Universidade do Minho

Trabalho Prático da Unidade Curricular de Computação Gráfica

Licenciatura em Ciências da Computação

Ano Letivo de 2022/2023

André Costa (A95869), Filipe Castro (A96156), Tiago Teixeira (A97666)

Fase 4

Normais e coordenadas de textura

Enunciado

Nesta fase a aplicação passa a gerar coordenadas de textura e normais para cada vértice. É também possível definir fontes de luz de três tipos: point, directional e spotlight. Através destes elementos aplicamos as texturas aos modelos desejados e calculamos a cor de cada vértice de acordo com a textura e o efeito das fontes de luz neste.

Decisões e abordagens

Mais uma vez, nesta fase, tivemos que alterar as duas aplicações que até agora desenvolvemos, para atingir os objetivos para esta última etapa.

Tais alterações foram as seguintes:

1.Generator: Gerar todos os pontos das coordenadas da normal e da textura;

2.Engine: Desenhar modelos com as respectivas cores e texturas;
Ser capaz de aplicar diferentes tipos de luzes;

Nota: Assim como dito em fases anteriores, mantemos o uso do rapidxml.

1. Generator

Aqui adiciona-se, às capacidades do generator da **fase 3**, as tais coordenadas novas:

- Vetor **normal** - constituído por três coordenadas, este vetor é sempre perpendicular à superfície a que se aplica;

- Pontos de **textura** - constituído por duas coordenadas, que referenciam a sua localização na imagem usada para textura.

Ou seja, além de serem colocadas nos ficheiros ".3d" as coordenadas dos modelos, temos escrito nas mesmas linhas, as novas coordenadas descritas em cima.

2. Engine

Cores e Texturas:

Neste ponto, temos então essas novas funcionalidades que podem ou não ser aplicadas simultaneamente. Dizemos isso pois dá-se ao utilizador essa liberdade de escolha.

Portanto, quando se escreve “model=file.3d”, de seguida, o utilizador decide se quer aplicar uma textura, cor ou as duas, ao modelo a que se refere.

Aplicando:

Somente **“texture=ficheiro.jpg”**, tendo em conta que o ficheiro se encontra numa pasta de imagens, procura pelo nome do ficheiro e seguindo as coordenadas definidas no ”.3d”, aplica a tal textura.
Nota: Depois disso é necessária a presença de uma cor pré-definida pelo código.

Somente o subgrupo **“color”**, aplica as cores que se encontram nesse grupo, sem contemplar a existência de uma textura (diffuse, specular, ambient, shininess e emissive).

As duas em simultâneo, aplicam tanto a textura representada pelo ficheiro imagem, como as cores presentes no subgrupo.

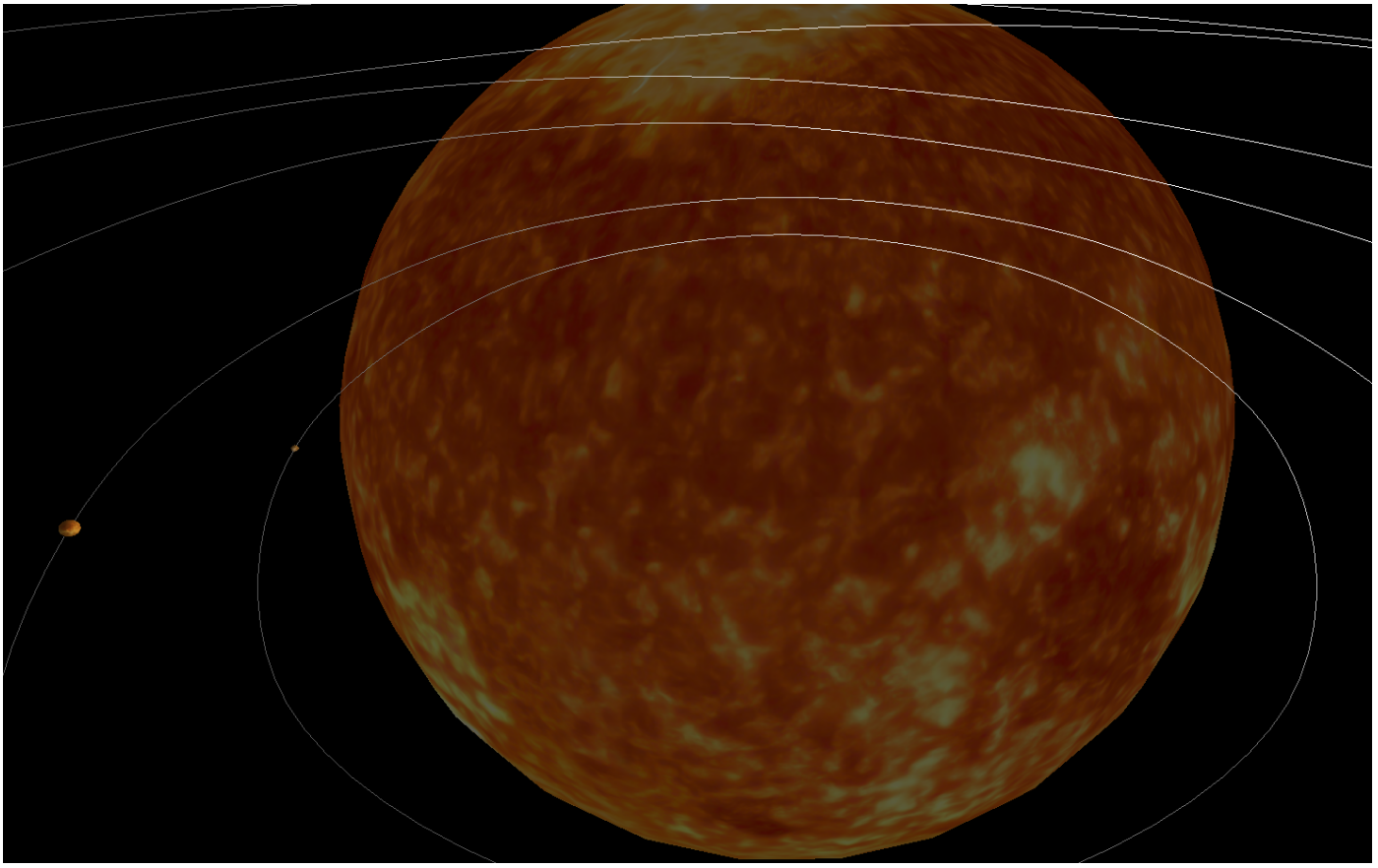
Luzes:

Nesta fase adicionamos também a possibilidade de se definir fontes de luz a partir de um ponto (point), um vetor (directional) ou um spotlight (spotlight).

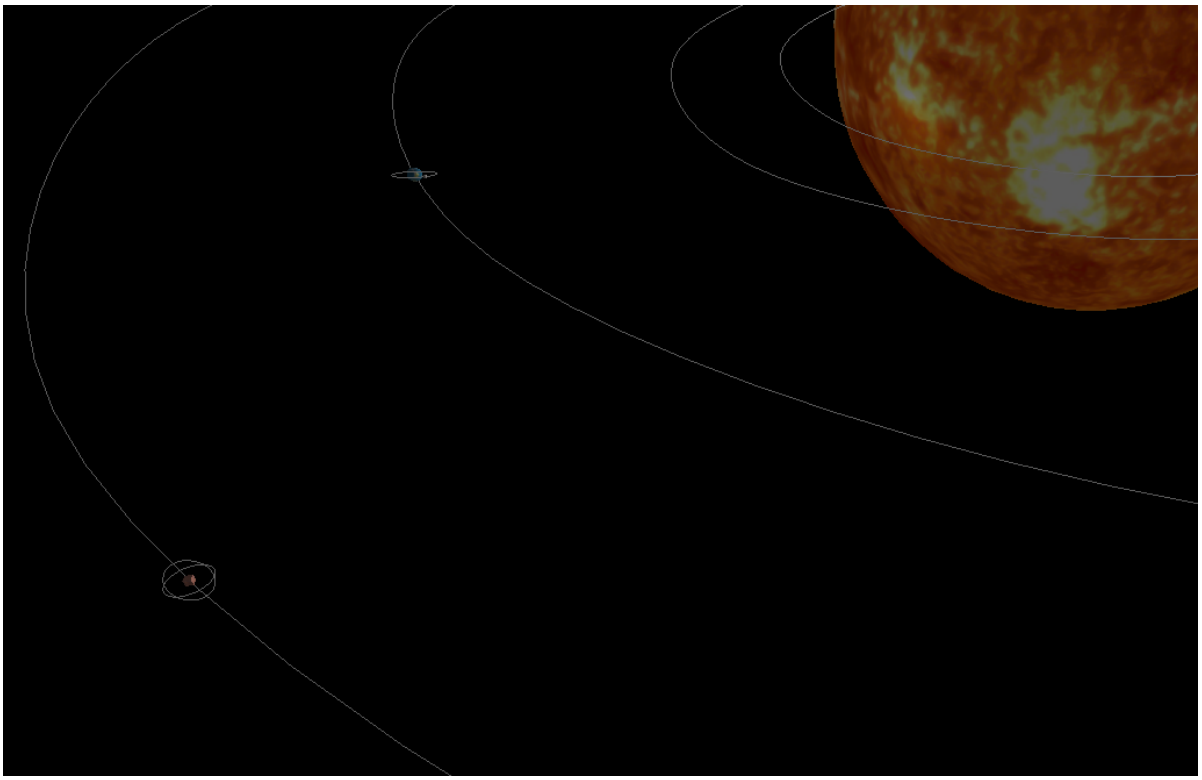
Para se aplicar as luzes, é necessário adicionar no ficheiro XML, após a câmara, a partição “lights” onde se pode adicionar até 8 luzes de qualquer tipo.

Sistema Solar com texturas e luzes

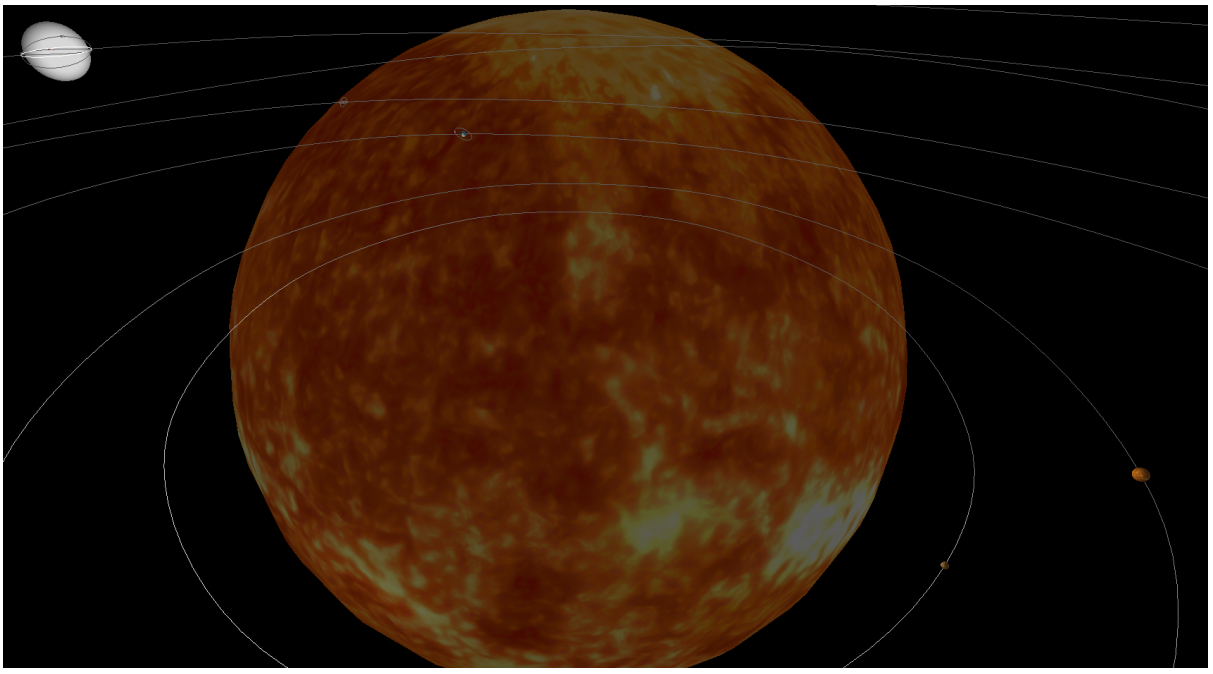
Para testar as capacidades da nossa engine, alteramos o ficheiro XML usado na fase anterior, de modo a que cada corpo tenha uma textura e o Sol tenha quatro pontos de luz ao seu redor.



Sol, Mercúrio e Vénus



Terra e Marte



Sistema Solar

Conclusão

Nesta fase apesar de acreditarmos ter implementado de forma correta tanto as luzes como as texturas e cores, deparamo-nos com um erro estrutural do nosso código que tornou a visualização da nossa demo muito difícil.

O nosso programa engine tem vários problemas de performance que se notam imenso em simulações a partir de cerca de 10 modelos diferentes a serem desenhados. Acreditamos que tal acontece pela forma que o engine está a desenhar cada modelo pois, no nosso código, primeiro é aglomerada toda a informação do ficheiro XML numa stringstream chamada actions que está a separar cada ação (com ação referimo-nos ao desenho de um modelo, uma translação do eixo, o início de um grupo, etc.) e depois de action conter toda a informação do ficheiro XML, vamos percorrer action pegando em cada ação e fazendo o que nos indica. Partindo então deste método de funcionamento de engine, cada modelo apenas tem oportunidade de ser desenhado quando o encontramos em actions, pois caso o desenhassemos noutra altura iriamos perder as transformações feitas ao eixo para o desenhar da forma que queremos. Ao termos a estratégia de desenho feita desta maneira, quando chegamos à fase 3 do trabalho e nos foi pedida a implementação de VBO's, a solução que encontramos foi a criação de um buffer de tamanho igual ao número de modelos que se teria de desenhar, e cada vez que se encontrava uma ação model, passava-se os pontos desse para um array que colocava os pontos no buffer e era imediatamente desenhado o modelo. No momento achamos que não havia problema em ter a implementação desta maneira e ainda não iríamos encontrar problemas sérios de performance até começarmos a fase 4.

Agora na fase 4 por causa das decisões que tivemos nas fases anteriores cada vez que é encontrado um model é necessário criar os 3 arrays para os triângulos do modelo, os vetores normais e os pontos das texturas, colocar os 3 arrays nos buffers para cada um deles e desenhar os 3 buffers, o que vai aumentar imenso o trabalho feito pelo engine sempre que se encontra um model e dando vários problemas de performance.

Se começássemos o trabalho sabendo o que sabemos agora, provavelmente optaríamos pela implementação de structs que, à medida que fossem extraídas as informações do XML, guardariam tudo associado a cada modelo individualmente, assim, achamos que no final do renderScene fosse possível desenhar todos os modelos de uma vez poupando bastante tempo de processamento.

Para além disso reparamos nesta fase também que apesar do ângulo de onde estamos a ver os modelos desenhados, estes são desenhados sempre na mesma ordem e, portanto, em várias situações, objetos que deveriam ficar atrás ficam à frente e vice-versa.