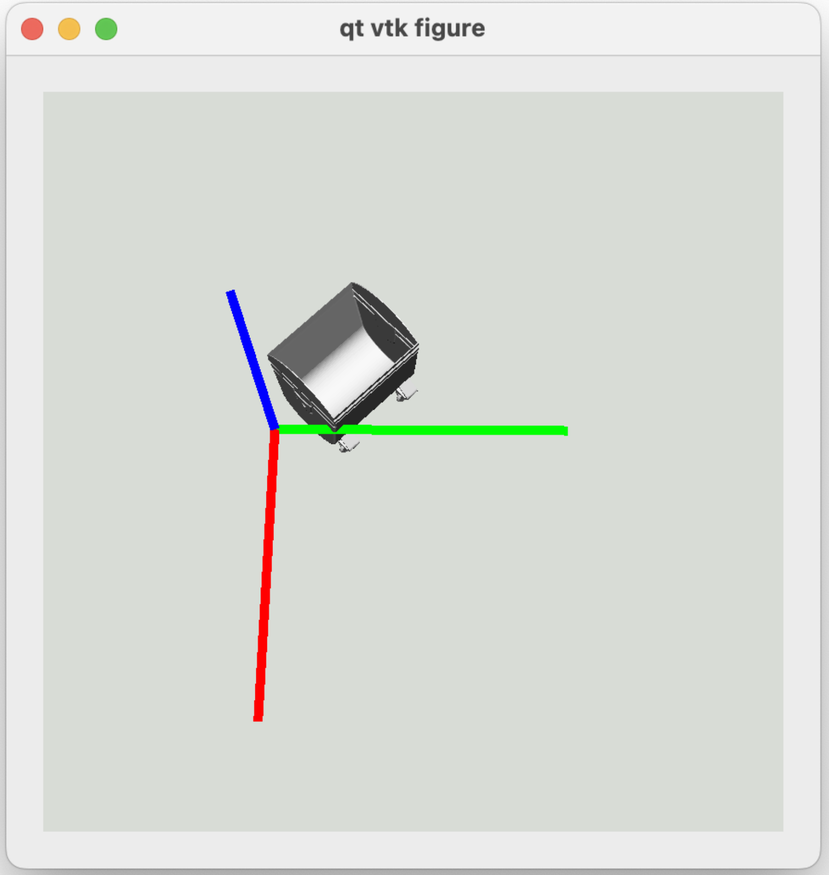
Traslación en X + rotación en Y

```
vpl.QtFigure()
# Calculamos la matriz de transformación para trasladar 1 metro
# en X y, además, rotar como antes.
T = numpy.identity(4)
T[0:3, 3] = [1.0, 0.0, 0.0]
# Mantenemos las matrices de rotación del ejemplo anterior.
Mt = T @ Rz @ Ry @ Rx
copia = deepcopy(modelo)
copia.transform(Mt)
vpl.mesh_plot(copia)
pinta_ejes()
vpl.show()
```



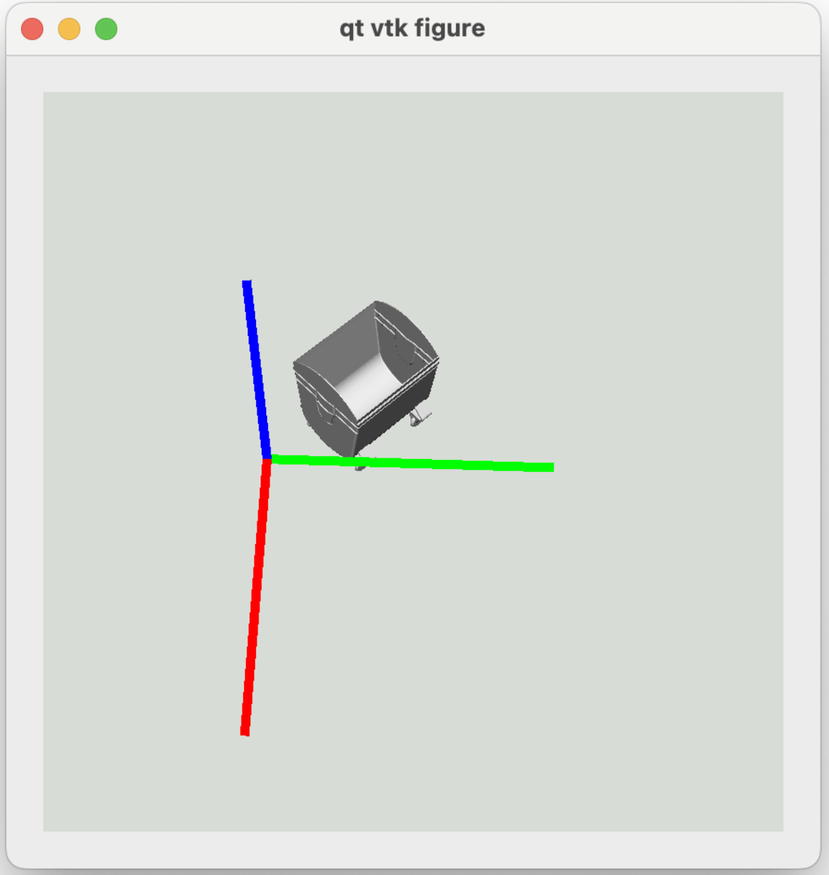
Rotación en Y + traslación en X

```
vpl.QtFigure()
# El orden de las matrices determina el orden al aplicar las transformaciones.
# Estas se hacen de izquierda a derecha y las tenemos que visualizar como si el eje de
# coordenadas se moviese con la pieza después de cada transformación.
# Comparando este ejemplo con el anterior, ahora primero se rota y después se traslada.
# En el cálculo de la matriz de transformación podemos omitir las matrices identidad,
# pero con esto perdemos generalidad (por ejemplo, si queremos meter esto en una función).
Mt = Ry @ T
copia = deepcopy(modelo)
copia.transform(Mt)
vpl.mesh_plot(copia)
pinta_ejes()
vpl.show()
```



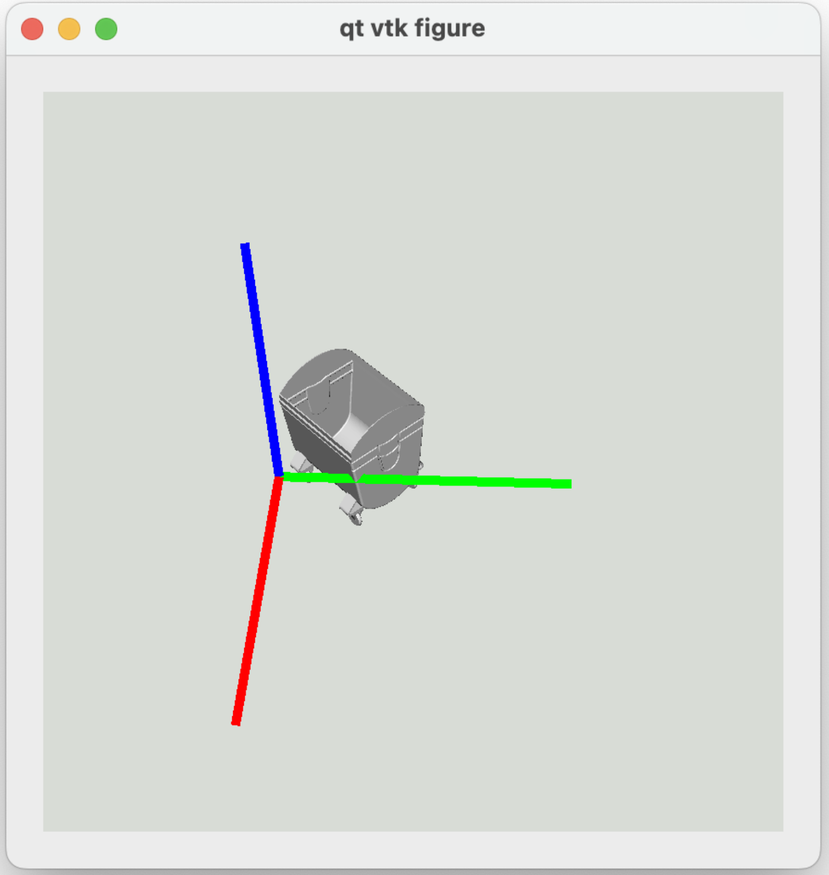
Traslación en X y Z + rotación en Y

```
vpl.QtFigure()
T = numpy.identity(4)
T[0:3, 3] = [1.0, 0.0, -1.0]
Mt = T @ Ry
copia = deepcopy(modelo)
copia.transform(Mt)
vpl.mesh_plot(copia)
pinta_ejes()
vpl.show()
```



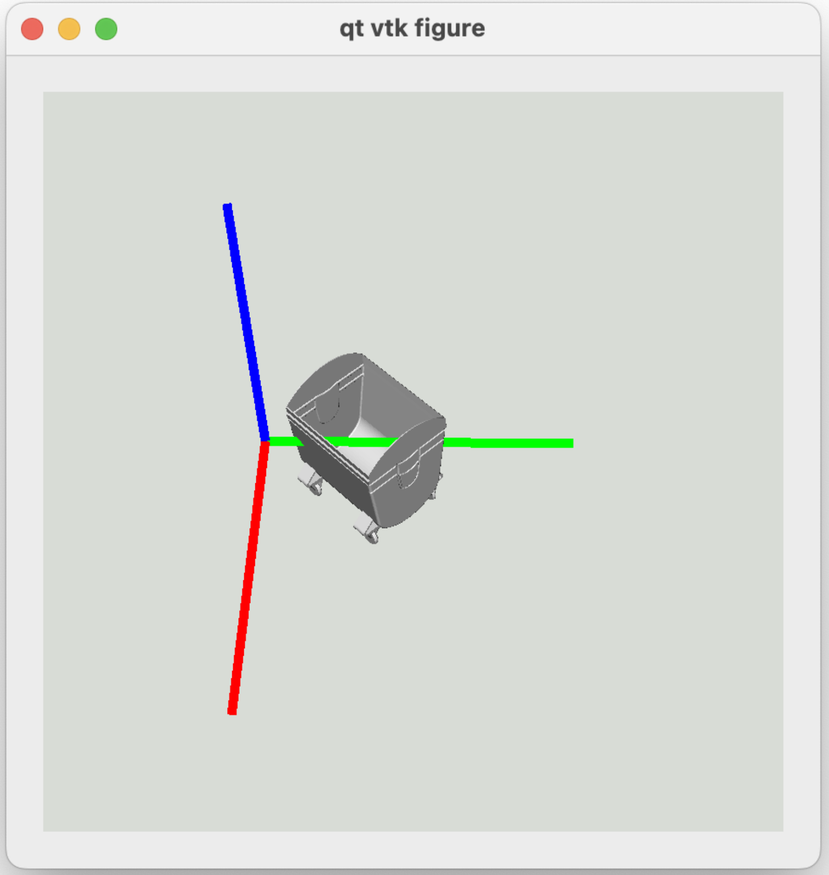
Aplicar 2 transformaciones

```
vpl.QtFigure()
# La 1ª transformación rota 45º en Y
T = numpy.identity(4)
Rz = numpy.identity(4)
Ry = rotate_y(numpy.radians(45))
Rx = numpy.identity(4)
Mt1 = T @ Rz @ Ry @ Rx
# La 2ª transformación traslada 1 metro en X y rota 90º en Y
T[0:3, 3] = [1.0, 0.0, 0.0]
Ry = rotate_y(numpy.radians(90))
Mt2 = T @ Rz @ Ry @ Rx
copia = deepcopy(modelo)
copia.transform(Mt1 @ Mt2)
vpl.mesh_plot(copia)
pinta_ejes()
vpl.show()
```



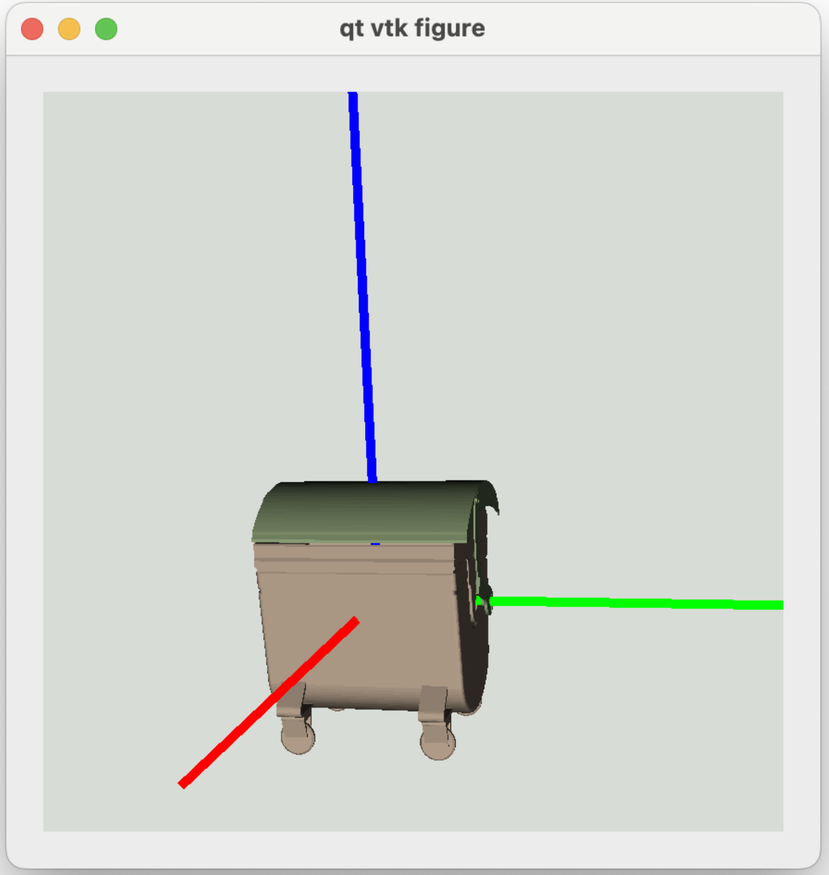
Aplicar 2 transformaciones… ¡diferente!

```
# Aplicamos las mismas 2 transformaciones de antes,
# pero en orden inverso.
copia = deepcopy(modelo)
copia.transform(Mt2 @ Mt1)
vpl.mesh_plot(copia)
pinta_ejes()
vpl.show()
```



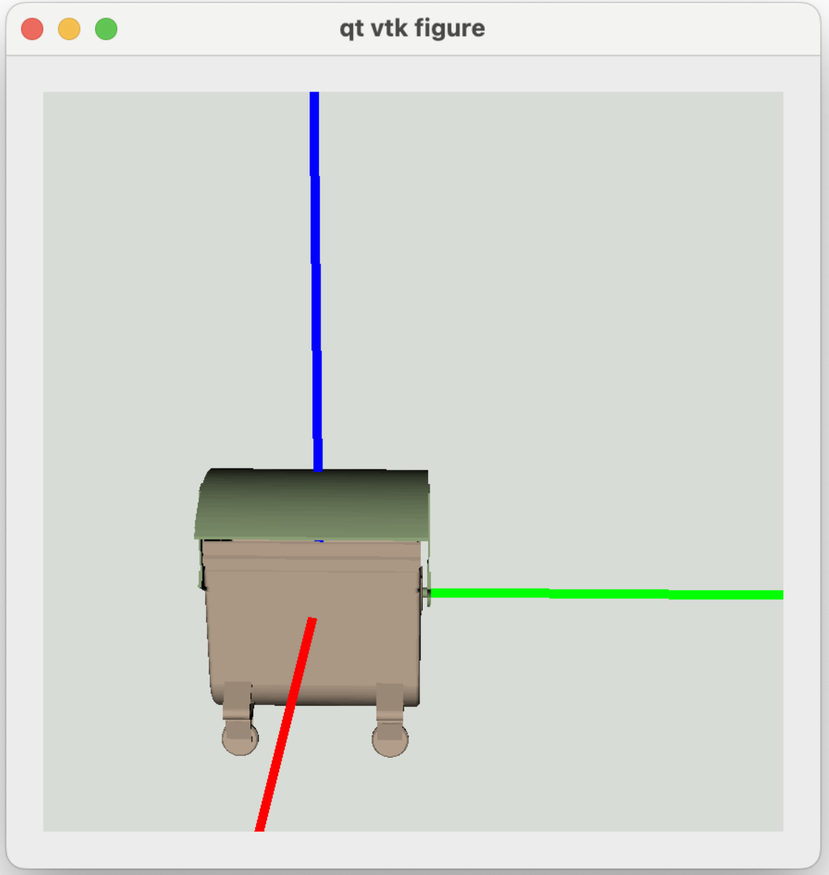
Cargar 2 modelos

```
vpl.QtFigure()
contenedor = mesh.Mesh.from_file("contenedor.stl")
tapa = mesh.Mesh.from_file("tapa.stl")
# Esta vez, ponemos colores a las piezas para distinguirlas mejor.
vpl.mesh_plot(contenedor, color=(0.71, 0.63, 0.55))
vpl.mesh_plot(tapa, color=(0.55, 0.63, 0.47))
pinta_ejes()
vpl.show()
```



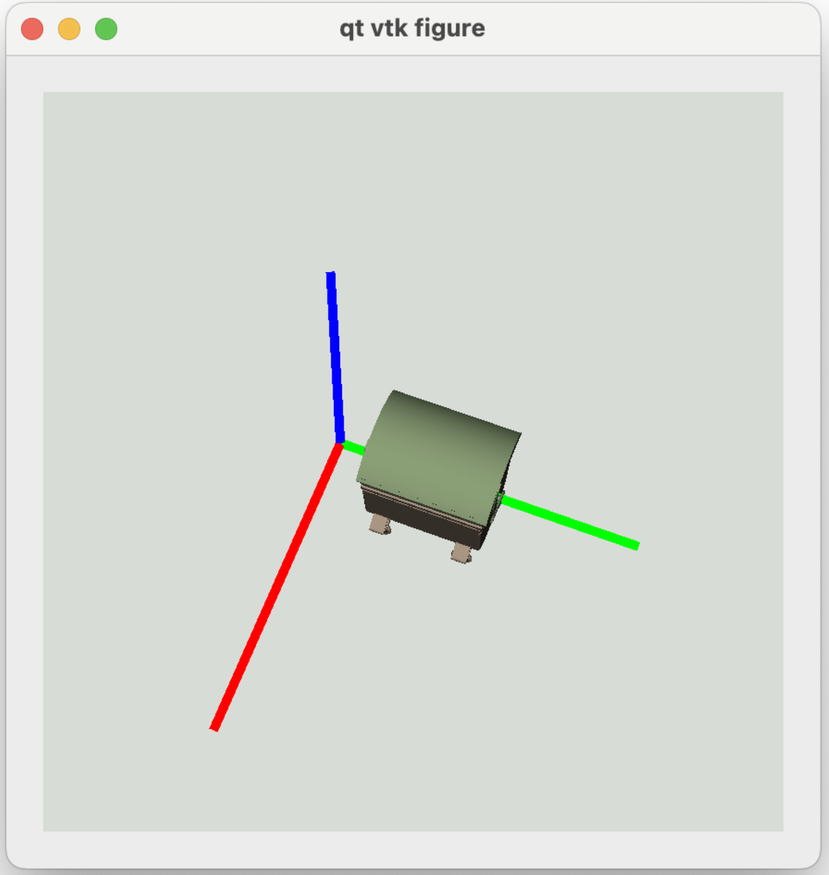
Ensamblar modelos

```
vpl.QtFigure()
# La tapa no está bien colocada.
# Aplicamos una trasformación para ponerla perfectamente encima
# del contenedor.
T = numpy.identity(4)
T[0:3, 3] = [-0.03811, 0.0, 0.0]
Mt_tapa = T
copia_tapa = deepcopy(tapa)
copia_tapa.transform(Mt_tapa)
vpl.mesh_plot(contenedor, color=(0.71, 0.63, 0.55))
vpl.mesh_plot(copia_tapa, color=(0.55, 0.63, 0.47))
pinta_ejes()
vpl.show()
```



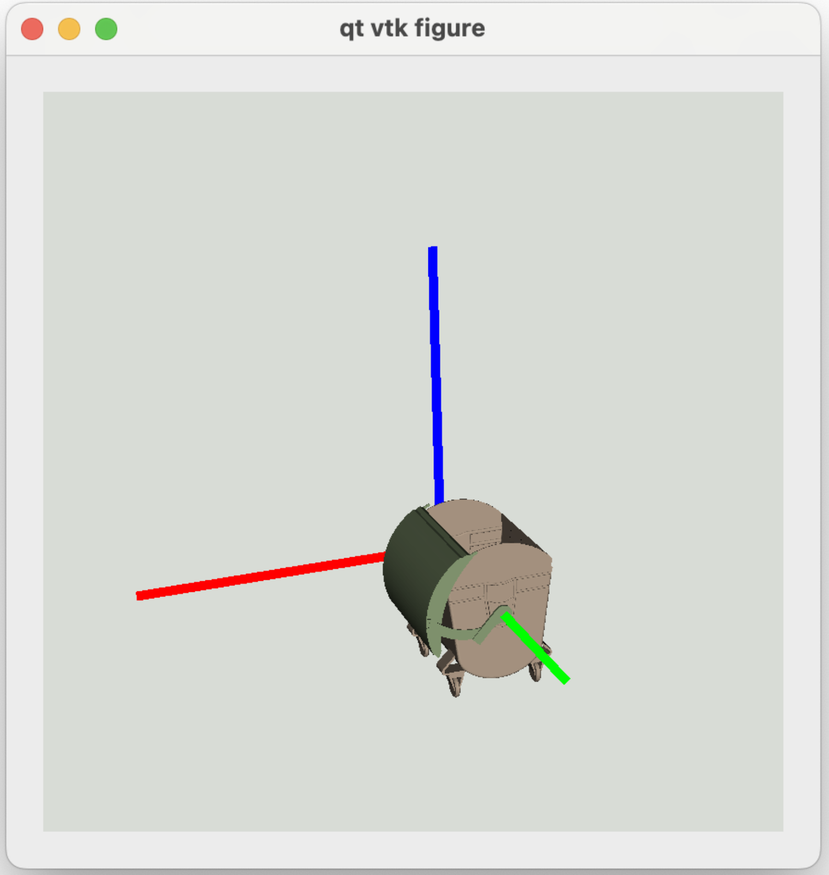
Mover el ensamblaje

```
vpl.QtFigure()
# Desplazamos el contenedor 1 metro en X.
T = numpy.identity(4)
T[0:3, 3] = [1.0, 0.0, 0.0]
Mt_contenedor = T
copia_contenedor = deepcopy(contenedor)
copia_contenedor.transform(Mt_contenedor)
# Desplazamos la tapa también 1 metro en X y después la colocamos
# bien en su sitio encima del contenedor.
copia_tapa = deepcopy(tapa)
copia_tapa.transform(Mt_contenedor @ Mt_tapa)
vpl.mesh_plot(copia_contenedor, color=(0.71, 0.63, 0.55))
vpl.mesh_plot(copia_tapa, color=(0.55, 0.63, 0.47))
pinta_ejes()
vpl.show()
```



Movimiento adicional en una pieza

```
vpl.QtFigure()
copia_contenedor = deepcopy(contenedor)
copia_contenedor.transform(Mt_contenedor)
# No se enseña esta función ;-)
Rx = rotate_x(numpy.radians(70.0))
Mt_tapa_abierta = Rx
copia_tapa = deepcopy(tapa)
# Desplazamos la tapa 1 m. en X,
# la colocamos bien encima del contenedor
# y la abrimos rotándola en X.
copia_tapa.transform(Mt_contenedor @ Mt_tapa @ Mt_tapa_abierta)
vpl.mesh_plot(copia_contenedor, color=(0.71, 0.63, 0.55))
vpl.mesh_plot(copia_tapa, color=(0.55, 0.63, 0.47))
pinta_ejes()
vpl.show()
```



Animación

- En la siguiente diapositiva animamos el movimiento de uno de los ejemplos anteriores.
- Ten en cuenta que este ejemplo solo cambia la posición en el eje X. En otros movimientos puede haber otras traslaciones y / o rotaciones.
- Para este caso particular algunas operaciones podrían simplificarse (por ejemplo, las matrices de transformación podían ser simplemente de traslación), pero se han dejado así por claridad.
- La función **paint()**:
  - Añade las piezas a la ventana.
  - Pinta el contenido de la ventana y continua sin esperar a que el usuario la cierre.
  - Elimina las piezas de la ventana.

Animación

```
def paint(maquina):
    lista = []
    for pieza in maquina:
        lista.append(vpl.mesh_plot(pieza))
    figure = vpl.gcf()
    figure.update()
    figure.show(block=False)
    for pieza in lista:
        figure.remove_plot(pieza)



vpl.QtFigure()
vpl.view(camera_position=(8.0, 4.0, 8.0), focal_point=(-1.0, 1.5, -1.0))
Rx = numpy.identity(4)
Ry = numpy.identity(4)
Rz = numpy.identity(4)
pinta_ejes()
```

```
n_steps = 100
for step in range(n_steps):
    x = step * 2.0 / n_steps
    # Movemos el contenedor
    copia_contenedor = deepcopy(contenedor)
    T = numpy.identity(4)
    T[0:3, 3] = [x, 0.0, 0.0]
    Mt_contenedor = T @ Rz @ Ry @ Rx
    copia_contenedor.transform(Mt_contenedor)
    # Movemos la tapa
    copia_tapa = deepcopy(tapa)
    T = numpy.identity(4)
    T[0:3, 3] = [-0.03811, 0.0, 0.0]
    Mt_tapa = T @ Rz @ Ry @ Rx
    copia_tapa.transform(Mt_contenedor @ Mt_tapa)
    paint([copia_contenedor, copia_tapa])
vpl.mesh_plot(copia_contenedor)
vpl.mesh_plot(copia_tapa)
vpl.show()
```

La traslación en el eje “x” cambia con el número de iteración del bucle:

- Cuando step vale 0 => x = 0
- Cuando step vale 100 => x = 2

Animación