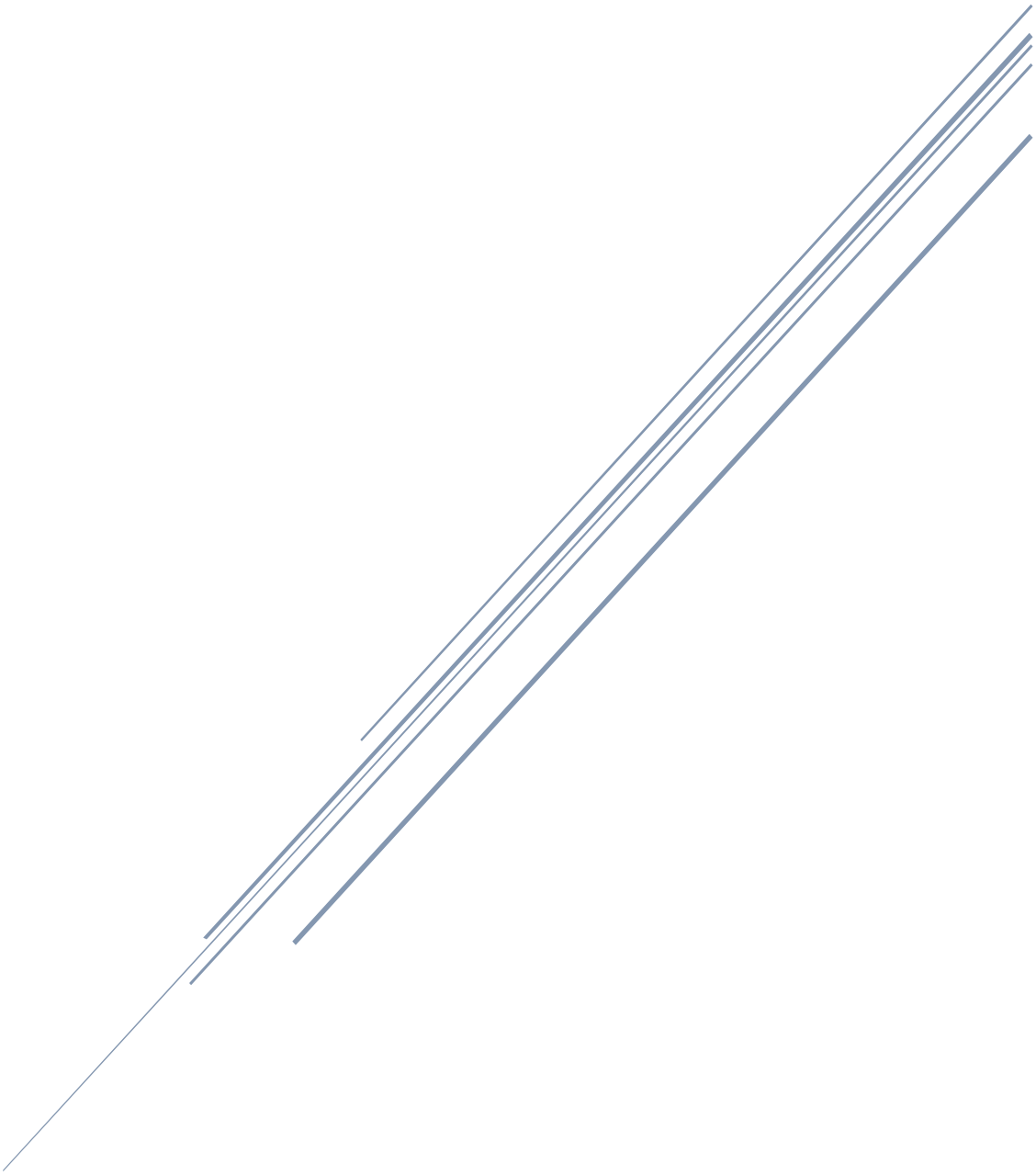
# MEMORIA DE LA PRÁCTICA

## Informática Gráfica 2021-22



Andreas Manuel Korn – X4890193W
Sergio Vega García — 43480752B

# Contenido

## Introducción

En este documento se encuentra la información pertinente a la práctica de la asignatura Informática Gráfica cursada en la Universitat de les Illes Balears el curso 2021-22. La meta de la práctica es entender los fundamentos de los gráficos producidos por ordenador y entender lo que hay debajo de programas como Blender u otros programas de renderización gráfica.

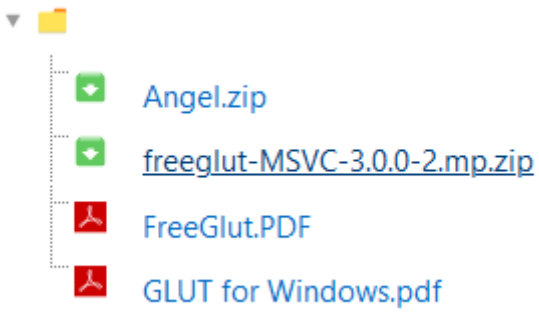
En este caso se ha hecho mediante la creación de una escena 3D con una lámpara articulada que se mueve sobre una mesa.

## Instalación

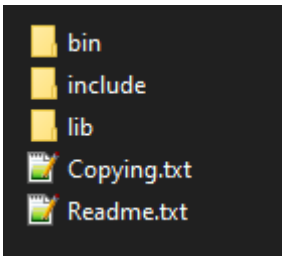
Para el correcto funcionamiento de este programa se recomienda utilizar el IDE **Visual Studio Code 2022**, ofreciéndose los siguientes pasos a modo de tutorial de instalación de las librerías.

1. Descargamos la librería. La que utilizamos se encuentra en el apartado de recursos del Aula Digital de Informática Gráfica, dentro de la carpeta freeglutdocs.


Nos interesa únicamente freeglut-MSVC-3.0.0-2.mp.zip



2. Al extraer los ficheros del comprimido, deberían quedarnos estos directorios




3. El directorio **bin** incluye las dll de freeglut. En un sistema de 64 bits, debemos coger el dll incluido en **bin\x64** y pegarlo en **C:\Windows\System32**.
   Para un sistema de 32 bits, el dll a usar es el que se encuentra en el propio directorio **bin**.
4. Los contenidos de **include** deben ubicarse en una carpeta de fácil acceso. Recomendamos pegarlos en **C:\freeglut** (de tal modo que el directorio del fichero freeglut.h quede **C:\freeglut\GL\freglut.h**) pues el proyecto está ya preparado para encontrarlo automáticamente de ese modo.
   En caso de que no sea posible, se puede guardar en cualquier otro directorio de fácil acceso del sistema: más adelante se darán pautas para que el proyecto lo lea correctamente.
5. El directorio lib incluye dos ficheros:
   a. **lib\freeglut.lib** debe ser copiado en **[directorio de instalación de Microsoft Visual Studio][1]\2022\Community\VC\Auxiliary\VS\lib\x86**
   b. **lib\x64\freeglut.lib** debe ser copiado en **[directorio de instalación de Microsoft Visual Studio][1]\2022\Community\VC\Auxiliary\VS\lib\x64**
6. En caso de que los contenidos de include se ubiquen en un sitio diferente a **C:\freeglut**: Entrar a la solución, acceder a la configuración del proyecto y en el apartado de C/C++, en la opción de Directorios de inclusión adicionales reemplazar **C:\freeglut** por el directorio del paso 4. Es importante hacer este paso para todos los proyectos (o etapas) de la solución.
7. Asegurarse de que el proyecto de inicio de la solución sea la etapa 6, y ejecutar.

## Practicas.h

Hemos usado esta librería para definir las constantes y algunas primitivas utilizadas a lo largo de la práctica, así como los *includes* necesarios para el proyecto. A lo largo de la práctica utilizaremos freeglut.h, iostream y math.h.

La librería freeglut.h incluye de por sí las librerías gl.h, glu.h y glut.h, pero se mantienen las tres comentadas para que, en el caso de que freeglut.h fallara, se pudieran incluir las tres por separado cómodamente.

Practicas.h es un fichero que ha estado en constante evolución desde la primera a la sexta etapa, y por tanto incluye constantes que no se utilizan hasta la sexta etapa, o primitivas que quedaron obsoletas allá por la quinta. Debido a la necesidad de preservar cada etapa por separado, las primitivas obsoletas no se han borrado.

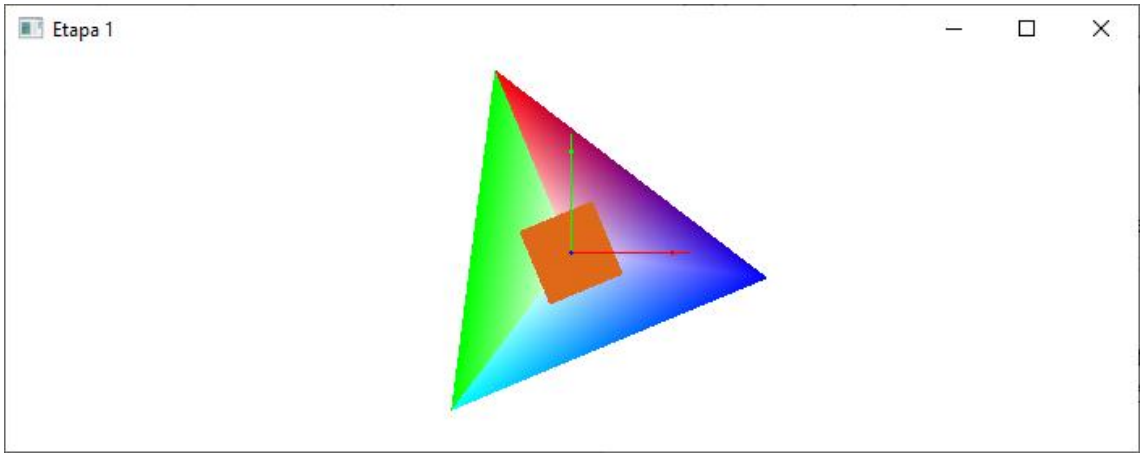
[1] Por defecto, el directorio de instalación de Microsoft Visual Studio 2022 es C:\Archivos de Programa\Microsoft Visual Studio

## Etapa 1

Esta etapa consiste únicamente en preparar el *workplace* para poder comenzar a trabajar en el resto de las etapas. Tras seguirse los pasos del apartado Instalación, podemos ver que se ejecuta correctamente.

En esta etapa fue que nosotros tratamos con el aspecto ratio. Si se redimensiona la pantalla, se puede ver que la proporción de la figura se mantiene constante. Esto es posible gracias a la función `reshape`, en la que guardamos la proporción de la imagen y modificamos las dimensiones acorde a la redimensión de la ventana.

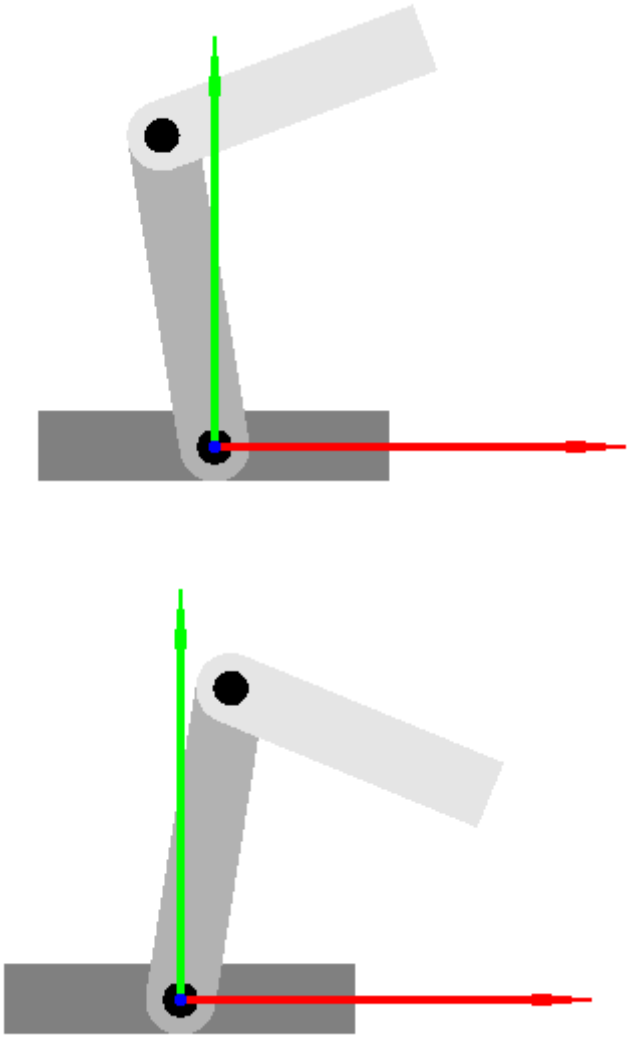
En la imagen inferior se puede ver cómo la ventana ha sido redimensionada solamente a lo alto, pero la figura se mantiene correctamente.




El aspecto ratio de la ventana varía, pero la de la imagen, no.

## Etapa 2

En esta etapa empezamos a trabajar con imágenes en 2D. Para familiarizarnos con el sistema, hemos creado un simple brazo articulado que se va moviendo hacia delante y hacia atrás. Para hacer más visualmente atractivo el punto de articulación fue necesario programar nuestra propia primitiva que se ubica en Practicas.h: `drawEllipse`, a la que se le deben pasar las coordenadas del centro, la longitud del radio mayor y del menor, así como el ángulo que se quiere dibujar (de 0 a 360).

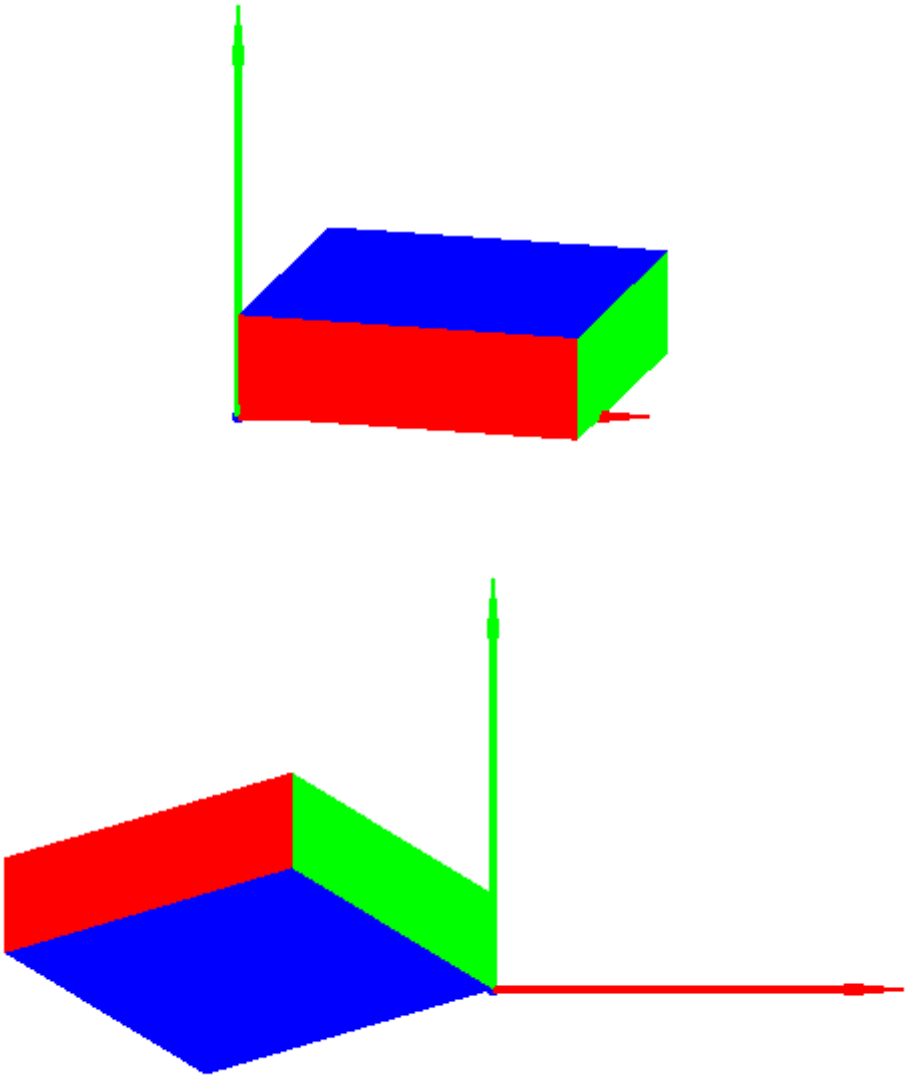


Dos de las posibles posiciones del brazo mecánico, demostrando su articulación.

## Etapa 3

En esta etapa hacemos por fin el salto al 3D. A pesar de que la perspectiva está fija, se han implementado cuatro teclas especiales capaces de rotar la figura alrededor de los dos ejes paralelos al plano de visualización. Con las teclas izquierda y derecha podemos hacer que la figura rote alrededor del eje Y, mientras que arriba y abajo permite rotar alrededor del eje X.

También ha sido necesario activar `GL_DEPTH_TEST`, para que las caras visibles sean siempre las más cercanas a la cámara.



Dos posiciones diferentes del cubo para demostrar el movimiento.

# Etapa 4

En la etapa 4 implementamos el movimiento de la cámara. Hemos establecido dos tipos de movimiento diferentes: el paneo de la cámara, que está activado por defecto, y el movimiento de ésta, que se activa pulsando F2.

También empezamos la base de la escena final en un estado muy inicial, para poder trabajar mejor con la cámara y asegurar que la perspectiva funciona correctamente.

Los controles de la cámara son los siguientes:

## Paneo de la cámara

Este modo de cámara, que es el modo por defecto y se activa pulsando F1, permite girar la cámara sobre sí misma utilizando las flechas del teclado.

La implementación de este modo de cámara supuso un quebradero de cabeza al principio, pero luego resultó ser muy fácil de implementar cuando caímos en que era tan fácil como hacer una transformación de rotación del punto centro alrededor del ojo. Para ello, trasladamos el centro de tal modo que el ojo esté en (0, Y, 0), hacemos la transformación de rotación y luego deshacemos la traslación previa.

El paneo hacia abajo y hacia arriba fue mucho más fácil: únicamente hace falta incrementar y decrementar el valor Y del punto centro, ya que no es necesario poder hacer un giro completo de arriba abajo.

Cabe destacar que en esta etapa y la siguiente la tecla derecha hace que la cámara gire hacia la izquierda y viceversa, pero después de tanto tiempo trabajando con la cámara nos dimos cuenta de que no se nos hacía intuitivo y decidimos invertirlo. Al final, este tipo de cosas siempre cae en preferencia personal y por ello muchos programas con movimiento de cámara como videojuegos ofrecen la opción de invertir los controles.

## Movimiento de la cámara

Con este modo de cámara podemos mover la cámara por la escena.

La implementación de este tipo de cámara resultó ser lo más complicado de la etapa, no solo en implementación sino en llegar a la lógica necesaria.

Al establecer una perspectiva, debemos pasar por parámetros tres tríos de valores: eye, center y up, cada uno con sus valores X, Y, Z correspondientes. Sin embargo es importante destacar que, aunque parezcan tres valores del mismo tipo, en realidad se tratan de un punto *eye*, un punto *center* y un vector *up*. El vector *up* es irrelevante para la primera parte de este problema, por lo que nos quedamos únicamente con los puntos *eye* y *center*.

Teniendo estos dos puntos, ayuda entenderlos a partir de ahora como un único vector que va desde *center* a *eye*.

Para mover la cámara hacia arriba o hacia atrás, debemos encontrar el vector unitario del vector, multiplicarlo por la distancia que queremos que se mueva la cámara y luego sumárselo (para avanzar) o restárselo (para retroceder) tanto al centro como al ojo.

Para mover hacia los lados la lógica es exactamente la misma, pero en lugar de usar el vector unitario del propio vector ojo → centro, debemos usar el vector perpendicular al plano formado

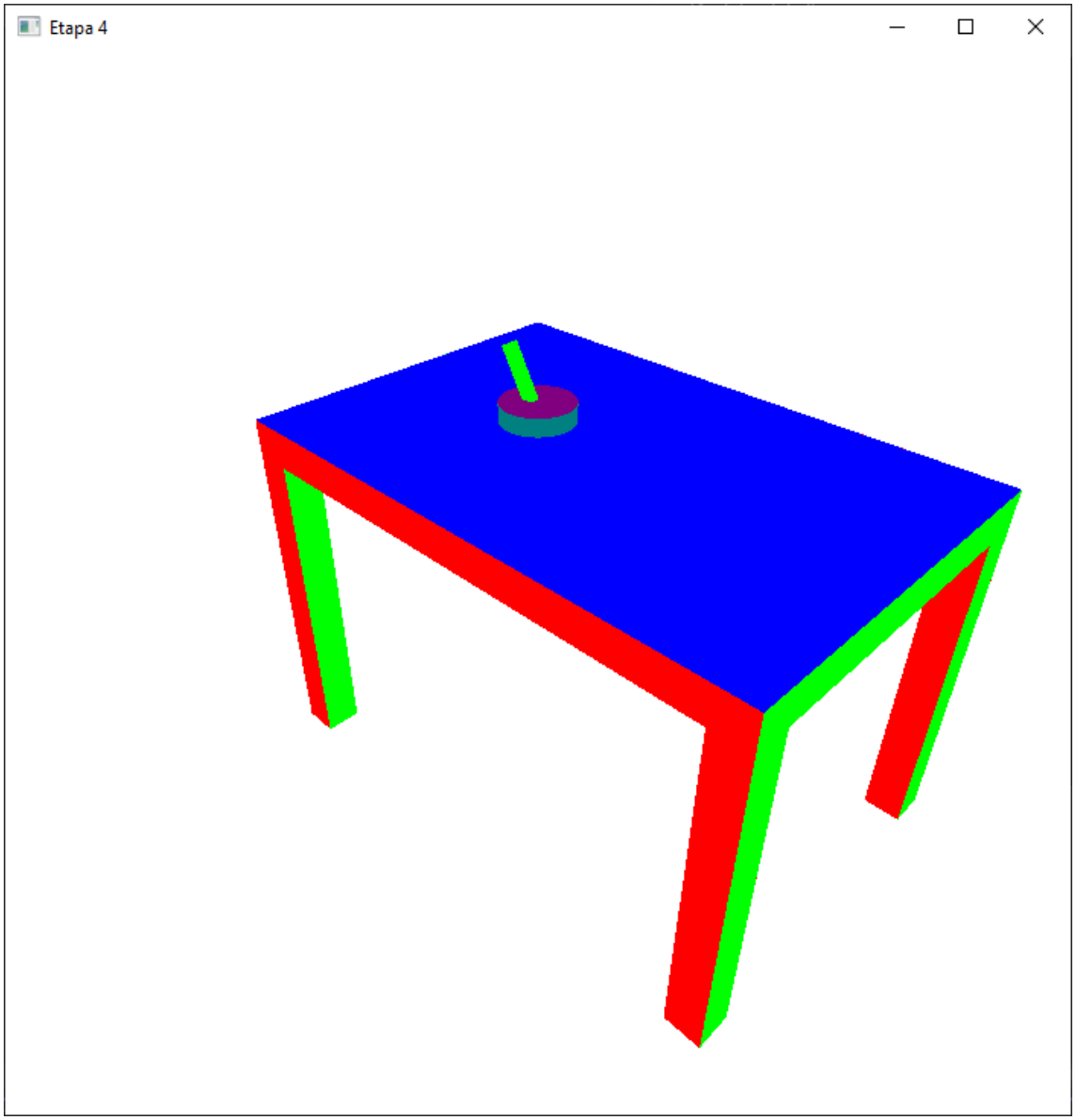
por dicho vector y el vector up. Como nuestro up siempre vale (0, 1, 0), el vector derecha equivale a (-z, 0, x), donde (x, y, z) es el vector ojo → centro.

### Otras perspectivas

Se han incluido también otras posibles perspectivas, a las que se puede cambiar utilizando las teclas F3 a F7.

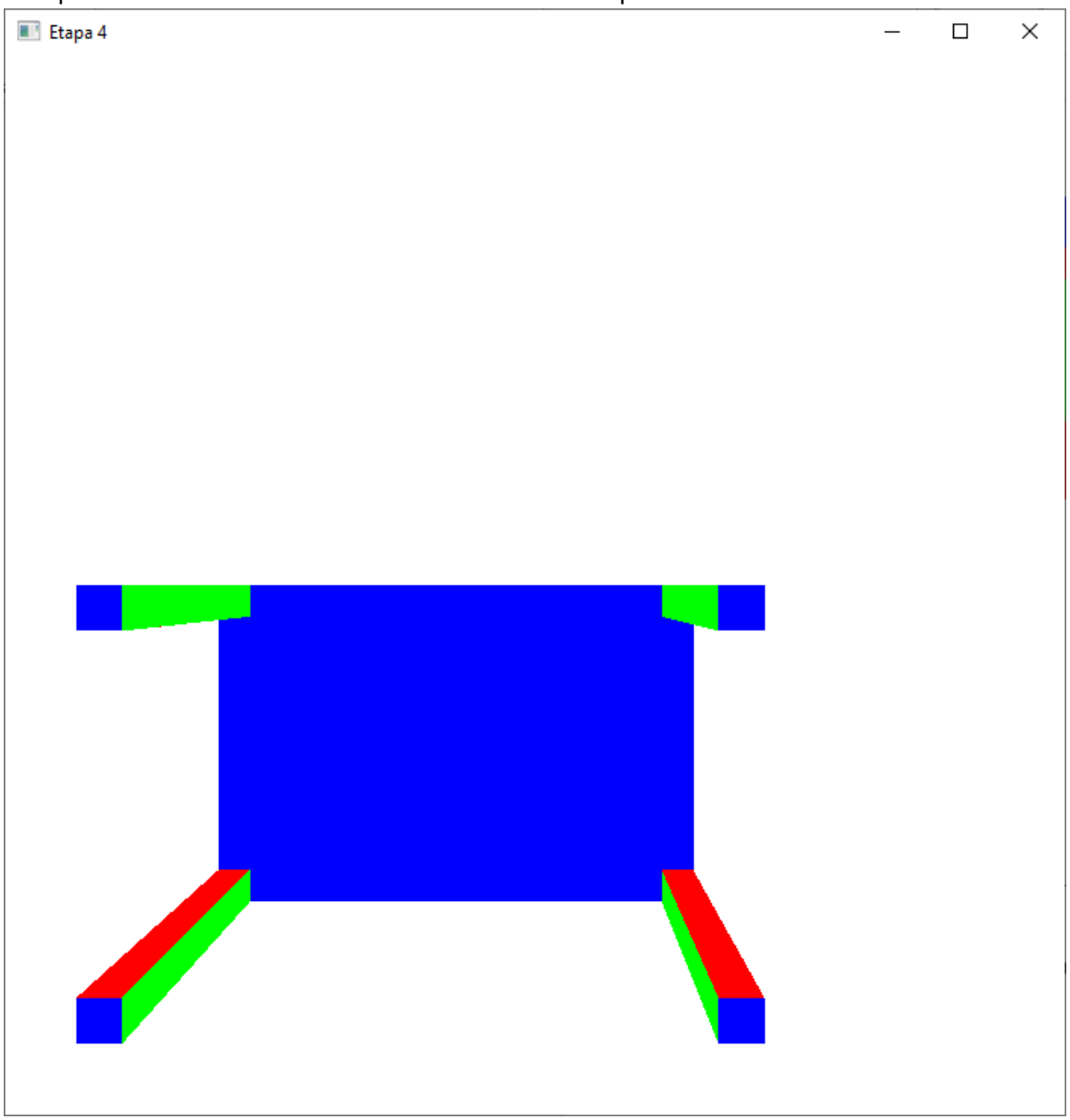
### Perspectiva por defecto

Por defecto, o pulsando F6, se mueve la cámara a una perspectiva desde un ángulo alto.



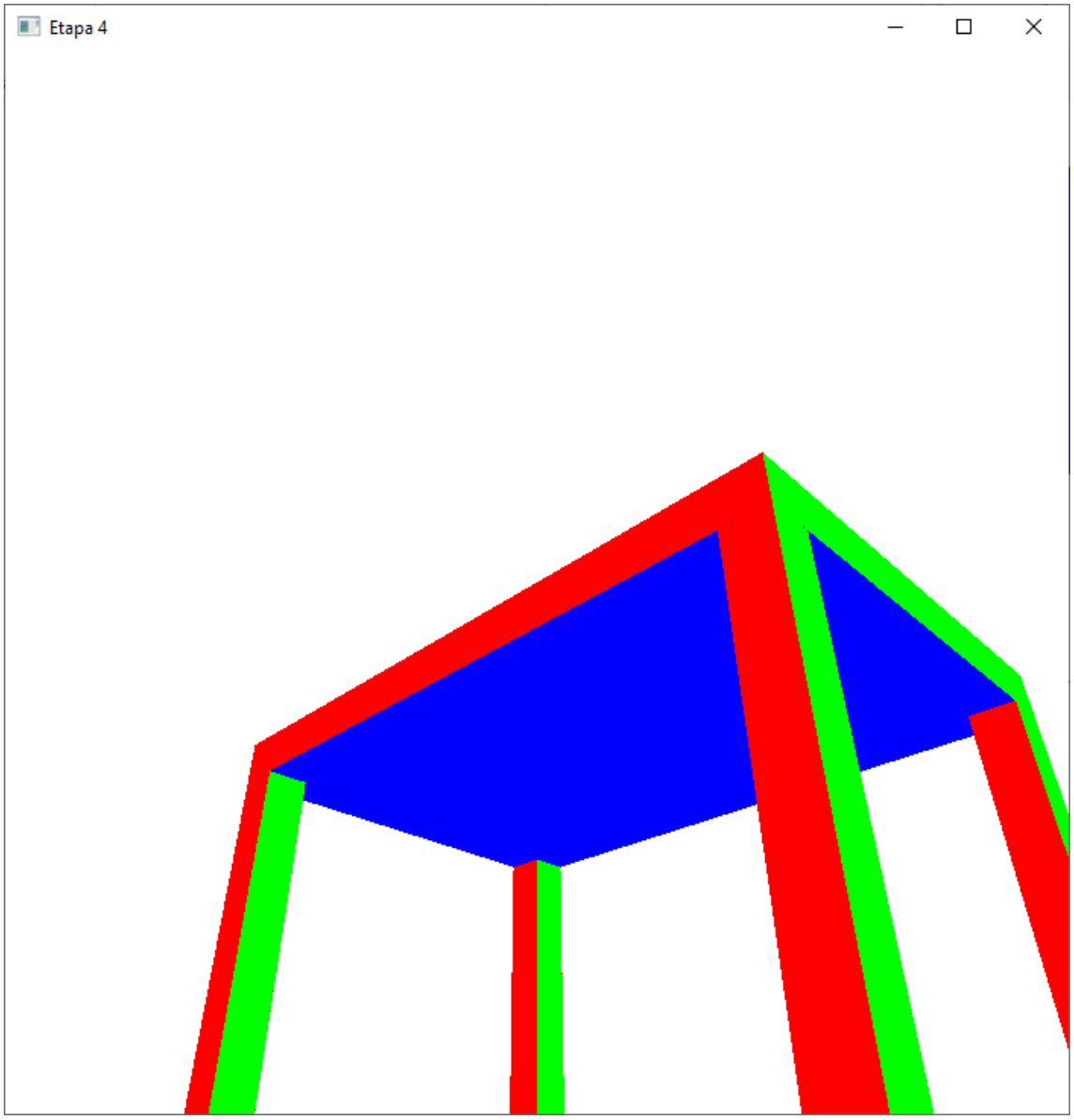
## Perspectiva nadir

Con F3 establecemos que la cámara mire completamente desde abajo. Desde esta perspectiva ni el paneo de cámara ni el movimiento a los lados son posibles.



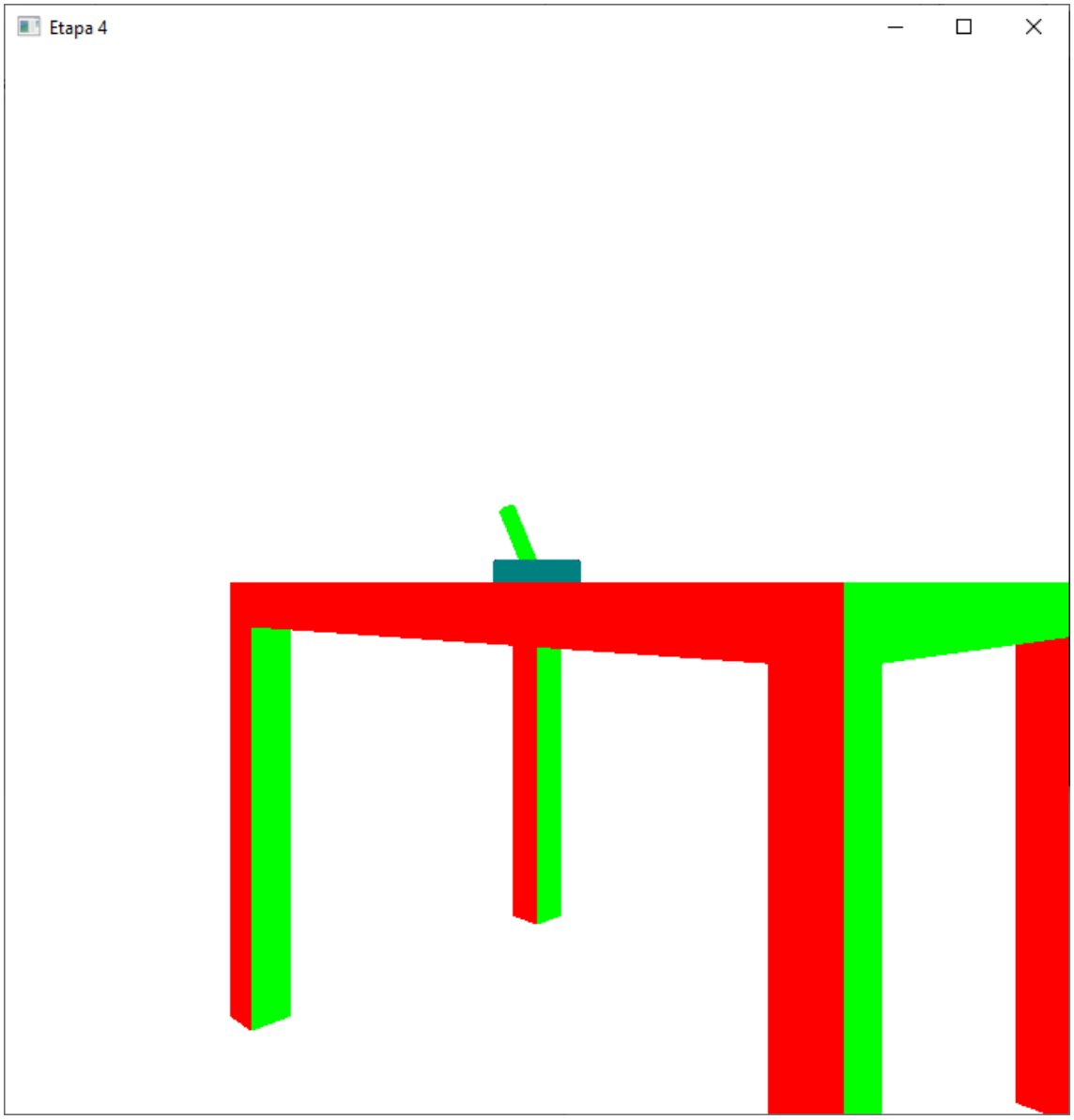
## Perspectiva desde ángulo bajo

Con F4 obtenemos este punto de vista desde un ángulo inferior.



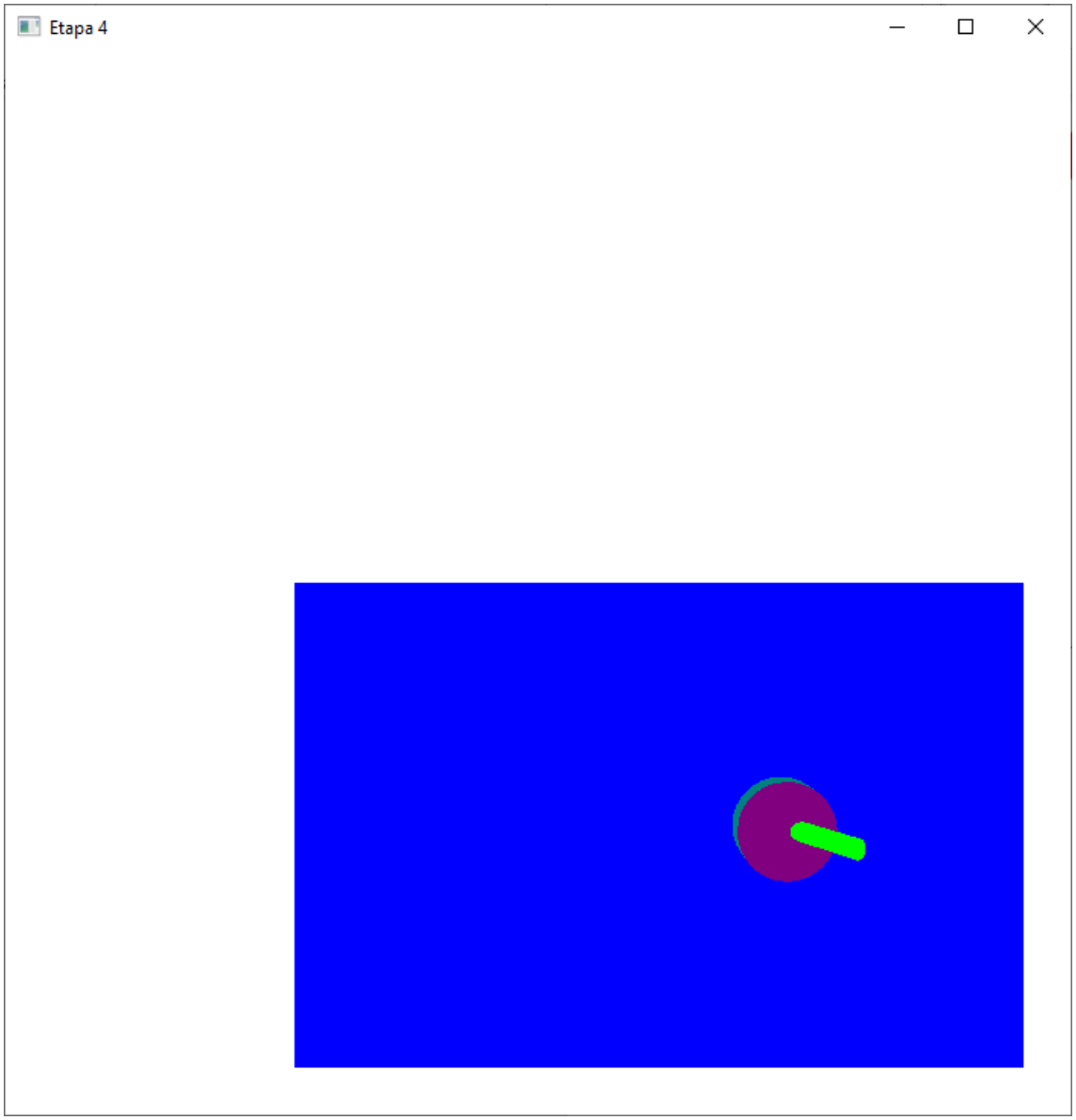
## Perspectiva horizontal

Con F5 obtenemos una perspectiva que mira desde la propia altura de la mesa.



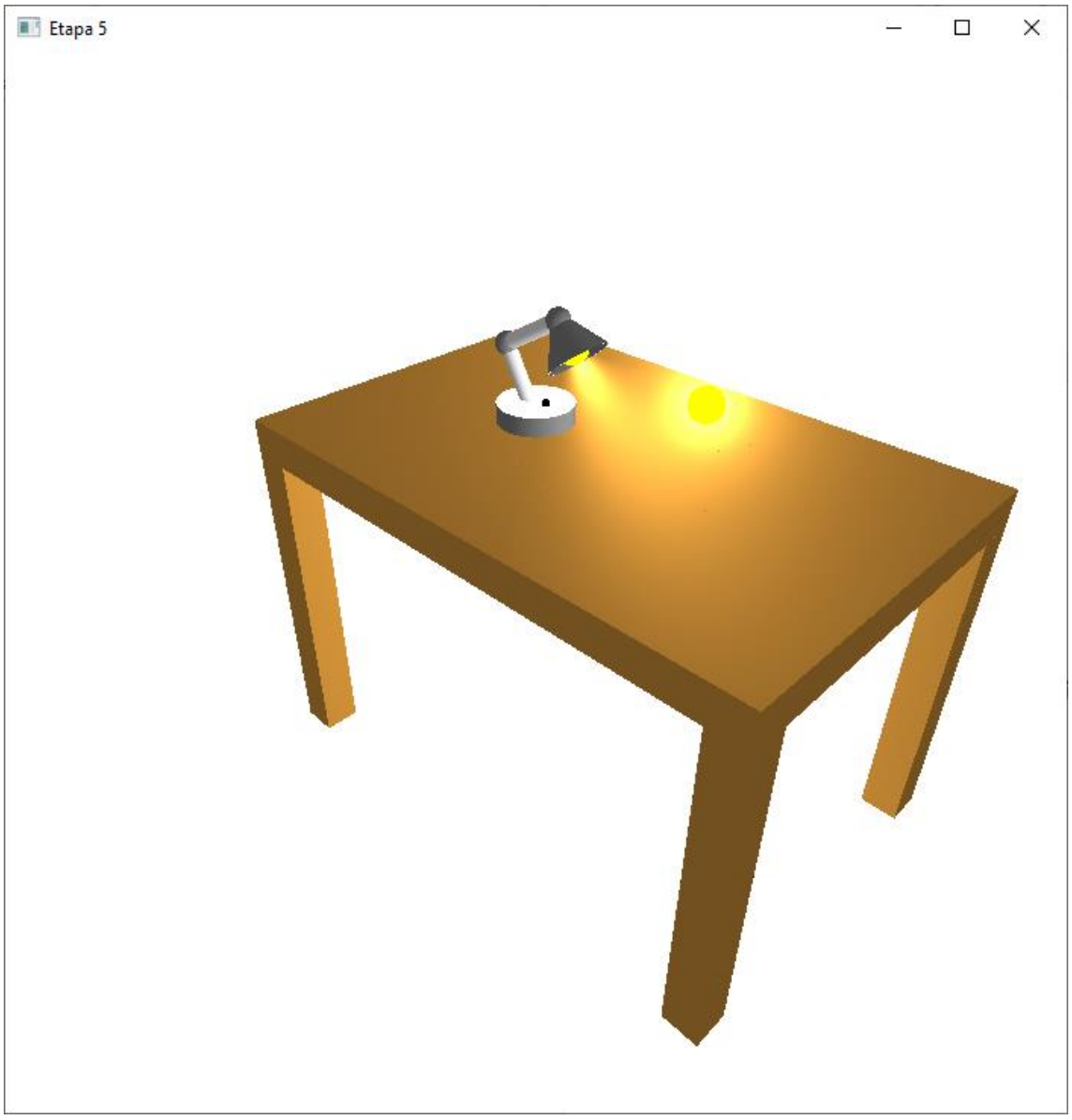
### Perspectiva cenital

Y, por último, con F7 obtenemos la perspectiva cenital. Igual que con la nadir, únicamente se puede avanzar y retroceder. En las etapas 4 y 5, la cámara queda muy cerca de la mesa, haciendo que no se pueda apreciar la escena a menos que se retroceda. Esto fue arreglado en la etapa 6, proporcionando una perspectiva mucho mejor.



## Etapa 5

Esta etapa consiste en implementar la iluminación de la escena y sus controles, así como los materiales de los objetos. También fue aquí cuando continuamos con la propia escena, completando la lámpara y dando color, alcanzando así ya un estado semejante al final.




La escena con todas las luces activadas.

## Luz 0 – Ambiental

Esta luz se puede encender y apagar utilizando la tecla L. Le hemos establecido un valor de 0.5 en los tres componentes de color, de modo que no ilumine demasiado toda la escena por completo, pero siga ofreciendo un poco de iluminación.


La escena con únicamente la luz ambiental.

## Luz 1 – Bombilla de la lámpara

La luz 1 representa la bombilla, y se puede activar y desactivar pulsando la tecla 1 del teclado principal o numérico. Para imitar el comportamiento de la luz que sale de una lámpara con cono, hemos utilizado `GL_SPOT_CUTOFF` para establecer un ángulo de luz de 45 grados, creando una luz focalizada. Su ubicación y dirección han sido implementadas al mismo nivel que la propia esfera que representa la bombilla, mientras que su difusión y el ángulo, al principio del *display*.

Esta luz es estática por defecto, pero se puede mover cambiando el booleano `bIsPaused` desde el código fuente a true. La animación es muy rudimentaria, pues es una animación temporal antes de implementar la final en la etapa 6, pero sirve para ver el movimiento del foco.


La escena con únicamente la lámpara encendida.

## Luz 2 – Luz libre

La luz 2 es una bola de luz que se puede mover libremente por la escena en las tres direcciones, utilizando las teclas QWEASD. Las teclas Q y E permiten el movimiento en el eje Y, las teclas A y D, en el eje X; y las teclas W y S, en el eje Z. Se puede encender y apagar usando la tecla 2 del teclado normal o del numérico.


La escena con solo la luz libre encendida.

## Funciones de Practicas.h implementadas en este nivel

### setMaterial

Función que simplifica el establecer materiales, permitiendo pasar cada valor por separado para que la propia función los establezca.

### resetMaterial

Función que aprovecha la función anterior para restablecer el valor por defecto de cada uno de los parámetros de material.[2]

### drawParallMaterial

Esta función con polimorfismo crea un paralelepípedo y establece las normales de cada una de las caras.

La función con parámetro `face_polygons` hace que cada cara esté formada por $n^2$ polígonos, donde n es el valor de `face_polygons`. Esto hace que la parte de una cara más alejada de la luz esté menos iluminada que una más cercana. Sabemos que no es la mejor implementación, y mejorarlo era una de las cosas que queríamos retomar si teníamos tiempo, pero no fue posible.

[2] Los valores por defectos fueron sacados del siguiente enlace
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glMaterial.xml

# Etapa 6

Esta es la última etapa que hemos implementado, y viene con varias mejoras.

La primera de las mejoras es el menú de opciones, accesible haciendo clic derecho en cualquier parte de la ventana. Desde este menú podemos encontrar diferentes opciones de la escena y la cámara.

## Escena final

La escena final es bastante sencilla, teniendo solo tres formas, pero de relativa complejidad.

La escena está formada por la mesa, la lámpara y la tetera. Las primeras dos han sido programadas por nosotros personalmente, y la lámpara está articulada como se puede ver en la animación en la que entraremos en más detalle más adelante.

## Opciones optativas

De las opciones optativas hemos implementado dos: la niebla —que se puede activar y desactivar desde el menú— y el uso de diferentes librerías, en este caso freeglut que se ha usado para la totalidad de la práctica.

## Movimiento fluido de la cámara

Este fue un añadido que añadimos por deseo personal: una cámara que se mueva de manera fluida. La manera de implementarlo fue haciendo que en lugar de tener nueve variables para controlar la cámara, tuviéramos dieciocho: las nueve "reales" y otras nueve que representan el destino al que las reales tienen que ir. Cuando cambiamos la perspectiva, modificamos el valor del destino, y luego en la función idle llamamos a la función `smoothTransition()` incluida en Practicas.h para que haga la transición fluida entre las dos.

Esta opción se puede desactivar y activar utilizando la opción "Toggle smooth camera" del menú.

## Animación

Esta animación que, en principio puede parecer sencilla, ha resultado muy complicada de hacer por el simple hecho de que fue necesario hacerla mediante funciones matemáticas.

La animación está formada por seis partes, tres de las cuales son inversas de las otras, y que forman un bucle de modo que el final de la sexta parte deja a la lámpara en la posición inicial de la primera parte.

Debido a limitaciones de OpenGL es imposible controlar la velocidad de la animación más allá de hacerlo con *frames*, y tampoco es posible establecer un *framerate* para poder controlar la velocidad de la animación. Por tanto, si la animación va demasiado rápido o demasiado lento, lo único que podemos recomendar es que se cambie la resolución de la mesa cambiando el valor en el código fuente de la variable `tableResolution`, que es la manera más sencilla de controlar el rendimiento de la escena. Para tener una idea, la animación debería durar entre 3 y 4 segundos.

La primera y cuarta parte, que duran 30 *frames*, es la lámpara dando saltos y avanzando. Para los saltos, hemos usado una función sinusoidal que controlara la posición y en función del tiempo de la animación, mientras que para el avance ha bastado simplemente con un incremento en la posición x.

Además, hemos unificado el ángulo de la articulación de la lámpara con la altura de ésta con el fin de que no quedara tan rígida.

La segunda y quinta parte, que duran 5 *frames*, son la lámpara girando su cono hacia un costado para prepararse para dar la vuelta. En este caso ha sido simplemente cuestión de incrementar el ángulo del cono.

La tercera y sexta parte, de 10 *frames*, son la lámpara dándose la vuelta. La lámpara pasa la mitad de esta parte saltando hacia arriba y la otra mitad bajando. También se modifica la dirección de la lámpara para que haga un giro de 180º y al mismo tiempo se devuelve el ángulo del cono a su posición inicial.

Hay dos opciones diferentes para la animación en el menú: pausar/continuar, o devolver la lámpara a su posición inicial.

### Detalles menores

Para terminar, añadimos dos detalles que no afectan demasiado a la escena en total pero añaden cierto realismo, y que están ambos relacionados a la lámpara en sí.

El primero, el hacer que la emisión y color de la bombilla dependiera de si ésta estuviera encendida o no. Emisión al máximo y bombilla amarilla cuando está encendida, emisión a cero y bombilla gris cuando está apagada.

El segundo es el interruptor de la lámpara, que está bajado cuando la lámpara está encendida y está alto cuando ésta está apagada.

## Conclusiones personales

Esta práctica, según hemos podido ver en nuestros compañeros de clase, es un poco divisiva.

Ha supuesto un reto interesante el ver cómo funcionan la mayoría de los gráficos a un nivel más inferior al que trabajan la mayoría de los diseñadores gráficos actualmente.

Al mismo tiempo ha podido resultar un poco “decepcionante” en el sentido de que no es un conjunto de habilidades que usará la mayoría de los alumnos a menos que se dediquen, justamente, a programar los programas de gráficos de alto nivel como lo son Blender o similares.

Obviamente es comprensible que esta es una asignatura optativa y por tanto está pensada con un cierto nivel de especificidad, pero si bien es cierto que es una asignatura optativa en términos de carrera, sigue siendo una asignatura del itinerario de computación e inteligencia artificial.

Es también especialmente decepcionante que ésta sea la única asignatura de gráficos de la carrera y por eso duele tanto que se quede solamente en los fundamentos, pero eso ya lo hemos tratado varias veces en clase y entendemos que es un asunto más a nivel de carrera y asignaturas ofertadas que al nivel de la propia asignatura, así que no vamos a entrar más en detalle.

En definitiva, ha sido una asignatura y una práctica muy disfrutable desde nuestra perspectiva, pero es triste que no se pueda llegar a un mayor nivel.