UNIVERSIDAD DE DARMSTADT
B. FRÖMMER, E. HERGENRÖTHER, B. MEYER

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

11 DE MAYO
de 2023

VISUAL INFORMÁTIC SOSE 2023
A
TAREA 3

3.1 Cámara en perspectiva

En la primera mitad de esta hoja de tareas queremos integrar una cámara en nuestro pequeño programa de ejemplo para obtener una proyección de perspectiva correcta en nuestras imágenes.

3.1.1 La ViewMatrix

Como se sabe por la clase, primero hay que determinar la `ViewMatrix` con la que se transforma la escena completa para que tu cámara (virtual) se sitúe en el origen y mire a lo largo del eje Z negativo. La librería matemática `glm` ya proporciona el método correspondiente: `glm::lookAt()`. Sólo necesitas definir la posición de la cámara, un punto de vista y un vector ascendente.

Atención: Si renderiza la imagen sólo con la matriz de vista como prueba, tenga cuidado de no dejar el *Espacio Normalizado de Dispositivo* entre -1 y 1 al posicionar la cámara.

3.1.2 La matriz de proyección

En el segundo paso hay que determinar la proyección en perspectiva de la cámara. Esto depende de los parámetros internos de la cámara. El método `glm::perspective()` se encarga de calcular la matriz correspondiente.

Para que la cámara se tenga en cuenta en el proceso de renderizado, aún tienes que cargar las dos nuevas matrices en el shader y utilizarlas correctamente allí.

Atención: Habíamos activado la prueba de profundidad en la última hoja de ruta, pero aquí todavía teníamos que hacer la prueba "al revés" porque faltaba la inversión del eje Z (véase la lección). Esto ya no es necesario, así que cambia tu prueba de profundidad a las siguientes líneas:

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS); // Valor por defecto, también puede omitirse
glClearDepth(1.0); // Valor por defecto, también puede omitirse
```

Juega con los distintos parámetros de la cámara: ¿qué hacen los cambios en los planos de recorte cercano y lejano, el ángulo de apertura (vertical) o la relación de aspecto?

3.2 Aplicación gratuita

Pon a prueba los conocimientos adquiridos Construye tu propia aplicación creativa.
Aquí tienes algunas sugerencias que puedes (pero no tienes por qué) integrar:

- Utiliza otra geometría: En lugar del cubo dado, puedes invitar a cualquier otro objeto. Estos pueden ser definidos por usted mismo, pero también puede encontrar muchas figuras geométricas predefinidas en Internet. Tampoco hay nada que te impida mostrar *varios* objetos diferentes.
- Diseñe animaciones creativas: Implementa patrones de movimiento para tu robot u otras figuras, transforma tu robot en un coche/avión por ejemplo.
- Una combinación de cambios de color y posición es excelente para crear animaciones más complejas.
- En el método update() del archivo Scene.cpp, puede utilizar el método
  `if (m_window->getInput().getKeyState(Key::W) == KeyState::Pressed)`
  Accede a las entradas del teclado. ¡Diseña tu propio control para tus objetos!
- Del mismo modo, puede reaccionar a los movimientos del ratón en `onMouseMove()` (para controlar la cámara con él, por ejemplo).
  Consejo: El objeto `MousePosition` guarda la nueva y la antigua posición del ratón:
  `auto nuevaPosición = glm::vec2(posicióndelratón.X, posicióndelratón.Y);`
  `auto oldPosition = glm::vec2(mouseposition.oldX,`
  `mouseposition.oldY); auto direction = (newPosition - oldPosition);`
  puede ayudar a determinar la dirección de movimiento del ratón.