Diseño Asistido por Computadora

Proyecto: El cubo de Rubik

Polo García, José Luis
4º Ingeniería Informática

Índice de contenidos

1. Descripción del problema

El objetivo de la práctica es realizar un programa en openGL que simule el comportamiento de un cubo de Rubik. Las funciones que se deben implementar son:

- Cambiar la posición de los componentes del cubo girando planos de 9 cubos en torno al eje X, Y o Z. Todo ello mediante el uso del ratón.

- Desordenar la estructura del cubo de forma que se aprecie los giros que se realizan en el proceso.

- Almacenar o cargar partidas anteriores del juego.

- Presentar un menú para el cambio de parámetros: número de movimientos aleatorios al desordenar, nombres de fichero para el guardado rápido, aumentar/disminuir zoom, abandonar el programa, etc...

- Otra funciones de cámara: permitir cambiar la posición de la cámara en todos los sentidos, incrementar o disminuir el zoom de la imagen, etc...

2. Estructuras de datos

Dos estructuras de datos se usan principalmente. La primera de ellas es un array de 3 dimensiones llamado *cuboRubik* que guarda 3x3x3 enteros. Los enteros en realidad son los identificadores de los cubos que ocupan la posición correspondiente. Así, por ejemplo, en cuboRubik[i][j][k] se guarda el identificador del cubo que ocupa la posición i, j, k con respecto a los ejes x, y, z respectivamente.

Gracias a esta estructura podremos comprobar fácilmente si se ha llegado a una solución, así como realizar los giros de los cubos a la vez (se giran "planos" de 9 cubos cada vez que se indica el giro de un cubo).

Por otro lado tenemos la estructura *model*, que es un vector que guarda las figuras que se usan en el modelo. En este problema se encuentran 27 figuras (cada uno de los cubos individuales), que se estructuran de forma conveniente. *model* es un vector de estructuras figura, en la que se guarda toda la información necesaria para cada cubo.

Los campos que componen dicha estructura son:

```
typedef struct {

   float x, y, z;
   float giroX, giroY, giroZ;
   float AntX, AntY, AntZ;

   int colores[6];

} figura;
```

En primer lugar tenemos las variables x, y, z. Guardan la posición física en el espacio de cada cubo. Por otro lado también será necesario guardar los giros que hay que aplicar a cada elemento. Esto es así porque cuando realizamos un giro (ya sea de forma manual o por el procedimiento de desornamiento aleatorio) no sólo cambia la posición de los cubos sino también la orientación que tienen y es necesario guardala para luego aplicar la transformación correspondiente.

Las variables AntX, AntY y AntZ guardan la posición anterior que ocupaba el cubo, pero la posición física. Es necesario almacenarla porque luego el comportamiento del programa es el siguiente: Cuando realizamos un giro con el ratón por ejemplo según movamos más o menos el cubo gira y cambia su posición y orientación adecuadamente. Cuando soltamos el ratón la posición de los cubos seleccionados tiene que ser ajustada a una posición posible, esto es, si hemos realizado un giro de 83º con respecto a la posición original, habrá que ajustar a 90º. No es posible que los cubos se queden en posiciones intermedias porque luego, al girar con respecto a otro eje, la estructura se deformaría. Del mismo modo en un giro que finalice con un incremento de 163ª será ajustado a 180 y así para todo los casos según corresponda (ver función *ajustarMov*).

Por último tenemos un vector de enteros que se llama *colores* que guarda los colores que tienen las

caras del cubo de la siguiente forma. La cara 0 es la superior y la 5 la inferior siendo de la 1 a la 4 las caras laterales. Los colores son valores enteros definidos como constantes cuyos valores posibles son BLANCO, AZUL, NARANJA, ROJO, AMARILLO, VERDE y NEGRO. La función *ponColor* es la encargada de asignar estos valores lógicos de colores a valores aceptables para openGL.

3. Funciones empleadas

Las funciones más importantes que se usan en el programa se describen a continuación:

```
void genDatos();
```

Se encarga de asignar los valores iniciales a las estructuras usadas. La posición inicial del cubo será la de un cubo resuelto por motivos didácticos (para comprobar si se llega a una solución fácilmente).

```
void inicia ();
```

Inicia los parámetros openGL para el modelo y habilita control de profundidad, color de borrado, etc...

```
void dibuja();
```

Función que se encarga de dibujar el modelo. Es un bucle que recorre la estructura *model* y llama a las primitivas de dibujo adecuadas para pintar el modelo.

```
void ejes (int grosor);
```

Se encarga de pintar unos ejes de distintos colores que parten del origen (0,0,0). El parámetro es el grosor de la línea que se usa para dibujarlos.

```
void ponColor (int color);
```

Se le pasa un color lógico entre los valores posibles (NEGRO, AZUL, NARANJA, VERDE...) y la función se encarga de llamar a glColor3f con los parámetros adecuados.

```
void cubo (float tam, int c[6]);
```

Primitiva para pintar un cubo. Se le pasa el tamaño del cubo que se quiere y los colores que tienen que llevar sus caras en forma de vector de enteros.

```
teclaEspecial, tecla, raton, movRaton
```

Son las funciones con las que manejar los eventos de teclas especiales, teclas normales, ratón y movimiento activo de ratón (con botón pulsado) respectivamente. Se asignan a las funciones glut en el programa principal adecuadamente.

```
int pick (int x, int y);
```

Función para la selección de objetos en el modelo. Recibe la posición actual del ratón y devuelve el identificador del cubo que fue seleccionado.

```
void lugar (int id, int *a, int *b, int *c);
```

Recibe como entrada el identificador de un cubo y almacena en (a, b, c) la posición lógica que ocupa dentro del cubo de Rubik.

```
void gestMovCubo (int d, int s, int id);
```

Recibe como argumentos el identificador de un cubo (id), la dirección y el sentido en que se produce el giro y gestiona el movimiento de los cubos adecuados. Es la función que controla el movimiento de los cubos de acuerdo a los movimientos del ratón: recalcula la posición del plano de cubos que se mueve según si el ratón se mueve de arriba a abajo, de izquierda a derecha o con respecto al eje z (con el botón derecho).

```
void giraCubo (int id, char eje, float grad);
```

Recalcula la posición del cubo que se pasa como argumento de acuerdo al eje de giro que se quiere y la cantidad de grados necesaria.

```
void ajustaMov (int id);
```

Tras soltar el ratón se llama a esta función que se encarga de ajustar los giros a posiciones "estables" adecuadas, esto es, los cubos siempre tienen que acabar formando un cubo tras los giros. Los valores de giros posibles tienen que ser 90, 180, 270 o 0 grados ya que los cubos no se pueden quedar a medio giro.

```
void ajusteLogico (int d, int s, int fc, int numInc);
```

Esta es la función responsable de modificar los identificadores que guarda la estructura *cuboRubik* tras producirse un giro. Es necesario conocer la dirección y el sentido en que se produjo (d, s). La variables fc es la dimensión que no cambió en el giro y por tanto determina el eje en que se produzco, dicha variable resulta de gran utilidad en la función. El número de incrementos (numInc) es el número de giros posibles que se produjo, esto es, si vale 1 quiere decir que se giraron 90º, si vale 2, 180º y así sucesivamente.

```
int siguientePos(int v[4][2], int i, int j, int f, int n);
```

Función interna que calcula la siguiente posición que debe ocupar un cubo de acuerdo a la posición actual que ocupa (i, j) y el número de giros que se produjo. La variable f vale -1 o 1 según el sentido en quiere cambiar la posición.

```
int solucion();
```

Función booleana que devuelve TRUE si el cubo ha sido resuelto y FALSE en caso contrario. Es un bucle que recorre las caras de los cubos para ver si coinciden.

```
void cuboResuelto();
```

Escribirá un mensaje para avisar de que se llegó a la solución.

```
salvarFichero, cambiarFicheroSalida, cargarFichero
```

Controlan los aspectos relativos a guardar una partida o recuperarla. La función *cambiarFicheroSalida* se encarga de alterar el nombre de la partida actual con el fin no sobreescribir el fichero actual.

```
void desordenaCubo();
```

Se encarga de generar aleatoriamente movimientos y de ejecutarlos visualmente.

```
void movAleat(int idCubo, int nRot, int sent, int eje);
```

Altera el modelo gráfico para visualizar cómo se producen las rotaciones en el cubo.

4. Manual de usuario

El programa se ejecuta sin argumentos:

```
$> rubik
```

Se abre una ventana donde vemos el cubo de rubik, que inicialmente está ordenado. Con el botón central del ratón podemos activar un menú que nos muestra las opciones posibles. Si no se dispone de 3 botones para el ratón se pueden pulsar simultáneamente el izquierdo y el derecho. (En ratones pulsar la rueda)

Para mover los cubos usamos el ratón de la siguiente forma: Pulsamos en el cubo que queremos mover y desplazamos el ratón en algún sentido. Si pulsamos el botón izquierdo y movemos de abajo a arriba los cubos adecuados giran con respecto al eje Z. Si usamos el botón izquierdo pero movemos horizontalmente los cubos giran con respecto al eje Y. Si usamos el botón derecho y movemos verticalmente entonces conseguiremos un giro con respecto al eje Z.

Por otra parte con los cursores podemos cambiar la posición del cubo girándolo sobre su centro.

Las opciones que se presentan en el menú son las siguientes:

- Más zoom : incrementa el tamaño del cubo
- Menos zoom : decremente el tamaño del cubo

- Desordenar: realiza una serie de movimientos aleatorios en cualquier sentido y con respecto a cualquier eje (sin repetir dos ejes consecutivos). Por defecto y de inicio se realizan 10 iteraciones aleatorias, parámetro que se puede modificar con la siguiente opción del menú.

- Cambiar desorden: Pedirá por consola el nuevo valor para las iteraciones aleatorias.

- Cambiar fichero de salida: En principio el archivo sobre el que jugamos se llama "noname.cub", para cambiar esto llamamos a esta opción que nos pide por la consola también el nuevo nombre del fichero que se usará para guardar la información cuando se pida.

- Almacena el fichero de nombre actual que aparece en el título de la ventana. Se puede cambiar el fichero con la opción anterior.

- Abrir fichero: Carga un fichero anteriormente almacenado con el que se puede continuar jugando.

- Ejes: Activa o desactiva la visión de los ejes.

- Salir: Abandona el programa

Todas estas funciones se pueden llamar también usando el teclado, por si el botón central no existiese ni ninguna de las alternativas tampoco y para llamar a las funciones de forma más rápida. Los atajos de teclado existentes son:

- Escape: salir.
- Enter: desordenar cubo.
- Espacio: salvar el fichero actual.
- C : cambiar fichero de salida.
- A : cargar ficharo anteriormente salvado.
- W: incrementar zoom.
- S: decrementar zoom.
- D: cambiar número de iteraciones aleatorias para desordenar.
- E: activar / desactivar visión de los ejes.