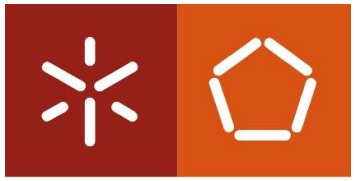


Universidade do Minho

Licenciatura em Ciências da Computação

Computação Gráfica (3º ano de Curso)

Fase 4

Relatório de Desenvolvimento

Diogo Fernandes (A87968) | Luís Guimarães (A87947) | Ivo Lima (A90214)

30 de maio de 2021

Conteúdo

Capítulo 1

Introdução

Uma vez que esta é a última fase foi pedido uma série de novas implementações. O *generator* passou a gerar, para além das coordenadas dos pontos, as normais e as as coordenadas de textura.

Para conseguirmos tirar partido dessa nova informação incluída nos modelos 3D, tivemos de ativar as luzes e as texturas no OpenGL e definir as propriedades de cada modelo, para assim utilizarmos este novo tipo de coloração.

Posto isto, estabelecemos um conjunto de tarefas que incidiram tanto sobre o *generator* como a *engine* para que esta possa suportar os requisitos indicados no enunciado e outros definidos por nós:

Atualizações no *generator*:

1. Gerar as normais e as coordenadas de textura dos modelos atualizados;

Atualizações na *engine*:

1. Atualizar o parser de modo a dar suporte aos novos tipos de configuração xml e atualizar/criar estruturas de dados para guardar as propriedades das luzes, materiais e texturas;

2. Introduzir uma câmara fps, assim como outras features para facilitar o debug/visualização do ambiente;

Capítulo 2

Atualização do *generator*

Com a introdução desta fase, percebemos que para se dar a criação das normais e das coordenadas de textura apenas teriamos de fazer uma leve atualização na estratégia adotada até ao momento, ou seja, após a geração dos três pontos que definem um dos três vértices do triângulo serão introduzidas três coordenadas para as normais e outras duas que representarão os pontos da textura.

2.1 Nova configuração de geração dos pontos

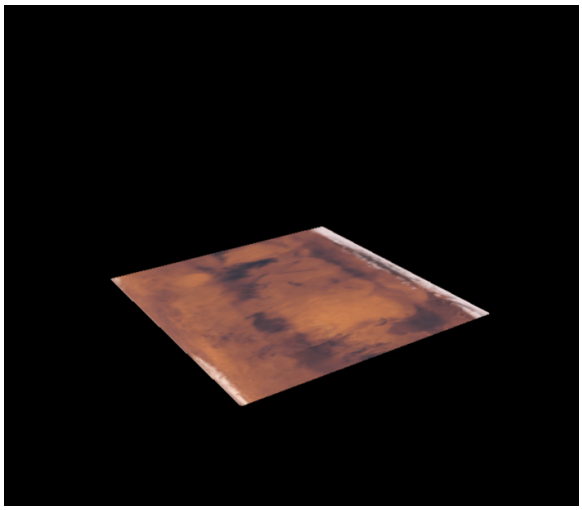
Tendo isto posto, segue-se nas próximas secções uma breve descrição daquilo que foi alterado através da apresentação do código para gerar as normais e as coordenadas de textura.

2.1.1 *Plane*

Neste primeiro caso um dos mais simples foram acrescentadas as linhas apresentadas a seguir:

```
...
<< 0 << " " << 1 << " " <<
0 << " "
<< 1 << " " << 0 << "\n";
...
<< 0 << " " << 1 << " " <<
0 << " "
<< 1 << " " << 1 << "\n";
...
<< 0 << " " << 1 << " " <<
0 << " "
<< 0 << " " << 1 << "\n";

...
<< 0 << " " << 1 << " " <<
0 << " "
<< 1 << " " << 0 << "\n";
...
<< 0 << " " << 1 << " " <<
0 << " "
<< 0 << " " << 1 << "\n";
...
<< 0 << " " << 1 << " " <<
0 << " "
<< 0 << " " << 0 << "\n";

...
```

```
  << 0 << " " << -1 << " "
<< 0 << " "
  << 1 << " " << 1 << "\n";
  ...
  << 0 << " " << -1 << " "
<< 0 << " "
  << 1 << " " << 0 << "\n";
  ...
  << 0 << " " << -1 << " "
<< 0 << " "
  << 0 << " " << 1 << "\n";

  ...
  << 0 << " " << -1 << " "
<< 0 << " "
  << 0 << " " << 1 << "\n";
  ...
  << 0 << " " << -1 << " "
<< 0 << " "
  << 1 << " " << 0 << "\n";
  ...
  << 0 << " " << -1 << " "
<< 0 << " "
  << 0 << " " << 0 << "\n";
```



2.1.2 *Box*

A *Box* já começou a exigir um pouco mais de racíocínio e fez com que tivéssemos de acertar de alguns pontos e acabou ficando assim:

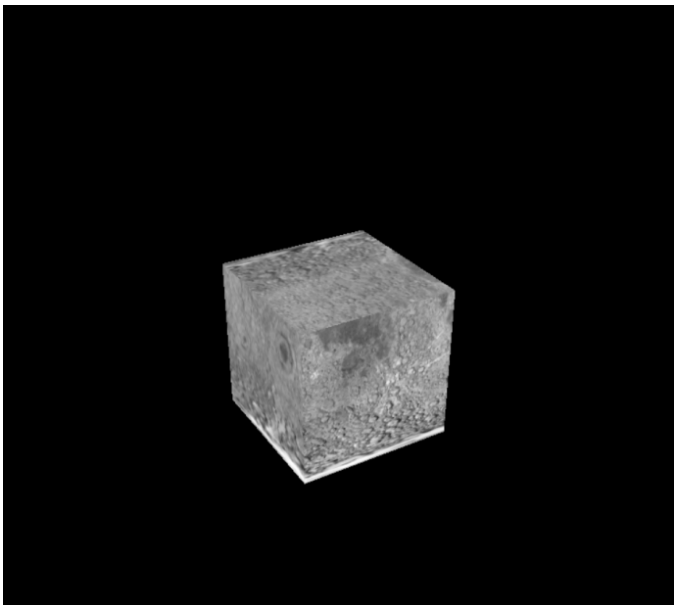
```
    ...
    << 0 << " "
<< 1 << " " << 0
 << " "
    << 0.3 << " "
 << 0.5 << "\n";
    ...
    << 0 << " "
<< 1 << " " << 0
 << " "
    << 0.3 << " "
 << 1 << "\n";
    ...
    << 0 << " "
<< 1 << " " << 0
 << " "
    << 0 << " "
<< 1 << "\n";


    ...
    << 0 << " "
<< 1 << " " << 0
 << " "
    << 0.3 << " "
 << 0.5 << "\n";
    ...
    << 0 << " "
<< 1 << " " << 0
 << " "
    << 0 << " "
<< 1 << "\n";
    ...
    << 0 << " "
<< 1 << " " << 0
 << " "
    << 0 << " "
<< 0.5 << "\n";


    ...
    << 0 << " "
<< -1 << " " <<
0 << " "
    << 1 << " "
<< 0.5 << "\n";
    ...
    << 0 << " "
<< -1 << " " <<
0 << " "
```

```
    << 1 << " "
<< 1 << "\n";
    ...
    << 0 << " "
<< -1 << " " <<
0 << " "
    << 0.6 << " "
 << 0.5 << "\n";


    ...
    << 0 << " "
<< -1 << " " <<
0 << " "
    << 0.6 << " "
 << 0.5 << "\n";
    ...
    << 0 << " "
<< -1 << " " <<
0 << " "
    << 1 << " "
<< 1 << "\n";
    ...
    << 0 << " "
<< -1 << " " <<
0 << " "
    << 0.6 << " "
 << 1 << "\n";


    ...
    << 1 << " "
<< 0 << " " << 0
 << " "
    << 0.6 << " "
 << 0.5 << "\n";
    ...
    << 1 << " "
<< 0 << " " << 0
 << " "
    << 0.3 << " "
 << 1 << "\n";
    ...
    << 1 << " "
<< 0 << " " << 0
 << " "
    << 0.3 << " "
 << 0.5 << "\n";


    ...
    << -1 << " "
<< 0 << " " << 0
 << " "
    << 0 << " "
<< 0.5 << "\n";
    ...
    << -1 << " "
<< 0 << " " << 0
 << " "
    << 0 << " "
<< 0 << "\n";
    ...
    << -1 << " "
<< 0 << " " << 0
 << " "
    << 0.3 << " "
 << 0.5 << "\n";


    ...
    << -1 << " "
<< 0 << " " << 0
 << " "
    << 0.3 << " "
 << 0.5 << "\n";
    ...
    << -1 << " "
<< 0 << " " << 0
 << " "
```

```
    << 0 << " "
<< 0 << "\n";
    ...
    << -1 << " "
<< 0 << " " << 0
 << " "
    << 0.3 << " "
 << 0 << "\n";

    ...
    << 0 << " "
<< 0 << " " << 1
 << " "
    << 0.6 << " "
 << 0 << "\n";
    ...
    << 0 << " "
<< 0 << " " << 1
 << " "
    << 0.6 << " "
 << 0.5 << "\n";
    ...
    << 0 << " "
<< 0 << " " << 1
 << " "
    << 0.3 << " "
 << 0.5 << "\n";

    ...
```

```
    << 0 << " "
<< 0 << " " << 1
 << " "
    << 0.6 << " "
 << 0 << "\n";
    ...
    << 0 << " "
<< 0 << " " << 1
 << " "
    << 0.3 << " "
 << 0.5 << "\n";
    ...
    << 0 << " "
<< 0 << " " << 1
 << " "
    << 0.3 << " "
 << 0 << "\n";

    ...
    << 0 << " "
<< 0 << " " <<
-1 << " "
    << 0.6 << " "
 << 0.5 << "\n";
    ...
    << 0 << " "
<< 0 << " " <<
-1 << " "
```

```
    << 0.6 << " "
 << 0 << "\n";
    ...
    << 0 << " "
<< 0 << " " <<
-1 << " "
    << 1 << " "
<< 0.5 << "\n";

    ...
    << 0 << " "
<< 0 << " " <<
-1 << " "
    << 1 << " "
<< 0.5 << "\n";
    ...
    << 0 << " "
<< 0 << " " <<
-1 << " "
    << 0.6 << " "
 << 0 << "\n";
    ...
    << 0 << " "
<< 0 << " " <<
-1 << " "
    << 1 << " "
<< 0 << "\n";
```



2.1.3 *Sphere*

Nesta figura os cálculos ficaram da seguinte maneira:

```
    ...
  << (radius * cos(next_b)
* sin(next_a))/radius << "
" << (radius * sin(next_b))
/radius << " " << (radius *
 cos(next_b) * cos(next_a))
/radius << " "
  << next_a / pi_mul_2 << "
 " << (next_b + pi_div_2)/
M_PI << "\n";
```

```
    ...
  << (radius * cos(next_b)
* sin(next_a))/radius << "
" <<  (radius * sin(next_b)
)/radius << " " << (radius
* cos(next_b) * cos(next_a)
)/radius << " "
  << a / pi_mul_2 << " " <<
 (next_b + pi_div_2)/ M_PI
<< "\n";
```

```
    ...
  << (radius * cos(next_b)
* sin(next_a))/radius << "
" << (radius * sin(next_b))
/radius << " " << (radius *
 cos(next_b) * cos(next_a))
/radius << " "
  << next_a / pi_mul_2 << "
 " << (b + pi_div_2) / M_PI
 << "\n";
```

```
    ...
```

```
  << (radius * cos(next_b)
* sin(next_a))/radius << "
" <<  (radius * sin(next_b)
)/radius << " " << (radius
* cos(next_b) * cos(next_a)
)/radius << " "
  << next_a / pi_mul_2 << "
 " << (next_b + pi_div_2)/
M_PI << "\n";
```

```
    ...
  << (radius * cos(next_b)
* sin(next_a))/radius << "
" <<  (radius * sin(next_b)
)/radius << " " << (radius
* cos(next_b) * cos(next_a)
)/radius << " "
  << a / pi_mul_2 << " " <<
 (next_b + pi_div_2)/ M_PI
<< "\n";
```

```
    ...
  << (radius * cos(next_b)
* sin(next_a))/radius << "
" <<  (radius * sin(next_b)
)/radius << " " << (radius
* cos(next_b) * cos(next_a)
)/radius << " "
  << a / pi_mul_2 << " " <<
 (b + pi_div_2) / M_PI << "
\n";
```

2.1.4 *Cone*

Por fim foi adicionado o seguinte:

```
    ...
  << 0 << " " << -1 << " "
<< 0 << " "
  << 1/6 << " " << 1/6 << "
\n";
    ...
  << 0 << " " << -1 << " "
<< 0 << " "
  << 1/6 * sin(next_a) << "
 " << 1/6 * cos(next_a) <<
"\n";
    ...
  << 0 << " " << -1 << " "
<< 0 << " "
  << 1/6 * sin(a) << " " <<
 1/6 * cos(a) << "\n";
   ...


    ...
  << cos(atan(radius/height
))*cos(next_a) << " " <<
sin(atan(radius/height)) <<
 " " << cos(atan(radius/
height))*sin(next_a) << " "
  << sin(next_a) * ((height
 - next_h) / height) * cos(
next_a)  << " " << next_h/
height << "\n";
    ...
  << cos(atan(radius/height
))*cos(a) << " " << sin(
atan(radius/height)) << " "
 << cos(atan(radius/height)
)*sin(a) << " "
  << sin(a) * ((height -
next_h) / height) * cos(a)
 << " " << next_h/height <<
 "\n";
    ...
  << cos(atan(radius/height
))*cos(next_a) << " " <<
sin(atan(radius/height)) <<
 " " << cos(atan(radius/
height))*sin(next_a) << " "
  << sin(next_a) * ((height
 - next_h) / height) * cos(
next_a)  << " " << next_h/
height << "\n";


    ...
  << cos(atan(radius/height
))*cos(a) << " " << sin(
atan(radius/height)) << " "
 << cos(atan(radius/height)
)*sin(a) << " "
  << sin(a) * ((height -
next_h) / height) * cos(a)
 << " " << next_h/height <<
 "\n";
    ...
  << cos(atan(radius/height
))*cos(a) << " " << sin(
atan(radius/height)) << " "
 << cos(atan(radius/height)
)*sin(a) << " "
  << sin(a) * ((height -
next_h) / height) * cos(a)
 << " " << next_h/height <<
 "\n";
    ...
  << cos(atan(radius/height
))*cos(next_a) << " " <<
sin(atan(radius/height)) <<
 " " << cos(atan(radius/
height))*sin(next_a) << " "
  << sin(next_a) * ((height
 - next_h) / height) * cos(
next_a)  << " " << next_h/
height << "\n";
```

Capítulo 3

Atualização da *engine*

3.1 Nova configuração do xml

O ficheiro de configuração do *xml* que temos vindo a utilizar deu suporte às luzes, que definirão a nossa *scene* que podem ser definidas de 3 tipos distintos: ***spot***, ***point*** e ***directional***.

A definição das luzes deverá aparecer da seguinte forma:

As ***spotlight*** simulam uma luz parecida aquela vinda de uma lanterna, sendo que para obtermos tal funcionalidade devemos especificar um ponto, uma direção, a abertura da luz (cutoff) e um expoente para definir a sua intensidade.

```
<lights>
    <light type="SPOT" posX="..." posY="..." posZ="..."
                       dirX="..." dirY="..." dirZ="..."
                       cutoff="..." exponent="..." />
<lights>
```

No caso das luzes de tipo ***point*** que são definidas num ponto e emitem em todas as direções. Terá a seguinte definição:

```
<lights>
    <light type="POINT" posX="..." posY="..." posZ="..." />
</lights>
```

Já as ***directional*** *lights* não estão definidas num ponto pois seguem uma dada direção.

```
<lights>
   <light type="DIRECTIONAL" dirX="..." dirY="..." dirZ="..." />
</lights>
```

Os modelos passam a ser desenhados com cor, sendo possível especificar cada tipo de cor no modelo RGB:

```
< model file = "..." ambiR="..." ambiG="..." ambiB="..."
                     emisR="..." emisG="..." emisB="..."
                     specR="..." specG="..." specB="..."
                     diffR="..." diffG="..." diffB="..." />
```

Podem também ser desenhados modelos com uma determinada textura:

```
< model file = "..." texture="..." />
```

Ou com cor e textura:

```
< model file = "..." ambiR="..." ambiG="..." ambiB="..."
                     emisR="..." emisG="..." emisB="..."
                     specR="..." specG="..." specB="..."
                     diffR="..." diffG="..." diffB="..."
                     texture="..." />
```

3.2 Implementação

Para que a iluminação e a texturização sejam possíveis, é necessário adicionar as seguintes linhas de código à função main:

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);

glEnable(GL_LIGHTING);
glEnable(GL_TEXTURE_2D);
```

Foram criados dois *vectors*, um para armazenar as normais e outro para armazenar as coordenadas de textura.

Estes vectors são preenchidos na função $readXML\_aux$ ao percorrer os ficheiros *xxx.3d* associados aos modelos, que contêm, para cada ponto, as suas coordenadas, as normais e as suas coordenadas de textura.

Foram também criados mais dois buffers, um que armazena as normais e outro que armazena as coordenadas de textura.

Ficamos então com três buffers na totalidade, que são gerados da seguinte forma:

```
glGenBuffers(3, buffers);
glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
glBufferData(
  GL_ARRAY_BUFFER, sizeof(float) * v.size(), v.data(),
    GL_STATIC_DRAW);

glBindBuffer(GL_ARRAY_BUFFER, buffers[1]);
glBufferData(
  GL_ARRAY_BUFFER, sizeof(float) * n.size(), n.data(),
    GL_STATIC_DRAW);

glBindBuffer(GL_ARRAY_BUFFER, buffers[2]);
glBufferData(
  GL_ARRAY_BUFFER, sizeof(float) * t.size(), t.data(),
    GL_STATIC_DRAW);
```

Estes buffers são depois ativados na função *renderScene* da seguinte forma:

```
glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
glVertexPointer(3, GL_FLOAT, 0, 0);

glBindBuffer(GL_ARRAY_BUFFER, buffers[1]);
glNormalPointer(GL_FLOAT, 0, 0);

glBindBuffer(GL_ARRAY_BUFFER, buffers[2]);
glTexCoordPointer(2, GL_FLOAT, 0, 0);
```

3.2.1 Luzes

Foram definidas duas *structs* que armazenam as informações sobre as luzes. LIGHT é usada para os tipos POINT e DIRECTIONAL, enquanto SPOTLIGHT é usada para o tipo SPOT. A distinção entre POINT e DIRECTIONAL está no último argumento de pos (1.0 se for POINT e 0.0 se for DIRECTIONAL).

```
struct LIGHT {
    int n;
    float pos[4];
};
struct SPOTLIGHT {
    int n;
    float pos[4];
    float cutoff[1];
    float exponent[1];
    float spotDir[3]; };
```

Foram criados dois *vectors* que armazenam as informações sobre as luzes.

```
vector<LIGHT> lights;
vector<SPOTLIGHT> spotlights;
```

Na função recursiva $readXML\_aux$, que trata de percorrer o ficheiro xmlconfig, foi adicionado um caso para "lights"que trata de armazenar a informação sobre as luzes na estrutura de dados correspondente.

As luzes são depois ativadas na *renderScene* com recurso às funções glEnable e glLightfv do openGL.

3.2.2 Cores

Foram adicionadas as seguintes variáveis à struct FIGURE:

```
float dif[4];
float amb[4];
float emi[4];
float spe[4];
float shi;
```

Na função recursiva $readXML\_aux$, são adicionados à struct FIGURE correspondente os valores RBA das cores difusa, ambiente, emissiva, especular e brilho. No caso de não ter sido definida alguma das cores, o seu valor será o default do openGL.

Estes valores são depois associados ao modelo na renderScene, com recurso à função glMaterialfv.

3.2.3 Texturas

Foi adicionada a seguinte variável à struct FIGURE:

```
GLuint texture;
```

Foi criada a função *loadTexture* que recebe uma string como argumento e carrega e retorna a textura correpondente.

Na função $readXML\_aux$, no caso de ser especificado um ficheiro de textura, esta é carregada com recurso à função *loadTexture* e guardada na struct FIGURE correspondente.

A textura será depois ativada na renderScene antes de desenhar cada modelo através de glBindTexture. No caso de não ter sido especificado um ficheiro de textura, glBindTexture receberá 0 como argumento o que corresponde a não desenhar textura.

3.3 Outras implementações

3.3.1 CMakeLists

Uma vez que o nosso projeto começou a crescer e a quantidade de *features* também resolvemos gastar um pouco do nosso tempo a atualizar o *CMakeList*.

Para além da óbvia verificação e cópia da *DevIL.lib* da pasta *TOOLKITS_FOLDER* resolvemos anular o trabalho de toda a vez que faziamos o *Configure* e *Generate* no programa *CMake* ter de inserir manualmente na pasta *build* os ficheiros com as figuras do tipo *xxx.3d*, as texturas com a assinatura *xxx.jpg* e ainda o *xmlconf.xml*. Portanto por questões de organização esses ficheiro devem estar em pastas para que as suas diretivas possam ser passadas ao programa, a nova pasta *FIGTEX_FOLDER* deverá conter duas subpastas a *textures* bem como a *figs*, já o *XML_FOLDER* precisará conter uma subpasta *config* com um ficheiro nomeado de *xmlconf.xml*.

```
...

message(STATUS "Xml_DIR set to: " ${XML_FOLDER})
set(XML_FOLDER "" CACHE PATH "Path to XML folder")

message(STATUS "FigTex_DIR set to: " ${FIGTEX_FOLDER})
set(FIGTEX_FOLDER "" CACHE PATH "Path to FIGTEX folder")

...

if (EXISTS "${XML_FOLDER}")
    file(COPY ${XML_FOLDER}/config/xmlconf.xml DESTINATION ${
    CMAKE_BINARY_DIR})
endif(EXISTS "${XML_FOLDER}")

if (EXISTS "${FIGTEX_FOLDER}")
       file(COPY ${FIGTEX_FOLDER}/textures/ DESTINATION ${
    CMAKE_BINARY_DIR})
       file(COPY ${FIGTEX_FOLDER}/figs/ DESTINATION ${
    CMAKE_BINARY_DIR})
endif(EXISTS "${FIGTEX_FOLDER}")
```

O código escrito pede as diretivas das pastas para que possa copiar todos os ficheiros das suas subpastas e os cole na *CMAKE_BINARY_DIR* que é a pasta *build* gerada pelo *CMake*.

3.3.2 Câmara fps

Embora este não seja um dos aspetos mais importantes do trabalho resolvemos dedicar uns parágrafos à nossa implementação de uma câmara *fps*.

O raciocínio passa por considerar que a câmara possui uma posição P e um ponto para onde está a olhar e através desse conhecimento, criar o vetor da direção para onde a mesma está a olhar. Com esse vetor podemos reproduzir o movimento para a frente e para trás (teclas $\uparrow$ e $\downarrow$) relativamente à direção do olhar.

Já o movimento lateral pode ser obtido calculando o vetor resultante do up com a direção do olhar multiplicando o resultado por -1 ou 1 caso seja para a esquerda ou direita (teclas $\leftarrow$ e $\rightarrow$).

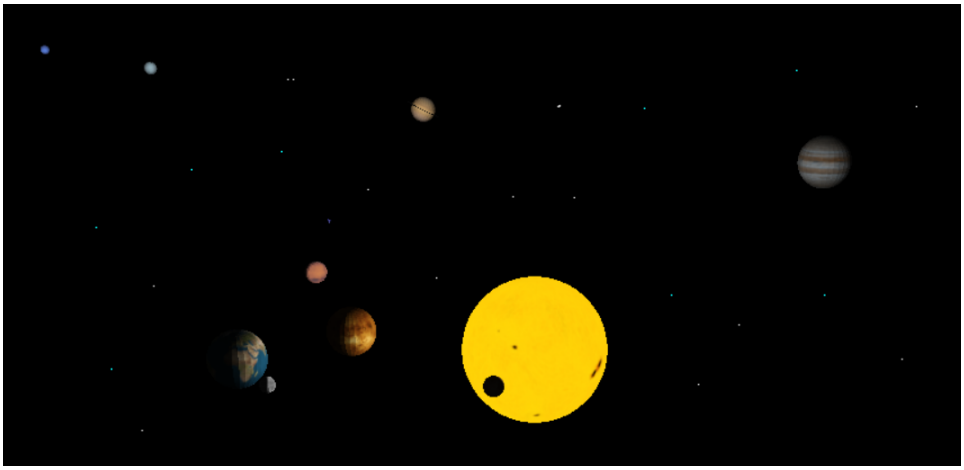
A mesma também consegue olhar à sua volta e rodar sobre o seu próprio eixo utilizando os ângulo alfa e beta para definir a posição do olhar com coordenadas esféricas (movimento de *drag* com o rato).

As variáveis de controlo da câmara bem como a sua aplicação encontra-se definida em na *engine.cpp*.

Capítulo 4

Resultado Final

Conseguimos portanto nesta última fase juntar tudo o que aprendemos o que culminou na criação deste magnífico Sistema Solar.



Capítulo 5

Conclusão

Nesta última fase, o desenvolvimento faseado que o nosso grupo tem vindo a estabelecer termina, resultando num software com uma flexível configuração de novas *features* em xml tirando partido do processamento que a API do OpenGL nos proporciona. Com isto queremos dizer que embora o resultado final seja muito importante, consideramos um bem maior o conseguir e conhecimento adquirido em cada uma das fases. O que deu uma oportunidade de aplicar os conceitos teóricos e comprovar o seu funcionamento através da formalização de código e algoritmos de geração de modelos, a implementação do movimento dos mesmos, as luzes, texturas,...

Em forma de resumo podemos dizer que esta última fase revelou-se a mais enriquecedora do projeto, pois atribuiu lhe um aspeto visual próprio ao nosso Sistema Solar através da implementação de texturas próprias para cada planeta e a implantação/definição do Sol como a única fonte de luz emissiva de todo o sistema. Concluimos então que todos os objetivos para este projeto foram cumpridos na totalidade, no entanto, a nossa *engine* ainda poderia continuar a sofre modificações e aprimorações para possibilitar a introdução de cada vez mais conceitos/funcionalidades novas.