

## Modelado Jerárquico FI UNAM 2007

En gran cantidad de aplicaciones, es conveniente poder crear y manejar partes individuales de una imagen sin afectar otras partes de las mismas.

Definir cada objeto con una imagen como modulo separado nos permite realizar modificaciones en la imagen con mayor facilidad.

Esta capacidad hace posible modificar unas partes mientras que otras permanecen inalterables.

Modelado: Creación y manipulación de un sistema.

Modelo: Cualquier representación particular.

Existen modelos descriptivos (ecuaciones) y modelos gráficos o geométricos: ya que están formados de líneas, polígonos o circunferencias.

En esta materia, se utiliza el modelado geométrico o representación geométrica de un sistema generado por una computadora.

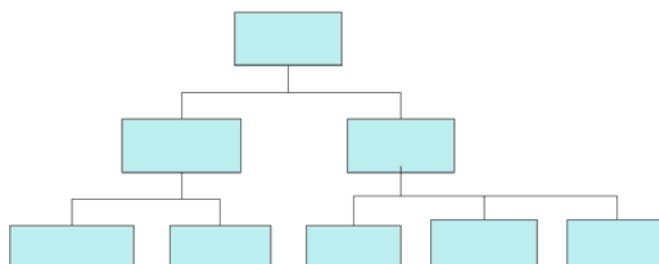
Existen 2 métodos para especificar la información necesaria para crear y manipular un objeto.

|  |   |
|--|---|
| Un método consiste en almacenar la información en una estructura de datos, como una tabla o una lista relacionada. | Un método que se especifica la información en procedimientos. |
| En general, la especificación de un modelo contendrá tanto estructuras de datos como procedimientos.               |   |

Jerarquía de símbolos.

En este caso, se separan en bloques básicos ó “módulos” de simples geometrías, y de aquí construir bloques compuestos (módulos compuestos).

Esto es semejante a trabajar con clases, se crea una plantilla o clase y luego se reutiliza para crear una o varias clases derivadas, pero con sus limitaciones de trabajar con estructuras; las estructuras solo manejan datos ya que es una variable estática, mientras que en las clases apenas se crea el tipo de dato y faltaría además crear su instancia.



Modelado jerárquico con estructura. Se crea un modelo jerárquico de un sistema con estructuras al integrar las estructuras entre si para formar un árbol jerárquico. Conforme se dispone cada estructura en la jerarquía, se le asigna una transformación apropiada de manera que se adapte en forma adecuada el modelo general.

### Modelos jerárquicos Heterogéneos

Cuando se descomponen los elementos a modelar en partes que estos contienen información diferente a la del propio elemento. Este tipo de modelos suele tener un nivel de profundidad fijo, almacenando en cada nivel un tipo distinto de información.

Por ejemplo, un elemento está representado por un conjunto de líneas, y cada línea está representada por dos puntos. En este caso, la jerarquía tendrá solo dos niveles. Usualmente podemos ver cada nivel como un tipo de primitivas diferentes. (También llamadas primitivas relacionales).

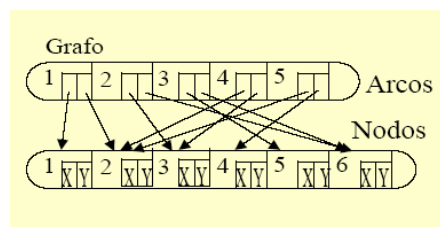
### Estructuras multinivel homogéneas

Si se descompone el elemento a modelar en componentes que contienen el mismo tipo de información obtenemos una estructura jerárquica homogénea. La jerarquía leída de abajo hacia 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. Cada nodo representa un componente del modelo.

Un componente se puede usar más de una vez en el modelo.

Hay autores que dicen que el modelo jerárquico es como una generalización de un modelo basado en instanciación de símbolos.

Es una ruta dirigida en la que cada nodo es una secuencia ordenada de elementos. Los elementos podrán ser primitivas, transformaciones geométricas o referencias a otros nodos.

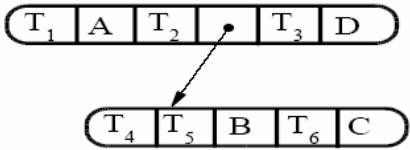


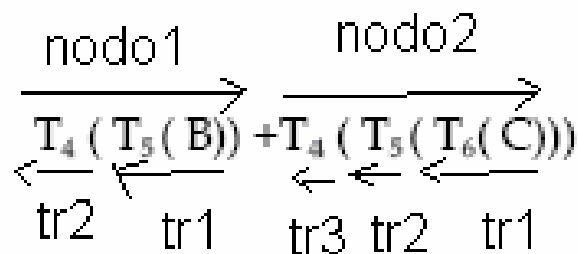
Cada nodo se interpreta, leyendo de izquierda a derecha, del siguiente modo:

El nodo representa las primitivas que aparecen él, tras aplicarle las transformaciones geométricas que les preceden (que están más a la izquierda) y los símbolos representados por los nodos que están instanciados en él, tras aplicarles, igualmente las transformaciones geométricas que les proceden.

En la siguiente figura se muestra un esquema de un modelo jerárquico en el que las Ts indican transformaciones, mientras que A, B y C son las primitivas. Las referencias a otros nodos (instancias) se indican mediante flechas.

Las interpretaciones son:

|   |  |
|---|--|
|  | <p>Nodo superior conectado a inferior conG</p> $T_1(A) + T_1(T_2(G)) + T_1(T_2(T_3(D)))$ <p>nodo inferior</p> $T_4(T_5(B)) + T_4(T_5(T_6(C)))$ |
|---|--|



Y finalmente sustituyendo G

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

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, desapilándola, 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 .
4. Partiendo de los niveles inferiores, asignar valores a la geometría de las primitivas y a los parámetros de las transformaciones 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. Así, 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.

## INTRODUCCIÓN A MODELO JERÁRQUICO (matrices OpenGL).

En cualquier aplicación o paquete grafico, así como también OpenGL, toda la geometría se ve afectada por la CTM (Current Transformation Matrix) ó matriz de transformación actual; esta matriz guarda la información sobre todas las matrices que se han ido acumulando. Cualquier vértice que pase por la tubería grafica de vértices será multiplicado por esta matriz y consecuentemente transformado.

La CTM se compone de 2 matrices. La del modelView y la de projection. Ambas se concatenan y de su producto se crea la CTM.

Tubería de vértices

$$\begin{array}{c}
 \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ W_0 \end{pmatrix} \begin{pmatrix} **** \\ **** \\ **** \\ **** \end{pmatrix} \\
 \text{Factor de escala} \quad \text{Matriz ModelView}
 \end{array}
 =
 \begin{array}{c}
 \begin{pmatrix} X_e \\ Y_e \\ Z_e \\ W_e \end{pmatrix} \begin{pmatrix} **** \\ **** \\ **** \\ **** \end{pmatrix} \\
 \text{Matriz Projection}
 \end{array}
 =
 \begin{array}{c}
 \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ W_c \end{pmatrix} \begin{pmatrix} **** \\ **** \\ **** \\ **** \end{pmatrix} \\
 \text{Coord. corte}
 \end{array}
 =
 \begin{array}{c}
 \begin{pmatrix} X_c/W_c \\ Y_c/W_c \\ Z_c/W_c \end{pmatrix} \\
 \text{Normalización}
 \end{array}
 \longrightarrow \dots$$

Pilas de matrices de glMatrixMode:

GL\_MODELVIEW :para mover objetos alrededor de la escena

GL\_PROJECTION :Volumen de corte

GL\_TEXTURE : manipula coordenadas de textura

En openGl utilizaremos la propiedad de pila que se tiene en las funciones de sus matrices. En este caso, se van apilando (push) las transformaciones hasta que se comiencen a sacar (popMatriz) o se regresa al inicio con LoadIdentity.

La matriz modelview puede resetearse para evitar los efectos de la función intrínseca que es acumulativa. Ya que con cada llamado, la matriz se construye y multiplica por la matriz actual de modelview. Evitando se produzcan transformaciones anidadas.

O sin resetear para utilizar el lugar de cada objeto, frecuentemente se busca salvar el estado actual de transformación y después de esto restablecer la posición de algunos objetos. Esta aplicación es más conveniente cuando se tiene transformaciones iniciales en la matriz modelview como una vista de transformación.

La matriz identidad es un elemento neutro en multiplicación de matrices, y aquí solo nos aseguramos que se limpie o resetee la matriz dada.

$$\begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ W_0 \end{pmatrix} \begin{pmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ W_0 \end{pmatrix}$$

Para facilitar esto, OpenGL, mantiene una pila de matrices para ambas matrices ya sea modelview ó projection. Una matriz pila trabaja como un programa ordinario de pilas. Se puede ingresar (push) a la pila y salvar, y entonces hacer los cambios para tener la matriz actual. Para sacar(pop) de la pila y restablecer la pila.

Y aquí la idea es poder salvar estado de la pila en cualquier momento para recuperarlo después, con las funciones `glPushMatrix()`; => salvamos estado actual; y `glPopMatrix()`; => recuperamos el estado actual.

Servirá en el caso que se tenga que aplicar alguna transformación a una pequeña parte de la geometría. El resto no debería verse afectado por esos cambios.

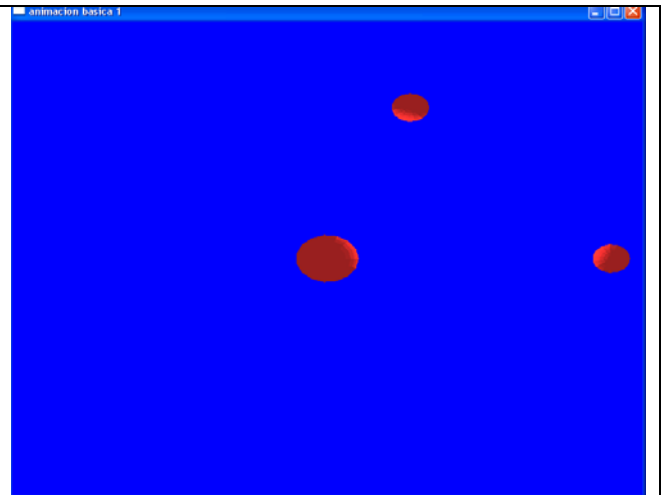
Lo que se hace es definir las transformaciones generales que afectan a todos. Entonces se salva la matriz y se añade otras transformaciones.

### Ejemplo

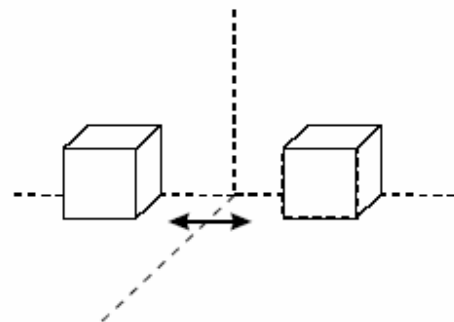
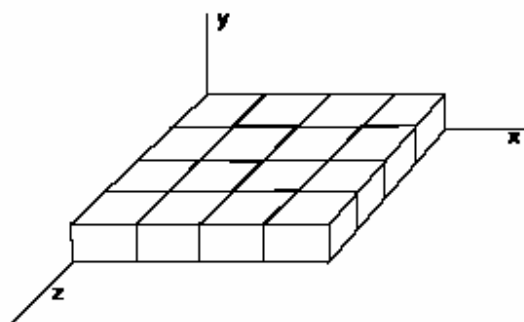
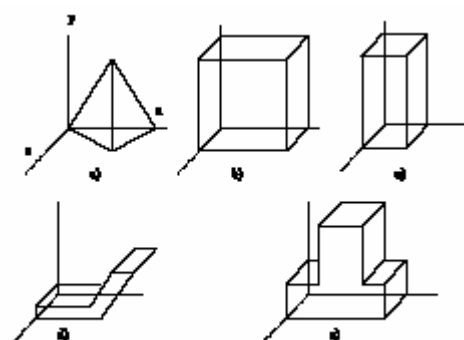
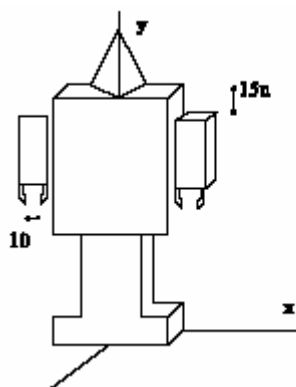
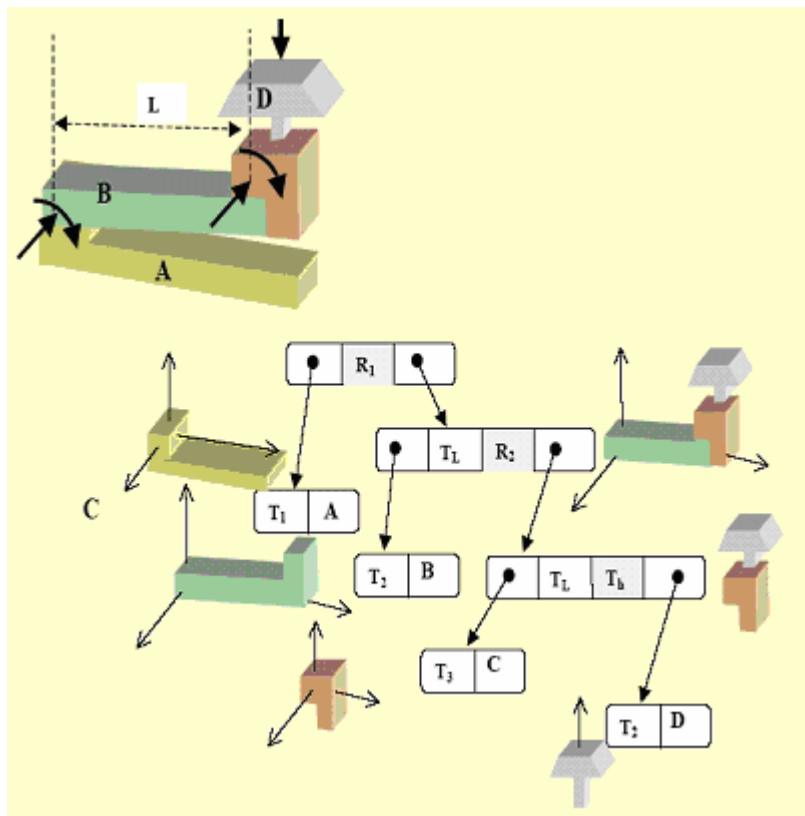
```
glPushMatrix();
```

```
glTranslatef(90.0f,0.0f,0.0f);
    glutSolidSphere(6.0f,15,15);
glRotatef(45.0f,0.0f,0.0f,1.0f);
```

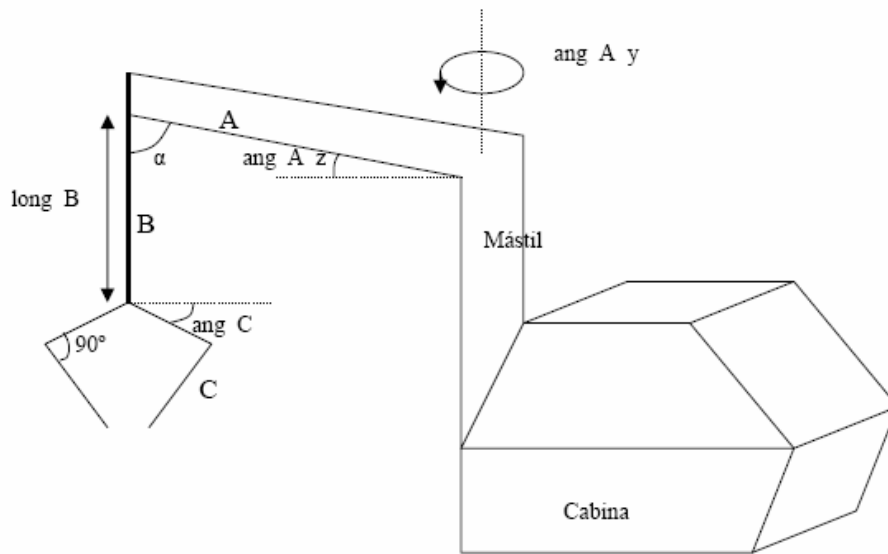
```
glTranslatef(0.0f,90.0f,0.0f);
    glutSolidSphere(6.0f,15,15);
glPopMatrix();
```



Ejemplos de Modelado jerárquico.

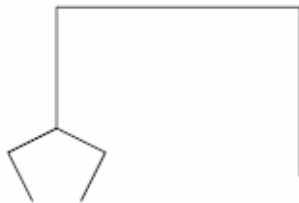


## Ejemplo

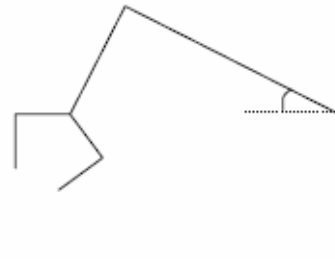


$\text{ang\_A\_y} \rightarrow$  indicará el ángulo de rotación de la pieza A ( y también B y C) respecto del eje marcado por la pieza “mástil”

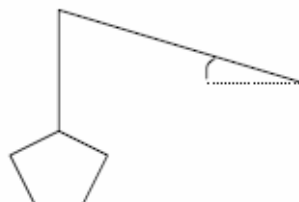
$\text{ang\_A\_z} \rightarrow$  indicará el ángulo de rotación de la pieza A (sólo de A , no de B ni C) respecto de la horizontal. Sí afectará a B y C porque cuando este ángulo sea mayor, B y C alterarán su posición, pero no “rotarán” lo marcado por  $\text{ang\_A\_z}$  (como sí ocurría con  $\text{ang\_A\_y}$ ). Es decir, si incrementamos este parámetro queremos obtener esta forma para la grúa :



Partiendo de esta situación ,  
Incrementando  $\text{ang\_A\_z} \rightarrow$



No buscamos esto....



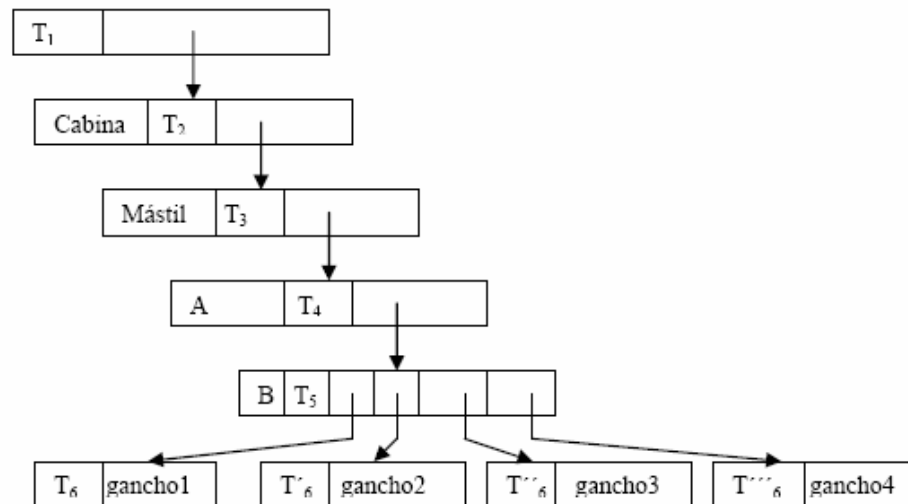
Sino esto otro.



Para ello, habrá que tener presente lo ya comentado, y además , hacer siempre  $\alpha = 90^\circ - \text{ang\_A\_z}$

$\text{long\_B} \rightarrow$  la longitud de la pieza B (afectará también a la pieza C puesto que esta pieza “cuelga” de la anterior) .  
 $\text{ang\_C} \rightarrow$  ángulo de rotación respecto de la horizontal de la pieza C

Por lo tanto, montaremos la grúa de puerto con el siguiente diseño jerárquico :



En el dibujo del diseño de la grúa mostrado anteriormente, no se incluía el hecho de que la pinza estará compuesta por 4 “ganchos”, de la forma :



Así, llamaremos gancho1, gancho2, gancho3 y gancho4 a cada uno de los ganchos de la pinza.

Hecha esta aclaración, podemos especificar las transformaciones realizadas por la construcción jerárquica de la grúa:

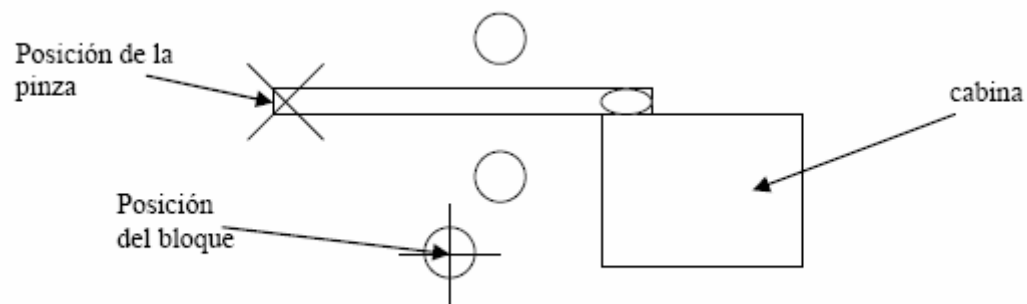
- $T_1 \rightarrow$  transformación genérica para toda la grúa.
- $T_2 \rightarrow$  traslación para colocar el brazo de la grúa junto a la cabina, adyacente pero no dentro de ella : `glTranslatef(0, 0, -0.4)`.
- $T_3 \rightarrow$  traslación para colocar las piezas restantes a la altura convenida +  
Rotación en el eje vertical del mástil, dada por el ángulo `ang_A_y` +  
Rotación en el eje horizontal , perpendicular a la pieza A, dada por el ángulo `ang_A_z`  
`glTranslatef(0, 7.75, 0);`  
`glRotatef(ang_A_y, 0, 1, 0);`  
`glRotatef(-ang_A_z, 0, 0, 1);`
- $T_4 \rightarrow$  traslación para colocar las piezas restantes colgando del extremo de la pieza A  
Rotaciones para que todo lo que cuelga de A esté inclinado, de la forma explicada en el comentario de arriba, si A lo está:  
`glTranslatef(-9.5, 0, 0);`  
`glRotatef(ang_A_z, 0, 0, 1);`  
`glRotatef(ang_A_y, 0, 1, 0);`
- $T_5 \rightarrow$  traslación para colocar los ganchos en el extremo de la pieza B  
`glTranslatef(0, -long_B+0.2, 0);`
- $T_6 \rightarrow$  rotaciones para colocar el gancho correctamente. En función del gancho de que se trate, estas rotaciones se realizarán sobre un eje u otro, en función del valor `ang_C`, de ahí las 4 variantes de  $T_6$ , especificadas con comillas, que se muestran arriba  
 $T_6 :$  `glRotatef(-ang_C, 0, 0, 1);`  
 $T'_6 :$  `glRotatef(ang_C, 0, 0, 1);`  
 $T''_6 :$  `glRotatef(-ang_C, 1, 0, 0);`  
 $T'''_6 :$  `glRotatef(ang_C, 1, 0, 0);`

Mástil será un cilindro construido mediante el generador de sólidos por revolución, tendrá altura 8 y diámetros de la base 0.8x0.8

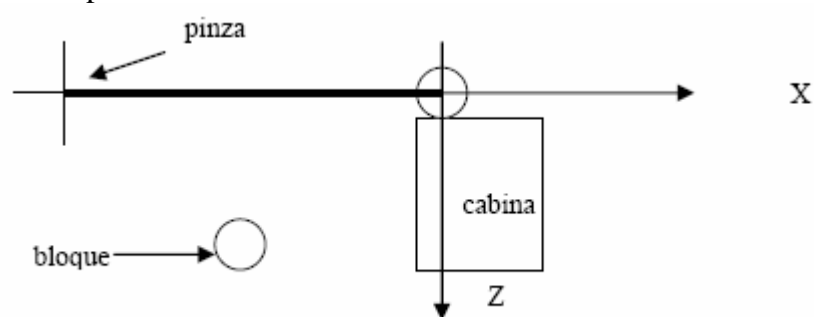
A será un cilindro construido mediante el generador de sólidos por revolución, de medidas 0.5x10x0.5 con 0.5x0.5 diámetros de la base, y 10 unidades de altura

B será un cilindro muy fino, generado como los anteriores, de longitud `long_B` y diámetros para la base y altura de 0.15x0.15

**Automatización del movimiento de recogida de un bloque previamente seleccionado.**



buscar posición



alzar posición.

