

Universidade do Minho

LICENCIATURA EM CIÊNCIAS DE COMPUTAÇÃO

[17-18] COMPUTAÇÃO GRÁFICA [CCOM]

Relatório Técnico - Fase 2 – Transformações Geométricas

Grupo - 11

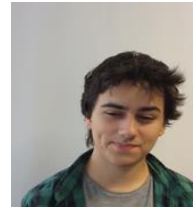
8 de Abril de 2018

Nome: ADRIANO VAZ DE CARVALHO CAMPINHO

E-mail: a79032@alunos.uminho.pt

Número: A79032

Curso: L.C.C.



Nome: ARTUR JORGE GOMES QUEIROZ

E-mail: a77136@alunos.uminho.pt

Número: A77136

Curso: L.C.C.



Nome: LUÍS PAULO FERREIRA GOMES NETO

E-mail: a73233@alunos.uminho.pt

Número: A73233

Curso: L.C.C.



Nome: VASCO LUZIO LEITÃO

E-mail: a79220@alunos.uminho.pt

Número: A79220

Curso: L.C.C.



Resumo

O objetivo principal deste relatório técnico revolve em explicar e esclarecer os passos que os alunos identificados na pagina frontal comprometeram nesta segunda etapa e as pretendidas resoluções e demonstrações para cada problema exposto no enunciado que foi fornecido pelo docente da unidade curricular.

Nesta segunda fase o tema central tem o seu foco na criação de um cenário utilizando transformações geométricas. O cenário demonstrado nesta fase revolveu na criação de um sistema solar estático, incluindo o sol, planetas e luas definidas numa hierarquia. Este foi definido num ficheiro *.xml que foi interpretado e disposto graficamente pelo motor desenvolvido.

Conteúdo

1	Introdução	1
1.1	Estrutura do Relatório	2
2	Análise e Especificação	3
2.1	Descrição dos Problemas & Especificação de Requisitos	3
2.1.1	Configuração XML	3
3	Esquemas de Concepção	5
3.1	Génese do Cenário	5
3.2	Mecânica geral de parsing de Etiquetas	6
4	Codificação e Testes	7
4.1	Alternativas, Decisões e Problemas de Implementação	7
4.1.1	Implementação de etiquetas XML	7
4.1.1.1	Transformação Geométrica - Translação	7
4.1.1.2	Transformação Geométrica - Rotate	8
4.1.1.3	Transformação Geométrica - Escala	9
4.1.1.4	Cor	10
4.1.2	Motor 3D	11
4.1.2.1	Cenário - Sistema Solar	11
4.1.2.2	Keybinds do Motor 3D	13
4.1.2.3	Mobilidade da câmara	13
4.1.2.4	Contador FPS	13
4.2	Testes realizados e Resultados	14
4.2.1	Gerador de Ficheiros	14
4.2.1.1	Planetas e Luas	14
4.2.2	Motor 3D	15
4.2.2.1	Cenário - Sistema Solar	15
5	Conclusão	16
A	Código Parcial do Programa	

Lista de Figuras

3.1	Esquema exemplificando o programa generator a gerar o cenário ditado pelo ficheiro <i>espaco.xml</i>	5
3.2	Esquema exemplificando o parsing da tag translate	6
4.1	Demonstração do Cenário - Sistema Solar no motor 3D	11
4.2	Demonstração de como gerar a primitiva gráfica - Esfera	14
4.3	Demonstração de como iniciar o motor 3D com o Cenário - Sistema Solar	15

Capítulo 1

Introdução

Nesta segunda fase concentrou-se o desenvolvimento nos seguintes pontos:

- Implementar a arquitetura necessária para o motor3d suportar diversas transformações geométricas, e.g. :
 - Translações
 - Escalas
 - Rotações
- Implementar a arquitetura necessária para o motor3d suportar o parsing de novas tags *xml*, e.g.:
 - $\langle \textit{translate } X = \text{"..."} Y = \text{"..."} Z = \text{"..."} \backslash \rangle$
 - $\langle \textit{scale } X = \text{"..."} Y = \text{"..."} Z = \text{"..."} \backslash \rangle$
 - $\langle \textit{rotate angle} = \text{"..."} \textit{axis} X = \text{"..."} \textit{axis} Y = \text{"..."} \textit{axis} Z = \text{"..."} \backslash \rangle$
 - $\langle \textit{color } R = \text{"..."} G = \text{"..."} B = \text{"..."} \backslash \rangle$
- Cartografar o cenário - Sistema Solar através de um ficheiro *xml* e gerar todas as figuras necessárias.

Para este efeito foram utilizadas principalmente as seguintes ferramentas na duração desta fase:

- C++
 - Pugixml-1.8
- OpenGL
 - GLUT
 - GLEW

1.1 Estrutura do Relatório

No capítulo 2 começamos por fornecer a descrição do problema, transcrevendo os dados que nos foram dados no enunciado pelo docente e prosseguimos para uma análise digestiva dos requisitos impostos na descrição do problema.

No capítulo 3 oferecemos um esquema manufaturados na plataforma '*draw.io*' a demonstrar a engenharia que suporta a génese do cenário.

O capítulo 4 é o capítulo mais "recheado", na sua primeira secção temos "Alternativas, Decisões e Problemas de Implementação" onde entramos numa maior profundidade na síntese e no propósito dos programas que foram produzidos para atacar os problemas estando essa síntese acompanhada com exemplos ilustrativos, e na outra secção "Testes realizados e Resultados" procedemos a um exemplo demonstrativo de os programas em execução com os respetivos comandos utilizados e imagens destes em execução.

No capítulo 5 concluímos com uma análise crítica dos resultados e um alívio sobre o estado final do projeto.

Nas páginas finais é apresentada a bibliografia seguida imediatamente de um "Apêndice A", que contem parte do código dos programas que foram produzidos e utilizados para resolver os problemas adotados.

Capítulo 2

Análise e Especificação

2.1 Descrição dos Problemas & Especificação de Requisitos

2.1.1 Configuração XML

Esta fase revolve na criação de cenas hierárquicas usando transformações geométricas. Uma cena é definida como uma árvore em que cada nodo contém um conjunto de transformações geométricas (translações, rotações e escalas) e opcionalmente um conjunto de modelos. Cada nodo pode também conter sub-nodos¹ Eis um exemplo de um ficheiro de configuração XML com um único grupo:

```
<scene>
  <group>
    <translate X=5 Y=0 Z=2 />
    <rotate angle=45 axisX=0 axisY=1 axisZ=0 />
    <models>
      <model file="sphere.3d" />
    </models>
  </group>
</scene>
```

¹Children nodes.

Eis um exemplo de um ficheiro de configuração XML com um grupo com subgrupo:

```
<scene>
  <group>
    <translate X="1" />
    <models>
      <model file="sphere.3d" />
    </models>
    <group>
      <translate Y="1" />
      <models>
        <model file="cone.3d" />
      </models>
    </group>
  </group>
</scene>
```

Neste segundo exemplo, o sub-grupo irá herdar as transformações geométricas do grupo parente. As transformações geométricas podem existir apenas dentro de um grupo e são aplicadas à todos os modelos e subgrupos.

O cenário de demonstração requisitado para esta fase trata-se de um modelo estático do sistema solar, incluindo o sol, os planetas e as suas luas definidas numa hierarquia.

Capítulo 3

Esquemas de Concepção

3.1 Génese do Cenário

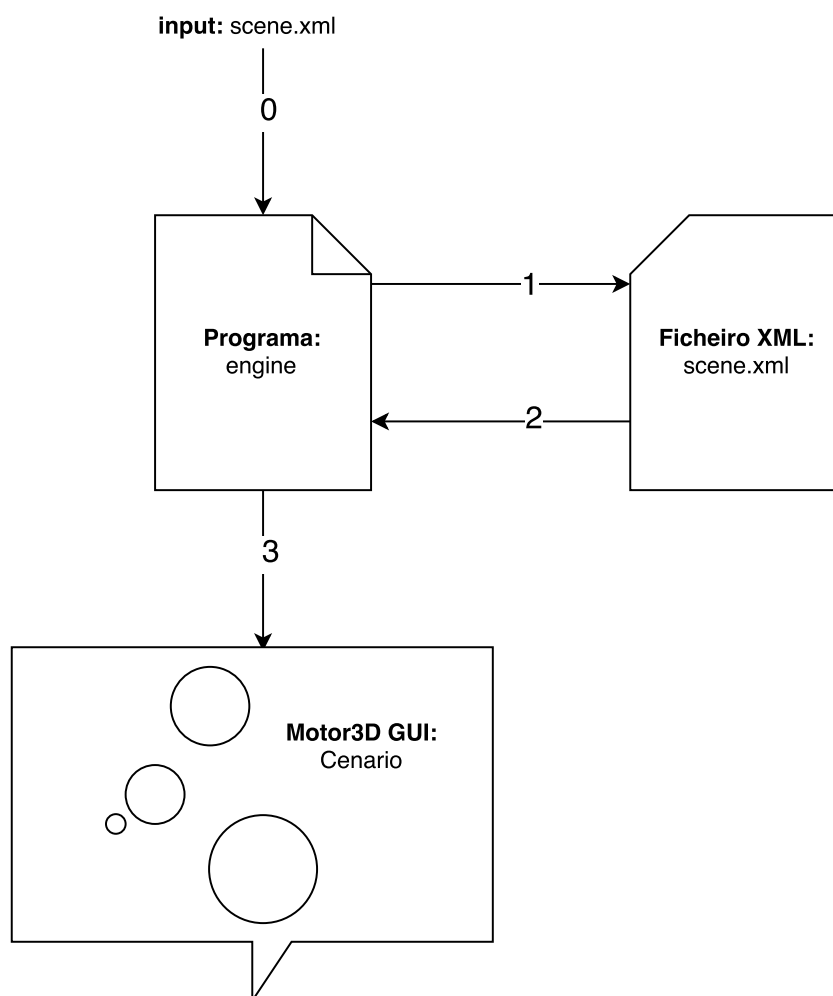


Figura 3.1: Esquema exemplificando o programa generator a gerar o cenário ditado pelo ficheiro *espaco.xml*

Note que pode visualizar o conteúdo do ficheiro *espaco.xml* no apêndice A.2.

3.2 Mecânica geral de parsing de Etiquetas

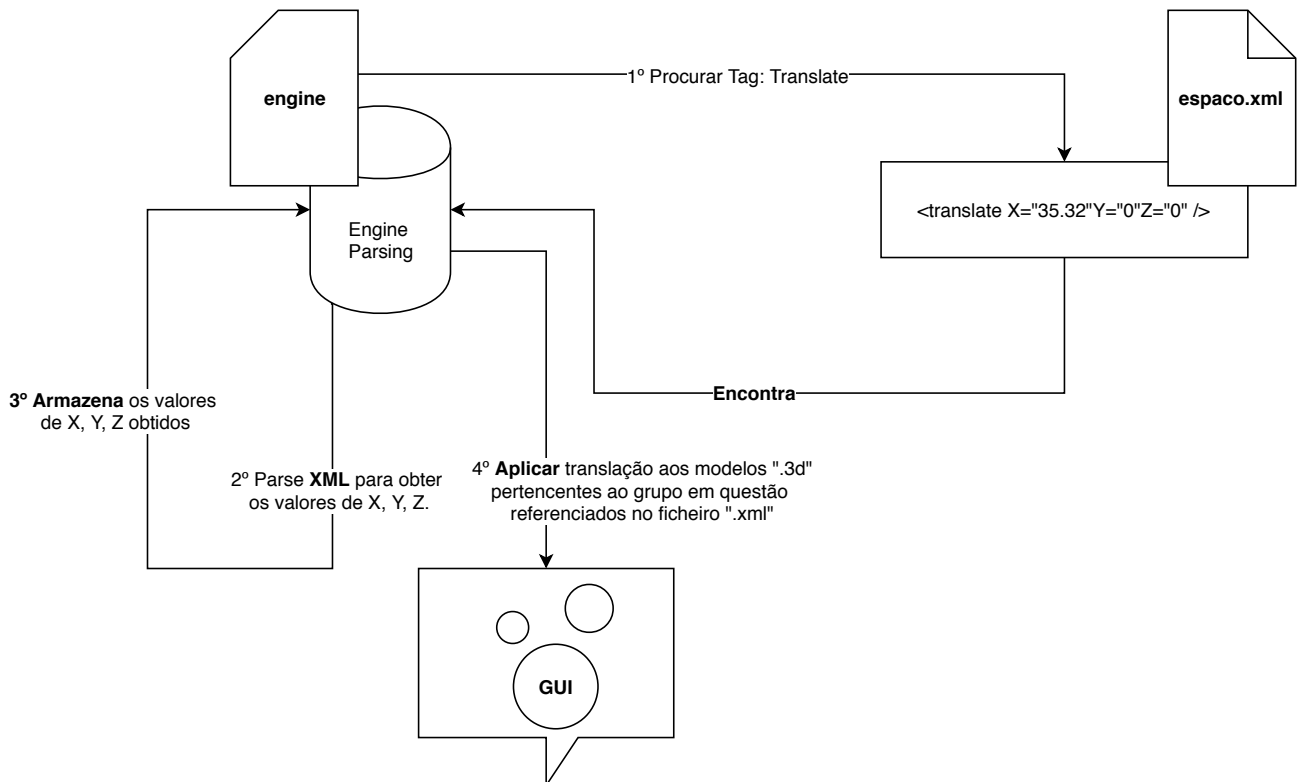


Figura 3.2: Esquema exemplificando o parsing da tag translate

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

4.1.1 Implementação de etiquetas XML

4.1.1.1 Transformação Geométrica - Translação

Para facilitar o parsing da tag `<translate X="argX"Y="argY"Z="argZ"/>` foi criado a classe **Translate**:

Listing 4.1: class Translate

```
1 class Translate : public PhysicScene{
2     float x, y, z;
3     public:
4     Translate(float,float,float);
5     Translate(xml_node);
6     void draw();
7 };
```

Numa instância da classe, os métodos de **Translate()** irão armazenar nessa instância os valores de **X**, **Y**, e **Z** que foram recebidos no parsing da tag como, neste caso, **argX**, **argY** e **argZ** respetivamente, e o método **draw()** irá aplicar o **glTranslatef(this->x,this->y,this->z)** a esses valores armazenado, permitindo assim a translação de primitivas gráficas dentro do mesmo grupo XML. Note que pode visualizar mais concretamente a forma como estes métodos foram implementados no apêndice A.1.

4.1.1.2 Transformação Geométrica - Rotate

Para facilitar o parsing da tag `<rotate angle=argAngle axisX="argX"axisY="argY"axisZ="argZ"/>` foi criado a classe **Rotate**:

Listing 4.2: class Rotate

```
1 class Rotate : public PhysicScene{
2     float angle;
3     float x, y, z;
4     public:
5     Rotate(float,float,float,float);
6     Rotate(xml_node);
7     void draw();
8 };
```

Numa instância da classe, os métodos de **Rotate()** irão armazenar nessa instância os valores do **Ângulo**, **X**, **Y**, e **Z** que foram recebidos no parsing da tag como, neste caso, *argAngle*, *argX*, *argY* e *argZ* respetivamente, e o método **draw()** irá aplicar o **glRotatef(this->angle,this->x,this->y,this->z)** a esses valores armazenado, permitindo assim a rotação da primitivas gráficas dentro do mesmo grupo XML. Note que pode visualizar mais concretamente a forma como estes métodos foram implementados no apêndice A.1.

4.1.1.3 Transformação Geométrica - Escala

Para facilitar o parsing da tag `<scale X="argX"Y="argY"Z="argZ"/>` foi criado a classe **Scale**:

Listing 4.3: class Scale

```
1 class Scale : public PhysicScene{
2     float x, y, z;
3     public:
4         Scale(float,float,float);
5         Scale(xml_node);
6         void draw();
7 };
```

Numa instância da classe, os métodos de **Scale()** irão armazenar nessa instância os valores de **X**, **Y**, e **Z** que foram recebidos no parsing da tag como, neste caso, *argX*, *argY* e *argZ* respectivamente, e o método **draw()** irá aplicar o **glScalef(this->x,this->y,this->z)** a esses valores armazenados, permitindo assim a escala de primitivas gráficas dentro do mesmo grupo XML. Note que pode visualizar mais concretamente a forma como estes métodos foram implementados no apêndice A.1.

4.1.1.4 Cor

Para facilitar o parsing da tag `<color R="argR"G="argG"B="argB"/>` foi criado a classe **Color**:

Listing 4.4: class Color

```
1 class Color : public PhysicScene{
2     float redVal, greenVal, blueVal;
3     public:
4     Color(float, float, float);
5     Color(xml_node);
6     void draw();
7 };
```

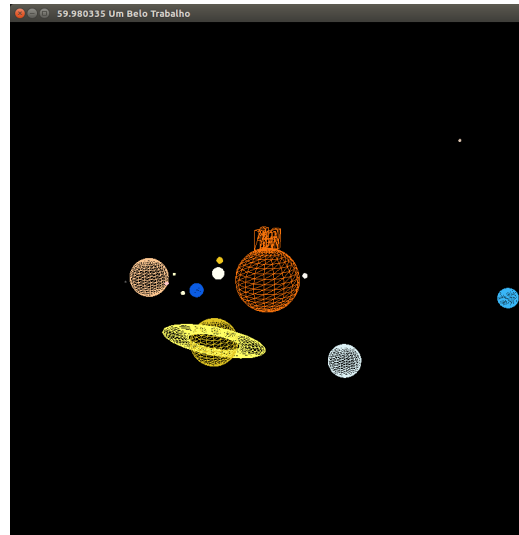
Numa instância da classe, os métodos de **Color()** irão armazenar nessa instância os valores de **R**, **G**, e **B** que foram recebidos no parsing da tag como, neste caso, **argR**, **argG** e **argB** respectivamente, e o método **draw()** irá aplicar o **glColor3ub(this->redVal, this->greenVal, this->blueVal)** a esses valores armazenado, permitindo assim a coloração diferencial de primitivas gráficas dentro do mesmo grupo XML. Note que pode visualizar mais concretamente a forma como estes métodos foram implementados no apêndice A.1.

4.1.2 Motor 3D

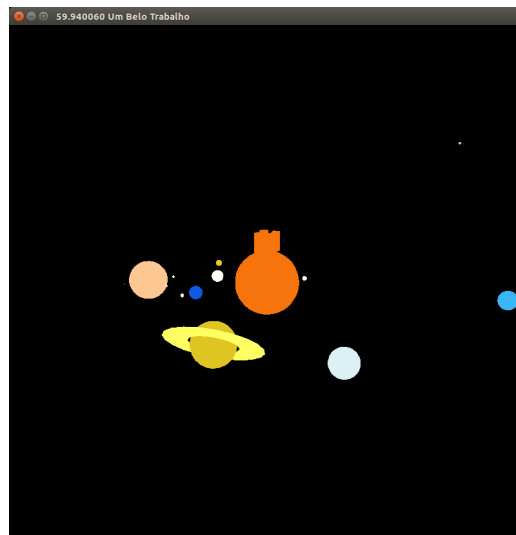
4.1.2.1 Cenário - Sistema Solar



(a) GL_POINT



(b) GL_LINE



(c) GL_FILL

Figura 4.1: Demonstração do Cenário - Sistema Solar no motor 3D

Para gerar o **Cenário - Sistema Solar** através do **Motor 3d** é necessário considerar várias peças em jogo, de seguida será explicado brevemente a mecânica por detrás deste.

Numa primeira fase, o motor vai procurar pelo ficheiro *.xml* que foi lhe passado como argumento e irá efetivamente fazer o parsing desse ficheiro, absorvendo informações relevantes, para uma class Scene.

Numa segunda fase, o motor fazer o metodo draw de Scene, e de seguida representar graficamente e fielmente os *planetas* e *luas* em questão, pois será necessários posteriormente aplicar, por exemplo, as transformações geométricas do grupo em que o ficheiro *.3d* se enquadra.

De seguida, numa terceira fase será efetivamente **representado graficamente** o(s) *planeta(s)* e *lua(s)* relevante(s) através das informações que o programa adquiriu durante as duas primeiras fases, demonstrando assim o cenário.

4.1.2.2 Keybinds do Motor 3D

Key	Sumário
1	<i>glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);</i>
2	<i>glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);</i>
3	<i>glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);</i>
S / Scroll Down	Afastar câmera dos eixos;
W / Scroll Up	Aproximar câmera dos eixos;
X	Mostrar eixos coloridos: X - Vermelho, Y - Verde, Z - Azul;
↑	Mover a câmera para cima;
←	Mover a câmera para a esquerda;
→	Mover a câmera para a direita;
↓	Mover a câmera para baixo;
MOUSE1 + DRAG	Mover a câmera;

4.1.2.3 Mobilidade da câmara

A câmara é um aspeto importante de um cenário virtual, pois é com ela que é possível a visualização do mesmo.

Para esta segunda fase decidimos manter a câmara da fase anterior, mantendo o "olhar" para o centro (0,0,0), e ter uma livre rotação perante o mesmo, numa das proximas fases sera dada a possibilidade da utilizacao de uma camera no modo "explorador".

A rotação da câmara é feita de forma a contornar uma esfera, dando um raio e dois angulos, um para representar 360 graus orientado pelo eixo y (tratado por alfa) e o outro para representar 180 graus orientado pelo eixo perpendicular ao eixo y e ao eixo criado pelo angulo alfa no plano XZ (tratado por beta). Com isto temos que não é possível a rotação completa por beta.

4.1.2.4 Contador FPS

Ao longo do trabalho, foi nos muito útil a criação de um contador FPS para medir a performance do motor 3d desenvolvido. Este contador é ativo no inicio do programa e de um em um segundo atualiza o título da janela para que apareçam o numero de frames desenhados no segundo anterior.

4.2 Testes realizados e Resultados

4.2.1 Gerador de Ficheiros

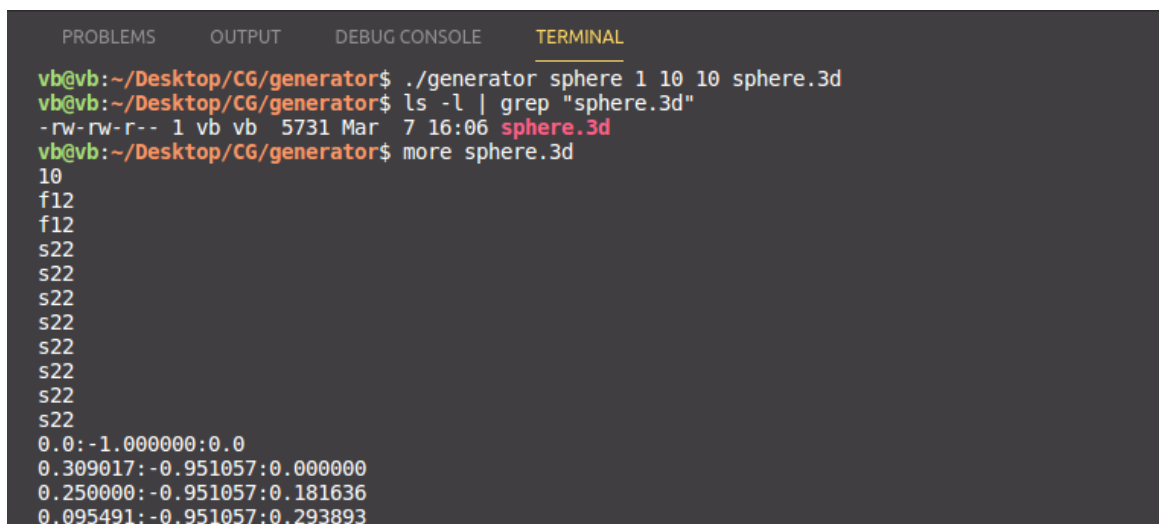
4.2.1.1 Planetas e Luas

Note que nesta fase foi utilizado a primitiva gráfica - **Esfera** para formular a base gráfica dos planetas e luas que compõem o modelo do sistema solar.

Exemplo ilustrativo do gerador a gerar a primitiva gráfica - esfera na linha de comandos:

1. `./generator sphere 1 10 10 sphere.3d`
2. `ls -l | grep "sphere.3d"`
3. `more sphere.3d`

1. O comando efetivamente gera o ficheiro "sphere.3d".
2. Verificamos na diretoria local pela existência do ficheiro "sphere.3d" criado.
3. Demonstramos parcialmente o conteúdo do ficheiro "sphere.3d".



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
vb@vb:~/Desktop/CG/generator$ ./generator sphere 1 10 10 sphere.3d
vb@vb:~/Desktop/CG/generator$ ls -l | grep "sphere.3d"
-rw-rw-r-- 1 vb vb 5731 Mar  7 16:06 sphere.3d
vb@vb:~/Desktop/CG/generator$ more sphere.3d
10
f12
f12
s22
s22
s22
s22
s22
s22
s22
s22
s22
0.0:-1.000000:0.0
0.309017:-0.951057:0.000000
0.250000:-0.951057:0.181636
0.095491:-0.951057:0.293893
```

Figura 4.2: Demonstração de como gerar a primitiva gráfica - Esfera

4.2.2 Motor 3D

4.2.2.1 Cenário - Sistema Solar

Exemplo ilustrativo do motor a representar a o Cenário - Sistema Solar e linha de comandos relevante:

1. `./engine espaco.xml`

1. Será passado o ficheiro **espaco.xml** para o motor 3d levando a que este procure por pelos ficheiros ***.3d** relevantes, aplique as transformações geométricas para cada grupo conforme a configuração do ficheiro XML e gere graficamente o cenário.

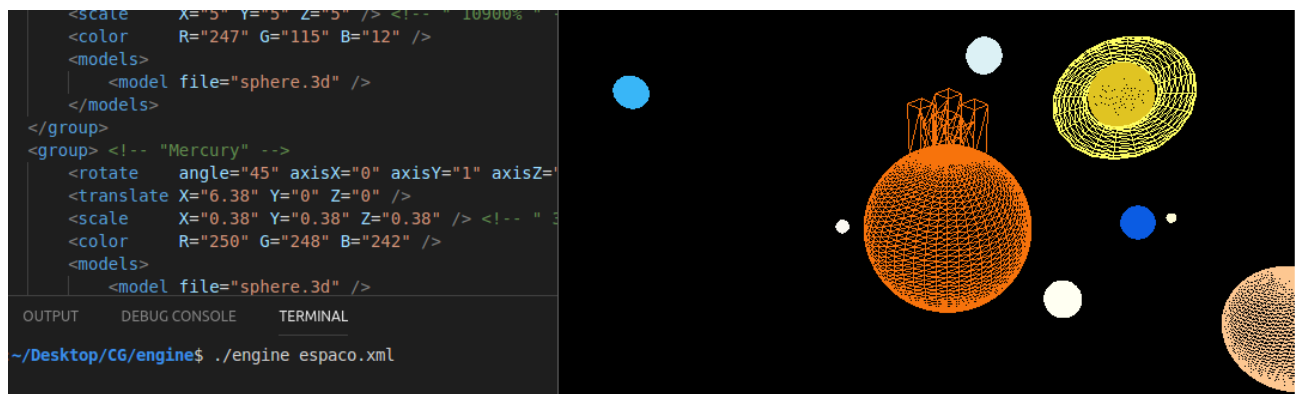


Figura 4.3: Demonstração de como iniciar o motor 3D com o Cenário - Sistema Solar

Descrição do cenário:

1. Todos os objectos do cenário estão organizados hierarquicamente (estrelas > planetas > luas);
2. Todas as estrelas, planetas e luas são representadas por esferas (objecto sphere.3d);
3. Todos os planetas estão em orbitas em diferentes distancias do sol;
4. Tanto o planeta "Terra" como "Júpiter" tem luas na sua orbita (1 e 3 respectivamente);

Capítulo 5

Conclusão

Nesta fase deu-se a renovação bem sucedida da arquitetura de suporte e parsing de ficheiros XML dentro do contexto do motor 3D de modo a que fosse de encontro as novas exigências nesta etapa e fosse possível representar fielmente o cenário relevante. Esta nova arquitetura é incrivelmente legível e altamente escalável, facilitando assim a implementação de novas funcionalidade que serão necessárias suportar nas seguintes etapas que nos esperam. Foram atingidos todos os objetivos propostos pelo docente nesta fase e ainda houve lugar para a implementação de certas funcionalidades adicionais. Com isto damos por encerramos esta segunda fase.

Bibliografia

- [1] Shreiner, D., Woo, M., Neider, J., Davis, T. (2006).
- [2] OpenGL Programming Guide (5th ed.). Addison Wesley. Angel, E.
- [3] (1999). Interactive Computer Graphics (2nd ed.).
- [4] Addison Wesley. Lengyel, E. (2011).
- [5] Mathematics for 3D Game Programming and Computer Graphics (3rd ed.).

Apêndice A

Código Parcial do Programa

Listing A.1: Excerto Ficheiro: scene.cpp

```
1 #include "scene.h"
2 #include <stdlib.h>
3 #include <iostream>
4
5 //-Model-----//
6
7 Model::Model(xml_node node) {
8     FILE* file;
9     int i, total;
10    float x, y, z;
11
12    file = fopen(node.attribute("file").as_string(), "r");
13    //strcpy((char*)this->file, node.attribute("file").as_string());
14    if(!file) error("opening file");
15    fscanf(file, "%d\n", &(this->N));
16
17    this->index = 0;
18    this->pos = new int[this->N];
19    this->len = new int[this->N];
20    this->typ = new char[this->N];
21
22    for(i=0, total=0; i<this->N; i++) {
23        fscanf(file, "%c%d\n", &typ[i], &len[i]);
24        pos[i] = total;
25        total += len[i];
26    }
27    this->arrayFloat = new float[total * 3];
28    while(fscanf(file, "%f:%f:%f\n", &x, &y, &z) != EOF) {
29        this->arrayFloat[this->index++] = x;
30        this->arrayFloat[this->index++] = y;
31        this->arrayFloat[this->index++] = z;
32    }
33 }
34
35 void Model::draw() {
36     int i, gl;
37     glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
38     glBufferData(GL_ARRAY_BUFFER, this->index * 3 * sizeof(float), this->arrayFloat,
39                 GL_STATIC_DRAW);
39     glVertexAttribPointer(3, GL_FLOAT, 0, 0);
40
41     for(i=0; i<this->N; i++) {
```

```

42     switch(this->typ[i]) {
43         case 's' : gl = GL_TRIANGLE_STRIP; break;
44         case 'f' : gl = GL_TRIANGLE_FAN; break;
45         case 't' : gl = GL_TRIANGLES; break;
46         default : error("array type bad specified");
47     }
48     glDrawArrays(gl, this->pos[i], this->len[i]);
49 }
50 }
51
52 //-Translate-----//
53
54 Translate::Translate(float x, float y, float z) {
55     this->x = x;
56     this->y = y;
57     this->z = z;
58 }
59
60 Translate::Translate(xml_node node) {
61     //cout << "parse de Translate camecou" << endl;
62     xml_attribute aux_x = node.attribute("X");
63     xml_attribute aux_y = node.attribute("Y");
64     xml_attribute aux_z = node.attribute("Z");
65
66     this->x = aux_x ? aux_x.as_float() : 0.0f;
67     this->y = aux_y ? aux_y.as_float() : 0.0f;
68     this->z = aux_z ? aux_z.as_float() : 0.0f;
69     //cout << "parse de Translate acabou" << endl;
70 }
71
72 void Translate::draw() {
73     glTranslatef(this->x, this->y, this->z);
74 }
75
76 //-Rotate-----//
77
78 Rotate::Rotate(float angle, float x, float y, float z) {
79     this->angle = angle;
80     this->x = x;
81     this->y = y;
82     this->z = z;
83 }
84
85 Rotate::Rotate(xml_node node) {
86     //cout << "parse de Rotate camecou" << endl;
87     xml_attribute aux_angle = node.attribute("angle");
88     xml_attribute aux_x = node.attribute("axisX");
89     xml_attribute aux_y = node.attribute("axisY");
90     xml_attribute aux_z = node.attribute("axisZ");

```

```

91
92     this->angle = aux_angle ? aux_angle.as_float() : 0.0f;
93     this->x     = aux_x     ? aux_x.as_float()   : 0.0f;
94     this->y     = aux_y     ? aux_y.as_float()   : 0.0f;
95     this->z     = aux_z     ? aux_z.as_float()   : 0.0f;
96     //cout << "parse de Rotate acabou" << endl;
97 }
98
99 void Rotate::draw() {
100     glRotatef(this->angle, this->x, this->y, this->z);
101 }
102
103 //-Scale-----//
104
105 Scale::Scale(float x, float y, float z) {
106     this->x = x;
107     this->y = y;
108     this->z = z;
109 }
110
111 Scale::Scale(xml_node node) {
112     //cout << "parse de Scale camecou" << endl;
113     xml_attribute aux_x = node.attribute("X");
114     xml_attribute aux_y = node.attribute("Y");
115     xml_attribute aux_z = node.attribute("Z");
116
117     this->x = aux_x ? aux_x.as_float() : 0.0f;
118     this->y = aux_y ? aux_y.as_float() : 0.0f;
119     this->z = aux_z ? aux_z.as_float() : 0.0f;
120     //cout << "parse de Scale acabou" << endl;
121 }
122
123 void Scale::draw() {
124     glScalef(this->x, this->y, this->z);
125 }
126
127 //-Color-----//
128
129 Color::Color(float redVal, float greenVal, float blueVal) {
130     this->redVal = redVal;
131     this->greenVal = greenVal;
132     this->blueVal = blueVal;
133 }
134
135 Color::Color(xml_node node) {
136     //cout << "parse de Color camecou" << endl;
137     xml_attribute aux_redVal = node.attribute("R");
138     xml_attribute aux_greenVal = node.attribute("G");
139     xml_attribute aux_blueVal = node.attribute("B");

```

```

140
141     this->redVal = aux_redVal ? aux_redVal.as_float() : 0; //f
142     this->greenVal = aux_greenVal ? aux_greenVal.as_float() : 0; //f
143     this->blueVal = aux_blueVal ? aux_blueVal.as_float() : 0; //f
144     //cout << "parse de Color acabou" << endl;
145 }
146
147 void Color::draw() {
148     glColor3ub(this->redVal,this->greenVal,this->blueVal);
149 }
150
151 //-Models-----//
152
153 Models::Models(std::vector<Model*> models) {
154     this->models = models;
155 }
156
157 Models::Models(xml_node node) {
158     for(xml_node trans = node.first_child(); trans; trans = trans.next_sibling()) {
159         this->models.push_back(new Model(trans));
160     }
161 }
162
163 void Models::draw() {
164     for(int i = 0; i<models.size();i++) {
165         this->models[i]->draw();
166     }
167 }
168
169 //-Group-----//
170
171 Group::Group(std::vector<PhysicScene*> transforms) {
172     this->transforms = transforms;
173 }
174
175 Group::Group(xml_node node) {
176     //cout << "parse de Group comeceu" << endl;
177     for(xml_node trans = node.first_child(); trans; trans = trans.next_sibling()) {
178         if(strcmp(trans.name(),"translate") == 0) {
179             this->transforms.push_back(new Translate(trans));
180
181         }else if(strcmp(trans.name(),"rotate") == 0) {
182             this->transforms.push_back(new Rotate(trans));
183
184         }else if(strcmp(trans.name(),"scale") == 0) {
185             this->transforms.push_back(new Scale(trans));
186
187         }else if(strcmp(trans.name(),"color") == 0) {
188             this->transforms.push_back(new Color(trans));

```

```
189
190     }else if(strcmp(trans.name(),"models") == 0) {
191         this->transforms.push_back(new Models(trans));
192
193     }else if(strcmp(trans.name(),"group") == 0) {
194         this->transforms.push_back(new Group(trans));
195
196     }else{
197         cout << "Erro no formato do xml, foi lido: " << trans.name() << endl;
198     }
199 }
200 //cout << "parse de Group acabou" << endl;
201 }
202
203 void Group::draw() {
204     glPushMatrix();
205     for(int i = 0; i<transforms.size(); i++) {
206         this->transforms[i]->draw();
207     }
208     glPopMatrix();
209 }
210 //-Scene-----//
211
212 Scene::Scene(Group* group) {
213     this->group = group;
214 }
215
216 Scene::Scene(const char* xml_file) {
217     //cout << "parse de Scene começou" << endl;
218     xml_document doc;
219     xml_parse_result result;
220
221     if( !(result = doc.load_file(xml_file)) ) {
222         std::cout << "XML [" << xml_file << "] parsed with errors, attr value: [" <<
                doc.child("node").attribute("attr").value() << "]\n";
223         std::cout << "Error description: " << result.description() << "\n";
224         std::cout << "Error offset: " << result.offset << " (error at [... " <<
                (xml_file + result.offset) << "]\n\n";
225     }
226     xml_node models = doc.child("scene");
227
228     this->group = new Group(models.first_child());
229     //cout << "parse de Scene acabou" << endl;
230 }
231
232 void Scene::draw() {
233     this->group->draw();
234 }
```

Listing A.2: Excerto Ficheiro: espaco.xml

```
1 <scene>
2   <group> <!-- " Terra.Scale = 1" -->
3     <group> <!-- "Sol" -->
4       <translate X="0" Y="0" Z="0" />
5       <group>
6         <translate X="0" Y="6.5" Z="0" />
7         <color    R="247" G="115" B="12" />
8         <models>
9           <model file="N64.3d" />
10        </models>
11      </group>
12    <group> <!-- "Mercury" -->
13      <rotate   angle="45" axisX="0" axisY="1" axisZ="0" />
14      <translate X="6.38" Y="0" Z="0" />
15      <scale    X="0.38" Y="0.38" Z="0.38" /> <!-- " 38% " -->
16      <color    R="250" G="248" B="242" />
17      <models>
18        <model file="sphere.3d" />
19      </models>
20    </group>
21    <group> <!-- "Venus" -->
22      <rotate   angle="180" axisX="0" axisY="1" axisZ="0" />
23      <translate X="8.71" Y="0" Z="0" />
24      <scale    X="0.95" Y="0.95" Z="0.95" /> <!-- " 95% " -->
25      <color    R="255" G="255" B="242" />
26      <models>
27        <model file="sphere.3d" />
28      </models>
29    </group>
30    <group> <!-- "Earth" -->
31      <rotate   angle="230" axisX="0" axisY="1" axisZ="0" />
32      <translate X="11.66" Y="0" Z="0" />
33      <group> <!-- "Moon" -->
34        <rotate   angle="10" axisX="0" axisY="1" axisZ="0" />
35        <translate X="2.27" Y="0" Z="0" />
36        <scale    X="0.27" Y="0.27" Z="0.27" /> <!-- " 27% " -->
37        <color    R="254" G="252" B="215" />
38        <models>
39          <model file="sphere.3d" />
40        </models>
41      </group>
42      <scale    X="1" Y="1" Z="1" /> <!-- " 100% " -->
43      <color    R="11" G="92" B="227" />
44      <models>
45        <model file="sphere.3d" />
46      </models>
47    </group>
```

```

48     <group> <!-- "Mars" -->
49         <rotate    angle="150" axisX="0" axisY="1" axisZ="0" />
50         <translate X="14.19" Y="0" Z="0" />
51         <scale     X="0.53" Y="0.53" Z="0.53" /> <!-- " 53% " -->
52         <color     R="240" G="198" B="29" />
53         <models>
54             <model file="sphere.3d" />
55         </models>
56     </group>
57     <group> <!-- "Jupiter" -->
58         <rotate    angle="200" axisX="0" axisY="1" axisZ="0" />
59         <translate X="18.72" Y="0" Z="0" />
60         <group> <!-- "moon 1" -->
61             <rotate    angle="40" axisX="0" axisY="1" axisZ="0" />
62             <translate X="4" Y="0" Z="0" />
63             <scale     X="0.1" Y="0.1" Z="0.1" />
64             <color     R="100" G="100" B="100" />
65             <models>
66                 <model file="sphere.3d" />
67             </models>
68         </group>
69         <group> <!-- "moon 2" -->
70             <rotate    angle="140" axisX="0" axisY="1" axisZ="0" />
71             <translate X="4.5" Y="0" Z="0" />
72             <scale     X="0.3" Y="0.2" Z="0.3" />
73             <color     R="255" G="200" B="200" />
74             <models>
75                 <model file="sphere.3d" />
76             </models>
77         </group>
78         <group> <!-- "moon 3" -->
79             <rotate    angle="210" axisX="0" axisY="1" axisZ="0" />
80             <translate X="4.1" Y="0" Z="0" />
81             <scale     X="0.2" Y="0.2" Z="0.2" />
82             <color     R="253" G="255" B="200" />
83             <models>
84                 <model file="sphere.3d" />
85             </models>
86         </group>
87         <scale     X="3" Y="3" Z="3" /> <!-- " 1120% " -->
88         <color     R="253" G="199" B="145" />
89         <models>
90             <model file="sphere.3d" />
91         </models>
92     </group>
93     <group> <!-- "Saturn" -->
94         <rotate    angle="280" axisX="0" axisY="1" axisZ="0" />
95         <translate X="25.52" Y="0" Z="0" />
96         <scale     X="2.8" Y="2.8" Z="2.8" /> <!-- " 945% " -->

```

```

97     <group>
98         <rotate    angle="40" axisX="0" axisY="0" axisZ="1" />
99         <scale     X="1.1" Y="0.1" Z="1.1" /> <!-- " 945% " -->
100        <color     R="255" G="255" B="100" />
101        <models>
102            <model file="torus.3d" />
103        </models>
104    </group>
105    <color     R="224" G="196" B="34" />
106    <models>
107        <model file="sphere.3d" />
108    </models>
109</group>
110<group> <!-- "Uranus" -->
111    <rotate    angle="310" axisX="0" axisY="1" axisZ="0" />
112    <translate X="31.12" Y="0" Z="0" />
113    <scale     X="1.8" Y="1.8" Z="1.8" /> <!-- " 400% " -->
114    <color     R="220" G="241" B="245" />
115    <models>
116        <model file="sphere.3d" />
117    </models>
118</group>
119<group> <!-- "Neptune" -->
120    <rotate    angle="10" axisX="0" axisY="1" axisZ="0" />
121    <translate X="35.32" Y="0" Z="0" />
122    <scale     X="1.4" Y="1.4" Z="1.4" /> <!-- " 388% " -->
123    <color     R="57" G="182" B="247" />
124    <models>
125        <model file="sphere.3d" />
126    </models>
127</group>
128<group> <!-- "Pluto" -->
129    <rotate    angle="50" axisX="0" axisY="1" axisZ="1" />
130    <translate X="37.92" Y="0" Z="0" />
131    <scale     X="0.2" Y="0.2" Z="0.2" /> <!-- " 20% " -->
132    <color     R="221" G="196" B="175" />
133    <models>
134        <model file="sphere.3d" />
135    </models>
136</group>
137<scale     X="5" Y="5" Z="5" /> <!-- " 10900% " -->
138<color     R="247" G="115" B="12" />
139<models>
140    <model file="sphere.3d" />
141</models>
142</group>
143</group>
144</scene>

```
