|  |  |
| --- | --- |
| http://upload.wikimedia.org/wikipedia/en/2/20/Instituto_Superior_T%C3%A9cnico_logo.png | Relatório 1 AVT  Grupo 12  Bruno Rodrigues nº66954  João Ferreira nº67018 |

**Base Utilizada**

A fim de concretizarmos o primeiro assignment para este projecto, decidimos usar o código do “Cube Example” disponibilizado nas aulas, bem como as técnicas que foram usadas nos Laboratórios 1 e 2.

De seguida, iremos explicar como realizámos cada uma das tarefas.

**Tarefa 1**

Em primeiro lugar decidimos começar com a modelação dos elementos do jogo que foram pedidos no enunciado.

Inicialmente tentámos alterar o cubo do exemplo de forma a transformá-lo em um rectângulo correspondente à estrada. Para tal, alterámos o método *renderScene* de forma a realizar uma transformação de escala.

Tendo completado esta primeira etapa, seguiu-se criar as bermas da estrada. De forma a simplificar o trabalho nesta fase inicial, duplicámos o código referente à estrada, fazendo as escalas e translações necessárias. Para não termos de carregar a *Model Matrix* por cada elemento a ser desenhado, fazemos *push* da matriz antes das transformações, e *pop* depois de desenhar o elemento.

Os restantes elementos do jogo foram conseguidos de forma idêntica, sendo que todos eles são compostos por cubos ou paralelepípedos.

Uma etapa fundamental na realização desta tarefa foi dar a cor pretendida aos elementos. Para isso, definimos uma variável *uniform* do tipo *vec4* no *Fragment Shader*, e um conjunto de cores no código principal. Antes de desenhar cada elemento, atribuímos à variável *uniform* a cor pretendida.

**Tarefa 2**

Para a definição das 3 câmeras optámos por atribuir variáveis aos argumentos da função *lookAt*, alterando essas variáveis quando é premida uma das teclas. Ao escolher uma das câmeras, carregamos também a matriz identidade na *Projection Matrix* e chamamos as funções *ortho* ou *perspective,* consoante a escolha.

Como na câmera móvel o *lookAt* terá de ser chamado com os parâmetros actualizados, existe também uma flag, *isIn3*, que simboliza quando está a câmera 3 (móvel) escolhida, e indica se se deverá proceder a essa actualização. Esta flag é posta a *true* ou *false* no momento de premir a tecla.

Como não percebemos se a câmera móvel deveria seguir sempre o sapo ou orbitar em torno deste, como nos exemplos, optámos por implementar as duas versões.

**Tarefa 3**

Uma vez que no “Cube Example” estáva já definido um movimento de câmera semelhante ao pretendido, utilizámos esse código como base.

A maior diferença foi, como referido no ponto anterior acerca da escolha da câmera, os parâmetros da função *lookAt*, com o *eye* da câmera a coincidir com a posição do sapo (mais o devido acerto para o sapo ser visível).

Como pensámos que o movimento da câmera devia ser apenas horizontal, no cálculo dos parâmetros retirámos as variáveis que lidavam com o movimento vertical. Na próxima entrega teremos já o movimento como era esperado.

**Tarefa 4**

Para esta tarefa, associámos os movimentos do sapo às respectivas teclas pedidas no enunciado, de modo a conseguir que o sapo se movesse na cena. Modificámos a função *processKeys* para que cada vez que pressionássemos uma das teclas de movimento o sapo se movesse uma casa nessa direcção, e movendo-se continuamente se a tecla fosse deixada pressionada. Para os movimentos horizontais temos uma variável, *moveX,* e para os movimentos verticais a *moveZ*, sendo que estas variáveis são usadas na translação do sapo para os movimentos ao longo do eixo X e Z respectivamente.

Para o movimento pela tecla direita do rato modificámos as funções *processMouseButtons* e *processMouseMotion,* na qual determinamos qual a posição anterior do cursor do rato na janela em função dos eixos X e Z (MousePositionX e MousePositionZ respectivamente), sendo que de seguida foi necessário definir uma diferença mínima entre a posição actual e a posição anterior do rato de modo a que o sapo não se mexa muito abruptamente, devido à sensibilidade do rato.

**Tarefa 5**

De forma a implementar o movimento e geração aleatória dos objectos, criámos a função *myTimer*, chamada a cada 1000/60 milisegundos. A função faz com que a velocidade de cada objecto seja aumentada a cada três segundos. Quando um autocarro ou um carro chegam ao fim da estrada, é calculado um valor *random* compreendido entre 0 e 3 que determina qual das quatro hipóteses definidas para a geração de novos objectos no método *renderScene* vai ser executada. Como tarefa futura, iremos criar classes para cada elemento, de forma a ter uma forma de gestão de objectos mais correcta e menos confusa.