

**Laboratório de Aplicações com Interface  
Gráfica**  
Aulas Práticas

MIEIC – 2020/2021

**Trabalho Prático 1**  
**Desenvolvimento de um Motor**  
**Gráfico em WebGL**

rev 2020.09.30.1122

## 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 devidamente especificada através de um ficheiro de texto a ser lido pela aplicação.

O ficheiro de texto deve respeitar um esquema próprio, a que chamaremos LSF – LAIG Scenegraph Format - especificado na secção 3 deste documento, e que obedece a um conceito muito vulgar em computação gráfica: o Grafo de Cena (Scene Graph, secção 2). A sintaxe de base obedece ao formato de tags do XML (*Extensible Markup Language*).

A aplicação deve, através de um *parser*, efetuar a leitura de um documento LSF (cuja extensão do ficheiro deve ser sempre .xml) que descreve a cena, construindo simultaneamente a estrutura de dados correspondente ao grafo de cena. Só depois deve realizar a geração da imagem respectiva. As fontes de luz devem iniciar-se (on/off) de acordo com a especificação do esquema LSF e devem poder ser alteradas por meio de controlos na interface gráfica 2D. A seleção da vista ativa é também feita através de uma lista disponível na interface gráfica 2D.

### 1.1. Componentes do trabalho

#### 1. Preparação de uma cena de teste

Preparar uma cena especificada no esquema LSF e num ficheiro com extensão .xml, de acordo com

## 2. Construção do parser e estrutura de dados

- a. Implementar a componente de leitura e interpretação do ficheiro .XML (parsing), utilizando para esse efeito a biblioteca [WebCGF](#) (um exemplo é fornecido no Moodle, com a componente de leitura parcialmente implementada, juntamente com este enunciado).
- b. Implementar uma estrutura de dados capaz de armazenar o grafo de cena apresentado na secção 2 deste documento.

## 3. Desenho da cena

Implementar um conjunto de funcionalidades que efetue a visita da estrutura de dados (e, portanto, do grafo) e que, com recurso à biblioteca WebCGF, construa a imagem correspondente usando a tecnologia WebGL.

O trabalho deve ser desenvolvido de forma incremental:

1. Versões iniciais, básicas, do parser, da estrutura interna de dados e das rotinas de visualização que permitam respetivamente ler, armazenar e visualizar primitivas e transformações simples, ignorando inicialmente luzes, materiais e texturas.
2. Versões progressivamente estendidas do parser, da estrutura de dados e visualizador com as restantes funcionalidades enunciadas.

## 1.2. Notas sobre a avaliação do trabalho

**Composição dos Grupos:** Os trabalhos devem ser efetuados em grupos 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 perante o docente respetivo. O trabalho poderá ser sujeito a uma bateria de testes, com origem em cenas representadas por ficheiros .xml, sendo a classificação atribuída dependente da adequação da resposta dada.

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
Primitivas e Geometria	4 valores
Transformações Geométricas - descrição e herança	4 valores
Materiais - descrição e herança	3.5 valores
Texturas - descrição, dimensões e herança	3.5 valores

trabalho.

**Avaliação Global do Trabalho:** Igual à avaliação do trabalho de grupo.

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

$$80\% * 25\% = 20\%$$

#### Datas Principais:

- Data limite de entrega do ficheiro XML no esquema LSF: 04/10/2020
- Data limite de entrega do trabalho completo: 25/10/2020
- Avaliação do trabalho: semana com início em 26/10/2020
- Prova de avaliação individual: (não aplicável)

#### Plano de trabalhos sugerido:

- **Semana de 21/09:** Início do trabalho; definição da cena de teste e descrição inicial em ficheiro XML; preparação do ambiente de desenvolvimento.
- **Semana de 28/09:** Criação e carregamento da estrutura de dados. Submissão do ficheiro XML.
- **Semana de 05/10:** Travessia do grafo de cena com desenho de primitivas base e transformações.
- **Semana de 12/10:** Aplicação de materiais e texturas; garantia de herança; iluminação
- **Semana de 19/10:** Verificações finais (entrega no final da semana)

### 1.3. Notas sobre o repositório do trabalho prático

Para o desenvolvimento e controlo de versões dos trabalhos práticos, deverão utilizar o sistema de **Gitlab da FEUP (git.fe.up.pt)**. Quando os grupos de trabalho estiverem definidos, os estudantes serão convidados para o repositório do seu grupo (p.ex: LAIG\_T01\_G01 será o repositório do grupo 1 da turma 1). **Nota:** Para mais informação, verifique o documento sobre o funcionamento das aulas práticas (disponibilizado no Moodle):

<https://drive.google.com/file/d/1Ov06x30DTwZGOWbbGGeoaoWZORr3NIhU/view?usp=sharing>

O repositório terá inicialmente a pasta da biblioteca de WebCGF (*lib*) e uma pasta para cada trabalho prático da unidade curricular (*TP1*, *TP2*, *TP3*). A pasta *TP1*, correspondente a este trabalho prático, terá o código-base necessário para iniciarem o desenvolvimento.

A gestão do repositório fica ao critério de cada grupo, mas sugerimos que tenham o *branch* principal (*master*) atualizado com a última versão funcional do trabalho, e que usem *branches* para desenvolvimento.

Para melhor visualização e acompanhamento semanal do progresso de cada grupo, deverão definir **tags** no repositório a cada semana. Estas *tags* devem identificar o fim de cada fase do plano de trabalhos sugerido

- **Semana de 05/10:** "tp1-v0.2"
- **Semana de 12/10:** "tp1-v0.3"
- **Semana de 19/10 - Versão de Entrega:** "tp1-v1.0"

Na mensagem de *commit* associada a cada tag deve ser indicada também a identificação da versão, e a lista de funcionalidades implementada até ao momento. Devem seguir o modelo seguinte:

*"Versão: tp1-v0.1*

*Data:*

*Funcionalidades implementadas até ao momento:*

*- ...*  
*- ..."*

Para uma explicação mais detalhada sobre *tags*, visitem o artigo de Git: <https://git-scm.com/book/en/v2/Git-Basics-Tagging>

## 2. Grafo de Cena

Um grafo de cena é um grafo acíclico que pode ser visitado de forma semelhante a uma árvore que especifica, de forma hierárquica, uma cena 3D. Cada nó corresponde a um objecto, simples (folha) ou composto (nó intermédio).

Todo e qualquer nó deve ter um identificador do tipo *string*. Cada nó só pode ser instanciado uma vez, mas pode ser referenciado por vários nós seus ascendentes; por exemplo, um nó pode representar a roda de um automóvel e, por isso, ser referenciado quatro vezes diferentes na definição do automóvel.

### 2.1. Nós tipo Folha

Cada folha refere-se exclusivamente a um objeto simples ou primitiva, sem descendentes, cuja geometria é interpretada e imediatamente desenhada. Deve por isso conter todos os parâmetros exigidos pela instrução respetiva.

Um nó-folha não contém transformações geométricas nem propriedades de aspeto (materiais, etc.).

### 2.2. Nós Intermédios

Subindo na hierarquia, um nó intermédio da árvore possui um ou mais nós como seus descendentes diretos, sendo que estes poderão ser folhas ou outros nós intermédios. Está reservada aos nós intermédios a declaração de eventuais transformações geométricas e propriedades de aspeto.

**Transformações Geométricas:** Um nó recebe do seu antecessor uma matriz de transformações geométricas  $Ma$ . Sendo um nó intermédio, possui as suas próprias transformações geométricas, representadas por uma matriz única  $Mp$ . A matriz a aplicar ao objeto, assim como a passar aos seus eventuais descendentes, é calculada pela expressão  $M=Ma*Mp$ .

**Propriedades de aspeto:** Cada nó recebe propriedades de aspeto do seu antecessor (devem ser definidos valores de "default" para o nó raiz) e pode ter definidas as suas

"nula". As regras de desambiguação a usar neste caso são definidas na especificação do esquema LSF, na secção 3 deste documento.

## 2.3. Outras Entidades

Além de objetos, o esquema LSF pressupõe a existência de outras entidades, como sejam as fontes de luz, as texturas e os materiais. As entidades de iluminação e de visualização, tais como as fontes de luz, interferem com todo o grafo e, por isso, não devem ser ligadas a qualquer dos nós do grafo. Por isso, o esquema exige a sua declaração na parte inicial do ficheiro XML. As texturas e os materiais servem para utilização nos nós intermédios, pelo que também é previsto prepararem-se no início do ficheiro para posteriormente poderem ser referenciadas. Ao declarar-se uma textura esta deve ser lida, para memória, a partir de ficheiro .jpg ou .png correspondente (atenção: o comprimento e a largura de cada textura devem ser potências de 2).

A figura 1 apresenta um exemplo de um grafo de cena, com os nós intermédios representados por retângulos, e as folhas ("leaves") por círculos. As folhas correspondem a primitivas, ou seja, geometrias básicas como triângulos ou retângulos. As primitivas a implementar no contexto deste projeto são descritas em maior detalhe na secção 3.1.

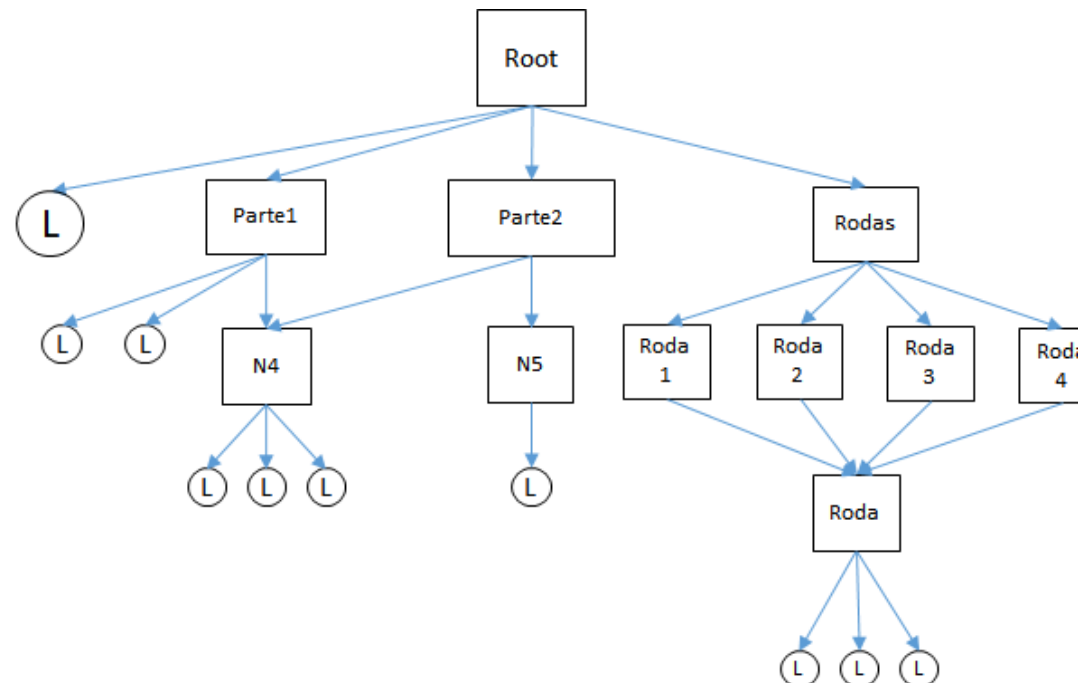


Figura 1 - Exemplo de um grafo de cena

Note-se que o número de descendentes directos de um nó intermédio é indeterminado, mas tem de existir pelo menos um descendente. Todo e qualquer nó deve ter um identificador, que é fornecido no ficheiro XML.

Cada nó intermédio pode ser instanciado várias vezes, ou seja, pode ser referenciado por mais do que um nó

possuam diferentes transformações geométricas. Assim, são criados os nós intermédios de instanciação *Roda1*, *Roda2*, *Roda3* e *Roda4*, todos referindo serem compostos pelo nó *Roda*, mas cada um dos quatro com transformações geométricas diferentes.

### 3. Esquema LSF

O esquema LSF - LAIG Scenegraph Format - constitui uma forma de especificar cenas 3D de uma forma muito simples e fácil de interpretar. Um documento em LSF 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 no esquema LSF deve ser verificado em termos de sintaxe e de consistência da estrutura, devendo a aplicação gerar mensagens de erro ou avisos, identificando eventuais irregularidades encontradas ou situações anómalas ou indesejáveis (por exemplo a referência a um nó inexistente).

Cada comando representa-se por um ou mais *tags*, contendo os parâmetros respectivos (se existirem). Um grupo de caracteres ou mesmo linhas limitado por `<!--` e `-->` é considerado um comentário. Todo o ficheiro deve ser escrito em minúsculas, incluindo o conteúdo dos atributos identificadores.

Um documento escrito segundo o esquema LSF estende-se por vários blocos. Cada bloco inicia-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 `<lights>`) e termina com o mesmo identificador antecedido de uma barra de divisão (no mesmo exemplo, `</lights>`). Entre as duas ocorrências, descreve-se o conteúdo do elemento identificado pela *tag*. A sequência de blocos é a seguinte:

```
initials          <!--global values →
cameras           <!--specification of
all camera views -->
illumination      <!--illumination parameters
-->
lights            <!--specification of all
lights -->
textures          <!--specification of all
textures -->
materials         <!--specification of all
materials -->
nodes             <!--specification of all
combined objects -->
```

Os blocos anteriores contêm definições de entidades do tipo correspondente (várias câmaras, luzes, várias texturas, 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).

**Nota:** a ordem de blocos pode ser desrespeitada; por exemplo, o bloco "materials" pode aparecer somente depois do bloco "nodes". A aplicação deve poder

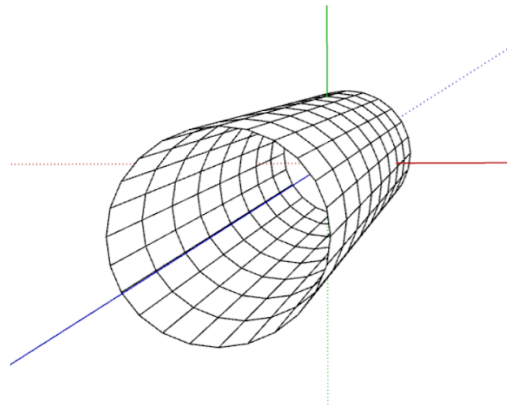
### 3.1 Novas primitivas

O esquema LSF contém a definição de cinco primitivas, que correspondem aos nós-folha no grafo de cena. As primitivas a implementar são **rectangle**, **triangle**, **cylinder**, **sphere**, e **torus**. As várias classes deverão ser implementadas como descrito nas próximas seções.

#### 3.1.1 Cilindro atualizado (MyCylinder)

Desenvolva a classe **MyCylinder** correspondente a um cilindro com os seguintes parâmetros: raio na base (*bottomRadius*); raio no topo (*topRadius*); altura (*height*); número de divisões em rotação (*slices*); número de divisões em altura (*stacks*).

O cilindro deve ser desenhado com a base centrada na origem, e o seu eixo central deve ser coincidente com o eixo positivo de Z, como mostra a figura 1. Devem também ser desenhados dois círculos nos topos - "tampas".



**Figura 1:** Imagem exemplo de um cilindro com base centrada na origem.  
(tampas não incluídas para facilitar visualização)

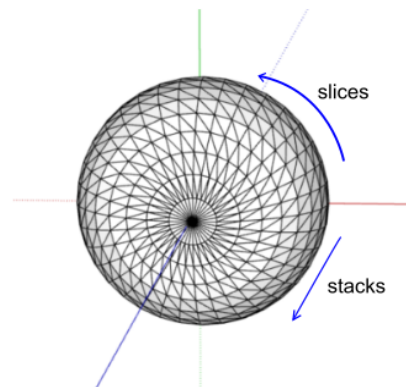
#### 3.1.2 Triângulo (MyTriangle)

Crie uma nova classe **MyTriangle** correspondente a um triângulo caracterizado pelos vértices definidos no ficheiro da cena. Defina as coordenadas de textura seguindo como referência o documento "*Triângulos e Coordenadas de Textura*" disponível no Moodle.

**Nota:** Esta classe difere da classe criada com o mesmo nome na unidade curricular de Computação Gráfica, que representava um triângulo rectângulo.

#### 3.1.3 Esfera (MySphere)

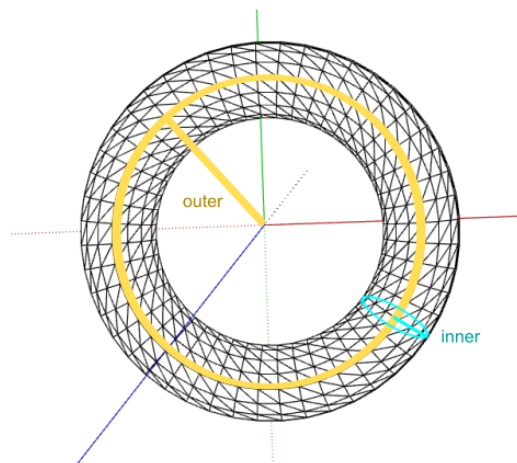
respetivamente nos lados positivo e negativo do eixo ZZ) e raio variável (*radius*). A esfera deverá ter um número variável de "lados" à volta do eixo Z (*slices*), e de "pilhas" (*stacks*, correspondentes ao número de divisões ao longo do eixo Z, desde o "equador" até um dos "pólos", ou seja, número de "fatias" da semi-esfera). A Figura 2 tem uma representação visual desta esfera, assim como o documento "*Geometria de Quádricas*" no Moodle.



**Figura 2:** Imagem exemplo de uma esfera centrada na origem.

### 3.1.4 Torus (MyTorus)

Pretende-se que crie a classe **MyTorus** para que esta consiga desenhar um *torus* centrado na origem, com raios interior e exterior variáveis (*innerRadius/outerRadius*) à volta do eixo Z, como representado na Figura 3. O torus deverá ter um número de "lados" (*slices*) variável à volta do raio interior (16, na figura), e um número de "voltas" (*loops*) à volta do eixo circular ( $12 \times 4 = 48$ , na figura).





---

 origem a volta do eixo z.

## 3.2. Síntaxe

Segue-se uma listagem representativa da sintaxe pretendida. A seguir, os símbolos possuem este significado:

ii: integer value  
 ff: float value  
 ss: string value  
 cc: character "x" or "y" or "z"  
 tt: "0" or "1" with Boolean significance false/true, respectively

```
-----
<lsf>

<initials>
  <reference length="ff" />          <!-- axis
length; "0" means no axis displayed -->
  <root id="ss" />          <!-- identifier of root node of
the scene graph; this node -->
                          <!-- must be defined in one of the
following NODE declarations -->
</initials>
<views default="ss" >
  <!-- at least one of the following type of cameras must
be declared -->
  <perspective id="ss" near="ff" far="ff" angle="ff">
    <from x="ff" y="ff" z="ff" />
    <to x="ff" y="ff" z="ff" />
  </perspective>
  <ortho id="ss" near="ff" far="ff" left="ff" right="ff"
top="ff" bottom="ff" >
    <from x="ff" y="ff" z="ff" />
    <to x="ff" y="ff" z="ff" />
    <up x="ff" y="ff" z="ff" /> <!-- optional, default
0,1,0 -->
  </ortho>
</views>

<illumination>
  <ambient r="ff" g="ff" b="ff" a="ff" />          <!--
- global ambient -->
  <background r="ff" g="ff" b="ff" a="ff" />      <!--
background color -->
</illumination>

<lights>
  <!-- NOTE: this block "light" must be repeated as
necessary with different "id". At least one light should be
present, and a maximum of eight lights is accepted. -->
  <light id="ss">          <!--
light identifier -->
    <enable value ="tt" />          <!--
enable/disable -->
    <position x="ff" y="ff" z="ff" w="ff" />
    <!-- light position; w=1: point light; w=0:
directional light -->
```

```

specular component -->
</light>
</lights>

<textures>
  <!-- NOTE: this block "TEXTURE" must be repeated as
necessary with different "id" -->
  <texture id="ss" path="ss" />
</textures>

<materials>
  <!-- NOTE: the "MATERIAL" block may be repeated as
required. Each defined material
requires a distinct "id". At least one material
should be present. -->
  <material id="ss">
    <shininess value="ff" />
    <specular r="ff" g="ff" b="ff" a="ff" />      <!--
specular reflection -->
    <diffuse r="ff" g="ff" b="ff" a="ff" />      <!--
- diffuse reflection -->
    <ambient r="ff" g="ff" b="ff" a="ff" />      <!--
- ambient reflection -->
    <emissive r="ff" g="ff" b="ff" a="ff" />      <!--
- emissive component -->
  </material>
</materials>

<nodes>
  <node id="ss">      <!-- defines one intermediate
node; may be repeated as necessary -->

    <!-- next two lines are mandatory -->
    <material id="ss" />      <!-- this superimposes
the material received from parent node
                                id="null" maintains
material from parent node -->

    <texture id="ss">      <!-- declared texture
superimposes the texture received from parent node
                                id="null" maintains
texture from parent node

                                id="clear" clears
texture declaration received    from parent node -->

    <amplification afs="ff" aft="ff" />
    <!-- for primitives = dx/afs, dy/aft -->
  </texture>
  <!-- geom. transf. are optional and may be
repeated, in any order, as necessary: -->
  <transformations>
    <translation x="ff" y="ff" z="ff" />
    <rotation axis="cc" angle="ff" />
    <scale sx="ff" sy="ff" sz="ff" />
  </transformations>

  <!-- declaring descendants, at least one node or one
leaf must be present
descendants may be mixed, nodes and leafs -->
  <descendants>
    <noderef id="ss" />      <!-- "ss" is the
identifier of a node or of leaf; -->
                                <!-- may be repeated as
necessary. It can refer an -->

```

```

y2="ff" />
        <!-- 2D coordinates on the XY plane for
bottom-left and top-right vertices. -->

        <leaf type="cylinder" height="ff" topRadius="ff"
bottomRadius="ff" stacks="ii" slices="ii"/>
        <!-- height, top radius, bottom radius,
sections along height (stacks), parts per section (slices);
main axis is aligned with ZZ; must have top and bottom caps
-->

        <leaf type="sphere" radius="ff" slices="ii"
stacks="ii"/>
        <!-- radius, sections along radius, parts
per section -->

        <leaf type="triangle" x1="ff" y1="ff" x2="ff"
y2="ff" x3="ff" y3="ff" />
        <!-- x y coordinates of each vertex;
triangle on plane XY -->

        <leaf type="torus" inner="ff" outer="ff"
slices="ii" loops="ii" />
        <!-- inner and outer radius, sections
around the inner radius, sections around the outer radius
-->
    </descendants>
</node>
</nodes>

</lsf>

```

#### Notas adicionais:

- 1) Em *runtime*, as fontes de luz podem ser ligadas/desligadas através da interface.
- 2) As *tags* devem respeitar a capitalização das letras, tal como especificadas acima
- 3) Os fatores de ampliação das texturas (*afs* e *aft*) só devem ser aplicados às primitivas básicas retângulo e triângulo. A forma de aplicação está ilustrada nos documentos Moodle.
- 4) A ordem pela qual os nós são declarados é arbitrária. Como tal, um nó pode referir como descendente um outro nó que seja declarado mais cedo ou mais tarde. Caso haja referência a nós que não existem, o programa deve assinalar o erro com uma mensagem.
- 5) Qualquer secção ou atributo em falta/errada deve ser assinalada com uma mensagem. Se for possível desenhar a cena sem esse elemento em falta/errado, deverá ser apresentada uma mensagem de aviso (*warning*) e o programa continuará a correr; caso contrário deverá ser apresentada uma mensagem de erro (*error*) e o programa deverá parar.
- 6) Na situação anterior de "*warning*", tente substituir o valor errado ou em falta por um valor padrão, de forma a que a cena seja desenhada.

