Laboratório de Aplicações com Interface Gráfica
Aulas Práticas

MIEIC – 2019/2020

Trabalho Prático 2
Aperfeiçoamento de técnicas utilizadas em Computação Gráfica

1. Introdução

O objetivo deste trabalho é explorar técnicas gráficas como NURBS (Non-uniform Rational B-spline), animação por keyframes e o uso de técnicas de *Render-to-texture*, com uso de *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 LXS 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. Cada transformação geométrica é medida em relação à situação inicial do objeto.
Considerando pares consecutivos de keyframes, a animação deve interpolar os valores relativos a cada uma das 3 transformações geométricas mencionadas, de modo a produzir uma transição suave do objecto entre o estado inicial e o estado final, representados pelo par de keyframes.
NOTA 1: a posição inicial de um objeto é a que resulta das transformações geométricas recebidas do seu ascendente e do próprio nó e corresponde a uma hipotética chave Nº zero a que corresponde T=(0, 0, 0); R=(0, 0, 0); S=(1, 1, 1).
NOTA 2: após a última chave de uma animação, o objeto permanece imóvel.

Por exemplo, considere-se a seguinte definição conceptual de uma animação:

keyframe 1:
Instante: 10
Translação: (0.0, 0.0, 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, inicia a sua animação no segundo 0 (efetivamente 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 objecto até este ficar num estado semelhante ao da keyframe 1. Após a keyframe 3, o objeto fica imóvel na posição resultante.

Pretende-se:

1. Implementar a classe abstrata *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. Implementar a classe *KeyframeAnimation*, que estende *Animation* para o suporte à animação por keyframes (quadros-chave). Para um nó do *scenegraph* 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 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 de Transformações Geométricas a aplicar ao nó em questão e a passar aos seus descendentes é portanto: M=Mn*Ma.

3.1. Superfícies 2D/3D

As superfícies vão ser modeladas através da representação de NURBS (https://en.wikipedia.org /wiki/Non-uniform_rational_B-spline).

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 LXS.
2. Criar uma nova primitiva *patch* a incluir na linguagem LXS 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 LXS que represente uma superfície cilíndrica (sem tampas) respeitando as definições já conhecidas de *stacks* e de *slices*.

3.1. Render-to-Texture

Pretende-se criar um segundo ecrã que apresenta a cena a partir de um ângulo diferente, como um ecrã de uma câmara de segurança, utilizando técnicas de render to texture (RTT). Este objeto não será implementado com a linguagem LXS, e requer criar uma nova classe, e fazer modificações em ***XMLscene***.

Criação da classe *MySecurityCamera*

Crie uma nova classe ***MySecurityCamera*** baseada na classe CGFobject. Nessa classe, defina um objeto da classe ***MyRectangle***.

Crie um vertex e fragment shader para ser aplicado no objeto ***MyRectangle*** na função *display()* da classe ***MySecurityCamera***. O vertex shader deverá definir os vértices sem aplicar as matrizes de cena e de projeção. O fragment shader aplica a textura criada a partir da renderização da cena.

Os parâmetros de ***MyRectangle*** (*x1, x2, y1, y2*) deverão ser definidos de forma a que, aplicando os shaders criados, o rectângulo ocupe 25% da largura e altura do ecrã, e que seja colocado no canto inferior direito, como demonstra a figura 1.

NOTA: Consulte o ficheiro sobre shaders (Introduction to shaders using GLSL ES).

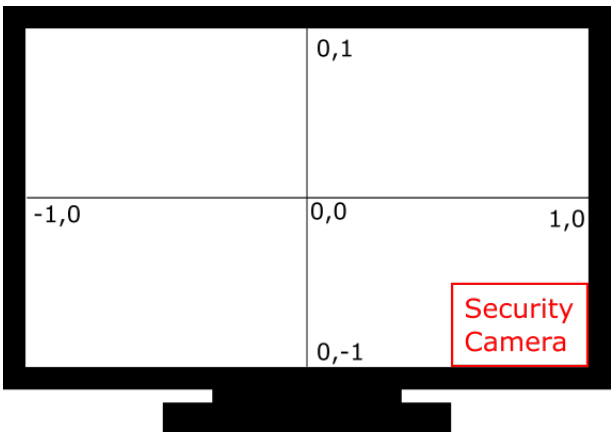

Figura 1: Coordenadas de ecrã e posicionamento de ***MySecurityCamera***.

Modificações na classe *XMLscene*

Para criar uma segunda imagem da cena, é necessário renderizar a cena duas vezes, uma para a textura, e outra para a apresentação normal da cena.

- Na ***XMLscene***, crie uma textura RTT da classe ***CGFtextureRTT***, que recebe como argumentos uma referência a ***XMLscene***, e largura e altura da textura (que serão iguais às dimensões do canvas).
- A função atual *display()* será renomeada para *render()*, e uma nova função de *display()* será criada. Nesta nova função, faça duas chamadas à função *render()*, sendo que antes de cada chamada deve associar ou desassociar o framebuffer à textura, respetivamente. Isto é, em pseudo-código:

```
CGFtextureRTT.attachToFrameBuffer()
XMLscene.render()
CGFtextureRTT.detachFromFrameBuffer()
XMLscene.render()
```

- A câmara a usar em cada chamada do *render()* será diferente. Deverá então criar uma nova câmara que representa a câmara de segurança, e deverá passar a câmara a ser usada no *render()* como argumento.
- Na função *display()* classe ***MySecurityCamera***, aplique a textura RTT criada no retângulo.
- Crie um objeto da classe ***MySecurityCamera*** na cena, e faça chamada à função *display()* do mesmo depois das chamadas a *render()***.**
- **Importante:** Para desenhar objetos por cima da imagem da cena (UI), como é o caso da câmara de segurança, é necessário desativar o teste de profundidade (depth test). Para tal, faça disable antes e enable do mesmo depois de chamar a função *display()*:

```
XMLscene.enable(this.gl.DEPTH_TEST)
```

Efeitos nos shaders

● Aplique gradiente radial no fragment shader de forma a que a cor escureça do centro para os cantos da imagem.
● A imagem final da câmara de segurança deverá ter linhas horizontais brancas que se sobrepõem à imagem original (por adição de cor). Use as coordenadas de textura para calcular a intensidade da linha num dado pixel. As linhas deverão ser animadas, movendo-se de baixo para cima, usando um valor de tempo.

Modificações na interface

● A interface deve agora permitir definir duas câmaras: a normal (trabalho 1) e a nova câmara. No arranque do programa, a câmara de default deve ser usada nas duas situações.

3. Requisitos da cena

Deve ser criada uma cena que utilize as funcionalidades referidas acima, nomeadamente:
● A animação por keyframes.
● As superfícies 2D e 3D.
● A técnica Render-to-texture.

4. 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 27 de Novembro de 2019, 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% * 40% * 50% = 16%.

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	2 valores
Interface, aspeto geral e criatividade	2 valores
Animação por keyframes	6 valores
Superfícies 2D/3D	4 valores

Render-to-texture	6 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 21/10/2019): Animação
- Semana 2 (início em 28/10/2019): Semana de interrupção
- Semana 3 (início em 04/11/2019): Superfícies 2D/3D
- Semana 4 (início em 11/11/2019): Técnica Render-to-texture
- Semana 5 (início em 18/11/2019): Finalizações

Datas principais

- Data limite de entrega do trabalho completo: 24/11/2019 (via moodle)
- Avaliação dos trabalhos de grupo: aulas práticas, semana de 25/11/2019
- Prova de avaliação individual: 04/12/2019, 14:30-17:30

5. Extensão à Linguagem LXS

A linguagem LXS encontra-se globalmente definida no enunciado do trabalho prático 1. Nesta secção são apresentadas as extensões ao formato LXS 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 LXS 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 LXS. As tags / atributos acrescentados encontram-se escritos à cor vermelha. À cor preta encontram-se elementos definidos na versão original da linguagem LXS, usados para melhor contextualizar as alterações. Os comentários estão escritos à cor verde.

```
<lxs>
...
<!--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 -->
      <animation id=“ss” >
            <!-- Deve existir pelo menos 1 elemento keyframe. -->
            <!-- instant e’ o tempo expresso em segundos -->
            <!-- desde o inicio da animação. -->
               <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 -->
                <!-- a instrução rotação corresponde a:       -->
                <!--    <rotate axis="x" angle="angle_x" />   -->
                <!--    <rotate axis="y" angle="angle_y" />   -->
                <!--    <rotate axis="z" angle="angle_z" />   -->

                <translate x="ff" y="ff" z="ff" />
                <rotate angle_x="ff" angle_y="ff" angle_z="ff" />
                <scale x="ff" y="ff" z="ff" />
            </keyframe>
        </animation>
</animations>


<primitives>
<primitive id="ss">

        <!-- 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 -->
        <!-- e com a face visível apontando para +Y -->
        <!-- com divisão em cinco partes por oito partes -->
        <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 -->
        <patch npointsU="ii" npointsV="ii" npartsU="ii" npartsV="ii" >
            <controlpoint xx="ff" yy="ff" zz="ff" />
            ...
        </patch>

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


<components>
        <component ...>
        ...
        <!-- o elemento "animationref" e' opcional; deve -->
        <!-- declarar-se imediatamente após as transformacoes -->
        <!-- geométricas do componente -->
        <animationref id="ss" />
        ...
        </component>
</components>
</lxs>
```