Diseño asistido por ordenador

4ª Curso Ingeniería Informática

J.C. Torres
Dpt. Lenguajes y Sistemas Informáticos
ETS. Ingeniería Informática
Universidad de Granada

TEMA 2. Modelado geométrico

Un **modelo** es una representación de algunas de las características de una entidad concreta o abstracta" [Fole90,pp,286].

Un **modelo geométrico** describe componentes con propiedades geométricas inherentes. Entre sus características destacan: su estructura espacial, la conectividad entre elementos, y las propiedades asociadas a componentes espaciales. [Newm81, Cap.9, pp.111-114] [Sproull Cap.12]

Este tema aborda el estudio de la estructura del modelo geométrico y su funcionalidad.

Indice

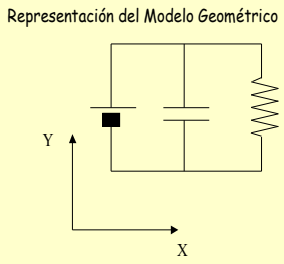
2.1 El modelo geométrico de un sistema CAD

El modelo geométrico contiene toda la información necesaria para representar el objeto que se está diseñando, esto es, para realizar todas las operaciones requeridas sobre el modelo: editarlo, visualizarlo, realizar cálculos y simulaciones.

En esencia, el modelo geométrico es simplemente un conjunto de datos referentes a la geometría, estructura y propiedades del objeto (ver figura 2.1). Los datos que contenga dependerán de la naturaleza del objeto a diseñar. Esto es, será distinta la estructura de un modelo geométrico para representar un circuito eléctrico de la usada para representar el fuselaje de un avión. Además, a la hora de diseñar un modelo se debe de tener en cuenta que la estructura deberá servir para realizar determinadas operaciones antes mencionadas: edición, visualización, cálculo de propiedades, etc.

Ejemplo 2.1 [Sproull 1986, pp.312-314]

Para representar el esquema 2D de un circuito se podría utilizar una secuencia de líneas, que se pueden almacenar en la siguiente estructura de datos:



Modelo	
(X1,Y1)	**(X2,Y2)**
(1,1)	(3,1)
(3,1)	(3,3)
(3,2)	(1,2)
(1.7,2)	(2,2)
(1.7,2.2)	(2.3,2.2)
(2.3,2.2)	

Esta estructura permite realizar la **visualización** del modelo. Esta se puede hacer de forma simple usando el siguiente procedimiento:

```
int i;
// Model es un array de registros con la estructura mostrada arriba
// max es el numero de elementos en el modelo
for(i=0; i<max;++i){
        setcolor(Model[i].Color);
        drawLine(Model[i].X1,Model[i].Y1,Model[i].X2,Model[i].Y2);
        }
```

Sin embargo la edición no sería tan simple. De entrada es identificad los componentes del modelo (líneas) para poder borrarlos o modificarlos (con este fin se puede añadir un identificador al esquema anterior). Pero esto no es suficiente. Trabajando de este modo, las operaciones se realizarán a nivel de líneas. Es decir, no será posible, por ejemplo, "mover una resistencia". El no contemplar la estructura natural del objeto a diseñar en el modelo puede hacer el sistema muy dificil o imposible manejar.

Sería más complejo, aún, realizar otras operaciones, tales como el cálculo la corriente en un componente, ya que el "concepto" de componente no está en el modelo, y debería de reconocerse a partir de la información geométrica. Hacer este calculo requeriría información de conexiones entre líneas, que no aparecen en el modelo. Esta información se debería de obtener, en este modelo, comparando vértices.

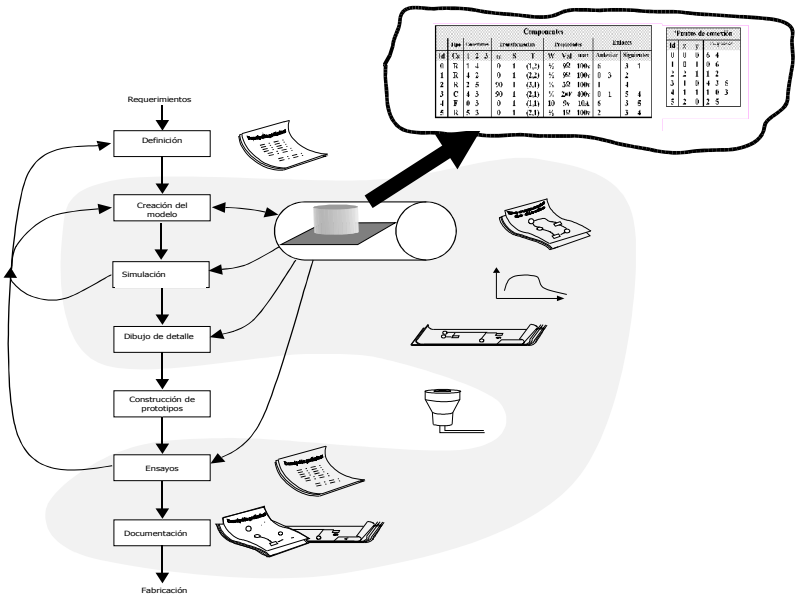


Figura 2.1 Modelo geométrico en el proceso de diseño

Normalmente se necesitan estructuras más complejas para el modelo geométrico, para almacenar la estructura natural del elemento a diseñar, para facilitar la interacción y para mantener las dependencias entre componentes que lo forman, entendiendo por dependencias las relaciones de repetición, conexión, construcción.

Un **modelo geométrico** describe componentes con propiedades geométricas inherentes. Entre sus características destacan: su estructura espacial, la conectividad entre elementos, y las propiedades asociadas a componentes espaciales.

En el proceso de diseño se utilizan representaciones gráficas de los objetos a diseñar, sobre las que trabaja el ingeniero. En muchos casos estas representaciones son imágenes sintéticas del objeto a diseñar (una pieza mecánica, una botella, la carrocería de un coche). En estos casos el modelo geométrico debe describir la geometría del objeto de la forma más precisa posible. No obstante, en otras situaciones, la información sobre la que trabaja el ingeniero es un esquema del objeto (un circuito eleéctrico, la planta de un edificio). En estos casos la información contenida en el modelo debe permitir generar el esquema, pero la geometría del esquema en si (por donde pasa linea que representa la conexión entre dos componentes) no es relevante. Esta situación, y algunas otras que veremos más adelante, hace que en determinadas ocasiones sea preferible que el modelo geométrico, o parte de este, se genere dinámicamente a partir del código. En este sentido hablaremos de modelos representados preceduralmente o mediante estructuras de datos.

2.1.1 Representación procedural.

Independientemente de la estructura lógica dada al modelo, la información geométrica de este puede residir en estructuras de **datos o en código**. Si bien es difícil almacenar toda la estructura en código (no habría posibilidad de realizar modificaciones), puede colocarse en éste una gran parte de la información geométrica [Newm81, Cap.10, pp.128-136]. Esta información puede ser de dos tipos:

1. Información *gráfica no relevante* para el modelo. P.e. el recorrido de un cable de conexión en un esquema puede generarse automáticamente, a partir de las coordenadas de las conexiones.
2. *Información fija.* P.e. la definición de un símbolo, o de un componente complejo de estructura predefinida, como una ventana.

El principal inconveniente de esta estrategia es que limita la capacidad de interacción, pero, por contra, hace más flexible el dibujo, permitiendo que un mismo objeto puede tener varias representaciones.

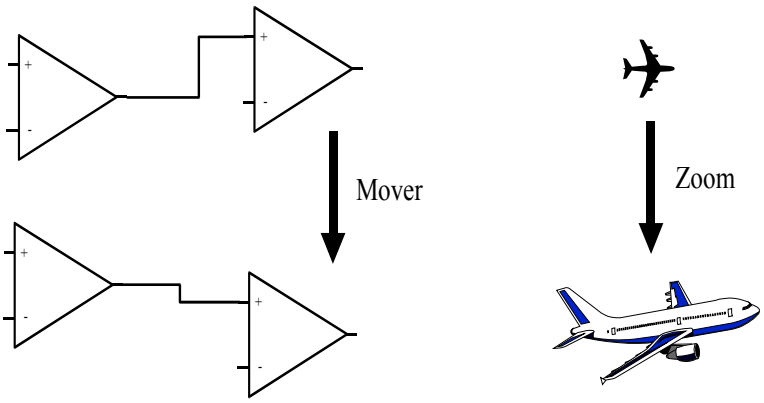


Figura 2. 2 *Ejemplo de modificaciones en modelos procedurales*

2.1.2 Utilización de estructuras de datos.

En el caso opuesto toda la información gráfica puede estar en una estructura de datos, que mantenga la estructura del dibujo. En este caso la visualización consistirá en interpretar la estructura según un determinado algoritmo de recorrido. La edición se realizará directamente sobre la estructura de datos [Salmon pp.304] [Newman pp.137].

2.2 Instanciación

Con frecuencia, determinados símbolos aparecen reiteradamente en el modelo, especialmente en esquemas. Pensemos, por ejemplo, en los **símbolos** que representan componentes en un circuito electrónico, o en el esquema de la instalación eléctrica de una vivienda. En estos casos es útil almacenar la definición geométrica del símbolo una única vez, referenciandola cada vez que se *coloque, o instancie,* éste en el esquema. Este proceso es semejante al que utiliza un delineante cuando copia el símbolo usando una plantilla de dibujo.

En el modelo geométrico se puede seguir este mismo esquema [Newm81, Cap.9,pp.117-123]. El modelo geométrico estaría formado por un lado por la definición de los símbolos, y por otro, por la información de donde se colocan estos en un diseño concreto. Posibilitando que se haga una única definición de símbolo y se utilicen multiples instancias del mismo. Desde el punto de vista del usuario, los símbolos son elementos prediseñados que puede seleccionar, por ejemplo, mediante un menú. En algunos casos el conjunto de símbolos puede no ser cerrado, pudiendo ampliarse con la incorporación de nuevos componentes.

Antes de estudiar como incorporar símbolos en el modelo geométrico debemos entender como se especifica su colocación en el diseño.

Ejemplo 2.2. . Proceso de colocación de un símbolo

En primer lugar, el usuario selecciona el símbolo a colocar (ver figura 2.3 (2)).

A partir de ese momento el dibujo del símbolo seleccionado sigue al cursor (está atado a él).

El usuario mueve el cursor arrastrando al símbolo hasta colocarlo en la posición deseada.

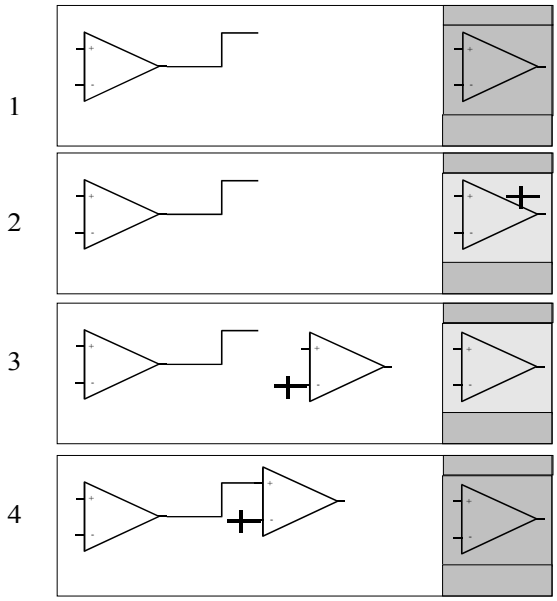


Figura 2.3. Proceso de colocación de un símbolo.

Si el sistema CAD hace uso de mecanismos de instanciación deberá contener, como indicábamos anteriormente, la definición de la geometría de los símbolos. Al sistema de coordenadas en el que se definen los símbolos se le denomina **coordenadas maestras** (MC de "*master coordinates*").

El proceso de colocación de los símbolos en el diseño implica conocer la ubicación, orientación y tamaño que tendrá cada instancia. Esta información se representa como una transformación geométrica que escala, rota y traslada el símbolo a su posición en el diseño; o si lo preferimos, convierte este del sistema de coordenadas maestras, en que esta definido, al sistema de coordenadas utilizado en el modelo, que se suele denominar sistema de **coordenadas del mundo** (WC de "*world coordinates*").

Habitualmente se realiza en primer lugar el escalado, seguido de la rotación, y por último la traslación (ver figura 2.4). Esto permite calcular de forma simple los parámetros de las transformaciones a partir de la posición donde se quiere colocar el símbolo, de su orientación y su tamaño final, sabiendo la colocación inicial del símbolo en coordenadas maestras.

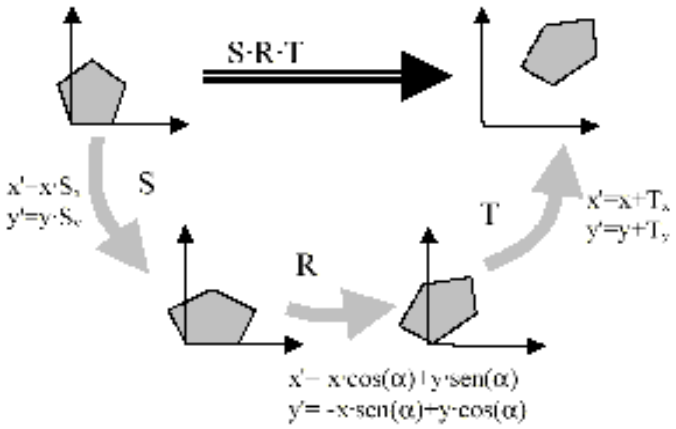


Figura 2.4. Transformación de un símbolo

Fundamentos 2.1. Transformaciones geométricas

Un punto se identifica por sus coordenadas, que están referidas a un sistema de coordenadas conocido. El sistema de coordenadas utilizado habitualmente es el cartesiano, en el que los ejes son perpendiculares. Especificar un sistema de coordenadas implica determinar donde se encuentra su origen y cual es la dirección de sus ejes. Lógicamente si cambiamos el sistema de coordenadas cambiará el valor de las coordenadas de los puntos. Las coordenadas de un punto son la tupla de distancias de sus proyecciones sobre los ejes al origen. En un espacio tridimensional los puntos se representan mediante una terna (x,y,z).

A los puntos, y por extensión, a los elementos geométricos se les pueden aplicar **transformaciones geométricas,** para modificar, por ejemplo, su posición, orientación o tamaño. Las transformaciones geométricas más utilizadas son:

Escalado, que modifica el tamaño de un elemento. Matemáticamente el resultado de aplicar un escalado con factores S_x,S_y,S_z a un punto (x,y,z) es:

$$(x',y',z') = S_{Sx,Sy,Sz}(x,y,z) = (x \cdot S_x, y \cdot S_y, z \cdot S_z)$$

Rotación, que modifica la orientación del elemento En dos dimensiones una rotación se especifica por el centro de rotación (que es un punto) y el ángulo rotado (se suelen considerar positivos los ángulos medidos en sentido antihorario. La ecuación de una rotación respecto al origen de ángulo á es:

$$(x',y') = R_{á}(x,y) = (x \cdot \cos(á) - y \cdot \sin(á), x \cdot \sin(á) + y \cdot \cos(á))$$

En 3D los objetos se rotan respecto a un eje. Si el eje pasa por el origen de coordenadas el eje se puede indicar simplemente como un vector.

Traslación. Una traslación desplaza al elemento al que se le aplica. La traslación se especificamente simplemente por el vector que se añade a los puntos:

$$(x',y',z') = T_{(Tx,Ty,Tz)}(x,y,z) = (x + T_x, y + T_y, z + T_z)$$

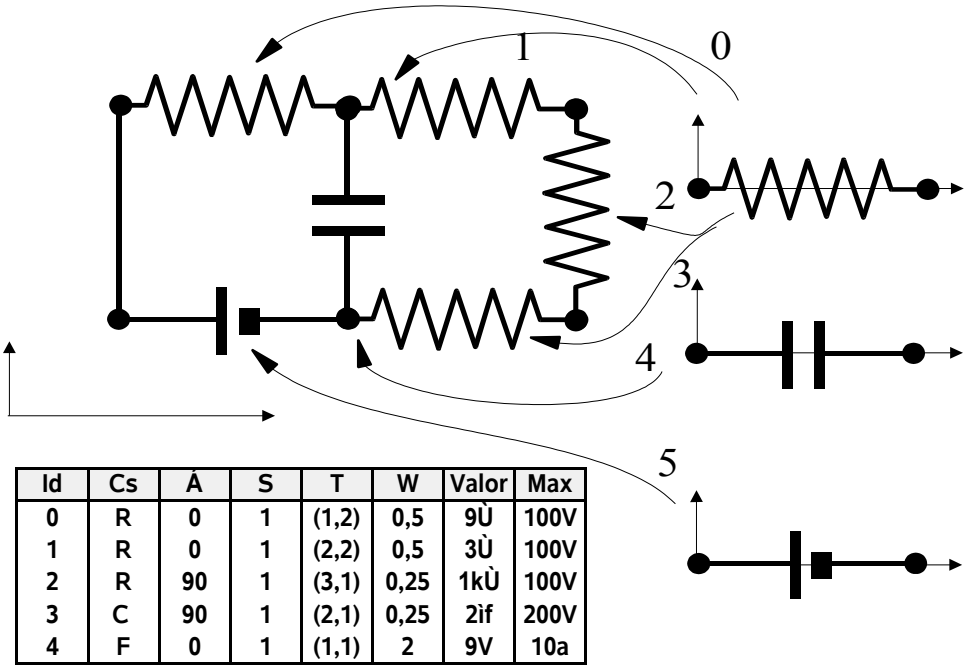
De la misma forma en que cambiamos las coordenadas de los objetos aplicandoles transformaciones geométricas, podemos cambiar el sistema de coodenadas; esto es, modificamos nuestras sistema de referencia, con lo que las coodenadas de todos los elementos cambiaran de valor. El efecto de aplicar una transformación al sistema de coordenadas es el inverso al obtenido si se aplica a los objetos. Esta relación nos permite ver las transformaciones aplicadas a los objetos como cambios en el sistema de coordenadas.

El modelo geométrico contendrá una lista de los símbolos utilizados, indicando su tipo junto con información suficiente para calcular la transformación que se debe aplicar. Esta información puede ser la propia transformación geométrica expresada como una matriz (ver recuadro fundamentos 2.2), o los parámetros que definen estas (factores de escala, ángulos de rotación y vector de traslación), o incluso la posición en que deben transformarse puntos predefinidos del símbolo. La elección de la información a utilizar deberá hacerse teniendo en cuenta los requisitos de eficiencia de cada uno de los procesos a realizar con el modelo y del tipo de simulaciones a realizar. En algunos casos, habrá que optar por incluir información redundante.

Al dibujar los símbolos se debe aplicar la transformación de instanciación a cada una de las instancias. Esto es fácil de implementar si se dispone de una librería gráfica. El código de

Nótese que las transformaciones se aplican comenzando por la más cercana al elemento, esto es en sentido ascendente en el texto. Debe tenerse en cuenta que las transformaciones geométricas no conmutan. Esto es el resultado depende del orden de aplicación.

La representación de los símbolos podrá ser procedural, como se indicó en la sección anterior. En este caso, el procedimiento de dibujo de símbolos se limitará a seleccionar el símbolo a dibujar:



Id	Cs	Á	S	T	W	Valor	Max
0	R	0	1	(1,2)	0,5	9Ù	100V
1	R	0	1	(2,2)	0,5	3Ù	100V
2	R	90	1	(3,1)	0,25	1kÙ	100V
3	C	90	1	(2,1)	0,25	2ìf	200V
4	F	0	1	(1,1)	2	9V	10a

Figura 2.5. Utilización de símbolos en el diseño de un circuito.

Ejemplo 2.3. Utilización de símbolos en el diseño de circuitos.

La utilización de símbolos es una buena opción en un sistema CAD para el diseño de circuitos. En este sistema los distintos componentes de un mismo tipo comparten la misma representación gráfica.

La figura 2.5 muestra el proceso de instanciación en el esquema de un circuito simple, y el contenido que tendría el modelo geométrico para almacenar las instancias de los símbolos (además de esta información se deberán almacenar las conexiones entre componentes, volveremos sobre este punto más adelante).

Para cada instancia el modelo contiene un identificador único, que nos permitirá, entre otras cosas, identificar al componente a la hora de editarlo, el tipo de componente (o lo que es lo mismo, el tipo de símbolo), la transformación geométrica y los parámetros físicos del componente.

En este caso la transformación se ha expresado usando los parámetros de escala, rotación y traslación, asumiendo que el escalado es el mismo en las dos direcciones.

Además de facilitar el dibujo, la utilización de símbolos en el modelo nos permite identificar los componentes que forman el objeto que se está diseñando, permitiendonos, editar el modelo de un modo más natural y asociar parámentros a los componentes. Esto es vital para la realización de simulaciones y, tambien para generar la documentación del modelo.

Alternativamente, el símbolo se puede almacenar en una estructura de datos. En este caso, el código de dibujo del símbolo se limitará a interpretar esta estructura

```
void DrawSymbol (int tipo) {
        glBegin( GL_LINES);
        for(i=0;i<sim[i].N;++i)
                glLine(sim[i].x1,sim[i].y1,sim[i].x2,sim[i].y2);
        glEnd();
```

Fundamentos 2.2 Representación de transformaciones geométricas

Las transformaciones geométricas se representan mediante matrices cuadradas de dimensión mayor en uno a la del espacio en el que se aplican. Esto permite representar con la misma notación todas las transformaciones geométricas, incluyendo las traslaciones y las transformaciones de perspectiva, que no son transformaciones afines. Para transformar los puntos, en un espacio de dimensión n, se pasan estos al espacio de dimensión n+1 (que se suele denominar espacio en coordenadas homogéneas), añadiendo la última componente (denominada homogénea) con valor 1. Seguidamente se multiplica el punto por la matriz, y finalmente se proyecta el resultado en el espacio de dimensión n, dividiendo todas las coordenadas por el valor resultante de la componente homogénea, que no tiene por que ser uno. Esquematicamente, en 3D, el proceso es el siguiene:

1- Pasar a coodenadas homogeneas: (x,y,z) -> (x,y,z,1)

2- Aplicar transformación:

$$(x', y', z', w') = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

3- Proyectar al espacio 3D: (x″,y″,z″) = (x'/w',y'/w',z'/w')

Normalmente no es necesario modificar las componentes de la matriz de transformación directamente. Cualquier librería gráfica calcula internamente las matrices de transformación a partir de parámetros, como el eje y ángulo de giro.

Trabajar de esta forma permite, entre otras cosas concatenar las transformaciones geométricas. Esto es, en el caso de que a un conjunto de puntos se le deba aplicar una secuencia de transformaciones, se puede multiplicar previamente las matrices, aplicando la transformación resultante a los puntos. De este modo se aplica una única transformación a cada punto. Esta estrategia es ampliamente usada por los sistemas gráficos, componiendo incluso la transformaciones de modelado con las de visualización.

La edición del modelo se puede realizar a nivel de componentes (instancias de símbolos). Para ello, el sistema debe incluir mecanismos de selección de instancias. Es fácil implementar procedimientos que, una vez seleccionada una instancia, la borren, modifiquen sus propiedades fisicas o su transformación geométrica. La modificación de la transformación debe realizarse usando métodos interactivos.

Para seleccionar las instancias se deben usar métodos interativos. Un procedimiento no interactivo simple es que el usuario indique el identificador del símbolo. Obviamente este método es inviable en aplicaciones prácticas. Una alternativa simple, para modelos basados en símbolos, es almacenar la caja englobante de cada símbolo. La **caja englobante** es el menor rectángulo, alineado con los ejes del sistema de coordenadas, que contiene al símbolo. La caja englobante se calcula simplemente obteniendo el mínimo y máximo de las coordenadas de los vértices del símbolo. Más adelante veremos métodos generales de selección.

La *transformación de visualización* se aplica a todos los elementos. Cuando se procesa un símbolo se puede concatenar ésta con su transformación de instanciación. Al terminar de dibujar el símbolo se restablece la matriz de transformación original. Hay que prestar atención al orden en las concatenaciones, ya que el producto de transformaciones no es conmutativo. Esto evita transformar dos veces cada elemento geométrico de cada instancia.

Es fácil implementar un modelo basado en instanciación de símbolos sobre cualquier librería gráfica. Por este motivo esta estructura se utiliza frecuentemente en sistemas simples, en los que no es necesario almacenar relaciones topológicas entre elementos.

Fundamentos 2.3 Pipeline de visualización

Para dibujar el modelo es necesario transformar la información geométrica contenida en este en una imagen, formada por un conjunto de pixels. Esta operación la realiza habitualmente una librería gráfica, como OpenGL, sobre la que construimos el sistema. El proceso seguido se puede descomponer en varios pasos:

1. Cálculo de iluminación. En función de la posición de las fuentes de lúz se calcula el color con el que se debe dibujar cada polígono (o cada vértice, si se hace suavizado).

2. Tranformación a coordenadas de dispositivo (*Transformación de visualización*). Cada vértice se transforma de coordenadas del mundo a coordenadas de dispositivo. El sistema de coordenadas de dispositivo tiene los ejes x e y en el plano de dibujo y el eje z perpendicular a este.

3. Recortado. Se eliminan los elementos geométricos que quedan fuera de la zona de dibujo. Normalmente se hace también un recortado en profundidad, eliminando los elementos que están muy cerca o muy lejos del observador. (El recortado puede hacerse en coordenadas del mundo, antes de realizar la transformación de visualización). La zona de dibujo, que se denomina *viewport*, está deimitada, usualmente, por un rectángulo.

4. Rasterización. Las partes visibles de los polígonos se descomponen en pixels.

5. Eliminación de partes ocultas. La técnica más usada es el z-buffer, que suele estar implementado en el hardware de todas las tarjetas gráficas 3D. En esta técnica el sistema mantiene una memoria, con la misma estructura que la de imagen, conteniendo la profundidad a la que esta el objeto que se ha dibujado en cada pixel. De este modo, basta comparar la profundidad (coordenada z) de cada pixel con el valor almacenado en el z-buffer, para saber si ese punto del objeto es visible.

6. Asignación de color a los pixels. Si el pixel es visible se asigna su color a la posición correspondiente del z-buffer. Cuando se realiza suavizado, el color del pixel se obtiene por interpolación de los colores de los vértices.

Complementos 2.1 Recortado de símbolos

Un problema parecido al de concatenación de la transformación de instanciación con la de visualización nos podemos plantear con el recortado. En principio, siguiendo el procedimiento descrito previamente, es necesario transformar todas las instancias a coordenadas de dispositivo, para después determinar si son o no visibles. En modelos grandes, con un gran volumen de símbolos esto puede ser muy ineficiente.

La alternativa es anticipar el recortado de los símbolos. Este puede hacerse en coordenadas de mundo (lo que impedirá concatenar las dos transformaciones), o en coordenadas maestras. En cualquier caso, esto implica invertir la transformación de visualización, para calcular la imagen del *viewport* en coordenadas del mundo o maestras. Esta estrategia impide utilizar el algoritmo de recortado a nivel hardware. Además, será necesario disponer de un algoritmo que funcione con ventanas no paralelas a los ejes, si se permite que los símbolos estén girados en el modelo.

Otra alternativa es realizar un test previo para determinar si el símbolo queda completamente fuera del ViewPort, transformando una caja que lo encierre (bounding box). De este modo evitamos transformar las instancias que están totalmente fuera del viewport [Newman pp.134].

2.3 Modelos jerárquicos

La utilización de símbolos permite estructurar el modelo de un modo limitado. Tienen limitaciones, por ejemplo , en situaciones en las que es necesario que el usuario modifique dinámicamente la definición de los símbolos, o utilice símbolos complejos.

No obstante, la misma filosofía de descomposición de la que hemos hecho uso en los símbolos puede ser usada para generar modelos jerárquicos, que sean más complejos y más flexibles.

En general, los modelos geométricos suelen tener **estructura jerárquica**, obtenida por descomposición del elemento a modelar en componentes más pequeños. Cuando todos los niveles de la jerarquía contengan el mismo tipo de información hablaremos de modelos jerárquicos homogéneos, por el contrario, cuando cada nivel contenga información de un tipo tendremos un modelo heterogéneo. Estudiaremos cada tipo por separado, aunque con frecuencia ambos tipos de estructuras conviven en un mismo modelo, utilizando una descomposición homogénea en los niveles superiores y una heterogénea a nivel geométrico.

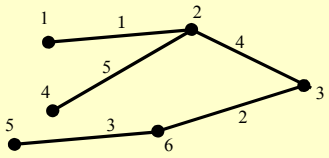
2.3.1 Modelos jerárquicos heterogéneos

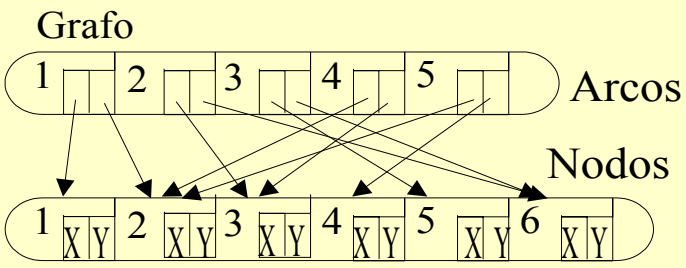
Cuando descomponemos el elemento a modelar en partes que contienen información de naturaleza diferente a la del propio elemento, obtenemos un modelo heterogéneo. En este tipo de modelos la jerarquía suele tener un nivel de profundidad fijo, almacenando en cada nivel un tipo distinto de información. Por ejemplo, un elemento puede estar representado por un conjunto de líneas, y cada línea estar representada por dos puntos. En este caso, la jerarquía tendrá solo dos niveles. Usualmente podemos ver cada nivel como un tipo de primitivas diferentes, por este motivo algunos autores llaman a estos modelos *primitivas relacionadas* [Sproull pp.316].

Este tipo de modelos es útil cuando existe dependencias topológicas entre componentes, lo que suele ocurrir cuando hay conexiones entre componentes [Newm81, Cap.9]. En los temas siguientes veremos la utilización de este tipo de modelos para representar curvas, superficies y sólidos.

Ejemplo 2.4. Modelo jerárquico heterogéneo.

Para modelar un grafo podemos utilizar un modelo jerárquico heterogéneo, en el que en el primer nivel se almacenen los arcos y en el segundo nivel los nodos. Esta representación nos permite almacenar la geometría del nodo (sus coordenadas) una única vez y mover los nodos garantizando que se mantiene la topología del grafo.



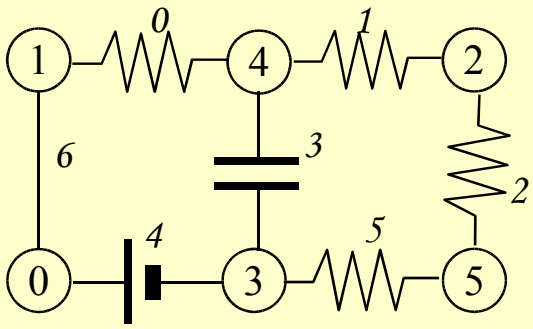


Nota: Esta no es la estructura óptima si es necesario estudiar caminos sobre el grafo

Ejemplo 2.5 Diseño de circuitos

A nivel de estructura habría dos tipos de elementos: conexiones y componentes. De estas últimas se almacenará su código, puntos de conexión, transformación, propiedades físicas e información de los componentes con los que está conectado. Obsérvese que hay información redundante.

Los puntos de conexión referencian a los componentes que están conectados en ellos.



Componentes												
Tipo	Conexiones			Transformación			Propiedades			Enlaces		
Cs	1	2	3	R	S	T	W	Val	max	Anterior		Siguientes
R	1	4		0	1	(1,2)	½	9W	100v	6		3 1
R	4	2		0	1	(2,2)	½	9W	100v	0	3	2
R	2	5		90	1	(3,1)	½	3W	100v	1		4
C	4	3		90	1	(2,1)	¼	2mF	400v	0	1	5 4
F	0	3		0	1	(1,1)	10	9v	10A	6		3 5
R	5	3		0	1	(2,1)	½	1W	100v	2		3 4

ºPuntos de conexión			
Id	x	y	*Componentes*
0	0	0	6 4
1	0	1	0 6
2	2	1	1 2
3	1	0	4 3 5
4	1	1	1 0 3
5	2	0	2 5

Conexiones				
	Conexiones		Enlaces	
Id	1	2	Anterior	Siguientes
6	0	1	4	0

2.3.2 Estructuras Multinivel Homogéneas:

Si descomponemos el elemento a modelar en componentes que contienen el mismo tipo de información obtenemos una estructura jerárquica homogénea. La estructura de este tipo de modelos se pueden representar como un grafo acíclico dirigido. La jerárquica, leída de abajo a arriba, se puede interpretar como un proceso de construcción, en el que los hijos de un nodo son componentes que utilizamos para construir el nodo padre. En cada nivel estamos construyendo objetos complejos utilizando componentes más simples [Salmon87, pp.278]. Cada nodo representa un componente del modelo.

Un componente se puede usar más de una vez en el modelo. Por este motivo la estructura del modelo es un grafo acíclico dirigido en lugar de un árbol.

En cierto modo podemos ver un modelo jerárquico[1] como una generalización de un modelo basado en instanciación de símbolos, en el que se utiliza el mismo proceso de instanciación para definir los símbolos.

A nivel conceptual representaremos los modelos jerárquicos por un grafo acíclico dirigido en el que cada nodo es una secuencia ordenada de elementos. Los elementos podrán ser primitivas, transformaciones geométricas o referencias a otros nodos.

Cada nodo se interpreta, leyendo de izquierda a derecha, del siguiente modo: El nodo representa las primitivas que aparecen en él, tras aplicarle las transformaciones geométricas que les preceden (es decir que están más a la izquierda) y las submodelos representados por los nodos que están instaciados en él, trás aplicarles, igualemnte, las transformaciones geométricas que les preceden.

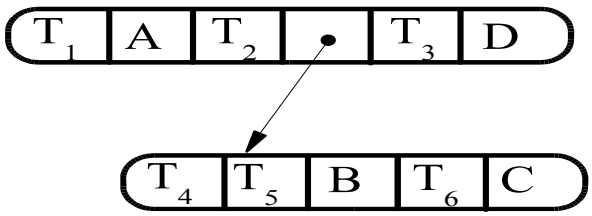


Figura 2.6. Esquema de un modelo jerárquico.

La figura 2.6 muestra el esquema de un modelo jerárquico en el que las tes indican transformaciones, A, B y C son primitivas. Las referencias a otros nodos (instanciaciones) se representan mediante flechas. La interpretación del nodo inferior es:

$$T_4 (T_5 (B)) + T_4 (T_5 (T_6(C)))$$

Esto es, se dibuja la primitiva B aplicandole la transformación T_5 y posteriormente la T_4; después se dibuja la primitiva C aplicandole las transformaciones T_6 , T_5 y T_4, en este orden. Obsérvese que las transformaciones se aplican de derecha a izquierda, como las funciones. El signo más en la expresión se utiliza como separador de primitivas, indicando que las dos primitivas se añaden a la imagen.

Cada transformación geométrica indica un cambio de posición, orientación o tamaño del objeto. Estos cambios se pueden interpretar como cambios en el sistema de coordenadas. Pensemos que estamos construyendo un modelo del sistema solar. Hemos hecho un modelo de la Luna, lo que implica definir su geometría (que tomamos como una esfera). Lo más cómodo es definir una esfera centrada en el origen de coordenadas (el sistema de coordenadas de la luna). Ahora hay que modelar la tierra, para lo que podemos utilizar el mismo método. Ahora tenemos dos esferas centradas en el origen. Podemos interpretar que están las dos en el mismo sistema de coordenadas y que hay que "colocar la luna"; o que están ya en su sitio en el sistema solar, y que tenemos la geometría de cada una referida a su propio sistema de coordenadas, y que debemos convertir la geometría de la luna de su sistema de coordenadas al de la tierra. El resultado es el mismo, solo cambia la forma de entender el proceso. En cada caso debemos hacer lo que nos sea más fácil de entender, y por tanto de calcular.

[1] En adelante utilizaremos el término modelo jerárquico para referirnos a modelos homegéneos, cuando no se indique lo contrario.

Volviendo al modelo de la figura anterior, y llamando "G" al nodo de segundo nivel, la interpretación del primer nodo es:

$$T_1(A) + T_1(T_2(G)) + T_1(T_2(T_3(D)))$$

Para ver el significado del modelo completo sustituimos la interpretación de G:

$$T_1(A) + T_1(T_2(T_4(T_5(B)))) + T_1(T_2(T_4(T_5(T_6(C))))) + T_1(T_2(T_3(D))) \quad (1)$$

El proceso de construcción garantiza que las transformaciones que se encuentran en los niveles inferiores se aplican antes que las que se encuentran en los niveles superiores.

Para visualizar, o en general interpretar el modelo, las transformaciones geométricas se gestionan como en un modelo basado en símbolos, con la única salvedad de que se pueden concatenar varias transformaciones de modelado (cada nodo tendrá una). En cada nodo se van aplicando transformaciones, que se pueden concatenar directamente con las previamente indicadas en el nodo. Sin embargo, el conjunto de las transformaciones del nodo se deben dejar de aplicar cuando terminamos de dibujarlo y volvemos al nivel superior. Tal como ocurre en el ejemplo anterior después de dibujar el nodo G.

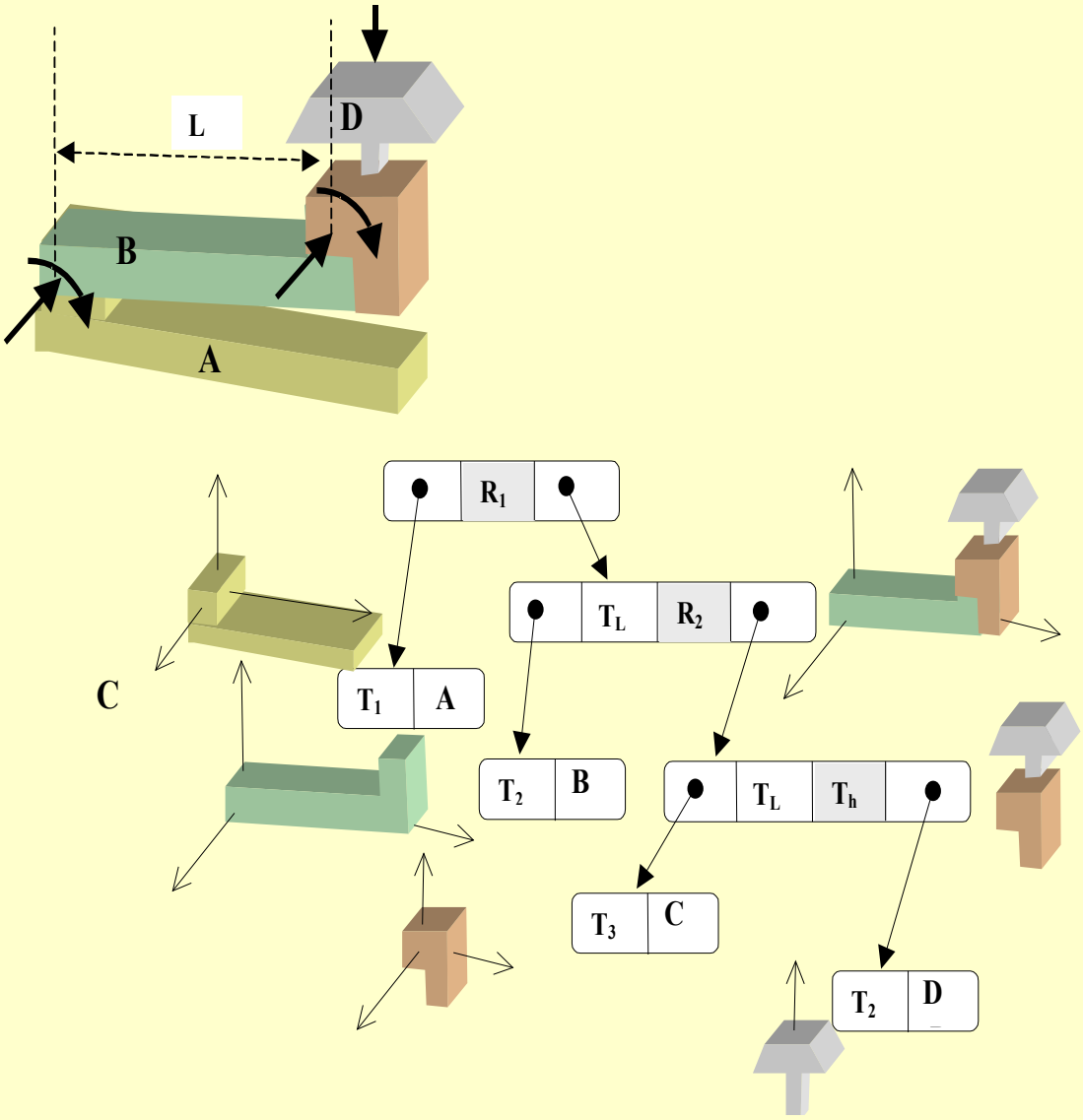
Una forma conveniente de gestionar las transformaciones es construyendo una *pila de transformaciones,* en la que añadiremos la transformación actual (composición de todas las que se están aplicando) cuando bajemos de nivel, y restauraremos la anterior, desapilandola, al subir de nivel.

OpenGL no dispone de ningún mecanismo de estructuración de la escena, pero gestiona el apilamiento de transformaciones geométricas, lo que facilita la construcción de modelos jerárquicos. De hecho es basta con encerrar cada nodo entre llamada a glPushMatrix y glPopMatrix. Como OpenGL no almacena el modelo, tenemos que entregar, cada vez que se va a redibujar, un recorrido completo de la jerarquía. Por tanto, si queremos que un nodo esté instanciado en dos puntos tenemos que ejecutar el código que lo dibuja dos veces (obviamente lo más simple es crear un procedimiento que lo dibuje.

Para crear un modelo jerárquico de un objeto se debe proceder sistemáticamente. Un método simple es el siguiente:

1. Crear un grafo del objeto. Para ello descomponer el objeto de componentes más simple, y repetir el proceso con los componentes, hasta conseguir elementos simples, directamente implementables.
2. Hacer un boceto de cada nodo, indicando el sistema de coordenadas.
3. Partiendo de los nodos de nivel determinar las transformaciones geométricas necesarias. Sin calcular los parámetros de éstas. En este momento tendremos una representación semejante a la mostrada en el figura 2.6.
4. Partiendo de los niveles inferiores, asignar valores a la geométria de las primitivas y a los parámetros de las tranformaciones geométricas.

En cualquier caso, el modelo obtenido no es único. Es posible descomponer el sistema de diversos modos, y colocar las transformaciones en diferentes sitios o con diferentes parámetros. Asi, por ejemplo, en la figura 2.6, la transformación T_2 se podría haber suprimido, si las transformaciones T_3 y T_4 se hubiesen sustituido por $T_2 \cdot T_3$ y $T_2 \cdot T_4$ respectivamente.



Ejemplo 2.6 Diseño de un modelo jerárquico.

Complementos 2.2 PHIGS

PHIGS es una librería gráfica 3D. La librería fue diseñada a finales de la década de los 80, y es estándar ISO. A finales de los 80 llego a ser la librería gráfica más utilizada, siendo posteriormente desplazada por OpenGL.

PHIGS almacena internamente la información gráfica utilizando la siguiente estructura:

Structure = Secuencia(**Element**)

Elemment = Primitiva | Atributo | Transformación. | **Call(Structure)**

Structure es el nombre que se da en PHIGS a un nodo del modelo. Los componentes de una estructura está numerados de 0 a n, existe un puntero por cada estructura, que apunta a un elemento de la estructura, con el que se pueden realizar operaciones de edición (borrar, insertar, modificar). A continuación mostramos el esqueleto de un fragmento del código que crea un modelo

```
Procedure Triangulo( var tr:integer);      // Crea una nodo que contiene un triángulo
        SPH_openStructure( Tr )            // Crea un nodo PHIGS (estructura)
                PolyLine(...               // Almacena una poligonal en la estructura
                ...
        SPH_closeStructure;                // Cierra la estructura

Procedure Estrella( var str:integer);      // Crea una estrella
        var tr:integer;

        Triangulo(tr);                     // Llama a triángulo para que cree el nodo
        SPH_openStructure( str );          // Crea un nuevo nodo PHIGS
        SPH_executeStructure( Tr );        // Llama (instancia) el nodo triángulo
        for i := 1 to 5
                SetLocalTransformation( Rotate( a ), PRECONCATENATE);
                SPH_executeStructure( Tr );        // Instancia otro triángulo
        SPH_closeStructure;                // Cierra la estructura
```

El utilizar PHIGS permite que el programa se desentienda de la visualización y de las operaciones de entrada (selección, etc.). Por otra parte, esto puede conllevar una duplicación de datos entre las estructuras del programa y las del PHIGS.

2.4 Arquitectura del sistema

Algunas librerías gráficas, como PHIGS, almacenan una copia del modelo geométrico, decimos que funcionan con modelo retenido.

Otras librerías, como OpenGL, no almacenan el modelo geométrico, dejando la gestión de éste a la aplicación. La aplicación debe disponer de una función que entregue la información geométrica del modelo a la librería cada vez que se deba actualizar la imagen de salida. Este modo de funcionamiento se conoce como inmediato.

En principio, la utilización de librerías que funcionen con modelo retenido puede simplificar las aplicaciones, ya que no es necesario desarrollar código para gestionar el modelo. No obstante en muchos casos la estructura, y la forma de gestión impuesta por la librería no es la más adecuada para la aplicación, por lo que

es necesario almacenar el modelo en ésta. En estos casos, la utilización de una librería de modelo retenido es un inconveniente, pues, además de no simplificar la aplicación, crea un problema adicional, al ser necesario mantener la consistencia de los dos modelos.

Complementos 2.3 Display lists

Un display list es un conjunto de primitivas precompilados que se almacenan para poder redibujarlos repetidas veces de un modo eficiente. Este modo de estructurar la información geométrica ha sido utilizado profusamente por los sistemas vectoriales, que necesitaban redibujar el modelo geométrico periódicamente [Newman 82]. El nacimiento de los sistemas raster hizo que fuese sustituido por la memoria de imagen, como estructura a usar en el proceso de refresco.

No obstante, las listas de visualización han resurgido debido a su eficiencia, en sistemas en los que, por distintos motivos, es necesario regenerar la imagen a partir del modelo. Esto ocurre, por ejemplo, cuando se desea realizar animaciones, o visualizaciones interactivas, en las que el usuario puede cambiar algún parámetro (focos, punto de vista etc.). Esta es la filosofía que se ha seguido en OpenGL.

En OpenGL un display list es una secuencia de primitivas precompiladas (listas para ser visualizadas), sin posibilidad de edición. El objetivo es aumentar al máximo la velocidad de dibujo. La lista puede contener primitivas, transformaciones geométricas, cambios de atributos. La visualización se realiza invocando la lista. Cualquier proceso de edición implica regenerar la lista.

Bibliografía

Emmerik90 Emmerik M.J.G.M. van: "A Direct Manipulation Technique for Specifying 3D Object Transformations with a 2D Input Device". **Computer Graphics Forum**. Vol. 9, N.4, Dec. 1990. pp. 355-362.

Foley82 J. Foley, V. Wallace: "The Art of Natural Graphic Man-Machine Conversation". *Proceeding of the IEEE*, 1974. Reproducido en Beatty, Botch (Ed). *Tutorial: Computer Graphics*, IEEE Press, 1982.

Foley90 Foley J.D.; van Dam A.; Feiner S.K.; Hughes J.F.: *Computer Graphics. Theory and Practice*. Addison-Wesley 1990.

Glassner90 Glassner A.S.: "A Two dimensional View Controller". **ACM Transactions on Graphics**. Vol.9, N.1, Jan. 1990. pp. 138-141.

Hear94 Hearn D.D.; Baker M.P.: **Gráficas por computadora**. Prentice Hall 1994

Hopgood92 Hopgood F.; Duce D.A.; Johnston D.J.: *A Primer for PHIGS. C programmers' Edition*. John Wiley & Sons. 1992.

Hudson92 S.E. Hudson: "Adding shadows to a 3D Cursor". **ACM Transations on Graphics**. Vol. 11, N.2, pp. 193-199. 1992.

Myers82 W. Myers: "Interactive Computer Graphics: Poised or Takeoff". **ACM Computer Graphics**. Proceeding of the 4th Conference on Computer Graphics & Interaction Techniques 1977. Reproducido en Beatty, Botch (Ed). *Tutorial: Computer Graphics*, IEEE Press, 1982

Newman81 Newman W.M.; Sproull R.F.: *Principles of Interactive Computer Graphics*. McGraw-Hill 1981

Nielson87 G. Nielson, D. Olsen: "Direct Manipulation Techniques for 3D Objects using 2D Locator Devices". *ACM workshop on Interactive 3D Graphics*. pp.175-182. 1987

Salesin93 Salesin C.; Barzel R.: "Adjustable Tools: An Object-Oriented Interaction Metaphor". **ACM Transactions on Graphics**. Vol.12, N.1, Jan. 1993. pp. 103-107.

Salmon87 Salmon, R.; Slater, M.: *Computer Graphics: Systems and Concepts*. Addison Wesley, 1987.

Sproull86 Sproull R.; Southerland: *Device independent Graphics*. McGraw-Hill 1986

Ejercicios

Modelado geométrico.

1. Redactar una función *Transform*(Sx, Sy, angulo, tx,ty,n,vertices) para transformar un símbolo, expresado como un polígono de n vértices (contenidos en el array vertices).
2. Describir un procedimiento de visualización para un sistema 2D que utilice símbolos, usando enmarcado para filtrar las instancias de símbolos que quedan fuera de la ventana.
3. ¿Cómo se puede calcular el marco de un símbolo?
4. Describir la estructura del modelo geométrico de un sistema de diseño de circuitos electrónicos, que utilice un modelo basado en instancias
5. Encontrar una estructura para el modelo de una aplicación de diseño del amueblamiento de una planta de un edificio. ¿Tendría alguna ventaja utilizar un modelo jerárquico?
6. Se dispone de dos *display lists* conteniendo un cubo unitario y una piramide de base cuadrada y lado 1. Redactar un procedimiento que genere una figura formada por un cubo central de tamaño 10 y seis pirámides cuadradas de lado 5 adosadas a sus caras.
7. ¿Es posible construir un sistema gráfico basado en instancias sobre OpenGL ?. ¿Cómo? .

Modelos jerárquicos.

8. Se desea usar la librería OpenGL para diseñar la grua de la figura 2.12. El modelo se debe diseñar de forma que sea posible, y fácil, girar el brazo, desplazar la pluma, y subir y bajar el gancho. Debe tenerse en cuenta las restricciones de cada uno de estos movimiento. Realizar el grafo del modelo, indicando de forma clara el contenido de cada nodo (primitivas y transformaciones). Tener en cuenta que la cuerda debe cambiar de tamaño al moverse el gancho. Especificar las transformaciones geométricas que sería necesario introducir para realizar los movimientos anteriores.

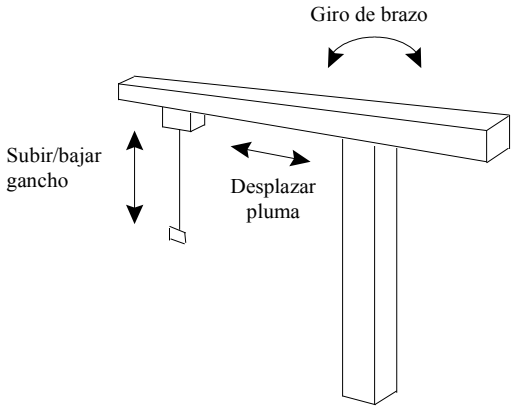


Figura 2.12

9. Describir la estructura que tendría en OpenGL el modelo del sistema de la figura 2.13. El sistema está formado por una secuencia de tres segmentos conectados mediante articulaciones, el primero de los cuales se une a un cuarto segmento fijo mediante otra articulación. Los segmentos poseen una guía central que fuerza a estos a mantener una altura constante, haciendo que de hecho el sistema posea un solo grado de libertad. Dibujar el grafo de estructuras, indicando las transformaciones geométricas, y el proceso que se debe seguir para modificar el ángulo.

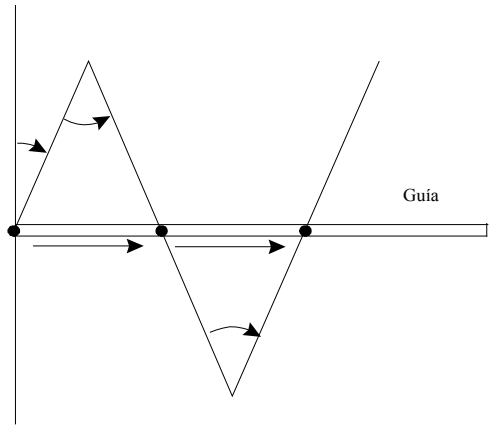


Figura 2.13

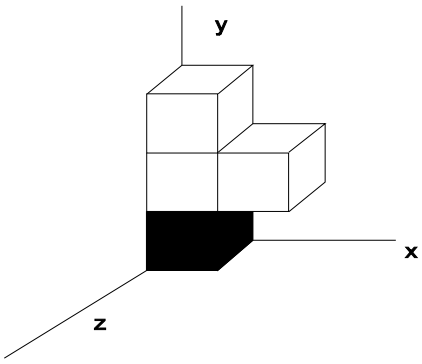


Figura 2.14

10. Realizar en OpenGL el modelo mostrado en la figura 2.14. Los cubos poseen dimensión unidad, y se dispone de un procedimiento que crea un cubo definido en el origen.
11. Generar el modelo necesario para obtener un mosaico de la figura 2.15, partiendo de la figura generada en el ejercicio anterior.

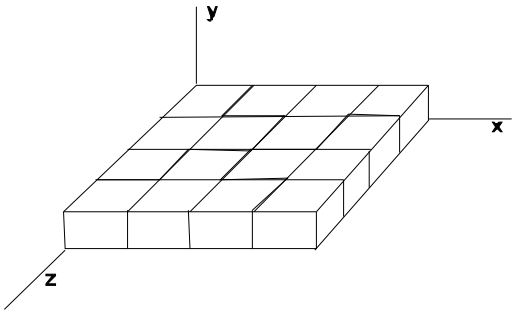


Figura 2.15

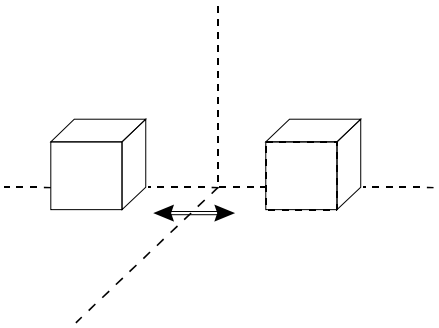


Figura 2.16

12. Indicar la forma en que se podría construir en OpenGL el modelo de la figura 2.16, de tal modo que simplemente añadiendo una transformación geométrica se puedan separar los dos cubos de forma simétrica del centro (suponer definida la estructura cubo).

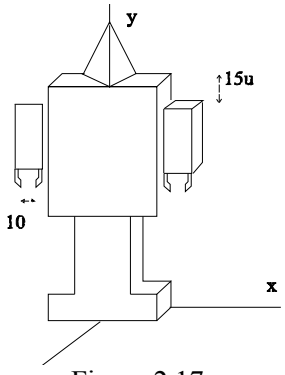


Figura 2.17

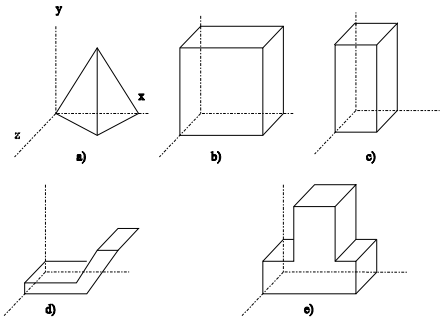


Figura 2.18

13. Se desea usar la librería OpenGL para diseñar el robot de la figura 2.17. Para ello, se dispone de un conjunto básico de elementos que se muestra en la figura 2.18. El robot se compone de cabeza, tronco, base, brazos y pinza, estando situado sobre el origen de coordenadas, mirando a lo largo del eje z. Crear el grafo de estructuras.
    El extremo inferior izquierda de todas las piezas está situado sobre el origen de coordenadas.
    El tamaño de cada una de las piezas es el siguiente:
    Cabeza: Xmáx = 30, Ymáx = 20, Zmáx = 20
    Tronco: Xmáx = 50, Ymax = 80, Zmáx = 20
    Brazo: Xmáx = 15, Ymáx = 40, Zmáx = 20
    Pinza: Xmáx = 10, Ymáx = 5, Zmáx = 20
    Base: Xmáx = 40, Ymáx= 40, Zmáx = 20
    (La parte superior de la base tiene un ancho de 20)
    Diseñar el robot usando OpenGL, partiendo del conjunto de piezas ya creado como display lists, con la ubicación que se muestra.

14. Introducir en el diseño anterior los siguientes movimientos:
    a) giro del tronco 90º en sentido antihorario
    b) elevación del brazo izquierdo 90º

15. Considerar y explicar razonadamente cómo quedaría el grafo (DAG) si el movimiento de los dos brazos fuese solidario, (elevación) y se conservase el movimiento independiente de cada una de las muñecas y pinzas.

16. Obtener un modelo de una persiana de varillas, de forma que podamos subir y bajar las varillas, y además, rotarlas solidariamente (para oscurecer). La figura 2.19 muestra un esquema 2D del movimiento.

17. Modelar una balanza como la mostrada en la figura 2.20. Redactar un procedimiento de edición para girar la balanza un determinado ángulo (siendo la posición del dibujo de reposo).

18. Se desea modelar el mecanismo de una biela, ver figura 2.21. Dado el giro (de b), se debe modificar la posición de la biela (a) de tal modo que su longitud permanezca constante. EL pistón (c) se moverá verticalmente (subir y bajar) dependiendo del ángulo de giro. Se parte de un cubo unidad situado en el origen con los extremos P0(0,0,0) y P1(1,1,1). Las longitudes de los elementos es a,b,y c.

19. ¿Es posible modelar un cubo rubik 2x2x2 con OpenGL?.

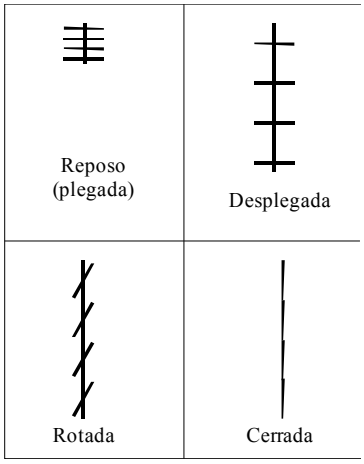


Figura 2.19

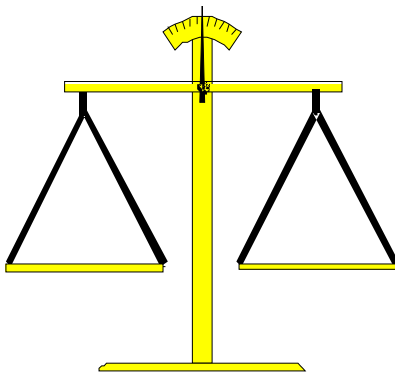


Figura 2.20

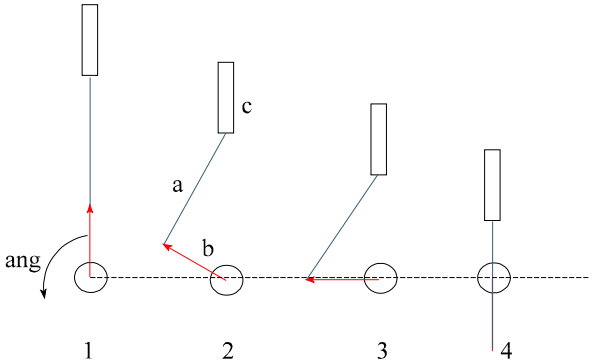


Figura 2.21