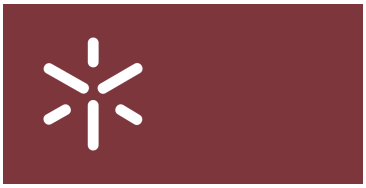

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática
Licenciatura em Engenharia Informática

Unidade Curricular - Computação Gráfica

Ano Letivo de 2024/2025

Fase 4 - Normais e Coordenandas de Texturas
Grupo 36

Tomás Henrique Alves Melo a104529
Carlos Eduardo Martins de Sá Fernandes a100890
Hugo Rafael Lima Pereira a93752
Alexandre Marques Miranda a104445

CG

Abril, 2025

Resumo

Esta fase do projeto centrou-se na implementação do suporte a normais e coordenadas de textura no ambiente 3D, fundamentais para a aplicação de iluminação e texturização dos objetos. A geração e atribuição de normais permitiu simular corretamente os efeitos de luz sobre as superfícies, melhorando a perceção de profundidade e forma. Paralelamente, a integração de coordenadas de textura possibilitou o mapeamento de imagens sobre os modelos 3D, conferindo-lhes um aspeto mais realista.

Área de Aplicação: Representação visual realista de objetos 3D em ambientes gráficos, através da aplicação de texturas e de iluminação; Utilização de normais para melhorar a perceção de forma e profundidade; Mapeamento de coordenadas de textura para aplicar imagens sobre superfícies 3D.

Palavras-Chave: Normais, Coordenadas de Texturas, Iluminação, Texturização.

Índice

Lista de Figuras

1. Introdução

Nesta fase do projeto, o objetivo principal consistiu na introdução de funcionalidades avançadas de texturização e iluminação na engine, de forma a melhorar significativamente o realismo visual das cenas renderizadas. Para tal, foi necessário atualizar o generator, de modo a incluir coordenadas de textura e normais por vértice nos ficheiros resultantes da geração da primitiva.

2. Arquitetura Desenvolvida

Para esta fase, a estrutura da fase anterior foi mantida. Apenas foram alteradas:

- A struct Primitive, para que englobasse as respetivas normais e coordenadas de textura
- A struct XMLDataFormat, para armazenar a informação das texturas e da iluminação
- A struct Transform, para incluir cada passo da transformação, permitindo a ordem correta das mesmas.

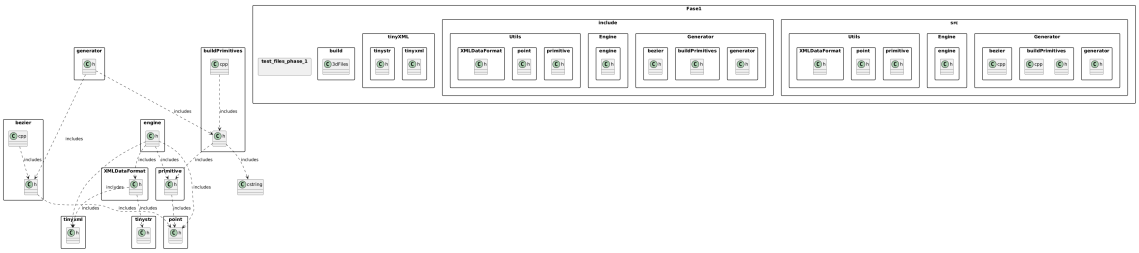

Figura 1: Arquitetura para a Fase 4

3. Generator

Nesta fase, foram implementadas a geração de normais e texCoords no generator para suportar as funcionalidades de iluminação e texturização na engine. Para além disso foi adicionada a primitiva Cilindro, de modo a ser possível criar novos objetos com textura para demonstração da aplicação de texturas a primitivas.

3.1. Geração de Normais

3.1.1. Plane

O plano é orientado segundo o eixo especificado. Para o eixo 'Y' com altura (h):

- Se $h \geq 0$, $\boldsymbol{n} = (0, 1, 0)$.
- Se $h < 0$, $\boldsymbol{n} = (0, -1, 0)$.

3.1.2. Box

Cada face é um plano, ou seja, herda as respetivas normais:

- Topo ('Y', $h = \frac{l}{2}$): $\boldsymbol{n} = (0, 1, 0)$.
- Fundo ('Y', $h = -\frac{l}{2}$): $\boldsymbol{n} = (0, -1, 0)$.
- Frente ('Z', $h = -\frac{l}{2}$): $\boldsymbol{n} = (0, 0, -1)$.
- Trás ('Z', $h = \frac{l}{2}$): $\boldsymbol{n} = (0, 0, 1)$.
- Direita ('X', $h = -\frac{l}{2}$): $\boldsymbol{n} = (-1, 0, 0)$.
- Esquerda ('X', $h = \frac{l}{2}$): $\boldsymbol{n} = (1, 0, 0)$.

As normais são perpendiculares a cada face, garantindo iluminação correta em cada lado do cubo.

3.1.3. Sphere

Para $(x, y, z) = (r \sin\varphi \sin\theta, r \cos\varphi, r \sin\varphi \cos\theta)$, com $\theta \in [0, 2\pi]$, $\varphi \in [0, \pi]$:

- $\boldsymbol{n} = \left(\frac{x}{r}, \frac{y}{r}, \frac{z}{r}\right) = (\sin\varphi \sin\theta, \cos\varphi, \sin\varphi \cos\theta)$.

3.1.4. Cone

Para um cone com raio r, altura h, s slices, t stacks:

- **Base**: $\boldsymbol{n} = (0, -1, 0)$.
- **Corpo**: Para $(x, y, z) = (r' \cos\theta, y, r' \sin\theta)$, $r' = r\left(1 - \frac{y}{h}\right)$, $y \in [0, h]$:
  - $\boldsymbol{n} = \left(\cos\theta, \frac{r}{\sqrt{r^2+h^2}}, \sin\theta\right)$.
  - Normalizada: $\boldsymbol{n} = \frac{\boldsymbol{n}}{\sqrt{\cos^2\theta + \left(\frac{r}{\sqrt{r^2+h^2}}\right)^2 + \sin^2\theta}}$.
- **Topo**: $\boldsymbol{n} = (0, 1, 0)$.

3.1.5. SaturnRing

Anel com raio interno r_{int}, externo r_{ext}, s slices, t stacks:

- Face Superior: $\boldsymbol{n} = (0, 1, 0)$.

- Face Inferior: $\boldsymbol{n} = (0, -1, 0)$.

3.1.6. Cylinder

Cilindro com raio r, altura h, s fatias, t camadas.
- Base inferior: $\boldsymbol{n} = (0, -1, 0)$.
- Base superior: $\boldsymbol{n} = (0, 1, 0)$.
- Corpo: Para $(x, y, z) = (r\cos\theta, y, r\sin\theta)$, $y \in \left[-\frac{h}{2}, \frac{h}{2}\right]$: $\boldsymbol{n} = (\cos\theta, 0, \sin\theta)$.

3.1.7. Bezier patch

Baseadas nas derivadas $\partial\frac{\boldsymbol{s}}{\partial}u$ e $\partial\frac{\boldsymbol{s}}{\partial}v$:

$$\partial\frac{\boldsymbol{s}}{\partial}u = \sum_{i=0}^{3}\sum_{j=0}^{3}\left(\partial\frac{B_{i(u)}}{\partial}u\right)B_{j(v)}\boldsymbol{p}_{i,j}$$

$$\partial\frac{B_{i(u)}}{\partial}u = 3m_{\{i,0\}}u^2 + 2m_{\{i,1\}}u + m_{\{i,2\}}$$

$$\partial\frac{\boldsymbol{s}}{\partial}v = \sum_{i=0}^{3}\sum_{j=0}^{3}B_{i(u)}\left(\partial\frac{B_{j(v)}}{\partial}v\right)\boldsymbol{p}_{i,j}$$

$$\partial\frac{B_{j(v)}}{\partial}v = 3m_{\{j,0\}}v^2 + 2m_{\{j,1\}}v + m_{\{j,2\}}$$

Calculadas via produto vetorial:

$$\boldsymbol{n} = \left(\partial\frac{\boldsymbol{s}}{\partial}v\right) \times \left(\partial\frac{\boldsymbol{s}}{\partial}u\right)$$

e posteriormente normalizadas.

3.2. Geração de texCoords

3.2.1. Plane

Para (x, y, z):
- 'Y': $u = \frac{x}{l} + 0.5$, $v = 1 - \left(\frac{z}{l} + 0.5\right)$, com $x, z \in \left[-\frac{l}{2}, \frac{l}{2}\right]$.
- 'X': $u = \frac{z}{l} + 0.5$, $v = \frac{x}{l} + 0.5$.
- 'Z': $u = \frac{x}{l} + 0.5$, $v = 1 - \left(\frac{y}{l} + 0.5\right)$.

3.2.2. Box

- **Topo/Fundo**: $u = \frac{x}{l} + 0.5$, $v = 1 - \left(\frac{z}{l} + 0.5\right)$.
- **Frente/Trás**: $u = \frac{x}{l} + 0.5$, $v = 1 - \left(\frac{y}{l} + 0.5\right)$.
- **Direita/Esquerda**: $u = \frac{z}{l} + 0.5$, $v = \frac{x}{l} + 0.5$.
- Assegura continuidade nas arestas.

3.2.3. Sphere

- $u = \frac{\theta}{2\pi}$, com $\theta = 2\pi \cdot \frac{\text{slice}}{s}$.
- $v = 1 - \frac{\varphi}{\pi}$, com $\varphi = \pi \cdot \frac{\text{stack}}{t}$.

3.2.4. Cone

- **Base**: $u = 0.5 + 0.5\cos\theta$, $v = 0.5 + 0.5\sin\theta$.

- **Corpo**: $u = \frac{\theta}{2\pi}$, $v = \frac{y}{h}$.
- **Topo**: $u = 0.5$, $v = 1.0$.

3.2.5. SaturnRing

Para raio $r = r_{\text{int}} + (r_{\text{ext}} - r_{\text{int}}) \cdot \frac{\text{stack}}{t}$, $\theta = 2\pi \cdot \frac{\text{slice}}{s}$:
- $u = \frac{\theta}{2\pi}$.
- $v = \frac{r - r_{\text{int}}}{r_{\text{ext}} - r_{\text{int}}}$.

3.2.6. Cylinder

- **Bases**: $u = 0.5 + 0.5\cos\theta$, $v = 0.5 + 0.5\sin\theta$.
- **Corpo**: $u = \frac{\theta}{2\pi}$, $v = \frac{y + \frac{h}{2}}{h}$.

3.2.7. Bezier patch

Para uma mesh com $u = \frac{i}{n}$, $v = \frac{j}{n}$ ($i, j = 0, ..., n$, n é o nível de tesselação):

$$u = \frac{i}{n}, \quad v = \frac{j}{n}$$

Através do ponto s(u,v) calculado anteriormente para as normais.

4. Engine

4.1. Iluminação

4.1.1. Iluminação do Espaço

Foram definidos três tipos de luzes para iluminar a cena:

- **Point**: Caracterizadas apenas pela sua posição no espaço, definida como um ponto $\boldsymbol{p} = (x, y, z)$.
- **Directional**: Especificadas por um vetor de direção $\boldsymbol{d} = \big(d_x, d_y, d_z\big)$, que indica a orientação da luz.
- **Spotlight**: Requerem a posição $\boldsymbol{p}$, o vetor de direção $\boldsymbol{d}$, o ângulo de corte θ (em graus, limitando o cone de luz) e um expoente de atenuação e para controlar a intensidade.

A ativação da iluminação foi realizada com as funções OpenGL:
- `(GL_LIGHTING)`, para habilitar o sistema de iluminação.
- `(GL_LIGHTi)`, para ativar luzes individuais.

A configuração das luzes foi feita com a função **glLightfv**, utilizando parâmetros como:
- **GL_POSITION**: Define a posição $\boldsymbol{p}$ (ou direção para luzes direcionais, com $w = 0$).
- **GL_SPOT_DIRECTION**: Especifica o vetor $\boldsymbol{d}$ para focos.
- **GL_SPOT_CUTOFF**: Define o ângulo θ do cone de luz.
- **GL_SPOT_EXPONENT**: Controla a atenuação com o expoente e.

4.1.2. Iluminação dos Objetos

Os objetos da cena possuem quatro componentes de cor que determinam sua interação com a luz:

- **Difusa**: Representa a reflexão da luz espalhada uniformemente, definida por uma cor $\boldsymbol{c}_d = (R_d, G_d, B_d, A_d)$.
- **Especular**: Modela reflexos brilhantes, especificada por $\boldsymbol{c}_s = (R_s, G_s, B_s, A_s)$.
- **Ambiental**: Simula a luz ambiente refletida, dada por $\boldsymbol{c}_a = (R_a, G_a, B_a, A_a)$.
- **Emissiva**: Representa a luz emitida pelo objeto, definida por $\boldsymbol{c}_e = (R_e, G_e, B_e, A_e)$.

Estas componentes são configuradas com a função glMaterialfv, usando os parâmetros:
- GL_DIFFUSE para $\boldsymbol{c}_d$.
- GL_SPECULAR para $\boldsymbol{c}_s$.
- GL_AMBIENT para $\boldsymbol{c}_a$.
- GL_EMISSION para $\boldsymbol{c}_e$.

O brilho dos reflexos especulares é ajustado com a função e o parâmetro , que recebe um valor $s \in [0, 128]$ para controlar a concentração do reflexo.

4.2. Texturas

4.2.1. Conversão para Formato RGBA

Após o carregamento da imagem, procede-se à conversão para o formato RGBA com canais de 8 bits por componente. Esta escolha visa garantir a compatibilidade com a maioria das funções de textura do OpenGL e assegurar que existe um canal alfa disponível para transparência, mesmo que a imagem original não o contenha. Esta uniformização facilita o pipeline de renderização e previne comportamentos gráficos inesperados.

4.2.2. Geração de Textura no OpenGL

O OpenGL não manipula diretamente imagens; necessita que estas sejam convertidas para texturas associadas a identificadores gerados com `glGenTextures`. Após gerar o ID, este é ligado ao contexto ativo com `glBindTexture`. Esta associação permite que os dados da imagem sejam enviados para a memória do GPU e utilizados durante a renderização.

4.2.3. Configuração dos Parâmetros de Textura

As opções definidas para o **wrapping** (`GL_REPEAT`) e para o **minification filter** (`GL_NEAREST`) refletem decisões orientadas para diferentes tipos de aplicação. O uso de `GL_REPEAT` permite que a textura seja repetida indefinidamente sobre uma superfície, o que é ideal para terrenosdos planetas, por exemplo. Por outro lado, a escolha de `GL_NEAREST` prioriza desempenho e um estilo visual mais nítido, sem interpolação, destacando a importância de cada texel.

4.2.4. Envio dos Dados para o GPU

A função `glTexImage2D` é responsável por transferir os dados da imagem já convertida para a memória do GPU. Aqui, especifica-se o formato interno (`GL_RGBA`), a resolução da imagem e o tipo de dados. Esta operação é crítica, pois determina como os dados são interpretados no espaço gráfico e como serão acedidos durante o rastreamneto dos objetos.

4.2.5. Geração Automática de Mipmaps

A utilização de mipmaps melhora significativamente a performance e qualidade visual em cenas 3D. Ao gerar múltiplas versões da textura com resoluções reduzidas (`glGenerateMipmap`), o sistema gráfico pode selecionar a resolução apropriada com base na distância da câmara ao objeto, reduzindo artefactos visuais como aliasing e melhorando o desempenho do GPU.

4.2.6. Libertação de Recursos Temporários

Após a transferência dos dados para o GPU, os dados originais na RAM tornam-se redundantes. A remoção destas imagens temporárias com `ilDeleteImages` é essencial para prevenir memory leaks e manter a aplicação leve.

4.2.7. Considerações de Eficiência e Manutenção

Ao encapsular todo o processo de carregamento e configuração numa função modular com cache, o sistema torna-se escalável e de fácil manutenção. Esta abordagem reduz a duplicação de código e facilita o diagnóstico de erros relacionados com texturas, além de preparar o projeto para futuras extensões, como suporte a texturas normais, especulares ou animações baseadas em mapas. A integração cuidadosa do carregamento, conversão e mapeamento de texturas garante que o motor gráfico oferece não apenas qualidade visual, mas também robustez e eficiência na gestão de recursos.

5. Extras Adicionados na Fase 4

5.1. Câmera FPS

Foi implementada uma câmera em primeira pessoa (FPS), ideal para visualizar a demonstração do um sistema solar devido à sua navegação imersiva, permitindo "voar" entre planetas e explorar livremente, com movimentos controlados por teclas e rato.

5.1.1. Estrutura

- **Posição**: $\boldsymbol{p} = (x_c, y_c, z_c)$, inicial $(0, 1.75, 5)$.
- **Direção**: Vetor unitário $\boldsymbol{d} = \left(d_x, d_y, d_z\right)$, baseado em **yaw** ($\psi$, inicial $-90°$) e **pitch** (θ, inicial $0°$, limitado a $[-89°, 89°]$).
- **Parâmetros**: Velocidade $s = 0.2$, sensibilidade $k = 0.1$.

5.1.2. Direção

Calcula $\boldsymbol{d}$ com base em ψ e θ:

- $d_x = \cos(\theta_r)\cos(\psi_r)$
- $d_y = \sin(\theta_r)$
- $d_z = \cos(\theta_r)\sin(\psi_r)$

Normalizado: $\boldsymbol{d} = \frac{d_x, d_y, d_z}{\sqrt{d_x^2 + d_y^2 + d_z^2}}$.

5.1.3. Movimento

Desloca $\boldsymbol{p}$ via teclas:

- Frente: $\boldsymbol{p} = \boldsymbol{p} + s * \boldsymbol{d}$.
- Trás: $\boldsymbol{p} = \boldsymbol{p} - s * \boldsymbol{d}$.
- Lateral ($\boldsymbol{r} = \boldsymbol{d} \times (0, 1, 0)$, normalizado):
  - Esquerda: $\boldsymbol{p} = \boldsymbol{p} - s * \boldsymbol{r}$.
  - Direita: $\boldsymbol{p} = \boldsymbol{p} + s * \boldsymbol{r}$.

5.1.4. Rotação

Ajusta ψ e θ pelo rato (Δx, Δy):

- $\psi = \psi + k\Delta x$,
- $\theta = \max(-89°, \min(89°, \theta + k\Delta y))$

Atualiza $\boldsymbol{d}$ após cada rotação.

6. Demo Scene - Sistema Solar

Na seguinte imagem encontra-se a representação do sistema solar desenvolvida pelo grupo ao longo da realização do trabalho prático. É possível identificar todos os planetas do sistema solar incluindo as luas de Marte Fobos e Deimos, e as luas Europa, Io, Ganimedes e Calisto de Júpiter.

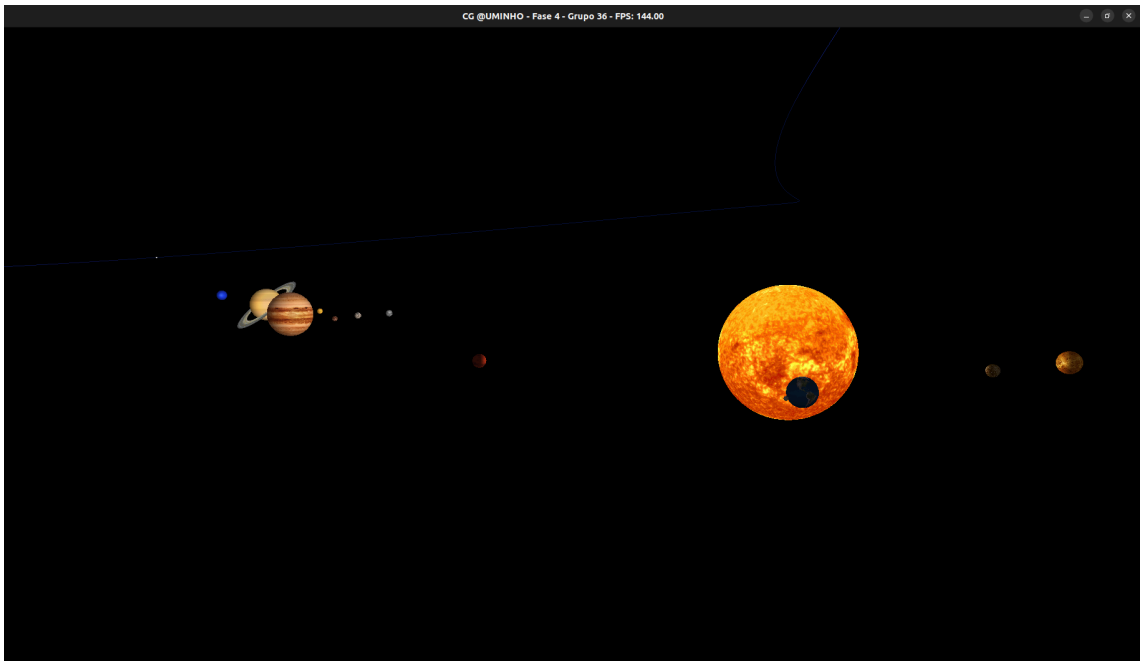

Figura 2: Sistema Solar - Final

Esta imagem representa o sistema solar zoomed-in com destaque para as componentes de iluminação refletidas nos diferentes planetas.

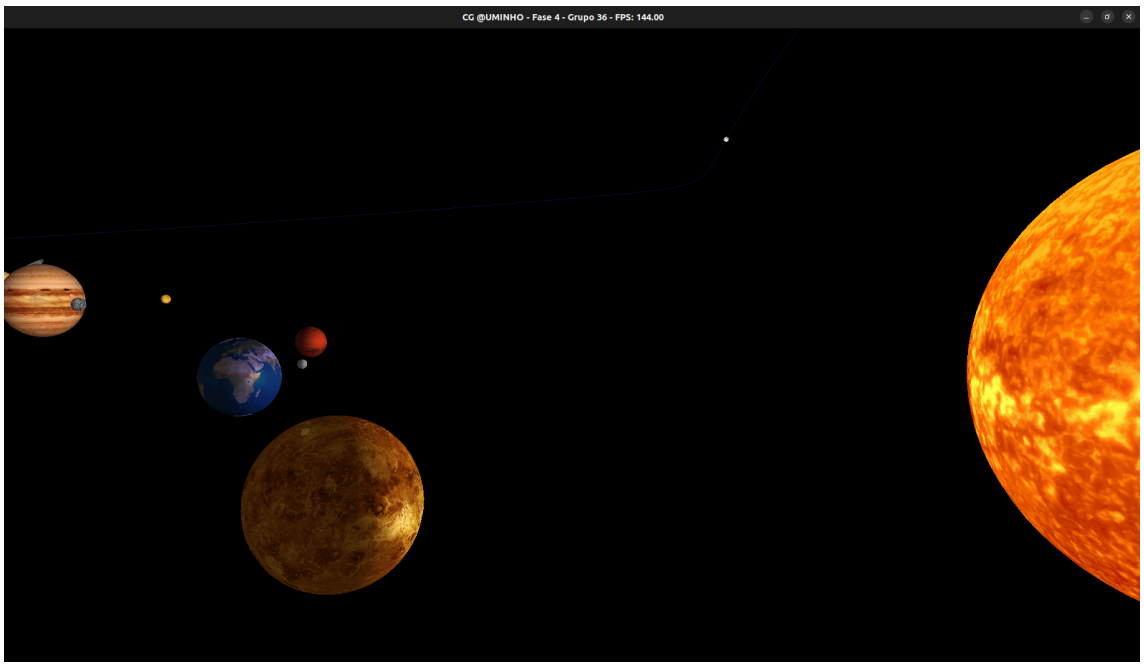

Figura 3: Sistema Solar - Zoomed-in

Também aplicamos uma textura genérica para o cometa que desenvolvemos.

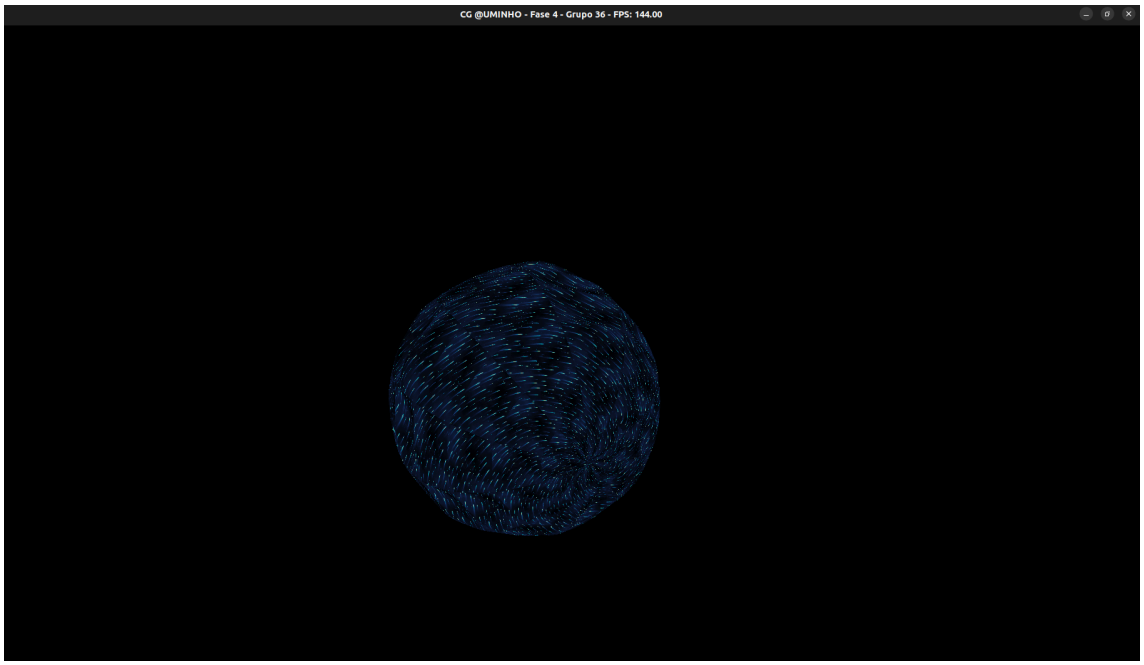

Figura 4: Cometa

7. Testes & Criação de Novos Objetos com Texturas

Recriação dos cenários de teste propostos pelos docentes.

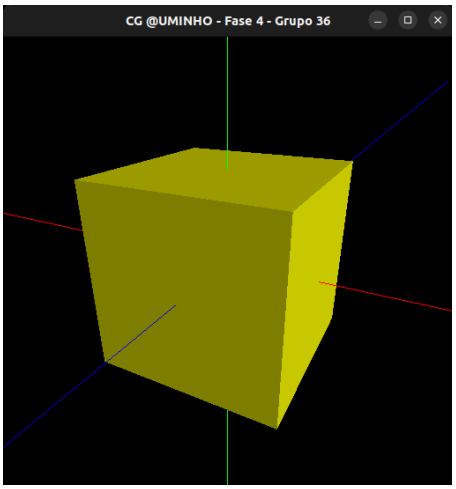

Figura 5: test_4_1.xml

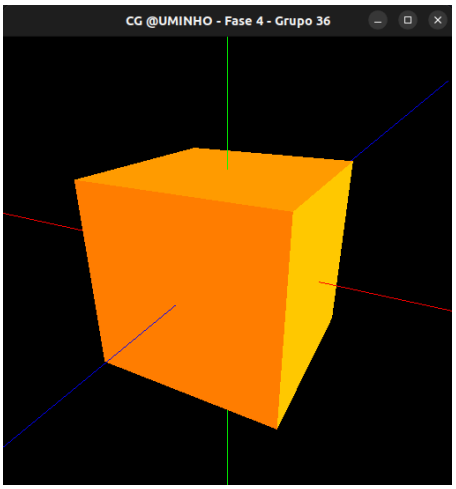

Figura 6: test_4_2.xml

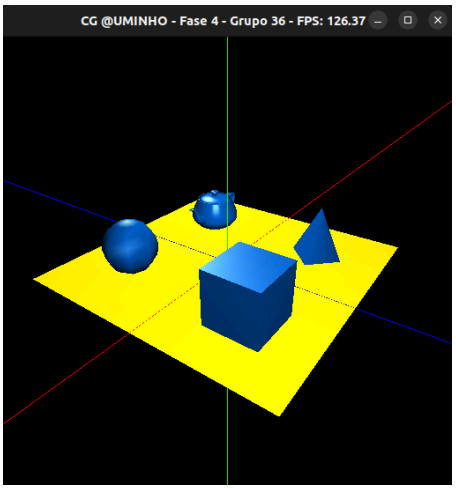

Figura 7: test_4_3.xml

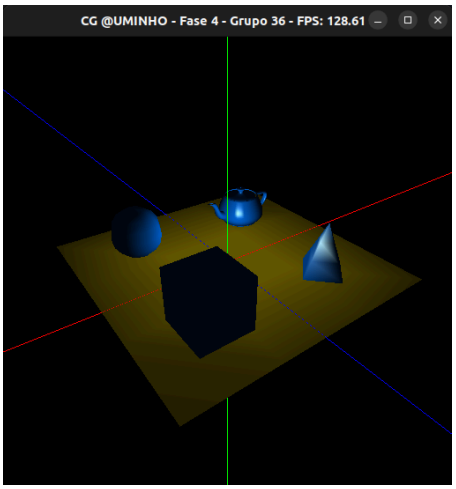

Figura 8: test_4_4.xml

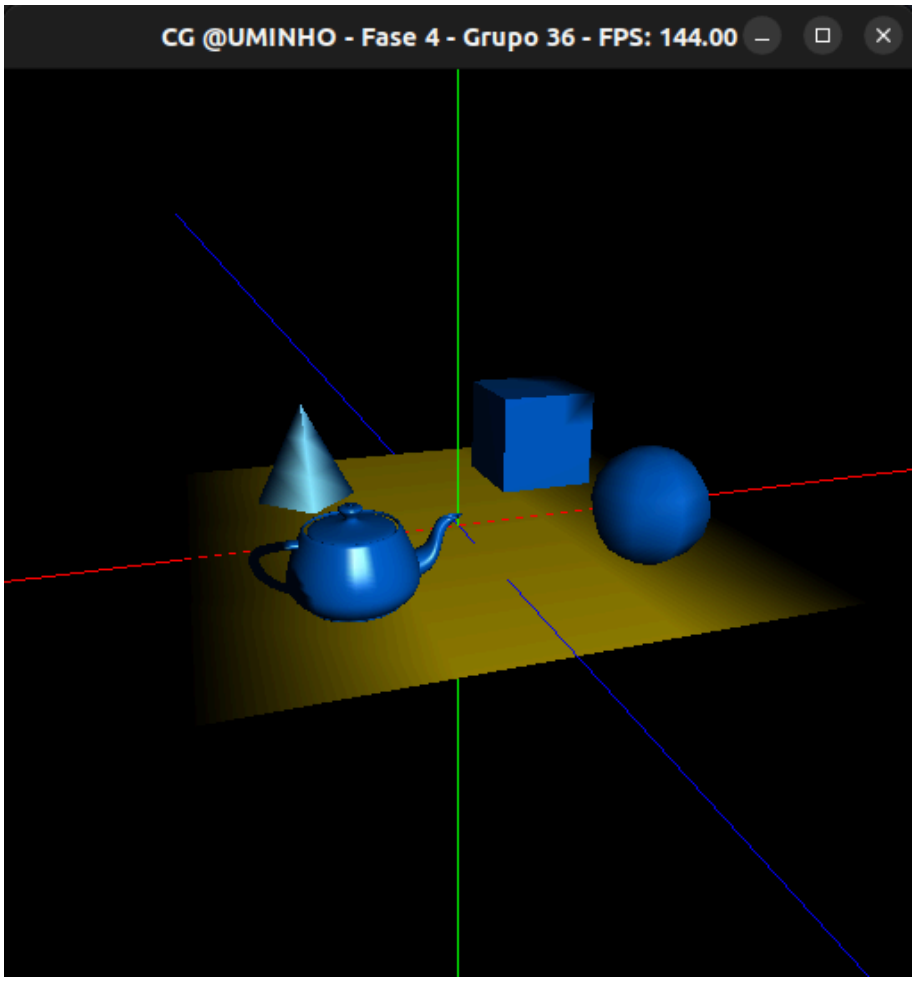

Figura 9: test_4_5.xml

Como trabalho extra, recolhemos diferentes imagens de textura para simular objetos do mundo real. Com as primitivas criadas foi possível recriar, em OpenGL, uma caixa de madeira, um chapéu de festa e ainda aplicar textura ao teapot gerado a partir do ficheiro .patch da Fase 3.

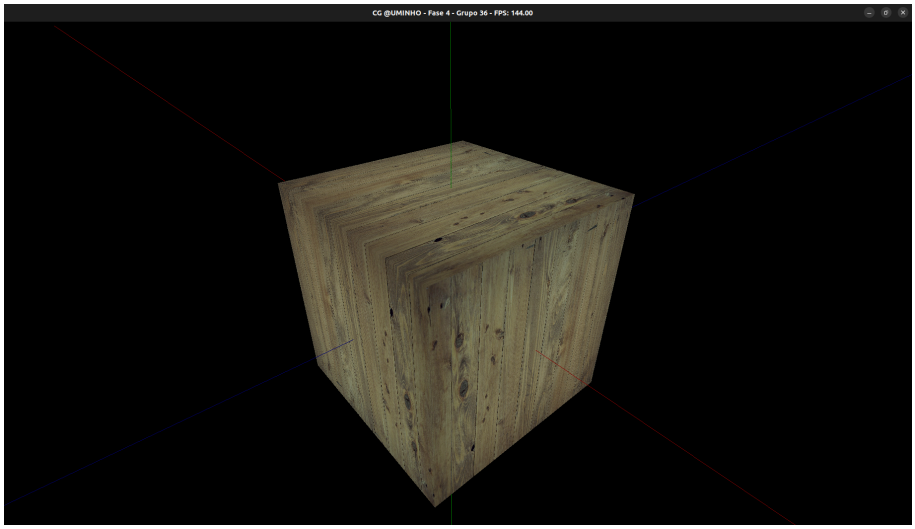

Figura 10: Caixa de Madeira

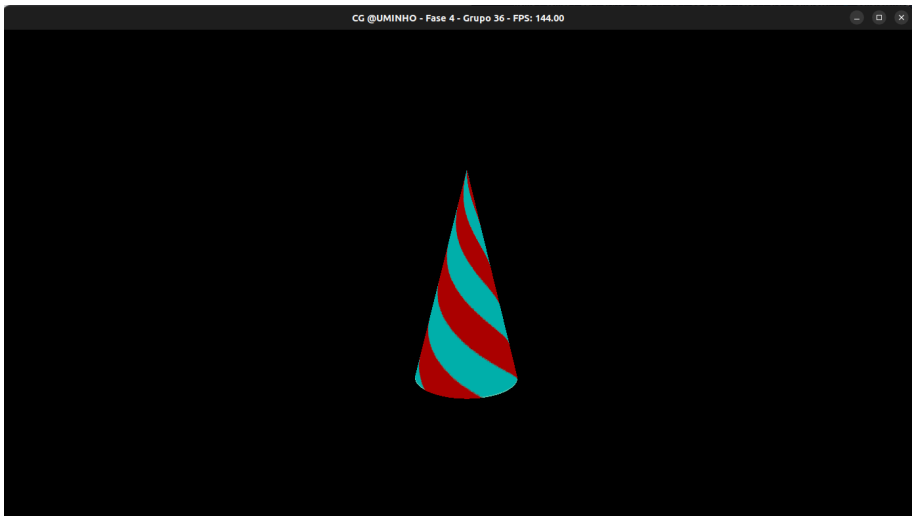

Figura 11: Chapéu de Festa

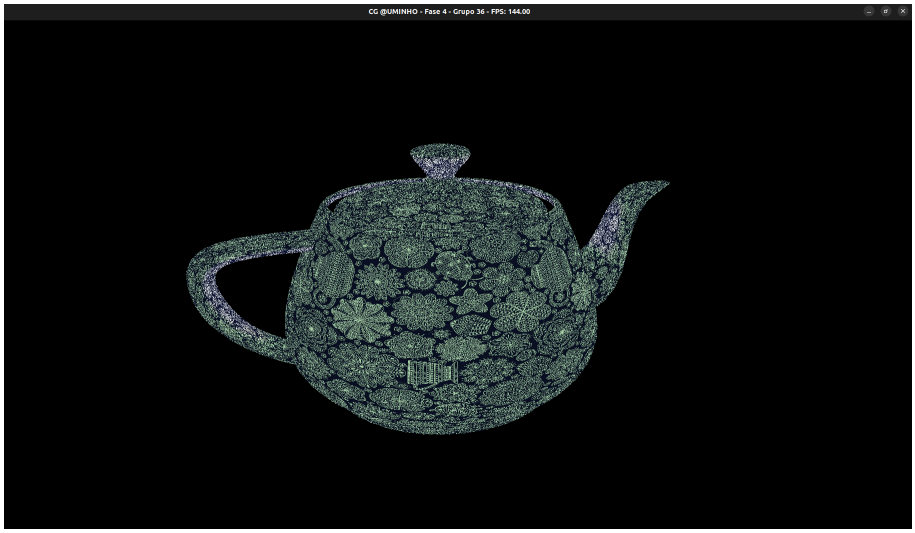

Figura 12: Teapot

8. Conclusão

Nesta última fase do projeto, tivemos a oportunidade de aplicar e consolidar os conhecimentos adquiridos ao longo das aulas práticas e teórico-práticas de Computação Gráfica, com especial foco nos temas de iluminação e texturização.

Através da implementação das normais e das coordenadas de textura nos modelos previamente desenvolvidos, conseguimos dar um passo importante rumo a um motor gráfico mais completo e visualmente realista.

O cálculo rigoroso das normais permitiu simular de forma mais precisa os efeitos da luz sobre as superfícies, enquanto a atribuição de coordenadas de textura possibilitou o mapeamento eficaz de imagens nos objetos 3D, enriquecendo consideravelmente o seu aspeto. Para além dos requisitos da fase, conseguimos ainda integrar funcionalidades adicionais, como a câmara em primeira pessoa, que aumentou a imersividade da navegação nas cenas renderizadas.

No geral, consideramos que esta fase foi concluída com sucesso. O trabalho desenvolvido reflete um progresso significativo desde as etapas iniciais, integrando todos os componentes necessários para criar um sistema gráfico robusto. A experiência adquirida foi valiosa e permitiu-nos consolidar de forma prática os conteúdos explorados ao longo do semestre.

Referências - APA

Free Textures for Basketball, Beach Ball, Pool Balls, Softball, and Tennis Balls. (2025). Robinwood.com. https://www.robinwood.com/Catalog/FreeStuff/Textures/TexturePages/BallMaps.html?utm_source=chatgpt.com

Category:Solar System Scope - Wikimedia Commons. (2022). Wikimedia.org. https://commons.wikimedia.org/wiki/Category:Solar_System_Scope

Solar System Scope. (2017). Solar System Scope. https://www.solarsystemscope.com/textures/

Planet Texture Map Collection. (2025). Planetpixelemporium.com. http://planetpixelemporium.com/planets.html

Ahn, S. H. (2024). OpenGL Cylinder, Prism & Pipe. Songho.ca. https://www.songho.ca/opengl/gl_cylinder.html

9. Manual de Execução

Mover para a diretoria build, caso exista.

Se não existir, criar a diretoria 'build' dentro de 'Fase4' e mover-se para esta.

```
mkdir build ; cd build
```

Executar os comandos de configuração e compilação de projetos que usam o sistema de build CMake:

```
cmake .. ; cmake --build .
```

Gerar as primitivas desejadas:

```
./generator plane 5 3 plane.3d

./generator box 4 8 box.3d

./generator sphere 2 15 15 sphere.3d

./generator cone 1 3 15 15 cone.3d

./generator ring 1 1.2 15 15 ring.3d

./generator cylinder 1 4 15 15 cylinder.3d
```

Gerar o cometa:

```
cd .. ; gcc -o comet cometgen.c -lm ; ./comet ; cd build
>> Ficheiro 'comet.patch' gerado com sucesso!

./generator patch ../comet.patch 10 comet.3d
```

Iniciar a engine com qualquer ficheiro .xml das pastas de teste (test_files_phase_*):

```
./engine ../test_files_phase_4/solar_system.xml
```

Neste caso, estamos a correr a engine com a cena final do sistema solar que desenvolvemos na Fase 4 do trabalho prático.

Em caso de dúvida, executar o seguinte comando que fornece instruções de como gerar as primitivas:

```
./generator --help
```