Parte 2

Cena Simples Interativa com Câmara Móvel e Colisões

Objectivos

Os objectivos da segunda parte dos trabalhos de laboratório são explorar o conceito de câmara virtual, as diferenças entre câmara fixa e câmara móvel, as diferenças entre projecção ortogonal e projecção perspectiva, a compreensão das técnicas básicas de animação e a detecção de colisões.

A avaliação da segunda parte do trabalho será realizada na semana de **27 a 31 de Outubro** e corresponde a **5 valores** da nota do laboratório. A realização deste trabalho tem um esforço estimado de **10 horas** por elemento do grupo, distribuído por duas semanas.

Tarefas

As tarefas para a segunda parte são:

1. Definir duas câmaras adicionais tendo o cuidado de **manter a câmara definida anteriormente**. Deve ser possível alternar entre as três câmaras utilizando as teclas "1", "2" e "3". A câmara 2 deve ser fixa e permitir visualizar todo o terreno de jogo através de uma projecção perspectiva. A câmara 3 deve também utilizar uma projecção perspectiva mas é móvel. Esta deve estar colocada atrás do sapo e acompanhar o seu movimento (o sapo deve estar visível). [**1,5 valores**]

2. Implementar o movimento dos carros e troncos. Este deve ser um movimento rectilíneo uniforme, à semelhança do jogo original. Quando a tecla respectiva estiver pressionada, o movimento do sapo também deve ser uniforme e contínuo. Diferentes tipos de elementos devem movimentar-se com velocidade diferente e a sua

velocidade vai aumentando com o tempo de jogo, i.e quanto mais tempo o utilizador levar a atravessar a estrada e o rio, mais rápido os elementos se movem. Após sair do campo de jogo (da estrada ou do rio) os carros e os troncos devem desaparecer. O aparecimento dos objectos no jogo deve seguir um comportamento aleatório. [**2,0 valores**]

3. Detectar a colisão do sapo com os limites do jogo, do sapo com os carros, e troncos. Estas colisões são detectadas em duas dimensões usando caixas envolventes alinhadas com os eixos (axis aligned bounding boxes - AABB), conforme exemplificado na Figura 1, e fazendo comparações dos limites das caixas, conforme ilustrado na Figura 2. Quando atingir os limites do jogo o sapo deve parar, mesmo que o utilizador mantenha premida a tecla que o movimenta para o lado onde a colisão está a ocorrer. Ao colidir com um carro o sapo morre. Caso esteja no rio, o sapo morre sempre que não estiver sobre um tronco. Quando morre, o sapo volta à posição inicial. [**1,5 valores**]

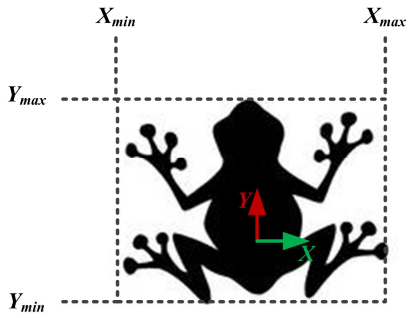

Figura 1 – Caixa envolvente alinhada com os eixos (AABB).

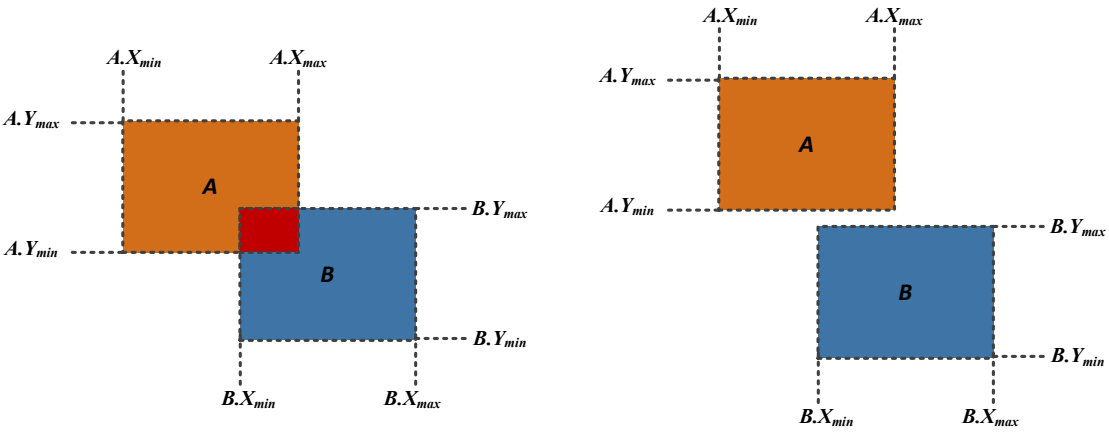

Figura 2 – Exemplo de detecção de colisão entre caixas envolventes alinhadas com os eixos (AABB) em duas dimensões.
Esquerda: com colisão. Direita: sem colisão.

Importante

A implementação do trabalho desenvolvido nos laboratórios de computação gráfica **deve usar** o ciclo de animação (*update/display cycle*). Este é um padrão de desenho usado nas aplicações de computação gráfica interactiva. Este ciclo, ilustrado na Figura 4, separa o desenho da cena no ecrã da actualização do estado do jogo em duas fases distintas. Na fase de *display* são cumpridos três passos base: limpar o buffer; desenhar a cena e forçar o processamento dos comandos. Na fase de *update* todos os objectos do jogo são actualizados de acordo com a física inerente. Para este trabalho pode-se aplicar apenas a equação do movimento linear uniforme. É ainda nesta fase que se processa a detecção de colisões e implementação dos respectivos comportamentos.

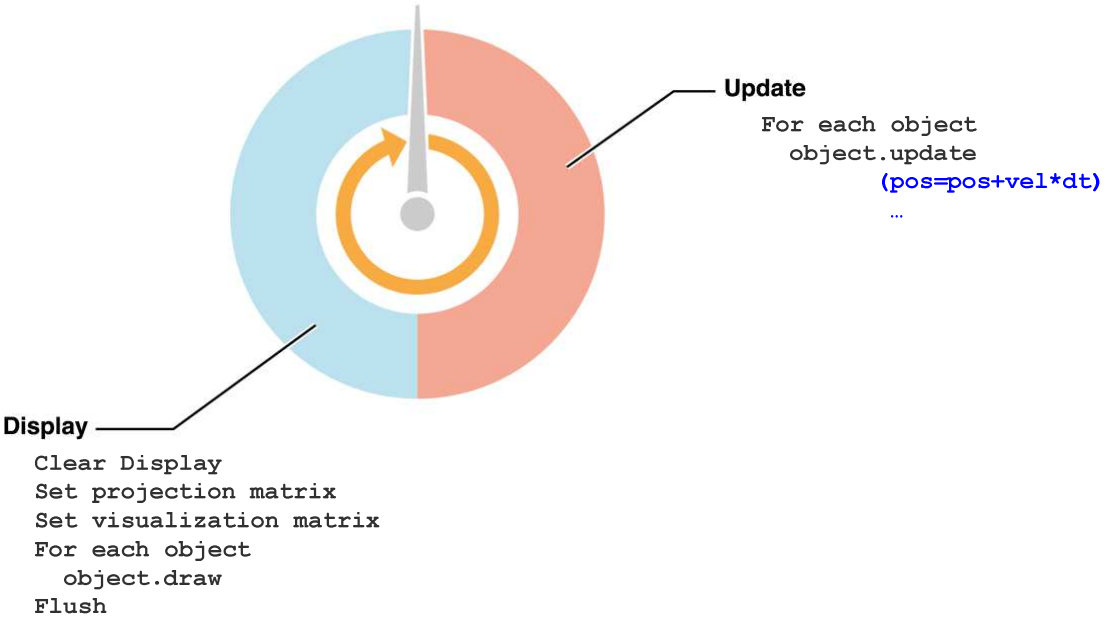

Figura 3 - Ciclo de animação (update/display cycle)

Os acontecimentos restantes, tais como teclas pressionadas ou soltas, temporizadores e redimensionamento, devem ser tratados pelas respectivas funções de *callback* de forma independente.

Sugestões

1. Deve usar um temporizador com um intervalo aleatório para definir quando um determinado objecto do jogo (carro ou tronco) volta a aparecer.

2. Para incrementar a velocidade dos objectos com o tempo de jogo é recomendado o uso de um temporizador com algumas dezenas de segundos. Ao disparar o temporizador, a velocidade dos carros e dos troncos aumenta.

3. Para obter um resultado mais correcto, a colisão entre objectos deve ser detectada considerando o percurso realizado por estes entre a posição actual (Pos_t) e a posição anterior ($Pos_{t-\Delta t}$), tal como ilustrado na Figura 4.

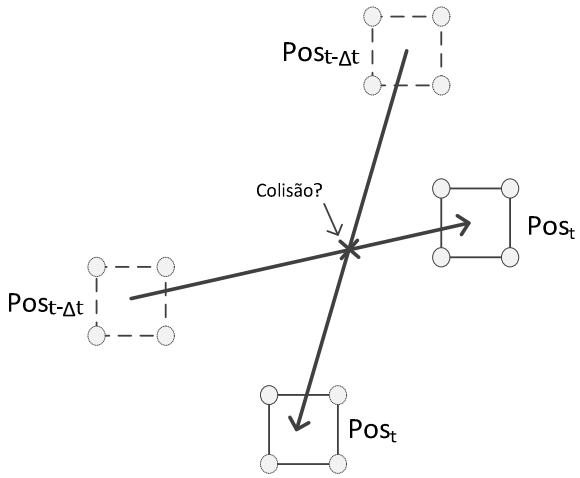

Figura 4 - Detecção de colisões entre AABBs em movimento.

4. A implementação da detecção de colisões deve ser única para todos os objectos. Esta afirmação é válida tanto para colisões entre objectos como para as colisões com os limites do campo de jogo.

5. Algumas das funções a estudar:

   - **`gluPerspective, gluLookAt`**
   - **`glutTimerFunc`**
   - **`glutIdleFunc`**
   - **`glutGet(GLUT_ELAPSED_TIME)`**
   - **`glEnable(GL_DEPTH_TEST)`**
   - **`rand, srand`**