Universidade do Minho
Escola de Engenharia

Computação Gráfica
Fase 4 - Grupo 11
2022/2023

Afonso Xavier Cardoso Marques a94940
Ana Filipa da Cunha Rebelo a90234
Tomás Cardoso Francisco a93193
Simão Paulo da Gama Castel-Branco e Brito a89482

Junho 2023




Índice

List of Figures

1 Introdução

A quarta e última fase do projeto, é proposta a continuação do desenvolvimento do sistema solar num cenário 3D. Para tal foi-nos pedida a adição das texturas em todos os elementos do sistema solar, e ainda, toda a iluminação de modo a obter uma representação mais realista das cenas.

Assim, foi necessário efetuar algumas alterações no Generator e Engine. No Generator, para cada primitiva, passaram a ser calculados os pontos de textura e vetores normais para serem escritos no ficheiro .3d. No Engine foram efetuadas alterações de modo a ser possível realizar a leitura dos novos ficheiros 3d e também para suportar a aplicação de texturas e iluminação às primitivas. Além disso, tivemos que refazer todo o trabalho desenvolvido para o Engine na fase anterior dado que não conseguimos resolver o problema que nos estava a dar.

No presente relatório, apresentamos e explicamos em detalhe todas as funcionalidades pedidas no enunciado de modo a ilustrar o raciocínio usado.

2 Generator

Para esta fase começamos por calcular as coordenadas de textura e das normais de cada vértice de cada primitiva. Deste modo, foi possível escrever no ficheiro 3d esses valores calculados para a correta implementação das texturas e da iluminação no que diz respeito ao Generator. Estas alterações foram realizadas nas respetivas funções do Generator e são explicadas em seguida em mais detalhe.

2.1 Cálculos nas Primitivas

2.1.1 Plano

O cálculo da normal de um plano é trivial, dado que este se encontra no plano XZ, o vetor normal superior será (0, 1, 0) para todos os pontos da face voltada para cima e (0, -1, 0) para todos os pontos da face voltada para baixo.

De forma a manter o código o mais inalterado possivel, a estratégia geral para o cálculo das coordenadas de textura e das normais de cada vértice nesta figura passou por adicionar à função *generatePlane_Texture_Normal* a funcionalidade de, para cada vértice calculado, escrever no ficheiro 3d as coordenadas X, Y e Z do vértice juntamente com as coordenadas de textura, e as coordenadas da normal do vértice (0.0f, 1.0f, 0.0f), onde a normal aponta para cima ao longo do eixo Y.

É importante observar que esta implementação assume uma superfície plana e define as coordenadas normais como (0.0f, 1.0f, 0.0f) para todos os vértices.

Assumindo que **z_min = x_min = - comprimento do lado do plano / 2**; a forma como se calcula as coordenadas de textura é a seguinte:

```
coordenada_textura_x = (x - x_min) / comprimento do lado do plano
coordenada_textura_z = (z - z_min) / comprimento do lado do plano
```

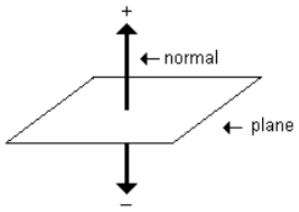

Figure 1: Normais do plano

2.1.2 Caixa

As alterações feitas na geração da figura da caixa andam à volta daquilo que foi feito para a geração do plano. A diferença está no facto de serem gerados seis planos em orientações diferentes o que significa que cada plano tem uma normal diferente e o cálculo das coordenadas das texturas repete o mesmo algoritmo das fórmulas anteriores para cada plano da caixa.

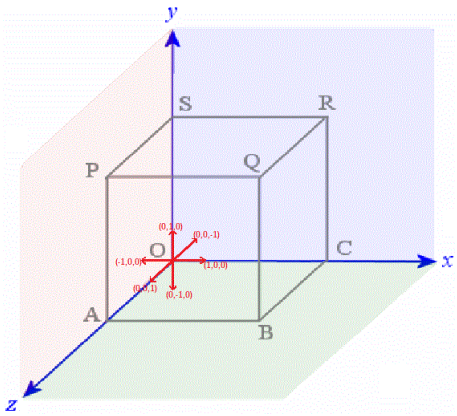

Figure 2: Normais da figura da caixa

2.1.3 Cone

Para superfícies curvas ou complexas, o cálculo das normais é diferente e dependente da geometria em si. Para a base do cone, que se encontra no plano XZ, o vetor normal é o mesmo para todos os pontos e é igual

a (0,-1,0). Por outro lado, o vetor normal para qualquer ponto da superfície lateral do cone pode ser obtido da seguinte forma:$(sin(\alpha), cos(atan(h/R)), cos(\alpha))$.

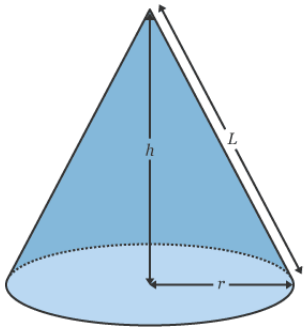

Figure 3: Normais do Cone

2.1.4 Esfera

No caso da esfera, as normais são facilmentes calculadas dado que é necessário apenas os ângulos presentes para o cálculo dos vértices. O vetor normal num dado ponto é igual ao vetor normalizado do vetor que vai do centro da esfera até ao ponto em questão.

Assim, a normal em qualquer ponto é a seguinte: $(cos(\beta)sin(\alpha), sin(\beta), cos(\beta)cos(\alpha))$. Onde β representa a variação das stacks e α a variação das slices.

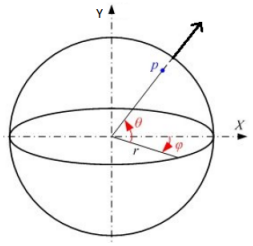

Figure 4: Normais da Esfera

2.2 Ficheiro .3d

Os ficheiros 3d resultantes, após a adição das normais e das texturas, contêm em cada linha informação sobre cada ponto. A figura abaixo corresponde à representação dos pontos no ficheiro .3d tendo três pontos diferentes. Os primeiros três valores são as coordenadas do vértice, os dois valores seguintes correspondem às coordenadas de textura e os ultimos três valores são as coordenadas do vetor normal àquele vértice.

]

```
0.000000; 0.000000; 1.000000
0.250000; 0.181636; 0.951057
0.000000; 0.000000; 1.000000
0.000000; 0.000000; 1.000000
0.095491; 0.293893; 0.951057
0.000000; 0.000000; 1.000000
0.000000; 0.000000; 1.000000
-0.095492; 0.293893; 0.951057
-0.000000; 0.000000; 1.000000
-0.000000; 0.000000; 1.000000
-0.250000; 0.181636; 0.951057
```

Figure 5: Ficheiro .3d

3 Engine

3.1 Alterações efetuadas relativas à fase anterior

Em termos de alterações feitas, foram feitas as seguintes:

- Foi adicionado a struct de Model, uma substituição do sistema antigo, onde se usava uma string para representar o nome do ficheiro representado para a figura. Isto foi feito para não só para adicionar o nome do ficheiro de textura dado, como também para obter os componentes para calcular a luz no material do modelo. Como tal, todas as variáveis que se referiam ao modelo com um formato string foram alteradas para este formato Model.
- Foi adicionado a struct de Light, uma forma de representação das luzes em termos de Iluminação do OpenGL. Este modelo vai permitir-nos guardar várias informações sobre os vários tipos de fontes de iluminação, fazendo a distinção com as suas variáveis para representar pontos de luz, luzes direcionais e focos de luz.
- A função set_up_from_files foi, mais uma vez, alterada como forma de receber a nova informação do ficheiro XML relacionada com as fontes de iluminação.
- A subfunção de set_up_from_files, group_set_up, foi alterada para receber a nova informação relacionada com as figuras a desenhar, ou seja, a potencial textura, tal como as variáveis de luz relacionada com os materiais.
- A struct Storage, criada para guardar toda a informação de um ficheiro XML, possui agora um vector de Light, para representar as várias fontes de iluminação que se encontram no ambiente de OpenGL.

3.2 Leitura do Ficheiro XML

A leitura do ficheiro XML foi feita através da funções que nas fases anteriores já efetuavam o parsing do ficheiro XML, set_up_from_files e group_set_up. Vamos agora explicar como cada uma foi alterada.

3.2.1 Fontes de Luz

O novo ficheiro XML para representar iluminação possui um novo campo antes de entrar na secção das figuras, designado "lights". Este campo contém várias fontes de luz que nós devemos representar.

Para tal, criamos um novo vector de Light, "luzes". Este vetor será depois alocado para dentro de xml_data pela variável do mesmo nome. Começando por um for loop que irá entrar no campo "lights", examina-se cada uma das luzes, começando por identificar se se trata de um ponto de luz("point"), uma luz direcional("directional") ou um foco de luz("spot" ou "spotlight"). Dependendo do tipo de luz obtido, o tipo(variável "type") fica definido com a string respectiva e definimos os parâmetros para cada uma das luzes respetivas:

- No caso de um ponto de luz, recebemos as coordenadas de tal ponto - posx, posy e posz.
- No caso de uma luz direcional, recebemos as coordenadas do vetor de direção - dirx, diry e dirz.
- No caso de um foco de luz, obtemos as coordenas do ponto onde começa o foco - posx, posy e posz -, as coordenadas do vetor de direção - dirx, diry e dirz -, como também o ângulo dentro do qual a luz está toda iluminada pelo foco - cutoff.

Com estes parâmetros definidos para todas as luzes encontradas no ficheiro, associamos o vetor onde guardamos as variáveis Light à variável "luzes" de xml_data.

3.2.2 Materiais e Texturas

Cada campo "model" no ficheiro XML agora possui dois novos campos opcionais, o campo "texture", que contém o ficheiro que irá servir como textura, e o campo "color", que serve para representar as várias reações que o material do modelo terá perante as fontes de luz. De notar que caso o campo "texture" não esteja presente, só se considera o campo "color", e se este não estiver presente, utiliza-se uma série de valores por defeito fornecidos pelo enunciado.

Durante a leitura do campo "model", agora abrimos duas secções opcionais, uma para encontrar a textura e outra para encontrar as várias reações da luz. Dado a troca da representação do modelo de uma variável string para a sua própria estrutura Model, temos agora de adaptar os 3 passos de criação de modelo:

- No primeiro passo, iremos criar o Model em si mesmo, usando a variável do "file" para extrair do XML o nome do ficheiro que contém os pontos que servem para o desenho do modelo, que será guardado na variável "name", sendo modificado para conter o caminho para a pasta onde está guardado o ficheiro.
- No segundo passo, verificamos se o campo "texture" encontra-se dentro do XML para o modelo. Se existir, extraímos o nome e guardamos na variável "texture" do Model, sendo modificado para conter o caminho para a pasta onde está guardada a textura.
- No terceiro passo, verificamos se o campo "color" encontra-se dentro do XML para o modelo. Se existir, extraímos as diversas variáveis para as várias reações da luz que descrevem o material do modelo, luz difusa, ambiente, especular, emissiva e o valor de "shininess" do objeto. Estes valores são depois guardados para os campos respetivos de Model. Importante denotar que, caso o campo "color" não exista, as variáveis de materiais ainda são inicializadas com os valores fornecidos no enunciado como os por defeito.

Após estes parâmetros serem definidos em Model, os modelos são armazenados num vetor de ficheiros que será depois guardado da mesma forma como nas fases anteriores, guardado no respetivo group para depois ser guardado em xml_data.

3.3 Iluminação

A iluminação foi tratada com o uso de uma função nova chamada de lighting_time e uma mudança na função fromFileToShape.

3.3.1 Função lighting_time

Esta função utiliza o vetor "luzes" de xml_data como o seu argumento e irá iterar sobre este de forma a ativar cada uma das luzes.

Para explicar em maior detalhe, esta função é chamada após o axis de x, y e z ser desenhado sem as luzes ativadas, de forma a manter a sua coloração vermelho, verde e azul. Após isto, iremos ativar o parâmetro GL_LIGHTING e chamar lighting_time.

Em lighting_time, após obtermos o vetor de "luzes", iremos iterar sobre cada um dos membros para definir as potenciais fontes de luz.

Para isto, iremos definir uma variável "i". Esta variável serve como uma forma de identificar quais das fontes de luz estamos a usar, e mais importante do que isso, qual das variáveis do OpenGL para fontes de luz estamos a usar, já que só são permitidos 8 fontes no total (GLLIGHT0-GLLIGHT1). Entrando no caso switch, entramos num padrão igual para qualquer umas das fontes de luz, mas onde o processo essencial é o mesmo.

O padrão começa por ativar a fonte de luz, usando a instrução glEnable(), onde depois entra numa série de condicionais para perceber que tipo de luz está a ser criada (ponto de luz, luz direcional, foco de luz). Após ser determinado o tipo de luz, usa-se as variáveis guardadas para criar os vários necessários, como por exemplo arrays indicadores de pontos de coordenadas ou de vetores de direção, e depois, estes são utilizados para criar a própria luz através das funções glLightfv() e glLightf(), dependendo do caso. Eis o que cada caso origina:

- Ponto de luz.
  - Cria vetor de coordenadas do ponto, usando as suas três coordenadas pos e o valor 1.0 para tal (**float pos[4] = luz.posX, luz.posY, luz.posZ, 1.0;**)
  - Utiliza o vetor de coordenadas de ponto para criação da luz (**glLightfv(GL_LIGHT6, GL_POSITION, pos);**)
- Luz direcional.
  - Cria vetor de direção do vetor, usando as suas três coordenadas dir e o valor 1.0 para tal (**float dir[4] = luz.dirX, luz.dirY, luz.dirZ, 0.0;**)
  - Utiliza o vetor de direção do vetor para a criação da luz (**glLightfv(GL_LIGHTi, GL_POSITION, dir);**)
- Foco de Luz.

- Cria vetor de coordenadas do ponto, usando as suas três coordenadas pos e o valor 1.0 para tal (**float spot_pos[4] = luz.posX, luz.posY, luz.posZ, 1.0;**)
- Cria vetor de direção do vetor, usando as suas três coordenadas dir para tal (**float spot_dir[3] = luz.dirX, luz.dirY, luz.dirZ;**)
- Utiliza o vetor de coordenadas de ponto para criação da luz (**glLightfv(GL_LIGHTi, GL_POSITION, spot_pos);**)
- Utiliza o vetor de direção do vetor para a criação da luz (**glLightfv(GL_LIGHTi, GL_SPOT_DIRECTION, spot_dir);**)
- Utiliza o valor de cutoff para a criação da luz (**glLightf(GL_LIGHTi, GL_SPOT_CUTOFF, luz.cutoff);**)

3.3.2 Função fromFileToShape

Nesta função que já existia em fases anteriores, é lhe dado como argumento um dos modelos para ser efetuado o desenho. Um novo campo foi adicionado a esta função, onde são executadas instruções que extraem os parâmetros de iluminação e criam-se arrays com estes parâmetros, e depois usam-se esses mesmos arrays nas funções glMaterialfv e glMaterialf para computar os parâmetros. Eis o formato das instruções para cada tipo de luz:

- Luz difusa.
  - Criação do vetor de parâmetros de luz difusa (**float diffuse_parameters[4] = file.diffuse_R, file.diffuse_G, file.diffuse_B, 1.0;**)
  - Computação dos parâmetros (**glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse_parameters);**)
- Luz ambiental.
  - Criação do vetor de parâmetros de luz ambiental (**float ambient_parameters[4] = file.ambient_R, file.ambient_G, file.ambient_B, 1.0;**)
  - Computação dos parâmetros (**glMaterialfv(GL_FRONT, GL_AMBIENT, ambient_parameters);**)
- Luz especular.
  - Criação do vetor de parâmetros de luz especular (**float specular_parameters[4] = file.specular_R, file.specular_G, file.specular_B, 1.0;**)
  - Computação dos parâmetros (**glMaterialfv(GL_FRONT, GL_SPECULAR, specular_parameters);**)
- Luz emissiva.
  - Criação do vetor de parâmetros de luz emissiva (**float emissive_parameters[4] = file.emissive_R, file.emissive_G, file.emissive_B, 1.0;**)
  - Computação dos parâmetros (**glMaterialfv(GL_FRONT, GL_EMISSION, emissive_parameters);**)
- Brilho do objeto.
  - Criação da variável de parâmetros de brilho do objeto (**float shininess_parameter = file.shininess_value;**)
  - Computação dos parâmetros (**glMaterialf(GL_FRONT, GL_SHININESS, shininess_parameter);**)

3.4 Texturas

Para esta última fase foi necessário fazer a aplicação de textura aos diversos objetos. Para tal começamos por carregar as texturas para memoria gráfica usando para o efeito a função fromFileToShape que recebe como argumento um Model que contém informações sobre o ficheiro a ser carregado.

Primeiro é aberto o ficheiro de leitura e verifica-se se este foi aberto corretamente. De seguida são realizadas as etapas típicas de carregamento da textura, como a geração do identificador de imagem, a associação da imagem atual etc. Em seguida, o identificador de textura é gerado, associado ao alvo da textura GL_TEXTURE_2D e definidos os parâmetros de textura, incluindo repetição, filtro de ampliação e redução. Por fim, os dados da textura são enviados para o OpenGL usando glTexImage2D. Por fim,é necessário ativar o uso de texturas através das funções glEnable(GL_TEXTURE_2D) e glDisable(GL_TEXTURE_2D).

4 Demos

De modo a demonstrar de forma visual o trabalho realizado, nesta secção iremos dar exemplos alguns do resultados obtidos.

4.1 Demos de algumas primitivas

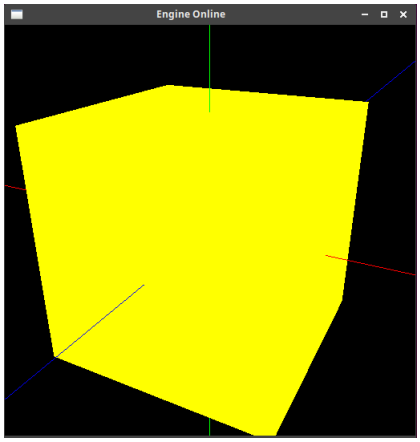

Figure 6: Demo da caixa

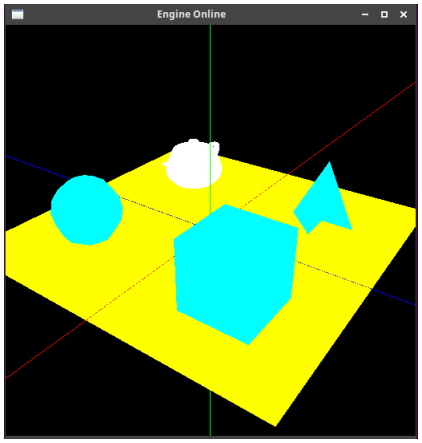

Figure 7: Demo 1

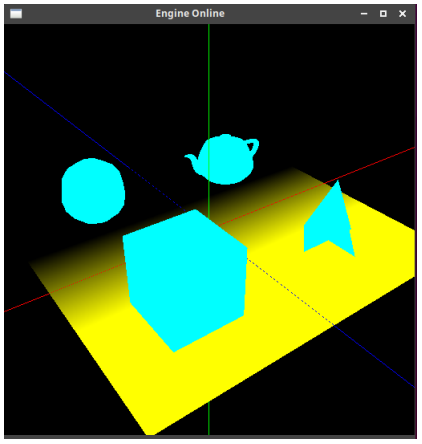

Figure 8: Demo 2

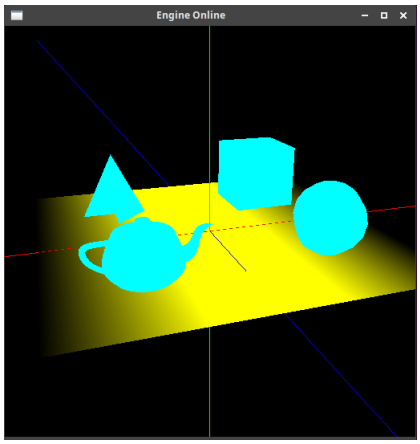

Figure 9: Demo 3

4.2 Sistema Solar

Apresentamos agora o sistema solar final, com tudo o que fomos aprendendo e aplicando em todas as fases realizadas. Nesta última consguimos apenas adicionar a iluminação para cada um dos planetas.

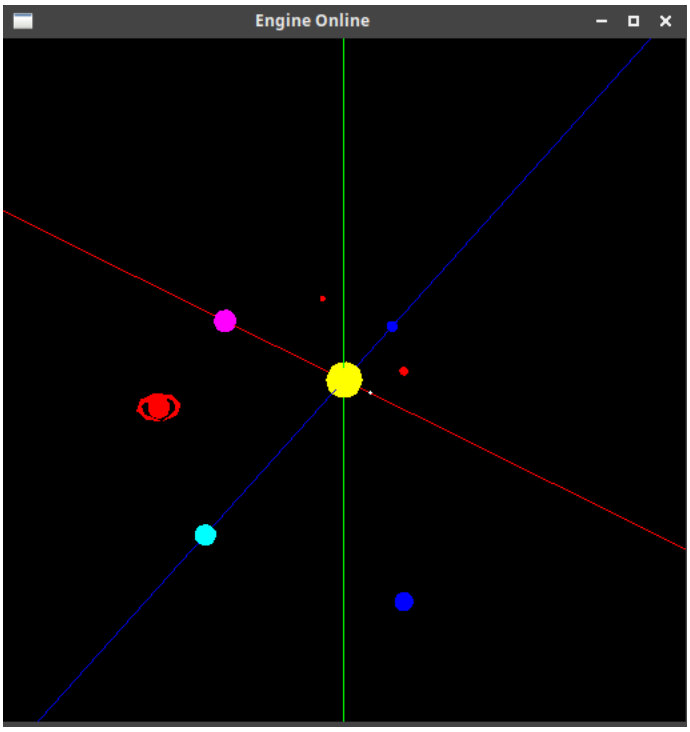

Figure 10: Sistema Solar

5 Conclusão

Dada por concluída a última fase do projeto, consideramos importante realizar uma análise crítica, e ainda, realçar os pontos positivos e negativos.

O facto de terem sido feitas alterações à fase anterior e conseguirmos implementar a quase totalidade da fase 4 dentro do tempo estipulado consideramos ser um ponto positivo.

Um ponto negativo foi não termos implementado a totalidade das funcionalidades e podiamos também ter posto um sistema solar mais realista.

Para concluir, consideramos que houve um balanço positivo do trabalho realizado dado que as dificuldades sentidas foram quase todas superadas e foram cumpridos quase todos os requisitos.