# Informática Gráfica

Juan Carlos Torres

Curso 2024/25

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

**Disclaimer**
You can edit this page to suit your needs. For instance, here we have a no copyright statement, a colophon and some other information. This page is based on the corresponding page of Ken Arroyo Ohori's thesis, with minimal changes.



**Colophon**
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.

# Transformaciones geométricas 3



Las transformaciones geométricas son funciones que modifican las coordenadas de los puntos. Aplicadas a un objeto permiten modificar su posición, orientación o tamaño. En la sección 1.2.3 vimos como definir traslaciones y escalados.

En este tema nos ocuparemos de las rotaciones y de la composición y representación de las transformaciones. Las rotaciones son también transformaciones geométricas básicas, que permiten girar los objetos. El objeto de la parte superior de Figura 3.1 aparece girado en la parte inferior de la Figura.

## 3.1. Rotaciones

Una rotación en 2D gira los puntos en torno al origen de coordenadas un ángulo $\alpha$. La transformación $R_\alpha$ de **rotación** de un punto $\mathbf{p} = (x, y)$ es:

$$R_\alpha(\mathbf{p}) = (x\ \cos(\alpha) - y\ \sin(\alpha),\ x\ \sin(\alpha) + y\ \cos(\alpha))$$

Un giro con valor de $\alpha$ positivo gira en sentido contrario a las agujas del reloj (sentido antihorario o CCW de las siglas en Inglés de Counter Clockwise).

En 3D hay tres posibles rotaciones básicas, una para cada eje (Figura 3.2). La rotación respecto a un eje no modifica el valor de la coordenada correspondiente a ese eje, para las otros dos coordenadas la expresión es como la de una rotación 2D, con los cambios correspondientes de coordenadas y de signo por la dirección de los ejes.

Por ejemplo, para el giro respecto al eje $Y$ es:

$$R^y_\alpha(\mathbf{p}) = (x\ \cos(\alpha) + z\ \sin(\alpha),\ y\ ,\ -x\ \sin(\alpha) + z\ \cos(\alpha))$$

Los ángulos de rotación en 3D se consideran positivos si se rota en sentido antihorario mirando la escena desde el eje de rotación.

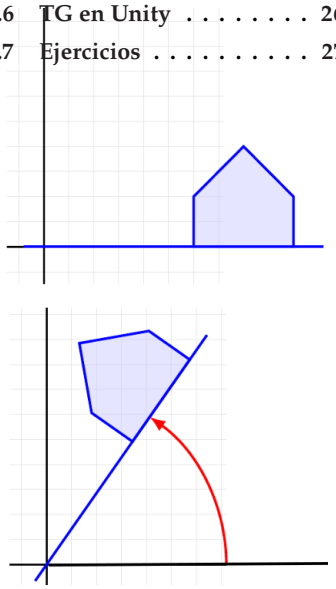

**Figura 3.1:** Rotación en dos dimensiones.

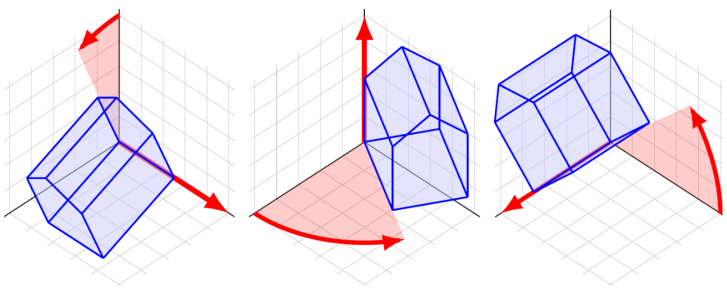

**Figura 3.2:** Rotaciones básicas en tres dimensiones. Izquierda rotación respecto al eje $X$, en el centro respecto al $Y$, derecha respecto al $Z$.

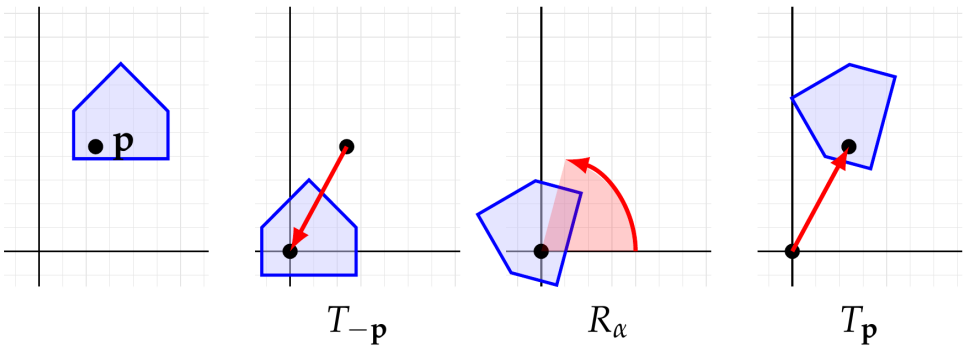


**Figura 3.3:** Rotación 2D respecto a un punto arbitrario. La figura de la izquierda se rota respecto al punto $P$ realizando las siguientes transformaciones: una traslación $T_{-P}$, una rotación $R_\alpha$ y una traslación $T_P$.

## 3.2. Composición de transformaciones

Las transformaciones geométricas se pueden componer. La composición de transformaciones permite realizar la transformaciones complejas usando varias transformaciones simples. Por ejemplo, para realizar una rotación en 2D respecto a un punto diferente del origen de coordenadas podemos trasladar la figura para llevar el centro de giro al origen, rotar respecto al origen y finalmente volver a llevar el centro de giro a su posición original (Figura 3.3):

$$R_\alpha^P(Q) = T_P(R_\alpha(T_{-P}(Q)))$$

La mayor parte de las transformaciones geométricas tienen inversa. La inversa de una transformación $T$ es la transformación $T-1$ que compuesta con $T$ es la identidad. En el ejemplo anterior $T_{-P}$ es la inversa de $T_P$.

Podemos componer transformaciones para realizar giros en 3D respecto a ejes arbitrarios. Para girar sobre un eje que pasa por el origen que no coincide con los ejes del sistema de coordenadas, debemos girar respecto a uno de los ejes del sistema de coordenadas para llevar el eje de giro a uno de los planos del sistema de coordenadas (En la Figura 3.4 se ha girado el eje $e$ respecto al eje $Y$ para llevarlo al plano $XZ$). Seguidamente giramos respecto al eje perpendicular a este plano para llevar el eje de giro hasta uno de los ejes del sistema de coordenadas (en la Figura se gira respecto a $Z$ para llevarlo al eje $X$). Hacemos el giro y aplicamos las transformaciones inversas para devolver el eje de giro a su posición original.

Calcular los ángulos de giro ($\beta$ y $\gamma$ en la Figura 3.4) es simple. Basta con observar que el complementario del ángulo $\beta$ (naranja en Figura 3.5) está en un triángulo rectángulo cuyos catetos son $e_x$ y $e_z$, por tanto[1]:

$$\beta = 90 - atan2(e_x, e_z)$$

En la Figura 3.5, los valores de $e_x$, $e_y$ y $e_z$ están representados como líneas roja, verde y azul respectivamente.

El ángulo $\gamma$ se puede calcular a partir de la longitud de la proyección del vector $e$ en el plano $XZ$ y de $e_y$ como:

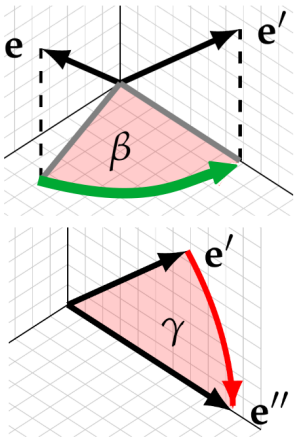


**Figura 3.4:** Rotación respecto a un eje arbitrario. Para girar respecto al eje $e$ hacemos tres rotaciones: respecto a Y,Z y X.

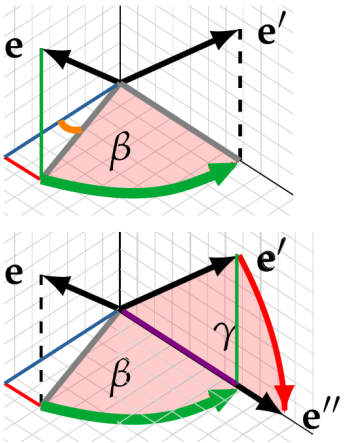


**Figura 3.5:** Cálculo de los ángulos de rotación para girar respecto a un eje arbitrario.

1: $atan2$ devuelve el ángulo cuyo tangente es el cociente de dos números, tomando en cuenta los signos de ambos argumentos para determinar el cuadrante correcto del ángulo.

$$\gamma = atan2(e_y, \sqrt{e_x^2 + e_z^2})$$

La rotación completa sería

$$R_\alpha^e(Q) = (R_{-\beta}^Y(R_{-\gamma}^Z(R_\alpha^X(R_\gamma^Z(R_\beta^Y(Q))))))$$

Si el eje de rotación no pasa por el origen podemos añadir una traslación para mover el eje al origen de coordenadas.

## 3.3. Coordenadas homogéneas

Todas las transformaciones que hemos visto (traslación, escalado y rotación) son transformaciones **afines**, esto es, que transforman líneas rectas en líneas rectas, manteniendo las proporciones entre longitudes de segmentos en dichas rectas.

Los escalados y las rotaciones son transformaciones **lineales**. Una transformación $T$ será lineal si y solo si cumple:

$$T(aP + bQ) \;=\; a\,T(P) \;+\; b\,T(Q)$$

para cualquiera par de puntos $P$ y $Q$, y valores reales $a$ y $b$. Todas las transformaciones lineales son afines, pero no al contrario.

Las transformaciones lineales se pueden representar como matrices. Para cualquier transformación lineal en $\mathbb{E}_3$, existen nueve valores reales $(m_{00}, \ldots, m_{22})$ que determinan la transformación:

$$T(P) = (m_{00}x + m_{01}y + m_{02}z, m_{10}x + m_{11}y + m_{12}z, m_{20}x + m_{21}y + m_{22}z$$

Es decir la transformación está representada por la matriz $M = (m_{ij})$ (3x3), y su aplicación sobre un punto se puede calcular multiplicándola por el punto (escrito como vector columna) [2]:

2: En 2D las matrices serían 2x2

$$P' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = M\,P$$

La utilización de matrices facilita la representación de las trasformaciones, su composición y su aplicación.

Las **traslaciones** son afines pero no son lineales, por lo que no pueden representarse como matrices 3x3. Para poder representar las traslaciones como matrices se utilizan **coordendas homogéneas**.

En coordenadas homogéneas los puntos en $\mathbb{E}_n$ se representan mediante $n + 1$ coordenadas. Es decir, un punto 3D está representado por cuatro coordenadas:

$$P = (x, y, z) = (wz, wy, wz, w)$$

La última componente representa un factor de escala. La tupla $(x, y, z, w)$ representa:

- el punto 3D de coordenadas $(x/w, y/w, z/w)$, si $w \neq 0$.
- la dirección del vector que va desde el origen a $(x, y, z)$, si $w = 0$ (a estas tuplas se les llama puntos en el infinito).

Los puntos y direcciones 2D en el plano cartesiano también pueden representarse usando el espacio de coordenadas homogéneas 2D, que ahora son ternas de la forma $(x, y, w)$, que representan

- el punto del plano de coordenadas $(x/w, y/w)$, si $w \neq 0$.
- la dirección del vector que va desde el origen a $(x, y)$, si $w = 0$.

El espacio de coordenadas homogéneas 2D se puede visualizar de forma semejante a $\mathbb{E}_3$. La Figura 3.6 representa este espacio. El plano verde es el plano euclideo bidimensional en el que cada punto $P$ esta representado por una recta que pasa por el origen del espacio en coordenadas homogéneas.

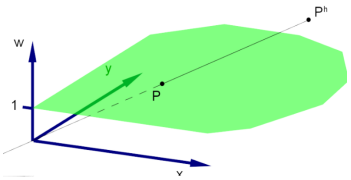


**Figura 3.6:** Cálculo de los ángulos de rotación para girar respecto a un eje arbitrario.

La ventaja de utilizar coordenadas homogéneas es que permite representar todas transformaciones geométricas mediante matrices 4×4, componiendo transformaciones como producto de matrices [3] y aplicarlas multiplicando esta matriz por la matriz columna de coordenadas homogéneas de un punto, que producirá las coordenadas homogéneas del punto transformado.

3: Para transformar un modelo es necesario aplicar la transformación a todos sus puntos, poder componer las matrices de transformación permite aplicar todas las transformaciones realizando una única multiplicación matricial a sus puntos.

Dado un punto 3D de coordenadas $(x, y, z)$ se transforma pasándolo a coordenadas homogéneas como $(x, y, z, 1)$, se aplica la transformación multiplicando por la matriz 4x4 que la representa, obteniendo las componentes $(x', y', z', w')$ que se proyecta al punto $(x'/w', y'/w', z'/w')$.

Cualquier transformación lineal $T$ en coordenadas euclídeas 3D (con matriz $M$) se puede convertir en una transformación lineal en coordenadas homogéneas extendiendo la matriz 3x3 $M = (m_{ij})$ añadiéndole una fila y una columna:

$$\begin{pmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La Figura 3.7 muestra la matriz de una transformación de escalado.

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Figura 3.7:** Matriz de escalado 3D en coordenadas homogéneas, con vector de escalado $(s_x, s_y, s_z)$.

En coordenadas cartesianas, la traslación no puede expresarse como una matriz. Sin embargo, en coordenadas homogéneas, una traslación puede representarse fácilmente con una matriz de transformación, facilitando la combinación de múltiples transformaciones en una sola operación (Figura 3.8).

$$\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Figura 3.8:** Matriz de traslación 3D en coordenadas homogéneas, con vector de traslación$(d_x, d_y, d_z)$.

Además, las coordenadas homogéneas permiten una representación unificada de puntos y vectores (usando el valor cero para $w$).

En transformaciones afines, el valor de la última coordenada ($w$) siempre será 0 o 1, sin embargo, en transformaciones proyectivas, $w$ puede tomar otros valores, lo que permite representar proyecciones y perspectivas.

## 3.4. TG en OpenGL

OpenGL almacena, como parte de su estado, una matriz 4x4 que codifica una transformación geométrica como transformación de modelado **modelview** que se aplica a todos los elementos que se dibujan.

Podemos modificar el valor de la matriz modelview usando las llamadas: $glLoadIdentity()$ para hacer la matriz de modelado igual a la matriz identidad, o usar llamadas para componer una transformación a la de modelado $glScale$, $glRotate$, o $glTranslate$, con la siguiente sintaxis:

```
glRotatef( GLfloat a, GLfloat ex, GLfloat ey, GLfloat ez )
```

Premultiplica la matriz de modelado por una rotación con eje $(ex, ey, ez)$ de $a$ grados.

```
glTranslatef( GLfloat dx, GLfloat dy, GLfloat dz )
```

Premultiplica por una traslación según el vector $(dx, dy, dz)$.

```
glScalef( GLfloat sx, GLfloat sy, GLfloat sz )
```

Premultiplica por un escalado de factor de escala $(sx, sy, sz)$.

Premultiplicar implica que la transformación que se está indicando se aplicará a los objetos antes que las transformaciones que se habían cargado previamente en la matriz de modelado.

El siguiente código dibuja los tres anillos de la figura 3.9.

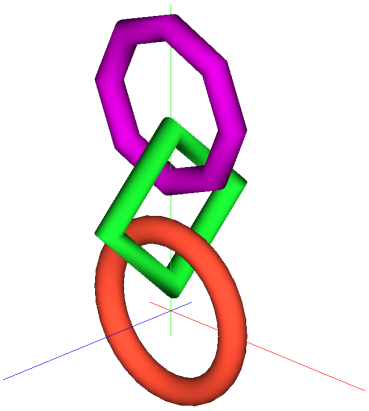

**Figura 3.9:** Imagen generada con el código de dibujo de tres anillos.

**Listing 3.1**: Función de dibujo del modelo de la figura 3.9

```
void Dibuja( void ){
float rojo[4]={1.0,0.3,0.2,1.0}, verde[4]={0.1,1.0,0.2,1.0},morado
    [4]={0.8,0.0,0.8,1.0};

glPushMatrix();
glClearColor(1,1,1,1);
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

dibujar_ejes();
glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,rojo);
glutSolidTorus(0.5,3,24,32); // anillo rojo

glTranslatef(0,4,0);
glRotatef(90,0,1,0);
glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,verde);


glutSolidTorus(0.5,3,24,4); // anillo verde


glTranslatef(0,4,0);
glRotatef(90,0,1,0);
glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,morado);
glutSolidTorus(0.5,3,24,8); // anillo morado

glPopMatrix();
glutSwapBuffers();
}
```

Los anillos se han dibujado con la función $glutSolidTorus(a, b, n, k)$ que dibuja un toro de radio de brazo a, radio del toro b, aproximado

como una malla con n vértices en los brazos y k vértices en el anillo, sobre el plano $XY$ centrado en el origen de coordenadas. Se han usado diferentes colores y valores de k para distinguir los tres anillos.

El primer anillo dibujado es el rojo (linea 10), no se le ha aplicado ninguna transformación, por lo que se dibuja en el origen.

El segundo anillo dibujado es el verde (linea 17), antes de dibujarlo se han añadido a la transformación de modelado una traslación de $(0, 4, 0)$ y una rotación de 90º en el eje $Y$, por lo que el anillo verde se rotará 90º y después se elevará 4 unidades en $Y$[4].

4: Observa que el orden de aplicación es el inverso al orden en que aparecen en el código

Por último, para dibujar el tercer anillo (linea 23) se están añadiendo a la transformación de modelado una segunda traslación y otro giro de 90º, por lo que el tercer anillo se rota 90º, se eleva 4 unidades, se vuelve a rotar 90º y se vuelve a elevar 4 unidades, ya que también está afectado por las transformaciones anteriores. El resultado es que se ha elevado 8 unidades y se ha girado 180º.

Además podemos inicializar la transformación de modelado con la identidad:

```
glLoadIdentity()
```

También hay funciones para apilar y desapilar la transformación de modelado:

```
glPushMatrix();
glPopMatrix();
```

Al llamar a $glPushMatrix$ se guarda el valor de la transformación que se recupera invocando a $glPopMatrix$. Este mecanismo es útil cuando se necesita que una transformación no afecte a los elementos que se van a dibujar mas adelante. Por ejemplo, supongamos que queremos girar el anillo cuadrado verde para que se muestre tal como se ve en la figura 3.10. Será necesario rotar el anillo verde 45 grados, para ello insertamos una llamada a $glRotatef$ en la línea 16 en el siguiente listado. Pero ese giro no debe afectar al anillo morado. Para evitarlo un $glPushMatrix$ antes de la rotación (linea 15) y un $glPopMatrix$ después del dibujo del anillo verde (linea 18).

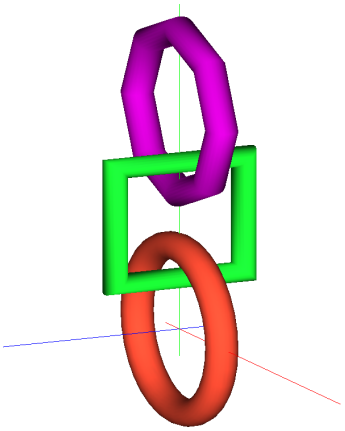

**Figura 3.10:** Modelo con el anillo verde girado.

```
1  void Dibuja( void ){
2  float rojo[4]={1.0,0.3,0.2,1.0}, verde[4]={0.1,1.0,0.2,1.0},morado
       [4]={0.8,0.0,0.8,1.0};
3
4  glPushMatrix();
5  glClearColor(1,1,1,1);
6  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
7
8  dibujar_ejes();
9  glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,rojo);
10 glutSolidTorus(0.5,3,24,32);
11
12 glTranslatef(0,4,0);
13 glRotatef(90,0,1,0);
14 glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,verde);
15 glPushMatrix();
16   glRotatef(45,0,0,1);
17   glutSolidTorus(0.5,3,24,4);
18 glPopMatrix();
```

```
19
20 glTranslatef(0,4,0);
21 glRotatef(90,0,1,0);
22 glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,morado);
23 glutSolidTorus(0.5,3,24,8);
24
25 glPopMatrix();
26 glutSwapBuffers();
27 }
```

## 3.5. Unity

Unity 3D es un motor de desarrollo de aplicaciones gráficas interactivas que ofrece un entorno amigable y versátil. Permite la creación de experiencias tanto en 2D como en 3D y es compatible con múltiples plataformas, incluyendo PC, consolas, móviles y realidad virtual. La programación en Unity se realiza en C#, y cuenta con una comunidad de desarrollo activa y una gran cantidad de recursos y activos disponibles.

Para comenzar a usar Unity, es necesario descargar el Unity Hub, disponible para Linux, Mac y Windows. A través del Unity Hub se gestionan los proyectos y la instalación del editor. Es importante asegurarse de incluir el soporte para Linux Build Support, ya que algunas entregas deberán realizarse en Linux.

Los elementos principales de la interfaz de Unity son (figura 3.12):

**Vista de Escena (Scene View)** : Permite navegar y editar visualmente la escena. Puede mostrar una perspectiva 2D o 3D, dependiendo del tipo de proyecto.

**Ventana de Proyecto (Project Window)** : Muestra los assets disponibles en la librería del proyecto. Aquí aparecen los assets importados.

**Ventana de Jerarquía (Hierarchy Window)** : Representa en texto jerárquico cada objeto en la escena, revelando la estructura de cómo están agrupados.

**Ventana del Inspector (Inspector Window)** : Permite visualizar y editar las propiedades del objeto seleccionado. El contenido varía según el objeto.

**Barra de Herramientas (Toolbar)** : Proporciona acceso a herramientas esenciales para manipular la vista de escena y los objetos, controles de reproducción, y opciones de layout del editor[1].

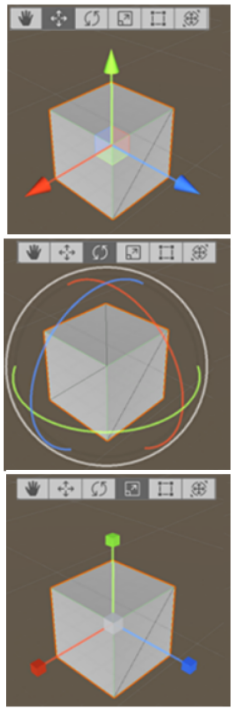

**Figura 3.11:** Manejadores de transformación en Unity. De arriba a abajo: traslación, rotación y escalado.

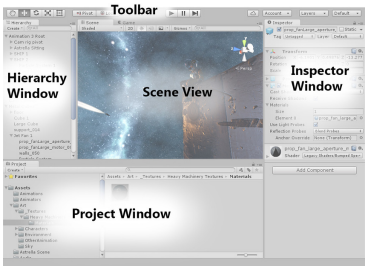


**Figura 3.12:** Elementos principales de la interfaz de Unity.

### 3.5.1. Creación de Objetos

En Unity, la creación de objetos comienza con primitivas simples cuyas propiedades se pueden ajustar en el panel inspector. Los objetos pueden ser transformados geométricamente seleccionándolos y modificando sus parámetros de transformación o utilizando herramientas de edición interactivas. Las transformaciones incluyen posición, orientación y escalado.

### 3.5.2. Cámara y Luces

Cada escena en Unity contiene al menos una cámara y una luz. Al seleccionar una cámara, aparece una ventana con la vista de la cámara y sus propiedades se pueden ajustar en el panel inspector. Las luces también tienen propiedades ajustables en el panel inspector.

### 3.5.3. Interacción

Se pueden asociar scripts a los objetos para definir su comportamiento. Cada script tiene dos métodos principales: $Start$, que se ejecuta al iniciar el sistema, y $Update$, que se ejecuta en cada frame. Por ejemplo, un script puede permitir que el programa responda a eventos de entrada, como presionar una tecla para salir de la aplicación.

## 3.6. TG en Unity

Cada objeto en Unity tiene asociado una transformación, que está formada por un escalado, una rotación y traslación. Los valores de las transformaciones se pueden ver en el panel del inspector (Figura 3.13).

Para transformar un objeto debemos seleccionarlo (en el panel Hierarchy o en la escena) y modificamos los parámetros de sus transformaciones en el panel del inspector.

Alternativamente, podemos modificar sus transformaciones interactivamente usando los manejadores de traslación, rotación o escalado (Figura 3.11).

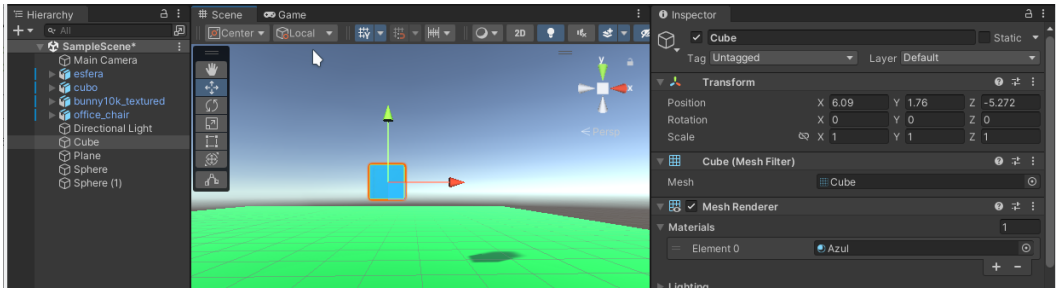


**Figura 3.13:** Las transformaciones aplicadas a un objeto en Unity se muestran en su panel $Inspector$.

## 3.7. Ejercicios

1. Se dispone de una función $caja()$ que crea un paralelepípedo de dimensiones $4x1x2$ como el mostrado en la parte superior de la figura 3.14, escribe el código en OpenGL para dibujar el modelo de la parte inferior de la figura.
2. Utilizando la misma función $caja()$ escribe el código en OpenGL para dibujar el modelo de la figura 3.15.
3. ¿Que dibuja el siguiente código OpenGL?

```
1 for(i=0;i<20;i++){
2     glTranslatef(0.0, 1.0, 0.0);
3     glRotatef(20.0,0.0,1.0,0.0);
4     caja();
5 }
6 glPopMatrix();
```

   En el que $caja()$ crea un paralelepípedo de dimensiones $4x1x2$ como el mostrado en la parte superior de la figura 3.14.
4. ¿Que secuencia de transformaciones se deben usar para realizar un giro respecto a eje vertical que pasa por el punto $(2, 0, -1)$?
5. Calcula las rotaciones que ese necesario realizar para hacer un giro respecto al eje (1,1,1).
6. ¿Como se puede deducir formula de rotación respecto al eje Z?
7. ¿Es igual gira 30º respecto al eje (0,1,0) que hacerlo con al eje (0,-1,0)?
8. La linea 16 del segundo listado de código hace un $glRotatef(45, 0, 0, 1)$ rotando respecto al eje Z ¿Es correcto o debería rotar respecto al eje X?
9. Escribe el código OpenGL necesario para crear el modelo de la figura 3.16 usando la función $glutSolidTorus$. Los toros tienen radio interior 1 y radio exterior 5.
10. Escribe el código OpenGL necesario para crear el modelo de la figura 3.17 usando la función $glutSolidTorus$. Los toros tienen radio interior 1, el radio exterior de los interiores es 8.

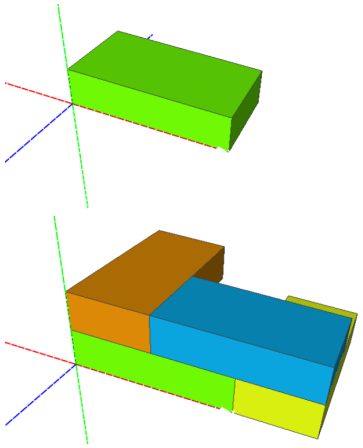

**Figura 3.14:** Ejercicio 4.1.

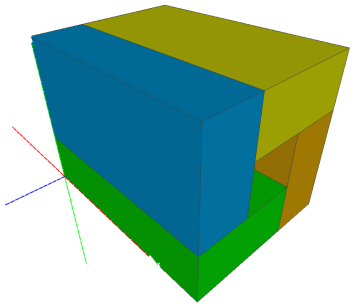

**Figura 3.15:** Ejercicio 4.2.

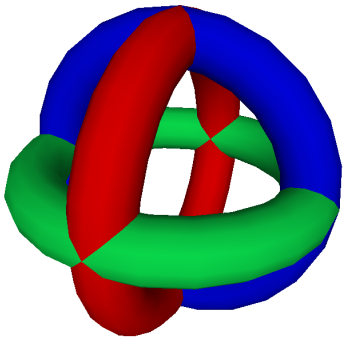

**Figura 3.16:** Ejercicio 4.9.

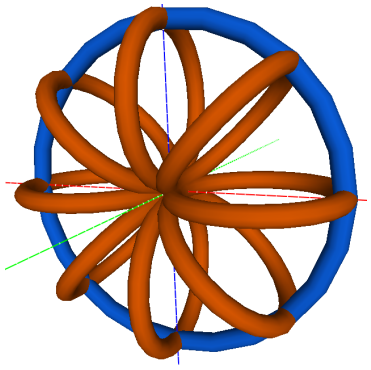

**Figura 3.17:** Ejercicio 4.10.