Universidade Lusófona de Humanidades e Tecnologias

ULHT260-2267
Licenciatura em Engenharia Informática
Computação Gráfica
Nuno Cruz Garcia – p6040@ulusofona.pt

Aula Prática 06

Temas

Câmara
- Controlar a câmara com o teclado e navegar na cena
- A função *glm::lookat*

Breve resumo da aula passada:

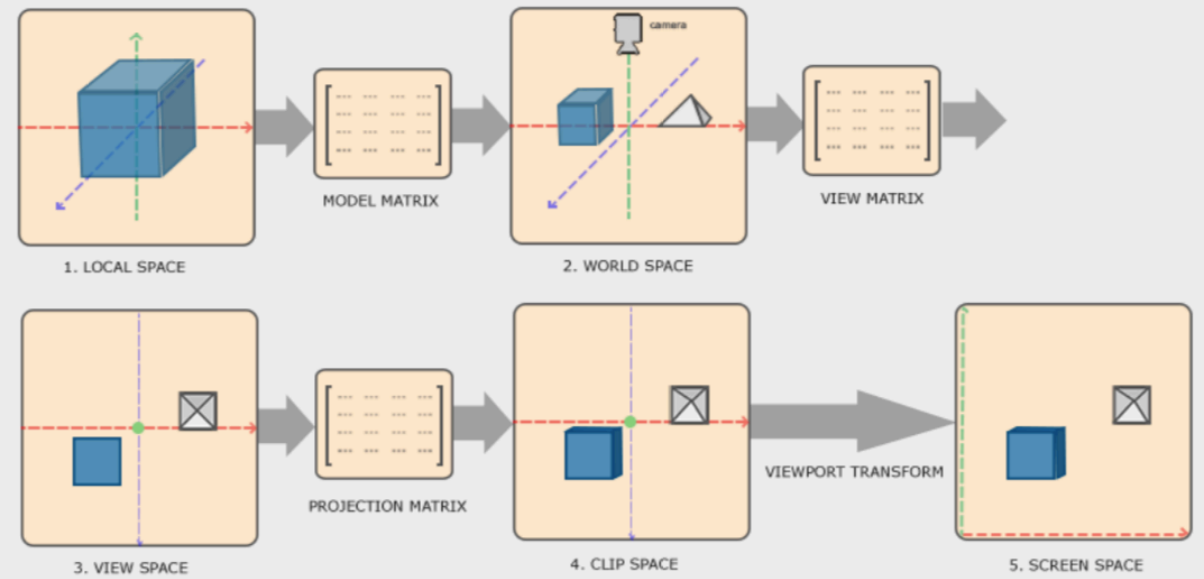


Figura 1 - https://learnopengl.com/Getting-started/Coordinate-Systems

A Matriz de Visualização transforma as coordenadas do Mundo em coordenadas de Visualização - os vértices dos objetivos passam a ser vistos como se a câmara fosse a origem da cena.

Para definir a câmara precisamos de:
- posição da câmara em coordenadas do Mundo;
- a direção em que está a apontar;
- definir o sistema de eixos da câmara.

A função *LookAt*

Esta secção trata da função *LookAt,* que nos permite posicionar a câmara na cena 3D a apontar para um ponto à escolha.

Posição da câmara

glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f); // o eixo dos zz aponta em direção a nós.

Direção da câmara

Aqui definimos para que ponto a câmara está a apontar. Convencionou-se que a câmara aponta na direção do eixo dos zz.
Relembrar que uma coordenada = vector que aponta da origem para o ponto que definimos.

glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f); // apontando para a origem da cena
glm::vec3 cameraDirection = glm::normalize(cameraPos - cameraTarget);
(C - A)

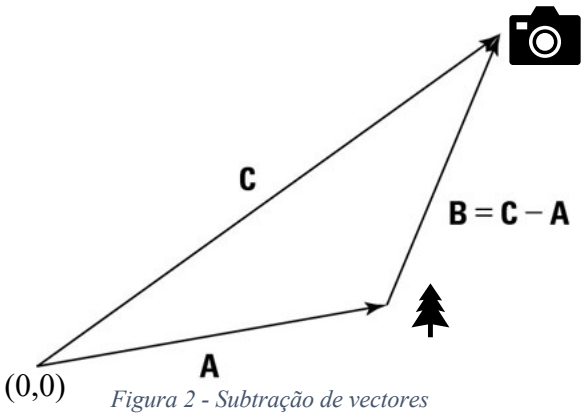


Figura 2 - Subtração de vectores

Nota: a direção da seta aponta para a própria câmara devido a convenções do Opengl entre o *view space* e *world space*.

Sistema de coordenadas da câmara

Vamos definir o sistema de eixos da câmara, que corresponde ao sistema de eixos do View Space na Figura 1, e que servirá para a rodarmos e etc. Definimos o eixo dos xx e dos zz.

- Eixo dos xx - aponta para a direita

glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f);
glm::vec3 cameraRight = glm::normalize(glm::cross(up, cameraDirection));

É definido pelo produto vectorial entre um vector que aponta para cima (em coordenadas mundo) e o vector *cameraDirection* = vector *B* da figura 2.

O produto vectorial de dois vectores resulta num terceiro que lhes é perpendicular. E por isso, se: (1) cameraDirection aponta da direcção do eixo dos zz, e (2) o vector *up* aponta para cima, então, (3) o resultado do produto vectorial é perpendicular a estes dois, e resulta no eixo dos xx.

- Eixo dos yy - aponta para cima

Da mesma forma, o eixo dos yy (agora em coordenadas de Visualização) é definido por
glm::vec3 cameraUp = glm::cross(cameraDirection, cameraRight);

A função e matriz LookAt

Esta é uma função chave de todos os pipelines gráficos. Usando os três vectores definidos acima (cameraDirection = D, cameraRight = R, cameraUp = U), e o vector da posição da câmara (cameraPos = P), podemos definir a matriz que nos permite colocar a câmara em qualquer ponto da cena 3D e que aponta para um ponto à escolha.

A matriz LookAt tem o seguinte aspecto:

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

É uma combinação da rotação da câmara (matriz da esquerda) com a translação para a posição da câmara (matriz da direita).

Naturalmente, a biblioteca *glm* oferece esta funcionalidade.

```
glm::mat4 view;
view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f),      // cameraPos
                   glm::vec3(0.0f, 0.0f, 0.0f),      // cameraDirection
                   glm::vec3(0.0f, 1.0f, 0.0f));     // up vector
```

O ficheiro source1.cpp implementa uma câmara que gira à volta da cena.

Movimentar a Câmara com o teclado

Movimentar a câmara usando o teclado significa alterar a função *LookAt* consoante a tecla pressionada.

Vamos começar por definir as variáveis para a função *LookAt*.
O vector câmaraPos refere-se ao ponto de partida da câmara, e pode ser o mesmo que antes.

```
glm::vec3 cameraPos   = glm::vec3(0.0f, 0.0f,  3.0f);
```

Para já, queremos garantir que a câmara está sempre a olhar em frente, isto é, na direção do eixo dos zz (*cameraFront*). O vector de *cameraDirection* passa assim a ser definido por:

```
glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);
glm::vec3 cameraDirection = cameraPos + cameraFront
```

Por fim, o terceiro parâmetro da função LookAt mantem-se igual.

```
glm::vec3 cameraUp    = glm::vec3(0.0f, 1.0f,  0.0f);

view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
```

Se quisermos andar para a frente, isto é, andar na direcção em que aponta a câmara (cameraDirection), então adicionamos este deslocamento à coordenada de Posição da câmara (cameraPos).

```
cameraPos = cameraPos + deslocamento * cameraFront;
```

Da mesma forma, se quisermos andar para trás, subtraímos o deslocamento.

A ideia é exatamente a mesma para os deslocamentos à esquerda e direita. No entanto, como a câmara aponta para frente (cameraFront), o vector a adicionar é perpendicular a este cameraFront.
Já vimos anteriormente que vectores perpendiculares são o resultado de produtos vectoriais. Assim, um deslocamento para a direita é definido como:

```
cameraPos += glm::cross(cameraFront, cameraUp) * deslocamento;
```

Nota: é preciso normalizar o produto vectorial, por isso a linha é a seguinte

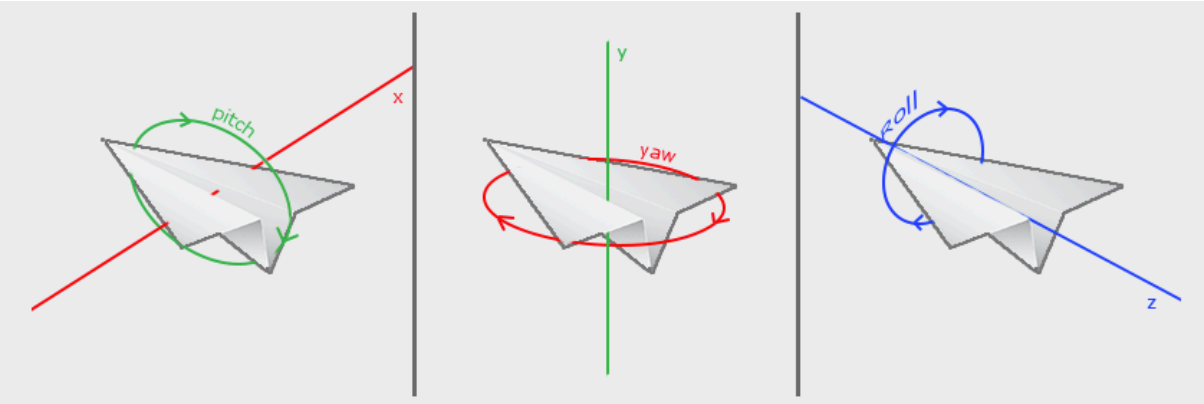
```
cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) * deslocamento;
```

O ficheiro source2.cpp implementa uma câmara que navega pela cena 3D.

Usar o rato para apontar a câmara

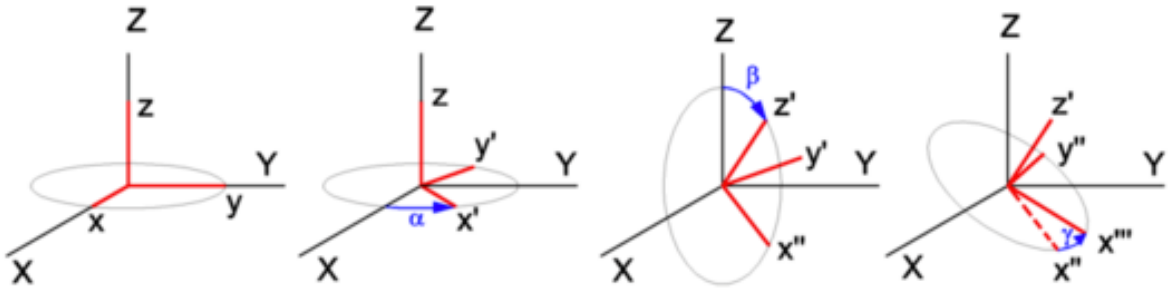
Na última secção, a câmara apenas se movia no espaço, sendo que apontava sempre na mesma direção. Nesta secção vamos usar o rato para rodar a câmara. Para isso iremos alterar o valor de cameraFront.

Existem 3 graus de liberdade para girar a câmara, ou de outra forma, a orientação de um objeto pode ser descrita com base nos chamados ângulos de Euler. As rotações neste sistema de coordenadas chamam-se Pitch, Yaw, e Roll.



Só estamos interessados nos valores de Pitch e Yaw, uma vez que só movemos o rato em duas direções. O Pitch refere-se ao ângulo para olhar mais para cima/baixo, e o Yaw refere-se ao ângulo usado para olhar mais para a esquerda/direita.

Os detalhes de trigonometria serão explorados mais tarde. No fundo cada rotação é decomposta nas suas várias componentes em relação a um sistema de coordenadas.



Se trabalharmos um pouco cada uma destas componentes, chegamos à conclusão de que o nosso vector que define para onde aponta a câmara tem a seguinte forma:

```
direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
direction.y = sin(glm::radians(pitch));
direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
```

Configurar o input do rato

A ideia é capturar as coordenas do rato a cada frame. Quanto maior for a diferença das coordenadas de frame para frame, maior será o ângulo que a câmara faz.

Avisar a biblioteca *glfw* para capturar as coordenadas do rato:

```
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
```

Assim como definimos uma callback para o teclado, vamos definir para o rato:

```
void mouse callback(GLFWwindow* window, double xpos, double ypos);
```

Há quatro passos:
1- Calcular a diferença das coordenadas entre as frames consecutivas;
2- Adicionar esta diferença ao valor de pitch e yaw, que vão acumulando;
3- Verificar restrições de valores mínimos e máximos;
4- Por fim, calcular o vector de direção da câmara.

Vamos dizer que a posição inicial do rato é o centro do ecrã, definidas como variáveis globais.

```
float lastX = WIDTH/2, lastY = HEIGHT/2;
```

Dentro da função de mouse callback:
Passo 1:

```
float xoffset = xpos - lastX;
float yoffset = lastY - ypos;
lastX = xpos;
lastY = ypos;
```

Passo 2:

```
yaw   += xoffset;
pitch += yoffset;
```

Nota: yaw e pitch são variáveis globais, inicializadas como:

```
float yaw = -90.0, pitch = 0.0;
```

Passo 3: (parecido a deteção de colisões: check coordenada, se estiver no limite, não incrementar/decrementar)

```
if(pitch > 89.0f)
        pitch =  89.0f;
if(pitch < -89.0f)
        pitch = -89.0f;
```

Passo 4:

```
glm::vec3 direction;
direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
direction.y = sin(glm::radians(pitch));
direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
cameraFront = glm::normalize(direction);
```

O ficheiro src3 implementa esta funcionalidade.

Exercício

1- (importante funcionar para as próximas aulas)

O ficheiro src2 implementa o input do teclado para movimentar a câmara e o ficheiro scr3 implementa o input do rato para girar a câmara.
Crie uma classe Camera.hpp que combine estas duas funcionalidades para a câmara poder navegar livremente no ambiente.

2-

Escreva a sua própria função lookAt, que retorne a matrix lookAt. A função recebe a posição da câmara, o ponto para onde a câmara aponta, e o vector Up. Assinatura da função deverá ser a seguinte:
glm::mat4 my_lookAt_matrix(glm::vec3 position, glm::vec3 target, glm::vec3 worldUp)