Universidade do Minho
Escola de Engenharia

Computação Gráfica

TRABALHO PRÁTICO

2ª Fase

Mestrado Integrado em Engenharia Informática

Fábio Quintas Gonçalves, a78793

Francisco José Moreira Oliveira, a78416

Raul Vilas Boas, a79617

Vitor Emanuel Carvalho Peixoto, a79175

Ano letivo 2017/2018

Março de 2018

ÍNDICE

1. INTRODUÇÃO

Após uma primeira fase, onde se desenvolveu um *Generator* capaz de gerar ficheiros com os pontos de uma dada figura, um *Engine* capaz de ler um XML onde esses ficheiros com os pontos são referenciados e desenhar, nesta segunda fase, é-nos pedido para aumentar as capacidades do ficheiro XML, adicionando transformações às figuras, seguindo um conjunto de regras (hierarquização, etc.).
As transformações especificadas são as seguintes:

- Translação
- Rotação
- Alteração da escala

Foram desenvolvidos ainda alguns extras, capazes de aumentar a fidelidade do objetivo final desta fase, que é obter um modelo estático do Sistema Solar.

2. SISTEMA SOLAR

Para além dos planetas e do Sol, como requisitado pelo enunciado, foram adicionados também outros corpos celestes, como as luas, anéis e as órbitas.

Todos estes objetos foram definidos numa hierarquia que facilitasse a leitura do ficheiro XML, pelo que fomos capazes de esboçar o seguinte organograma:

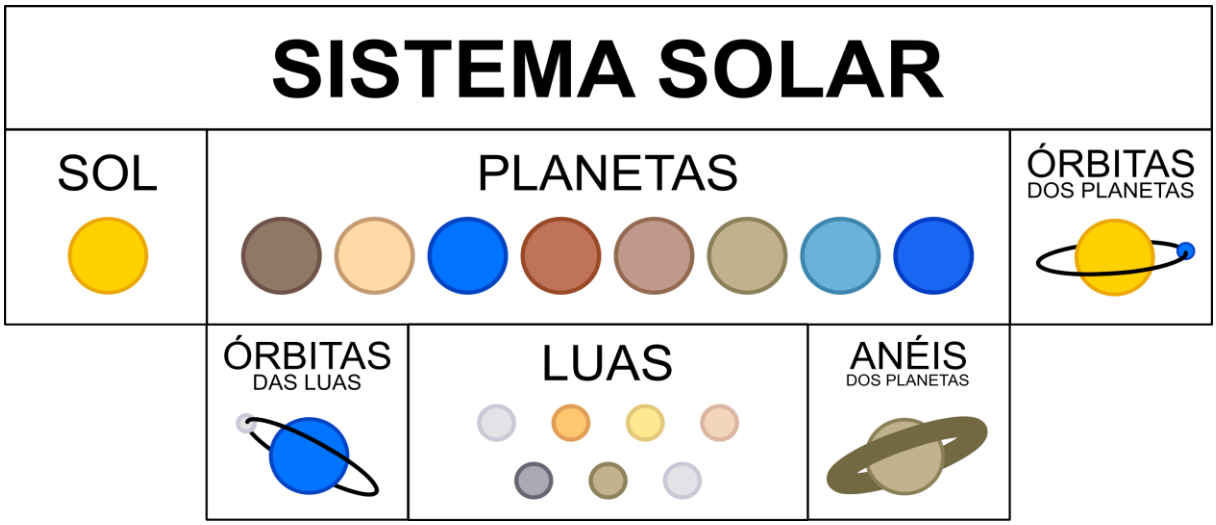


Figura 1 Organograma do Sistema Solar

Para além do Sol, definimos 8 planetas e as suas respetivas órbitas:

- Mercúrio;
- Vénus;
- Terra;
- Marte;
- Júpiter;
- Saturno;
- Úrano;
- Neptuno.

Dentro de cada planeta definimos luas e suas respetivas órbitas e os anéis dos planetas:

- Mercúrio;
- Vénus;
- Terra (Luas e respetivas órbitas: “Lua”);
- Marte;
- Júpiter (Luas e respetivas órbitas: “Io”; “Europa”; “Ganimedes” e “Calisto”);
- Saturno (Anéis; Luas e respetivas órbitas: “Titã”);
- Úrano;
- Neptuno (Luas e respetivas órbitas: “Tritão”).

Apenas representamos as maiores luas dos planetas gasosos, visto que alguns ultrapassam as 50 luas. Marte tem duas luas, mas são de dimensões reduzidas (≈ 20 km).

3. *ENGINE* E *PARSER*

3.1 Leitura dos ficheiros XML

Tal como na primeira fase, a *engine* carrega os ficheiros necessários para a renderização.

Essa renderização é efetuada através dos dados fornecidos no *scene.xml*, sendo lidos por um *parser*, recorrendo à biblioteca do *tinyxml2*.

O *scene.xml* é encarregado por indicar os ficheiros "*.3d*" gerados pelo *Generator*, onde são armazenadas as coordenadas dos pontos das figuras a gerar.

Contudo, nesta fase, ele irá ser capaz também de realizar transformações sobre os objetos antes de os desenhar, especificamente translações, rotações e escalas:

- **Translação**: recebe 3 parâmetros com o deslocamento nos eixos X, Y e Z respetivos;
- **Rotação**: recebe o ângulo de rotação e mais 3 parâmetros que indicam qual o eixo de rotação em causa;
- **Escala**: recebe 3 parâmetros com a escala que será aplicada aos eixos X, Y e Z respetivamente.

Toda especificação das transformações a aplicar a um determinado objeto estão hierarquizadas em grupos, sendo que se for aplicada uma transformação num dado grupo, todos os seus subgrupos irão herdar essa transformação.

Excerto do *scene.xml*.

```
1  <group a="Terra">
2      <rotate angle="23.5" axisX="0" axisY="0" axisZ="1" />
3      <group a="Lua">
4          <rotate angle="-30" axisX="0" axisY="0" axisZ="1" />
5      </group>
6  </group>
```

No caso acima, por exemplo, o objeto "Lua" irá sofrer uma rotação de 23,5° – 30° = -6,5° sobre o eixo dos Z visto ser um subgrupo da "Terra".

Para receber as diferentes transformações, foram adicionadas variáveis referentes às mesmas na classe *Figure*, nomeadamente as instâncias de *Coordinate* (que tem 4 *floats*: *x*, *y*, *z* e *ang*).

Classe que armazena as informações de um objeto.

```
1   class Figure {
2       public:
3           vector<vector<Coordinate>> figuras;
4           Coordinate rotate, translate, scale;
5   };
```

Estas transformações serão posteriormente aplicadas pela função *drawGroup*. Essa função inicia-se com um *glPushMatrix* e termina com um *glPopMatrix*, que permitem aplicar todas as transformações efetuadas dentro desse grupo a todos os subgrupos. Ainda dentro do *glPushMatrix*, logo após as transformações, invocamos a função *drawFigures* que desenha os objetos e a mesma função (recursividade) *drawGroup* para os subgrupos.

3.2 **Extras**

3.2.1 Cores

Para além da capacidade de reconhecer transformações no ficheiro *scene.xml* adicionamos a funcionalidade de coloração dos objetos.

Para tal adicionamos à classe *Figure*, uma instância de *Color* que regista os valores decimais de RGB.

No ficheiro *scene.xml*, os objetos onde a cor irá ser especificada, terão a seguinte linha adicionada:

```
<color R="0.9" G="0.9" B="0.9" />
```

Esta transformação tem o mesmo processo de aplicação que as anteriormente explicadas.

3.2.2 Luas

Adicionamos também as principais luas de alguns planetas, como será clarificado posteriormente. Estas foram definidas como subgrupos dos planetas, herdando as transformações definidas no seu supergrupo (mas sendo sujeitas às suas próprias transformações também).

3.2.3 Anéis

Os anéis dos planetas (Saturno e Úrano) foram descritos como subgrupos destes planetas.

3.2.4 Órbitas

As órbitas de todos os corpos celestes foram desenhadas. As órbitas dos planetas foram especificadas como um grupo dentro do sistema solar e por isso, não herdando qualquer transformação. As órbitas das luas foram especificadas dentro dos planetas respetivos, herdando as suas transformações (i.e., a órbita da Lua estará centrada na Terra).

A adição das órbitas trouxe também a necessidade de novas adições ao *Engine.*

De facto, foi necessário adicionar uma variável booleana à classe *Figure*, para referir se essa figura era uma órbita ou não, visto ser necessário distinguir as órbitas de todos os outros objetos desenhados até agora.

Isto reside no facto de todos os corpos celestes desenvolvidos até agora serem desenhados através de triângulos (*GL_TRIANGLES*), sendo que uma órbita terá de ser desenhada recorrendo a linhas (*GL_LINE_STRIP*).

Assim, tivemos de desenhar uma alternativa à *drawFigures*, visto que esta desenha apenas triângulos, e criamos a *drawOrbit* que permite desenhar linhas.

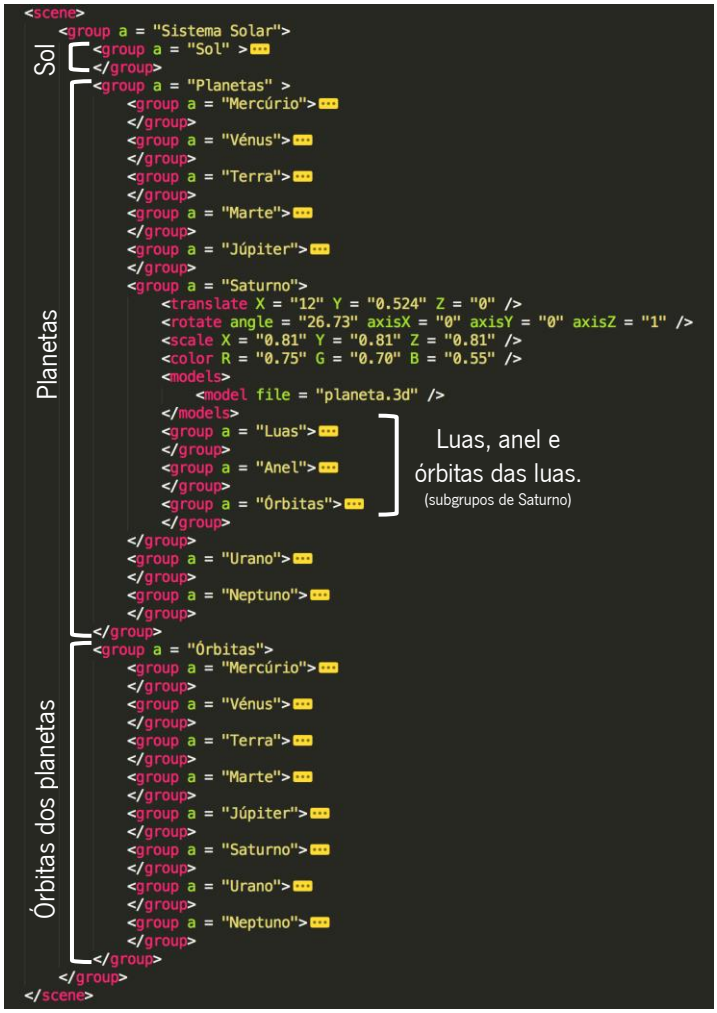


Figura 2 Versão final do scene.xml

4. *GENERATOR*

Dado o facto de termos adicionado extras em algumas tarefas, é necessário analisar a necessidade de desenhar mais objetos. As luas, planetas e Sol não irão necessitar de ser desenhados, visto serem uma esfera, que já havia sido desenvolvida na fase anterior.

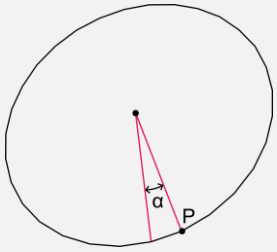
No entanto, as órbitas e os anéis não têm nenhum objeto desenvolvido na fase anterior, capaz de os representar, pelo que será necessário desenvolver estes dois.

4.1 **Órbitas**

Uma órbita será basicamente uma circunferência. Esta irá receber como argumentos o raio e o número de divisões.

$\alpha = 2*\pi + nDivisoes$

$P(x,y,z) = (raio*\sin(\alpha), 0, raio*\cos(\alpha))$



Assim, basta percorrer todos os pontos e escrever no ficheiro *.3d* as suas coordenadas. Estes pontos irão ser lidos na *Engine* e desenhados como uma linha (*GL_LINE_STRIP*).

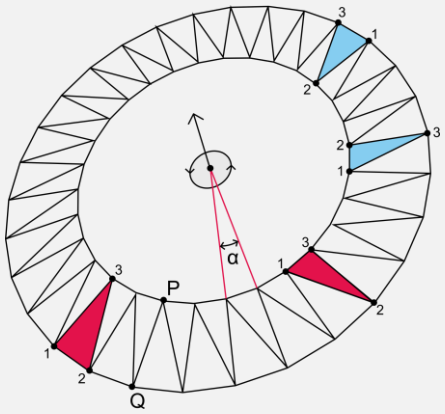
4.2 **Anéis**

Um anel será composto por 2 circunferências, sendo que o espaço entre estas duas é preenchido. A função recebe como argumentos os raios das 2 circunferências e o número de divisões. O método de cálculo dos pontos é o mesmo, sendo que muda apenas o modo de união entre estes (a ordem de escrita no ficheiro *.3d*).

$\alpha = 2*\pi + nDivisoes$

$P(x,y,z) = (raioInt*\sin(\alpha), 0, raioInt*\cos(\alpha))$

$Q(x,y,z) = (raioExt*\sin(\alpha), 0, raioExt*\cos(\alpha))$



Para desenhar o anel, usamos triângulos (*GL_TRIANGLES*). Para tal, temos de percorrer as circunferências 4 vezes: 2 para o lado de cima, e outras 2 para o lado de baixo.

Para cima, temos de respeitar a regra da mão direita, pelo que a ordem de escolha dos pontos é aquela que se vê nos triângulos a vermelho na figura acima.

Para baixo é o inverso e a ordem de escola dos pontos é aquela que se vê nos triângulos azuis.

Assim podemos demonstrar um extrato de um dos 4 ciclos, sendo que este descreve o primeiro dos triângulos vermelhos, a contar da esquerda.

Note que *pontos1* e *pontos2* são listas dos pontos da circunferência interior e exterior, respetivamente.

Ciclo *for* de um dos triângulos superiores do anel.

```
1    for (unsigned int i = 0; I < pontos1.size()-1; i++) {
2        fprintf(f, “%f %f %f\n”, pontos2[i].getX(), pontos2[i].getY(), pontos2[i].getZ());
3        fprintf(f, “%f %f %f\n”, pontos2[i+1].getX(), pontos2[i+1].getY(), pontos2[i+1].getZ());
4        fprintf(f, “%f %f %f\n”, pontos1[i].getX(), pontos1[i].getY(), pontos1[i].getZ());
5    }
```

5. CÂMARA DE VISUALIZAÇÃO

Para ajudar no desenvolvimento desta fase, também fizemos alterações à câmara de modo a nos permitir uma melhor visualização do modelo que estamos a desenvolver. Para isso, desenvolvemos uma câmara FPS.

Esta nova câmara resulta de modificações à câmara que tínhamos originalmente para a 1ª Fase deste projeto, pela qual as noções de coordenadas esféricas se mantêm iguais.

A principal diferença reside no facto de pretendemos que o centro da esfera não se situe apenas na origem, mas sim que consigamos alterá-la de modo a se mover ao longo do nosso modelo. Assim, a nossa câmara, que se situa na superfície da esfera e aponta para o centro, poderá visualizar partes diferentes do modelo como nós desejarmos.

Para este fim, precisamos de saber o que alterar na chamada de *gluLookAt* na *renderScene*.

```
1   gluLookAt ( px, py, pz,      // camera position
2               dx,  dy,  dz,    // look at point
3               ux, uy, uz);     // "up vector" (0.0f, 1.0f, 0.0f)
```

As variáveis *px*, *py* e *pz* correspondem às coordenadas de origem da esfera, por isso se quisermos mover a câmara temos de adicionar a estes três pontos, um vetor de movimento. Se pretendemos que a câmara se mova ao longo do eixo *XX,* temos de somar a *px* um valor correspondente à distância que nos pretendemos mover. É importante notar que também precisamos de alterar os valores correspondentes a *look at point* para a câmara apontar para o centro da esfera.

A ideia implementada foi ao pressionar as teclas designadas para tal efeito, alterar os valores do *look at point* e na função *gluLookAt* alterar os valores *px, py, pz* para a soma entre *px + dx*, etc.

```
1   gluLookAt ( px+dx, py+dy, pz+dz,
2               dx,     dy,     dz,
3               0.0f,  1.0f,  0.0f);
```

Contudo, temos ainda mais um problema a tratar, que é como independentemente do ângulo onde estamos (variável "*alpha*" que é usado para rodar a câmara ao longo da superfície da esfera), conseguimos com que o tecla que estará destinada a representar a deslocação para a direita assim o faça para qualquer ângulo.

Para resolver este problema, ao invés de adicionar um valor a *dx* para obtermos a movimentação pretendida efetuamos a seguinte transformação:

$$(P_x + \sin(\alpha), P_y, P_z + \cos(\alpha))$$

Onde os três parâmetros acima correspondem aos valores a somar a *dx, dy* e *dz.*

Não é necessário efetuar nada em particular no eixo yy uma vez que as movimentações verticais não serão afetadas pelo ângulo "alpha".

Assim, temos a certeza que independentemente do ângulo onde estamos posicionados, ao pressionar a tecla correspondente ao movimento para a direita, temos de facto um movimento para a direita de onde estamos posicionados.

As teclas mapeadas para os movimentos foram as seguintes:

'w' e 'W' para movimentar a câmara para a frente.
's' e 'S' para movimentar a câmara para trás.
'a' e 'A' para movimentar a câmara para a esquerda.
'd' e 'D' para movimentar a câmara para a direita.

'8' para, mantendo o mesmo centro de visualização, mover a câmara para cima.
'2' para, mantendo o mesmo centro de visualização, mover a câmara para baixo.
'4' para, mantendo o mesmo centro de visualização, mover a câmara para a esquerda.
'6' para, mantendo o mesmo centro de visualização, mover a câmara para a direita.

'LEFT_BUTTON' e 'KEY_UP' para aproximar a câmara do centro.
'RIGHT_BUTTON' e 'KEY_DOWN' para afastar a câmara do centro.

'i' e 'I' para mudar o modo de visualização do modelo para *GL_FILL*.
'o' e 'O' para mudar o modo de visualização do modelo para *GL_LINE.*
'p' e 'P' para mudra o modo de visualização do modelo para *GL_POINT.*

Foram atribuídas pares de teclas de movimentação da câmara para acomodar a rapidez com que queremos percorrer o modelo, sendo que a segunda tecla atribuída nestes pares resulta numa maior alteração nas variáveis correspondentes.

6. RESULTADO FINAL

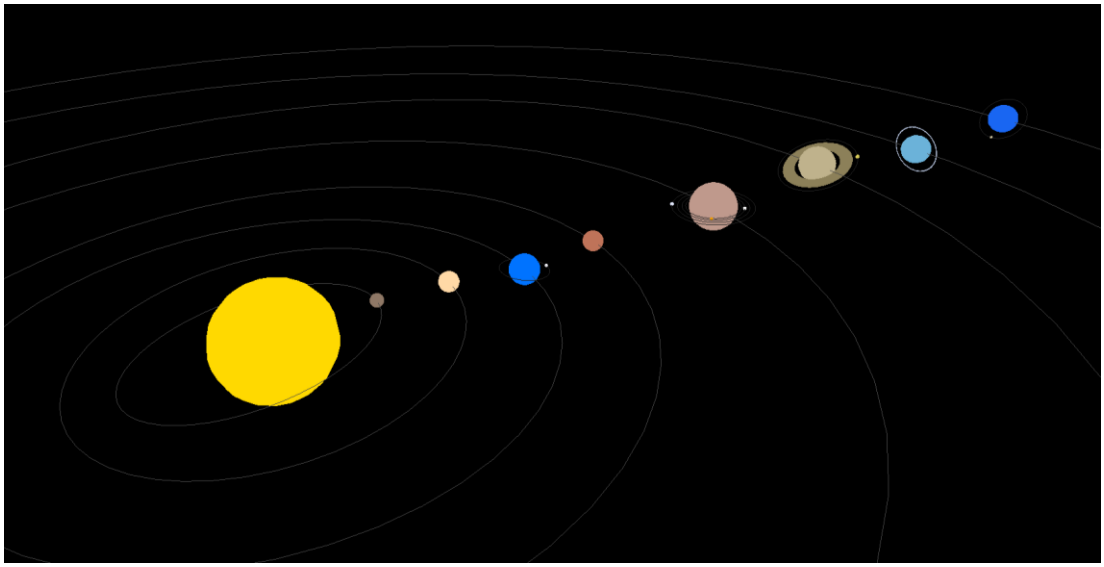


Figura 3 O Sistema Solar.

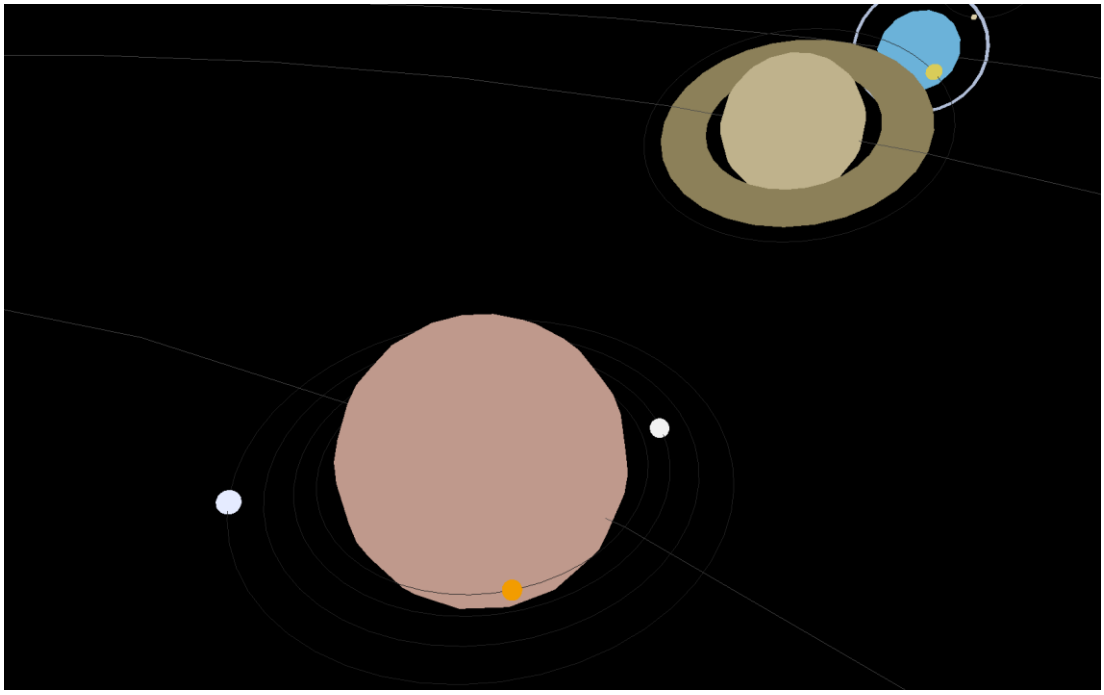


Figura 3 Luas de Júpiter, lua e anel de Saturno, anel de Úrano.

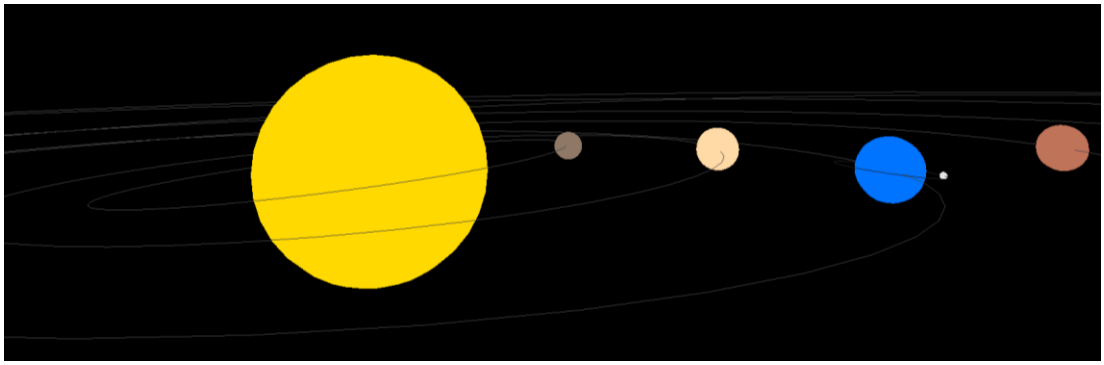


Figura 5 Órbitas elípticas e não complanares.

7. CONCLUSÕES E TRABALHO FUTURO

Esta segunda fase permitiu absorver conhecimento relativamente à utilização do XML como meio de desenhar figuras mais facilmente, visto que não requer conhecimento muito profundo acerca de grafismo e quase nenhum acerca de *OpenGL*.

Permitiu também adquirir algum conhecimento adicional acerca do funcionamento da câmara.

Esta fase não exige muito manuseamento em *OpenGL*, uma vez que a sua implementação nos parece maioritariamente pronta.

No entanto, indiretamente acabamos por adquirir algum conhecimento em *OpenGL*, uma vez que desenvolvemos as ferramentas de transformação dos objetos e definimos a "base de dados" dos astros do Sistema Solar. Isto permite-nos manipular ângulos, translações, etc. e, consequentemente, obter alguma prática nesta medida.