Entrega III Apartados del 38 al 53
Fecha de entrega: 3 de abril de 2025

Instrucciones de la práctica

Para llevar a cabo este trabajo se tendrá en cuenta lo siguiente:

- Esta práctica se compone de movimientos de cámara y varias escenas opcionales.
- Es necesario implementar una serie de pasos, en forma de apartados.
- El código que se desarrolle deberá ser reutilizable por las escenas posteriores.
- Se tendrá especial atención a la calidad del código: inexistencia de fugas de memoria, variables con tipo correctamente definido, código bien organizado, comentado y legible, siguendo los principios de la programación orientada a objetos.

Rúbrica de evaluación

Criterio	**Pesos**
Movimiento LR-FB-UD 1	1
Cambio de Proyección	1
Movimiento pitch/roll/yaw	1
Movimiento orbital	1
Cambio de vista 2D - 3D	1
Vista cenital	1
Dos vistas	1
Movimientos de ratón	1
Calidad del código	1
Ausencia de fugas	1
Total	**10.0**
Defensa (individual)	-1.5

Opcionales	**Pesos**
Puerto de vista con dos escenas	+0.5
Ratón y puertos de vista	+0.5

Apartado 38

Añade a la clase `Camera` los atributos `glm::vec3` `mRight`, `mUpward` y `mFront`, más el método protegido `void setAxes()` que les da valor usando la función `row()`.

Apartado 39

Modifica aquellos métodos de la clase `Camera` que deban invocar el método `setAxes()` para mantener el valor de estos atributos coherentemente con cualquier cambio en la matriz de vista.

Apartado 40

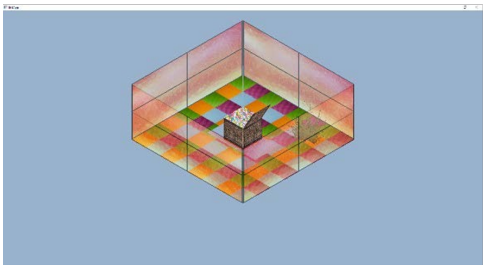
Añade a la clase `Camera`, métodos para desplazar la cámara en cada uno de sus ejes, sin cambiar la dirección de vista:

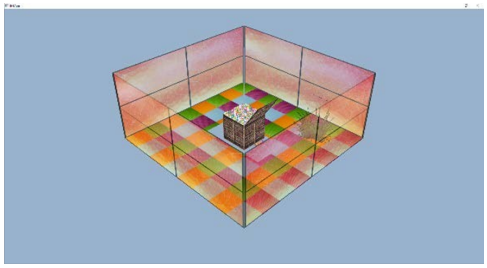
```
void moveLR(GLfloat cs); // A izquierda/A derecha
void moveFB(GLfloat cs); // Adelante/Atrás
void moveUD(GLfloat cs); // Arriba/Abajo
```

Asigna las pulsaciones `a` y `d` a `moveLR`, `w` y `s` a `moveUD`, y `W` y `S` a `moveFB` (son las mismas teclas, pero con las mayúsculas activas).

Apartado 41

Añade a la clase `Camera` un método público `void changePrj()` para cambiar de proyección ortogonal (izquierda) a perspectiva (derecha, en las capturas de más abajo)), y viceversa. El cambio de proyección está mediado por el atributo booleano `bOrto` de la clase `Camera`.





Apartado 42

Modifica aquellos métodos de la clase `Camera` afectados por el cambio de proyección (por ejemplo, `setPM()` para que el zoom siga funcionando correctamente).

Apartado 43

Define el evento de la tecla [p] para cambiar entre proyección ortogonal y perspectiva.

Apartado 44

Comprueba que funcionan correctamente los métodos del apartado 40, tanto con proyección ortogonal como con proyección perspectiva.

Apartado 45

Añade a la clase `Camera`, métodos para rotar la cámara en cada uno de sus ejes u, v y n:

```
void pitchReal(GLfloat cs);
void yawReal(GLfloat cs);
void rollReal(GLfloat cs);
```

Asocia las teclas [←] y [→] a `yawReal` si no está pulsada la tecla [Ctrl] y a `rollReal` si lo está; y las teclas [↑] y [↓] a `pitchReal`. Hay una demo en el Campus Virtual que muestra cuál debe ser el comportamiento esperado de cada una de ellas.

Apartado 46

Incorpora a la clase `Camera` el método `orbit(incAng, incY)` para desplazar la cámara a lo largo de una circunferencia situada sobre el plano **XZ**, a una determinada altura sobre el eje Y, alrededor de `mLook`, tal como aparece definido este método en las transparencias.

Apartado 47

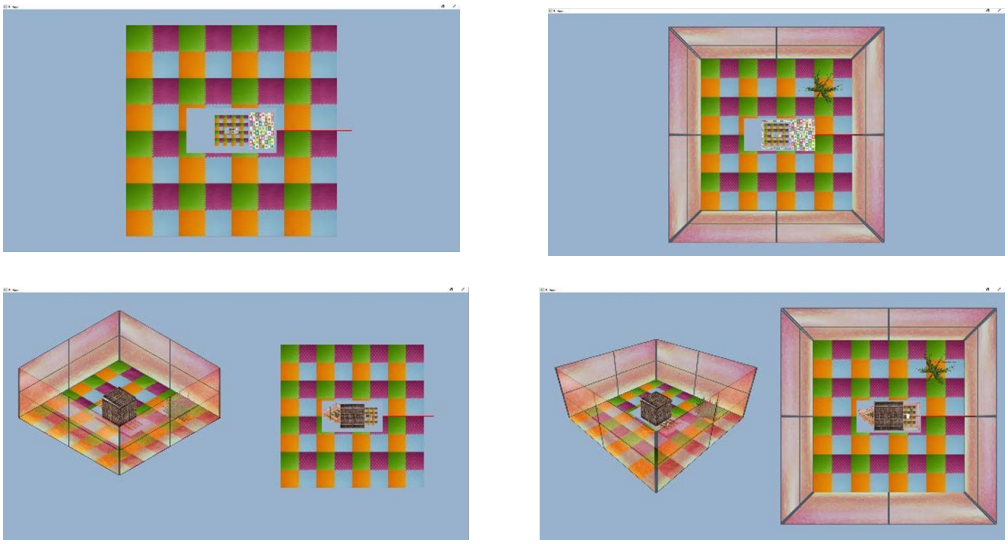
Modifica los métodos `set2D()` y `set3D()` de manera que se inicialicen de forma coherente los atributos de la cámara `mEye`, `mLook`, `mUp`, `mAng` y `mRadio`.

Apartado 48

Añade a la clase `Camera` un método `setCenital()` que muestra la escena desde una posición cenital. Abajo tienes dos capturas que muestran una escena cenitalmente en proyección ortogonal (izquierda) y perspectiva (derecha).

Apartado 49

Define el evento de la tecla [k] para renderizar dos vistas de forma simultánea, tal como se muestra en las capturas de abajo. En cada una de ellas, a la izquierda se ve la escena con la cámara en su posición normal 3D, y a la derecha con la cámara en posición cenital.



La captura de la izquierda está hecha con proyección ortogonal y la de la derecha con proyección perspectiva. Añade a la aplicación un atributo booleano `m2Vistas` para mediar, en la tecla [k], entre la renderización de una vista o de dos.

Apartado 50

Añade a `IG1App` dos nuevos atributos: `dvec2 mMouseCoord`, para guardar las coordenadas del ratón, e `int mMouseButt`, para guardar el botón pulsado.

Apartado 51

Programa los siguientes callbacks de `IG1App`, definidos tal como se han explicado en clase:

- `void mouse(int button, int state, int x, int y)`: captura en `mMouseCoord` las coordenadas del ratón (x, y), y en `mMouseButt`, el botón pulsado.
- `void motion(int x, int y)`: captura las coordenadas del ratón, obtiene el desplazamiento con respecto a las anteriores coordenadas y, si el botón pulsado es el derecho, mueve la cámara en sus ejes `mRight` (horizontal) y `mUpward` (vertical) el correspondiente desplazamiento, mientras que si es el botón izquierdo, rota la cámara alrededor de la escena.
- `void mouseWheel(int n, int d, int x, int y)`: si no está pulsada ninguna tecla modificadora, desplaza la cámara en su dirección de vista (eje `mFront`), hacia delante/atrás según sea d positivo/negativo; si se pulsa la tecla [Ctrl], escala la escena, de nuevo según el valor de `d`.

Apartado 52

(Opcional) Renderiza dos escenas diferentes dividiendo la ventana en dos puertos de vista. En el de la izquierda se mostrará la Escena 4, es decir, la de la cristalera, y en el de la derecha, la Escena 2, es decir, la bidimensional de la primera entrega.

Apartado 53

(Opcional) Modifica la programación de los eventos de ratón de forma que este pueda actuar independientemente en cada puerto de vista, dependiendo de en cual se encuentre el cursor. Modifica el evento de la tecla [u] de forma que se actualice la escena del puerto de vista que contiene el cursor del ratón. Modifica el evento de la tecla [p] de forma que se cambie el tipo de proyección de la escena del puerto de vista que contiene el cursor del ratón. La foto puede seguir mostrando la ventana entera.