INSTITUTO POLITÉCNICO DO CÁVADO E AVE

TRABALHO PRÁTICO – FASE 2

RELATÓRIO

JORGE MIGUEL AREZES NORO | 15705

ENGENHARIA E DESENVOLVIMENTO DE JOGOS DIGITAIS

INTRODUÇÃO À PROGRAMAÇÃO 3D

DOCENTE: JOSÉ HENRIQUE BRITO

2º ANO - 1º SEMESTRE

OBJETIVOS

Este relatório serve de apoio ao código e programa entregue. Os objetivos definidos pelo professor para a segunda fase do trabalho prático da unidade curricular são adicionar iluminação à cena criada na fase 1 e adicionar os tanques. Os tanques devem aparecer renderizados e devem ser possíveis controlar pelo utilizador. Estes objetivos podem ser observados no programa entregue, dentro da pasta "programa". Todo o código fonte pode ser consultado na pasta "codigo".

Durante a elaboração deste trabalho foi tida em consideração a portabilidade e modularidade do código, de forma a poder ser reaproveitado ao máximo para outros projetos e facilitar futuras consultas.

Para realizar este trabalho, foi utilizado o software de edição e compilação de código Microsoft Visual Studio[1], a *framework* Monogame[2] 3.6 e GitHub[3] para *source control*.

[1] https://visualstudio.microsoft.com
[2] http://www.monogame.net
[3] https://github.com

ILUMINAÇÃO

Na primeira fase, a iluminação era realizada recorrendo ao *BasicEffect built in* no *Monogame*. Permitia uma fácil abordagem para começar a visualizar resultados, mas limitava as opções no que diz respeito a efeitos de renderização. Para esta fase, a iluminação sofreu um *upgrade* para *shaders* personalizados, desenvolvidos em HLSL[4]. Esta evolução irá permitir adicionar *features* mais avançadas que não estão disponíveis no *BasicEffect*. Estas técnicas são por exemplo, e a que se espera vir a implementar, *Normal Mapping*[5].

Para esta fase, decidiu-se apenas incluir *Ambient Lighting* e *Diffuse Lighting*, utilizando uma textura. A luz ambiente é responsável por adicionar detalhe aos polígonos dos objetos 3D não expostos diretamente a uma luz. Pode-se pensar nesta luz como o lado do objeto à sombra. *Diffuse Lighting* é a luz que ilumina diretamente os objetos. Na cena existe uma única luz infinita, que é constituída por uma direção, uma cor e uma intensidade (entre 0 e 1).

O cálculo tem como base a textura fornecida ao *shader*. Ambas as cores *ambient* e *diffuse* são dadas pela dita textura. A cor da luz é branca. A intensidade da *diffuse light* é medida pelo ângulo da normal do vértice (interpolada, pois é utilizado pixel *shading*) e a direção da luz. A luz ambiente é independente de qualquer direção, sendo de intensidade constante em todos os pixéis. Podemos observar em baixo um modelo simplificado de como é obtida a cor dos pixéis. Para a implementação completa, consultar o ficheiro *Content/Effects/Diffuse.fx*.

$$ambient = textureColor + ambientColor * ambientIntensity;$$
$$diffuse = diffuseColor * (normal \,.\, lightDirection) * lightIntensity;$$
$$pixelColor = textureColor * diffuse + ambient * ambientIntensity;$$

Figura 2- Tanques iluminados apenas com luz ambiente

Figura 1-Tanques iluminados com ambient + diffuse

[4] https://en.wikipedia.org/wiki/High-Level_Shading_Language
[5] https://en.wikipedia.org/wiki/Normal_mapping

TANQUES

A figura central da aplicação serão os tanques, controlados pelos jogadores. A implementação foi feita recorrendo a uma classe com toda a informação necessária para o seu comportamento. A classe é composta por métodos e propriedades para a lógica e renderização do tanque.

A lógica do tanque consiste em verificar input do jogador e responder conforme. Isto é feito no método *Update*. Os *inputs* que o jogador pode dar para controlar o comportamento do tanque, é a rotação no espaço e a velocidade com que se desloca. Primeiramente é atualizado o vetor que guarda os valores de rotação do tanque, com o *input* recebido. Após receber estes valores, atualizam-se os vetores direcionais do tanque, que são os vetores que indicam em que direção o tanque está orientado no espaço (*right*, *up*, *front*). É ainda tida em conta a superfície do terreno, pois a orientação em *Y* (*Up*) é dada pela interpolação das normais do terreno, em determinada posição. A implementação pode ser verificada no ficheiro de código *Tank.cs*, método *UpdateDirectionVectors*, linha 276.
De seguida, é verificado o *input* de movimento, e atualiza-se a variável *Speed*, que guarda a acumulação de velocidade. Esta é aplicada ao vetor *Velocity* (*Front* * Speed). Para evitar que o tanque atinja velocidades demasiado elevadas, limita-se essa velocidade, verificando se o vetor *velocity* supera o valor da variável *MaxVelocity*. Se for verdade, normaliza-se o vetor e multiplica-se pela velocidade máxima. Por fim, soma-se a velocidade à posição do tanque e aplica-se *dampening* na velocidade, para que diminua a velocidade simulando atrito com os elementos do mundo.

Após atualizar a posição do tanque, é verificado se o mesmo se encontra em posição irregular, ou seja, fora do terreno. Se for verdade, fazemos *clamp* ao *edge* mais próximo. Ainda fazemos mais um ajuste à posição, que é a elevação (*Position.Y*). A posição em Y é dada pela superfície do terreno. Para isso utilizamos o mesmo método que na fase 1, era utilizado pela *surface follow camera*.

Com todas as verificações feitas, atualiza-se a matriz do tanque, aplicando-a à *transform* do *root bone*. O modelo contém *bones*, que permitem facilmente implementar animações ao modelo 3D.

O render do tanque é efetuado através de *custom shaders*. O método é simples, é feito um ciclo por todas as partes de cada *mesh* do modelo, e aplica-se o *effect* com os devidos parâmetros. Os parâmetros passados ao *shader* são os seguintes:

- *World Matrix:* Matriz que especifica posição, escala e rotação, em coordenadas locais;
- *View Matrix:* Matriz que especifica posição e direção da câmara;
- *Projection Matrix:* Matriz que contém os valores de um *frustum*[6];
- *WorldInverseTranspose Matrix:* A inversa transposta da *World Matrix*. Usada para transformar a luz em vetor local ao vértice ou pixel;
- *DiffuseLightDirection:* Vector4 com a direção da luz;
- *DiffuseColor:* Vector4 com a cor da luz;
- *DiffuseIntensity:* float com a intensidade da luz;
- *Texture:* a textura a ser usada.

[6] https://en.wikipedia.org/wiki/Frustum

CONCLUSÕES

A segunda fase deste projeto trouxe um grande desafio com o objetivo de utilizar *shaders* em HLSL. Acreditando que irá trazer vantagens à qualidade do projeto, espero na próxima fase conseguir implementar efeitos mais avançados, se o tempo assim o permitir. Houve também uma grande preocupação em agilizar os controlos, pelo que foram criadas estruturas que facilita a leitura e compreensão e simplifica a utilização e customização.

As câmaras aplicadas em terceira pessoa nos tanques são também um motivo de orgulho pessoal. Apesar das dificuldades sentidas em matemática, foi de certa forma fácil pensar na modelação da câmara em terceira pessoa, seja no modo *Orbit* ou *Locked*.

Apesar do relatório ser relativamente curto, o leitor deve-se sentir à vontade para analisar o código fonte, pois este está desenvolvido de uma forma intuitiva e documentado.

Encoraja-se também o utilizador a experimentar as *features* adicionais. Estas podem ser consultadas ao pressionar a tecla 'H' dentro da aplicação.