

Computação Gráfica
Relatório de Trabalho Prático

Afonso Magalhães
A95250

Diana Teixeira
A97516

Artur Leite
A97027

4 de junho de 2023

Conteúdo

1	Introdução	2
2	Generator	3
2.1	Plano	3
2.2	Caixa	4
2.3	Cone	4
2.4	Esfera	5
2.5	Cilindro	6
2.6	Torus	8
2.7	Patches de Bezier	8
3	Engine	10
3.1	Texturas	10
3.2	Cores	10
3.3	Luz	10
4	Scenes	11
4.1	Sistema Solar	11
4.2	Cenas Teste	18
5	Conclusão	22

Capítulo 1

Introdução

Com o semestre a terminar, e com o objectivo de melhorar substancialmente o nosso modelo do Sistema Solar, tomamos então, como ponto focal nesta ultima fase do trabalho, definir as luzes e texturas em falta.

Para tal, foi necessário efetuar algumas mudanças nos ficheiros que definimos anteriormente, nomeadamente no Generator e no Engine do programa. As alterações à script geradora do modelo irão permitir a presença de texturas nos diversos astros, bem como a presença de luzes no sistema em questão.

Isto dito, iremos relatar agora as mudanças efetuadas ao longo do nosso projeto, as quais permitiram a presença das texturas e luzes realistas mencionadas anteriormente.

Capítulo 2

Generator

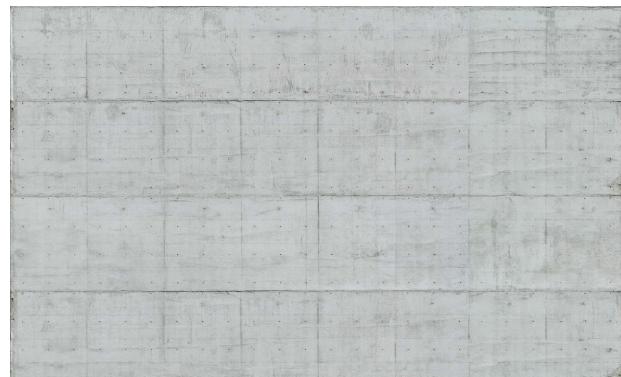
Tal como mencionado anteriormente, a alteração do modo de construção dos ficheiros .3d das primitivas já realizadas foi necessária, tendo sido alterado, para além dos pontos dos sólidos, os seus vetores normais, juntamente com as suas coordenadas de textura, visto uma textura em OpenGL estar contida num referencial 2D.

Tendo sido alteradas as seguintes primitivas:

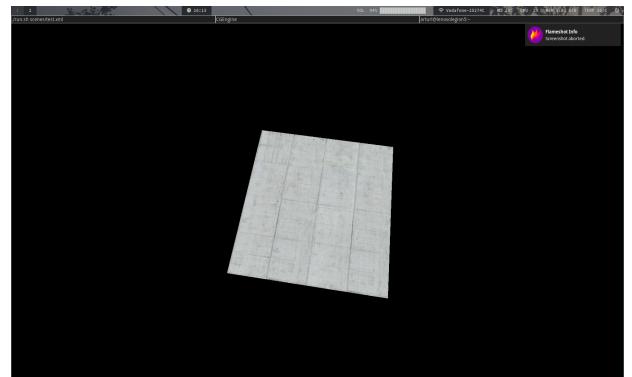
2.1 Plano

Sempre que um plano é gerado no nosso modelo, este irá estar paralelo ao plano xz. Tendo isso em conta, podemos aferir que as normais nesta primitiva irão ser sempre $(0,1,0)$.

Para aplicarmos a textura aos planos, basta repeti-la ao longo do plano, iterando sobre este N por M divisões, aplicando continuamente a mesma textura.



(a) Textura



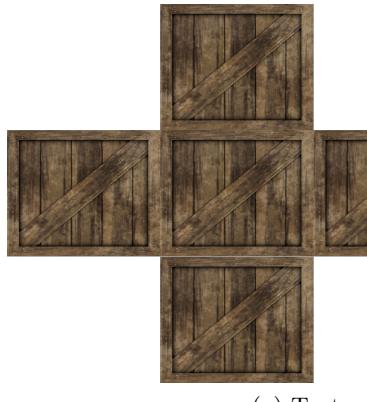
(b) Plano com textura

Figura 2.1: Textura e Plano com textura aplicada

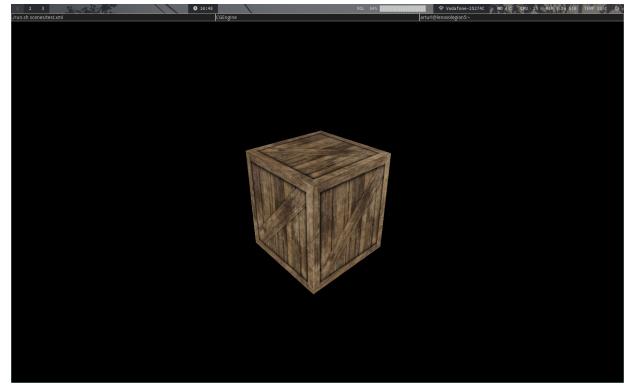
2.2 Caixa

Tendo em conta que podemos considerar uma caixa como um conjunto de planos que representam cada uma das faces, as normais dos pontos desta primitiva irão ser obtidas de forma semelhante. Os valores irão depender do lado aonde o ponto pretendido se encontra. A titulo de exemplo, caso este se encontre na base da caixa, a sua normal é $(0, -1, 0)$, caso se encontre no topo, a sua normal passa a ser $(0, 1, 0)$.

Por outro lado, para uma dada textura que queremos aplicar à caixa, iremos dividi-la em quatro partes horizontalmente e três partes verticalmente. Isto significa que cada face da caixa irá estar dividida em 12 triângulos, cada um deles sendo mapeado com a textura correspondente.



(a) Textura



(b) Caixa com textura

Figura 2.2: Textura e Caixa com textura aplicada

2.3 Cone

Primeiramente, com as conclusões retiradas das primitivas mencionadas anteriormente, podemos facilmente aferir que os pontos contidos na base do cone vão possuir uma normal igual a $(0, -1, 0)$. Apesar disso, os pontos restantes necessitam de um cálculo diferente. Iremos precisar de dois ângulos essenciais para este cálculo. Um angulo alpha, correspondente ao ângulo da slice onde o ponto se encontra. Um ângulo beta, obtido pela seguinte fórmula:

$$\text{atan} \left(\frac{\text{raio}}{\text{altura}} \right) \quad (2.1)$$

Retirada da seguinte figura:

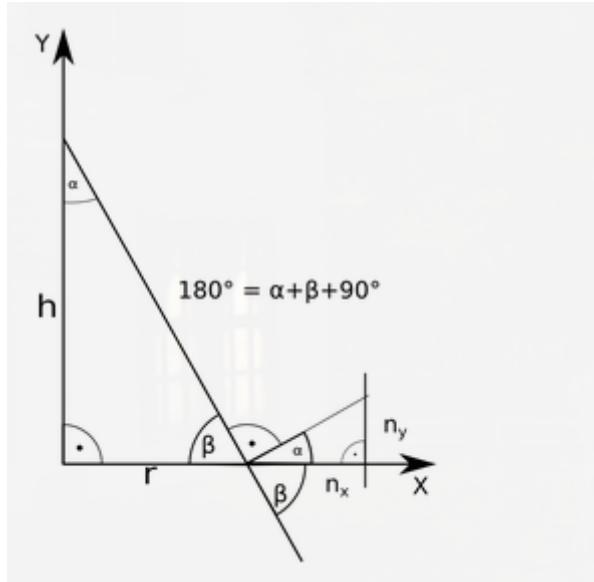


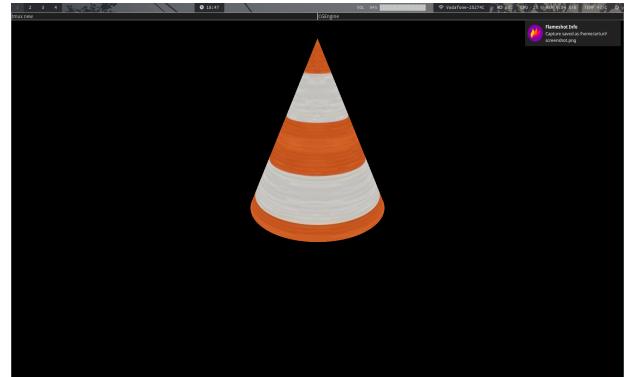
Figura 2.3: Caption

Quanto às texturas, mapeamos toda a textura como a lateral do cone, tendo todos os pontos constituintes da base do cone coordenadas de textura $(0, 0)$:

Tendo estas informações em conta, sabemos aplicar as texturas, tendo em conta a stack e a slice em que se encontra o ponto, de forma a adequar à posição da textura.



(a) Textura



(b) Cone com textura

Figura 2.4: Textura e Cone com textura aplicada

2.4 Esfera

O cálculo da normal de um ponto na esfera é efetuado através da normalização das coordenadas pontos, uma vez que podem representar o vetor que une ao centro da esfera. Para tal, obtemos a distância entre

o ponto e o centro da esfera e dividimos as coordenadas do ponto em questão por este valor, normalizando-o.

Em seguida, para corresponder um dado ponto a um ponto na textura, simplesmente dividimos o número da stack atual pelo número total de stacks e dividir o número da slice atual pelo número total de slices.



(a) Textura



(b) Esfera com textura

Figura 2.5: Textura e Esfera com textura aplicada

2.5 Cilindro

Neste caso, é fácil obter as normais dos pontos localizados na base e no topo do cilindro. Para a base, a normal é $(0, -1, 0)$, enquanto para o topo, a normal é $(0, 1, 0)$. No entanto, para os pontos localizados nas laterais, a normal é determinada pelo vetor que liga o ponto à projeção do centro da base do cilindro na mesma altura. Essa normal depende de um ângulo α e pode ser representada como:

$$(\sin \alpha, 0, \cos \alpha) \quad (2.2)$$

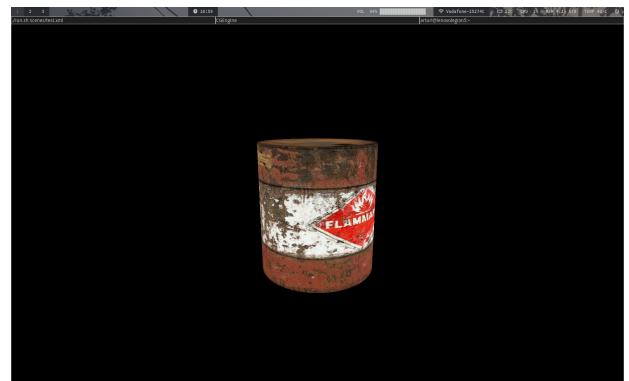
Quanto à textura, semelhante ao cone, vamos ter de alocar coordenadas dentro da textura pretendida de forma a identificar os elementos essenciais:

- Centro do topo: $(7/16, 3/16)$
- Centro da base: $(13/16, 3/16)$
- Início da lateral: $(0, 3/8)$
- Altura da lateral: $5/8$
- Raio: $3/16$

Tendo estas informações em conta, conseguimos deduzir as coordenadas de textura de todos os outros pontos. Os pontos presentes na lateral do cilindro são dependentes da slice da stack onde se encontram, pois isto influencia a distância ao ponto de início da lateral, enquanto que a base e o topo dependem do ângulo α , que por sua vez influencia a posição relativa à parte da textura que lhes é adequada.



(a) Textura



(b) Cilindro com textura

Figura 2.6: Textura e Cilindro com textura aplicada

2.6 Torus

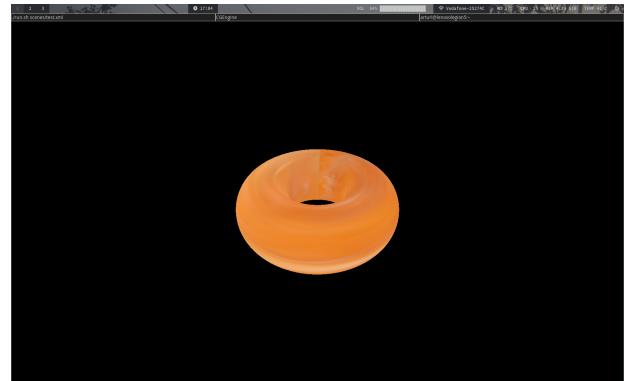
Por sua vez, as normas do torus seguem a mesma lógica que a esfera, mas orientam-se pelo centro do anel do torus, calculando-se de acordo com:

$$(\cos \phi * \cos \theta, \sin \phi, \cos \phi * \sin \theta) \quad (2.3)$$

Quanto às texturas, tal como na esfera, são obtidas dividindo a slice atual pelo número de slices e a stack atual pelo número de stacks.



(a) Textura



(b) Torus com textura

Figura 2.7: Textura e Torus com textura aplicada

2.7 Patches de Bezier

Para nos ser possível obter a normal de um ponto de um patch de Bezier, é necessário calcular dois vetores tangentes a esse ponto, os quais obtemos através das seguintes fórmulas:

$$\frac{\delta p(u, v)}{\delta u} = [3u^2 \ 2u \ 1 \ 0] \begin{pmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{pmatrix} M^T V^T \quad (2.4)$$

$$\frac{\delta p(u, v)}{\delta v} = U \ M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} 3v^2 \\ 2v \\ 1 \\ 0 \end{bmatrix} \quad (2.5)$$

Permitindo a normalização do cross product entre os dois vetores resultantes obter a normal pretendida.

Com isto, as coordenadas de textura são simples de obter, batendo dividir os vetores u e v pelo nível de tesselação.



(a) Textura



(b) Teapot com textura

Figura 2.8: Textura e Teapot com textura aplicada

Capítulo 3

Engine

Após vermos as modificações necessárias no Generator, passamos agora então para o Engine.

Ao qual foi necessária a ativação de funcionalidades de luz e texturas, juntamente com o requesito de ler e aplicar os novos dados dos ficheiros .3d gerados. Sendo importante mencionar que o ficheiro deverá então permitir indicar o tipo de luz e shininess de um objeto, bem como as origens da luz.

3.1 Texturas

Na nossa classe texture, realizamos um conjunto de funções que lê e carrega a texture para a memória, com o intuito de ser atribuída posteriormente no processo de renderização. Para além disso, na classe Renderer, quando uma textura que já foi carregada anteriormente é atribuída, não existe a necessidade de a carregar outra vez, o que leva a uma economização do processamento necessário.

3.2 Cores

Com o intuito de ser possível atribuir componentes de cor a um objeto, definimos uma classe Color onde estes atributos são definidos. Estes incluem diffuse, ambient, specular, emissive e shininess. Caso no ficheiro XML não estejam definidas as cores, definimos um valor default que será atribuído.

3.3 Luz

Para atribuir o elemento de luz no nosso modelo, definimos uma classe abstrata Light que possui funções de atribuição de luz ao ambiente e configurações adicionais de um método place(). Este método vai ser então crucial na diferenciação entre as classes que dão Override à classe Light. Estas incluem:

- DirectionalLight: define uma luz que segue uma direção;
- PointLight: define um ponto que emite luz em todas as direções, semelhante ao sol;
- SpotLight: define uma luz semelhante ao pointLight, mas define direção e raio aonde esta será emitida.

Capítulo 4

Scenes

4.1 Sistema Solar

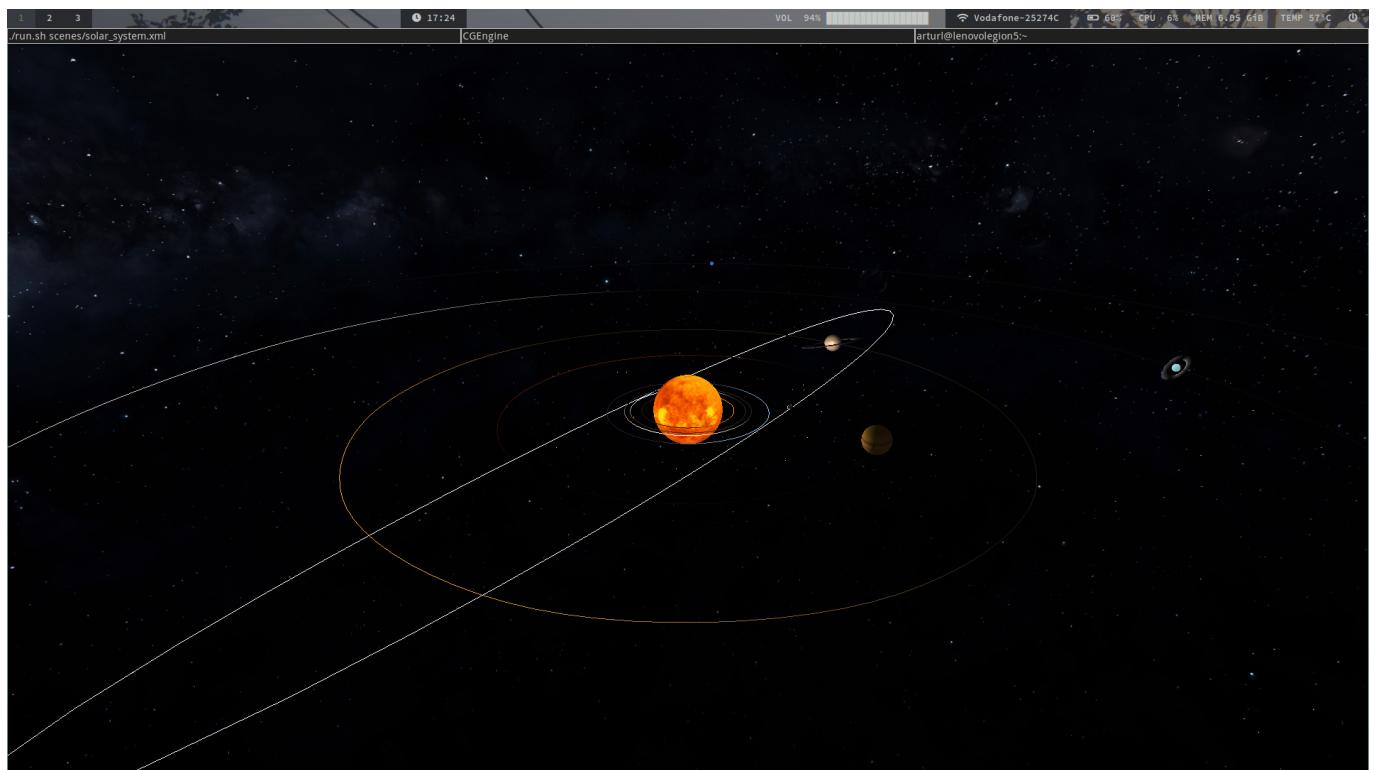


Figura 4.1: Sistema Solar

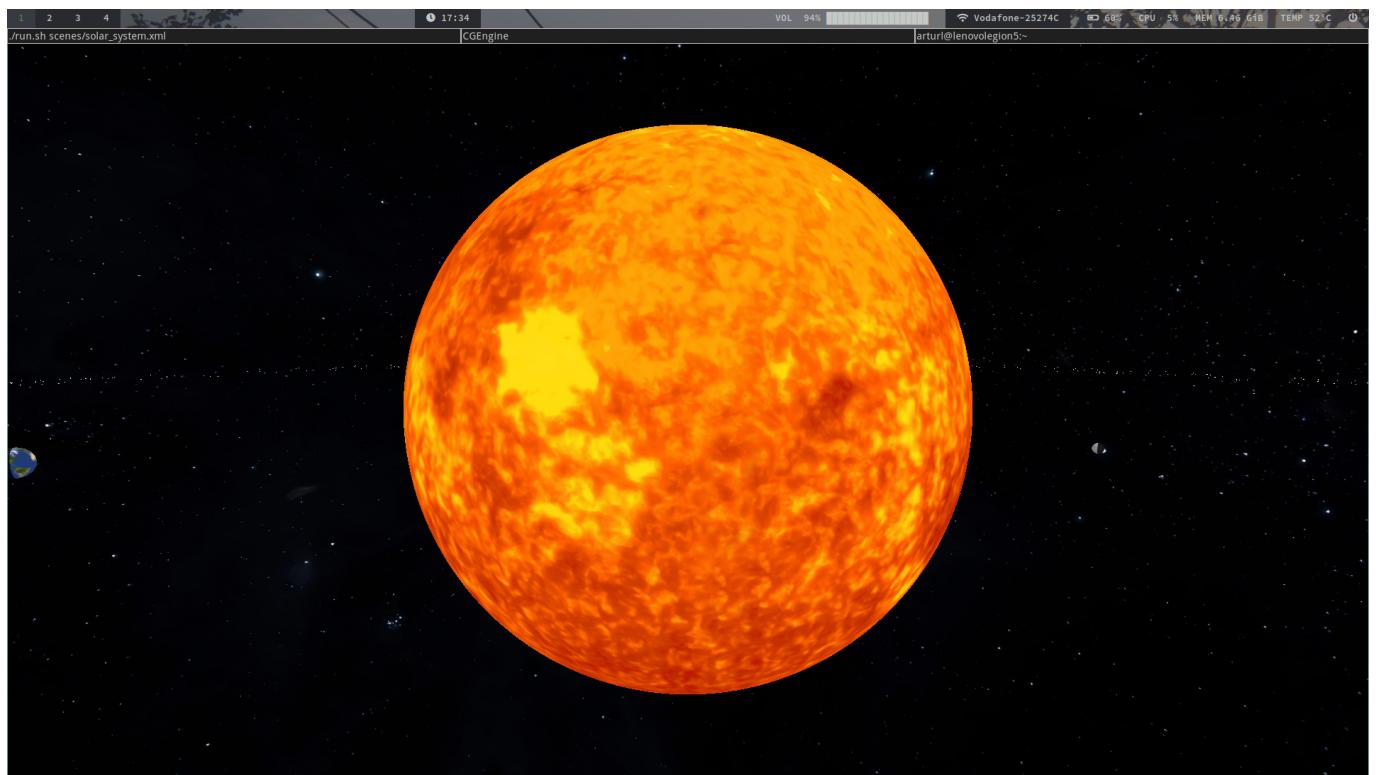


Figura 4.2: Sol



Figura 4.3: Mercurio

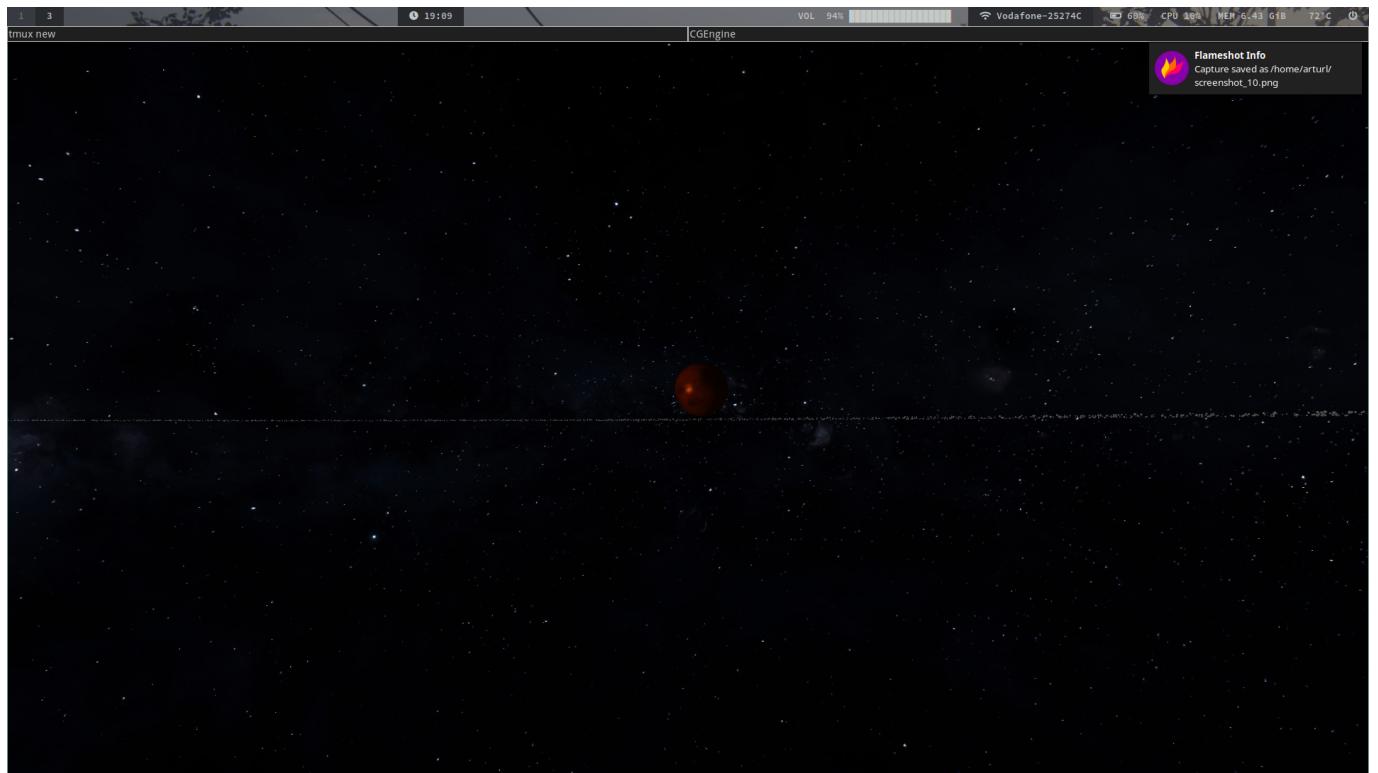


Figura 4.4: Venus

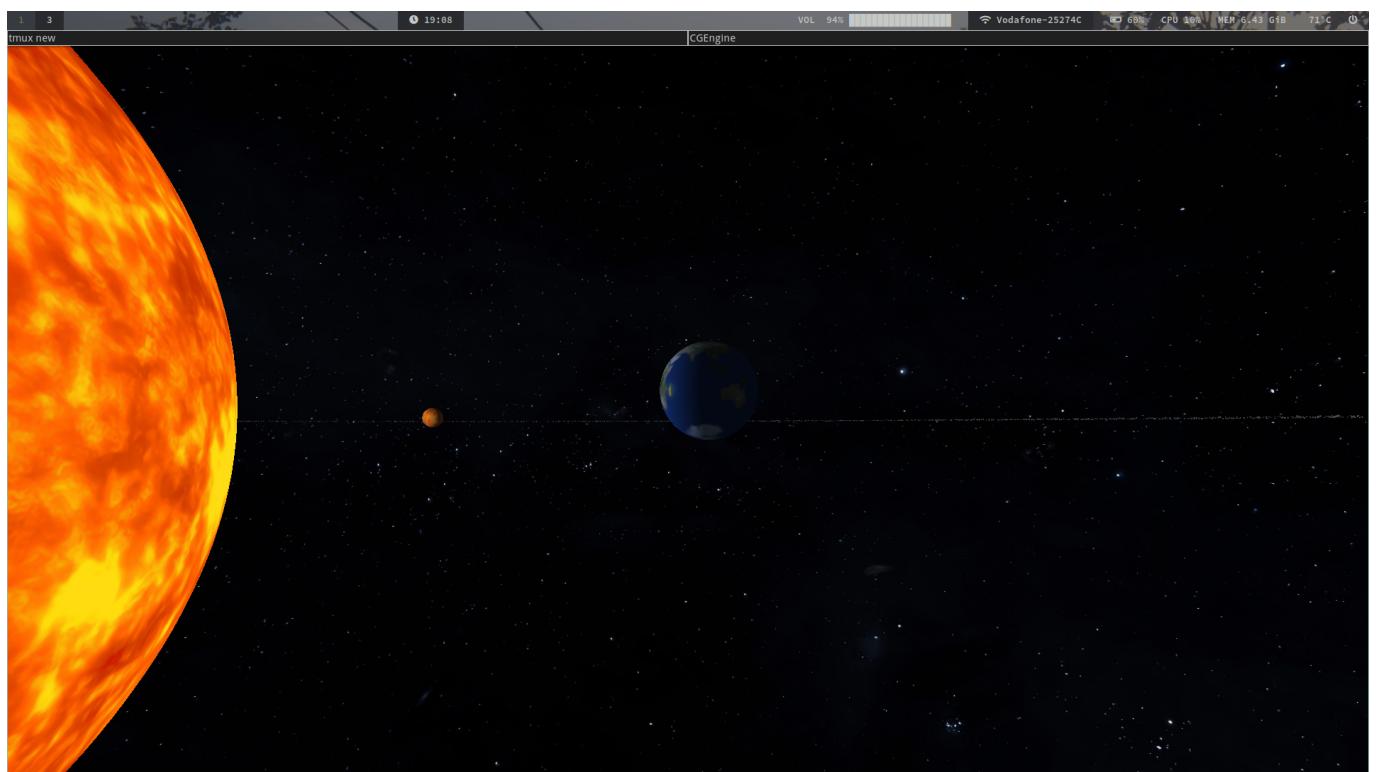


Figura 4.5: Terra



Figura 4.6: Marte



Figura 4.7: Cintura de Asteroides

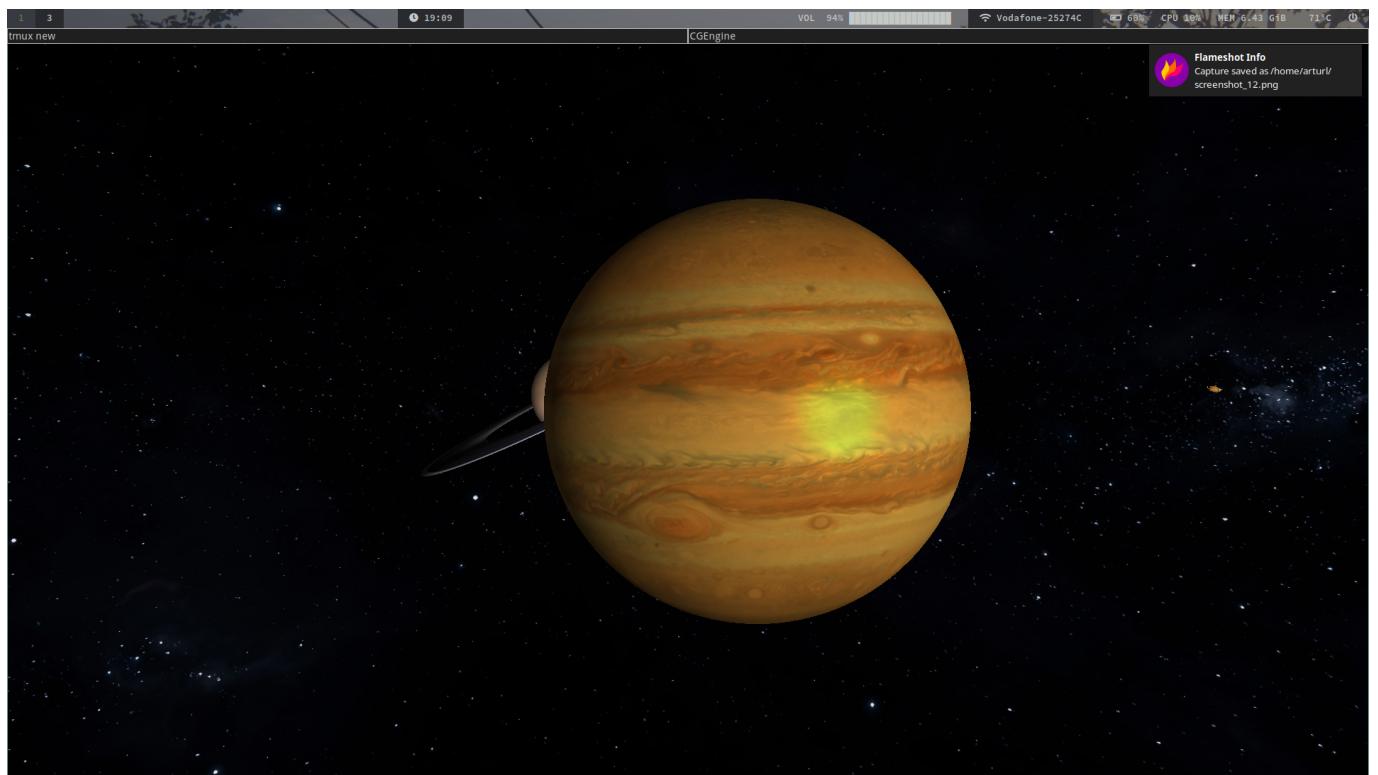


Figura 4.8: Jupiter



Figura 4.9: Saturno

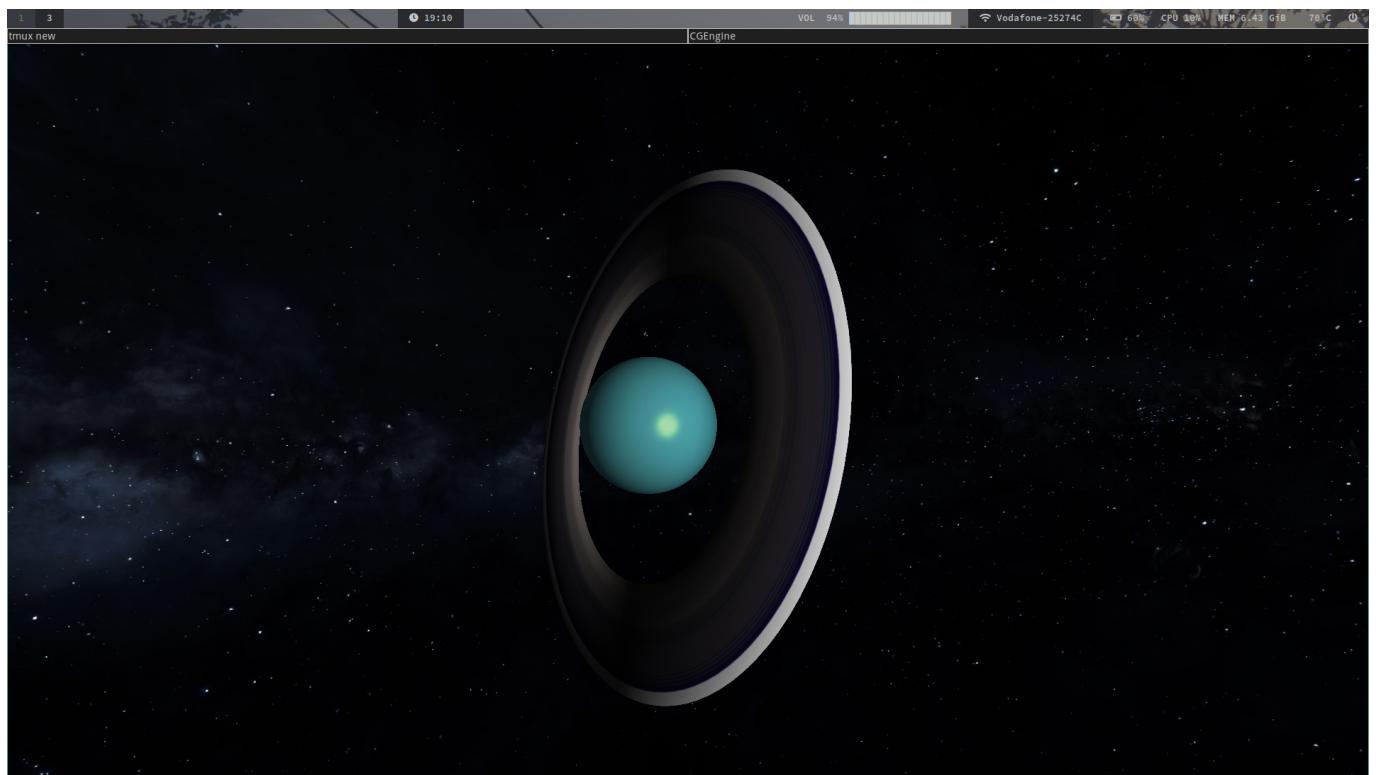


Figura 4.10: Urano

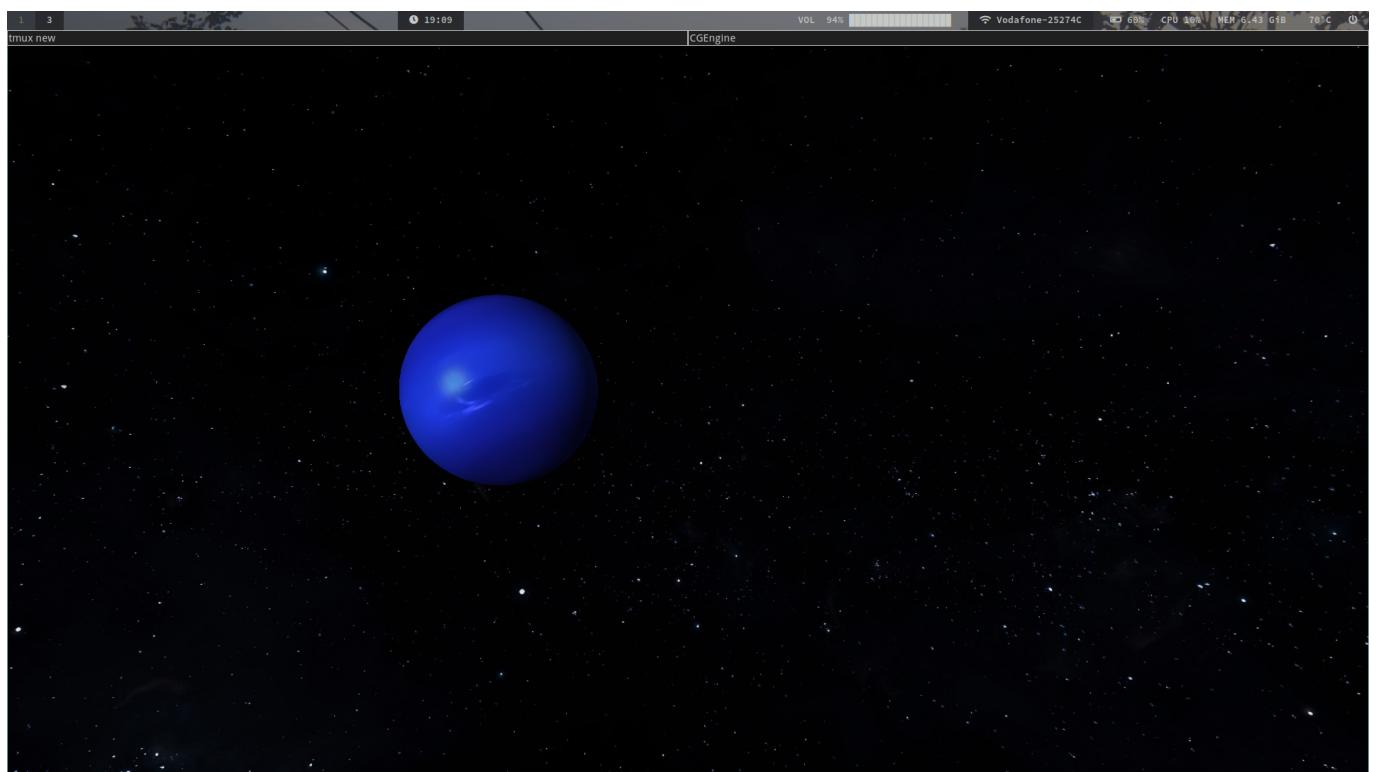


Figura 4.11: Neptuno

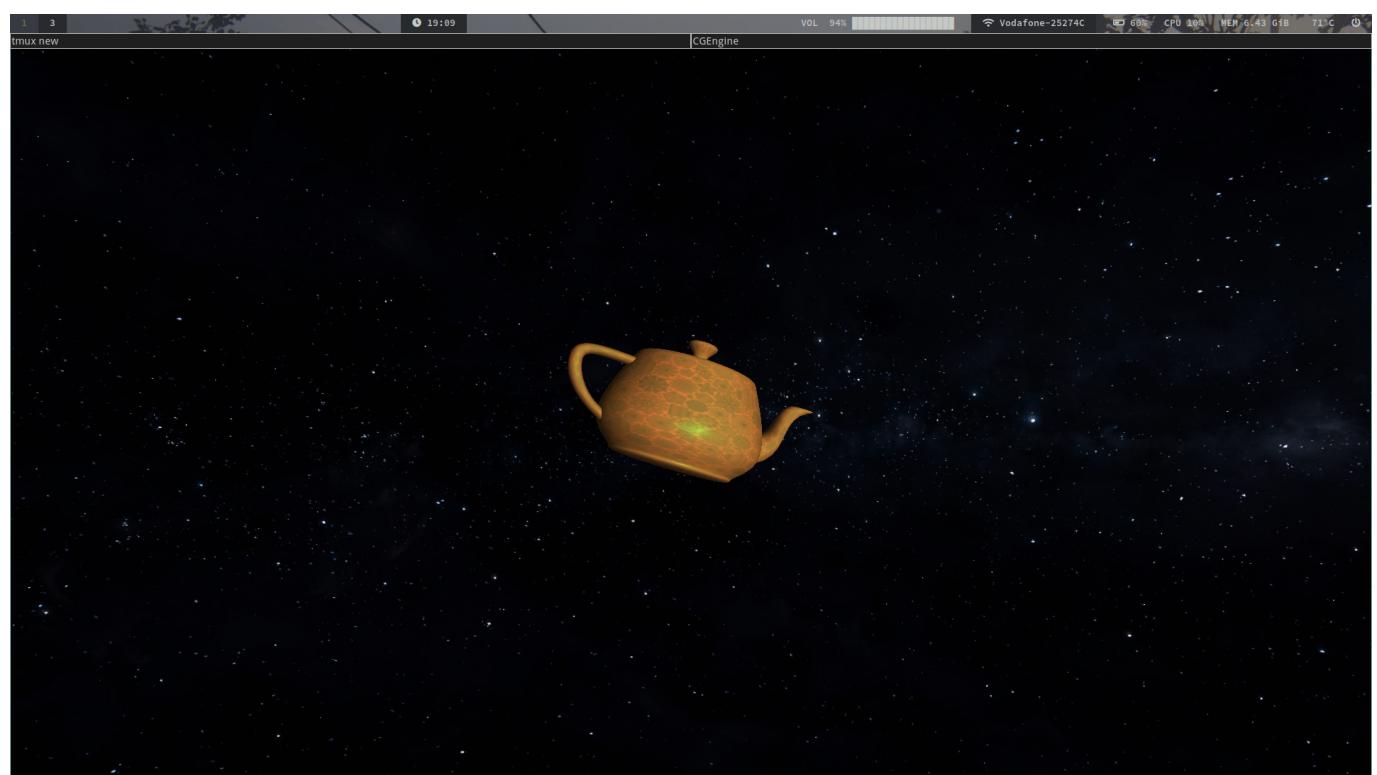


Figura 4.12: Cometa Haley

4.2 Cenas Teste

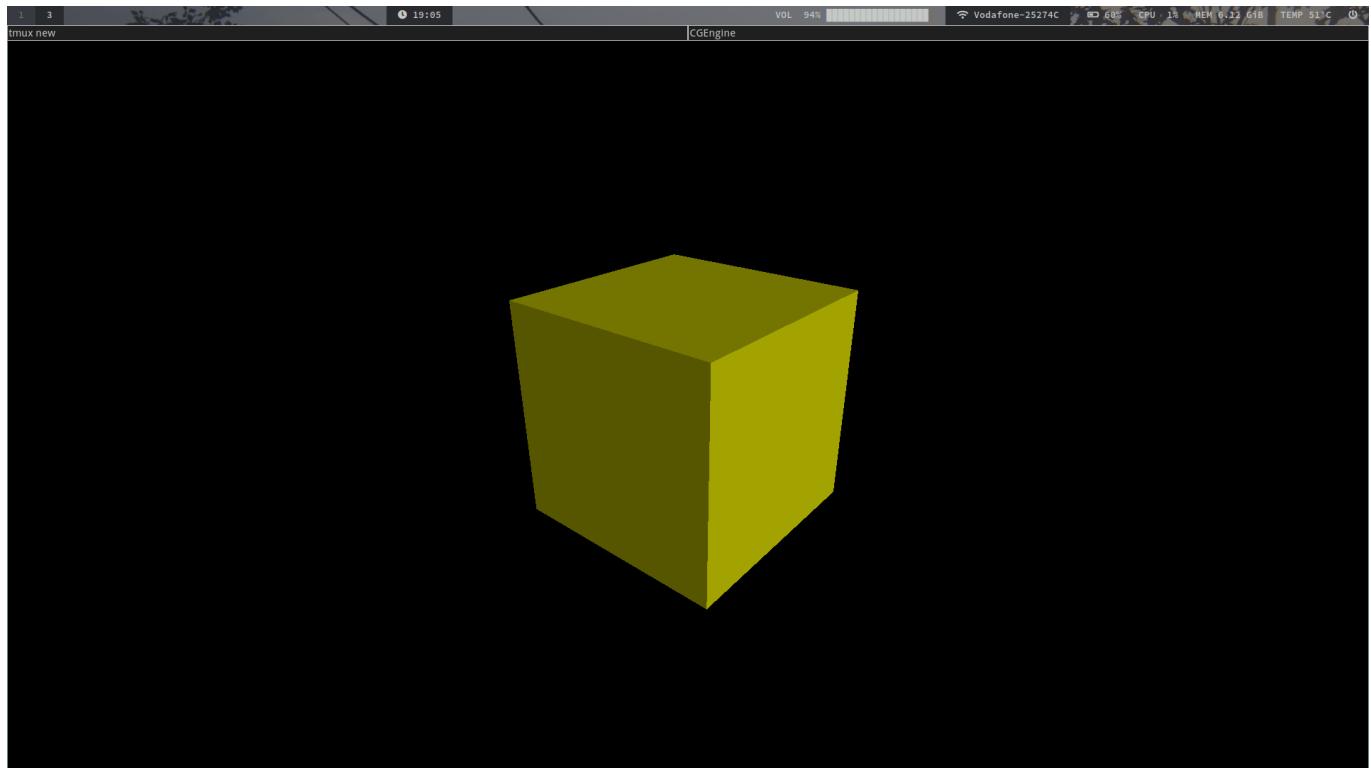


Figura 4.13: Teste 1

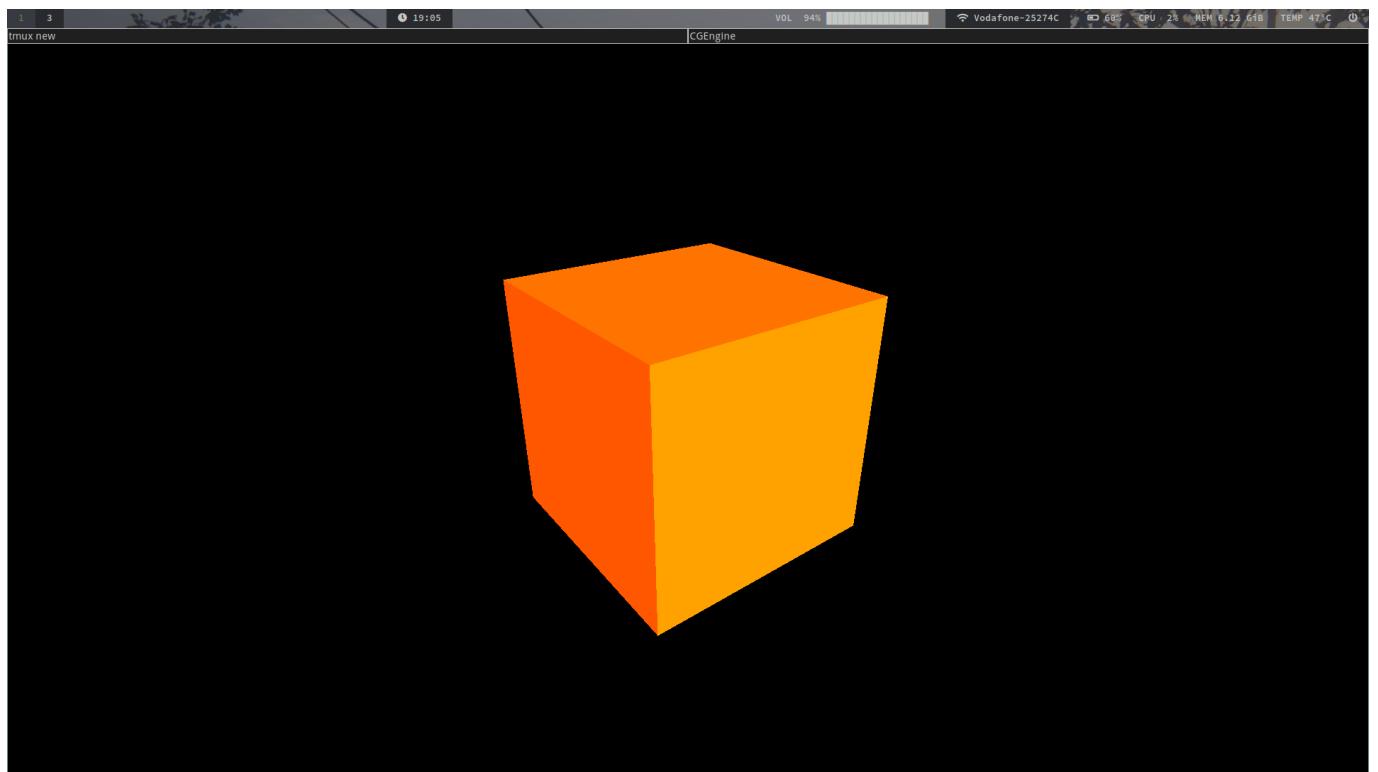


Figura 4.14: Teste 2

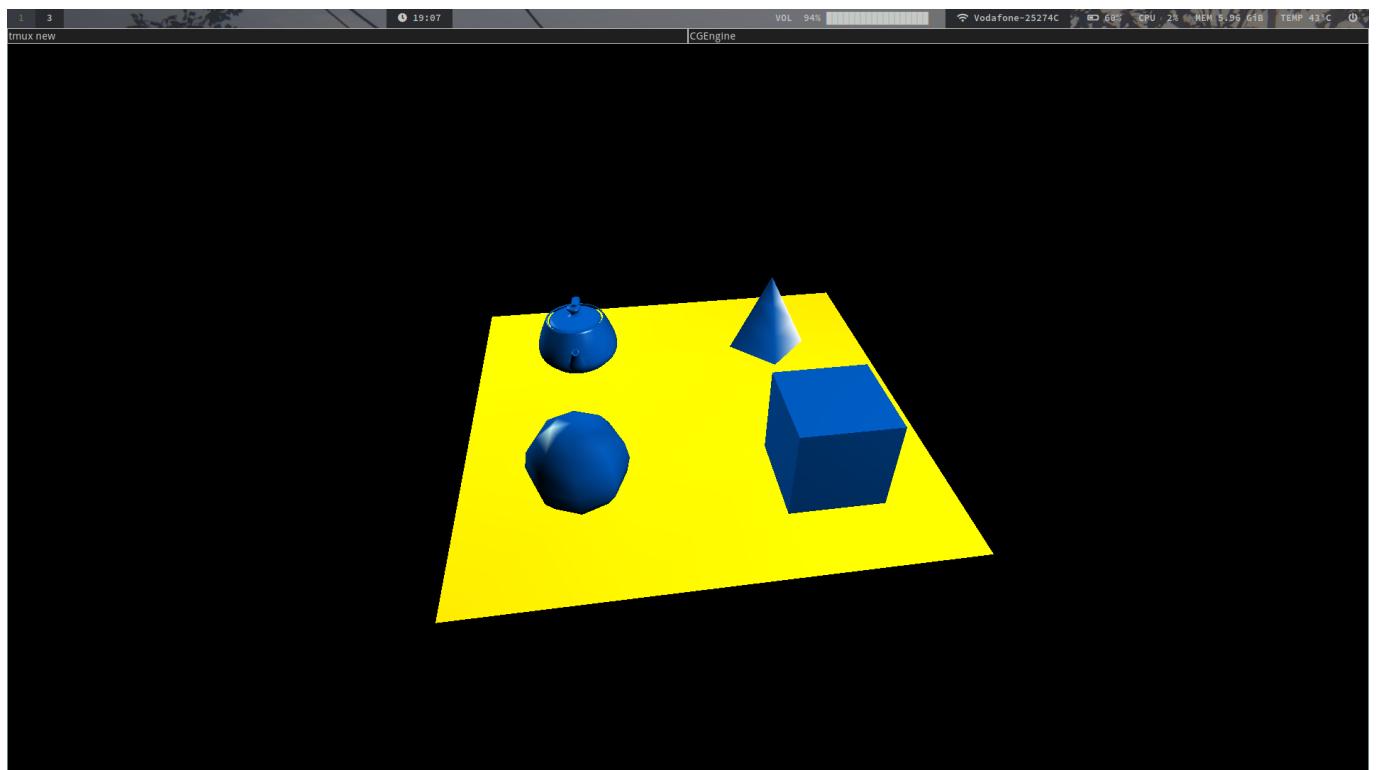


Figura 4.15: Teste 3

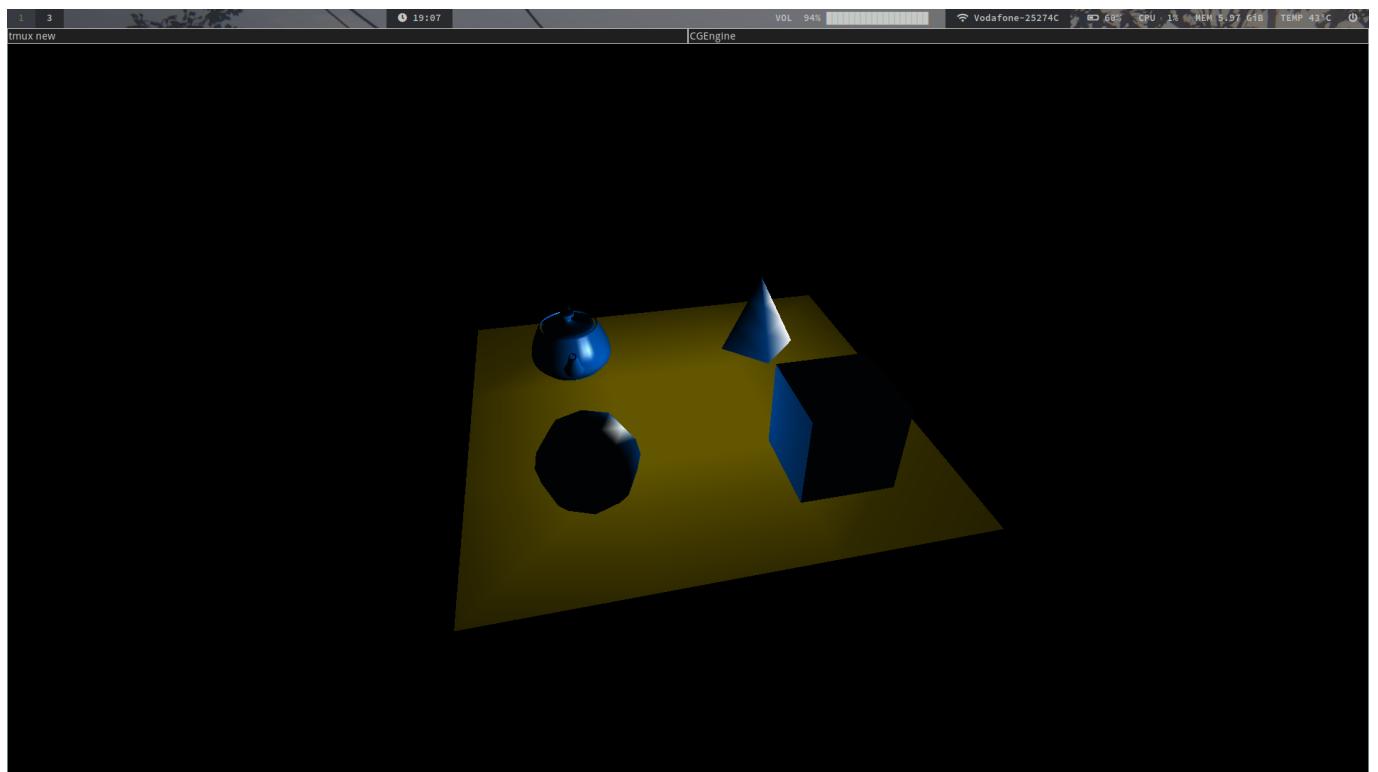


Figura 4.16: Teste 4

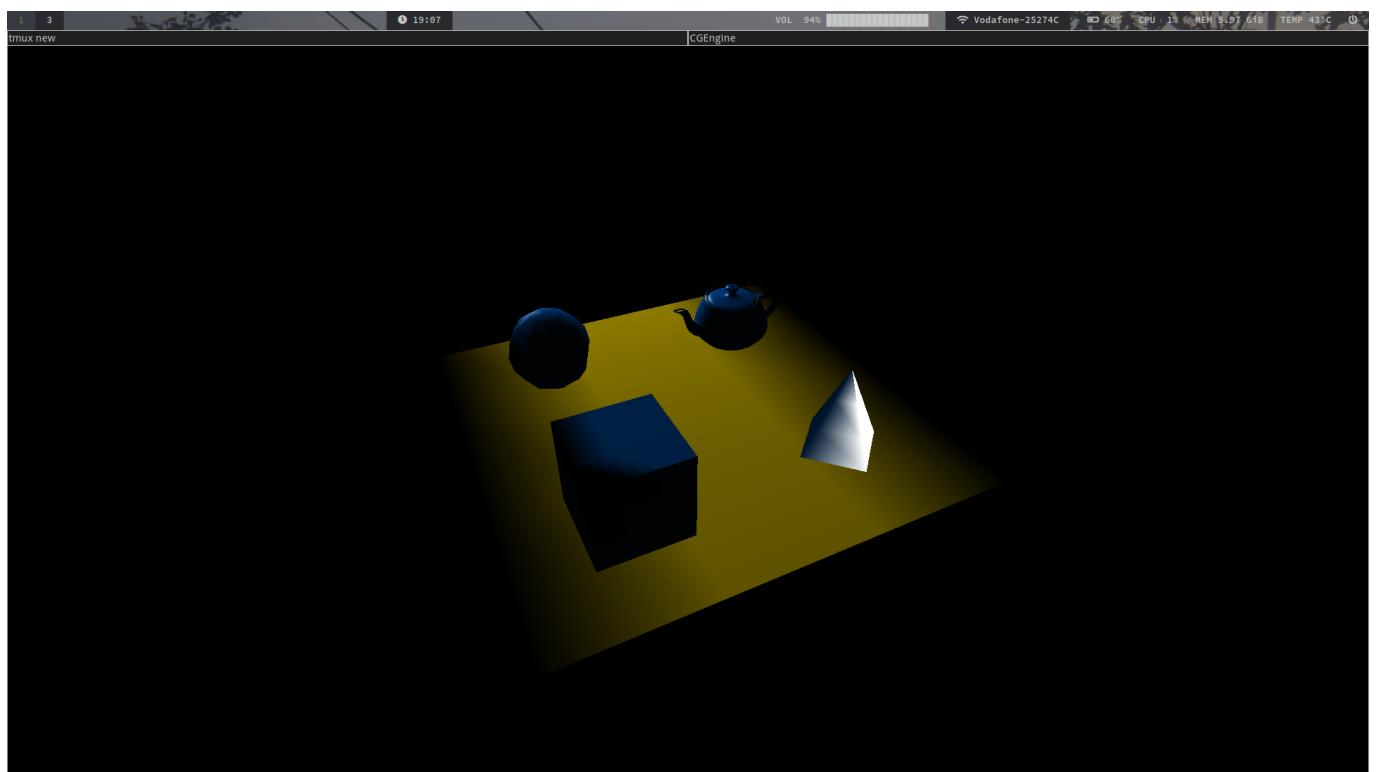


Figura 4.17: Teste 5

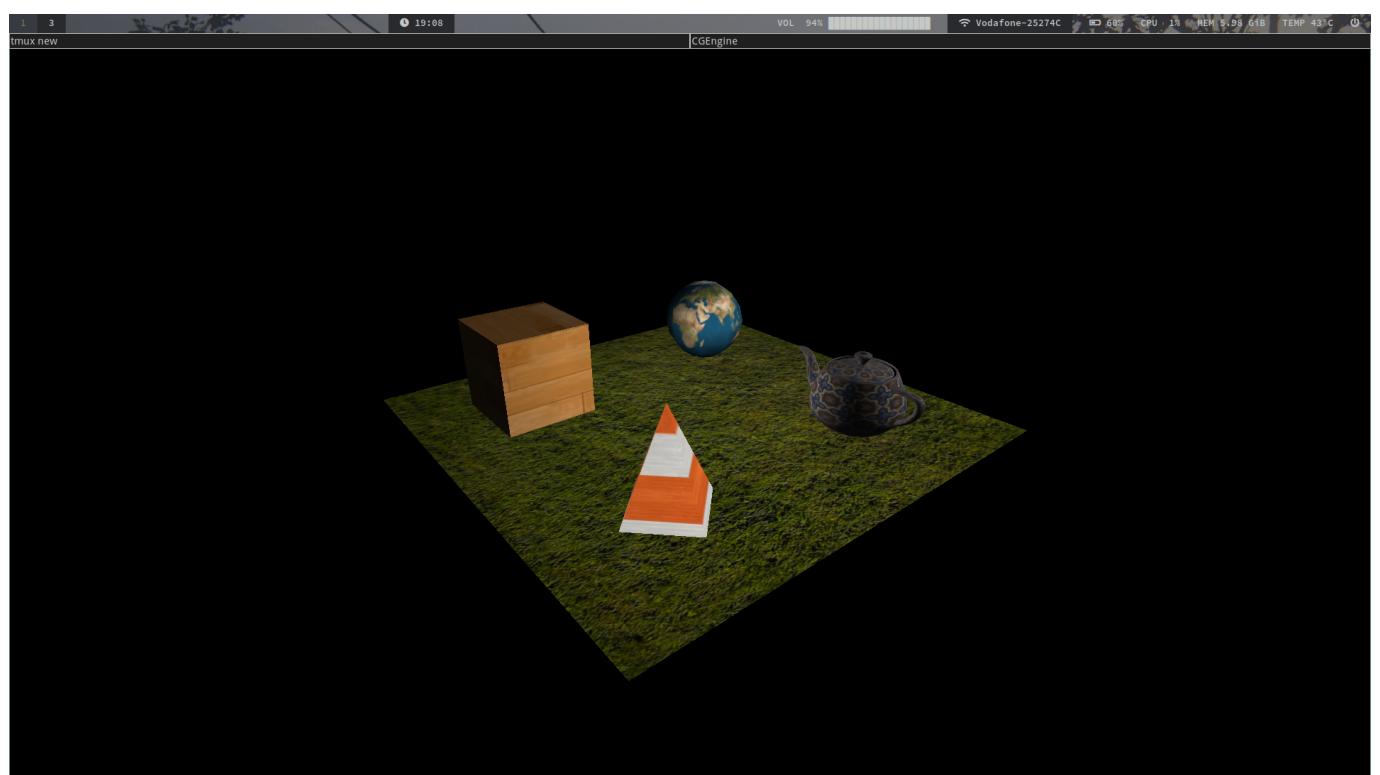


Figura 4.18: Teste 6

Capítulo 5

Conclusão

Com a realização desta última fase do nosso trabalho, acreditamos que fomos capazes de apresentar uma solução interessante e bem conseguida em função ao que foi proposto no enunciado. Consolidamos de forma extensa o nosso conhecimento de elementos associados à área de Computação Gráfica e na utilização de C++ e OpenGL.

Implementamos com sucesso ainda formas de otimizar o programa, tornando-o mais eficiente. Apesar disso, o nosso projeto teria beneficiado com a adição de uma espécie de menu para facilitar a utilização do programa.