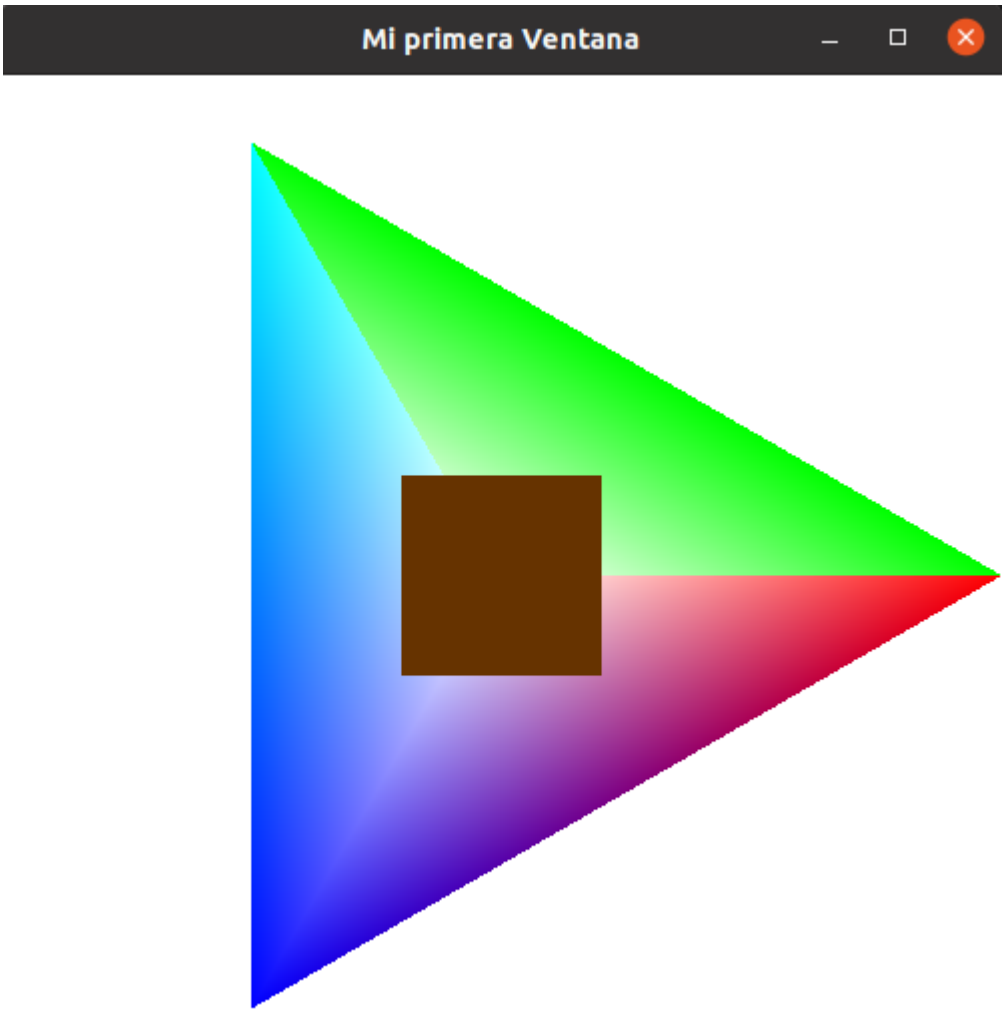
Documentación prácticas

Arnau Vidal 43467666N
Pere Joan Vives Morey 41620797C

Parte 1

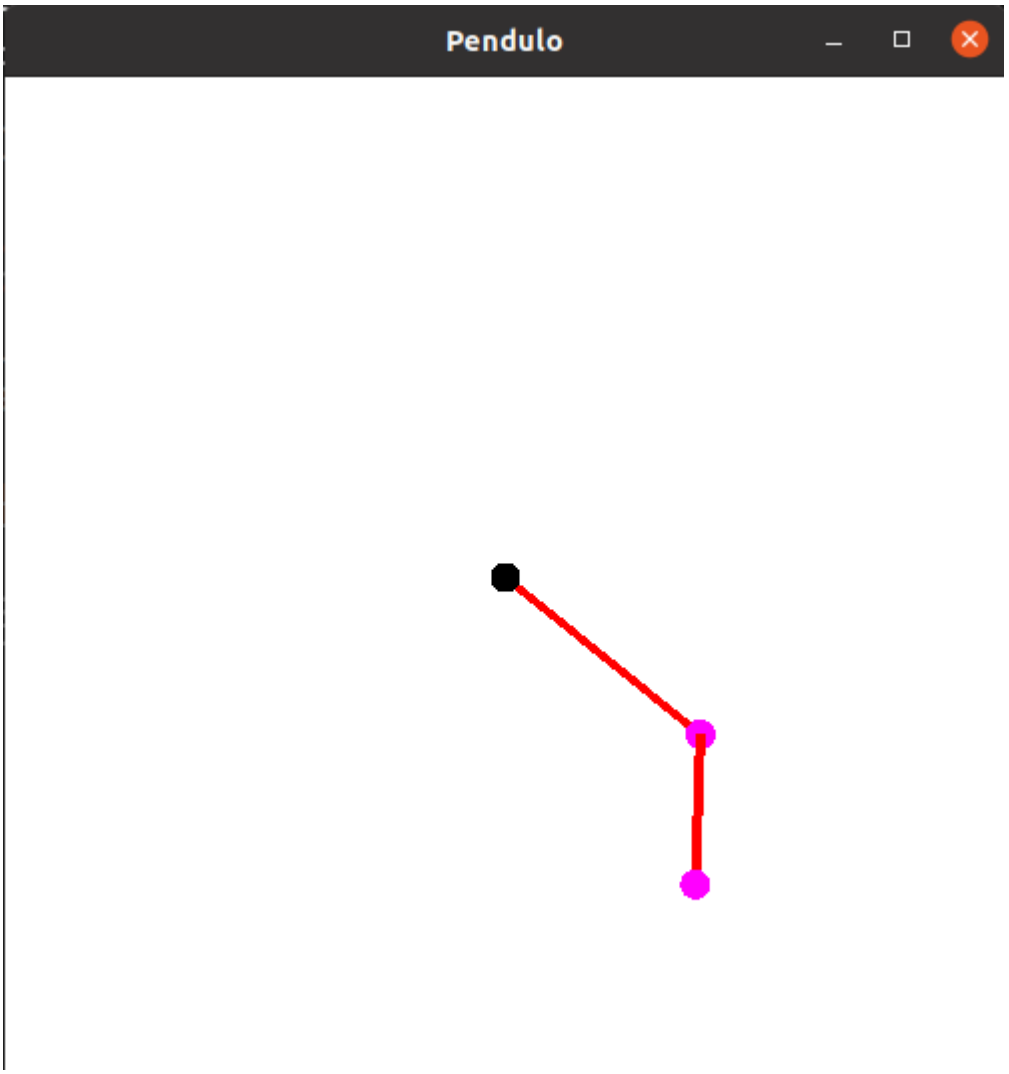


En esta primera parte hemos pintado la siguiente primitiva y hemos añadido los siguientes métodos y características:

1) Doble buffer para evitar el parpadeo
2) Una función reshape la cual se va a llamar cada vez que las dimensiones de la ventana se cambien. Para evitar la distorsión de la figura adaptamos los parámetros del glOrtho() a partir de la relación de la nueva altura y amplitud de la ventana.

3) Un método que reacciona a los botones del ratón: glutMouseFunc(mouse). En el caso de que pulsemos el botón central del ratón, se va a llamar a un método que asignamos a glutIdleFunc() el cual va a hacer girar la figura que vemos. Para hacer esta rotación se incrementa el ángulo inicial de rotación de la figura. Si se llega al máximo ángulo (360), empezaremos de 0. Si clicamos el botón derecho o izquierdo del ratón, se va a desplazar la figura hasta la derecha o izquierda respectivamente.

Parte 2



Como se puede apreciar en esta etapa se ha dibujado un péndulo simple dentro de un espacio bidimensional. Este péndulo no es más que la unión de tres puntos donde uno de ellos está siempre en la misma posición y los otros dos van cambiando como si una fuerza actuase sobre ellos.

Para lograr esto, en primer lugar definimos la ventana de la aplicación en una posición determinada y posteriormente definimos su altura y ancho. También indicamos que el display se hará en **RGBA** y contará con un **doble buffer**, para hacer que la imagen sea mucho más nítida.

El péndulo tiene las siguientes partes

1) primer brazo: guardaremos el angulo de inicio de este brazo
2) segundo brazo: también guardaremos su ángulo
3) circunferencias que unen los brazos

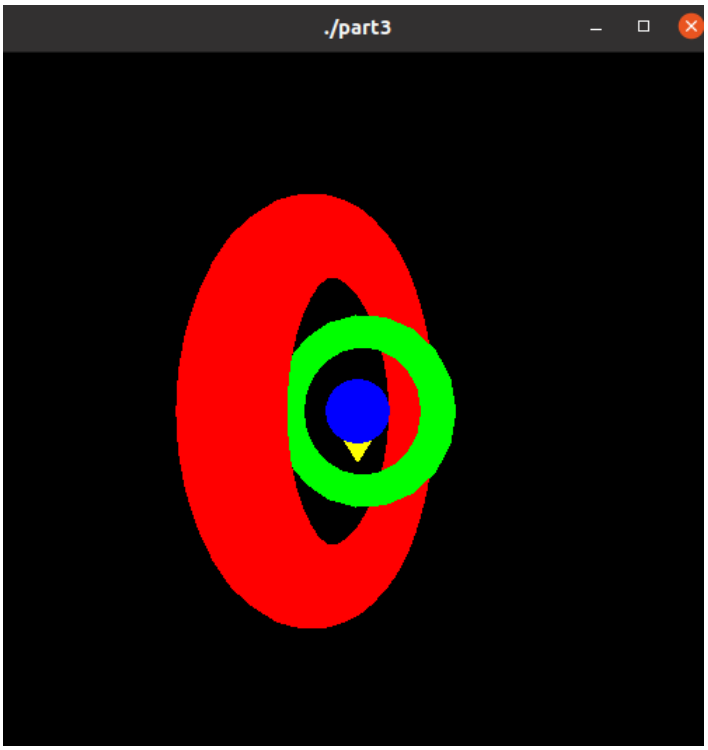
En un principio el péndulo tiene unos ángulos asignados, estos intentan ser grandes ya que buscamos que el péndulo gire violentamente por la gravedad.

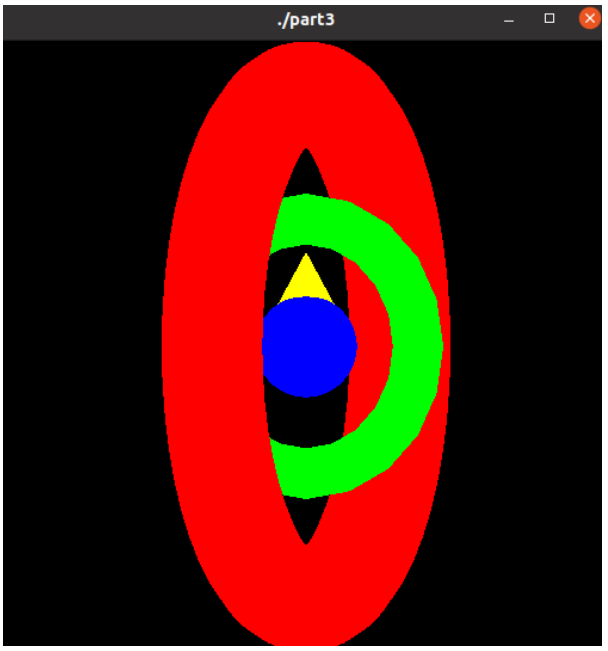
Para calcular el movimiento del péndulo dados sus ángulos, un peso para cada brazo y el efecto de la gravedad utilizamos las ecuaciones de lagrange, las cuales las encontramos en el método lagrange().

Pasando ya al propio dibujado la escena este se compone de dos bloques push/popmatrix. El primero de ellos contiene el dibujado de las dos primeras circunferencia y el brazo que las une, la posición de esta primera circunferencia es estática, **siempre es la misma**, mientras que la de la línea y el segundo punto va variando en función del angulo1 lel cual cambia en cada iteración del **idle.** Como idleFunc() tenemos asignado el método lagrange(). El segundo bloque de push/popMatrix dibuja la última circunferencia y el brazo que une al segundo con este, de nuevo, tanto esta línea como el punto ven su posición cambiada en función del angulo2 que es modificado con la función lagrange() **.** Al acabar este bloque de transformaciones aplicamos el **glutSwapBuffers()** para añadir esa fluidez a la imagen que nos permita ver el movimiento del péndulo en todo momento.

De la etapa 1 también conservamos el método Reproyectar() que cambia el tamaño del viewPort y modifica los parámetros del glOrtho() cada vez que se cambia de tamaño la ventana.

Parte 3





Esta es la primera etapa donde nuestro proyecto se sitúa en una escena 3D y por tanto todos nuestros objetos y polígonos pasarán a tener tres coordenadas, una para cada eje del espacio.

Se trata de una escena donde utilizamos primitivas incluidas en la librería openGL, estas van a ser la esfera, el cono y el torus. El usuario va a poder elegir cómo quiere estas primitivas, si las quiere en modo 'wire' donde únicamente se ven la unión de los vértices o 'solid' donde se ve como en el ejemplo de arriba.

La ventana de la aplicación tendrá por defecto un valor de 512x512, establecido por las variables *ancho/alto-ventana*, a la misma vez definimos donde se mostrará mediante las variables *posicionVentana-x/y*. Por último destacar que en la función **glutDisplayMode** le pasamos por parámetro los valores de **GLUT_DOUBLE y GLUT_DEPTH** para por un lado, poder usar el doble buffer y hacer que la imagen se vea mucho más nítida, y poder contar con el buffer de profundidad, ya que estamos en un espacio de 3D.

Una vez establecidas todas las condiciones que queremos para nuestra escena creamos ya la ventana con **glutCreateWindow,** a quien le pasamos un array de valores para que se cree de forma satisfactoria. Una vez establecida de forma definitiva la ventana creamos los eventos que generará el usuario que son 3

1. **Cambiar las proporciones de la ventana:** se usa la función *glutReshapeFunc,* donde se cambia el tamaño del ViewPort y luego se llama a la función proyección. Tenemos un atributo booleano global llamado ortho, el cual si se encuentra a true, la proyección va a ser ortogonal mediante glOrtho, de otro modo, vamos a utilizar la proyección glFrustrum.
2. **Cambiar el tipo de primitivas utilizadas:** Se ha creado una función para que si el usuario pulsa una tecla u otra se cambie las primitivas de solid a wire.
3. **Empezar / Finalizar el giro de las primitivas:** Si se pulsa click izquierdo se va llamar a un método que asignamos a glutIdlefunc() que va a hacer girar todas las primitivas sobre el eje y. Si el torus exterior gira en un sentido, el interior va a girar al contrario. El ángulo de giro de los torus se va a incrementar hasta que llegue a 360, que es cuando se va a reiniciar. Al pulsar click izquierdo también van a girar la esfera y el cono pero sobre el eje x. Si queremos parar las rotaciones, vamos a pulsar el click derecho.

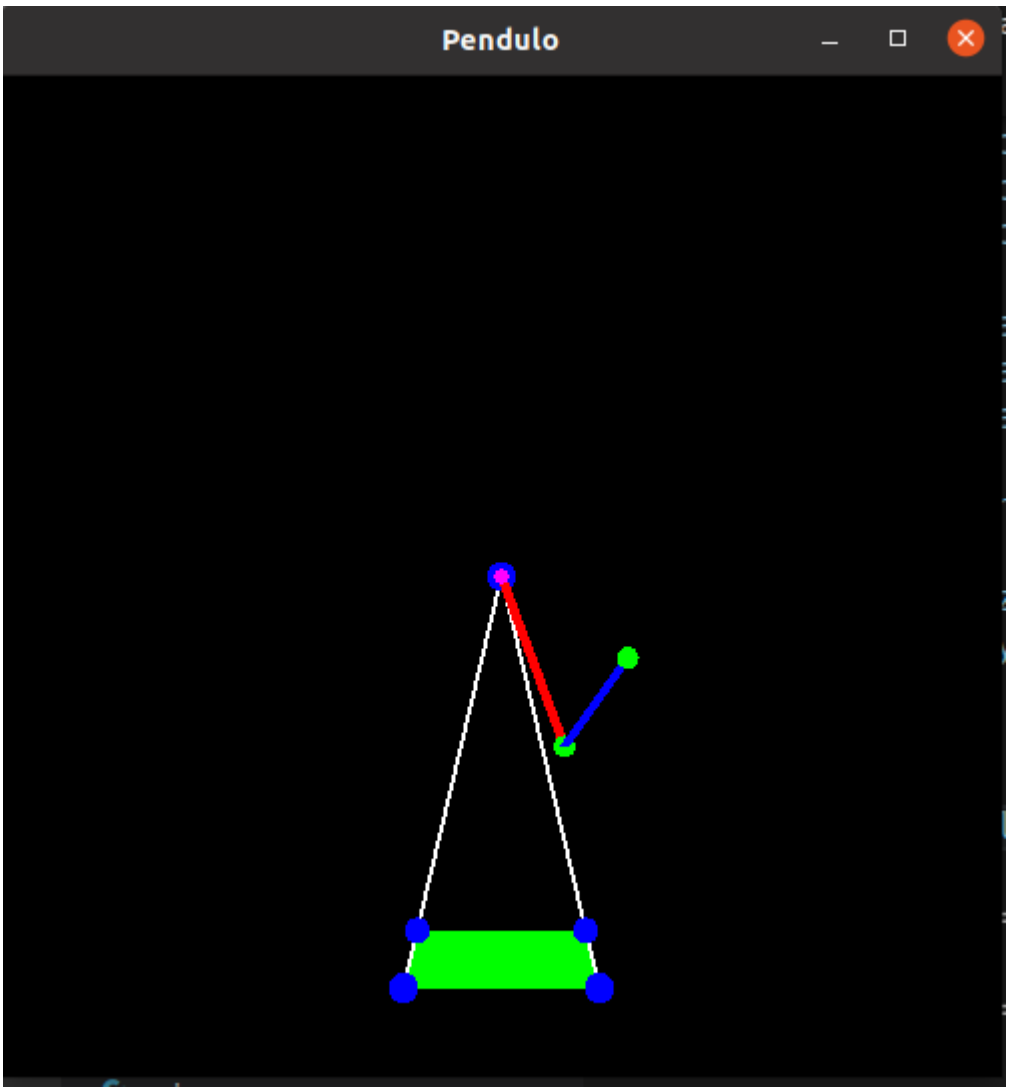
En el método Dibuja se dibujan todas las primitivas con los colores que vemos. También, en el caso de que la variable ortho esté a false, vamos a utilizar el gluLookAt(), para posicionar la cámara.

Manual de usuario

Cuando se ejecuta el programa por consola se muestran las opciones que se pueden realizar dentro de nuestra aplicación, aún así aquí están todas las opciones que puede hacer el usuario

- **Pulsar la tecla X → cambia a modo alambrico**
- **Pulsar la tecla Y → cambia a modo solido**
- **Pulsa Click Izquierdo → Arranca el movimiento**
- **Pulsa Click Derecho → Para el movimiento**

Parte 4



En esta práctica se ha realizado un péndulo, parecido al de la parte 2 pero adaptándolo al espacio 3D. El movimiento de este péndulo es el mismo que en el apartado 2, pero en este caso hemos introducido el movimiento de la cámara.

Al péndulo le hemos añadido unos soportes que nos van a servir para orientarnos en la escena cuando movamos la cámara.

Para realizar el movimiento de la cámara, hemos utilizado el método moure_camera(). Este método se ha decidido explicarlo con detalle en el apartado 6 (aunque se implementó en este apartado) , pero de manera resumida, nos permite mover la cámara por el plano x-z.

Este método lo llamamos si apretamos alguna de estas teclas : w,a,s,d.

El método canviar_posicio_camera que se llama al apretar la tecla q nos mueve la cámara a una de las 5 posiciones:
Central,Picado,Normal,Contrapicado,Nadir.

La dirección donde apunta la cámara se guarda en todo momento en las variables globales: posicio_apunta_x, posicio_apunta_y, posicio_apunta_z. I también su posición: posicio_camera_x, posicio_camera_y,posicio_camera_z.

Vemos que no guardamos el vector vertical de la cámara. Esto se debe a que como únicamente nos movemos en el plano x-z, entonces el vector vertical de la cámara va a ser siempre (0,1,0).

Para terminar con el movimiento de la cámara, tenemos el método voltaeixy, el cual se ejecuta al apretar la tecla 'e'. Este método va a rotar la cámara alrededor del eje y, y por tanto, alrededor del péndulo.

Para gestionar el movimiento de la vista de la cámara (dirección donde apunta la cámara) hemos utilizado glutSpecialFunc(ProcessSpecialKeys).

```
void ProcessSpecialKeys(int key, int x, int y)
{
    if (key == GLUT_KEY_LEFT)
    {
        direccio =-1;
        trasladar_inici();
        gira_vista_y(direccio*1*M_PI/180);
        tornar_lloc();
    }else if(key == GLUT_KEY_RIGHT){
        direccio = 1;
        trasladar_inici();
        gira_vista_y(direccio*1*M_PI/180);
        tornar_lloc();
    }else if (key == GLUT_KEY_UP){
        direccio = 1;
        trasladar_inici();
        float anglexz_copia = anglexz;
        girar_inici_xz();
        gira_vista_x(direccio*1*M_PI/180);
        gira_vista_y(anglexz_copia);
        tornar_lloc();
    }else if (key == GLUT_KEY_DOWN){
        direccio = -1;
        trasladar_inici();
        float anglexz_copia = anglexz;
        girar_inici_xz();
        gira_vista_x(direccio*1*M_PI/180);
        gira_vista_y(anglexz_copia);
        tornar_lloc();
    }
}
```

El método ProcessSpecialKeys se encarga de cambiar la dirección de la vista de la cámara dependiendo de la tecla apretada.

Hace uso de los siguientes métodos:

```
void trasladar_inici(){
    posicio_apunta_x -= posicio_camera_x;
    posicio_apunta_y -= posicio_camera_y;
    posicio_apunta_z -= posicio_camera_z;
}

void tornar_lloc(){
    posicio_apunta_x += posicio_camera_x;
    posicio_apunta_y += posicio_camera_y;
    posicio_apunta_z += posicio_camera_z;
}

void girar_inici_xz(){
        gira_vista_y(-anglexz);
}
```

traslada_inici(): mueve el vector de dirección al centro de coordenadas.

tornar_lloc(): devuelve el vector de dirección a su posición original

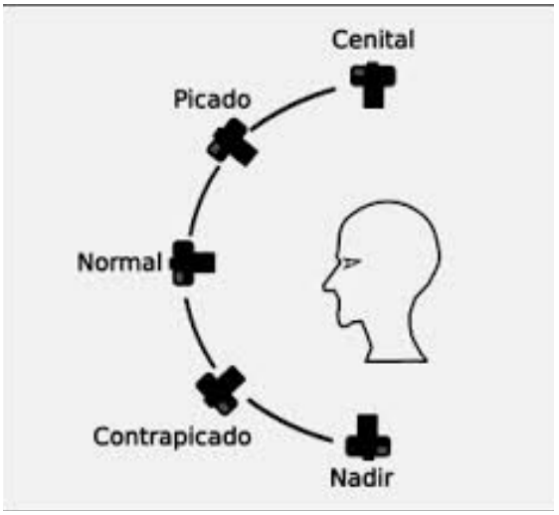
girar_inici_xz(): gira el vector al inicio del plano x-z, es decir, cuando anglexz =0.

En el caso de mover la cámara de izquierda a derecha o al revés, simplemente posicionamos el vector de dirección al inicio, hacemos un giro en el eje y, y lo volvemos a colocar en su lugar.

Pero en el caso de que queramos subir o bajar la vista, debemos posicionar el vector al inicio de coordenadas, girar el vector de dirección (en el eje y) al inicio del plano x-z, es decir, cuando el vector de dirección apunta a z y x=0, y luego hacer el giro en el eje x. Para terminar, volvemos a hacer el giro sobre el eje ‘y’ hasta su ángulo original y volvemos a posicionar el vector en su posición original. Este método es utilizado también el la parte 6.

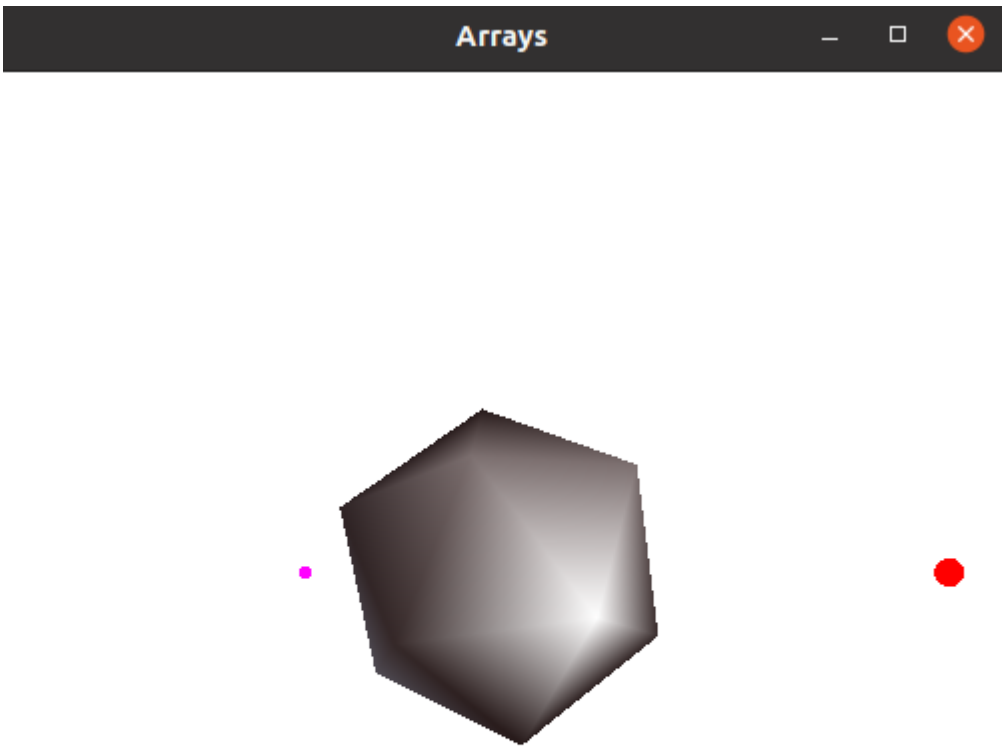
Manual de usuario

- Q → Cambiar a la proxima posición predeterminada de la cámara que son las siguientes
  Se ha omitido la normal



- R → Cambiar de sentido el movimiento del péndulo
- E → ???
- F → ???
- W A S D → Movimiento de la cámara básico (Adelante Izquierda Derecha Atras)
- Flechas → La dirección de cada una indica hacia donde queremos orientar la cámara, ejemplo, flecha arriba la cámara apuntará un poco más hacia arriba

Parte 5



Esta práctica pretende iniciarnos en el realismo dentro de la escena, para ello deberemos introducir nuevos elementos como la **luz** y las **sombras** que genera la misma. OpenGL nos permite introducir hasta un total de 8 fuentes de luz en la escena donde cada una de ellas produce a la vez unas sombras, estas sombras son generadas a través de **un modelo de iluminación**, que en nuestro caso usaremos el **GL_LIGHT_MODEL_AMBIENT.** Haremos uso de estas luces y de la introducción de un dodecaedro para dibujar una escena realista donde solo la parte a la que apunte la luz del dodecaedro estará iluminada.

Tendremos un total de 2 luces. Así como podemos ver en la imagen, la esfera roja y la esfera lila representan nuestras fuentes de luz. Para evitar que estas esferas tengan también el efecto de la luz, a la hora de crearlas en el display

hemos deshabilitado la iluminación. La luz lila se podrá apagar y encender de manera que podremos apreciar la diferencia.

Inicialmente en esta práctica indicamos la posición de la ventana de la aplicación y la grandaria de la misma en el método **main**. Posteriormente a la colocación de la ventana inicializamos el **glutInitDisplayMode** con el doble buffer, el buffer de profundidad y en RGB con canal alpha.

Habiendo ya iniciado la ventana donde se mostrará la escena en las condiciones que queremos nos disponemos a introducir las **luces,** elemento fundamental de esta escena Estas luz se situarán en unas coordenadas específicas que definimos nosotros. La luz roja será de tipo puntual, de manera que emite luz en un radio. La luz lila será de tipo spotlight, a esta luz le definiremos un ángulo de spotlight de 45 grados. Ambas luces apuntarán al objeto, que se encuentra en el centro de coordenadas.

Utilizaremos tanto el modelo **FLAT** como el **SMOOTH**, este podrá ser cambiado en todo momento por el usuario (se explicará en el manual de usuario). Ambas luces contarán con cada una de sus tres componentes: **ambiente, difusa y especular**, las que de nuevo, definimos con unos valores predeterminados. Iluminarán a nuestro dodecaedro, el cual es creado mediante un array de vértices y un array de índices que contiene, para cada triángulo del dodecaedro que vamos a dibujar, la posición de los 3 vértices dentro del array de vértices. Esta manera de pintar un objeto es mucho más eficiente que si tuviéramos que pintar los vértices individualmente.

Respecto al movimiento de la escena, hemos heredado de la práctica 4 únicamente el movimiento de la cámara en el eje y, de manera que vamos a poder rotar en el dodecaedro. Esto nos va a ser útil para poder apreciar todas las caras del objeto.

Las luces también se van a poder mover. El movimiento de una luz és el contrario de la otra, de manera que si una luz se acerca al objeto, la otra también se va a acercar. Si una luz sube en el eje y, la otra baja.

Comentar por último que todas las funciones excepto las del movimiento de cámara que puede realizar el usuario se “**bindean**” a sus respectivas teclas en la función **processNormalKeys**, estas últimas en **processSpecialKeys** ya que son un poco más tediosas.

Manual de usuario

- **Q → Activa/Desactiva la segunda luz de la escena**
- **A → Mueve un poco la luz hacia el eje X positivo**
- **Z → Mueve un poco la luz hacia el eje X negativo**
- **S → Mueve un poco la luz hacia el eje Y positivo**
- **X → Mueve un poco la luz hacia el eje Y negativo**
- **D → Mueve un poco la luz hacia el eje Z positivo**
- **C → Mueve un poco la luz hacia el eje Z negativo**
- **P → Cambia el modo de iluminación, flat → smooth o viceversa**
- **Flechas → Apuntan la cámara un poco más hacia el sentido que quieren: flecha arriba la cámara mira un poco más hacia arriba**

PARTE 6

Para la parte 6 hemos decidido escoger la opcionalidad:

-> Continuar con las funciones básicas de Opengl sin shaders. Como parte optativa hemos añadido, texturas, anti-aliasing y niebla.

Como personaje de la baraja de Química,Física i Matemàticas hemos decidido escoger a Marie Curie. La escena representa una sala de un museo compuesta por:

1) Una mesa donde vamos a poder ver un ejemplo de una reacción de fisión.
2) Dos bombillas articuladas, las cuales se pueden apagar, encender i mover.
3) Un retrato de Marie Curie, el cual se puede iluminar con un foco situado en el suelo.

En esta escena nuestro personaje va a poder hacer lo siguiente:

1) Moverse libremente por la escena. Este movimiento se limitará a las dimensiones de la habitación y a los elementos que hay en ella. Para poder realizar estos movimientos vamos a apretar una de estas teclas:

   w: mover el personaje hacia adelante
   a: mover el personaje hacia la izquierda
   s: mover el personaje hacia atrás
   d: mover el personaje hacia la derecha.

2) Girar la vista. El personaje podrá mover la vista en todas las direcciones. Para realizar estos movimientos vamos a apretar una de estas teclas:

   flecha hacia arriba : para subir la vista
   flecha hacia abajo: para bajar la vista
   flecha hacia la izquierda: para girar la vista a la izquierda
   flecha hacia la derecha: para girar la vista a la derecha.

3) Seleccionar el experimento, ejecutarlo, pararlo y salir del juego . Si apretamos el click derecho nos va a salir un menú en donde vamos a poder elegir entre:

   3.1) Seleccionar el experimento de fusión
   3.2) Salir de la escena

   Al seleccionar el experimento nos va a aparecer la escena de este sobre mesa del museo. Si queremos ejecutarlo vamos a apretar la tecla ‘m’.
   Si queremos parar el experimento en cualquier momento, vamos a apretar la tecla ‘n’.

   Si queremos volver al inicio del experimento vamos a tener que abrir el menú de nuevo y volver a seleccionar el experimento.

4) Encender, apagar y mover las luces. En el caso de las dos bombillas articuladas, vamos a tener las siguientes opciones:
    4.1) Subir las bombillas (botón p).
    4.2) Bajar las bombillas (botón o).
    4.3) Encender/Apagar la bombilla izquierda (botón k).
    4.4) Encender/Apagar la bombilla derecha (botón l).

    En el caso de la luz del cuadro, vamos a poder apagarla y encenderla con el botón 'j'.

Hay que tener en cuenta, que en cada caso, las teclas siempre tienen que estar en minúsculas, ya que de otro modo, ninguna de estas funcionalidades se podrían ejecutar.

A continuación vamos a ver cada parte del código y vamos a explicar su intención.

Empezaremos por el método main:

```
1344  int main(int argc, char **argv)
1345  {
1346      glutInit(&argc, argv);
1347
1348      // Indicamos como ha de ser la nueva ventana
1349      glutInitWindowPosition(100, 100);
1350      glutInitWindowSize(W_WIDTH, W_HEIGHT);
1351      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
1352
1353      glutCreateWindow("Part6");
1354
1355
1356      glutReshapeFunc(eventoVentana);
1357      glutDisplayFunc(Display);
1358
1359      glutSpecialFunc(ProcessSpecialKeys);
1360      glutKeyboardFunc(ProcessNormalKeys);
1361
1362
1363      opcionesVisualizacion();
1364      material();
1365      init();
1366
1367      // Comienza la ejecuci n del core de GLUT
1368      glutMainLoop();
1369      return 0;
1370  }
1371
1372
1373
```

En este método se crea una ventana llamada ‘Part6’ y se definen los métodos para el redimensionamiento y visualización de la ventana, y para las teclas (especiales y normales).

Además, llamamos a los métodos ‘opcionesVisualización’, material y init.

Vayamos a ver el método que asignamos a ‘glutReshapeFunc’:

```
883  void eventoVentana(GLsizei ancho, GLsizei alto)
884  {
885
886      glViewport(0, 0, ancho, alto);
887      glMatrixMode(GL_PROJECTION);
888      glLoadIdentity();
889      gluPerspective(45.0f,(GLdouble) ancho/alto , 0.0, 10.0);
890
891
892  }
```

Este método ha sido el mismo en anteriores niveles. En primer lugar se define el tamaño del ‘Viewport’ y luego se calcula el aspecto (aspecto = ancho/alto) para la visualización en perspectiva. Recordamos que esto se hace para evitar distorsiones en las proporciones de los objetos de la escena al cambiar de tamaño la ventana.

Antes de empezar a ver los demás métodos es necesario explicar qué variables globales hemos utilizado y cuál va a ser su utilidad.

1) MAX_TEXTURAS : total de texturas que hemos utilizado.
2) ALTO_TEXTURA_X/ANCHO_TEXTURA_X : guarda las dimensiones de una de las texturas.
3) W_WIDTH/HEIGHT : dimensiones de la ventana.
4) limit: nos va a ser útil para convertir los casos ‘-0.000’ que se pueden producir al utilizar ‘sin’ o ‘cos’ en ‘0.000’.
5) alto_textura/ancho_textura : va a ir cambiando dependiendo de la textura que queremos leer.

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <iostream>
#include <cmath>
#include <unistd.h>
#include <thread>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>

#define MAX_TEXTURAS       6 /* numero maximo de texturas */
#define ALTO_TEXTURA_PARED     1280   /* alto de la imagen a texturar */
#define ANCHO_TEXTURA_PARED    769  /* ancho de la imagen a texturar  */

#define ALTO_TEXTURA_SUELO     540  /* alto de la imagen a texturar */
#define ANCHO_TEXTURA_SUELO    360  /* ancho de la imagen a texturar  */

#define ALTO_TEXTURA_SOSTRE     600  /* alto de la imagen a texturar */
#define ANCHO_TEXTURA_SOSTRE   600  /* ancho de la imagen a texturar  */

#define ALTO_TEXTURA_MARIE     539  /* alto de la imagen a texturar */
#define ANCHO_TEXTURA_MARIE  902  /* ancho de la imagen a texturar  */

#define ALTO_TEXTURA_METAL     178  /* alto de la imagen a texturar */
#define ANCHO_TEXTURA_METAL  283  /* ancho de la imagen a texturar  */


const int W_WIDTH = 500; // Tama□o incial de la ventana
const int W_HEIGHT = 500;

const float limit = 0.000001;

int alto_textura = ALTO_TEXTURA_PARED;
int ancho_textura = ANCHO_TEXTURA_PARED;

```

```
float posicio_apunta_x = 0.0f;
float posicio_apunta_y =0.5f;
float posicio_apunta_z =-1.0f;

float posicio_camera_x = 0.0f;
float posicio_camera_y =0.5f;
float posicio_camera_z =-0.5f;

float anglexz = 0.0;
float anglexy = 0.0;
int direccio =1;
int sentit = 1;

//Per els llums

float angle1_llum = 0.0f;
float angle2_llum = 0.0f;

float altura_llums= 0.5f;

bool ences_esquerre = true;
bool ences_dreta = true;
bool ences_quadre = false;
bool fusio = true;
bool acaba = false;
bool dedins = true;


GLubyte textura_pared[ALTO_TEXTURA_PARED][ANCHO_TEXTURA_PARED][3]; /* vector de texturas */
GLubyte textura_enterra[ALTO_TEXTURA_SUELO][ANCHO_TEXTURA_SUELO][3]; /* vector de texturas */
GLubyte textura_sostre[ALTO_TEXTURA_SOSTRE][ANCHO_TEXTURA_SOSTRE][3]; /* vector de texturas */
GLubyte textura_marie[ALTO_TEXTURA_MARIE][ANCHO_TEXTURA_MARIE][3]; /* vector de texturas */
GLubyte textura_metal[ALTO_TEXTURA_METAL][ANCHO_TEXTURA_METAL][3]; /* vector de texturas */
GLuint nombreTexturas[MAX_TEXTURAS];

```

6) posicio_apunta_x/posicio_apunta_y/posicio_apunta_z : en conjunto guarda el vector de dirección de la cámara.
7) posicio_camera_x/posicio_camera_y/posicio_camera_z: en conjunto guarda el vector de posición de la cámara.
8) anglexz y angleyz (en la práctica anglexy -> angleyz) : guarda el angulo al que está el vector de dirección de la cámara con respecto los planos x-z y y-z. Esto nos va a ser útil para mover la vista de la cámara.
9) direcció: Si queremos mover la vista de la cámara en el eje x en un angulo 'a', esta variable nos da la dirección del giro.
10) angle1_llum y angle2_llum: guardan respectivamente los ángulos de los dos brazos de las bombillas articuladas. Angle1_llum será el mismo para el primer brazo de la bombilla izquierda i para la de la derecha, lo mismo con angle2_llum. Como se puede suponer, esto se hace así, porque las bombillas se van a mover a la vez y, por tanto, los ángulos seran los mismos en las dos bombillas.
11) altura_llums: guarda la altura (y) a la que se encuentran las bombillas. Va a ser útil a la hora de colocar las luces. El valor de esta variable va a cambiar con el movimiento de las bombillas.
12) ences_esquerre,ences_dreta,ences_quadre : estarán a true si la bombilla izquierda, derecha y la luz del cuadro están encendidas, respectivamente.
13) fisio(corrección) : estará a true si el usuario ha seleccionado la opción del menú del experimento de fisión.
14) dedins: Va a cambiar dependiendo de la posición de la cámara. Si la cámara se encuentra dentro de los límites definidos por la dimensión de la escena y la posición de los objetos, entonces dedins estará a true. Si la cámara llega a un límite, dedins estará a false. Como se puede suponer, esta variable nos va a ser útil a la hora de controlar el movimiento de la cámara.
15) sentit y acaba: en desuso.
16) textura_x : vectores para guardar el contenido de las texturas que se van a leer
17) nombreTexturas: va a guardar el índice de cada textura. Va a ser útil a la hora de hacer un BindTexture().

```
static GLfloat posicionsbolles [7][3] = {{-0.18,0.33,-2.5},{0.0,0.33,-2.51},{0,0.33,-2.49},
    {0.01,0.33,-2.51},{0.01,0.33,-2.49},{0.005,0.33,-2.53},{0.005,0.33,-2.47},};
static GLfloat paretsdata [24][3] = {{-1,0,0},{-1,1,0},{1,1,0},{1,0,0},
    {1,0,-1},{1,1,-1},{1,1,-2},{1,0,-2},{1,0,-3},{1,1,-3},{1,1,-4},{1,0,-4},
    {1,0,-5},{1,1,-5},{-1,1,-5},{-1,0,-5},{-1,0,-4},{-1,1,-4},{-1,1,-3},{-1,0,-3},
    {-1,0,-2},{-1,1,-2},{-1,1,-1},{-1,0,-1}
    };

static GLuint paretsindexs[12][4] = {{0,1,2,3},{2,3,4,5},{4,5,6,7},{6,7,8,9},
    {8,9,10,11},{10,11,12,13},{12,13,14,15},{14,15,16,17},{16,17,18,19},
    {18,19,20,21},{20,21,22,23},{22,23,0,1}};

static GLfloat normaleParets[24][3] = {{1,0,-1},{1,0,-1},{-1,0,-1},{-1,0,-1},
    {-1,0,0},{-1,0,0},{-1,0,0},{-1,0,0},{-1,0,0},{-1,0,0},{-1,0,0},{-1,0,0},
    {-1,0,1},{-1,0,1},{1,0,1},{1,0,1},{1,0,0},{1,0,0},{1,0,0},{1,0,0},
    {1,0,0},{1,0,0},{1,0,0},{1,0,0}
    };

static GLfloat sostredata[33][3] = {{-1,1,0},{0,1,0},{0,1,-0.5},{-1,1,-0.5},
    {1,1,0},{1,1,-0.5},{-1,1,-1},{0,1,-1},{1,1,-1},{-1,1,-1.5},{0,1,-1.5},{1,1,-1.5},
    {-1,1,-2},{0,1,-2},{1,1,-2},{-1,1,-2.5},{0,1,-2.5},{1,1,-2.5},{-1,1,-3},{0,1,-3},
    {1,1,-3},{-1,1,-3.5},{0,1,-3.5},{1,1,-3.5},{-1,1,-4},{0,1,-4},{1,1,-4},{-1,1,-4.5},
    {0,1,-4.5},{1,1,-4.5},{-1,1,-5},{0,1,-5},{1,1,-5}

};

static GLuint sostreindexs[20][4] = {{0,1,2,3},{3,2,7,6},{6,7,10,9},{9,10,13,12},{12,13,16,15},{15,16,19,18},
    {18,19,22,21},{21,22,25,24},{24,25,28,27},{27,28,31,30},{1,4,5,2},{2,5,8,7},{7,8,11,10},{10,11,14,13},{13,14,17,16},
    {16,17,20,19},{19,20,23,22},{22,23,26,25},{25,26,29,28},{28,29,32,31}
};

static GLfloat enterredata[33][3] = {{-1,0,0},{0,0,0},{0,0,-0.5},{-1,0,-0.5},
    {1,0,0},{1,0,-0.5},{-1,0,-1},{0,0,-1},{1,0,-1},{-1,0,-1.5},{0,0,-1.5},{1,0,-1.5},
    {-1,0,-2},{0,0,-2},{1,0,-2},{-1,0,-2.5},{0,0,-2.5},{1,0,-2.5},{-1,0,-3},{0,0,-3},
    {1,0,-3},{-1,0,-3.5},{0,0,-3.5},{1,0,-3.5},{-1,0,-4},{0,0,-4},{1,0,-4},{-1,0,-4.5},
    {0,0,-4.5},{1,0,-4.5},{-1,0,-5},{0,0,-5},{1,0,-5}
```

8) Igual que en la parte5, para las paredes, el suelo y el techo hemos guardado la posición de sus vértices en arrays. Además, para cada parte, tenemos un array de índices que guarda los vértices que queremos utilizar del array de vértices para cada polígono. Tanto el suelo como la pared y el techo se van a construir a partir de cuadrados.

9) Para las esferas que vamos a utilizar para el experimento de fisión, hemos guardado su posición en al array ‘posicionsbolles’.

```
static GLuint enterreindexs[20][4] = {{0,1,2,3},{3,2,7,6},{6,7,10,9},{9,10,13,12},{12,13,16,15},{15,16,19,18},
    {18,19,22,21},{21,22,25,24},{24,25,28,27},{27,28,31,30},{1,4,5,2},{2,5,8,7},{7,8,11,10},{10,11,14,13},{13,14,17,16},
    {16,17,20,19},{19,20,23,22},{22,23,26,25},{25,26,29,28},{28,29,32,31}
};
```

El método que asignamos a ‘glutDisplayFunc’:

```
845  void Display(void){
846
847      glClearColor(0.0,0.0,0.0,0.0);
848      init();
849      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
850
851      glMatrixMode(GL_MODELVIEW);
852      glLoadIdentity();
853      gluLookAt(posicio_camera_x, posicio_camera_y, posicio_camera_z, posicio_apunta_x, posicio_apunta_y, posicio_apunta_z, 0.0, 1
854
855      glEnable(GL_TEXTURE_2D);
856
857      crearParedes();
858      pintarSostre();
859      pintarQuadre();
860      glDisable(GL_LIGHTING);
861      pintarLlum();
862      pintarLLumQuadre();
863      glEnable(GL_LIGHTING);
864
865      pintarColumna();
866
867      if(!fusio){
868          fisio();
869      }
870
871      glDisable(GL_LIGHTING);
872      glPushMatrix();
873      glTranslatef(posicio_apunta_x,posicio_apunta_y,posicio_apunta_z);
874      glColor3f(0.0,0.0,0.0);
875      glutSolidSphere(0.001,30.0,30.0);
876      glPopMatrix();
877      glEnable(GL_LIGHTING);
878
879      glutSwapBuffers();
880      glFlush();
881  }
```

Este es el método Display, común en todos los niveles. En este caso va a hacer lo siguiente:

1) definir el color del fondo
2) init(): vuelve a posicionar las luces y definir los materiales de los objetos de la escena.
3) Posicionamos la cámara con el gluLookAt(). La posición y orientación de esta, como ya sabemos, está guardado en variables globales. El vector vertical de la cámara no lo guardamos, esto se debe a que la cámara solo se va a mover en el eje x-z y, por tanto, el vector vertical siempre va a ser (0,1,0).
4) Habilitamos las texturas
5) Llamamos a los métodos:

    5.1) crearParedes() : se va a encargar de pintar las paredes y de añadirles una textura que tendremos guardada. En este caso, utilizamos la textura nombreTexturas[0] la cual se ha definido en el método **material** que veremos posterior -mente. Como modo de mapeo hemos establecido el GL_MODULATE, el cual va hacer que la textura se mezcle con el color del polígono. Esto lo hemos hecho porque nos parecía que la textura tenia un color muy claro y, de este modo, la textura queda oscurecida.

En este método también nos vamos a encargar de pintar el suelo, que se hace de manera similar a las paredes.
5.2) pintarSostre(): se va a encargar de pintar el techo y de añadirle una textura guardada, de manera similar al método crearParedes()
5.3) pintarQuadre(): va a pintar el marco del cuadro y va a añadir la textura de la imagen de Marie Curie.

6) Deshabilitamos las luces a la hora de llamar los métodos 'pintarLLum()' i 'pintarLlumQuadre'. Esto lo hacemos para que los objetos que se supone que emiten la luz, no reciban sus efectos como si fuesen receptores.
7) Pintamos el pilar y la mesa que se encuentran en el centro de la escena con el método 'pintarColumna' además de añadir sus texturas.
8) En el caso de que el usuario haya seleccionado la opción de visualizar el experimento de fisión, como sabemos, la variable booleana 'fision' estará a 'true' y, por tanto, vamos a llamar al método 'fisio()'. Este método va a pintar la escena del experimento de la reacción de fisión encima de la mesa.
9) Para terminar, hemos decidido añadir una esfera pequeña de color negro que se va a posicionar donde apunta la cámara. Esto nos va a servir para visualizar hacia dónde se mueve la cámara. Antes de pintar esta esfera hemos deshabilitado la iluminación para que no afecte a esta, ya que este no es el objetivo de la esfera.

Vayamos a ver el método asignado a 'glutSpecialFunc':

El funcionamiento de este método es el mismo que en la parte4, que es donde queda explicado. Para cada tecla de dirección realizamos unas transformaciones al vector de dirección de la cámara, de esta manera, la cámara va a poder mover su vista.

Vayamos a ver el método asignado a 'glutKeyboardFunc':

```
void ProcessNormalKeys(unsigned char tecla, int x, int y){

    switch(tecla){
        case 'w':
                moure_camera(1);
            break;
        case 's':
              moure_camera(2);
                break;
        case 'd':
                moure_camera(3);
                break;
        case 'a':
                moure_camera(4);
                break;
        case 'p':
                if(angle1_llum < 90){
                    angle1_llum += 0.5;
                    angle2_llum += 0.5;
                    altura_llums += 0.5/180;
                }
                break;
        case 'o':
                if(angle1_llum > 0){
                    angle1_llum -= 0.5;
                    angle2_llum -= 0.5;
                    altura_llums -= 0.5/180;
                }
                break;
        case 'k':
                ences_esquerre = !ences_esquerre;
                break;
        case 'l':
                ences_dreta = !ences_dreta;
                break;
```

```
1303                    break;
1304            case 'j':
1305                    ences_quadre = !ences_quadre;
1306                    break;
1307            case 'm' :
1308                    if(!fusio){
1309                     glutIdleFunc(experiment);
1310                    }
1311                    break;
1312            case 'n' :
1313                    if(!fusio){
1314                     glutIdleFunc(NULL);
1315                    }
1316                    break;
1317        }
1318        glutPostRedisplay();
1319    }
1320
```

En este método tratamos los casos de cada tecla. Para las teclas de movimiento de la cámara llamamos al método ‘moure_camera()’. Para subir y bajar las bombillas, ajustamos el ángulo de cada brazo articulado (caso teclas ‘p’ y ‘o’). Además, actualizamos la altura a la que se debe encontrar la luz que se va a emitir (guardada en altura_llums).

En los casos de las teclas ‘k’,’l’,’j’ encendemos y apagamos las luces cambiando el estado de sus variables booleanas.

En los casos de las teclas ‘m’ y ‘n’, arrancamos y paramos el experimento de fusión.

El método moure_camera() hará lo siguiente:

```
1136  void moure_camera(int cas){
1137      float angle;
1138      trasladar_inici();
1139
1140      if(posicio_apunta_z<0){
1141          if(posicio_apunta_x>0){
1142              angle = atan(posicio_apunta_z/posicio_apunta_x) +360.0*M_PI/180;
1143          }else if(posicio_apunta_x <0){
1144              angle = atan(posicio_apunta_z/posicio_apunta_x) +180.0*M_PI/180;
1145          }else{
1146              angle = 270.0f*M_PI/180;
1147          }
1148      }else if(posicio_apunta_z>0){
1149          if(posicio_apunta_x>0){
1150              angle = atan(posicio_apunta_z/posicio_apunta_x);
1151          }else if (posicio_apunta_x<0){
1152              angle = atan(posicio_apunta_z/posicio_apunta_x) +180*M_PI/180;
1153          }else{
1154              angle = 90.0f*M_PI/180;
1155          }
1156      }else{
1157          if(posicio_apunta_x <0){
1158              angle = 180.0f*M_PI/180;
1159          }else{
1160              angle = 0.0f*M_PI/180;
1161          }
1162      }
1163
1164
1165      switch(cas){
1166          case 3:
1167          angle -= 90.0f*M_PI/180;
1168          break;
1169          case 4:
1170          angle -= 90.0f*M_PI/180;
1171          break;
1172      }
```

1) Trasladamos al inicio de coordenadas con ‘trasladar_inici()’ el vector de dirección de la cámara.
2) Calculamos el ángulo respecto el inicio del plano x-z. Lo primero, es determinar en que cuadrante estamos y luego, utilizando la cotangente de la división entre posicio_apunta_z y posicio_apunta_x sacamos el ángulo.
3) Los casos que pasamos por parámetro son los siguientes:
   1 y 2: movimiento adelante/atrás
   3 y 4: movimiento izquierda/derecha

Una vez calculado el ángulo, si estamos en el caso de que queremos movernos a la derecha o a la izquierda, vamos a restar 90 grados al ángulo.

```
1174    float posicio_x = cos(angle);
1175    float posicio_z = sin(angle);
1176
1177    if(fabs(posicio_x)< limit){
1178        posicio_x = 0.0f;
1179    }
1180
1181    if(fabs(posicio_z)< limit){
1182        posicio_z = 0.0f;
1183    }
1184
1185    float provisional_x = 0.0;
1186    float provisional_z = 0.0;
1187
1188    switch(cas){
1189        case 1:
1190        provisional_x = posicio_camera_x+0.05*posicio_x;
1191        provisional_z = posicio_camera_z+0.05*posicio_z;
1192
1193        estic_dedins(provisional_x,provisional_z);
1194
1195        if(dedins){
1196            posicio_camera_x += 0.05*posicio_x;
1197            posicio_camera_z += 0.05*posicio_z;
1198        }
1199        break;
1200        case 2:
1201        provisional_x = posicio_camera_x-0.05*posicio_x;
1202        provisional_z = posicio_camera_z-0.05*posicio_z;
1203
1204
1205        estic_dedins(provisional_x,provisional_z);
1206
1207        if(dedins){
1208            posicio_camera_x -= 0.05*posicio_x;
1209            posicio_camera_z -= 0.05*posicio_z;
1210        }
```

A continuación calculamos la posición x y z con este nuevo ángulo, teniendo en cuenta que posicio_x y posicio_z se inicializan a la posición 0, es decir, al inicio de coordenadas, esta posición aún no es definitiva respecto a nuestra escena.

Seguidamente, vamos a inicializar dos variables provisionales x y z en las cuales vamos a guardar la que sería la nueva posición de la cámara en la escena. Una vez hecho esto vamos a llamar al método estic_cedins al que le vamos a pasar estas variables.
estic_dedins:

```
121  void estic_dedins(float provisional_x,float provisional_z){
122      //Parets
123      dedins = true;
124
125      if((provisional_x<-0.8)||(provisional_x>0.8)||(provisional_z > -0.20)||(provisional_z < -4.8)){
126          dedins = false;
127      }else{
128          if((provisional_x>-0.23)&&(posicio_camera_x<0.23)){
129              if((provisional_z < -2.0)&&(provisional_z>-3)){
130                  dedins = false;
131              }
132          }
133      }
134  }
```

Este método va a poner la variable dedins a false si nos hemos pasado de los límites de la escena, estos límites se dan por los objetos y las paredes de la escena. En nuestro caso tenemos en cuenta las cuatro paredes y la mesa que se sitúa al centro de la escena.

Si después de llamar a estic_dedins, la variable dedins sigue a true, entonces ya vamos a actualizar la posición de la cámara. Si dedins se encuentra a false después de ejecutar este método, entonces las variables globales posicio_camera_x y posicio_camera_z no se van a actualizar, es decir, la cámara va a seguir al mismo sitio.

```
1211
1212            break;
1213            case 3:
1214            provisional_x = posicio_camera_x-0.05*posicio_x;
1215            provisional_z = posicio_camera_z-0.05*posicio_z;
1216
1217            estic_dedins(provisional_x,provisional_z);
1218
1219            if(dedins){
1220                posicio_camera_x -= 0.05*posicio_x;
1221                posicio_camera_z -= 0.05*posicio_z;
1222            }
1223            break;
1224            case 4:
1225            provisional_x = posicio_camera_x+0.05*posicio_x;
1226            provisional_z = posicio_camera_z+0.05*posicio_z;
1227
1228            estic_dedins(provisional_x,provisional_z);
1229
1230            if(dedins){
1231                posicio_camera_x += 0.05*posicio_x;
1232                posicio_camera_z += 0.05*posicio_z;
1233            }
1234            break;
1235        }
1236            tornar_lloc();
```

Una vez actualizado la posición de la cámara ya sólo queda devolver el vector de dirección de la cámara a su posición en la escena.

Pasemos a ver el método opcionesVisualizacion():

```
1316
1317  void opcionesVisualizacion(void)
1318  {
1319      glutCreateMenu(menuapp);
1320          glutAddMenuEntry("Fissio", 1);
1321          glutAddMenuEntry("Salir", 2);
1322          glutAttachMenu(GLUT_RIGHT_BUTTON);
1323
1324      printf(" flecha superior - enfocar la càmera cap a dalt\n");
1325      printf(" flecha inferior - enfocar la càmera cap a baix\n");
1326      printf("flecha esquerre - enfocar la càmera cap a l'esquerre\n");
1327      printf("flecha dreta - enfocar la càmera cap a la dreta\n");
1328      printf("p - puja bombilles\n");
1329      printf("o - baixa bombilles\n");
1330      printf("k - apaga/encen llum esquerre\n");
1331      printf("l - apaga/encen llum dreta\n");
1332      printf("j- apaga/encen llum quadre\n");
1333      printf("m - comença experiment\n");
1334      printf("n - acaba experiment\n");
1335
1336  }
1337
```

```
639  void menuapp(int value) {
640
641    switch(value) {
642    case 1:  fusio = false;
643    static GLfloat posicionsbolles1 [7][3] = {{-0.18,0.33,-2.5},{0.0,0.33,-2.51},{0,0.33,-2.49},
644      {0.01,0.33,-2.51},{0.01,0.33,-2.49},{0.005,0.33,-2.53},{0.005,0.33,-2.47},};
645      memcpy(posicionsbolles, posicionsbolles1, sizeof(posicionsbolles));
646    break;
647    case 2:  exit(2); break;
648    default:                          break;
649    }
650    glutPostRedisplay();
651
652  }
```

En primer lugar se crea un menú en el cual habrá dos casos, Fissio y Salir. En el caso de Fissio se reinicia la posición original de las esferas del experimento y en el caso de Salir, se termina la ejecución.

A continuación vamos a ver el método material:

```
954  void material (void) {
955    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
956    glGenTextures(MAX_TEXTURAS, nombreTexturas);
957
958    /* Asignaci?n de la imagen "pixels" como textura 2D */
959    char nom [] = "pared.tga";
960    leeTextura(nom,0);
961    char nom1 [] = "enterre.tga";
962    leeTextura(nom1,1);
963    char nom2 [] = "sostre.tga";
964    leeTextura(nom2,2);
965    char nom3 [] = "marie.tga";
966    leeTextura(nom3,3);
967    char nom4 [] = "metal.tga";
968    leeTextura(nom4,4);
969    /* Definici?n de los par?metros iniciales de texturacion */
970    glBindTexture(GL_TEXTURE_2D, nombreTexturas[0]);
971
972    GLfloat tparams[]={0,1.4,0,0.5};
973    GLfloat sparams[]={1,0.0,0.0,0.5};
974
975
976    glTexImage2D(GL_TEXTURE_2D, 0, 3, 1280, 769,
977                 0, GL_RGB, GL_UNSIGNED_BYTE, textura_pared);
978    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
979    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
980    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
981    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
982
983    glBindTexture(GL_TEXTURE_2D, nombreTexturas[1]);
984
985    glTexImage2D(GL_TEXTURE_2D, 0, 3, 540, 360,
986                 0, GL_RGB, GL_UNSIGNED_BYTE, textura_enterra);
987    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
988    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
989    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
990    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
991
992    glBindTexture(GL_TEXTURE_2D, nombreTexturas[2]);
```

Este método se va a encargar de leer las texturas y guardarlas en un array. A cada array vamos a declarar unos parámetros y se va a hacer un bind entre este y el array nombreTexturas.

Para leer las texturas hemos utilizado el método leeTextura:

```
894  void leeTextura (char *fichero, int torn) {
895      int   i, j;
896      char  r, g, b, c;
897      FILE  *tga;
898
899      /* Apertura del fichero TGA */
900      if ((tga = fopen(fichero, "rb")) == NULL)
901          printf ("Error abriendo el fichero: %s\n", fichero);
902      else {
903          /* Lee los 18 primeros caracteres de la cabecera */
904          for (j=1; j<=18; j++)
905              fscanf (tga, "%c", &c);
906
907          if(torn == 0){ //pared
908              alto_textura = ALTO_TEXTURA_PARED;
909              ancho_textura = ANCHO_TEXTURA_PARED;
910          }else if(torn == 1){
911              alto_textura = ALTO_TEXTURA_SUELO;
912              ancho_textura = ANCHO_TEXTURA_SUELO;
913          }else if(torn ==2){
914              alto_textura = ALTO_TEXTURA_SOSTRE;
915              ancho_textura = ANCHO_TEXTURA_SOSTRE;
916          }else if(torn ==3){
917              alto_textura = ALTO_TEXTURA_MARIE;
918              ancho_textura = ANCHO_TEXTURA_MARIE;
919          }else if(torn ==4){
920              alto_textura = ALTO_TEXTURA_METAL;
921              ancho_textura = ANCHO_TEXTURA_METAL;
922          }
923          /* Lee la imagen */
924          for (j=alto_textura-1; j>=0; j--) {
925              for (i=ancho_textura-1; i>=0; i--) {
926                  fscanf(tga, "%c%c%c", &b, &g, &r);
927                  if(torn ==0){
928                   textura_pared[j][i][0] = (GLubyte)r;
929                   textura_pared[j][i][1] = (GLubyte)g;
930                   textura_pared[j][i][2] = (GLubyte)b;
931                  }else if(torn == 1){
```

el cual abre el fichero de cada textura y guarda la información en los arrays declarados como variables globales para cada textura.

Vayamos a ver el método init() el cual se encarga de declarar las luces, materiales y fog de la escena:

En primer lugar, habilitamos el buffer de profundidad para darnos cuenta de la proximidad de los objetos, y también habilitamos el efecto anti-aliasing. El glEnable(GL_STENCIL_TEST) está en desuso, no se va a encontrar en el código.

A continuación, vamos a declarar las luces. Las luces 0 y 1 se van a corresponder a las bombillas izquierda y derecha respectivamente. La luz 2 representa la luz del cuadro. Como podemos ver las 3 luces son del tipo spotlight, que són comunes en una sala de museo. El que sean spotlight va a

significar que estas fuentes de luz van a emitir en un ángulo máximo hacia una dirección que definimos. También definimos los parámetros de atenuación de la luz para las 3, esto va a hacer que cuanto más lejos estemos de la fuente, menor luz va a recibir un objeto.

```
659  void init(void){
660      glEnable(GL_DEPTH_TEST);
661      glShadeModel(GL_SMOOTH);
662      glDepthFunc   (GL_LEQUAL);
663      glEnable(GL_NORMALIZE);
664
665      glEnable(GL_BLEND);
666      glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
667      glEnable(GL_STENCIL_TEST);
668
669      glEnable (GL_LINE_SMOOTH);
670      glHint ( GL_LINE_SMOOTH_HINT, GL_NICEST);
671      //fins aquí va bé
672
673      glEnable(GL_LIGHTING);{
674          GLfloat light_ambient[]   = { 0.05f, 0.0f, 0.0f, 1.0f };
675          GLfloat light_diffuse[]   = { 0.5f, 0.6f, 0.7f, 1.0f };
676          GLfloat light_specular[]  = { 1.0f, 1.0f, 1.0f, 1.0f };
677          GLfloat light_position_esquerre[]  = { -0.1f, altura_llums, -2.5f, 1.0f };
678          GLfloat light_position_dreta[]  = { 0.1f, altura_llums, -2.5f, 1.0f };
679          GLfloat light_position_quadre[]  = { 0.0f, 0.01, -4.75f, 1.0f };
680          GLfloat light_direction[] = { 0.0f,-1.0f,0};
681          GLfloat light_direction_quadre[] = { 0.0f,1.0f,-0.5};
682          GLfloat light_cutoff = 60.0f;
683          GLfloat light_cutoff_quadre =90.0f;
684
685          glLightfv(GL_LIGHT0, GL_AMBIENT,  light_ambient);
686          glLightfv(GL_LIGHT0, GL_DIFFUSE,  light_diffuse);
687          glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
688          glLightfv(GL_LIGHT0, GL_POSITION, light_position_esquerre);
689          glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, light_direction);
690          glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, light_cutoff);
691          glLightf(GL_LIGHT0,GL_CONSTANT_ATTENUATION,1.0);
692          glLightf(GL_LIGHT0,GL_LINEAR_ATTENUATION,0.5);
693          glLightf(GL_LIGHT0,GL_QUADRATIC_ATTENUATION,0.2);
694
695
```

```
696
697          glLightfv(GL_LIGHT1, GL_AMBIENT,  light_ambient);
698          glLightfv(GL_LIGHT1, GL_DIFFUSE,  light_diffuse);
699          glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular);
700          glLightfv(GL_LIGHT1, GL_POSITION, light_position_dreta);
701          glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, light_direction);
702          glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, light_cutoff);
703          glLightf(GL_LIGHT1,GL_CONSTANT_ATTENUATION,1.0);
704          glLightf(GL_LIGHT1,GL_LINEAR_ATTENUATION,0.5);
705          glLightf(GL_LIGHT1,GL_QUADRATIC_ATTENUATION,0.2);
706
707          glLightfv(GL_LIGHT2, GL_AMBIENT,  light_ambient);
708          glLightfv(GL_LIGHT2, GL_DIFFUSE,  light_diffuse);
709          glLightfv(GL_LIGHT2, GL_SPECULAR, light_specular);
710          glLightfv(GL_LIGHT2, GL_POSITION, light_position_quadre);
711          glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, light_direction_quadre);
712          glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, light_cutoff_quadre);
713          glLightf(GL_LIGHT2,GL_CONSTANT_ATTENUATION,1.0);
714          glLightf(GL_LIGHT2,GL_LINEAR_ATTENUATION,0.5);
715          glLightf(GL_LIGHT2,GL_QUADRATIC_ATTENUATION,0.2);
716
717          //GLfloat mat_ambient[]   = { 1.0f, 1.0f, 1.0f, 1.0f };
718          //GLfloat mat_diffuse[]   = { 1.0f, 1.0f, 1.0f, 1.0f };
719          //GLfloat mat_specular[]  = { 1.0f, 1.0f, 1.0f, 1.0f };
720          GLfloat mat_shininess[] = { 50.0 };
721
722          GLfloat mat_specular[] = {1.0, 0.901, 0.792, 1.0};
723          GLfloat mat_diffuse[] = {1.0, 0.901, 0.792, 1.0};
724          GLfloat mat_ambient[] = {1.0, 0.901, 0.792, 1.0};
725          GLfloat mat_emission[] = {0.3,0.3,0.2,0.0};
726
727          glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
728          glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
729          glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
730          glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
731          glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
732
```

Definimos los materiales

```
733
734         if(ences_esquerre){
735             glEnable(GL_LIGHT0);
736         }else{
737             glDisable(GL_LIGHT0);
738         }
739
740         if(ences_dreta){
741             glEnable(GL_LIGHT1);
742         }else{
743             glDisable(GL_LIGHT1);
744         }
745
746         if(ences_quadre){
747             glEnable(GL_LIGHT2);
748         }else{
749             glDisable(GL_LIGHT2);
750         }
751
752         glEnable(GL_FOG);
753         {
754         GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};
755         glFogi(GL_FOG_MODE, GL_EXP);
756         glFogfv(GL_FOG_COLOR, fogColor);
757         glFogf(GL_FOG_DENSITY, 0.04);
758         glHint(GL_FOG_HINT, GL_DONT_CARE);
759         }
760
761         /*
762         int submenu = glutCreateMenu(menuapp);
763         glutAddMenuEntry("Incrementar", 3);
764         glutAddMenuEntry("Decrementar", 4);*/
765     }
766 }
```

Dependiendo de las variables ences_x vamos a habilitar o no una luz u otra.

Por último vamos a habilitar el fog, el cual va a tener una densidad baja aunque se puede notar su efecto en la escena.

VALORACIÓN

A lo largo de la asignatura hemos aprendido a utilizar las primitivas de openGL, cómo modificar su color, aspecto,orientación y posición. Además, hemos aprendido a hacer nuestras propias primitivas.

También hemos aprendido las diferentes perspectivas que podemos utilizar en una escena y cual es más conveniente en un caso u otro.

El movimiento de la cámara ha sido la parte que más esfuerzo ha requerido, sobre todo porque nosotros hemos decidido mover la cámara con el lookAt(), en vez de mover la escena dando la impresión que movemos la cámara.

Hemos aprendido a utilizar diferentes fuentes de luz y a combinar los parámetros que podemos asignar a cada luz. Hemos aprendido la importancia de aplicar las normales de los vértices de nuestras primitivas correctamente, ya que estas afectan a cómo la primitiva recibe la luz. Cambiar los materiales convenientemente.

Y hemos aprendido a utilizar texturas y a tratarlas de manera que se integren de manera correcta en nuestra escena.