Sistemas Gráficos Interativos
Aulas Práticas

M.EIC – 2023/2024

Trabalho Prático 2
Desenv. de uma aplicação gráfica 3D

1. Introdução

Pretende-se, com este trabalho, constituir uma aplicação dotada de um pequeno motor gráfico 3D. A aplicação deve ser capaz de produzir imagens de qualquer cena, sendo esta especificada através de um ficheiro de texto a ser lido pela aplicação (ver secção 4 deste documento).

Os ficheiros de texto devem respeitar um formato próprio, a que chamaremos *YASF - Yet Another Scene Format*, especificada na secção 5 deste documento, que obedece a um conceito muito vulgar em computação gráfica: o Grafo de Cena (*Scene Graph*). A sintaxe obedece ao formato de tags do XML.

A aplicação deve efetuar a leitura de um ficheiro ".xml" que descreve a cena, construindo, simultaneamente, a estrutura de dados correspondente ao grafo de cena. Só depois deve efetuar a afixação da imagem respetiva. A cena deve conter fontes de luz que devem ser ativadas (on/off) de acordo com o especificado no ficheiro .xml, mas que podem posteriormente ver o seu estado alterado por meio de controlos na área de interação da aplicação. Os pontos de vista de observação assim como o ponto de vista inicial devem ser especificados no ficheiro .xml. O ponto de vista deve poder ser alterado por meio de um controlo na área de interação da aplicação.
Adicionalmente, na visualização da cena, deve ser possível alternar entre o modo poligonal "fill" e "wireframe" através de um controlo também localizado na área de interação.

2. Componentes do trabalho

As componentes do trabalho, a realizar de forma incremental ao longo de quatro semanas, são descritas no plano de trabalho sugerido (secção 3).

De forma a facilitar a implementação, é disponibilizado o código de um *parser* que Implementa a leitura de ficheiros .xml, colocando a respetiva informação em estruturas simples de dados, em memória central.

Os desenvolvimentos a efetuar no âmbito do trabalho incluem a transformação dessa informação num "scene graph", de acordo com o especificado nas secções 4 e 5 deste documento. Incluem de igual modo a implementação de um conjunto de funcionalidades que efetue a visita do grafo de cena e construa a imagem correspondente usando a tecnologia ThreeJS.

3. Notas sobre a avaliação do trabalho

Composição dos Grupos: Os trabalhos devem ser realizados em trabalho de grupo de dois estudantes. 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: O código resultante do trabalho de grupo será apresentado e defendido em sala de aula, perante o docente respetivo. Ambos os estudantes do grupo devem estar presentes na sessão de avaliação (em caso de impossibilidade, devem acordar previamente uma alternativa com o docente das aulas práticas). A classificação atribuída aos dois estudantes pode ser diversa.

Assim:
Será avaliada a forma de escrita de código
Será verificado o cumprimento das tarefas enunciadas acima
Será avaliada a inclusão dos vários objetos e funcionalidades enunciadas para a cena 3D

Os critérios a usar para efeitos de Avaliação do Trabalho de Grupo são os seguintes:

TBD	TBD
TBD	TBD

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%

Datas Principais:

- Início: 16/10/2023
- Entrega: 20/11/2023
- Avaliação nas aulas: 28 - 30/11/2023

Plano de trabalhos sugerido:

- **Semana de 16/10**: [Aula 1] Scene Graph - Implementação
  - Definição de globais (background, sombras, câmaras)
  - Caracterização de materiais/texturas
  - Objetos: simples, hard-coded.
- **Semana de 23/10**: [Aula 2] Scene Graph - Cont. da Implementação
  - Definição de primitivas (luzes,geometrias)
  - Herança de propriedades (mater./text., transf. geom.)
  - Visita Recursiva a nós do grafo
  - Modelação de uma cena 3D em xml/YASF (*)

  Semana de 30/10: [Sem Aula]
  - Semana da FEUP
- **Semana de 6/10:** [Aula 3] Funcionalidades adicionais (1)
  - LOD
  - Text. avançadas (video-tex, Bump-Map, Mip-Map)
  - Utilização especial de texturas
- **Semana de 13/11:** [Aula 4] Funcionalidades adicionais (2)
  - Buffer Geometry
  - Shaders
  - Sistemas de Partículas

(*) Neste contexto, far-se-á a preparação de uma cena (livre) especificada em ficheiro .xml, de acordo com as instruções das secções seguintes do presente documento. Todos os ficheiros serão posteriormente divulgados e partilhados, constituindo-se assim um acervo de cenas de teste.

4. Grafo de Cena

Com o conteúdo desta secção pretende-se transmitir o que se entende por *scene graph* em termos gerais. Podem existir algumas diferenças no que respeita à sua implementação neste trabalho, em ThreeJS.

Um grafo de cena é uma estrutura que especifica, de forma hierárquica, como se de uma árvore se tratasse, uma cena 3D. Cada nó do grafo representa um Objeto que, obrigatoriamente, é composto por Primitivas e/ou por Objetos Descendentes:

- Primitiva: corresponde a uma folha da árvore e descreve um objeto geométrico (polígono, retângulo, cilindro...) diretamente endereçável em ThreeJS.
- Descendente: ligação para um objeto-filho que, nesse caso, recebe por herança algumas propriedades.

Um nó pode ser dotado de propriedades locais de aspeto (material/textura, etc.) ou outras, assim como de um conjunto de transformações geométricas locais. Em ambos os casos aplicam-se regras de herança:

- Um nó recebe do seu antecessor um conjunto de propriedades de aspeto que funcionam para si próprio como valores por omissão (*default values*). O nó pode no entanto redefinir alguns desses valores, prevalecendo o novo valor, quer para si próprio, quer para os seus descendentes se os tiver. No nó raiz, os valores por omissão são os definidos pela própria aplicação e que deverão ser especificados.
- Um nó também recebe do seu antecessor uma matriz de transformações geométricas Ma. Por outro lado, pode possuir as suas próprias transformações geométricas locais, representadas por uma matriz única Mp. A matriz a aplicar ao objeto, assim como a passar aos seus eventuais descendentes, é calculada pela expressão M=Ma*Mp .

Todo e qualquer nó deve ter um identificador alfanumérico único e pode ser referenciado por mais do que um nó ascendente. Por exemplo, um nó pode representar o modelo de uma roda de um automóvel e, por isso, ser referenciado quatro vezes diferentes.

Qualquer nó pode ser referenciado previamente à sua declaração, isto é, a ordem pela qual os nós são declarados é irrelevante.

À partida, não se prevê que as câmaras possam ser associadas a nós, pelos que são independentes do grafo de cena. No entanto, as fontes de luz podem ser integradas em nós.

A figura 1 apresenta um exemplo de um grafo de cena. Os círculos (nós intermédios) representam objetos compostos por primitivas, fontes de luz e/ou outros nós; os quadrados (nós folha) representam primitivas.

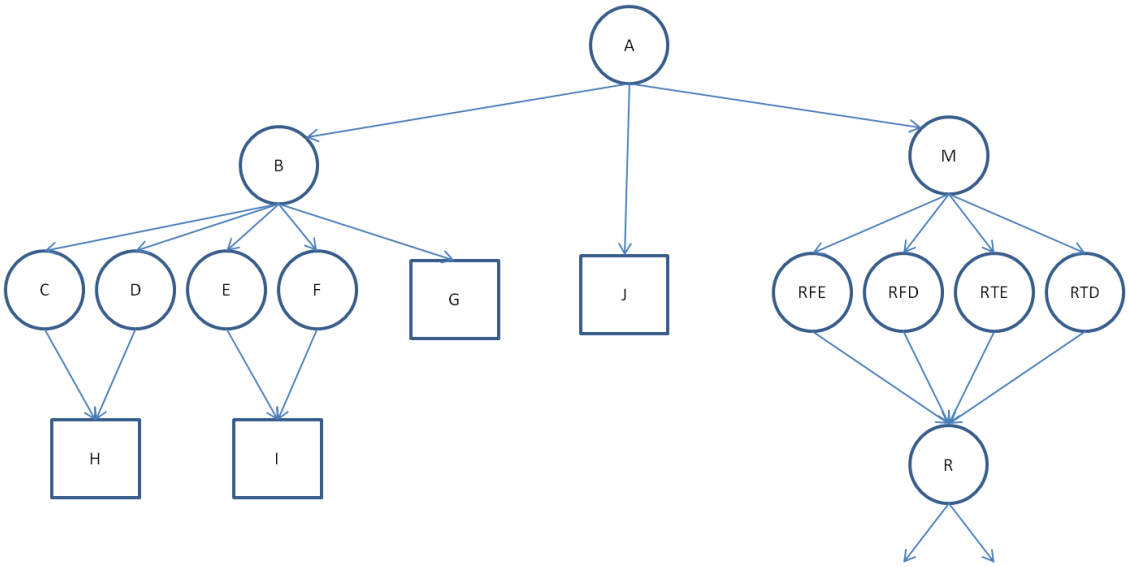

Figura 1 - Exemplo de um grafo de cena

Note-se que o número de descendentes diretos de um nó é indeterminado. O caso de um nó intermédio com um só descendente destina-se, entre outras funções, a fazer uma instanciação. Veja-se por exemplo o caso da figura 1: o objeto "R" (Roda) tem a sua subárvore (não representada para não sobrecarregar o desenho) e tem as suas transformações geométricas particulares. No entanto, para que as quatro rodas tenham distintas posições no espaço, é necessário que cada uma corresponda a uma instância da roda base com transformações geométricas diferentes das restantes. Assim, são criados os nós intermédios de instanciação "RFE" , "RFD" , "RTE" e "RTD" ; os quatro instanciam o nó "R" mas possuem transformações geométricas diferentes.

5. Linguagem YASF

A linguagem YASF constitui uma forma de especificar cenas 3D de uma forma muito simples e fácil de interpretar. Um documento em linguagem YASF pode ser escrito em qualquer editor de texto e obedece a regras de XML, baseadas em tags.
Ao ser lido e interpretado por uma aplicação gráfica, um ficheiro em linguagem YASF 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.

Cada comando representa-se por uma ou mais tags, contendo os parâmetros respetivos (se existirem). Um grupo de caracteres, ou mesmo linhas, envolvido pelos delimitadores "<!-- " e " --> " é considerado um comentário. O texto de comentário deve ser ladeado por espaços que o separam dos dois delimitadores.
NOTA: Todos os identificadores (de tags, atributos...) devem ser escritos exclusivamente com letras minúsculas.

Um documento YASF divide-se em blocos, cada um iniciando-se com um termo identificador de bloco, implementado em XML na forma de uma tag . A referência a uma tag inicia-se com um identificador alfanumérico entre os dois caracteres " < > " (por exemplo <lighting>) e terminando com o mesmo identificador antecedido de uma barra de divisão (no mesmo exemplo, </lighting>); entre as duas ocorrências, descreve-se o conteúdo do tag . A sequência de blocos principais é a seguinte:

```
globals:      global settings
cameras:      specification of cameras/viewpoints
textures:     identification of textures and their files
materials:    specification of material characteristics
                        (may include reference to textures)
graph:        list of graph nodes; each node includes:
                        geom. transf., material, list of descendants
```

Vários dos blocos anteriores contêm definições de entidades de diferentes tipos (várias luzes, várias aparências, etc...). Cada uma dessas entidades contém um identificador do tipo *string*. Cada identificador deve ser único dentro de cada bloco (por exemplo, não podem existir duas fontes de luz com o mesmo identificador).

Na descrição que se segue da sintaxe a seguir, os símbolos utilizados têm o seguinte significado:

```
    ii: integer value
    ff: floating-point value
    ss: string
    ee: character "x" or "y" or "z", specifying an axis
    bb: Boolean value in the form "true" or "false"
```

Segue-se uma listagem representativa da sintaxe pretendida.

```
<!-- Comments should have spaces at the beginning and end, to -->
<!--     separate from hyphens -->
<!-- Special characters (e.g. accents) should not be used -->
<!-- All tags and attributes are mandatory, except where it is explicitly -->
<!--     stated otherwise -->

<yaf>

    <!-- global configuration block: -->

    <globals background="ff ff ff ff"
             ambient="ff ff ff ff" />                <!-- mandatory -->
    <fog color="ff ff ff ff" near="ff" far="ff" />    <!-- optional  -->


    <!-- CAMERAS BLOCK -->

    <!-- the "initial" attribute identifies the -->
    <!-- initial camera/viewpoint -->

    <cameras initial="ss" >

        <!-- there must be at least one camera of the following types -->

        <orthogonal id="ss" near="ff" far="ff"
                location="ff ff ff" target="ff ff ff"
                left="ff" right="ff" bottom="ff" top="ff" />
        <perspective id="ss" angle="ff" near="ff" far="ff"
                location="ff ff ff" target="ff ff ff" />

    </cameras>


    <!-- TEXTURES BLOCK -->

    <!-- this block must exist but may be empty if no  -->
    <!-- textures are to be used -->

    <textures>

        <!-- there can be zero or more such lines: -->
        <texture id="ss" filepath="ss" />

    </textures>



    <!-- MATERIALS BLOCK -->

    <materials>

        <!-- there must be one or more blocks "material" -->
        <!-- IMPORTANT: may or may not include a texture -->
        <!--     if it does, it must specify -->
```

```
    <!--     textureref, texlength_s, textlength_t, textrect -->
    <!--     NOTE: see 'Endnotes' in the statement about the -->
    <!--       parameters texlength_*...) -->

    <material id="ss"
        color="ff ff ff ff"
        specular="ff ff ff ff"
        shininess="ff"
        emissive="ff ff ff ff"
        wireframe="bb"              <!-- optional, default=false -->
        shading="ss"                <!-- "none","flat","smooth";
                                          optional, default="smooth" -->
        textureref="ss"             <!-- optional, default=null -->
        texlength_s="ff"            <!-- optional, default=1 -->
        texlength_t="ff"            <!-- optional, default=1 -—>
        twosided="bb"               <!-- optional, default=false -—>
     />
</materials>



<!-- GRAPH BLOCK -->

<!-- "rootid" identifies the root node -->

<graph rootid="ss">

    <!-- there must be one or more "node" blocks composed of: -->
    <!--     node identification (required) -->
    <!--     local geometric transformations (required) -->
    <!--     material (optional) -->
    <!--     descendant (required) -->

    <node id="ss" castshadows="bb" receiveshadows="bb">

        <!-- Property "castshadows" is, by default, "false" -->
        <!-- If, in any node, the value is declared as "true",  -->
        <!-- then all its descendants (sons, grand-sons, etc)  -->
        <!-- assume that value and cannot change it. -->
        <!-- The same is valid for the property "receiveshadows"  -->

        <!-- One node is characterized by 3 blocks: -->
        <!--        transforms (optional),          -->
        <!--        material (optional),            -->
        <!--        children (mandatory)            -->

        <!--    children is the list of descendents of the node -->
        <!--        see below  -->


        <!-- BLOCK "TRANSFORMS" -->
        <!-- not required if no transformations are needed -->

        <transforms>

            <!-- may include the following -->
            <!-- geometric transformations, in this sequence -->

            <translate value3= "ff ff ff" />       <!-- optional -->
            <rotate value3= "ff ff ff" />          <!-- optional -->
            <scale value3= "ff ff ff" />           <!-- optional -->

        </transforms>


        <!-- BLOCK "MATERIAL" -->

        <!-- declaration of the material to be used by -->
```

```
<!--     this node and its descendents (may be omitted): -->
<!--     a) if declared, it overrides the material that -->
<!--           is received from its "parent" -->
<!--     b) if omitted, the node keeps (inherits) the  -->
<!--           material received from the "parent" -->

<materialref id="ss" />          <!-- optional -->



<!-- BLOCK "CHILDREN" ----- mandatory -->

<children>

    <!-- declare one or more descendents (children) -->
    <!-- a descendant can be:
    <!--     a local primitive (see below) -->
    <!--     a local light source (see below) -->
    <!--     a reference to other node -->


    <!-- CHILDREN = PRIMITIVES -->
    <!-- descendants of type "primitive" are: -->
    <!--     rectangle, triangle, cylinder, sphere... -->
    <!--     the cylinder must have tops or lids -->
    <!-- One (and only one) primitive must be involved in: -->
    <!--    <primitive> ... </primitive>                    -->

    <primitive>
        <!-- Possible primitives are: -->
        <rectangle xy1="ff ff" xy2="ff ff"
            parts_x="ii" parts_y="ii" /> <!-- opt. def. = 1 -->
        <triangle xyz1="ff ff ff"
                  xyz2="ff ff ff"
                  xyz3="ff ff ff" />
        <box xyz1="ff ff ff" xyz2="ff ff ff"
            parts_x="ii" parts_y="ii" parts_z="ii"  />
             <!-- opt. def. = 1 -->
        <cylinder base="ff" top="ff" height="ff"
               slices="ii" stacks="ii"
               capsclose="bb"   <!-- opt., default = false -->
               thetaStart="ff"  <!-- optional -->
               thetaLength="ff" <!-- optional -->
        />
        <sphere
               radius="ff"
               slices="ii"
               stacks="ii"
               thetastart="ff"   <!-- optional -->
               thetalength="ff"  <!-- optional -->
               phistart="ff"     <!-- optional -->
               philength="ff"    <!-- optional -->
        />

        <!-- primitiva nurbs -->
       <!-- – parâmetros: -->
       <!-- – degree_u: degree of the curve in the domain U -->
       <!-- – degree_v: degree of the curve in the domain V -->
       <!-- – parts_u: division in parts in domain U used -->
       <!--       in the surface drawing -->
       <!-- – parts_v: division in parts in domain V used -->
       <!--       in the surface drawing -->
       <nurbs degree_u="ii" degree_v="ii"
              parts_u="ii" parts_v="ii" >

           <!-- the number of control points in -->
          <!--     the nurbs primitive is: -->
          <!-- (degree_u+1) * (degree_v+1) -->
           <!-- next line to be repeated as needed -->
           <controlpoint xx="ff" yy="ff" zz="ff" />
```

```
                </nurbs>
            </primitive>


            <!-- CHILDREN = LIGHT SOURCES -->
            <!-- one or more lights can be declared: -->
            <!--    directionallight, pointlight or spotlight -->
            <!--    the identifiers "id" cannot be repeated -->

            <pointlight id="ss"
                enabled="bb"           <!-- optional, default = true -->
                color="ff ff ff ff"
                intensity="ff"         <!-- optional, default = 1 -->
                distance="ff"          <!-- optional, default = 1000 -->
                decay="ff"             <!-- optional, default = 2 -->
                position="ff ff ff"
                castshadow="bb"        <!-- optional, default = false -->
                shadowfar="ff"         <!-- opt., default = 500.0 -->
                shadowmapsize="ii"     <!-- opt., default = 512 -->
            />

            <spotlight id="ss"
                enabled="bb"           <!-- optional, default = true -->
                color="ff ff ff ff"
                intensity="ff"         <!-- optional, default = 1 -->
                distance="ff"          <!-- optional, default = 1000 -->
                angle="ff"
                decay="ff"             <!-- optional, default = 2 -->
                penumbra="ff"          <!-- opt., default = 1 -->
                position="ff ff ff"
                target="ff ff ff"
                castshadow="bb"        <!-- opt., default = false -->
                shadowfar="ff"         <!-- opt., default = 500.0 -->
                shadowmapsize="ii"     <!-- opt., default = 512 -->
            />

            <directionallight id="ss"
                enabled="bb"           <!-- optional, default = true -->
                color="ff ff ff ff"
                intensity="ff"         <!-- optional, default = 1 -->
                position="ff ff ff"
                castshadow="bb"        <!-- opt., default = false -->
                shadowleft="ff"        <!-- opt., default = -5 -->
                shadowright="ff"       <!-- opt., default =  5-->
                shadowbottom="ff"      <!-- opt., default = -5-->
                shadowtop="ff"         <!-- opt., default =  5 -->
                shadowfar="ff"         <!-- opt., default = 500.0 -->
                shadowmapsize="ii"     <!-- opt., default = 512 -->

            />

            <!-- CHILDREN = OTHER NODES -->
            <!-- descendant of type "node" = refer. to other node -->
            <!-- REMINDER: a node referenced here can be -->
            <!-- declared further down (the order of declaration -->
            <!--     is irrelevant  --> 2

            <noderef id="ss" />

        </children>
    </node>
</graph>
</yaf>
```