Laboratório de Aplicações com Interface Gráfica
Aulas Práticas

MIEIC – 2018/2019

Trabalho Prático 2
Aperfeiçoamento das Técnicas de Utilização de WebGL

A. Introdução

O objetivo deste trabalho é explorar técnicas gráficas, como animação ou *shaders* baseados em GLSL ES (OpenGL for Embedded Systems' Shading Language). Propõe-se assim a implementação de algumas funcionalidades em código, que se devem traduzir em extensões à linguagem YAS e exploradas através da criação de uma cena que as utilize. Este documento descreve as funcionalidades pretendidas, bem como as extensões propostas.

B. Funcionalidades pretendidas

1 Animação

Implementar um conjunto de classes para o suporte a animações.

1. Implementar a classe *Animation* como classe base para aplicar animações a um objeto. Esta classe deve conter um método ***update*** e um método ***apply***, para respetivamente atualizar o seu estado em função do tempo, e aplicar a transformação sobre a matriz de transformações da cena quando adequado.

2. Criar a classe *LinearAnimation*, derivada de *Animation*, para trajetórias poligonais, que permita definir uma animação caracterizada por um vetor de Pontos de Controlo e tempo de duração total em segundos.

   Exemplo:
   Pontos de Controlo = {(0,0,0), (1,0,0), (1,1,0)}
   Tempo= 10 s

   O objeto em movimento deve alterar a sua orientação horizontal (x,z), rodando em torno de um eixo vertical, de modo a corrigir a direção quando, de acordo com a trajetória, muda de segmento de reta (ou seja, um movimento de "helicóptero"). Essa transição de orientação pode ser imediata.
   NOTA 1: A transformação animada calculada num dado instante com base naqueles pontos de controlo será acumulada com (e portanto afetada por) outras transformações aplicadas anteriormente na hierarquia do grafo.
   NOTA 2: Considera-se que a "frente" do objeto na sua posição original aponta na direção positiva do eixo dos ZZ.

3. Criar a classe *CircularAnimation*, derivada de *Animation*, para trajetórias circulares, que permita definir uma animação caracterizada pelo centro e raio de circunferência, ângulo inicial (medido em relação à direção positiva do eixo XX, ângulo de rotação, e tempo de duração em segundos. Exemplo:

   Centro = (10, 10, 10)
   Raio = 5
   Ângulo Inicial = 40º
   Ângulo de rotação = 20º
   Tempo= 20 s

   O objeto em movimento deve alterar a sua orientação horizontal (x,z) de acordo com a trajetória, tal como no caso anterior.
   NOTA 1: Aplicam-se as mesmas notas do ponto anterior.
   NOTA 2: os ângulos deverão ser expressos em graus.

4. Implementar (usando a extensão proposta à YAS) uma animação para um veículo a definir (ver pontos seguintes) que inclua, no seu trajeto, pelo menos dois segmentos de reta e um segmento circular, recorrendo respetivamente às animações poligonais e circulares anteriormente referidas. O objeto mantém a sua horizontalidade (ou seja, mantém-se paralelo ao plano XZ), apenas podendo rodar em torno do seu eixo vertical, de forma a manter uma orientação coerente com a direção e sentido do seu movimento (como, por exemplo, um helicóptero).
   NOTA: no caso de sequências de animações num mesmo componente (ver extensão da YAS), a situação inicial de uma animação não depende da situação final da animação anterior. Logo, se se pretender continuidade entre duas animações consecutivas, cabe a quem criar a cena a responsabilidade de garantir que a posição final da primeira animação é igual à posição inicial da seguinte.

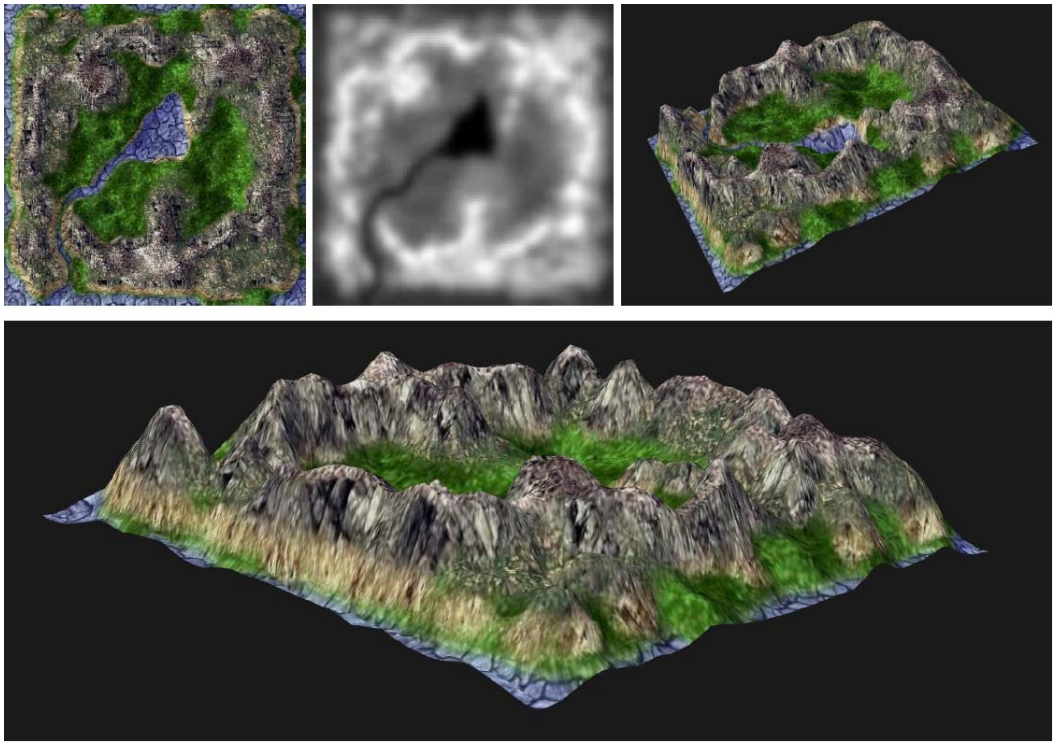
2 Superfícies 2D/3D

As superfícies vão ser modeladas através da representação *Non-Uniform rational basis spline* (NURBS) (https://en.wikipedia.org/wiki/Nonuniform_rational_Bspline).

1. Crie uma classe *Plane*, extensão de *CGFobject*, de forma a gerar, utilizando NURBS, um plano de dimensões 1 x 1 unidades, assente em XZ, centrado na origem e com a face visível apontando para +Y. O número de divisões nas direções U e V pode ser distinto e deve ser especificado no construtor da classe. Com esta classe, criar uma primitiva *plane* na linguagem YAS.
2. Criar uma nova primitiva *patch* a incluir na linguagem YAS que possa representar superfícies de grau 1, 2, 3 ou superior, nas duas direções U e V (admitem-se graus diferentes nas duas direções).
3. Criar uma nova primitiva *cylinder2* a incluir na linguagem YAS que represente uma superfície cilíndrica (sem tampas) respeitando as definições de *stacks* e de *slices*.
4. Criar, usando as extensões ao YAS propostas abaixo, um novo objeto que corresponde a um veículo voador que inclua pelo menos uma superfície não plana, gerada utilizando as primitivas *patch* e *cylinder2*.

3 Shaders

1. **Terreno**: Crie uma nova primitiva *terrain*, baseada na classe *Plane* criada anteriormente, para representar um terreno com elevações, recorrendo a uma textura de cor, uma textura que funciona como mapa de alturas (ver imagens abaixo) e um par de shaders - vertex e fragment.



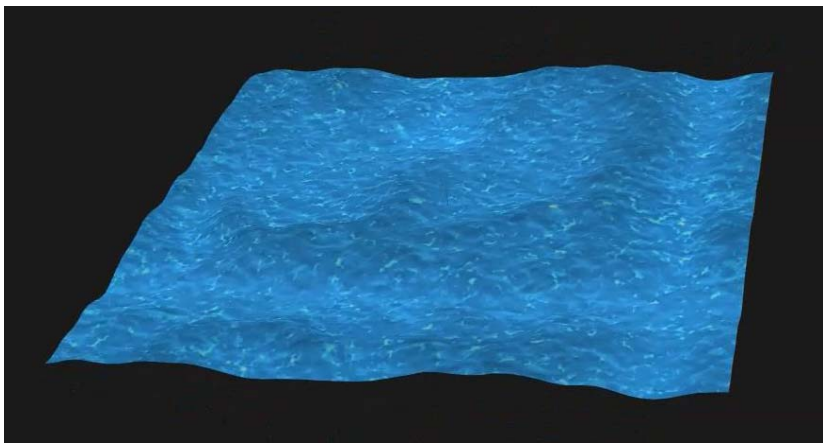
(textura de cor, mapa de alturas e geometria gerada; origem das texturas: Outside of Society http://oos.moxiecode.com/js_webgl/terrain/index.html)

NOTAS:
O *vertex shader* deve ser aplicado a um plano e alterar as coordenadas de cada vértice de modo a que a sua coordenada Y (altura) varie segundo a informação contida na imagem correspondente ao mapa de alturas.
O *fragment shader* deve aplicar a textura de cor sobre o terreno.
As texturas usadas devem ter dimensões que sejam potências de 2.

2. **Plano de água**: À semelhança da primitiva *terrain* definida no ponto anterior, crie uma primitiva *water* que use também uma textura (neste caso de água) e um mapa de alturas (que funcionará para simular a ondulação da água) para criar uma superfície de baixo relevo.

   Esta primitiva deve ser animada através dos seus próprios shaders, onde se deve variar a associação das coordenadas de textura aos vértices e fragmentos ao longo do tempo, para obter um efeito semelhante ao que pode ser visto no exemplo abaixo.

   NOTAS: Aplicam-se as notas do ponto anterior.



Vídeo: https://drive.google.com/open?id=1gSgOrhpVg10GxwIXMBcRewxV5dU8o1FH

C.Requisitos da cena

Deve ser criada uma cena que utilize as funcionalidades referidas acima, nomeadamente:

- As superfícies 2D e 3D.
- A instanciação e animação do veículo animado de acordo com as indicações acima (preferencialmente voando e/ou pousando/levantando).
- A instanciação de uma área de terreno usando a primitiva *terrain* proposta (numa parte do terreno usado, por exemplo uma região montanhosa envolvendo a cena, ou uma zona específica da cena com um determinado relevo).
- A instanciação de um plano de água que intersete o terreno, criando zonas de água como no exemplo abaixo.

Vídeo: https://drive.google.com/open?id=1wwJfKY7mUN2EC1m1RYqU0hlhap7ZHDVK

D.Avaliação do trabalho

Composição dos Grupos: Os trabalhos devem ser efetuados em grupos compostos por dois estudantes. Preferencialmente, cada grupo deve ser composto pelos mesmos elementos que trabalharam no TP1. Em caso de impossibilidade (por exemplo, por falta de paridade numa turma), deve ser discutida com o docente a melhor alternativa.

Avaliação do Trabalho de Grupo: A avaliação será feita em aula prática, numa apresentação de cada grupo ao docente respetivo.

Avaliação Individual: Na prova de avaliação individual, a realizar no dia 28 de Novembro de 2018, serão pedidas várias funcionalidades adicionais, a implementar sobre o código original desenvolvido em trabalho de grupo. Estas funcionalidades poderão envolver alterações em partes do código referentes ao TP1 .

Avaliação do Trabalho: Média aritmética das duas avaliações anteriores.
De acordo com a formulação constante na ficha de disciplina, a avaliação deste trabalho conta para a classificação final com um peso de: 80% * 30% = 24%.

Critérios de Avaliação do Trabalho de Grupo
Tendo em atenção as funcionalidades enunciadas, serão considerados os seguintes critérios para efeitos de Avaliação do Trabalho de Grupo:

Estruturação e Documentação do código	3 valores
Interface, aspeto geral e criatividade	2 valores
Animação	5 valores
Superfícies	5 valores
Shaders	5 valores

O enunciado incorpora, em cada alínea, a sua classificação máxima, correspondendo esta a um ótimo desenvolvimento, de acordo com os critérios seguintes, e que cumpra com todas as funcionalidades enunciadas. Sem perda da criatividade desejada num trabalho deste tipo, não serão contabilizados, para efeitos de avaliação, quaisquer desenvolvimentos além dos que são pedidos.

Planeamento do Trabalho
- Semana 1 (início em 22/10/2018): Animação
- Semana 2 (início em 29/10/2018): Semana de interrupção
- Semana 3 (início em 05/11/2018): Superfícies de grau 1, 2 e 3
- Semana 4 (início em 12/11/2018): Shaders
- Semana 5 (início em 19/11/2018): Finalizações

Datas principais
- Data limite de entrega do trabalho completo: 25/11/2018 (via moodle)
- Avaliação dos trabalhos de grupo: aulas práticas, semana de 26/11/2018
- Prova de avaliação individual: 28/11/2018, 17:00-20:00

E.Extensão à Linguagem YAS

A linguagem YAS encontra-se globalmente definida no enunciado do trabalho prático 1. Nesta secção são apresentadas as extensões ao formato YAS de modo a poder comportar as funcionalidades descritas no presente enunciado (TP2). Ao ser lido e interpretado por uma aplicação gráfica, um ficheiro *xml* que descreve uma cena YAS deve ser verificado em termos de sintaxe, devendo a aplicação gerar mensagens de erro ou avisos, identificando eventuais erros encontrados ou situações anómalas ou indesejáveis. Na descrição abaixo, os símbolos utilizados têm o seguinte significado:

- ii: valor inteiro
- ff: valor em vírgula flutuante
- ss: string
- cc: caracter "x" ou "y" ou "z", especificando um eixo
- tt: valor Booleano na forma "0" ou "1"

Segue-se uma listagem representativa da sintaxe pretendida, relativa às extensões à linguagem YAS. As tags / atributos acrescentados encontram-se escritos à cor vermelha. À cor preta encontram-se elementos definidos na versão original da linguagem YAS, usados para melhor contextualizar as alterações. Os comentários estão escritos à cor verde.

```
<yas>
...
<!--informacao de animacao -->
<!-- o bloco "animations" deve ser declarado -->
<!-- imediatamente antes do bloco  "primitives" -->
<animations>
<!-- O bloco animations pode ser vazio, isto é, pode -->
<!-- não ser declarada qualquer animação, linear ou circular -->
<!-- Span é o tempo, em segundos, que a animação deve demorar -->
      <linear id="ss" span="ff" >
      <!-- devem existir pelo menos dois pontos de controlo -->
            <controlpoint xx="ff" yy="ff" zz="ff" />
            ...
      </linear>
      <!-- center corresponde ao ponto que serve -->
      <!-- de centro da animação circular -->
      <!-- radius corresponde ao raio da rotacao -->
      <!-- para a animação circular -->
      <!-- startang e rotang correspondem, nomeadamente, -->
      <!-- ao angulo inicial (em graus) e -->
      <!-- total de rotação (em graus) -->
      <circular id="ss" span="ff" center="ff ff ff" radius="ff"
            startang="ff" rotang="ff" />
</animations>


<primitives>
<primitive id="ss">

      <!-- Nova primitiva: plano, gerado por NURBS -->
```

```
<plane npartsU="ii" npartsV="ii" />
<!-- ex: <plane npartsU="5" npartsU="8" />
<!-- um plano de dimensões 1 x 1 unidades assente →
<!-- em XZ, centrado na origem -->
<!-- e com a face visível apontando para +Y -->
<!-- com divisão em cinco partes por oito partes -->

<!-- Nova primitiva: patch, gerada por NURBS -->
<!-- – parâmetros: -->
<!-- – npartsU: divisão em partes no domínio U a -->
<!-- ser usada para o cálculo da superfície -->
<!-- – npartsV: divisão em partes no domínio V -->
<!-- a ser usada para o cálculo da superficie -->
<!-- - o número de pontos de controlo dentro da -->
<!-- primitiva patch é npointsU * npointsV -->
<patch npointsU="ii" npointsV="ii"
               npartsU="ii"  npartsV="ii" >
    <controlpoint xx="ff" yy="ff" zz="ff" />
    ...
</patch>

<!-- Nova primitiva vehicle: corresponde a um veículo voador.
     Inclui pelo menos uma superfície não plana gerada
     utilizando NURBS e em código javascript -->
<vehicle />

<!-- – Nova primitiva: cilindro baseado em NURBS -->
<!-- parâmetros iguais ao cilindro original -->
<cylinder2 base="ff" top="ff" height="ff" slices="ii"
                stacks="ii" />

<!-- Nova primitiva: terreno baseado em shaders -->
<!-- parametros: -->
<!-- id da textura que deve ser visualizada sobre o terreno
     (dimensões devem ser potências de 2) -->
<!-- id da textura que deve ser usada como mapa de alturas
     para formar o terreno (dimensões devem ser potencias de
     2) -->
<!-- numero de divisoes em s e t (parts="5" => plano com 5x5
     divisoes -->
<!-- fator de escala das alturas -->

<terrain idtexture="ss" idheightmap="ss" parts="ii"
     heightscale="ff"/>

...
<!-- Nova primitiva: plano de água baseado em shaders -->
<!-- parametros: -->
<!-- id da textura que deve ser visualizada sobre o terreno
     (dimensões devem ser potências de 2) -->
```

```
      <!-- id da textura que deve ser usada como mapa de ondulação
           para formar o terreno (dimensões devem ser potencias de
           2) -->
      <!-- numero de divisoes em s e t (parts="5" => plano com 5x5
           divisoes -->
      <!-- fator de escala das alturas -->
      <!-- fator de escala das coordenadas de textura (para o numero
           de repeticoes no plano) -->

      <water idtexture="ss" idwavemap="ss" parts="ii"
           heightscale="ff" texscale="ff" />

</primitive>
...
</primitives>


<components>
      <component ...>
      ...
      <!-- bloco "animations" e' opcional →
      <!-- as transformacoes resultantes da animacao devem -->
      <!-- declarar-se imediatamente após as transformacoes -->
      <!-- do componente -->

      <animations>
      <!-- pode conter zero ou mais referencias a "animations" -->
      <!-- declaradas anteriormente e que serão executadas em -->
      <!-- sequencia. Este elemento e' opcional -->
         <animationref id="ss" />
      </animations>
      ...
      </component>
</components>
</yas>
```