Práctica 5 Transformada de vista y transformada de proyección.

La matriz de vista define la posición y orientación de la cámara respecto al mundo virtual. Modificando sus valores, podemos alterar la posición o dirección de la cámara virtual (o lo que es lo mismo, alteramos el centro y el plano de proyección respecto al sistema de referencia del mundo).

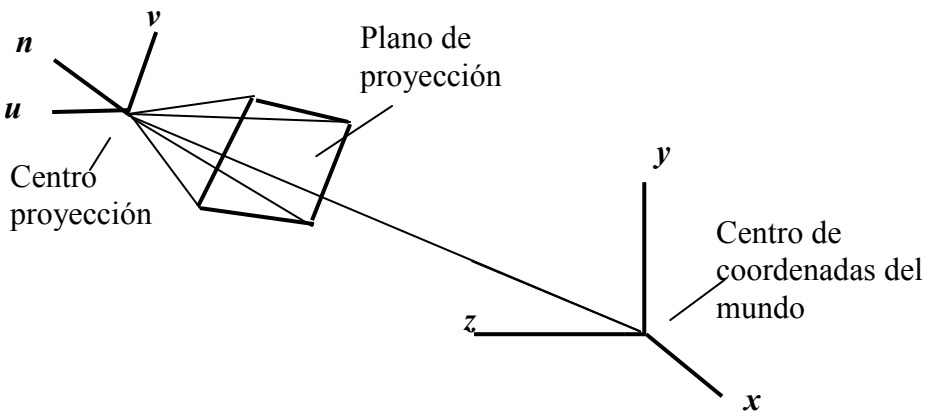

Figura 1

En la figura de arriba, los vectores ***n, u, v*** definen los ejes principales del sistema de referencia de la vista. Con estos vectores, además del vector posición de la cámara ***d***, se define la matriz de vista V. En su forma más general, dicha matriz esta formada por los siguientes elementos:

$$V = \begin{bmatrix} ux & uy & uz & dx \\ vx & vy & vz & dy \\ nx & ny & nz & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 2

Donde (*dx, dy, dz*) surgen de la multiplicación de la submatriz de rotación 3x3 de V por la matriz de traslación T que sitúa a la cámara en su posición. (V = R T)

Dentro de la OpenGL existe una función que permite modificar esta matriz de forma sencilla, tomando como parámetros la posición del centro de proyección , el punto de referencia situado sobre el eje ***n*** (vector ***ref***), y el vector ***up***. La función se llama gluLookAt y tiene el siguiente lista de argumentos:

```
gluLookAt(centrox,centroy,centroz,refx,refy,refz,upx,upy,upz);
```

Ejercicio 1

Vamos a dotar a la cámara de un movimiento orbital circular alrededor del origen (punto (0,0,0)).
Esta órbita será el meridiano que pasa por el punto (1,1,1).

El sistema de coordenadas más adecuado para expresar este problema es el sistema de coordenadas esféricas que se basa en tres parámetros: radio (**r**), ángulo polar **φ** y ángulo azimutal **θ**.
Para cualquier meridiano el ángulo polar φ es una constante, así como el radio. (Ver Figura 3)

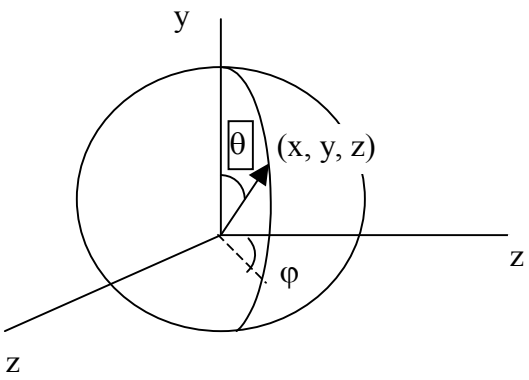

Figura 3

Las ecuaciones que describen la transformación a coordenadas cartesianas son las siguientes:
$x = r \sin\theta \cos\varphi$
$y = r \cos\theta$ (ecuaciones I)
$z = r \sin\theta \sin\varphi$

Apartado a) Movimiento de la escena alrededor del meridiano usando el ratón

- Construye una función que pasándole el punto del meridiano donde se encuentra la cámara, construya la matriz de transformación de vista (Figura 2) y la aplique a al pila de Modelview de manera adecuada. Prueba que funciona inciándola a la posición de cámara (1,1,1).

- Registra la función de gestión del evento correspondiente al movimiento del ratón por la pantalla con el botón izquierdo pulsado. Para ello usa el código

```
glutMotionFunc(mov_raton);
```

- La función `mov_raton(int x, int y)` debe hacer lo siguiente:

  - Debe determinar usando el valor de la coordenada *y* actual y el valor de la coordenada *y* de la posición anterior si el movimiento del ratón es hacia arriba o hacia debajo de la pantalla (CUIDADO: La *y* que recibe esta función tienen su origen en la esquina superior izda de la pantalla)

    - Si el movimiento del ratón es hacia arriba debe DECREMENTAR el valor del ángulo azimutal θ en 5 grados sexagesimales (NOTA: almacenar en θ el valor en radianes). Si el movimiento del ratón es hacia abajo debe INCREMENTAR el valor del ángulo azimutal θ en 5 grados sexagesimales.

- Calcular con el nuevo valor de θ la posición de la cámara en el meridiano usando las ecuaciones I

- Llamar con `glutPostRedisplay();` al redibujado.

Dinámica del programa en este apartado:

Definiremos varias variables globales; el `radio`, `tecta` y `phi` para las coordenadas esféricas y un vector `camara` de 3 componentes que almacena la posición de la cámara. Este vector es el que actualiza la función `mov_raton` .

Las variables `radio`, `tecta` y `phi` deben inicializarse a los valores de coordenadas cartesianas (1,1,1).

La función que calcula la matriz de vista es llamada dentro de `miDraw`. De esta manera calcula la matriz con la nueva posición de la cámara que ha calculado la función `mov_raton`. Situar también la tetera en la función `miDraw` así como los ejes cartesianos.
Pon la proyección a `glOrtho(-7.0,7.0,-7.0,7.0,-4.5,4.5);`

CUIDADO!!: En el cálculo de la matriz de vista estamos tomando como posición **up** el vector (0,1,0). Eso solo es cierto EN MEDIO MERIDIANO (correspondiente a los valores de θ entre 0 y 180 grados. Para la otra mitad del meridiano (Cuando estamos boca abajo) el vector **up** debe ser el (0,-1,0).

Apartado b) Movimiento orbital continuo.

Una vez tenemos hecho el apartado a, es muy fácil crear un movimiento continuo alrededor del meridiano. Basta introducir el trozo de código que actualiza la el valor de θ y los valores de la posición de la cámara dentro de la función `miIdle`. Mientras no hagamos nada se estará llamando continuamente a esta función.

Dinámica del programa en este apartado:

Vamos a activar este movimiento cuando pulsemos la tecla ‘o’ y vamos a pararlo cuando pulsemos la tecla ‘s’.

Para hacer esto definid una variable bandera que esté inicialmente a 0. Cuando pulsemos ‘o’ se debe poner a 1. Todo el código que hay dentro de miIdle debe estar dentro de un condicional que comprueba el valor de esta bandera. La tecla ‘s’ pone la bandera a 0 otra vez.

Apartado c) Proyecciones.

Dada la matriz de proyecciones de los apuntes:

$$M = \begin{bmatrix} 1 & 0 & -\frac{dx}{dz} & zp\frac{dx}{dz} \\ 0 & 1 & -\frac{dy}{dz} & zp\frac{dy}{dz} \\ 0 & 0 & -\frac{zp}{Q \cdot dz} & \frac{zp^2}{Q \cdot dz} + zp \\ 0 & 0 & -\frac{1}{Q \cdot dz} & \frac{zp}{Q \cdot dz} + 1 \end{bmatrix}$$

Preparad un función que active esta matriz en la pila GL_PROJECTION.

Como sabes, esta matriz ha sido calculada situando el plano de proyección perpendicular al eje z. Posiciona la cámara en este eje.

Visualiza la tetera y el cubo con estas proyecciones
a) `zp=0; Q=cte; dx=0; dy=0; dz=-1;`
La llamada a la función gluPerspective equivaldría a tomar dentro de la matriz de proyección c estos valores.

Para definir otras proyecciones paralelas distintas, como la caballera, tomamos:
b) `zp=0; Q=∞; dx=` $\cos(\alpha)$`; dy=`$\sin(\alpha)$`; dz=-1; (α = 45)`
O la de Gabinete, tomamos:
c) `zp=0; Q=∞; dx=`$\frac{\cos(\alpha)}{2}$`; dy=`$\frac{\sin(\alpha)}{2}$`; dz=-1; (α = 63.4)`

Dinámica del programa en este apartado:

Pulsando la tecla ‘p’ (de perspectiva) el programa cargará en la posición de la cámara los valores (0,0 –1); y luego cargará en la pila de matrices de proyección la matriz del apartado a)

Pulsando la tecla ‘c’ (de caballera) el programa cargará en la posición de la cámara los valores (0,0 –1); y luego cargará en la pila de matrices de proyección la matriz del apartado b)

Pulsando la tecla ‘g’ (de gabinete) el programa cargará en la posición de la cámara los valores (0,0 –1); y luego cargará en la pila de matrices de proyección la matriz del apartado c)

NOTA: Como esta matriz no contempla la transformación al volumen canónico, donde se hace el recorte, vamos a superponer estas proyecciones sobre una proyección “neutra” de tipo paralelo. Así pues la llamada será

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-7.0,7.0,-7.0,7.0,-4.5,4.5);
glMultMatrixf(M);   //M es la matriz construída por nosotros.
```