DISEÑO ASISTIDO POR COMPUTADORA

Proyecto de Prácticas :

Diseño de una grúa de puerto

Carlos Cano Gutiérrez

Grupo : **MIE 18-20**
Profesor : **Pedro Cano**

4º Ingeniería Informática
Febrero 2002

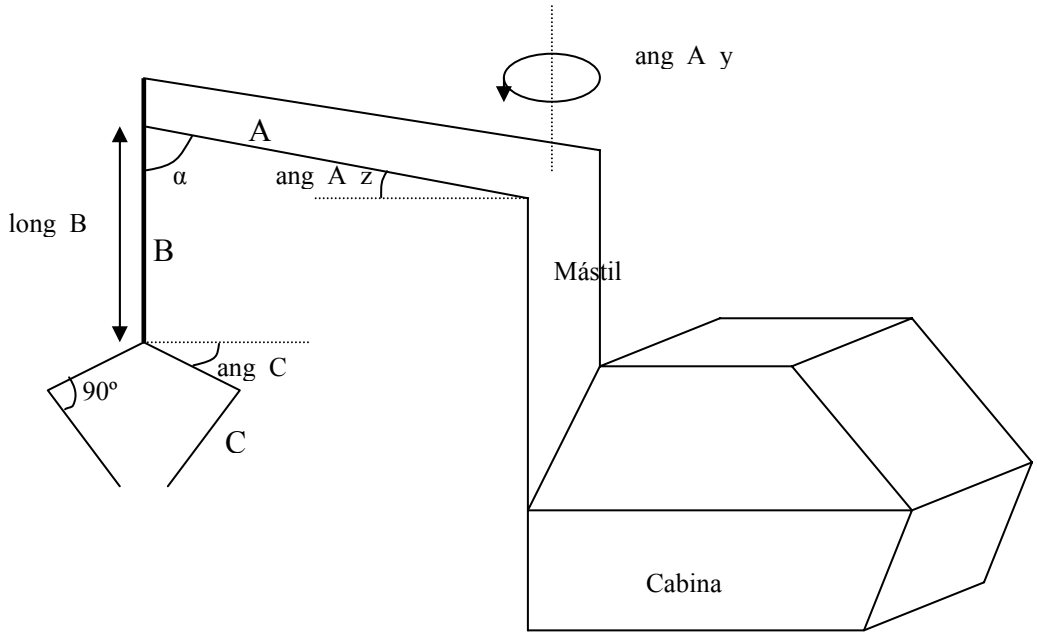
ÍNDICE DE CONTENIDOS:

1.- Contenidos relevantes del proyecto.

En el proyecto diseñado, que consistía básicamente en la construcción de una grúa de puerto mediante el empleo de un modelado jerárquico, he ido incluyendo "extras" y aspectos complementarios para permitir la interacción con el usuario y cierto comportamiento automatizado de la citada grúa. Además he incluido superficies generadas empleando Bsplines, objetos sólidos construidos por revolución de una curva sobre un eje, y otra serie de aplicaciones básicas para las cuales he empleando lo aprendido en las clases de la asignatura.

1.1.- Diseño jerárquico de una grúa de puerto.

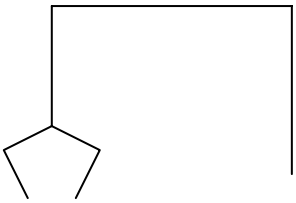
El aspecto aproximado de la grúa que se pretende construir será :



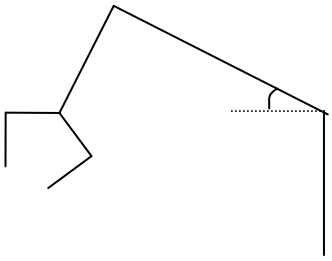
Como se puede ver en el dibujo, hemos identificado ciertas piezas para realizar más claramente el modelado jerárquico. También hemos señalado los grados de libertad que vamos a imponer a nuestro modelo:

ang_A_y → indicará el ángulo de rotación de la pieza A (y también B y C) respecto del eje marcado por la pieza "mástil"

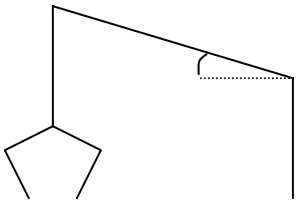
ang_A_z → indicará el ángulo de rotación de la pieza A (sólo de A , no de B ni C) respecto de la horizontal. Sí afectará a B y C porque cuando este ángulo sea mayor, B y C alterarán su posición, pero no "rotarán" lo marcado por ang_A_z (como sí ocurría con ang_A_y). Es decir, si incrementamos este parámetro queremos obtener esta forma para la grúa :



Partiendo de esta situación ,
Incrementando ang_A_z ➔



No buscamos esto....

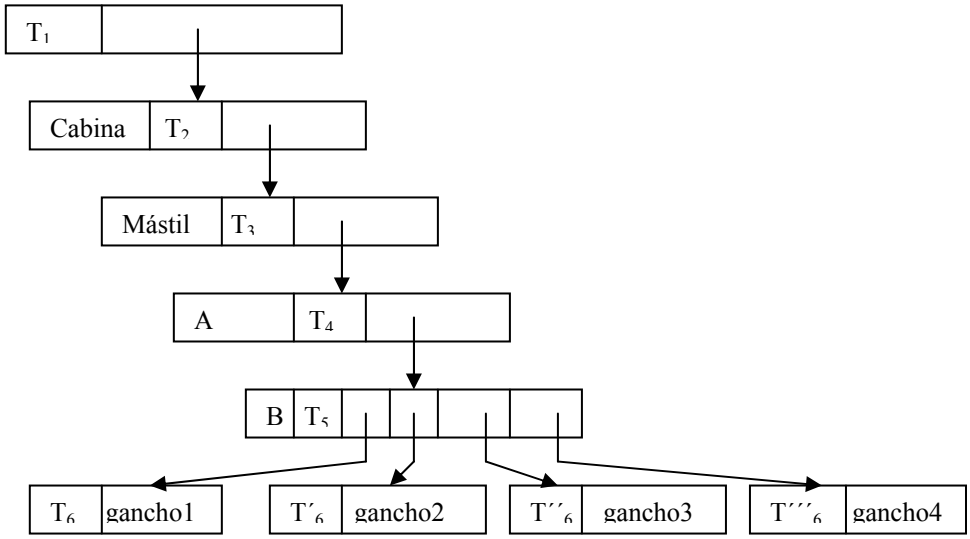


Sino esto otro.

Para ello, habrá que tener presente lo ya comentado, y además , hacer siempre
$\alpha = 90º$-ang_A_z

long_B → la longitud de la pieza B (afectará también a la pieza C puesto que esta pieza "cuelga" de la anterior) .
ang_C → 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 para la construcción jerárquica de la grúa:

T_1 → transformación genérica para toda la grúa.

T_2 → 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 → translación 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 → 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 → traslación para colocar los ganchos en el extremo de la pieza B

```
glTranslatef(0, -long_B+0.2 , 0);
```

T_6 → 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 de 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

1.2.- Cambios en la posición del observador.

Para definir distintas posiciones de observador, he empleado los siguientes parámetros : posición en la que se coloca el observador (V.R.P o view reference point), que representará el origen del sistema de coordenadas de vista; el vector que indica hacia donde mira el observador (V.P.N o Normal del plano de vista), siendo el sentido marcado por este vector, el opuesto hacia donde mira el observador; y la orientación hacia arriba, que define el sentido hacia arriba (V.UP o vector arriba de vista).

Dada una terna VRP, VPN , VUP, podemos definir el sistema de coordenadas de vista. Esto se realizará de la siguiente forma:

VRP es el origen del SC de vista
VPN es el eje Z del SC de vista, llamado n
El eje X del SC vista, llamado u, se obtiene : u= VUP@VPN donde @ indica la operación producto vectorial
El eje Y del SC vista, llamado v, se calcula como : v = VPN@u

Sin embargo, los objetos que dibujamos con OPENGL estarán definidos en base a otro sistema de coordenadas, llamado SC de mundo. Por lo tanto, para posicionar la cámara en la posición dada por VRP, orientada según VPN y VUP, (es decir, para visualizar el mundo representado actualmente según el SC de vista deseado), será necesario realizar la denominada "transformación de vista", es decir, un conjunto de transformaciones geométricas que permitan hacer coincidir el SC de mundo con el SC de vista. Para ello habrá que hacer coincidir los dos orígenes de los dos sistemas de coordenadas y luego alinear el eje n con Z, el u con el X, y el v con el Y:

Traslación del VRP al origen de SC de mundo
Rotación con respecto al eje X
Rotación con respecto al eje Y
Rotación con respecto al eje Z

Esta transformación de vista, dados los valores de VRP, VPN y VUP; la llevará a cabo la función "*transformación_vista*" definida en el código entregado. Esta función será llamada en la función "*Dibuja*" entre otras, antes de llevar a cabo la llamada a las primitivas para dibujar todo el modelo .

Se permite al usuario alterar la posición del observador, mediante el empleo de algunas teclas del teclado. He implementado con ellas ciertos movimientos de cámara interesantes, que permitan al usuario moverse cómodamente por el modelo. Así la pulsación de una de estas teclas, alterará el valore de los parámetros que definen el SC de vista (VRP, VPN o VUP), y por lo tanto, se obtendrá un nuevo SC de vista. Cada tecla alterará de la forma conveniente aquellos parámetros que se necesiten alterar para llevar a cabo el movimiento de cámara anunciado.

Posteriormente se detallará en el manual de usuario cuáles son estas teclas, y los movimientos asociados a las mismas. Simplemente adelantar que he implementado el movimiento del observador a modo de cámara en helicóptero, de forma que el observador puede desplazarse hacia arriba, abajo, adelante y atrás , izquierda y derecha, sin dejar de mirar hacia donde estaba mirando, resultando sólo afectada la posición del observador. (sólo alteramos VRP , de forma conveniente en cada caso). Las teclas del

cursor (arriba, abajo, izquierda, y derecha) cambiarán la dirección hacia donde mira el observador. Para más detalles, ir al manual de usuario, y para observar la implementación de estos movimientos (cómo se alteran los parámetros de vista con cada uno de ellos) , ver la función "*movimiento_observador*" del código adjunto.

1.3.- Construcción del casco de un barco.

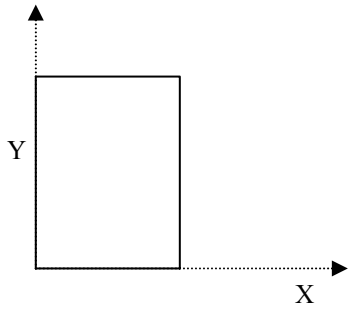
Me pareció útil diseñar en la práctica anterior, el casco de un barco para ahora acoplarlo al modelo. Ya fue entregado el boceto seguido para la construcción del casco de un barco como parte de esta práctica anterior. Únicamente comentar ahora, que he definido dos nuevas superficies a partir de la anterior, para construir la cubierta del barco. He empleado dos superficies para ello puesto que he dejado un hueco en mitad de la cubierta a modo de acceso a las bodegas que posteriormente emplearé para meter por ahí los bloques que la grúa cargue en el barco. Así he dividido la cubierta en dos mitades, y he definido una nueva superficie para cada una de estas mitades (cada una con 2 puntos en el plano U por 5 del plano V; orden 3 en el plano V por orden 2 en el U, e interpolando mediante un Bspline no periódico).

1.4.- Generador de Sólidos por revolución.

Para el diseño de ciertas piezas del modelo, principalmente cilíndricas, he utilizado un generador de sólidos por revolución sobre un eje, herramienta que ya había construido para la realización de unas prácticas anteriores (de la asignatura "Informática Gráfica" de 3º de Ingeniería Informática).

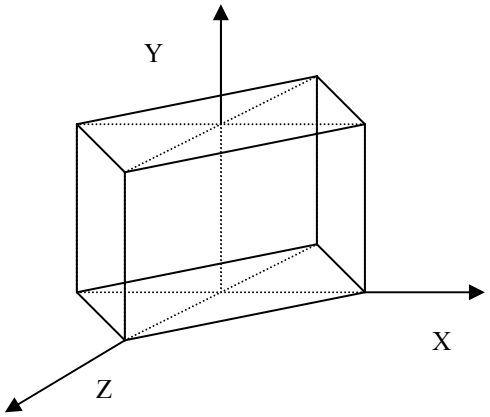
Este generador de sólidos trabaja de la siguiente forma: dado un vector de puntos, definidos sobre un plano cualquiera; un eje de rotación (X, Y o Z) alrededor del cual se desea generar el sólido por rotación; y dado el número de puntos para cada uno de los puntos anteriores que queremos obtener por rotación sobre los 360º (llamemos n a este nº de puntos) ; construye un sólido que resulta de la revolución en 360/n veces alrededor del eje dado, de la silueta definida por los puntos del plano dados al inicio.

Gráficamente, el algoritmo tomará como entrada lo siguiente :



Por ejemplo, le damos como entrada un vector con estos cuatro puntos del espacio, definidos sobre el plano X,Y; puntos que definen el rectángulo mostrado. Además le damos como eje de revolución el eje Y, y como valor de n el valor 4

Ante la entrada anterior el algoritmo generaría una matriz de puntos, que interpretada correctamente daría lugar al sólido siguiente :



Es decir, el algoritmo dará lugar a una matriz de puntos :

Rotación 0	Rotación 1	Rotación 2	Rotación 3
Punto1	Punto 1 rotado (360/n) grados alrededor de eje Y	Punto 1 rotado 2*(360/n) grados alrededor de eje Y	Punto 1 rotado 3*(360/n) grados alrededor de eje Y
Punto 2	Punto 2 rotado (360/n) grados alrededor de eje Y	Punto 2 rotado 2*(360/n) grados alrededor de eje Y	Punto 2 rotado 3*(360/n) grados alrededor de eje Y
Punto 3	Punto 3 rotado (360/n) grados alrededor de eje Y	Punto 3 rotado 2*(360/n) grados alrededor de eje Y	Punto 3 rotado 3*(360/n) grados alrededor de eje Y
Punto 4	Punto 4 rotado (360/n) grados alrededor de eje Y	Punto 4 rotado 2*(360/n) grados alrededor de eje Y	Punto 4 rotado 3*(360/n) grados alrededor de eje Y

Podemos, a partir de la matriz anterior, definir las aristas del sólido como las líneas que irán desde el punto1 al punto1 tras la rotación 1 , desde el punto1 tras la rotación 1 al punto 1 tras la rotación 2, y así sucesivamente, y de la misma forma para los otros puntos. Y de la misma forma, definir las caras, con sus normales correspondientes. Por ejemplo, podríamos tener una cara cuyos 4 vértices serían el punto1, el punto2, el punto1 tras la 1ª rotación , y el punto2 tras la 1ª rotación.

En el modelador implementado, a partir de esta matriz de puntos hemos construido la geometría del objeto, definiendo caras triangulares entre los puntos de la matriz en lugar de cuadradas como en el ejemplo anterior.

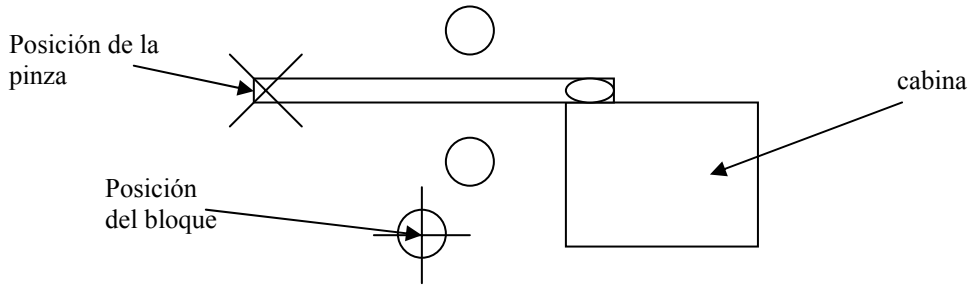
Para construir un objeto con una superficie curva, lo aproximaremos mediante la creación de un objeto utilizando un número de revoluciones muy alto sobre el eje. Por ejemplo, para construir un cilindro emplearemos la “plantilla” de puntos rectangular mostrada arriba, ordenaremos que la rotación se haga también sobre el eje Y, y daremos a n un valor alto, por ejemplo 200, para que la aproximación poligonal a nuestro cilindro sea medianamente buena.

1.5.- Selección de un bloque.

Para desarrollar la acción automatizada que comentaremos a continuación, hemos dotado a nuestro programa de capacidad de interacción con el usuario, además de en el movimiento de cámara ya comentado, en la posibilidad para éste de seleccionar con el ratón uno de los bloques que aparecen en la escena. De esto se encarga nuestra función "*pick*": capturar un hit del ratón sobre uno de los bloques presentes en la escena y ordenar, si todo ocurrió sin error, la recogida por parte de la grúa del bloque seleccionado. Para ello hacemos uso de las posibilidades que nos ofrece OPENGL para la selección. He procedido para ello tal y como lo hicimos en la correspondiente práctica anterior, y por lo tanto, no será necesario comentar nada del proceso, excepto, únicamente destacar que en el redibujado del modelo realizado tras entrar en modo selección, sólo se redibujan los bloques y el muelle. Los bloques se dibujaran porque son los elementos que nos interesa para la selección (el usuario sólo podrá seleccionar elementos de este tipo), y el muelle por cómo hemos implementado posteriormente la obtención de los hits tras salir de modo selección (siempre tomaremos como resultado de la selección el segundo objeto seleccionado , empezando desde el fondo hacia primer plano, y por lo tanto para poder seleccionar un bloque, éste tendrá que tener detrás sólo el muelle). Con el dibujado de estos dos tipos de objetos (una parte mínima de nuestro modelo), ganamos en eficiencia, ya que el redibujado del resto del modelo no es necesario para nuestro proceso de selección .

1.6.- Automatización del movimiento de recogida de un bloque previamente seleccionado.

Como ya mencionamos anteriormente, hemos incorporado a nuestro programa la posibilidad de que el usuario seleccione uno de los bloques que aparezcan sobre el muelle, para que la grúa lo coja automáticamente y lo meta en la bodega del barco. Para ello, partiremos siempre de la posición siguiente : (vista de la grúa desde arriba)



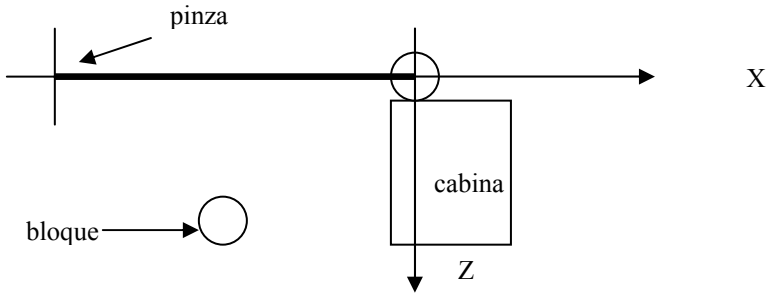
Para llevar a cabo esta tarea, no nos bastará con mover la grúa según un modelado jerárquico que rija dicho movimiento, sino que además será necesario la realización en paralelo de diversos cálculos, que empleando la posición de la grúa y la del bloque (tendremos que tener por lo tanto, ambas posiciones almacenadas en variables) nos indiquen si hemos de girar la grúa hacia un lado, hacia otro, subir la pinza, bajarla , etc...(es decir, que nos guíen en el movimiento de la grúa) . Por lo tanto, necesitaremos realizar cálculos en base a las posiciones mencionadas, que nos marquen el siguiente movimiento a realizar, para luego realizar dicho movimiento (redibujando la nueva imagen de la grúa, con sus parámetros de movimiento correctamente alterados) y recalcular la posición de la pinza de la grúa tras dicho movimiento realizado, para continuar con el proceso.

Por lo tanto, en primer lugar debemos calcular cómo colocar la pinza sobre el bloque que queremos coger (para poder cogerlo), dadas la posición del bloque, la posición de la pinza, y un conjunto de posibles operaciones a realizar sobre la posición de la pinza , que se corresponderán con los movimientos de que hemos dotado a nuestra grúa con la creación de los parámetros ang_A_y, ang_A_z, long_B y ang_C.

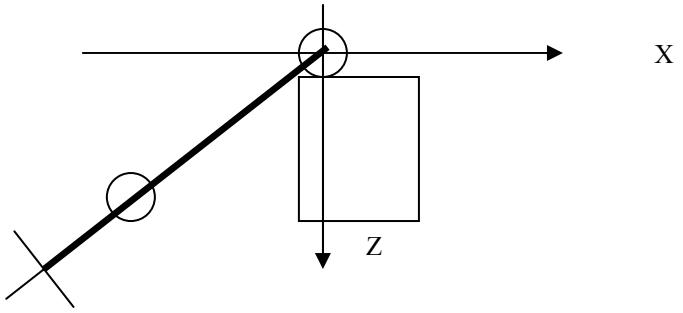
Para simplificar estos cálculos, hemos hecho coincidir en nuestro modelo, el eje Y del sistema de coordenadas con el eje central del mástil de nuestra grúa.

Para llevar a cabo esta maniobra de acercamiento, haremos que cuando se seleccione un bloque, se defina como función glutIdleFunc de OPENGL una función llamada "*ir_a_por_bloque*" que realizará en cada una de sus llamadas, un movimiento más de acercamiento hacia el bloque seleccionado, llamándose por lo tanto, al final de cada ejecución de esta función , a la función glutPostRedisplay de OPENGL para que redibuje el modelo, y los pequeños cambios se vayan apreciando , lográndose el efecto de animación que buscamos.

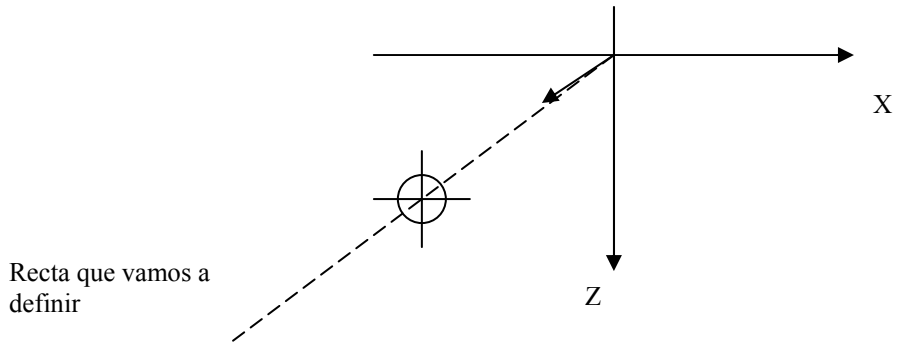
Para mover la pinza hasta colocarla sobre el bloque, partiendo de esta posición inicial:



Buscamos alcanzar esta otra posición :



Este será el primer cálculo que llevaremos a cabo. Tendremos que hacer girar el brazo de la grúa (la pieza A y todo lo que cuelga de ella) respecto del eje Y, objetivo que conseguiremos alterando el parámetro ang_A_y. A la vez, iremos recalculando la posición de la pinza tras las sucesivas rotaciones, ya que será esta posición la que nos indique cuando hemos logrado nuestro objetivo. Debemos establecer por lo tanto un mecanismo que partiendo de la posición de la pinza nos indique cuándo hemos llegado a la situación descrita arriba. Esto lo lograremos definiendo una recta en forma implícita, de la forma :



Para definir esta recta (que llamaremos recta 1) en el plano X,Z necesitamos un punto, que pudiera ser el origen de coordenadas, y un vector, que será el vector dado por las componentes en X y Z de la posición del bloque. Vamos a definir la recta en forma implícita porque tal y como vimos en teoría, esta forma nos permitirá evaluar fácilmente si un punto dado (en nuestro caso este punto será la posición de la pinza) se encuentra por encima o por debajo de la recta que es lo que nosotros deseamos, para determinar el movimiento a realizar por la grúa.

La recta 1 será por lo tanto de la forma Ax+By+C = 0; donde empleando los parámetros : pto(0,0); $\vec{u} = (posicion\_bloque.x, posicion\_bloque.z)$, obtenemos que :

A = posición_bloque.z
B = - posición_bloque.x
C = - (A*pto.x) – (B*pto.z)

De esta forma, dada la posición de la pinza (pos_pinza), evaluaremos empleando la recta1 si la pinza está por encima o por debajo de dicha recta, es decir, evaluaremos A*pos_pinza.x + B*pos_pinza.z + C , y según el resultado:

1º- Alteraremos el valor del parámetro que controla en el modelo el ángulo de rotación respecto al eje vertical (ang_A_y), incrementándole o decrementándole un grado, según el resultado obtenido, para que en el posterior redibujado de la imagen aparezca la grúa modificada, y en sucesivas llamadas a esta función se siga variando dicho ángulo, en un grado cada vez, hasta que nos posicionemos en la situación mostrada anteriormente, de manera que dé la impresión de que la grúa está girando buscando el bloque.

2º- Alteraremos el valor de pos_pinza, aplicándole los cálculos correspondientes a la transformación geométrica aplicada arriba (en este caso , rotación sobre el eje Y de +-1 grado).Así iremos actualizando la pos_pinza en concordancia con el movimiento que estamos haciendo con la grúa, para poder realizar más cálculos en siguientes iteraciones.

En este caso concreto, si obtenemos resultado > error ➔

- ang_A_y--;
- Modificamos el punto pos_pinza, que pasa a valer el punto que resulta de rotar pos_pinza –1º sobre el eje Y , es decir:

```
aux.x = pos_pinza.x ; aux.z = pos_pinza.z;
pos_pinza.x = aux.x* cos(-un_grado) + aux.z * sin(-un_grado);
pos_pinza.z = -aux.x * sin(-un_grado) + aux.z * cos(-un_grado);
```

donde un_grado representa π/180 radianes. Observar que pos_pinza.y queda inalterado porque estamos rotando respecto del eje Y.

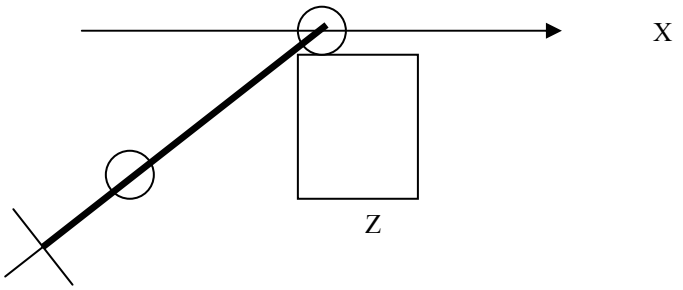
Si obtuviéramos resultado < - error ➔

- ang_A_y++;
- Modificamos el punto pos_pinza, que pasa a valer el punto que resulta de rotar pos_pinza 1º sobre el eje Y , es decir:

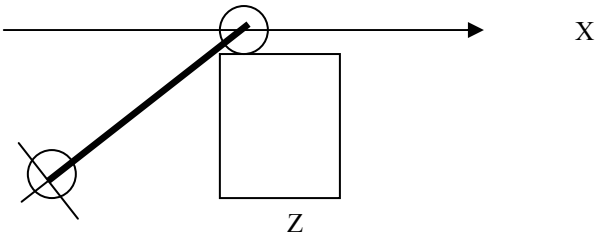
```
aux.x = pos_pinza.x ; aux.z = pos_pinza.z;
pos_pinza.x = aux.x* cos(un_grado) + aux.z * sin(un_grado);
pos_pinza.z = -aux.x * sin(un_grado) + aux.z * cos(un_grado);
```

En ambas expresiones empleamos un margen de error dado por la variable error.

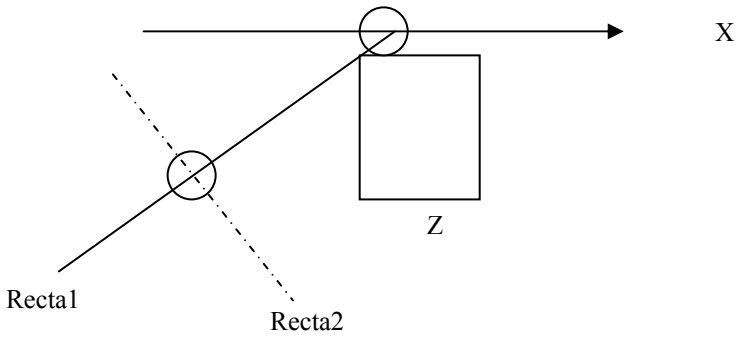
De esta forma, en cada llamada a la función, iremos haciendo que A*pos_pinza.x + B*pos_pinza.z + C sea cada vez más próximo a 0. Cuando el resultado dado por esta expresión esté acotado entre +error y –error, consideraremos que la aproximación es suficientemente buena, y daremos por concluido el primer paso. Ya tendremos la grúa respecto del bloque en esta posición:



Ahora interesará que el punto que marca el centro de la pinza quede en la misma vertical que el punto que marca el centro del bloque. Es decir, en esta segunda fase buscaremos la situación :



Para encontrar ahora un mecanismo que nos indique, dada la posición de la pinza, si hemos de subir más la inclinación de la pieza A de la grúa (alterando el ángulo ang_A_z), vamos a definir, como hicimos anteriormente una recta en forma implícita (recta2) que pase por :



Para definir esta recta necesitamos de nuevo, un punto y un vector. El punto puede ser la posición del bloque (pos_bloque) , y el vector podría ser un vector perpendicular al vector dado por la posición del bloque, definido en el plano X,Z; por ejemplo el vector $\vec{u} = (\frac{1}{pos\_bloque.x}, \frac{-1}{pos\_bloque.z})$ es perpendicular al vector $\vec{v} = (pos\_bloque.x, pos\_bloque.z)$ que es el vector asociado a la posición del bloque en el plano X,Z. Por lo tanto, empleando el vector u, y el punto en el que se encuentra el bloque, definimos recta2 :

Ax+By+C = 0 donde:
A = -1/pos_bloque.z
B = -1/pos_bloque.x
C = -(A* pto.x) - (B * pto.z);

donde pto= (pos_bloque.x, pos_bloque.z)

Tal y como hicimos anteriormente, evaluaremos la expresión de la recta2 en el punto dado por la posición de la pinza, y en función del resultado, incrementaremos o decrementaremos ang_A_z; llevando a cabo un cálculo equivalente para la posición de la pinza. En el caso anterior, este cálculo equivalente era muy simple. Únicamente teníamos que calcular el nuevo punto para la posición de la pinza obtenido al rotar 1º o –1º sobre el eje Y . Ahora tendremos que hacer lo mismo pero rotando 1º o –1º sobre el eje Z. El problema ahora es que no podemos aplicar directamente las expresiones para rotación con el eje Z desde la posición en la que nos encontramos, sino que primero tendremos que trasladar el punto pos_pinza al plano Y,X para rotarlo respecto de Z. Por lo tanto tendremos que realizar las siguientes operaciones sobre pos_pinza:

1.- traslación –long_B en Y , para rotar respecto del origen de coordenadas como si rotáramos respecto del pto. de rotación del mástil de la grúa (el extremo superior del mástil de la grúa) .

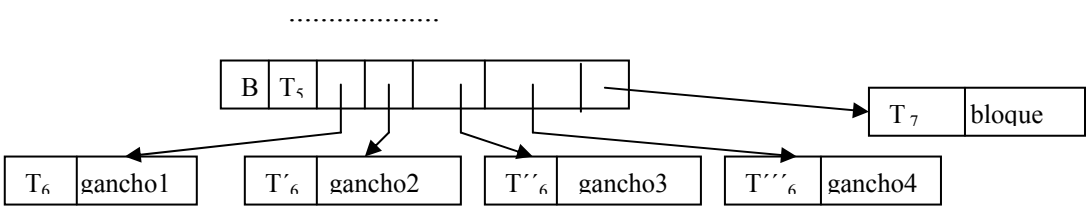
2.- rotación en Y –ang_A_y, para poner pos_pinza en el plano X,Y y poder ahora rotarlo sobre Z
3.- rotación en Z de –1 grado
4.- rotación en Y de ang_A_y , para deshacer la rotación del paso 2 anterior
5.- traslación long_B en Y, para deshacer la traslación del paso 1 anteior.

Esta descripción se corresponde al caso de que el resultado de la evaluación de la recta2 para pos_pinza sea mayor que el error permitido , y entonces tengamos que subir la pinza, aumentando la inclinación de la pieza A , y haciendo ang_A_z ++. Para el caso contrario (que haya que bajar la pinza) , todo sería igual excepto que se rotaría en el paso 3 en Z 1 grado, en lugar de –1 grado. Y se haría ang_A_z --.

Llegados a este punto, ya tendremos la pinza sobre el bloque, con un cierto error, y sólo quedará extender el cable para hacer que la pinza quede justo sobre el bloque, para poder cerrar los ganchos de la misma y coger el bloque. Para ello, hacemos que long_B se incremente hasta llegar a una longitud máxima. Ya no consideramos necesario seguir actualizando la variable pos_pinza, puesto que en lo que resta para finalizar la operación no la necesitaremos. Una vez que long_B llegue a un límite superior establecido, procederemos al cierre de las pinzas, incrementando grado a grado la variable ang_C que controla el grado de abertura de las pinzas. Iremos haciendo esto en sucesivas llamadas a ir_a_por_bloque, y al final de cada una de ellas, llamaremos a glutPostRedisplay para redibujar el modelo e ir visualizando los pequeños cambios que sobre el se van realizando, dando el efecto de animación que buscábamos.

Cuando long_B haya alcanzado el valor fijado, y ang_C también, podremos considerar que el bloque ya ha sido apresado por la pinza de la grúa. A partir de este punto, sería deseable que el bloque formara parte de la grúa, dibujándose como si fuera una pieza más del modelo de la misma. Así facilitaremos el redibujado, ya que en otro caso tendríamos que ir calculando cómo se va modificando la posición del bloque con los movimientos de la grúa, e ir dibujándolo de forma independiente a ésta, cometiendo errores que pueden llegar a ser visibles y además complicando mucho los cálculos en cada redibujado mientras la grúa lleve el bloque hacia el barco.

Por lo tanto, he adoptado la solución de que , antes incluso de que las pinzas se cierren para coger el bloque, estando ya la pinza justo sobre éste, el bloque pase a ser parte implícita del modelo jerárquico de la grúa, dibujándose después de hacerlo los ganchos de la misma, y viéndose afectado por todas las transformaciones, exceptuando las que afectan a cada gancho en particular. El último nodo del árbol jerárquico con el que he modelado la grúa, pasará a ser, cuando tengamos cogido un bloque el siguiente:



Donde T_7 será simplemente una traslación para poner el bloque a la altura correcta .

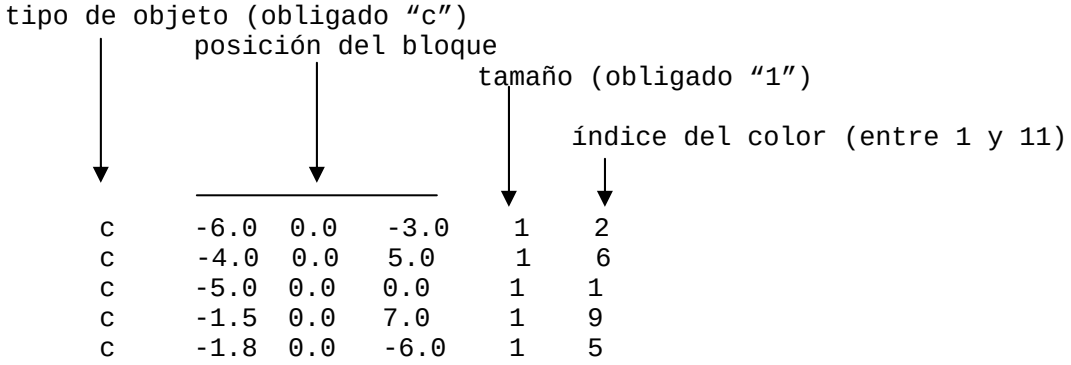
De esta forma, sólo tendremos que ir alterando los parámetros de control de la grúa (ang_A_y, ang_A_z, long_B, ang_C) de manera apropiada para acabar con el bloque dentro del barco. De esto se encargará la nueva función glutIdleFunc que definimos cuando "*ir_a_por_bloque*" logre acabar con éxito su misión. Esta función se llamará "*soltar_bloque"* y simplemente irá modificando convenientemente los parámetros mencionados para dejar el bloque en la bodega del barco.

Durante este proceso, el bloque llevado por la grúa no será dibujado como el resto de bloques del modelo, sino como parte de la propia grua, como ya hemos comentado. Al ser dejado dentro del barco, eliminaremos dicho bloque, haciendo que este deje de visualizarse, y llevaremos la grúa de nuevo , a la posición de partida para que el usuario pueda ordenar la recogida de otro bloque.

2.- Manual de usuario.

- **Ejecución del programa.**

Para la ejecución del programa, teclee “grua <fichero de datos>”, donde <fichero de datos> ha de ser un fichero de texto de la forma siguiente (puede ver el formato abriendo el fichero “entrada.dat”):



Cada línea se corresponde con un objeto. En principio sólo se admiten bloques, por lo tanto, todas las líneas deben comenzar con “c”(“c” de caja). Los tres números reales siguientes establecen la posición de cada bloque. El usuario es responsable de la posición dada a los bloques, ya que por ejemplo, dado que la pieza A de la grúa tiene una longitud de 9, no podrá recogerse ningún bloque que se encuentre más allá de estas dimensiones. Asimismo, tampoco tendría sentido colocar un bloque con una componente Y para su posición que sea distinta de 0.0 (se ha establecido el 0.0 para colocar el bloque justo sobre el muelle, un valor distinto lo hará “flotar” en el aire o hundirse dentro del muelle). De la misma forma un valor positivo para X haría que el bloque se saliese del muelle. Por lo tanto se recomienda no alterar en principio el valor de estos parámetros hasta haber ejecutado alguna vez el programa.

La cuarta columna de cada fila indicará el tamaño normalizado del bloque. Sólo se ha implementado el programa para bloques de tamaño normalizado igual a 1 , luego se recomienda no cambiar esta cifra (se ha mantenido para futuras ampliaciones del programa) .

El último número es un entero perteneciente al intervalo [1,11] que indicará el color asignado al bloque (hay 11 colores preestablecidos para los bloques).

- **Modos de ejecución.**

Una vez que el programa se esté ejecutando, pulsando el botón derecho del ratón visualizará un menú en el que podrá elegir entre las acciones :

"Exit"→ finaliza la ejecución del programa (también se produce si en cualquier momento durante la ejecución pulsa la tecla 'q' o la tecla 'Esc')

"Modo recogida de bloques" → entra en modo recogida de bloques. El programa cargará del fichero la configuración de bloques dada y el usuario podrá seleccionar (haciendo click con el botón izquierdo del ratón sobre el bloque que desea recoger) el bloque que desea que la grúa recoja automáticamente.

"Modo manual" → En este modo no se visualizan los bloques. Se le permite al usuario mover la grúa manualmente, mediante los controles que se indican a continuación:
'y' , 'Y' : girar la grúa sobre el mástil que la sostiene
'z', 'Z' : aumentar o disminuir la inclinación de la pieza A de la grúa
'c', 'C' : aumentar o disminuir la longitud de la cuerda de la que cuelga la pinza.
'd', 'D' : cerrar o abrir la pinza
estos controles estarán deshabilitados en "Modo recogida de bloques"

- **Movimiento de Cámara.**

En cualquier modo de ejecución, y cualquier momento, el usuario puede cambiar la posición de la cámara. Estos son los controles habilitados a tal efecto :

- Cursor arriba, abajo ,izquierda, derecha : cambia la orientación de la cámara, haciendo que ésta pase a mirar más arriba, abajo, hacia la izquierda o hacia la derecha.
- 'o', 'l' : incrementa o decrementa la posición vertical del observador
- 8, 2, 4, 6 : avanza recto, retrocede, avanza hacia la derecha o hacia la izquierda la cámara, manteniendo la orientación hacia la que mirábamos .

Recomendación : para los movimientos de cámara, active "Bloq Num" de su teclado y emplee el teclado numérico de la derecha para las teclas 8,2,4 y 6. Así manejará la cámara de manera más cómoda e intuitiva.