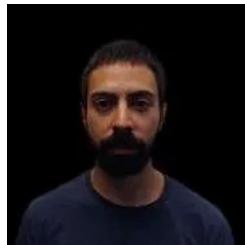


Computação Gráfica

Relatório Trabalho Prático - Fase 4

Grupo 33 LEI - 3º Ano - 2º Semestre

Ano Letivo 2024/2025



Adélio Fernandes
A78778



Tiago Carneiro
A93207



Tiago Guedes
A97369

Braga,
18 de maio de 2025

Conteúdo

1	Introdução	3
2	Fase 4	3
2.1	Cores	3
2.2	Normais	3
2.3	Luzes	5
2.3.1	Pontos, Direções e Focos	5
2.3.2	<i>lights.xml</i>	5
2.4	Texturas	5
2.5	Sistema Solar	6
2.5.1	Skybox	6
2.5.2	Representação do Sistema	7
3	Extras	8
3.1	Melhoria do <i>Tracking</i>	8
3.2	Representação Visual das Normais	8
3.3	<i>Time Stop</i>	9
3.4	<i>Shaders</i>	9
3.4.1	<i>CRT Filter</i>	9
3.4.2	<i>Outline</i>	10
3.5	<i>Vault of the Golden Biscuit</i>	11
4	Conclusão	11

Lista de Figuras

1	Representação das cores	3
2	Normais das Superfícies de Bezier	4
3	Normais da esfera e o <i>torus</i>	4
4	Classes light e light_spot	5
5	Classe lights_xml	5
6	Mapeamento da textura da Terra numa esfera com 8 stacks e 10 slices	6
7	Modelo da Terra com a textura aplicada	6
8	Skybox do Sistema Solar	7
9	Ficheiros XML de Saturno e Jupiter	7
10	Sistema solar com texturas e luzes	8
11	Representação visual das normais	8
12	CRT Shader	10
13	Outline Shader	10
14	<i>Vault of the Golden Biscuit</i>	11

1 Introdução

Neste relatório iremos apresentar detalhadamente os requisitos para a última fase do projeto, a Fase 4, incluindo a implementação de texturas, funcionalidade da luz e cálculo das normais.

Em seguida, serão descritos com detalhe os extras implementados nesta fase, explicando o motivo da sua implementação e as vantagens que acrescem ao projeto como um todo.

Por fim, faremos uma conclusão geral dos objectivos atingidos, os maiores desafios encontrados e uma respetiva geral do trabalho realizado.

2 Fase 4

2.1 Cores

A representação das cores foi trivial, foi criada uma nova classe *color*, onde são armazenadas as informações relativas às cores. Esta classe é criada no momento da leitura do ficheiros *XML*. É também aqui que são definidas as cores *default* para situações em que as cores dos modelos não são explicitamente definidas.

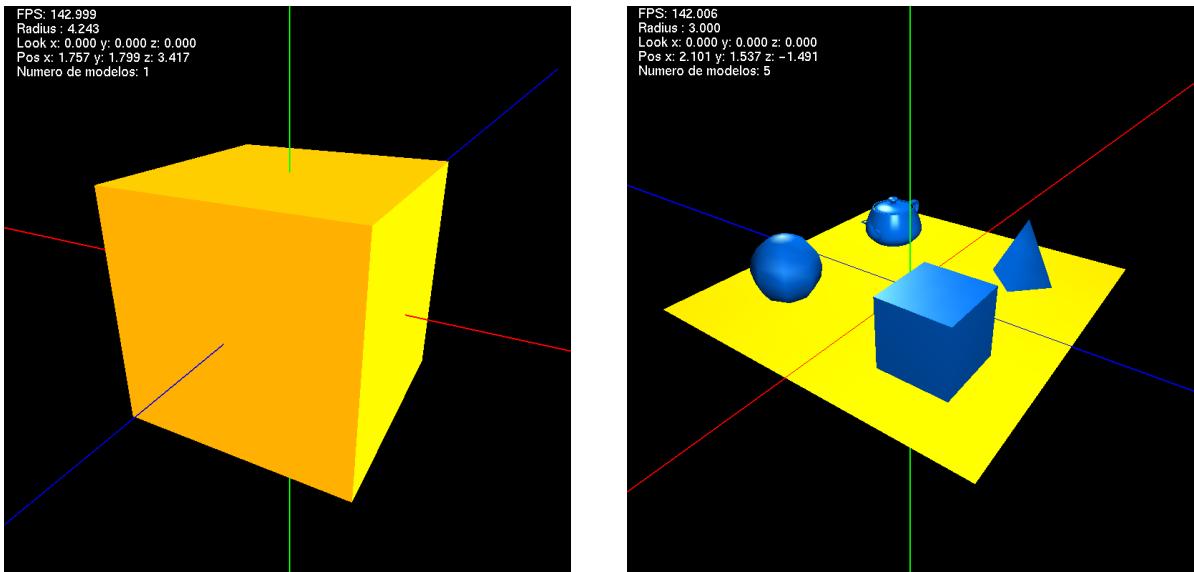


Figura 1: Representação das cores

2.2 Normais

Para a implementação das luzes, foi necessário fazer o cálculo das normais para cada uma das nossas primitivas. Tendo isso em conta, iremos agora falar um pouco de como esse cálculo foi realizado.

- **Plano** - em todos os seus vértices a normal é o vector (0,1,0)
- **Cubo** - para cada face, os vértices contêm normais perpendiculares as faces a que pertencem
- **Cone** - na base, as normais são (0,-1,0). Para as faces laterais, primeiro calcula-se o ângulo de inclinação $\theta = \arctan(\text{height}/\text{base})$, depois $\text{normal}_{xz} = \cos(\theta)$ e $\text{normal}_y = \sin(\theta)$
- **Cilindro** - para as bases de cima e de baixo, os vectores são (0,1,0) e (0,-1,0) respetivamente. Para cada vértice na superfície lateral, calcula-se a normal radial, sendo esta o vector $(x/\text{raio}, 0, z/\text{raio})$.

- **Esfera** - a normal é simplesmente o vector posição, ou seja, $(x/\text{raio}, y/\text{raio}, z/\text{raio})$, apontando radialmente para fora. Nos polos, as normais são $(0, -1, 0)$ no pólo sul e $(0, 1, 0)$ no pólo norte.
- **Torus** - para cada ponto, imaginamos o tubo como um pequeno círculo cujo centro descreve um grande anel. A normal é o vector que sai do centro desse círculo até os pontos da superfície. Em coordenadas, $\phi = \text{ângulo no tubo}$, $\theta = \text{ângulo no anel}$, a normal fica $(\cos \phi \cdot \cos \theta, \sin \phi, \cos \phi \cdot \sin \theta)$.
- **Superfícies de Bezier** - as normais foram criadas a partir dos vectores tangentes u e v para cada ponto. Estes vectores são calculados usando as derivadas dos polinómios de Bernstein para cada um deles. Tendo os vectores tangentes, fazemos o produto vectorial dos mesmos para obter o vector normal. Podemos observar uma representação visual destes vectores na seguinte imagem :

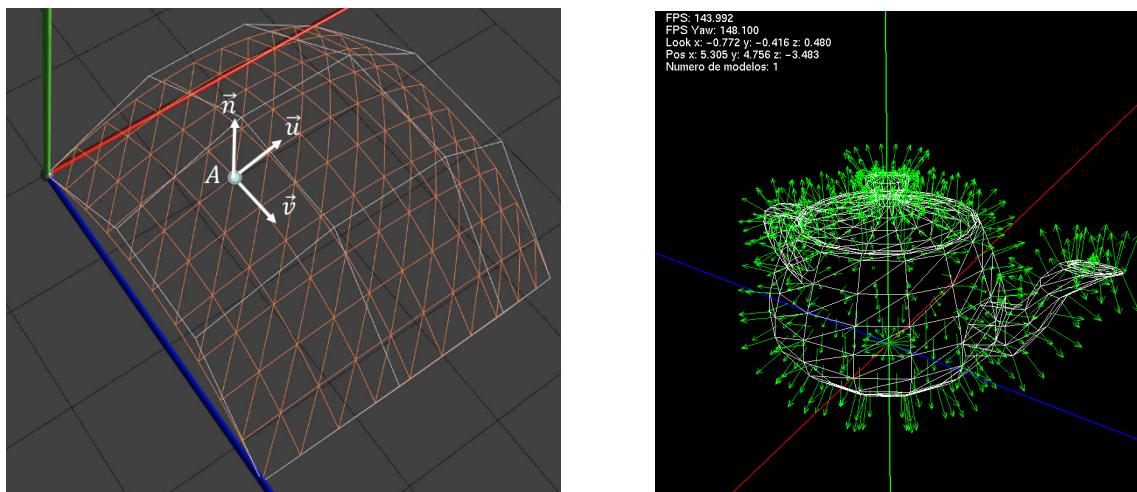


Figura 2: Normais das Superfícies de Bezier

De seguida podemos observar as normais de algumas das primitivas mencionadas anteriormente :

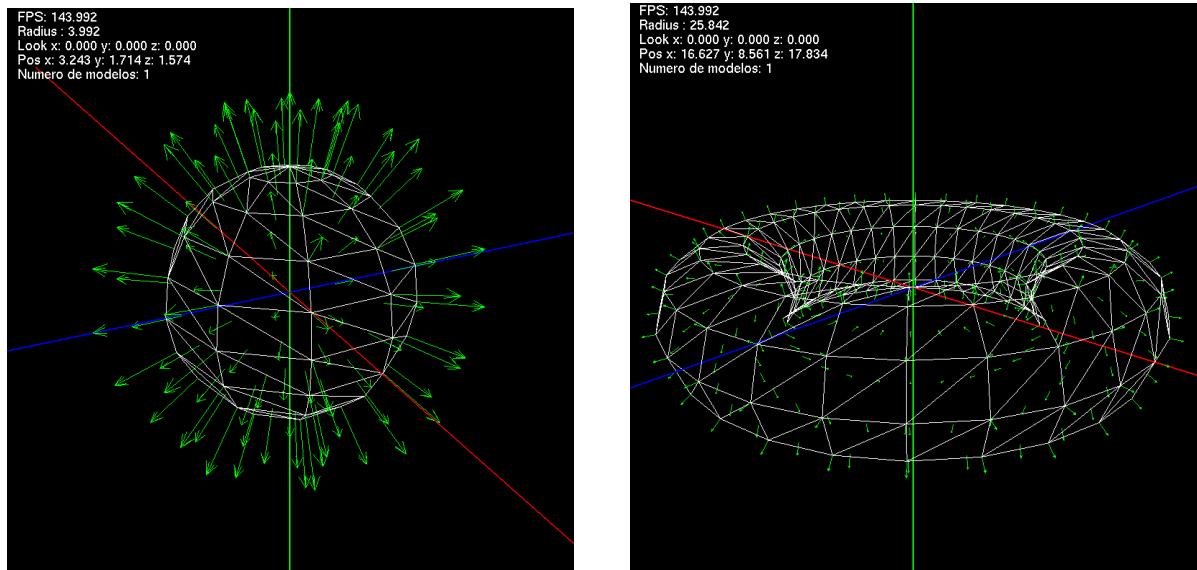


Figura 3: Normais da esfera e o torus

2.3 Luzes

Para as luzes, começamos por analisar as possíveis luzes que poderíamos encontrar. Muito rapidamente percebemos que uma luz num ponto e uma luz direcional contêm argumentos muito similares. Também reparamos que efetivamente, um foco de luz é basicamente uma luz direcional e um ponto de luz. Assim, falaremos agora da implementação das classes pertinentes a estas luzes.

2.3.1 Pontos, Direções e Focos

De seguida podemos observar as estruturas utilizadas para armazenar a informação relativa às luzes. De notar, tal como referido anteriormente, que para as luzes direcionais e pontos de luz, apenas uma classe é utilizada, a classe *light*, variando o tipo - 0 para os pontos e 1 para as direções.

```
1 class light{
2     public :
3         bool type;
4         float x;
5         float y;
6         float z;
7     };
1 class light_spot{
2     public :
3         light point;
4         light direction;
5         float cutoff;
6     };

```

Figura 4: Classes *light* e *light_spot*

Podemos observar na figura anterior que as variáveis pos e dir dos pontos de luz e direções de luz respetivamente são ambas traduzidas para variáveis que representam a sua respetiva coordenada.

2.3.2 *lights_xml*

Estas duas classes de luzes são armazenadas na seguinte classe, classe esta que é criada no processo de leitura dos ficheiros *XML*, chamada *lights_xml*. As mesmas são colocadas em vectores, de forma a que possam ser armazenadas múltiplas classes de cada tipo para o mesmo ficheiro *XML*. Esta classe está contida na estrutura *xml_parser*.

```
1 class lights_xml {
2     public :
3         std::vector<light> lights;
4         std::vector<light_spot> light_spots;
5     };

```

Figura 5: Classe *lights_xml*

2.4 Texturas

Falando agora um pouco sobre a representação das texturas, estas são carregadas para a *GPU* e é armazenado o seu identificador num dicionário, utilizando o nome do ficheiro como chave. Assim, somos capazes de evitar carregar a mesma textura mais de uma vez. Após isso, a aplicação da textura é feita recorrendo às coordenadas da mesma definidas para cada modelo.

De seguida podemos observar como a textura da Terra é mapeada, assim como é apresentada na esfera, tanto no seu ficheiro *XML* individual e, portanto, sem luz, mas também no Sistema Solar, apresentando reflexão da luz:

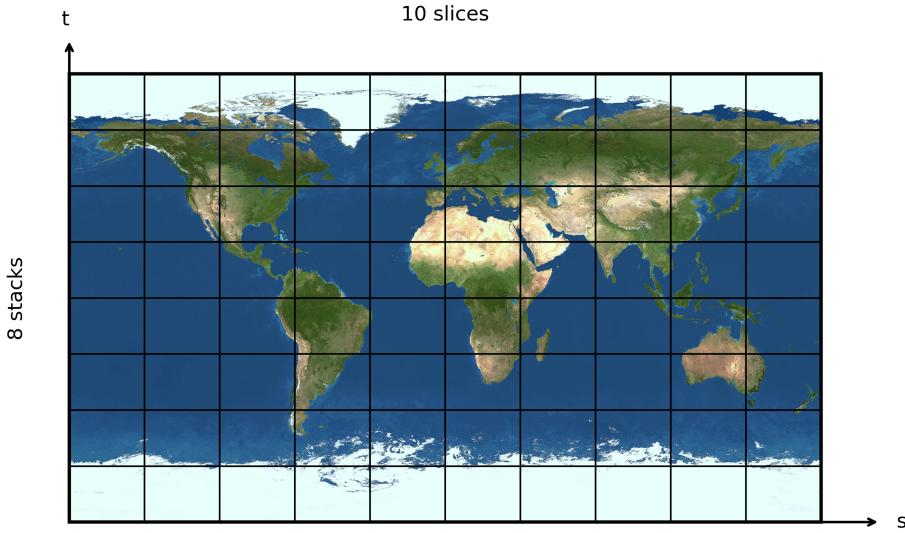


Figura 6: Mapeamento da textura da Terra numa esfera com 8 stacks e 10 slices

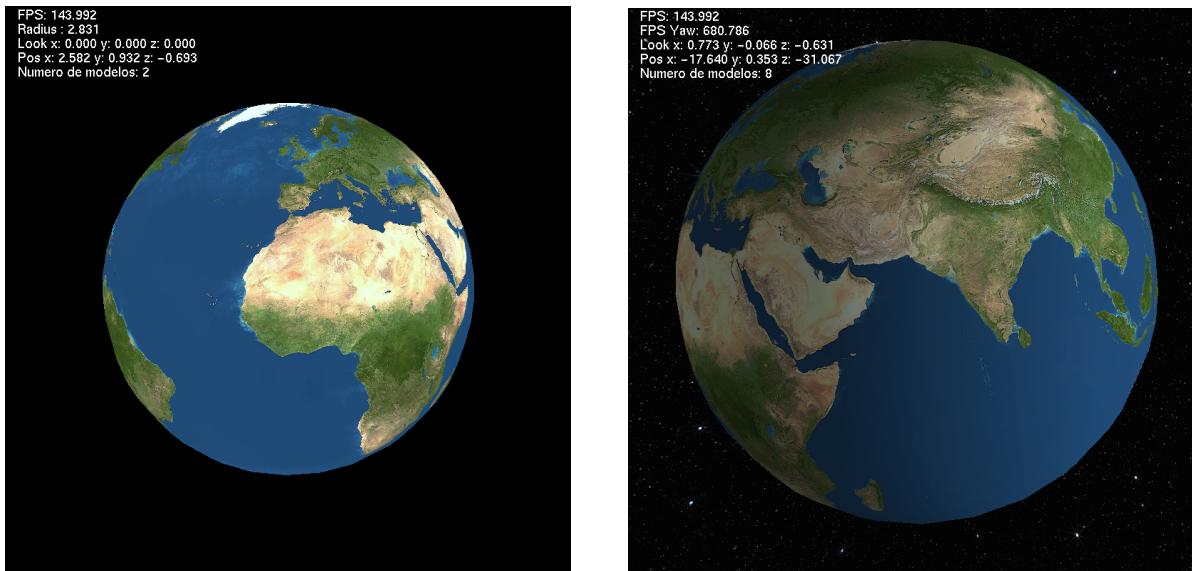


Figura 7: Modelo da Terra com a textura aplicada

2.5 Sistema Solar

Iremos agora entrar em detalhe nas atualizações feitas ao ficheiro do Sistema Solar, tanto relativamente às texturas, como também relativamente às luzes. Para além disso, iremos falar de um novo ficheiro *XML* introduzido ao Sistema Solar.

2.5.1 Skybox

De forma a termos uma representação mais real de um Sistema Solar, decidimos criar um ficheiro XML com seis planos criando, assim, uma caixa virada para dentro que envolve o Sistema Solar e utilizando uma textura do espaço estelar.

Isto revelou-se extremamente útil, pois permitiu implementar corretamente um dos *fragment shaders* mencionados na secção de extras, o *outline* dos vários modelos representados no Sistema Solar.

De seguida, podemos observar esta mesma skybox:

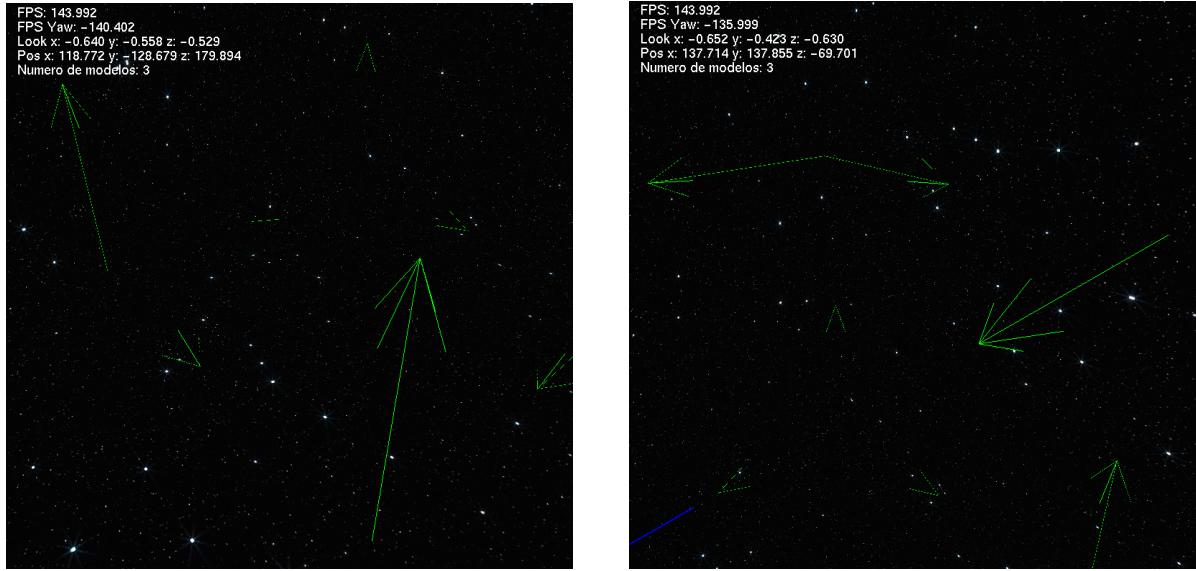


Figura 8: Skybox do Sistema Solar

Através das setas representativas das normais (algo que abordaremos mais à frente, na secção de extras), é possível verificar que estão representados vários planos virados para o interior.

2.5.2 Representação do Sistema

Para além da Skybox mencionada anteriormente, atualizamos os nossos astros de forma a conterem texturas reais para cada um deles.

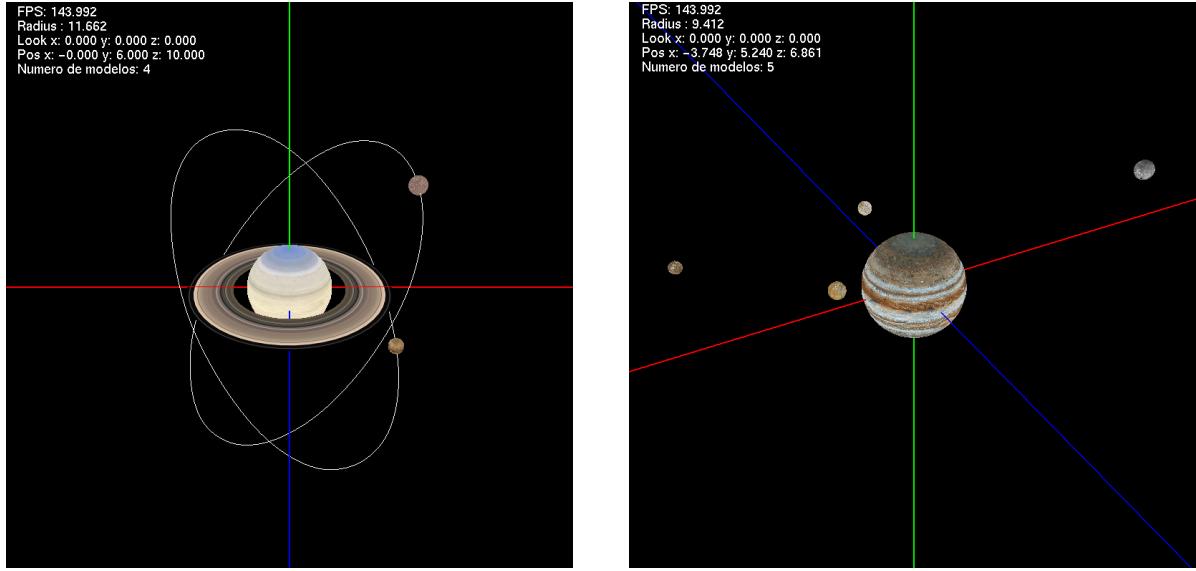


Figura 9: Ficheiros XML de Saturno e Jupiter

Para as texturas dos diversos astros (assim como as suas luas) tivemos o cuidado de procurar texturas fidedignas e de alta resolução, de forma a obter o resultado mais realista possível.

Para além das texturas foi adicionado um ponto de luz na posição do Sol, assim como dada uma emissão de luz ao Sol em si.

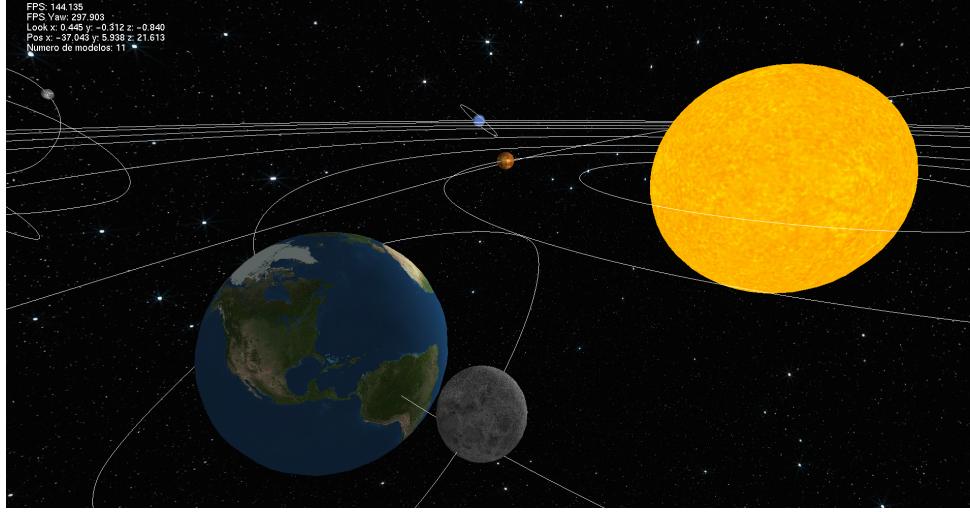


Figura 10: Sistema solar com texturas e luzes

3 Extras

3.1 Melhoria do *Tracking*

Tal como referido na fase anterior, procuramos melhorar o tracking dos modelos. Para tal, optámos por aumentar o número de pontos nas curvas de *Catmull-Rom* utilizadas para objetos com tracking, caso a câmara esteja de momento a seguir os mesmos, tornando a transição entre pontos mais suave.

Adicionalmente, reduzimos ligeiramente a quantidade de pontos nas curvas de *Catmull-Rom* para os objetos que não estão a ser seguidos (não *tracked*), de forma a melhorar o desempenho.

Nestes casos, a diferença visual é mínima, o que justifica a optimização.

3.2 Representação Visual das Normais

Após a criação das normais para as diferentes primitivas que desenvolvemos, surgiu a ideia de fazer uma representação visual dessas normais, de modo a facilitar o seu processo de debug.

Na figura seguinte, é possível observar como estas normais são representadas visualmente, utilizando setas verdes:

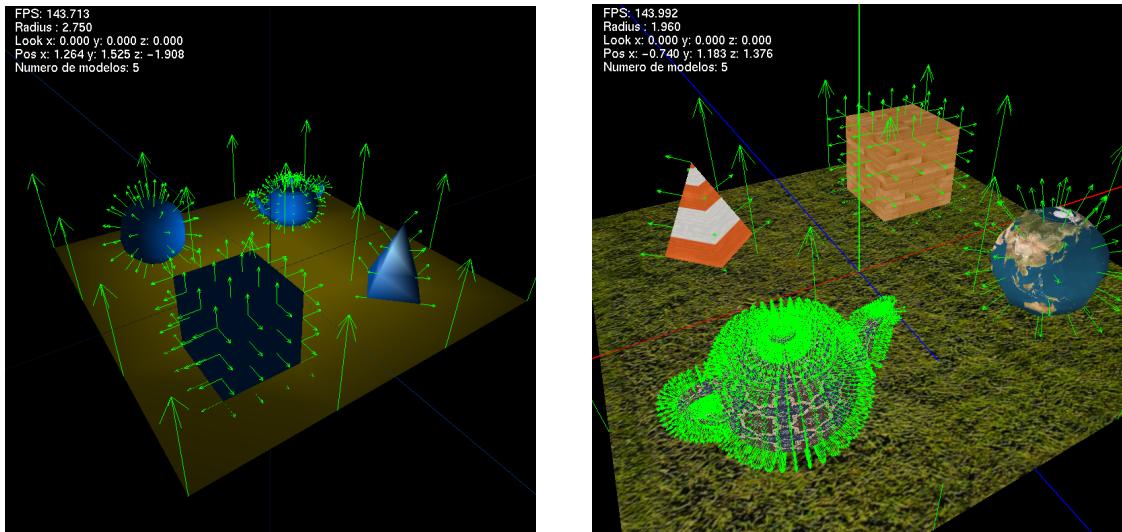


Figura 11: Representação visual das normais

3.3 Time Stop

Nesta fase mudamos a maneira como as rotações e translações temporais funcionavam. Previamente, estas usavam o *Elapsed_Time* do *Glut*. No entanto, achamos relevante adicionar a capacidade para parar o tempo.

Para tal, calculamos o novo *elapsed_time* da seguinte forma:

```
elapsed_time = glutGet(GLUT_ELAPSED_TIME) - delta_time
```

Este *delta_time* começa inicialmente por ser zero e, sempre que o tempo é parado, este é igualado da seguinte forma:

```
delta_time = glutGet(GLUT_ELAPSED_TIME) - elapsed_time
```

Assim, quando o tempo é parado, o *elapsed_time* não é acrescentado e, quando o tempo volta a andar, garantimos uma continuação natural das translações e rotações temporais.

3.4 Shaders

Para esta fase final, decidimos implementar alguns *fragment shaders* simples, pois achamos relevante explorar um pouco a utilização dos mesmos. Para tal, criamos dois *shaders*, um demonstrando um filtro parecido ao efeito que podemos encontrar nas televisões antigas que utilizavam a tecnologia *CRT*, e um demonstrando um *outline* a volta dos nossos vários modelos.

Começamos por criar uma classe *shader*, capaz de ler os vários ficheiros dos *fragment shaders* assim como os *vertex shaders* necessários para estas funcionalidades.

Após isso é criado uma *framework* para o pós-processamento que redireciona a renderização para texturas para um buffer separado, possibilitando a aplicação de efeitos visuais antes de apresentar o resultado na tela.

Ela inicializa e mantém esse buffer, ajustando as suas dimensões conforme o tamanho da janela muda, e organiza o fluxo para desenhar toda a cena off-screen. Em seguida, um *shader* é usado para processar essa imagem ao ser desenhada num *quad* que cobre toda a área visível. Por fim, o código restaura o estado da pipeline gráfica, assegurando que os componentes seguintes funcionem normalmente.

3.4.1 CRT Filter

Este *fragment shader* simula o efeito que um *Cathode-Ray Tube (CRT)* cria numa televisão. Isto é possível aplicando os efeitos visuais seguintes à imagem completa:

1. Distorcer as coordenadas para criar o efeito *Fish-eye*.
2. Intensificar linhas mais escuras e claras de forma a simular as *scanlines* presentes nas televisões *CRT*.
3. Desfocar os canais *RGB* para simular o *bleed* das cores.
4. Introduzir um pouco de granulação para simular o ruído eletrónico.
5. Criar um *flicker* sútil.

Podemos agora observar o resultado final destas várias transformações que os pixéis sofrem nas seguintes duas imagens:

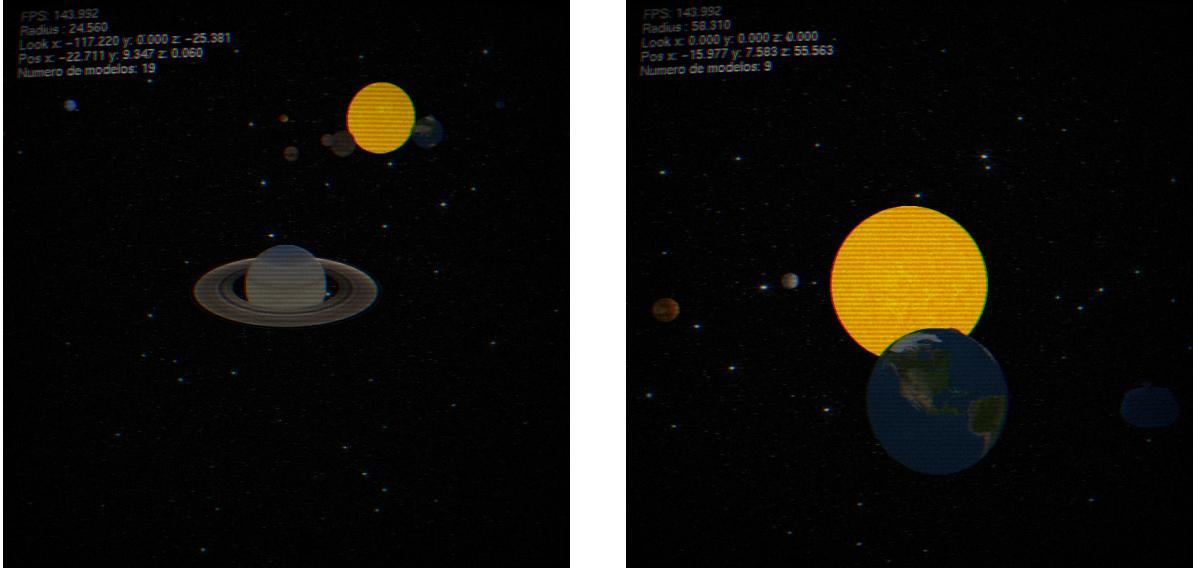


Figura 12: CRT Shader

3.4.2 Outline

Este *fragment shader* implementa uma deteção de *outlines* através das disparidades de profundidade entre os vários pixels. Para cada pixel, lê-se o valor de profundidade atual e os dos quatro vizinhos (acima, abaixo, esquerda e direita), calculando-se a maior diferença entre eles.

Se essa diferença ultrapassar um certo valor, considera-se que há um contorno naquele ponto. Em seguida, obtém-se a cor original do pixel e mistura-se com uma cor de destaque (neste caso, branco), aplicando-a apenas onde o teste de outline for positivo.

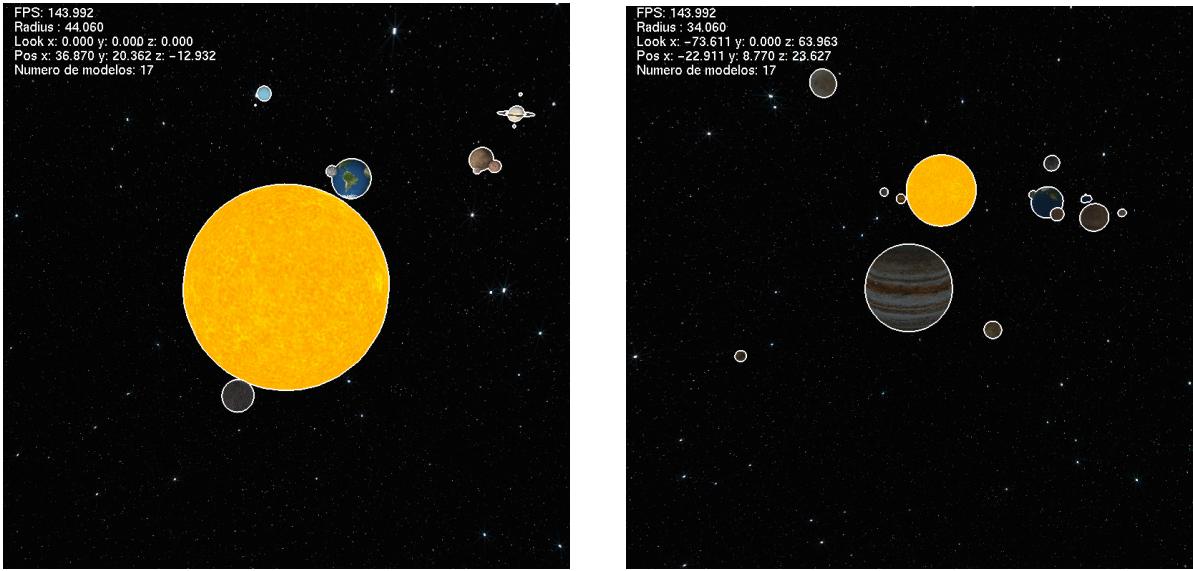


Figura 13: Outline Shader

Assim, o *shader* realça automaticamente arestas dos objetos na cena sem recorrer a geometria adicional ou processamento fora da GPU.

3.5 Vault of the Golden Biscuit

Como pequeno extra, decidimos imortalizar o nosso esforço neste projecto criando uma pequena demo, onde guardamos fotos dos nossos animais.

Assim foi criado o *Vault of the Golden Biscuit* !



Figura 14: *Vault of the Golden Biscuit*

4 Conclusão

Após a conclusão das 4 fases, ficamos extremamente contentes com o trabalho realizado. Todas as fases foram realizadas com sucesso, foram adicionados os 5 extras mencionados pelos professores (*View Frustum Culling*, novas primitivas, câmara orbital, demos complexas extra e extensão da formatação *XML*), assim como 5 extras adicionais não mencionados pelos professores (*menu*, *HUD* com texto, *Time Stop*, representação visual das normais e *shaders*).

Sentimos que muitos dos extras que colocamos foram fundamentais para o *debug* de algumas das funcionalidades que foram pedidas (como por exemplo a capacidade de ver visualmente as normais ou o texto representativo de quantos objetos estão a ser desenhados, algo fundamental para perceber se o *View Frustum* se encontra funcional).

A reestruturação da fase 1 e 2 na fase anterior demonstrou-se uma valia na realização desta fase, pois a implementação da mesma tornou-se muito clara e fácil graças a nova estrutura e clareza do código reestruturado anteriormente.

Foi referido no relatório anterior que estávamos a pensar implementar tanto *Spacial Partitioning* como *Occlusion Culling*. Para o *Occlusion Culling*, investigamos um pouco, mas não o suficiente para implementar essa otimização. Já o *Spacial Partitioning*, investigamos o suficiente para sermos capazes de implementar essa otimização, mas infelizmente não foi possível fazer a implementação da mesma devido a limitação de tempo.

Devido a esta falta de tempo encontramos também dificuldade na implementação das luzes, tendo conseguido implementar na totalidade as funcionalidades pedidas, mas sem termos tido tempo de implementar um novo extra em que tínhamos pensado, uma representação visual dos focos de luz.