

PROYECTO FINAL INFORMÁTICA GRÁFICA

ALBERTO ORTEGA VILCHEZ - 75943748S

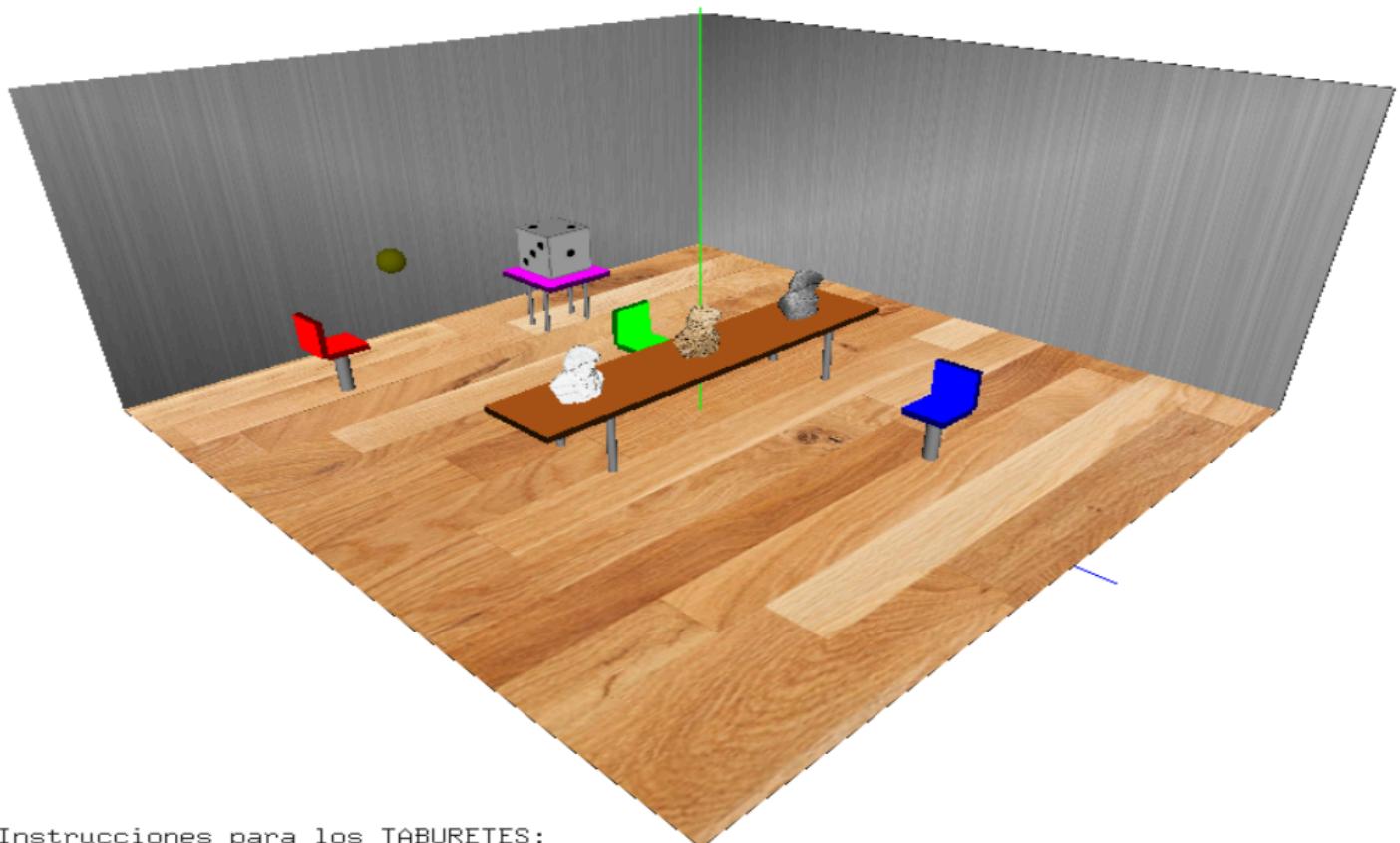
IG. Curso 2024/25. Autor: ALBERTO ORTEGA VILCHEZ

-Por que no lanzas el Dado?

-La cara actual es 1

-Has probado a darle a la tecla ESPACIO?

-CAMBIA LA CAMARA CON CLICK DERECHO



Instrucciones para los TABURETES:

- TABURETE ROJO -> Usamos A,a,Z,z
- TABURETE VERDE -> Usamos S,s,X,x
- TABURETE AZUL -> Usamos D,d,C,c
- Tambien puedes rotarlos usando: 1,2,3

o haciendo click con la ruedecilla del raton sobre cada uno

1 - ÍNDICE

1 - ÍNDICE	2
2 - INTRODUCCIÓN	3
2.1 - Objetos en la escena	3
3 - OBJETIVOS	4
4 - GUÍA DE TECLAS	9
4.1 - Visualización de la escena	9
4.2 - Modos de visualización	10
4.3 - Movimiento taburetes	11
4.3.1 - Movimiento automático	11
4.3.2 - Movimiento manual	12
4.3.3 - Alterar la posición de los taburetes	13
4.4 - Animación pelota	14
5 - GUÍA RATÓN	15
5.1 - Interacción con los taburetes	15
5.2 - Interacción con el Dado	16
5.3 - Interacción con la cámara	16
6 - IMPLEMENTACIÓN	17
6.1 - Estructura de clases	17
6.2 - Implementaciones reutilizadas	17
6.3 - Animaciones	18
6.4.1 - Animación taburetes	18
6.4.2 - Dado Lanzado	20
6.4.3 - Pelota rebotando de manera realista	22
6.4 - Texto por pantalla	24
6.5 - Cámaras	25
7 - CONCLUSIÓN	29

2 - INTRODUCCIÓN

Utilizaré este documento como memoria del proyecto final de prácticas de la asignatura Informática Gráfica.

Describiré resumidamente cómo he abordado cada elemento o funcionalidad implementada, tratando de utilizar todos los conocimientos adquiridos en la asignatura y plasmados en el resto de prácticas de manera individual.

Para empezar, recomiendo prestar atención a la [guía de teclas](#) y de [clics de ratón](#)

Para esta aplicación gráfica interactiva o escena he tratado de simular una sala de reuniones en la que encontramos una larga mesa con bustos de Beethoven con diferentes texturas sobre ella, con taburetes desordenados y esparcidas por la misma, además de un Dado sobre otra mesa más pequeña y una pelota. La interacción más evidente consiste en colocar los taburetes en la mesa principal de forma realista, mediante giros, traslaciones, etc...

A lo largo del documento comentaré el resto de interacciones o animaciones que se pueden ejecutar en mi escena.

A continuación describiré los objetivos que me he marcado para este proyecto final, destacando, de manera general, todas las funcionalidades que he desarrollado.

No obstante, desarrollo aspectos de implementación de aquellos aspectos que considero más importantes al en el apartado: [6 - Implementación](#)

2.1 - Objetos en la escena

DADO → Objeto Malla a partir de un .ply “cubo.ply” con textura

BEETHOVEN 1 → Objeto Malla a partir de un .ply “beethoven.ply” con textura

BEETHOVEN 2 → Objeto Malla a partir de un .ply “beethoven.ply” con textura

BEETHOVEN 3 → Objeto Malla a partir de un .ply “beethoven.ply” con textura

TABURETES DE COLORES → Modelo jerárquico formado por:

- Base (Cilindro)
- Asiento (Cubo escalado)
- Respaldo (Cubo escalado)

PELOTA AMARILLA → Esfera creada a partir de primitivas de glut

SUELO → Objeto Malla a partir de un .ply “cubo.ply” escalado con textura

PARED 1 → Objeto Malla a partir de un .ply “cubo.ply” escalado con textura

PARED 2 → Objeto Malla a partir de un .ply “cubo.ply” escalado con textura

MESAS (marrón y morada) → 4 cilindros y 1 cubo creados con primitivas glut a los que les aplico una serie de transformaciones para simular dos mesas.

3 - OBJETIVOS

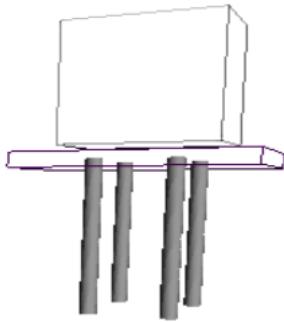
A continuación resumiré los objetivos que he cumplido en este proyecto final. Como he comentado anteriormente, he tratado de aglutinar todo lo aprendido a lo largo del curso, intentando agregar algo de valor añadido.

PRÁCTICA 1

- Geometrías simples con OpenGL
- Primitivas de dibujo usando OpenGL y glut

Re-implemento todas las funcionalidades relativas a los métodos de visualización (Puntos,Alambre,Con Iluminación, Sin Iluminación), gestionando estos modos desde teclado.

Esto me ha sido de gran utilidad para diseñar parte de mi escena. En concreto, las dos mesas que aparecen en ella.



PRÁCTICA 2 - MODELOS POLIGONALES

- Mallas de triángulos
- Objetos PLY
- Cálculo de normales
- Dibujos en modos FLAT y SMOOTH

Como es lógico, utilizo todo lo relativo a los objetos PLY (lectura de ficheros, cálculo de normales en modo FLAT y SMOOTH, representación, etc...) acompañado de lo aprendido a lo largo de la siguiente práctica relativo a texturas.

Apreciable en mi escena con los 3 objetos que hay sobre la mesa marrón, simulando 3 bustos de Beethoven.



PRÁCTICA 3 - MODELOS JERÁRQUICOS

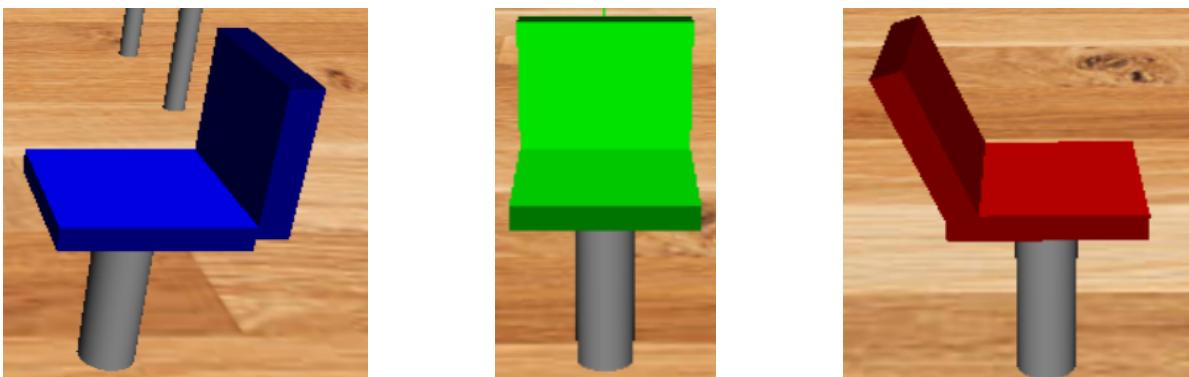
- Diseño e implementación de un modelo jerárquico
- Grafo de escena del modelo
- Animación sencilla

He utilizado el modelo jerárquico ya diseñado para que mi escena simule un aula en la que se encuentran una serie de taburetes (el modelo jerárquico).

Se puede interaccionar con estos taburetes mediante 3 tipos de interacciones y animaciones sencillas implementadas:

- Rotación, cambio de altura e inclinación de los taburetes manualmente mediante teclado.
- Movimiento automático de los 3 taburetes.
- Cambio de posición y giro mediante teclado y ratón

El grafo de escena y grados de libertad seguidos para las dos primeras interacciones con los taburetes son el mismo que el utilizado en la práctica en cuestión.



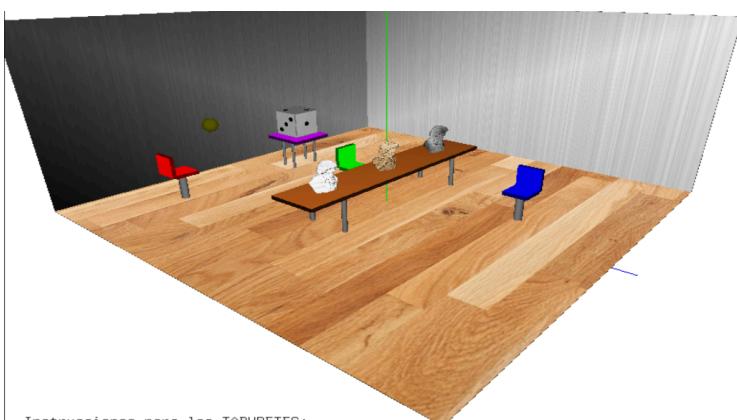
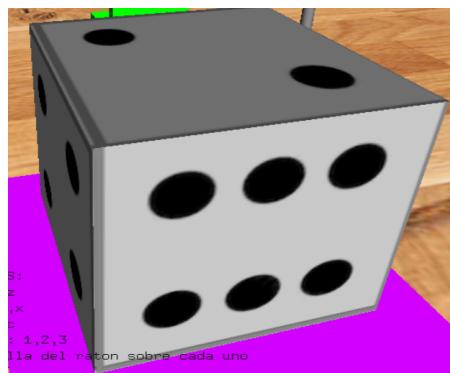
PRÁCTICA 4 - MATERIALES, FUENTES DE LUZ Y TEXTURAS

- Definir fuentes de luz y materiales en OpenGL
- Generar y representar coordenadas de textura en mallas de triángulos
- Visualizar modelos con textura

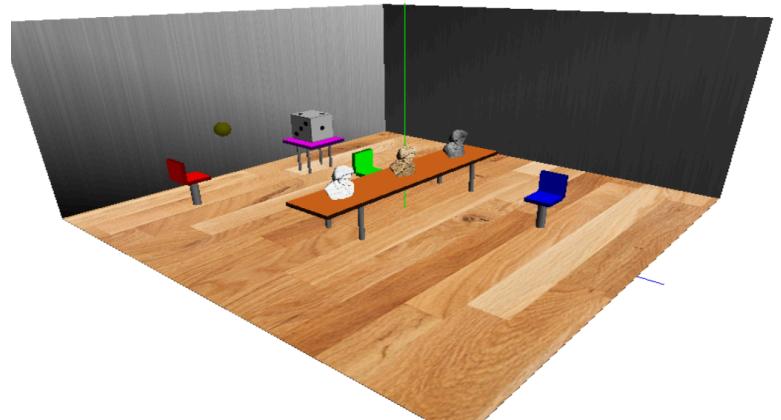
En este caso, como he comentado antes, utilizo todos los conocimientos aprendidos sobre texturas y materiales para dotar de realismo a la escena, haciendo que los 3 bustos de Beethoven adquieran texturas diferentes (mármol, madera y metálica).

Por otro lado, el dado sobre la mesa es representado correctamente.

Mediante [teclado](#) se puede alternar entre diferentes focos de luz definidos.



Instrucciones para los TABURETES:



Instrucciones para los TABURETES:

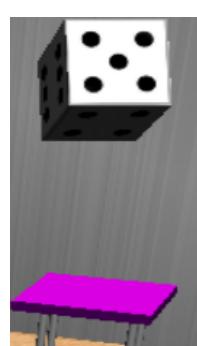
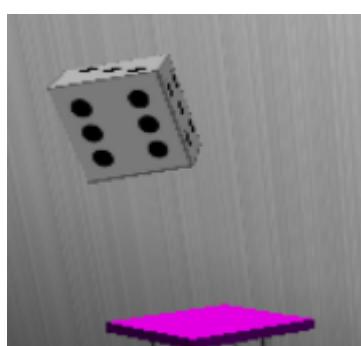
PRÁCTICA 5 - INTERACCIÓN

- Interacciones sencillas, gestionando eventos de entrada de teclado y ratón
- Operaciones de selección de objetos de la escena

En este caso, se trata del apartado que más potencial de mejora tenía ya que no puede desarrollar una buena práctica 5.

Se lleva a cabo correctamente la selección de los objetos interactuables de la aplicación gráfica: Los 3 taburetes y el Dado.

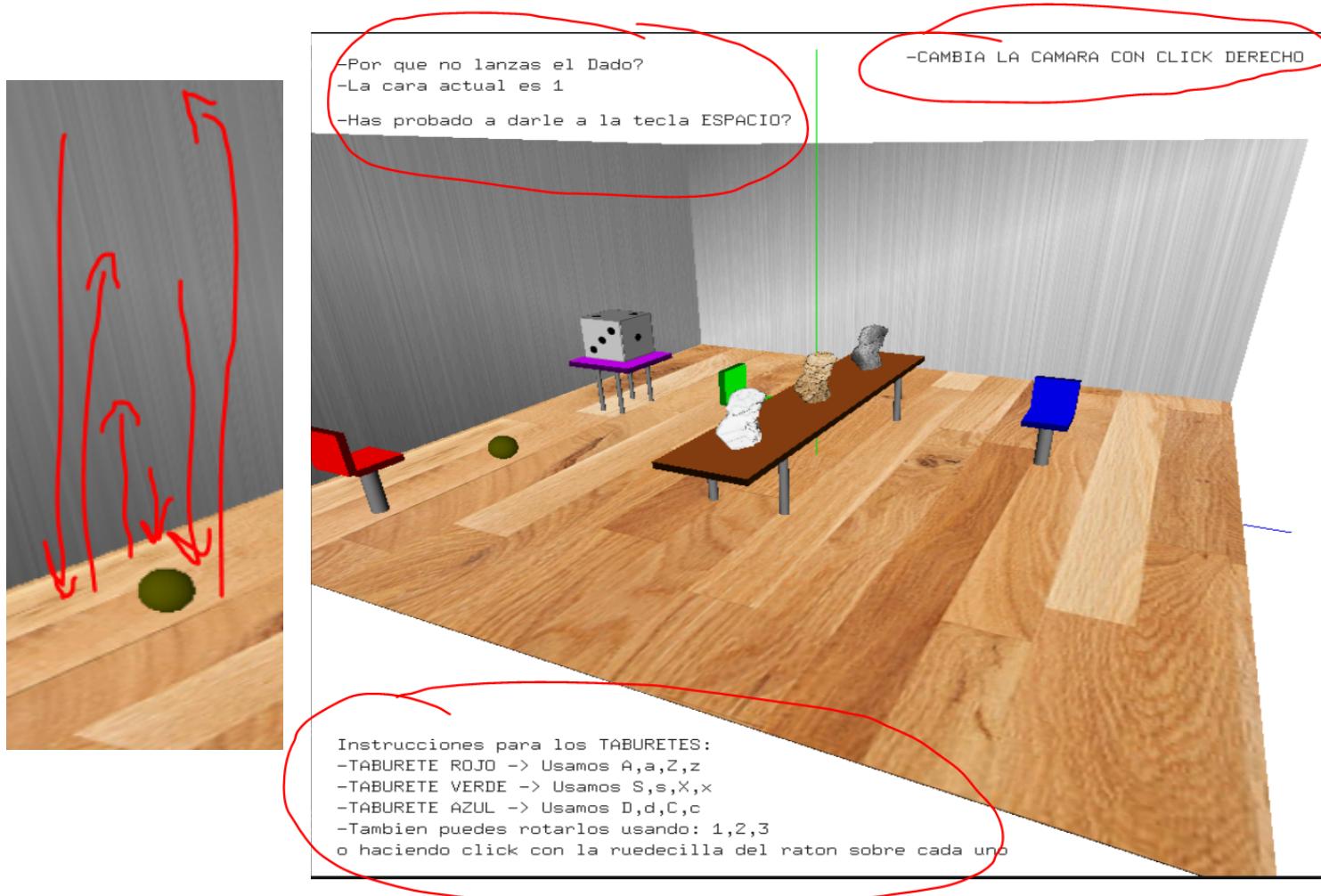
Además, he implementado una nueva animación relativa al Dado, simulando su lanzamiento y alternando mediante rotaciones aleatorias la cara que se muestra, considerándola como el número resultante del lanzamiento.



VALOR AÑADIDO

Además de cumplir con todos los objetivos establecidos en las 5 prácticas, he profundizado en algunas funcionalidades extra con el fin de mejorar mi aplicación gráfica:

- A) Inclusión de texto para mostrar funcionamientos y consejos
- B) Uso de colisiones para la animación de rebote de la pelota
- C) Nuevos puntos de vista con cámaras predefinidas



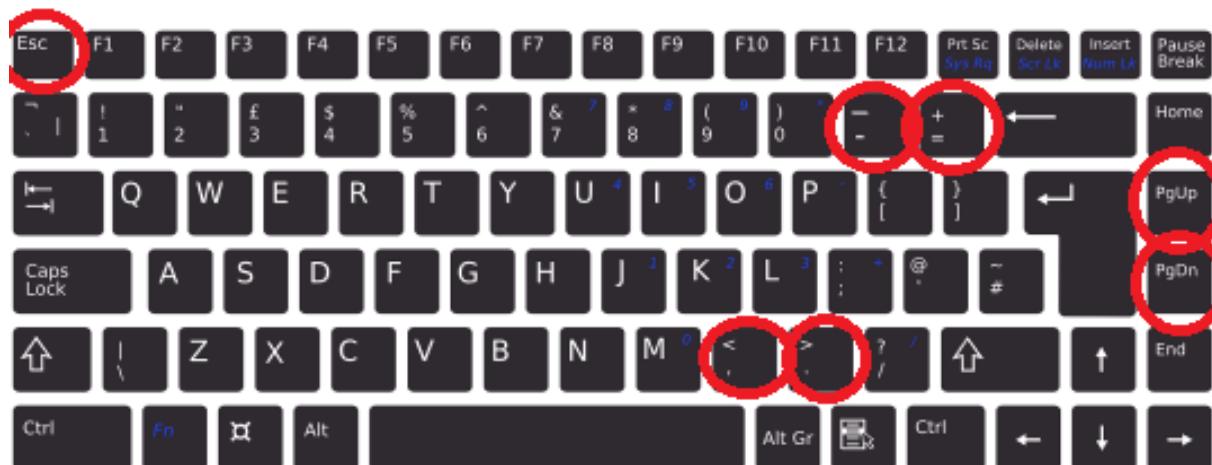
4 - GUÍA DE TECLAS

Utilizaremos las teclas '<' o '>' para mostrar por pantalla todo el panel de ayuda respecto a las teclas con las que se puede interactuar.

No obstante, dejo esta información a continuación también:

4.1 - Visualización de la escena

TECLA	FUNCIÓN
< y >	Mostrar el panel de ayuda por terminal
+ y répag	Acercar la cámara a la escena
- y avpág	Alejar la cámara de la escena
ESC	Cerrar la interfaz gráfica y la ejecución de la escena



4.2 - Modos de visualización

TECLA	FUNCIÓN
P y p	Los objetos se verán “punteados”
L y l	Tan sólo se mostrarán las aristas que forman los objetos
F y f	Los objetos se mostrarán “rellenos”
I y i	Activar y/o desactivar la iluminación de la escena
Q y q	Alternar entre los dos focos de luz

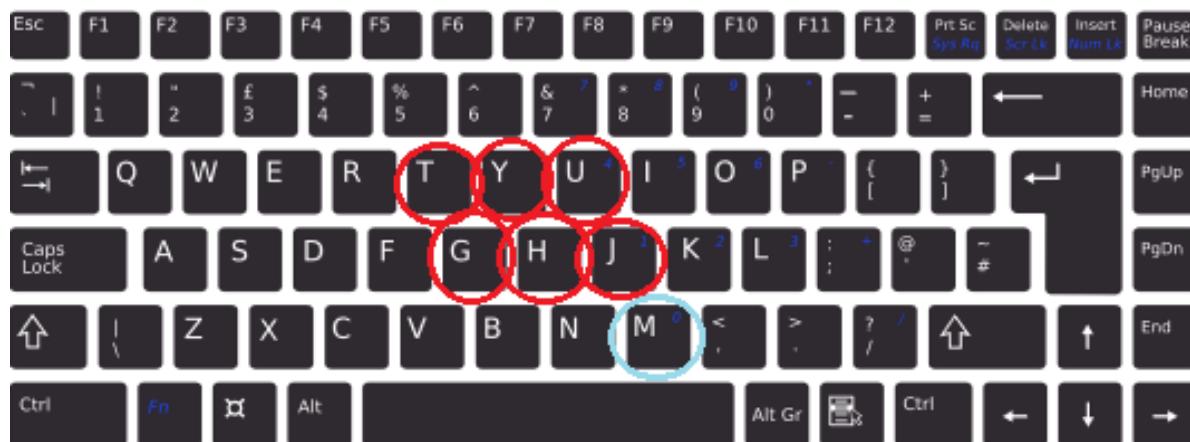


4.3 - Movimiento taburetes

4.3.1 - Movimiento automático

Mientras la animación automática de los taburetes está activa podemos controlar la velocidad con la que las partes del taburete (“Cilindro”, “Asiento” y “Respaldo”) interactúan, aumentando velocidades de subida y bajada, de giro o de reclinamiento del respaldo

TECLA	FUNCIÓN
M y m	Activa y/o desactiva la animación automática de los taburetes.
T y t	(Durante la animación) Aumenta la velocidad en la que los taburetes suben y bajan
G y g	(Durante la animación) Reduce la velocidad en la que los taburetes suben y bajan
Y y y	(Durante la animación) Aumenta la velocidad de giro de los taburetes
H y h	(Durante la animación) Reduce la velocidad de giro de los taburetes
U y u	(Durante la animación) Aumenta la velocidad con la que los taburetes se reclinan
J y j	(Durante la animación) Reduce la velocidad con la que los taburetes se reclinan



4.3.2 - Movimiento manual

Si lo deseamos, podemos girar ligeramente los taburetes manualmente, o incluso definir una altura o grados de reclinamiento de los taburetes utilizando las siguientes teclas.

Para apreciar bien su efecto, se recomienda que la animación automática esté desactivada

TECLA	FUNCIÓN
N (mayúscula)	Aumenta la altura de los taburetes
n (minúscula)	Decrementa la altura de los taburetes
V (mayúscula)	Rota los asientos de los taburetes hacia la izquierda
v (minúscula)	Rota los asientos de los taburetes hacia la derecha
B (mayúscula)	Inclina los respaldos de los taburetes hacia atrás
b (minúscula)	Inclina los respaldos de los taburetes hacia delante



4.3.3 - Alterar la posición de los taburetes

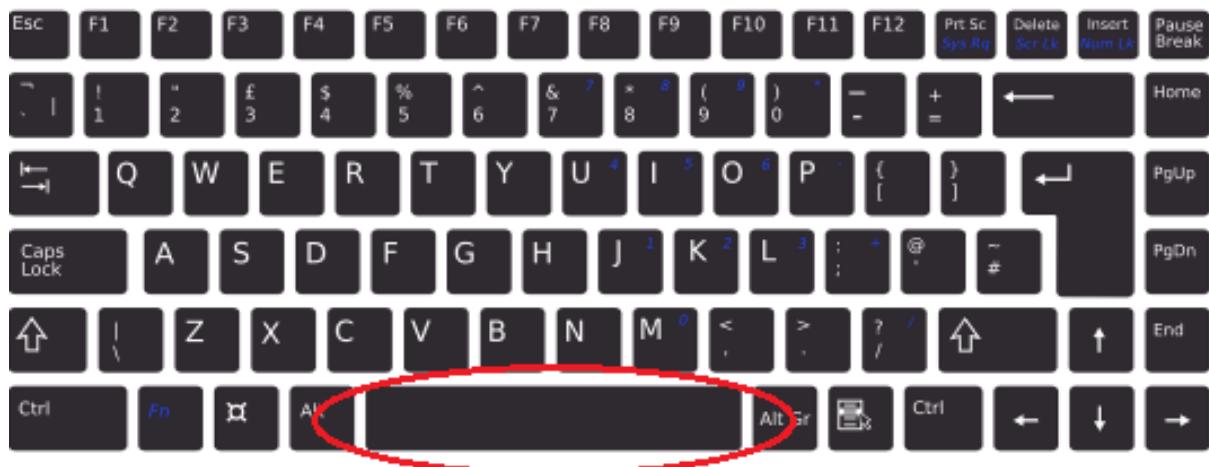
Además de cambiar el estado de los taburetes (su altura, rotación o reclinamiento del respaldo), podemos colocar los taburetes en la escena mediante las siguientes teclas:

TECLA	FUNCIÓN
A	Movemos el taburete (1) en el eje X +5 unidades
a	Movemos el taburete (1) en el eje X -5 unidades
Z	Movemos el taburete (1) en el eje Z +5 unidades
z	Movemos el taburete (1) en el eje Z -5 unidades
S	Movemos el taburete (2) en el eje X +5 unidades
s	Movemos el taburete (2) en el eje X -5 unidades
X	Movemos el taburete (2) en el eje Z +5 unidades
x	Movemos el taburete (2) en el eje Z -5 unidades
D	Movemos el taburete (3) en el eje X +5 unidades
d	Movemos el taburete (3) en el eje x -5 unidades
C	Movemos el taburete (3) en el eje Z +5 unidades
c	Movemos el taburete (3) en el eje Z -5 unidades
1	Giramos el taburete (1) 90°
2	Giramos el taburete (2) 90°
3	Giramos el taburete (3) 90°



4.4 - Animación pelota

TECLA	FUNCIÓN
ESPACIO	La pelota se lanza y esta rebota en el suelo mientras reduce su velocidad



5 - GUÍA RATÓN

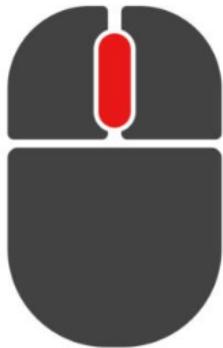
Además de para todas las interacciones definidas que pueden ejecutarse a través de las teclas, he implementado una serie de funcionalidades que se ejecutarán utilizando el ratón

La más sencilla de todas se trata de la selección de objetos.

Como se puede apreciar en la escena, hay una serie de objetos (también hay otros auxiliares las mesas) con los que hemos ido trabajando durante las prácticas. Estos objetos están identificados mediante un ID. Cada vez que pulsemos sobre cualquiera de ellos, podemos apreciar por pantalla sobre qué objeto acabamos de hacer clic, además de con qué parte del ratón lo hemos hecho.

Por otro lado, están definidas las siguientes interacciones:

5.1 - Interacción con los taburetes



“RUEDECILLA”, “SCROLL” o como conocemos en OpenGL: “GLUT_MIDDLE_BUTTON”.

Podremos usar este botón para girar, individualmente, cada Taburete 90°.



“CLICK IZQUIERDO” o “GLUT_LEFT_BUTTON”.

Usaremos este botón para seleccionar los taburetes, mostrándose por pantalla el identificador y número de taburete seleccionado.

5.2 - Interacción con el Dado



Haciendo click izquierdo (“**GLUT_LEFT_BUTTON**”) sobre el Dado apreciaremos cómo este simula la acción de ser lanzado.

Cada vez que se lance, de manera aleatoria, mostrará una cara distinta.

5.3 - Interacción con la cámara



Haciendo click derecho en cualquier parte de la escena se podrá alternar entre 3 posiciones de cámara distintas, cuyas interacciones serán intuitivas.

6 - IMPLEMENTACIÓN

En este apartado destacaré los aspectos relativos a implementación más relevantes de mi proyecto de prácticas y en especial aquellos que suponen una modificación respecto a entregas de prácticas anteriores, mejoras, o nuevas funcionalidades.

En concreto describo la [estructura de clases](#) seguida, [implementaciones reutilizadas](#), [animaciones](#) implementadas y cómo funciona la [funcionalidad relativa al texto que aparece por pantalla](#).

6.1 - Estructura de clases

include	JPEG	plys	Asiento.c	Cilindro.c	Sigo la misma estructura de clases que en las prácticas entregadas a lo largo del cuatrimestre.
Dado.c	entradaTeclado.c	file_ply_stl.cc	lector-jpg.cpp	Makefile	En la carpeta include se encuentran todos los archivos .h (definiciones de métodos, clases, structs, etc...) mientras que en la carpeta raíz se encuentran todas las implementaciones.
Malla.c	modelo.c	mouse.c	practicasIG	practicasIG.c	
Punto3D.c	Respaldo.c	Taburete.c	TabureteIG.png	Triangulo.c	Por otro lado, en las carpetas JPEG y plys se encuentran las texturas usadas en mi escena y los archivos .ply que uso en mis mallas respectivamente.
visual.c					

6.2 - Implementaciones reutilizadas

Como es lógico, he reutilizado todo lo relativo a la carga de texturas, mallas, gestión de .plys, y objetos básicos predefinidos. No obstante, todo ha sufrido mejoras y se han eliminado todas las funciones y métodos definidos que o bien repetían demasiado código o bien no se usaban.

6.3 - Animaciones

6.4.1 - Animación taburetes

En el caso de los taburetes, si bien es cierto que no trato a los taburetes como objetos “con vida propia”, aprovecho los identificadores que le doy a cada Taburete dibujado (**ID_TABURETE1**, **ID_TABURETE2**, **ID_TABURETE3**) para gestionar todas las interacciones que afectan a cualquier Taburete de manera individual mediante switches .

En el caso de los taburetes, las interacciones que se pueden hacer con ellos son 4:

1) MOVIMIENTOS AUTOMÁTICOS

En este caso afecta a todos los taburetes.

Consiste en activar todos sus grados de libertad (el respaldo se reclina, el asiento rota y la base sube y baja) para que se muevan a la vez.

Mediante la **tecla M** activo y desactivo el booleano que bloquea la animación en **idle()**

```
// Actualización de la velocidad de escalado de altura del cilindro
if(animacionActiva){
    if(alturaCilindro>=2.0f || alturaCilindro<=0.5f){
        VEL_Cilindro = -VEL_Cilindro;
    }

    alturaCilindro += VEL_Cilindro;

    // Actualización de la velocidad de rotación del asiento

    rotacionAsiento += VEL_Asiento;

    if(rotacionAsiento >= 360.0f || rotacionAsiento <= -360.0f){
        rotacionAsiento=0.0f;
    }

    // Actualización de la velocidad de rotación (inclinación) del respaldo

    if(inclinacionRespaldo >= 0.0f || inclinacionRespaldo <= -25.0f){
        VEL_Respaldo = -VEL_Respaldo;
    }

    inclinacionRespaldo += VEL_Respaldo;
}
```

Habrá otras letras que se usarán para aumentar y reducir la velocidad con la que los movimientos automáticos se ejecutan

2) MOVIMIENTOS MANUALES

Aunque mediante la **tecla M** podemos activar todos los movimientos automáticos de los taburetes, también podemos alterar sus movimientos relativos a los grados de libertad manualmente mediante otro conjunto de teclas y el propio ratón, pudiendo elegir qué parte de qué taburete concreto alterar.

Además, estos movimientos tendrán un límite, pues trato de simular un movimiento natural

3) CAMBIOS DE POSICIÓN

Se trata de la interacción más interesante relativa a los taburetes, que consiste en moverlos y rotarlos por toda la escena (respetando unos límites) mediante el teclado y el ratón.

El núcleo de esta interacción es el método **moverse(...)** de **modelo.c**.

```
void moverse(int ntaburete,char coordenada,float cantidad){  
    if(coordenada=='x'){  
        switch(ntaburete){  
            case 1:  
                if(comprobarTopeX(ntaburete,cantidad)){  
                    posTaburete1[0]+=cantidad;  
                }else{  
                    posTaburete1[0]+=(cantidad*-1);  
                }  
                break;  
            case 2:  
                if(comprobarTopeX(ntaburete,cantidad)){  
                    posTaburete2[0]+=cantidad;  
                }else{  
                    posTaburete2[0]+=(cantidad*-1);  
                }  
                break;  
            case 3:  
                if(comprobarTopeX(ntaburete,cantidad)){  
                    posTaburete3[0]+=cantidad;  
                }else{  
                    posTaburete3[0]+=(cantidad*-1);  
                }  
                break;  
        }  
    }  
}
```

Este método es llamado al pulsar sobre las teclas determinadas pasando como parámetros el ID del taburete a mover (1,2 o 3), la coordenada sobre la que se va a mover ('x' o 'z') y la cantidad de unidades que va a avanzar.

Para cada llamada se comprobará esta información.

Para decidir si el movimiento es válido o no utilizo **comprobarTope\$(...)**

```
case 'A':  
    moverse(1,'x',5);  
    break;  
case 'a':  
    moverse(1,'x',-5);  
    break;  
case 'S':  
    moverse(2,'x',5);  
    break;  
case 's':
```

```
bool comprobarTopeX(int ntaburete,float cantidad){  
    bool valido;  
    switch(ntaburete){  
        case 1:  
            if(posTaburete1[0]+cantidad<30 && posTaburete1[0]+cantidad>-30){  
                valido=true;  
            }else{  
                valido=false;  
            }  
            break;  
        case 2:
```

Este método comprueba que el movimiento a realizar no se exceda de los límites de la habitación. En caso de que sea válido, la posición del taburete en cuestión se actualizará añadiendo a su `posTaburete[i][0]` (si el movimiento es en el eje X) o en `posTaburete[i][2]` (si el movimiento es en el eje Z) la cantidad de unidades a desplazarse.

En caso de que no sea válido porque sumar esa cantidad implique sobrepasar los límites de la habitación, se interpretará como un rebote restando la cantidad en lugar de sumarla.

Existen por tanto dos métodos comprobadores de límites o "topes" (**comprobarTopeX(...)** y **comprobarTopeZ(...)**).

4) SELECCIÓN y ROTACIÓN CON RATÓN

Al igual que para el caso del [Dado](#), cada objeto con el que se puede interactuar con el ratón tiene un ID asociado con el que poder identificarlo al hacer click sobre él valiéndome de los métodos **pick** y **colorSelección** implementados en la práctica anterior.

```
#define ID_TABURETE1 1 //Identificador del primer taburete
#define ID_TABURETE2 2 //Identificador del segundo taburete
#define ID_TABURETE3 3 //Identificador del tercer taburete
#define ID_DADO 4 //Identificador del dado
```

En este caso, los taburetes se identifican con el 1,2 y 3 respectivamente.

```
if(pick(x,y,&id)){
    // printf("Objeto seleccionado: %d\n",id);
    setModoSeleccion(true);

    switch(id){
        case ID_TABURETE1:
            printf("\n(CENTRO) Has seleccionado el Primer Taburete\n");
            girar(1);
            break;
        case ID_TABURETE2:
            printf("\n(CENTRO) Has seleccionado el Segundo Taburete\n");
            girar(2);
            break;
        case ID_TABURETE3:
            printf("\n(CENTRO) Has seleccionado el Tercer Taburete\n");
            girar(3);
            break;
        case ID_DADO:
            printf("\n(CENTRO) Has seleccionado el Dado\n");
            break;
        default:
            printf("\nNo estás haciendo click central sobre ningún objeto\n");
            break;
    }
}
```

Cuando alguno de los taburetes es pulsado con la ruedecilla del ratón (GLUT_MIDDLE_BUTTON), se llama al método definido en **modelo.c** “**girar(...)**”

```
void girar(int ntaburete){
    switch(ntaburete){
        case 1:
            angulo1+=90.0f;
            break;

        case 2:
            angulo2+=90.0f;
            break;

        case 3:
            angulo3+=90.0f;
            break;

        default:
            printf("\nNingún objeto está girando");
            break;
    }
}
```

Este método aumenta 90° el ángulo de giro del taburete seleccionado, simulando así que el asiento gira.

6.4.2 - Dado Lanzado



El Dado se trata de un objeto Malla con un .ply asociado que dibuja un cubo.

A este cubo le he asociado la textura de Dado que utilicé en prácticas anteriores.

En la escena se encuentra sobre una mesa morada dibujada mediante primitivas de **glut** y aparece con una posición preestablecida.

```
Dado dado(4.0f, ID_DADO, 1);
bool dadoEnAnimacion=false;

float carasDado[6][2] = {
    {0.0f, 0.0f},
    {90.0f, 0.0f},
    {-90.0f, 0.0f},
    {0.0f, 90.0f},
    {0.0f, -90.0f},
    {180.0f, 0.0f}
};

float ajusteAlturaCaras[6] = {0.0f, 2.0f, 2.0f, 0.0f, 0.0f, 0.0f};
float ajusteProfundidadCaras[6] = {0.0f, -2.0f, 2.0f, 0.0f, 0.0f, 0.0f};

float alturaDado=0.0f;
float rotacionDadoX=0.0f;
float rotacionDadoY=0.0f;
float rotacionDadoZ=0.0f;
```

carasDado, **ajusteAlturaCaras** y **ajusteProfundidadCaras** serán utilizados para simular la acción de cambiar la cara visible del dado cada vez que se lance.

Uso los dos últimos para realizar ajustes en cuanto a la posición del dado tras lanzarlo, de forma que quede sobre la mesa en la misma posición que se encuentra en el estado inicial.

```
if(pick(x,y,&id)){
    // printf("Objeto seleccionado: %d\n", id);
    setModoSeleccion(true);

    switch(id){
        case ID_TABURETE1:
            printf("\n(IZO) Has seleccionado el Primer Taburete");
            break;
        case ID_TABURETE2:
            printf("\n(IZO) Has seleccionado el Segundo Taburete");
            break;
        case ID_TABURETE3:
            printf("\n(IZO) Has seleccionado el Tercer Taburete");
            break;
        case ID_DADO:
            printf("\n(IZO) Has seleccionado el Dado\n");
            if(!getDadoEnAnimacion()){
                setDadoEnAnimacion(true);
                setAlturaDado(0.0f);
                setRotacionDadoX(0.0f);
                setRotacionDadoY(0.0f);
                setRotacionDadoZ(0.0f);

                int nuevaCara = rand() % 6;
                cambiarCaraVisible(nuevaCara);
                printf("\nNUEVA CARA -> %d", nuevaCara+1);
            }
            break;
    }
}
```

Al hacer click izquierdo sobre él, en cualquier momento, se muestra por pantalla que ha sido seleccionado.

En caso de que no esté en mitad de la propia animación, se establece que el dado se encuentra en animación y se ponen los valores de altura y rotación en los diferentes ejes del dado a su estado inicial (0.0f)

Además, mediante un número aleatorio se simula la acción de alterar la cara resultante de lanzar el dado, que se mostrará por pantalla.

```
if(dadoEnAnimacion){

    if(alturaDado<10.0f){
        alturaDado+=0.25f;
    }else if(alturaDado>10.0f){
        alturaDado-=0.15f;
    }else{
        printf("\nYA SE HA LANZADO EL DADO");
        dadoEnAnimacion=false;
        alturaDado=0.0f;
        rotacionDadoX=0.0f;
        rotacionDadoY=0.0f;
        rotacionDadoZ=0.0f;
    }
}
```

Cuando tras hacer click sobre el Dado la variable **dadoEnAnimacion** se pone a true, aumento y decremento los valores de **alturaDado** en función de la posición en la que

se encuentre haciendo parecer así que el Dado sube y baja.

Además, en el momento en que está subiendo y alcanza cierta altura realiza una serie de rotaciones en los 3 ejes.

De esta forma el dado simula perfectamente la acción de ser lanzado.

Además, puesto que la **caraVisible** se define aleatoriamente cuando se selecciona el dado, en función del valor de esta variable se modificarán los valores de **rotacionDadoX** y **rotacionDadoY** de manera que sea esa cara la que se aprecie como resultado de lanzar el dado.

Todas estas modificaciones se aprecian cuando se dibuja el Dado en **dibujaEscena** gracias al uso de variables de desplazamiento en el momento de dibujo del objeto.

```
// DADO
int caraVisible = dado.getCarVisible();
float ajusteAltura = ajusteAlturaCaras[caraVisible-1];
float ajusteProfundidad = ajusteProfundidadCaras[caraVisible-1];

glPushMatrix();
    glTranslatef(4.0f,5.5f + alturaDado + ajusteAltura,-20.0f + ajusteProfundidad);
    glRotatef(rotacionDadoX,1,0,0);
    glRotatef(rotacionDadoY,0,1,0);
    glRotatef(rotacionDadoZ,0,0,1);
    dado.draw();
glPopMatrix();
```

6.4.3 - Pelota rebotando de manera realista

Para añadir interactividad a la escena he implementado una animación que simula el rebote de una pelota al ser lanzada de forma realista.

La Pelota se define en **modelo.c** como una estructura que contiene las propiedades físicas y geométricas necesarias para la animación en cuestión. Los valores de estas propiedades se inicializan en **initModel()**

```

typedef struct{
    float y;
    float velY;
    float radio;
    float gravedad;
    float rebote;
} Pelota;

Pelota pelota = {2.0f,0.0f,0.2f,0.01f,0.8f};

// PELOTA REBOTANDO //

pelota.y = 2.0f;
pelota.velY = 0.0f;
pelota.radio = 0.2f;
pelota.gravedad = 0.01f;
pelota.rebote = 0.8f;
setPelotaEnAnimacion(false);

```

La animación se activa al pulsar la tecla **ESPACIO**. Cuando esto sucede, se llama a la función **setPelotaEnAnimacion(true)** para reiniciar los valores de altura y velocidad de la pelota.

```

case ' ':
    setPelotaEnAnimacion(true); // Inicia la animación de la pelota (rebote)
    }

void setPelotaEnAnimacion(bool p){
    if(p){
        pelota.y=4.0f;
        pelota.velY=0.1f;
    }
    pelotaEnAnimacion=p;
}

bool getPelotaEnAnimacion(){
    return pelotaEnAnimacion;
}

```



Serán los valores de estos dos atributos (**y** y **velY**) los que establezcan la altura a la que ascenderá la pelota.

- **FÍSICA DE GRAVEDAD**

Para simular el ascenso, descenso y rebote de una forma realista, en la función **idle()** se ejecuta el siguiente comportamiento mientras el continela **pelotaEnAnimacion** no bloquee la animación (sucederá cuando la animación finalice)

```

if(pelotaEnAnimacion){
    pelota.velY-= pelota.gravedad;
    pelota.y += pelota.velY;

    if(pelota.y <= pelota.radio){
        pelota.y = pelota.radio;
        pelota.velY = -pelota.velY * pelota.rebote;
    }
}

```

Como podemos observar, la **gravedad** afecta a la **velocidad**

reduciéndola mientras la animación está en curso.

Por otro lado, la altura (**y**) de la pelota se va actualizando en función de la **velocidad** (de ahí que si la **velY** tuviera un valor muy elevado la altura a la que ascendería la pelota sería muy alta).

Si la altura de la pelota '**y**' llega a ser menor o igual que su **radio** significaría que esta ha tocado el "suelo". Cuando esto sucede corrijo la altura de la pelota para que esta no traspase el suelo e invierto la **velocidad**.

La clave del **rebote** reside precisamente en que cuando la pelota llega al suelo multipliquemos la velocidad por un **coeficiente de rebote** para que esta **velocidad** sufra un ligero decremento. Así, la pelota no subirá tan arriba como en el anterior rebote, hasta que finalmente se pare.

Cuando la **velocidad** y la **altura** de la pelota se reduzcan lo suficiente, se dará la animación por finalizada pues significará que la pelota está parada en el suelo y se llamará a **setPelotaEnAnimacion(false)**.

6.4 - Texto por pantalla

Para mostrar texto por pantalla hago uso de algunas funciones de OpenGL y GLUT en una función implementada en modelo.c llamada:

```
void dibujaTexto(const char *text, int length, int x,int y)
```

Tal y como intuitivamente se puede apreciar, la función será llamada para dibujar textos manualmente definidos en **dibujaEscena()** tantas veces como líneas de texto muestro.

En concreto, la función recibe como parámetros un texto, el número de caracteres y la posición en el plano XY que representa la ventana de visualización donde se mostrará el texto.

```
// Textos
void dibujaTexto(const char *text, int lenght, int x, int y){
    glMatrixMode(GL_PROJECTION);
    double *matrix = new double[16];
    glGetDoublev(GL_PROJECTION_MATRIX,matrix);
    glLoadIdentity();
```

Cambio la matriz de proyección para configurar configurar un sistema de coordenadas 2D usando **glMatrixMode(GL_PROJECTION)**. Esto permite posicionar el texto directamente en coordenadas de la propia ventana ([0,800] en X y [0,600] en Y).

La matriz de proyección original se guarda en **double *matrix** para restaurarla después.

Llamo a **glLoadIdentity()** para que las transformaciones previas no afecten a las que se harán a continuación (entre glPushMatrix y glPopMatrix)

La transformación consiste en posicionar el inicio del texto en las coordenadas **x,y** mediante la función **glRasterPos2i** y rendirezar cada carácter, iterando sobre el total uno a uno y dibujándolo usando la fuente **GLUT_BITMAP_8_BY_15**.

Al finalizar la transformación restauro la configuración original de la matriz de proyección para no afectar al resto de la escena.

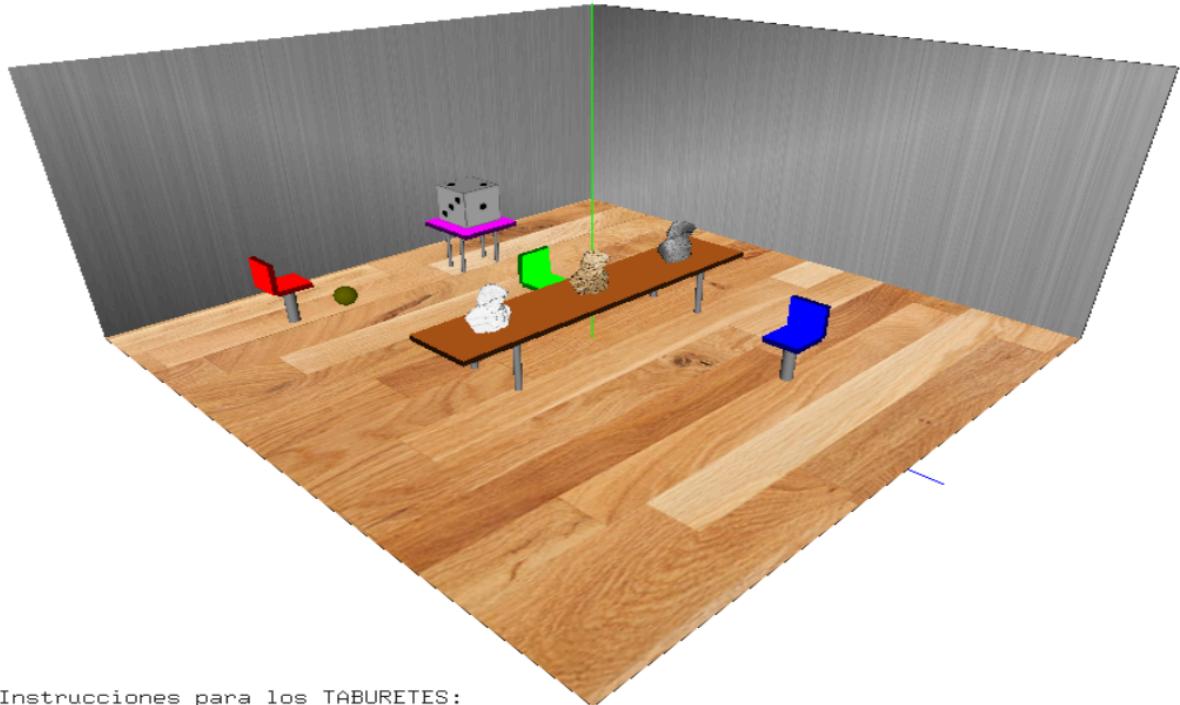
```
// TEXTO
glPushMatrix();
string text1;
text1="Instrucciones para los TABURETES:";
// glColor3f(1,0,0);
dibujaTexto(text1.data(),text1.size(),20,90);
glPopMatrix();
glPushMatrix();
string text2;
text2="-TABURETE ROJO -> Usamos A,a,Z,z";
// glColor3f(1,1,0);
dibujaTexto(text2.data(),text2.size(),20,75);
glPopMatrix();
```

Cada línea de texto mostrada sigue este formato. Defino el texto a mostrar en una variable (esto tampoco es estrictamente necesario) y llamo a la función anteriormente comentada pasando los parámetros pertinentes.

6.5 - Cámaras

Por último, he implementado la posibilidad de alternar entre 3 cámaras distintas (o posiciones de la cámara) siendo una de estas la original.

Además, cada una de ellas gestionará sus movimientos de manera intuitiva usando las mismas teclas en todos los casos.



Instrucciones para los TABURETES:

Cada cámara tiene una posición (eye), punto al que mira (center) y un vector “arriba (up) concretos, almacenados en un array de forma que cada componente de este represente el conjunto de propiedades de cada cámara definidos en **visual.c**.

```
// Al declarar las siguientes variables como "extern" en practicasIG.h podemos redefinirlas
int camaraActual=2;

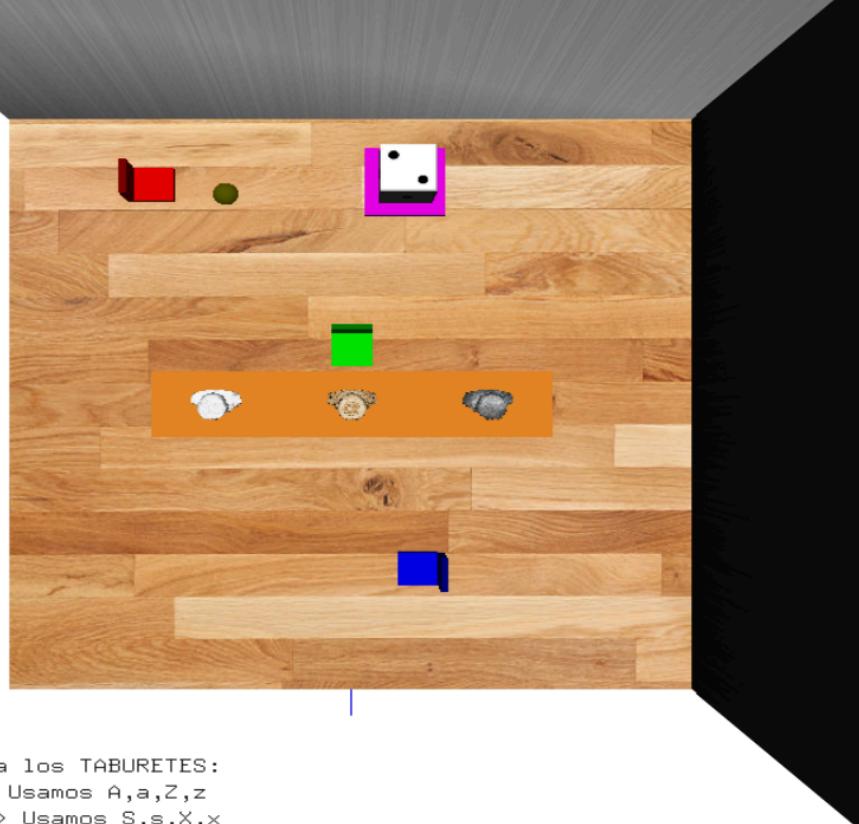
float camX=0.0f, camY=10.0f, camZ=0.0f;
float lookX=0.0f, lookY=0.0f, lookZ=0.0f;
float upX=0.0f, upY=1.0f, upZ=0.0f;

float eyeX = -D * sin(view_roty * M_PI / 180.0) * cos(view_rotx * M_PI /180);
float eyeY = D * sin(view_rotx * M_PI / 180.0);
float eyeZ = D * cos(view_roty * M_PI / 180.0) * cos(view_rotx * M_PI / 180.0);

PosicionCamara camaras []={
    {0.0f,10.0f,0.0f,0.0f,0.0f,0.0f,0.0f,-10.0f},
    {-10.0f,7.0f,0.0f,0.0f,6.5f,0.0f,0.0f,1.0f,0.0f},
    {eyeX,eyeY,eyeZ,0.0f,0.0f,0.0f,0.0f,1.0f,0.0f}
};
```

Además, las variables cam, look y up sirven para configurar la posición de la cámara dinámicamente.

probado a darle a la tecla ESPACIO?



ucciones para los TABURETES:
RETE ROJO -> Usamos A,a,Z,z
RETE VERDE -> Usamos S,s,X,x
RETE AZUL -> Usamos D,d,C,c

Mediante un evento de clic derecho en **mouse.c** cambio entre las cámaras predefinidas

```
if(pick(x,y,&id)){  
    // printf("Objeto seleccionado: %d\n",id);  
    camaraActual = (camaraActual+1)%3;  
    PosicionCamara cam = camaras[camaraActual];  
  
    camX = cam.eyeX;  
    camY = cam.eyeY;  
    camZ = cam.eyeZ;  
    lookX = cam.centerX;  
    lookY = cam.centerY;  
    lookZ = cam.centerZ;  
    upX = cam.upX;  
    upY = cam.upY;  
    upZ = cam.upZ;  
  
    printf("\nCambiando a posición de cámara %d\n",camaraActual);  
  
    if(camaraActual==0 || camaraActual==1){  
        actualizarCamara();  
    }  
  
    setModoSeleccion(true);  
}
```

Cada clic derecho cambiará a la siguiente cámara definida en el array **camaras**.

La **camaraActual** o cámara activa actualizará sus parámetros.

En el caso de que la nueva cámara actual sea una de las dos primeras (son las que he creado yo manualmente) la actualización se hará mediante la función **actualizarCamara()**, que simplemente le da a cada variable al valor correspondiente al estado actual para aplicar los cambios.

```
void actualizarCamara(){  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
    gluLookAt(camX,camY,camZ,lookX,lookY,lookZ,upX,upY,upZ);  
}
```



En el caso de que la nueva cámara actual sea la que se encuentra en la posición 2, se gestionará de la manera por defecto que venía ya definida anteriormente.

Para que esto sea así he modificado también lo que sucede al pulsar sobre las teclas “especiales” en la función **void especial(...)** de **entradaTeclado.c**:

```

void especial (int k, int x, int y){

  switch (k){
    case GLUT_KEY_UP:
      if(camaraActual==0){camZ -= 3.0f;}
      if(camaraActual==1){camX += 3.0f;}
      if(camaraActual==2){
        rotxCamara += 5.0; // Cursor arriba + rotacion x
        if (rotxCamara > 360) rotxCamara -= 360;
      }

    break;
  }
}

```

En función de la cámara actual en la que nos encontramos, se modificará una componente u otra (por un lado, para que sea más intuitivo, y por otro, para que no afecte a la gestión de la cámara original la cual ya estaba bien implementada).

De esta manera he logrado crear otras 2 cámaras sin alterar el funcionamiento de la original.

7 - CONCLUSIÓN

El desarrollo de este proyecto final me ha permitido consolidar conocimientos adquiridos durante el cuatrimestre en la asignatura, integrando aspectos esenciales como la representación de modelos jerárquicos, uso de texturas, implementación de interacciones mediante teclado y ratón...

A pesar de algunas limitaciones que he tenido a lo largo del curso respecto a, principalmente, la selección de objetos, el proyecto incorpora mejoras significativas a mi parecer con las prácticas previas.

Además, he tratado de hacerlo lo más escalable posible, puesto que, por ejemplo, para añadir nuevos taburetes por ejemplo o nuevos focos de cámara o luz tan sólo habría que añadir nuevas entradas en algunos métodos prácticamente calcadas a los que ya se pueden encontrar.

En resumen, creo que he logrado cumplir con los objetivos planteados y he conseguido añadir algún aspecto innovador (al menos, desde mi punto de vista teniendo en cuenta mis conocimientos previos) gracias a procesos de investigación.