# 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.

# Grafos de escena 4



Los modelos no suelen componerse de un único objeto (o una única malla de polígonos). Las escenas de de cualquier aplicación real está compuesta de multitud de objetos. Incluso elementos que podemos identificar conceptualmente como un objeto (como por ejemplo un personaje o un vehículo) están integrados por diferentes componentes.

Cada uno de estos objetos y componentes debe tener su geometría definida y debe estar ubicado en la posición correcta en la escena. Aunque es posible definir la geometría de los objetos indicando directamente su posición en la escena, esto no es práctico por varias razones:

- Dificulta la definición de la geometría. Si piensas en la escena de la figura 4.1 tendríamos que definir la posición de cada vértice de cada silla, teniendo en cuenta donde está y que orientación tiene.
- Utiliza información redundante. Los elementos que aparecen repetidos en la escena contienen su información completa.
- Dificulta la edición. Si queremos modificar el modelo de silla tendremos que editar todas las sillas. Incluso, si simplemente queremos desplazar un pupitre tendremos que editar todos los elementos vinculados a él.

Para reducir estos problemas se estructura la escena, de forma que la definición de la geometría sea independiente de la ubicación en la escena, usando transformaciones geométricas, y haciendo que los componentes que están conceptualmente conectados (como la rueda y el chasis de un vehículo) compartan parte de las transformaciones, permitiendo que se puedan transformar de forma solidaria.

En este capítulo veremos como estructurar la escena para minimizar redundancias y facilitar la edición.

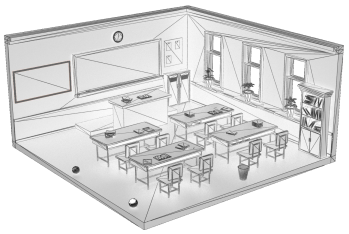

**Figura 4.1:** Escena simple (Philip Storm: Modelo 3D de aula de dibujos animados de Lowpoly).

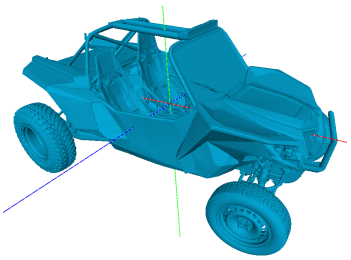

**Figura 4.2:** Modelo 3D de un buggy (Buggy de Artec 3D).

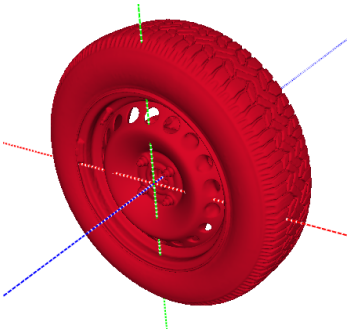

**Figura 4.3:** Rueda delantera del buggy (Buggy de Artec 3D).

## 4.1. Grafos de escena

Para facilitar el diseño y colocación de los objetos de la escena, estos se definen en su propio sistema de coordenadas (normalmente con su centro, o uno de los vértice de su caja envolvente, en el origen). Por ejemplo, para construir el Buggy de la figura 4.2 podemos crear una rueda delantera, colocada con el centro en el origen de coordenadas (Figura 4.10). De esta forma es fácil transformar sus posiciones.

Para colocar los componentes podemos hacer un escalado, $S$ (para cambiar su tamaño y sus proporciones), una rotación, $R$ (para cambiar su orientación) y una traslación, $T$ (para ubicarlo en su posición). Las transformaciones se realizan en este orden, porque es mas fácil pensar los parámetros de cada transformación de este modo. La figura 4.5 muestra el efecto de las tres transformaciones 2D en un cuadrado.

Por ejemplo, si queremos colocar el cubo de imagen izquierda de la figura 4.4 para crear el componente que aparece en la imagen a la derecha de la

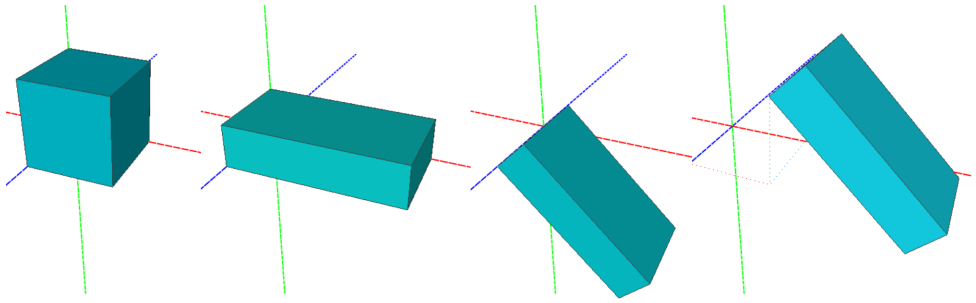

**Figura 4.4:** Instanciación del cubo de dimensiones $1x1x1$ de la izquierda como el paralelepípedo de la derecha. Aplicamos de izquierda a derecha: un escalado $S(2, 0.5, 1)$, una rotación $Rz(45)$ y una traslación $T(1, 1, 0)$

figura. Si aplicamos las transformaciones en el orden escalado, rotación y traslación (es decir $T(R(S(P)))$), podemos calcular el factor de escala mirando el tamaño final del modelo ($2x0.5x1$) y la rotación viendo que aparece inclinado 45º en $Z$, y por último la traslación comprobando que el vértice que está en el origen debe ir al punto $(1, 1, 0)$.

Las figuras 4.6, 4.7 y 4.8 muestran el efecto de aplicar las transformaciones en diferente orden.

No obstante, la transformación no es única. En este ejemplo podríamos haber alargado el cubo hacía el eje $Y$ y después haberlo girado 135º. También podríamos haber realizado alguna de las transformaciones en un orden distinto, por ejemplo trasladar antes de rotar, aunque de es forma es mucho mas difícil calcular la traslación que debemos aplicar, ya que la posición final va a depender también de la rotación.

Por otra parte, hay situaciones en las que se deben realizar las transformaciones necesariamente en otro orden. Por ejemplo, para crear el romboide de la figura 4.9 a partir del cubo anterior, debemos rotar 45 y después escalar en $(2, 0.5, 1)$.

Reutilizar los componentes evita duplicar su información. El modelo de la figura 4.2 tiene 200.000 vértices y ocupa en disco mas de 8MB.

Además, si queremos cambiar el diseño de rueda solo tenemos que cambiar el modelo que estamos instanciando. Y para cambiar su colocación en el modelo bastará con modificar la transformación geométrica.

Podemos usar el mismo proceso para ubicar el buggy en un escenario, instanciando varias veces el modelo utilizando diferentes transformaciones geométricas. De esta forma podemos generar una representación de la escena como Grafo Acíclico Dirigido (DAG), en la que definimos los elementos a un nivel instanciado los componentes del nivel inmediatamente inferior. Esta estructura se suele denominar Grafo de escena en las aplicaciones gráficas.

En el grafo de escena cada nodo representa un componente de la escena y puede contener transformaciones geométricas, geometría y referencias a nodos hijo. Los nodos internos pueden contener información geométrica propia y además instancian nodos a mas bajo nivel. El nodo raíz representa la escena completa.

Las transformaciones definidas en cada nodo afectan a la geometría de ese nodo y a sus nodos subordinados.

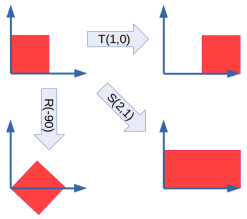


**Figura 4.5:** Transformaciones geométricas 2D.

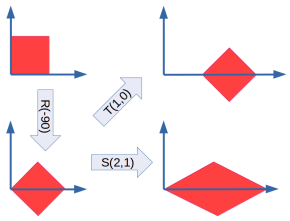


**Figura 4.6:** Efecto de aplicar la rotación en primer lugar.

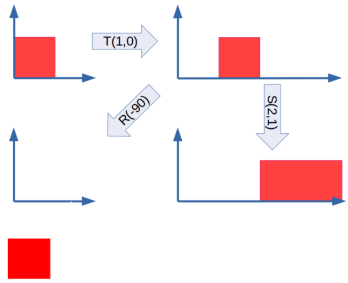


**Figura 4.7:** Efecto de aplicar antes la traslación.

## 4.2. Grafos de escena en OpenGL

OpenGL no guarda el modelo de la escena. Cada vez que se debe dibujar un nuevo frame se ejecuta el código de dibujo, por tanto la estructura de la escena estará reflejada en nuestro código.

Si el código está escrito en *C* (u otro lenguaje no orientado a objetos) la forma mas simple de representar el grafo de escena es creando una función para dibujar cada nodo. Las funciones de los nodos internos deben llamar a las funciones que dibujan los nodos hijos para instanciarlos. De esta forma se puede instanciar un nodo múltiples veces. En la función que dibuja cada nodo hacemos uso de las funciones **glPushMatrix** y **glPopMatrix** para limitar el ámbito de aplicación de las transformaciones geométricas.

Por supuesto podemos implementar el grafo sin hacer que cada nodo se correspondan con una función, usando bloques **glPushMatrix - glPopMatrix**. Cada bloque equivale a bajar un nivel en el grafo de escena, ya que las transformaciones que se incluyan en este bloque no afectarán a las elementos que se dibujen después de él.

A modo de ejemplo el siguiente código dibuja el modelo de figura 4.11 usando la función **glutSolidCube** para dibujar la primitiva cubo.

```
1  void Base(){
2    glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, green );
3    glPushMatrix();
4      glTranslatef(0.0,2.5,0.0);
5      glutSolidCube(5);
6    glPopMatrix();
7  }
8
9  void Tapa(){
10   glPushMatrix();
11     glTranslatef(1.25,0.25,0);
12     glScalef(0.5,0.1,1);
13     glutSolidCube(5);
14   glPopMatrix();
15 }
16
17 void Caja(){
18   Base();
19   glPushMatrix();
20     glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, blue );
21     glTranslatef(-2.5,5,0);
22     glRotatef(30,0,0,1);
23     Tapa();
24   glPopMatrix();
25   glPushMatrix();
26     glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, red);
27     glTranslatef(2.5,5,0);
28     glRotatef(-40,0,0,1);
29     glTranslatef(-2.5,0,0);
30     Tapa();
31   glPopMatrix();
32 }
```

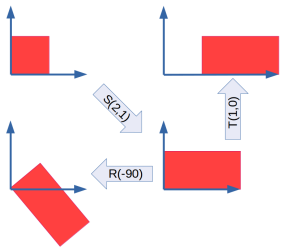


**Figura 4.8:** Efecto de aplicar el escalado en primer lugar.

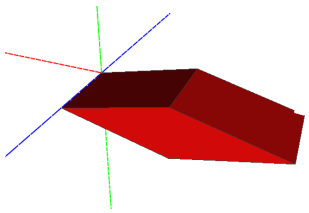

**Figura 4.9:** Romboide generado a partir del cubo de la figura 4.4. Se le ha aplicado una rotado $Rz(45)$ y un escalado $S(2, 0.5, 1)$.

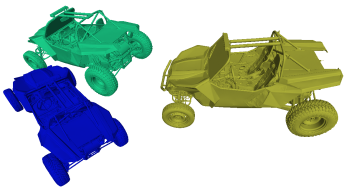

**Figura 4.10:** Escena con varios buggys (Buggy de Artec 3D).

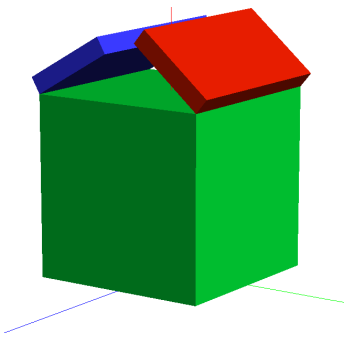

**Figura 4.11:** Ejemplo de modelo simple creado como un modelo jerárquico usando OpenGL.

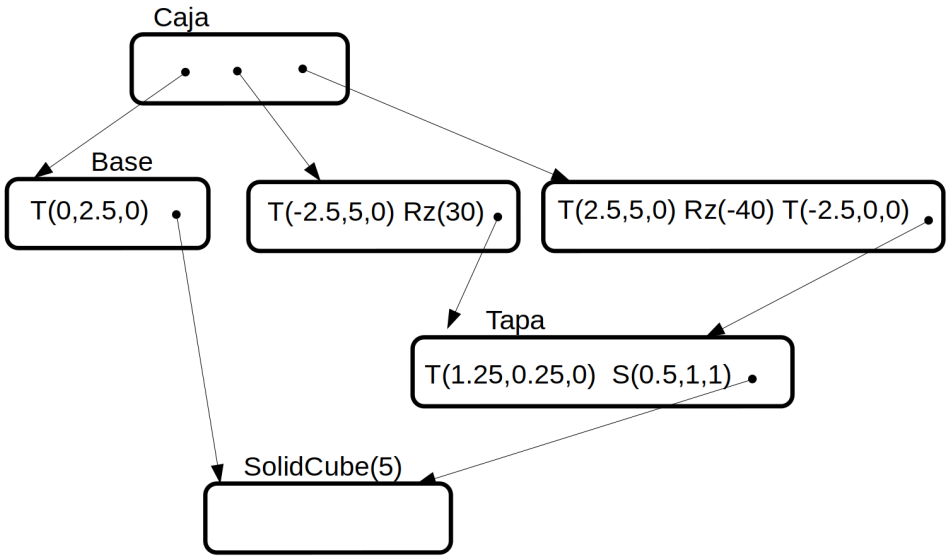


**Figura 4.12:** Grafo de escena del modelo de la figura 4.11.

Las dos tapas (azul y roja) se han dibujado con la función *Tapa* que es llamada dos veces en la función *Caja*.

El grafo de escena correspondiente a este código se muestra en la figura 4.12.

En este modelo se han utilizado escalados, traslaciones y rotaciones. Las transformaciones geométricas se aplican a las normales, y estas se utilizan en el cálculo de iluminación. Si se utilizan escalados es necesario que OpenGL normalice los vectores normales (los ajuste para que tengan módulo 1), esto se consigue utilizando la llamada

```
glEnable(GL_AUTO_NORMAL)
```

## 4.3. Grafos de escena en Unity

En Unity el grafo de escena se muestra en el panel *Hierarchy* situado a la izquierda (ver figura 4.13). En él aparece la jerarquía de nodos que conforman el grafo (y algunos elementos mas como luces y cámaras que veremos en la lección 6).

Los nodos contienen:

- Un objeto. Los objetos se instancian tomándolos de los assests del panel *Project*.
- Su transformación geométrica, dada como un escalado, una rotación y una traslación. Los valores de las transformaciones se pueden consultar y modificar en el panel *Inspector*.
- Opcionalmente pueden tener una lista de nodos hijos. En la figura 4.13) el nodo *Molino* contiene los nodos *A* y *cuerpo*.

Esta forma de asociar las transformaciones a los nodos facilita la representación y edición del grafo, pero impide realizar determinadas composiciones de transformaciones dentro de los nodos. Vimos que, por ejemplo, para girar respecto a un eje que no pasa por el origen era necesario trasladar antes de realizar la rotación. Para permitir que se

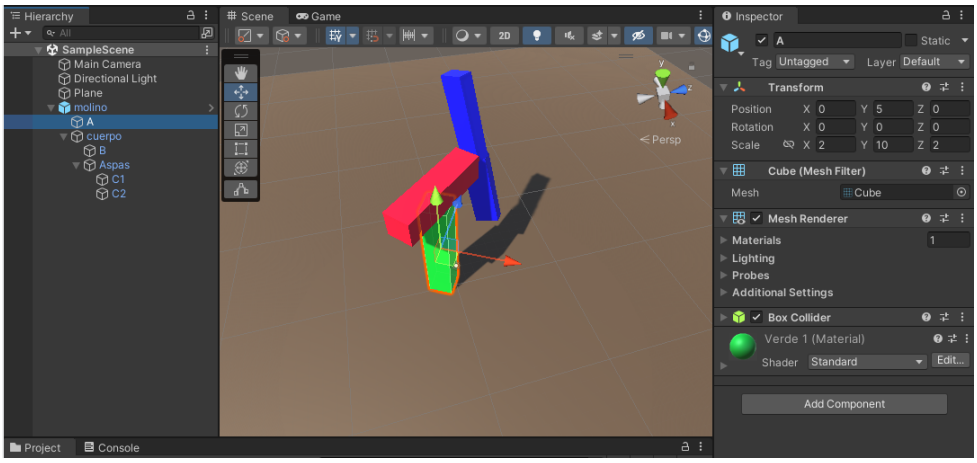


**Figura 4.13:** El grafo de escena de un proyecto de Unity se muestra en el panel izquierdo (Hierarchy).En este ejemplo Molino está formado por los componentes A y Cuerpo; y este último por B y Aspas; que a su vez esta formado por C1 y C2.

puedan realizar estas transformaciones se contempla la posibilidad de incluir nodos sin geometría (*Empty*). De esta forma podemos componer mas transformaciones colocando como padre del nodo un nodo vacío. En la figura 4.13 los nodos *Aspas* y *cuerpo* son nodos vacíos.

En el grafo de escena de la figura 4.12 hay un nodo que no puede crearse en Unity por tener una traslación antes de una rotación. Este nodos se tendría descomponer en dos nodos, en el padre estarían la primera traslación y la rotación y en el hijo la segunda traslación.

Hay otras dos características que ayuda a reducir la complejidad del grafo de escena, a pesar de tener solo un escalado, una rotación y una traslación en cada nodo. Por un lado las rotaciones se indican como tres ángulos de rotación: ángulo de rotación respecto a $X$, rotación respecto a $Y$ y respecto a $Z$, en este orden. Es decir

Por otro lado las rotaciones se pueden realizar respecto al centro geométrico del nodo o respecto a su punto pivote. Podemos colocar el punto pivote en la posición que deseemos colocando un nodo vacío en esa posición y haciendo que el objeto que queremos rotar dependa del nodo vacío.

## 4.4. Diseño de grafos de escena

Diseñar el grafo de escena implica pensar la estructura del modelo y las transformaciones geométricas que contiene. Para su realización se pueden seguir los siguientes pasos:

1. Descomponer la escena recursivamente en componentes mas simples. Este proceso genera el esqueleto del grafo de escena.
2. Decidir las primitivas y transformaciones geométricas que se deben incluir en cada nodo.

Tal como hemos visto anteriormente la estructura de los nodos

### 4.4.1. Descomposición de la escena

Descomponemos el objeto que queremos representar en partes mas simples. Por ejemplo si queremos crear el grafo de escena del molino de viento de la figura 4.13 podemos descomponerlo en el soporte vertical (pieza A, verde en la figura) y el resto del molino. Aplicamos este proceso a los componentes resultantes hasta obtener elementos que podamos construir de forma sencilla. En el ejemplo de la figura 4.13 no es necesario subdividir el pie ya que es un simple paralelepípedo pero si debemos descomponer el resto del molino.

En este proceso de descomposición debemos tener en cuenta que las partes del modelo que se vayan a reutilizar varias veces, como por ejemplo las sillas de una sala, deben ser un componente (que instanciaremos en distintas partes de la escena).

Por otra parte, los componentes que deban estar afectados por la misma transformación geométrica deben estar en la misma rama del grafo. Esto ocurre por ejemplo en los modelos articulados, como el buggy de la figura 4.2, todos los componentes de la rueda deben estar en la misma rama del grafo, para que giren solidariamente al girar esta.

Como resultado de este proceso obtendremos el grado del modelo y un boceto de cada uno de los nodos. El boceto es esencial para aclarar lo que queremos crear en cada nodo y para poder realizar el paso siguiente. La figura 4.14 muestra una posible descomposición del molino de la figura 4.13.

Tal como hemos visto, el contenido de cada nodo puede depender del sistema en el que se vaya a implementar el modelo, por lo que es posible que al incluir las transformaciones geométricas haya que realizar ajustes en el grafo.

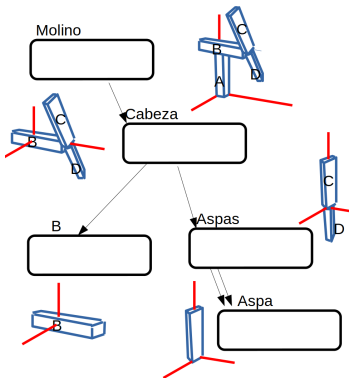


**Figura 4.14:** Ejemplo descomposición del modelo.

### 4.4.2. Diseño de los nodos

El siguiente paso es diseñar el contenido de los nodos, es decir, decidir que primitivas y transformaciones geométricas se van a incluir en cada nodo. Si se va a implementar en OpenGL deberemos también decidir en que orden se aplican. Este proceso se puede realizar de forma independiente para cada nodo, mirando de que componente partimos, lo que representan los nodos inferiores instanciados en el nodo que estamos diseñando, y lo que debe representar este nodo.

Para ello es esencial hacer un boceto de la escena y de cada uno de los nodos (indicando sus dimensiones y ubicándolo en su sistema de coordenadas). Esto es, saber exactamente que geometría debe generar cada nodo.

Por ejemplo, si en el grafo anterior (figura 4.14) partimos de una primitiva para dibujar el Aspa (*primitivaAspa*) como un paralelepípedo con tamaño $1x6x2$ dibujado en el primer octante con un vértice en el origen de coordenadas (figura 4.15), y nuestro nodo *Aspa* debe representar el componente mostrado en la figura 4.16, debemos realizar en el nodo *Aspa* una traslación de $(-0.5, 0, -1)$ antes de instanciar la *primitivaAspa*.

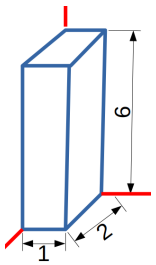


**Figura 4.15:** Primitiva usada para crear las aspas del molino.

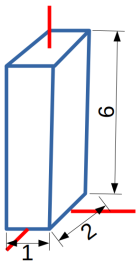


**Figura 4.16:** Boceto del nodo Aspa.

Si en el nodo *Aspas* las dos palas deben estar giradas 20 grados, cada una en un sentido diferente, deberemos rotar el aspa superior -20º respecto a Y, la segunda aspa se debe desplazar hacía abajo 6 unidades (traslación de $(0, -6, 0)$ ) y rotarla 40º respecto al eje Y, tal como se muestra en la figura 4.17.

Recuerda que las transformaciones se aplican de derecha a izquierda, por tanto la interpretación de este nodo es: dibujar un *Aspa* la trasladamos 6 unidades en dirección -Y, la rotamos 40º respecto a Y, dibujamos otra *Aspa* y rotamos el conjunto -20º respecto a Y.

El contenido permitido para los nodos será distinto dependiendo del sistema en el que se vaya a implementar el modelo. El nodo diseñado para las *Aspas* se puede implementar en OpenGl, pero no en Unity. En este último tendríamos que subdividirlo.

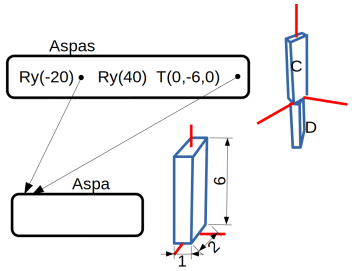


**Figura 4.17:** Diseño del nodo *Aspas*.

### 4.4.3. Implementación del modelo

Al implementar el modelo geométrico pueden aparecer errores, tanto por haberse producido fallos en el diseño como por cometerse errores en la programación. Es mas fácil detectar y corregir estos errores cuando se implementa un modelo pequeño que al probar un sistema grande. Por esta razón el modelo se debe implementar y probar componente a componente.

Esto, que es válido para cualquier tipo de sistema, en nuestro caso se realiza programando el grafo de escena nodo a nodo, comenzando con los nodos mas profundos y comprobando que el modelo generado por cada nodo se corresponden con lo previsto en los bocetos incluidos en el grafo de escena.

De esta forma los posibles errores encontrados estarán circunscritos a un nodo, y será mas fácil corregirlos tanto si son de código como si son de diseño.

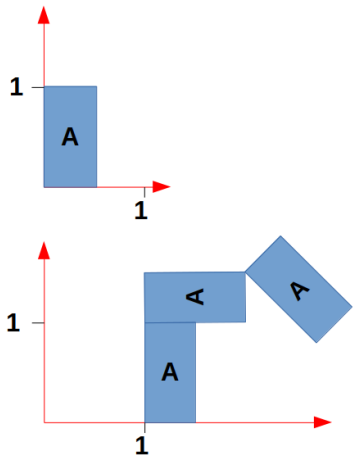


**Figura 4.18:** Calcula las transformaciones geométricas (ejercicio 3).

## 4.5. Ejercicios

1. ¿Que significa y que implica que el grafo de escena sea acíclico?
2. ¿Puede el grafo de escena ser un árbol?
3. Calcula las transformaciones geométricas 2D necesarias para generar cada uno de los elementos de la parte inferior de la figura 4.18 a partir de la primitiva de la parte superior.
4. Dibuja el boceto de cada nodo del grafo de escena de la figura 4.12.
5. Genera el grafo de escena del modelo de derecha de la figura 4.19 usando las primitivas A y B que aparecen a la izquierda.
6. Calcula las transformaciones geométricas necesarias para generar cada uno de los cubos de la parte derecha de la figura 4.20 a partir de la primitiva de la izquierda.
7. Diseña el grafo de escena para construir el objeto de la figura 4.21 usando como componente el objeto creado en el ejercicio 6.
8. Diseña el nodo *Aspas* (figura 4.17) para Unity.
9. Completa el grafo de escena de la figura 4.14.

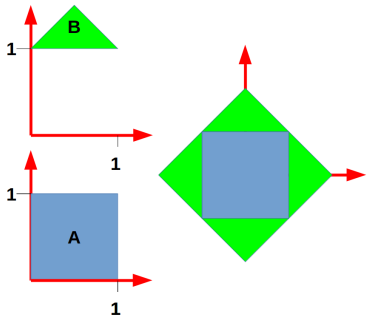


**Figura 4.19:** Genera el grafo de escena (5).

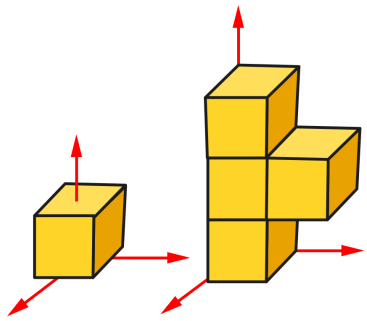

**Figura 4.20:** Calcula las transformaciones geométricas (ejercicio 6).

10. Diseña el grafo de escena del modelo articulado de la figura 4.22, teniendo en cuenta la pieza B puede girar sobre su eje vertical y las C y D giran respecto al horizontal. Supón que todas las piezas son prismas de dimensión 2x2x10 y que se usa como única primitiva un paralelpípedo de dimensiones $(2, 10, 2)$ con un vértice en el origen de coordenadas.
11. Diseña el grafo de escena del modelo articulado de la figura 4.23, teniendo en cuenta que hay dos bisagras en el modelo (entre A y B y entre C y D) y que la pieza C puede desplazarse dentro de B. Usa como única primitiva un paralelpípedo de dimensiones $(1, 10, 1)$ con un vértice en el origen de coordenadas.
12. Diseña el grafo de escena de la escalera de la figura 4.24, teniendo en cuenta que hay una bisagra entre los tramos A y D, que permite plegar A, y que los tramos B y C se pueden desplazar. Los cuatro tramos son iguales con la estructura mostrada en la parte derecha de la figura. Usa como única primitiva un cubo de lado 1 con el centro de su cara inferior en el origen de coordenadas.
13. ¿Como habría que modificar el modelo del ejercicio 12 para que las piezas D y A se cierren de forma simétrica (es decir que la parte inferior de ambas este a la misma altura).
14. Diseña el grafo de escena del modelo articulado que aparece a la derecha de la figura 4.25 formada por tres prismas triangulares unidos por dos bisagras colocadas en catetos alternos del prisma central usando como prinmitiva el prisma de la izquierda.
15. Programa los modelos de los ejercicios 5, 6, 10, 11 y 12 usando OpenGL.
16. Adapta los grafos y crea los modelos de los ejercicios 5, 6, 10, 11 y 12 Unity 3D.

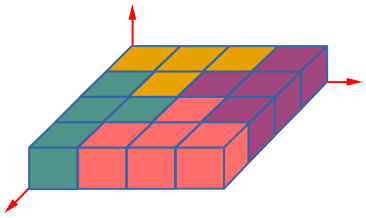

**Figura 4.21:** Modelo generado usando como componente el modelo de la figura 4.20 (ejercicio 7).

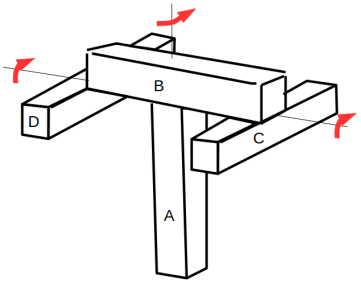


**Figura 4.22:** Modelo articulado del ejercicio 10).

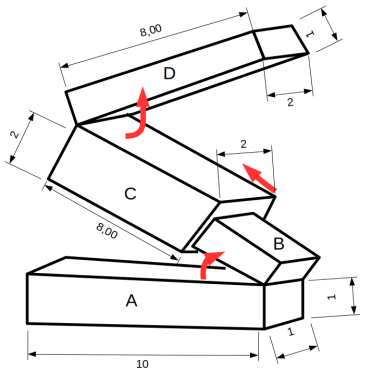


**Figura 4.23:** Modelo articulado del ejercicio 11).

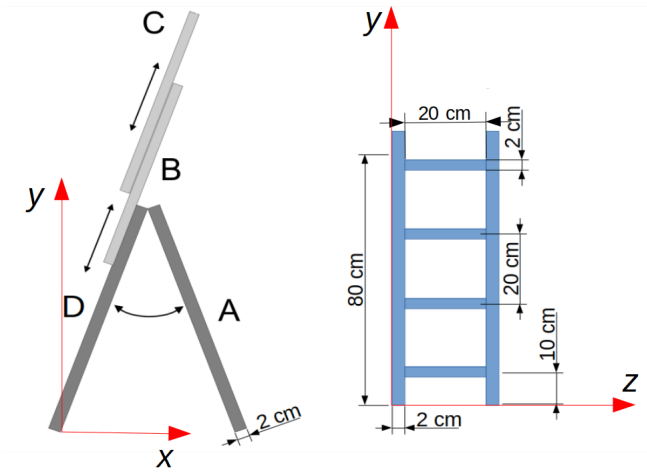


**Figura 4.24:** Escalera del ejercicio 12.

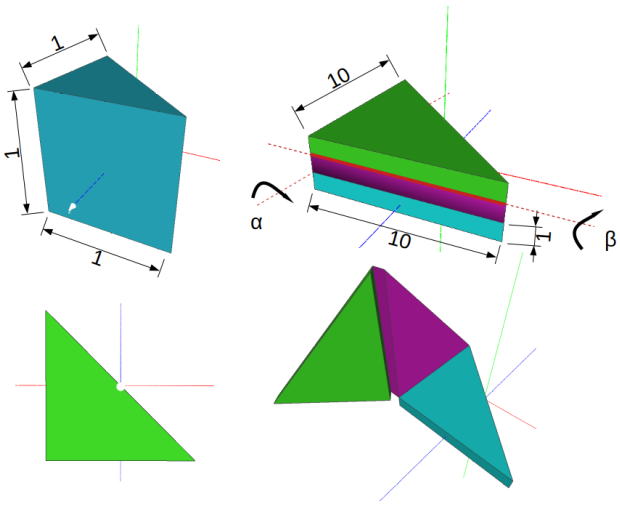


**Figura 4.25:** Componentes (izquierda) y modelo (derecha) del ejercicio 14).