Laboratório de Aplicações com Interface Gráfica
Aulas Práticas

MIEIC – 2020/2021

Trabalho Prático 2
Aperfeiçoamento de técnicas utilizadas em Computação Gráfica

(rev 2020.11.19.1630)

1. Introdução

O objetivo deste trabalho é explorar técnicas gráficas como NURBS (Non-uniform Rational B-spline), animação por *keyframes* e o uso de spritesheets para desenho de texto e efeitos visuais recorrendo a *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 LSF e exploradas através da criação de uma cena que as utilize. Este documento descreve as funcionalidades pretendidas, bem como as extensões propostas.

2. Funcionalidades pretendidas

2.1. Animação por *keyframes*

A animação por *keyframes* (quadros-chave) baseia-se na definição de uma sequência de chaves. Cada chave descreve, para um dado instante medido em segundos, a representação do estado de um objecto; apenas as transformações geométricas - translação, rotação e escalamento (a executar como se fossem escritas por esta ordem) - são consideradas neste trabalho. A matriz de transformação correspondente à animação deve ser multiplicada à direita das transformações declaradas no nó.
A animação inicia-se no instante declarado na *keyframe* 0 (ver abaixo), sendo que o objeto em causa deve permanecer invisível até esse momento. Considerando, a partir daí, pares consecutivos de *keyframes*, a animação deve interpolar os valores relativos a cada uma das 3

NOTA: após a última chave de uma animação, o objeto permanece visível e imóvel, de acordo com as transformações dessa chave. Manim=T*R*S

Exemplo

Considere-se a seguinte definição conceptual de uma animação:

- **Keyframe 0**: Instante: 5
  - Translação (0, 0, 0)
  Rotação (0, 0, 0)
  Escalamento (1, 1, 1)
- **Keyframe 1**: Instante: 10
  Translação: (0, 0, 0)
  Rotação: (0, 0, 0) *(equivalente a escrever R(0,x); R(0,y); R(0,z))*
  Escalamento: (1, 1, 1)
- **Keyframe 2**: Instante: 15
  Translação: (0, 20, 0)
  Rotação: (0, 360, 0) *(equivalente a escrever R(0,x); R(360,y); R(0,z))*
  Escalamento: (1.2, 1.2, 1.2)
- **Keyframe 3**: Instante: 20
  Translação: (0, 0, 0)
  Rotação: (0, 0, 0) *(equivalente a escrever R(0,x); R(0,y); R(0,z))*
  Escalamento: (1, 1, 1)

A definição anterior representa um comportamento que, quando aplicado a um objeto, ocorre da seguinte forma:

- Até ao instante 5 seg, o objeto permanece invisível, aparecendo nesse momento com a posição inicial e inicia a animação;
- Entre o instante 5 seg e o instante 10 seg, o objeto mantém-se imóvel, porque as transformações geométricas do *keyframe* 1 são neutras.
- Após o segundo 10, e durante 5 segundos, efetua simultaneamente interpolações de translação, rotação e escalamento, atingindo um total de vinte unidades de translação no sentido positivo do eixo dos YY, de 360º de rotação em Y, no sentido contrário ao dos ponteiros do relógio, e de aumento da sua dimensão para 120% do original.
- Depois, durante cinco segundos adicionais, a animação procede de modo inverso, isto é, desloca, roda e escala o objeto até este ficar num estado semelhante ao da *keyframe* 1.
- Após a *keyframe* 3, o objeto fica imóvel na posição resultante.

Para esta secção do trabalho prático, pretende-se:

1. Implementar a classe abstrata ***Animation*** como classe base para aplicar animações a um objeto. Esta classe deve conter:
   1.1. Um construtor que receba a informação de base da animação

internamente os valores da transformação***.***

1.3. Um método ***apply()*** para aplicar as alterações de transformação quando adequado.

2. Implementar a classe ***KeyframeAnimation***, que estende ***Animation*** para o suporte à animação por *keyframes* (quadros-chave). Para um nó do grafo da cena, consideram-se duas matrizes de transformação que se multiplicam:
   - *Mn*: a matriz de transformações geométricas resultante da multiplicação da matriz que é recebida do nó ascendente, pela matriz própria do nó, conforme já previsto no trabalho 1.
   - *Ma*: a matriz local de animação, que deve ser calculada a cada instante com base na interpolação das três transformações geométricas de acordo com os dois quadros chave, entre os quais esse instante se encontra.
3. A matriz final de transformações geométricas a aplicar ao nó em questão e a passar aos seus descendentes é portanto: ***M=Mn*Ma*.**
4. Implemente suporte no parser e nas estruturas de dados para processar a descrição XML das animações em ficheiros LSF, de acordo com a extensão proposta no final do enunciado.

2.2. Spritesheets para texto e animações 2D

Há situações em que se torna computacionalmente menos dispendioso utilizar **elementos gráficos 2D (bitmaps)** para gerar efeitos visuais, em vez de utilizar geometria 3D muito detalhada.
Dois exemplos típicos são o **desenho de texto** e a criação de **pequenos efeitos animados**, como, por exemplo, uma explosão.
Criando uma **representação *bitmap* (textura) para cada um dos elementos** - carateres de um tipo de letra, ou *frames* da sequência de animação da explosão (também conhecidas por *sprites*) - é possível mapear essa textura a uma geometria (por exemplo, um retângulo) posteriormente **alternar entre texturas**.

Por questões de eficiência, e dada a reduzida dimensão que cada um destes elementos normalmente tem, **é habitual reunir todos os elementos** (por exemplo, carateres ou sprites) numa única textura 2D, com uma organização pré-definida (tipicamente em **grelha regular** - ver exemplo da Figura 1).
Estas texturas, conhecidas como **spritesheets (ou *atlas*)**, englobam os diferentes elementos a

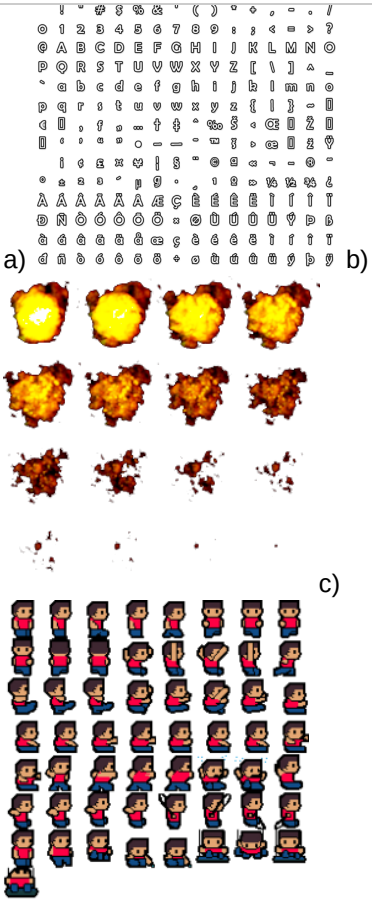


Figura 1: Exemplo de *spritesheets* representando:
(a) texto; (b) uma animação; (c) várias animações (OpenGameArt.org)

A escolha da parte da textura a usar para representar um caracter ou sprite é feita através do cálculo das coordenadas no espaço 2D da textura. Essas coordenadas podem ser calculadas ao definir a geometria, ou através de um *shader* específico, o que permite uma maior flexibilidade e eficiência, particularmente em casos em que há animação envolvida.

No contexto deste trabalho:

1. Implemente uma classe ***MySpriteSheet*** e um *shader* associado à mesma que suporte genericamente uma *spritesheet* com uma grelha de *M* colunas por *N* linhas, e permita ativar a célula de coordenadas *(m, n)* ou a célula na posição sequencial *p* (ver figura 2).

8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Célula selecionada:

Coordenadas (m,n) = (3,2)

Coordenada p = 19

Figura 2: Diagrama de uma *spritesheet* ilustrando uma célula selecionada e duas formas de a referenciar.

Propõe-se que o *shader* calcule as coordenadas ***(s, t)*** de textura a usar no preenchimento de uma geometria com base nos parâmetros das coordenadas *(m, n)* da célula selecionada e o número de células da grelha *(M, N)*.

A classe ***MySpritesheet*** deve contemplar os seguintes métodos:

a. Construtor: ***MySpritesheet(scene, texture, sizeM, SizeN)*** - carrega a textura e respetivas dimensões, bem como o shader associado.
b. ***MySpritesheet.activateCellMN(m, n)*** - "ativa" a célula de coordenadas *(m, n)*. Para isso, deve ativar a textura e o shader, e definir os parâmetros do shader de acordo com as coordenadas de entrada.
c. ***MySpritesheet.activateCellP(p)*** - ativa a célula na posição *p*, assumindo que a numeração das células começa em 0 no canto superior esquerdo da textura e avança da esquerda para a direita, e de cima para baixo (ver figura 2). Este método pode recorrer ao anterior.

2. Implemente uma classe ***MySpriteText*** para renderizar em 3D uma string de texto suportada por *spritesheets*. Esta classe deve receber como argumento a *string* desejada, e renderizar a mesma numa geometria, mapeando os caracteres a partir de uma *spritesheet* específica da aplicação e carregada explicitamente no código (ou seja, não é definida/dependente do ficheiro LSF, mas da aplicação). A *spritesheet* será carregada dentro de uma instância da

(sugestão: ***MyRectangle***, para criar quadrado) onde os caracteres serão mapeados, e inicializa a *spritesheet*.

b. ***MySpriteText****.getCharacterPosition(character)* - função auxiliar que recebe o caracter a ser renderizado, e devolve a posição da *sprite* correspondente na grelha da *spritesheet*.

c. ***MySpriteText****.display()* - função de desenho (a ser chamada durante o desenho do grafo), na qual a geometria criada será desenhada repetidamente para cada caracter. Cada caracter será mapeado na geometria utilizando a função ***MySpritesheet.****activateCellP()*.

d. Adicione suporte na aplicação a um novo tipo de primitiva - ***spritetext*** - e adicione uma ou mais instâncias dessa primitiva no ficheiro de cena (ver extensão ao LSF no final do enunciado)

```
<leaf
type="spritetext"
text="ss" />
```

Nota: A *spritesheet* a ser usada fica ao critério dos grupos, dado que duas *spritesheets* podem ter cada caracter em posições diferentes. A função *getCharacterPosition()* deve ser por isso adequada à *spritesheet* de texto usada.

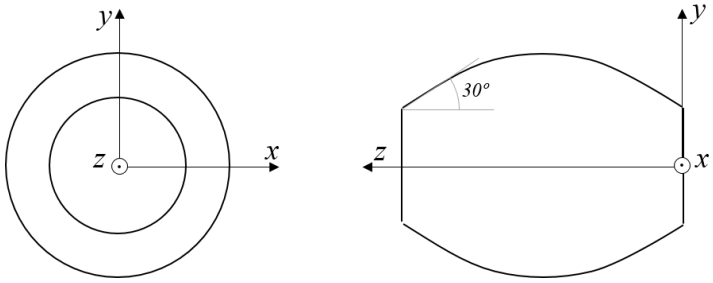
3. Implemente uma classe ***MySpriteAnimation*** para representar uma animação baseada em *sprites*. Dada uma spritesheet, uma célula de início e de fim, e um período de tempo para a animação, a classe deve ir alterando a célula da spritesheet usada em função do tempo para criar uma animação em ciclo (sugestão: usar uma função ***update()***). Ao nível do suporte em LSF, deve ser observado o seguinte (ver detalhes na extensão ao LSF no final do enunciado):
   a. As *spritesheets* para este efeito devem ser identificadas numa nova secção do LSF
   b. Deve ser criado um novo tipo de folha ***spriteanim*** para representar uma animação deste tipo.

```
<leaf type="spriteanim"
ssid="ss" duration="ff"
startCell="ii"
endCell="ii" />
```

2.3. Superfícies 2D/3D

funcionalidades disponibilizadas pela biblioteca WebCGF.

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 LSF.
2. Criar uma nova primitiva *patch* a incluir na linguagem LSF 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 *defbarrel,* a incluir na linguagem LSF, em forma de barril correspondente a uma superfície cilíndrica sem tampas, em que:
    a. Os raios do barril nas duas extremidades é igual;
    b. O raio no centro é indicado por um segundo parâmetro;
    c. São mantidas as definições já conhecidas de *stacks* e de *slices*;
    d. A base do barril encontra-se na origem das coordenadas e o seu eixo principal coincide com o eixo positivo ZZ.



3. Requisitos da cena

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

- A animação por keyframes.
- Objetos de texto.
- Animações baseadas em spritesheets.
- As superfícies 2D e 3D.

4. Avaliação do trabalho

Composição dos Grupos: Os trabalhos devem ser efetuados em grupos de dois estudantes. Preferencialmente, cada grupo deve ser composto pelos mesmos elementos que trabalharam no TP1. Em caso de necessidade de alguma alteração, deve ser discutida com o/a docente a melhor alternativa.

Avaliação do Trabalho de Grupo: O código resultante do trabalho de grupo será apresentado e defendido perante o docente respetivo. O trabalho poderá ser sujeito a uma bateria de

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

Estruturação e Documentação do código	2 valores
Interface, aspeto geral e criatividade	2 valores
Animação por keyframes	6 valores
Spritesheets para texto e animação	6 valores
Superfícies 2D/3D	4 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 pedidos.

De acordo com a formulação constante na ficha de unidade curricular, a avaliação deste trabalho conta para a classificação final com um peso:

80% * 35% = 28%

Planeamento do Trabalho

- Semana 1 (início em 26/10/2020): Animação
- Semana 2 (início em 02/11/2020): Spritesheets
- Semana 3 (início em 09/11/2020): Semana de interrupção
- Semana 4 (início em 16/11/2020): Superfícies 2D/3D

Datas principais

- Data limite de entrega do trabalho completo: 22/11/2020 (via moodle)
- Avaliação dos trabalhos de grupo: aulas práticas, semana de 23/11/2020

5. Extensão à Linguagem LSF

A linguagem LSF encontra-se globalmente definida no enunciado do trabalho prático 1. Nesta secção são apresentadas as extensões ao formato LSF 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 LSF 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:

especificando um eixo
- tt: valor Booleano na forma "0" ou "1"

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

```
<lsf>
...
<!--informacao de animacao -->
<!-- o bloco "animations" é
opcional, mas quando existir -->
<!-- deve ser declarado
imediatamente antes do bloco
 "nodes" -->
<animations>
<!-- O bloco animations pode ser
vazio, isto é, pode -->
<!-- não ser declarada qualquer
      animação -->
      <animation id=“ss” >
            <!-- A animação inicia-se
                  no tempo da primeira
                  keyframe, -->
            <!-- sendo que o objeto
                  em causa deve
                  permanecer -->
            <!-- invisível até esse
                  momento -->

            <!-- Deve existir pelo
      menos 1 elemento keyframe. -->
            <!-- instant e’ o tempo
      expresso em segundos -->
            <!-- desde o inicio da
      animação. -->
            <!-- Os keyframes devem
      ser declarados por -->
            <!-- ordem crescente do
      tempo. -->
               <keyframe instant=“ff”>
                  <!-- translate,
            rotate e scale
            representam as -->
                  <!-- quantidades das
            transformações -->
                  <!--
            correspondentes, medidas
            em relação à -->
                  <!-- situação
            inicial (instante zero) -
            -> -->
                  <!-- Para uma mesma
            keyframe, os elementos --
            >
                  <!-- translate,
            rotate e scale sao
            obrigatorios -->
                  <!-- e fornecidos
            por esta ordem -->
                  <translation x=“ff”
                        y=“ff” z=“ff”
                        />
                  <rotation axis="x"
            angle="ff" />
                  <rotation axis="y"
            angle="ff" />
                  <rotation axis="z"
```

```
      </animation>
</animations>

<!-- spritesheets -->
<!-- o bloco "spritesheets" deve ser
declarado -->
<!-- imediatamente após o bloco
"textures" -->
<spritesheets>
        <!-- Semelhante à declaração
de textura,  →
      <!-- mas inclui a definição do
nº de colunas e linhas -->
        <spritesheet id=”ss”
path=”ss” sizeM=”ii” sizeN=”ii” />
        ...
</spritesheets>

<nodes>
        <node ...>
           ...
          <!-- o elemento
                    "animationref"
                    e' opcional;
                    deve -->
          <!-- declarar-se
                    imediatamente
                    após as
                    transformacoes
                    -->
          <!-- geométricas do
                    componente -->
          <animationref id=“ss” />
            ...
                      <descendants>
              <!-- Nova primitiva:
                    texto baseado
                    em spritesheet
                    -->
              <leaf
              type=”spritetext”
              text=”ss” />

              <!-- Nova primitiva:
                    animação
                    baseada em
                    spritesheet →
              <!-- identifica a
                    spritesheet a
                    usar, a célula
                    inicial e final
                    da animação, e
                    a duração -->
              <leaf
              type=”spriteanim”
              ssid=”ss”
              startCell=”ii”
              endCell=”ii”
              duration=”ff” />


              <!-- Nova primitiva:
                    plano, gerado
                    por NURBS -->
              <!-- ex: <plane
                    npartsU=“5”
                    npartsU=“8” />
              <!-- um plano de
                    dimensões 1 x 1
                    unidades
                    assente -->
              <!-- em XZ, centrado
                    na origem -->
```

```
                                    por oito partes
                                    -->
                               <leaf type="plane"
                                    npartsU="ii"
                                    npartsV="ii" />

                               <!-- 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 -->
                               <leaf type="patch"
                                    npointsU="ii" npointsV="ii"
                                    npartsU="ii"
                                    npartsV="ii" >
                                   <controlpoint
                                    x="ff" y="ff"
                                    z="ff" />
                                   ...
                               </leaf>

                               <!-- – Nova
                                    primitiva:
                                    forma de barril
                                    baseado em
                                    NURBS -->
                               <!-- parâmetros
                                    semelhantes ao
                                    cilindro
                                    original, mas
                                    sem tampas -->
                               <!-- base representa
                                    o raio na base
                                    e topo do
                                    cilindro -->
                               <!-- middle
                                    representa o
                                    raio no centro
                                    do objeto -->
                               <leaf
                                                type="defbarrel"
                                                base="ff" middle="ff" height="ff
                                                slices="ii" stacks="ii" />

                          </descendants>
                     </node>
               </nodes>
               </lsf>
```