20. La clase Camera

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

  void moveLR(GLdouble cs); // Left / Right

  void moveFB(GLdouble cs); // Forward / Backward

  void moveUD(GLdouble cs); // Up / Down

  Añade los atributos mRight, mUpward y mFront para cada uno de los ejes; y el método protegido:

  void setAxes(): Da valor a los tres ejes. Tendrás que incluir <gtc/matrix_access.hpp>

  Todos los métodos que modifiquen algún elemento de la cámara tienen que actualizar todos los atributos necesarios para que sus valores sean coherentes.
  Modifica el método

  void setVM() { mViewMat = lookAt(mEye, mLook, mUp); setAxes(); };

  Quita (comenta) los métodos pitch, yaw y roll.

- Añade a la aplicación (IG1App) dos nuevos atributos: dvec2 mMouseCoord para guardar las coordenadas del ratón y, int mMouseButt para guardar el botón pulsado.
  Añade también los callbacks para los eventos del ratón:

  - glutMouseFunc(s_mouse);

    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.

  - glutMotionFunc(s_motion);

    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, desplaza la cámara en sus ejes mRight (horizontal) y mUpward (vertical) el correspondiente desplazamiento.

  - glutMouseWheelFunc(s_mouseWheel);

    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).

- Añade a la clase Camera el método orbit para desplazar la cámara (mEye) siguiendo una circunferencia (en los ejes X y Z) a una determinada altura (eje Y) alrededor de mLook.

  void orbit(GLdouble incAng, GLdouble incY): modifica la posición y la dirección de vista de la cámara

  Añade los atributos para el ángulo y el radio (p. ej. 1000) de la circunferencia que recorrerá la cámara con este método. Tendrás que dar un valor adecuado al ángulo en los métodos set2D, set3D, …

  Modifica los métodos de la aplicación (IG1App):

  motion(int x, int y): Además, si el botón pulsado es el izquierdo, actualiza la posición de la cámara en la circunferencia en función del desplazamiento del ratón.

  mouseWheel(int n, int d, int x, int y): Además, si está pulsada la tecla CTRL escala la vista.

- Añade a la clase Camera un método changePrj() para cambiar de proyección ortogonal a perspectiva. Modifica los métodos de la clase Camera afectados por el cambio de proyección.

  Define la tecla p para cambiar entre proyección ortogonal y perspectiva

- NOTA sobre la clase Camera. Siempre que se añade un atributo a una clase hay que repasar todos los métodos para comprobar si hay que hacer algún ajuste para que los valores de los atributos sean coherentes con los valores que representan.

  Hemos añadido a la clase Camera los atributos para sus ejes (mRight, mUpward y mFront). Por tanto, hay que repasar los métodos de la clase y hacer los ajustes necesarios para que los valores de los atributos sean coherentes.

  Por ejemplo, siempre que cambia la posición u orientación de la cámara cambia la matriz de vista (mViewMat), por eso los métodos set2D(), set3D(), moveLR, moveUD y moveFB llaman a setVM() (actualiza la matriz de vista). Pero siempre que cambia la matriz de vista cambian los ejes, por eso modificamos setVM() para que además llame a setAxes() (actualiza los ejes).

  También hemos añadido a la clase Camera atributos para los valores de la circunferencia del método orbit() (mAng y mRadio). Por tanto, hay que repasar los métodos de la clase y hacer los ajustes necesarios para que los valores de los atributos sean coherentes.

  Los métodos moveLR, moveUD y moveFB mueven los atributos mLook y mEye exactamente igual. La nueva posición de la cámara es coherente con la circunferencia de orbit() porque el centro de la circunferencia es mLook y por tanto, estos métodos no modifican ni el radio ni el ángulo, lo que hacen es trasladar la circunferencia con la cámara

  Los métodos set2D() y set3D() no mueven igual mLook y mEye, lo que hacen es asignarles unos valores absolutos (500, 10, …) que no tienen en cuenta la circunferencia de orbit, y los atributos no quedan coherentes. Por ejemplo,

el método set3D() modifica la posición y orientación de la cámara:

```
mLook = dvec3(0, 10, 0);
mEye = dvec3(500, 500, 500);
mUp = dvec3(0, 1, 0);
```

¿El nuevo valor de mEye está en la circunferencia de centro mLook y radio mRadio?

No → tenemos que asignar a mEye un valor en función de mRadio y mAng

¿Cuál es ángulo (mAng) de la circunferencia que corresponde con la nueva posición de la cámara (mEye)? → tenemos que asignar a mAng el valor adecuado:

Si la cámara está en el eje X positivo, el ángulo es 0.

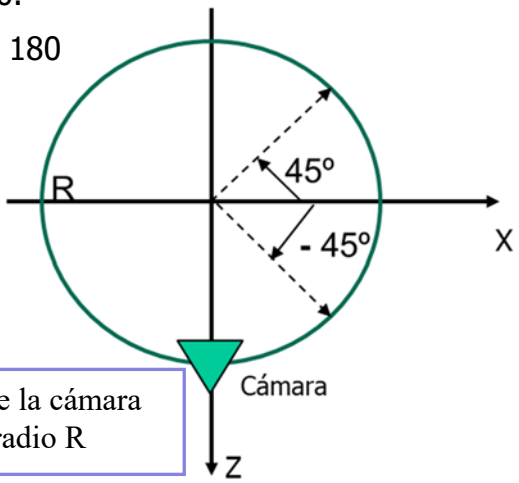
Si la cámara está en el eje X negativo, el ángulo es 180

Si la cámara está en el eje Z positivo, …

La cámara 2D está en el eje …

La cámara 3D está en …

Consultar la transparencia 2



Vista cenital de la órbita de la cámara alrededor del eje Y y radio R

21. Añade a la aplicación la opción (tecla k) de visualizar dos vistas simultáneamente. Añade a la aplicación un atributo bool m2Vistas para la opción, y utilízalo en el método display de la aplicación. Define el método display2Vistas() para dividir la ventana en dos puertos de vista y visualizar en el lado izquierdo la vista actual, y en el lado derecho la vista cenital (utiliza una cámara auxiliar, la escala debe ser la misma). display2Vistas() debe dejar la cámara (atributo mCamera, incluido el puerto de vista) sin modificar. Puedes añadir a la clase Camera un método setCenital.

```
void IG1App::display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  if (m2Vistas) display2Vistas();
  else { … }
  glutSwapBuffers();
}
```