

Universidade do Minho
Escola de Engenharia

Relatório da 4^a Fase do Projeto de Computação Gráfica

Engenharia Informática
Universidade do Minho 2020/2021



Alexandre Gomes (a89549)



Carolina Oliveira (a89523)



Manuel Barros (a89560)



Maria Ramos (a89541)

Índice

1. Introdução	2
2. Generator	3 a 8
2.1. Plano	3
2.2. Esfera	3 e 4
2.3. Cone	4 e 5
2.4. Caixa	6 e 7
2.5. Bézier Patch	7 e 8
3. Engine	8 a 9
3.1 Texturas	8
3.2 Materiais	9
3.3 Luzes	9
4. Sistema Solar	10 e 11
5. Funcionalidades adicionais	12 a 15
5.1 Terreno	12
5.2 Torus	12 e 13
5.3 Cilindro	13 e 14
5.4 Caixa invertida	14 e 15
6. Cenas	15 a 18
6.1 Bolas de Desporto	15
6.2 Lanterna	16 e 17
6.3 Noite estrelada	17 e 18
6.4 Luzes Coloridas	18
7. Controladores	19
8. Conclusão	20

1. Introdução

O trabalho a seguir apresentado foi realizado no âmbito da UC de Computação Gráfica do 2º semestre do 3º ano do curso de Engenharia Informática.

Nesta quarta fase do projeto foi nos pedido para gerar as normais e as coordenadas de textura para cada vértice gerado. No final desta fase, a *Engine* também passa a suportar as funcionalidades relacionadas com a textura e iluminação.

2. Generator

Nesta fase foi necessário adicionar normais e coordenadas de textura a todos os modelos previamente criados.

2.1. Plano

Visto que todos os planos criados são paralelos ao plano xz então as normais de todos os pontos vão ser $(0,1,0)$, para as coordenadas de textura simplesmente associamos a cada ponto um canto da textura.



2.2. Esfera

Calcular as normais da esfera é bastante simples, sendo que a normal de cada ponto é o vetor normalizado que vai do centro da esfera ao ponto, o valor normalizado deste vetor coincide com o valor normalizado do ponto.

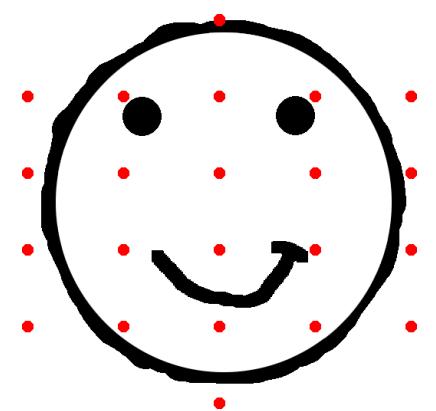
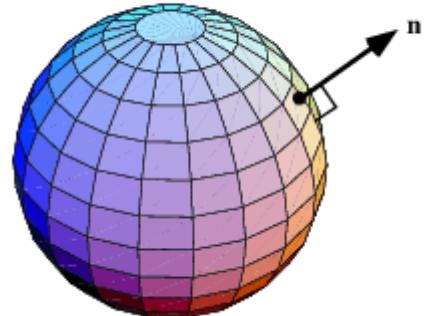
$$x = \cos(stackAng) * \cos(sliceAng)$$

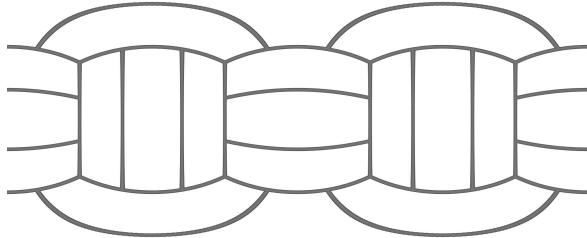
$$y = \sin(stackAng)$$

$$z = \cos(stackAng) * \sin(sliceAng)$$

Sendo $stackAng$ e $sliceAng$ o ângulo da stack e slice da normal a representar.

Para calcular as coordenadas de textura primeiramente decidimos assumir que o ponto do topo da esfera teria a coordenada de textura $(0.5,1)$ e o ponto da base da esfera teria a coordenada de textura $(0.5,0)$, os restantes pontos são calculados a partir da slice e stack do ponto em questão. Ao lado vemos um exemplo dos pontos de textura de uma esfera com 5 stacks e 4 slices.





2.3. Cone

Para todos os pontos na base do cone todas as normais são viradas para baixo (0,-1,0) e para o ponto no topo do cone a normal é virada para cima (0,1,0) para os restantes pontos o cálculo das normais é mais complexo, primeiramente precisamos do vetor horizontal do ponto em questão este é simples de obter visto que usamos este valor para o cálculo das coordenadas do ponto, sendo sliceAng o ângulo da slice em que o ponto está inserido temos

$$xh = \sin(sliceAng)$$

$$yh = 0$$

$$zh = \cos(sliceAng)$$

De seguida é necessário o vetor vertical do ponto sendo este sempre:

$$xv = 0$$

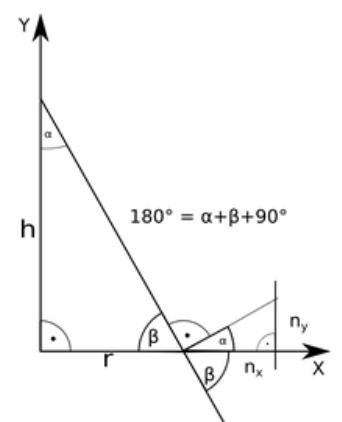
$$yv = 1$$

$$zv = 0$$

, e o valor da normal da hipotenusa da slice onde o ponto está inserido (n_x, n_y), este valor é calculado a partir da inclinação da superfície do cone

$$\alpha = \text{atan}(r/h)$$

$$nx = \cos(\alpha)$$



$$ny = \sin(a)$$

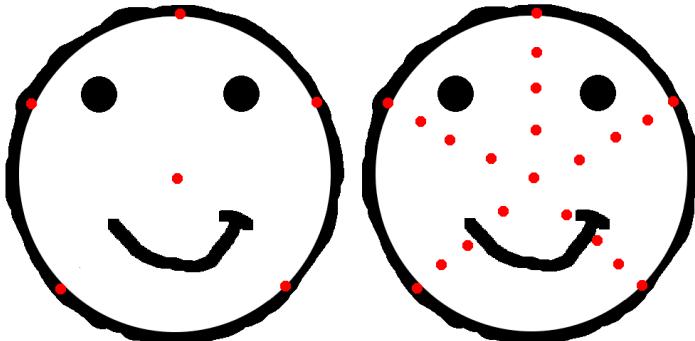
Agora que temos todos os valores necessários para calcular a normal do ponto é necessário juntar o vetor horizontal com o vetor vertical tendo em conta o valor da normal da hipotenusa.

$$x = nx * xv + nx * xh = \cos(a) * \sin(sliceAng)$$

$$y = ny * yv + ny * yh = \sin(a)$$

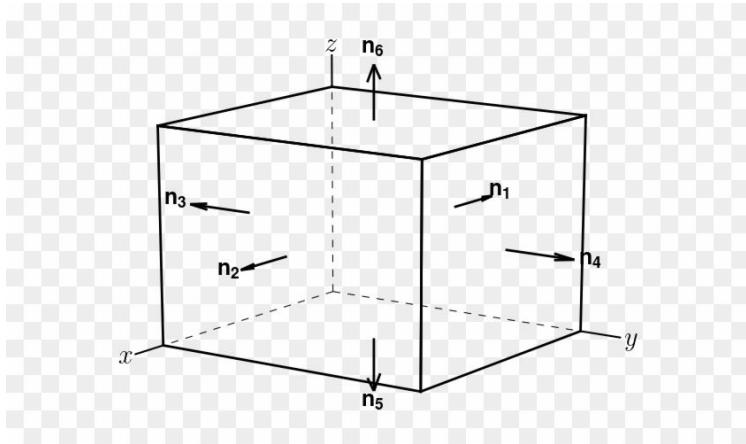
$$z = nx * zv + nx * zh = \cos(a) * \cos(sliceAng)$$

Para calcular as coordenadas de textura do cone admitimos a imagem da textura será constituída por dois círculos sendo que o primeiro representa a base do cone e o segundo representa a área lateral. Sendo assim começamos por atribuir ao ponto central da base e ao ponto do topo do cone as coordenadas de textura (0.25,0.5) e (0.75,0.5) respectivamente, para os restantes pontos tiramos proveito do ângulo da slice e raio da stack em que estavam inseridos para calcular as suas coordenadas de textura. Em baixo vemos um exemplo dos pontos de textura de um cone com 4 stacks e 5 slices.

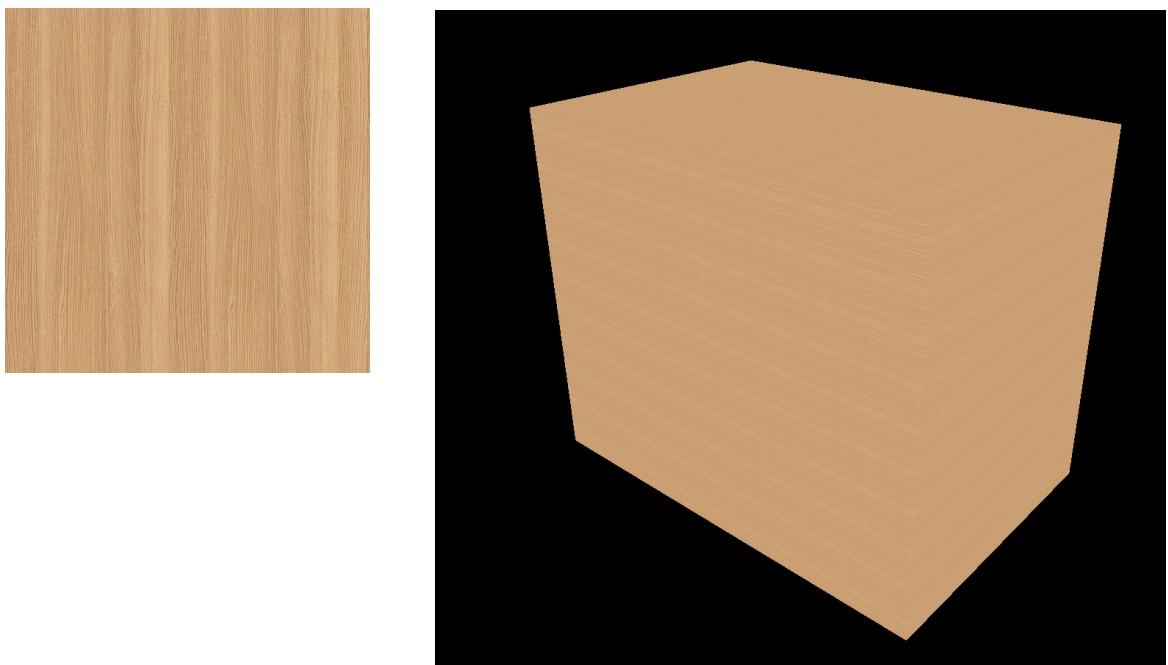


2.4. Caixa

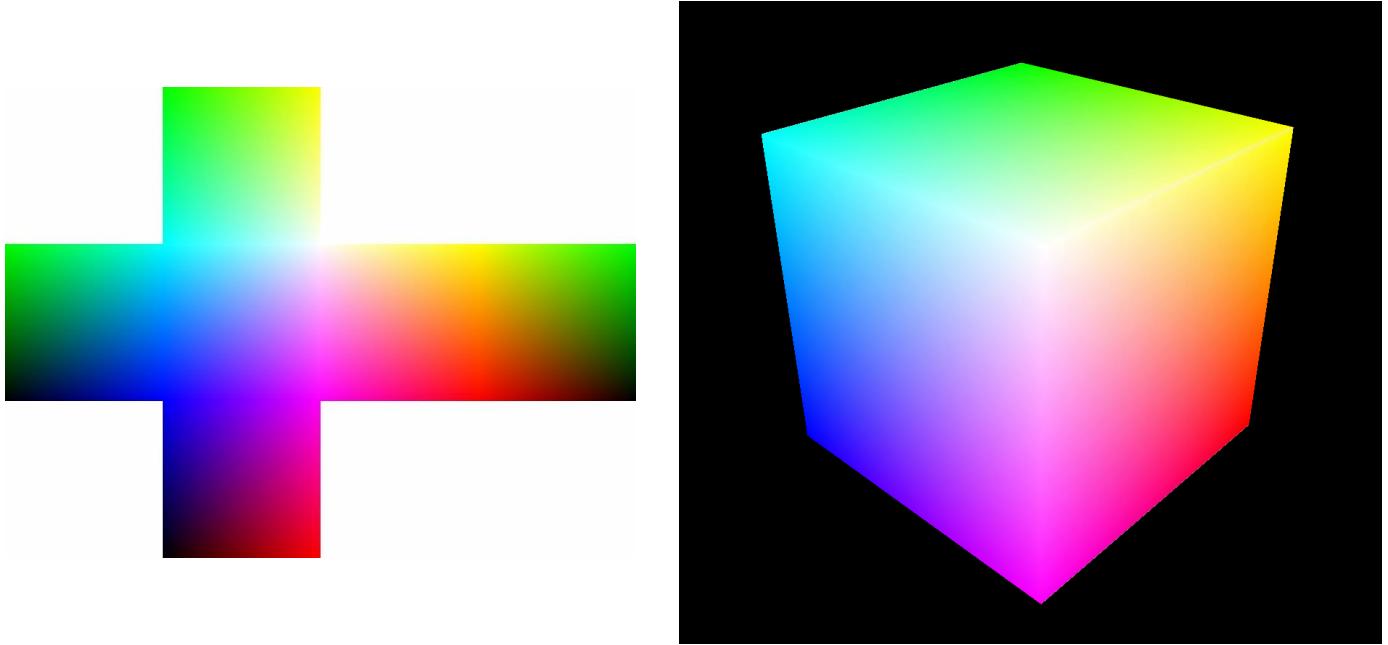
As normais dos pontos da caixa são simples de calcular visto que sabemos qual é o plano do ponto que estamos a analisar.



Para as coordenadas de textura foram feitos dois tipos de processamento, no primeiro repetimos a textura para cada divisão da caixa.



No segundo tipo de processamento, assumimos que cada face terá a sua própria imagem, a disposição destas imagens na textura é feita como se fosse um mapa de um cubo. Cada uma das imagens terá $1/4$ de comprimento e $1/3$ de textura, apenas temos de saber qual é a face que estamos a desenhar para saber onde se encontra a correspondente imagem na textura.



2.4. Bézier Patch

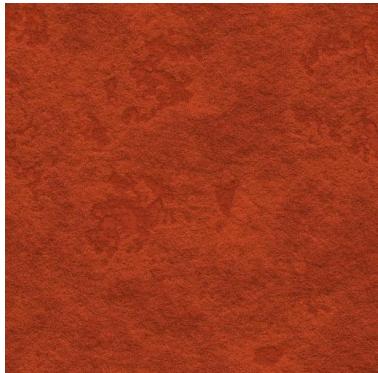
Para calcular as normais dos pontos de um bézier patch é necessário calcular os dois vetores tangentes ao ponto em questão.

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

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

De seguida é necessário calcular o produto vetorial destes vetores e normalizá-los, obtendo assim a normal do ponto.

Para calcular as coordenadas de textura de um patch simplesmente associamos a cada patch a textura toda de forma uniforme.



3. Engine

3.1. Texturas

A primeira funcionalidade a implementar na engine foi permitir que o utilizador pudesse associar texturas ao seu respectivo modelo no ficheiro xml.

```
<models>
|   <model file="../models/sphereP.3d" texture="../textures/volleyball.jpg"/>
</models>
```

Foi criada uma nova classe Texture que trata de inicializar a textura e de fazer bind sempre que o objeto for desenhado, todas as texturas são criadas com mipmap sendo estes os seus parâmetros.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
```

3.2. Materiais

De seguida adicionamos a capacidade de definir materiais para cada objeto, sendo possível definir a componente difusa, especular, ambiente e emissiva.

```
<models>
  <model file = ".../sphere.3d" diffR="0" diffG="0.7" diffB="0" />
  <model file = ".../cone.3d" specR="0" specG="0.7" specB="0"/>
  <model file = ".../box.3d" ambR="0.4" ambG="0.4" ambB="0.4"/>
  <model file = ".../plane.3d" emissR="0.2" emissG="0.2" emissB="0.2"/>
  <model file = ".../torus.3d" diffR="0" diffG="0.7" diffB="0" ambR="0.1" ambG="0.1" ambB="0.1"/>
</models>
```

Foi criada uma nova classe Material que trata de armazenar os valores introduzidos pelo utilizador para mais tarde serem aplicados à figura sempre que esta é desenhada. É usada a função glPushAttrib() e glPopAttrib() em conjunto com a máscara GL_LIGHTING_BIT para impedir que os materiais definidos para uma figura sejam aplicados a próxima figura a ser desenhada.

3.3. Luzes

Por fim implementamos um sistema de luzes, no início da cena xml é possível que o utilizador defina um conjunto de luzes estáticas, sendo estas de três tipos, para a luz do tipo POINT é apenas obrigatório definir o ponto de onde esta luz é emitida, para a luz do tipo DIRECTIONAL é apenas obrigatório definir a direção da luz emitida e por fim para a luz do tipo SPOT é necessário não só o ponto de onde a luz é emitida como também a sua direção, ângulo e expoente (64 por defeito), se o utilizador definir apenas estes parâmetros então a luz definida terá sempre uma cor pré definida (branca) no entanto o utilizador pode alterar esta cor especificando a sua componente difusa, especular e ambiente.

```
<lights>
  <light type="DIRECTIONAL" dirX="1" dirY="0" dirZ="1"/>
  <light type="POINT" posX="3" posY="3" posZ="1" />
  <light type="SPOT" posX="5" posY="5" posZ="5" dirX="-1" dirY="-1" dirZ="-1" cutoff="30" exponent="124"/>
  <light type="DIRECTIONAL" dirX="1" dirY="0" dirZ="0" diffR="1" diffG="0" diffB="0" />
</lights>
```

Foi criada uma classe Light para guardar os valores das luzes e ativá-las no início de cada frame.

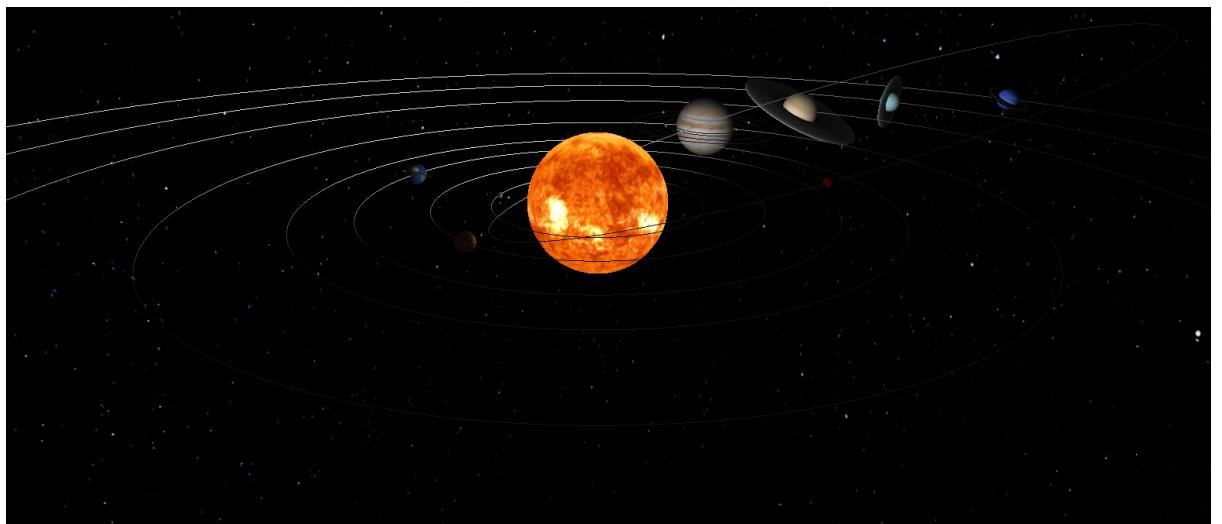
4. Sistema Solar

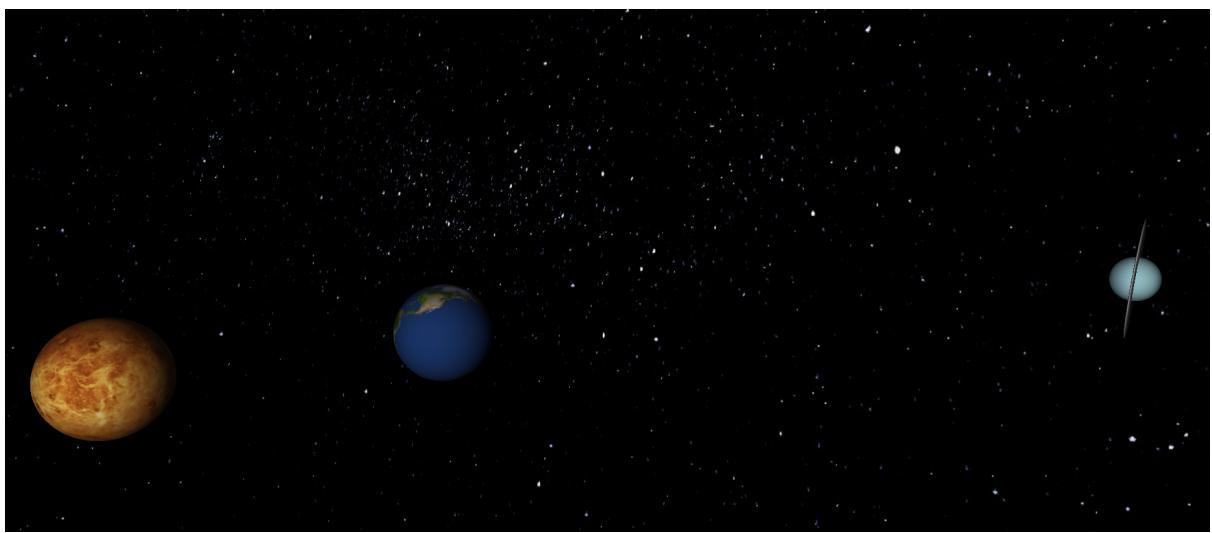
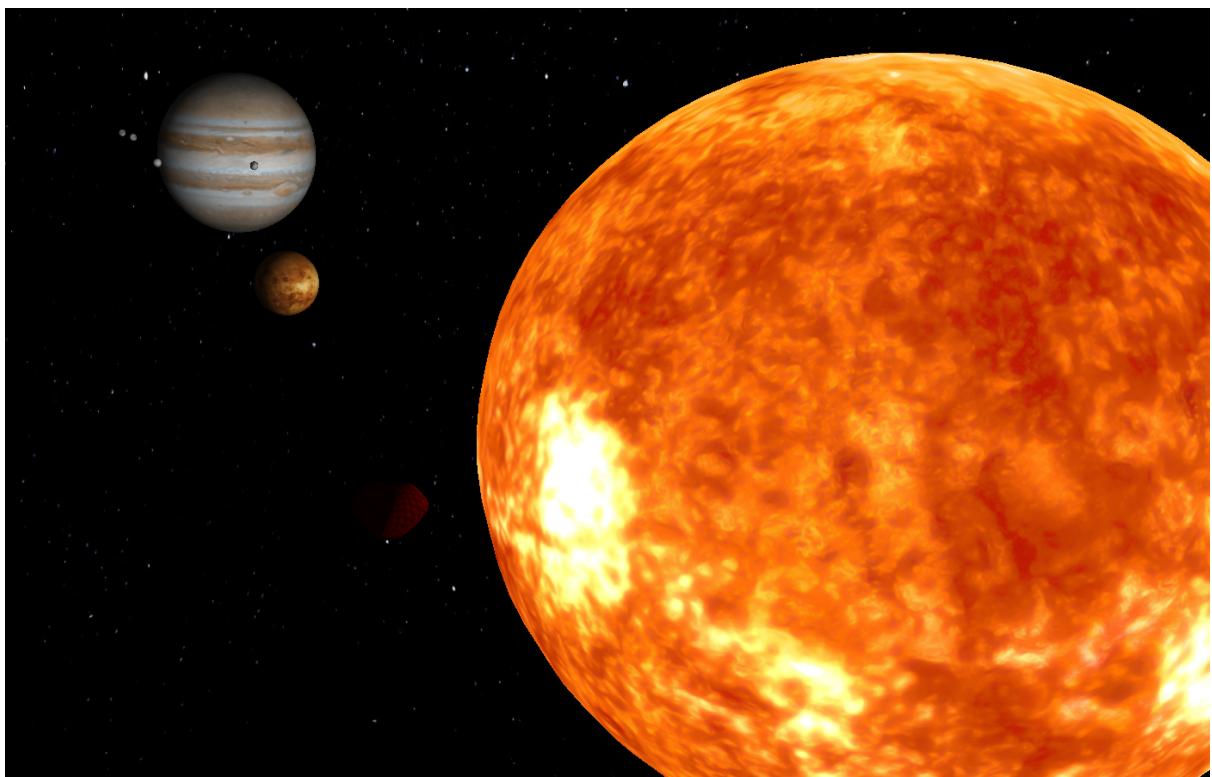
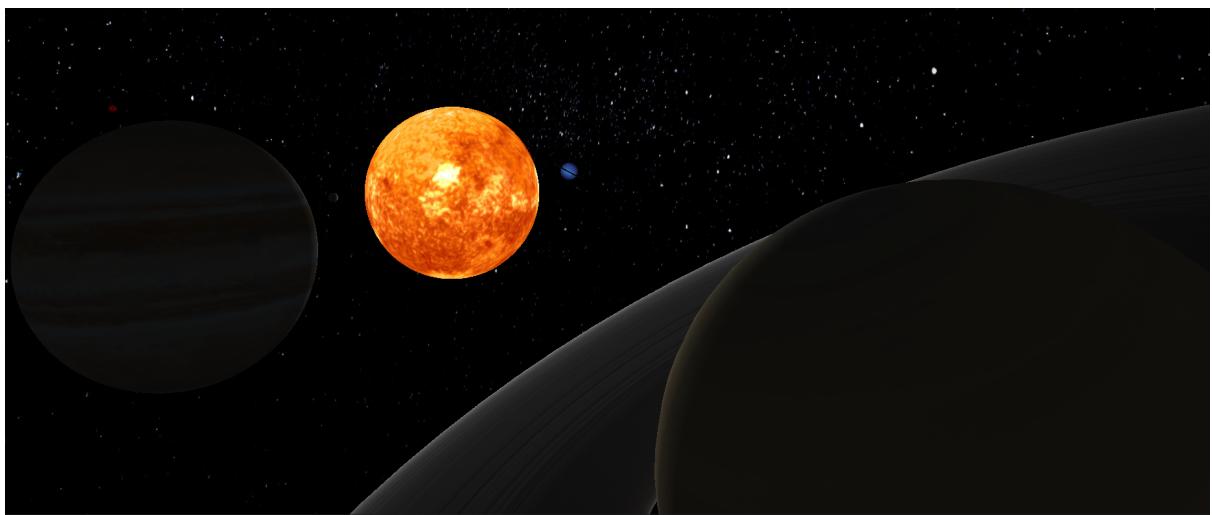
Previamente os anéis dos planetas foram criados a partir de uma esfera com apenas uma stack, visto que agora temos a possibilidade de criar novos modelos a partir da primitiva torus (ver ponto 5.2), todos os anéis foram substituídos por este novo modelo sendo que é feito um scale do modelo no eixo dos y com fator 0.05 para se parecer com um anel, (não é feito com fator 0 apenas por razões estéticas).

Foi adicionada uma caixa invertida (ver ponto 5.4) para servir como espaço, esta tem a componente emissiva no máximo visto que as estrelas emitem a sua luz e não são influenciadas pela luz do sol.

Foram adicionadas texturas a todos os modelos, o sol tem uma componente emissiva máxima visto que este simula uma fonte de luz.

Foi adicionada uma luz branca no centro do modelo para simular a luz emitida pelo sol.





5. Funcionalidades Adicionais

5.1. Terreno

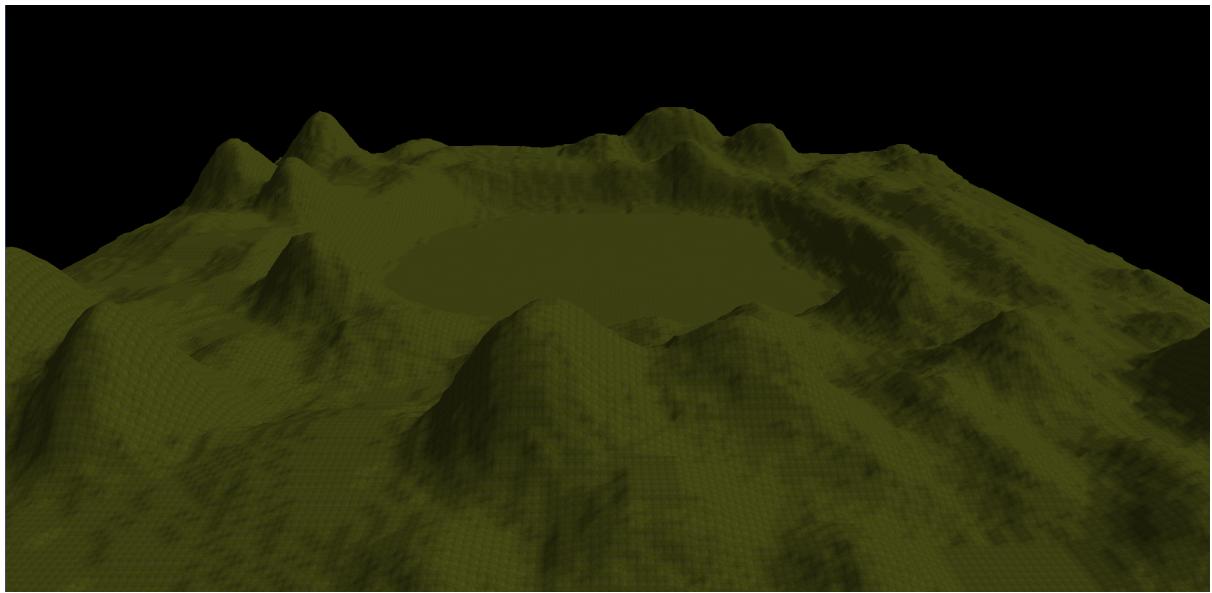
Foram adicionadas normais e pontos de textura na geração de terreno.

Para calcular as normais temos de calcular o vetor normalizado do produto vetorial das derivadas parciais do ponto, logo vamos precisar dos valores dos quatros pontos adjacentes a este, as bordas não são adicionadas ao modelo servindo apenas para calcular as normais dos seus pontos adjacentes.

Para as coordenadas de textura simplesmente associamos a cada canto de cada quadrado no modelo um canto da textura.

Também foi acrescentada a possibilidade de atenuar o valor das alturas passando como parâmetro no gerador.

```
build ./generator terrain ../terrain.jpg 0.4 terrain.3d
build
```



5.2. Torus

Agora o generator permite a criação de um novo tipo de modelo a partir da primitiva de torus.

Para calcular os pontos desta primitiva é necessário o raio do corpo do torus (r), o raio total do torus (R), o número de anéis(rings) e o número de divisões(sides) de cada anel.

```
~/build ./generator torus 3 6 64 64 torus.3d
```

Para calcularmos o ângulo de anel e ângulo ponto ao longo do anel.

$$\text{ringAng} = (\text{ring} * 2\pi/\text{rings})$$

$$\text{sideAng} = (\text{side} * 2\pi/\text{sides})$$

De seguida usamos as seguintes fórmulas para calcular as coordenadas do ponto.

$$x = (R + r * \cos(\text{sideAng})) * \cos(\text{ringAng})$$

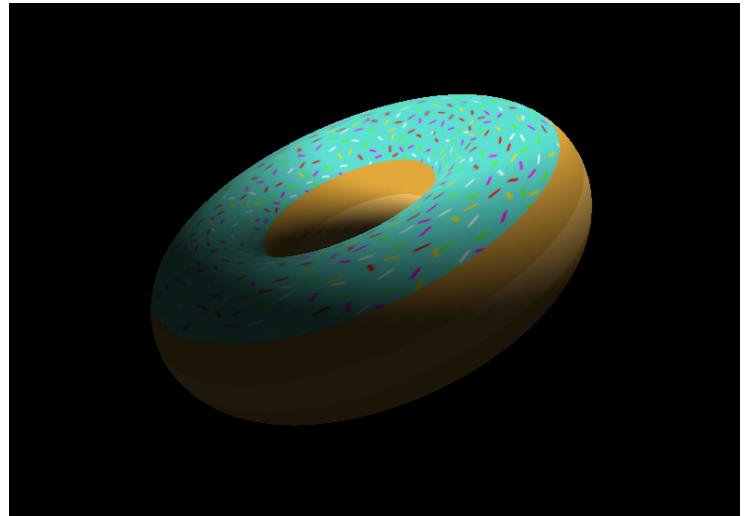
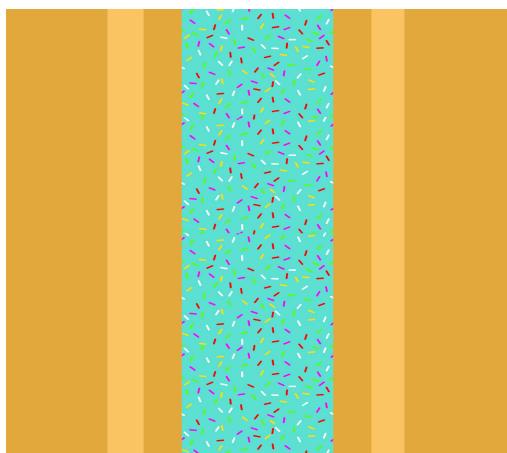
$$y = r * \sin(\text{sideAng})$$

$$z = (R + r * \cos(\text{sideAng})) * \sin(\text{ringAng})$$

Para os normais, simplesmente normalizamos os valores dos pontos.

Por fim para as texturas dividimos a imagem em faixas horizontais e atribuímos a cada anel a sua respectiva faixa.

É de notar que para ser mais fácil de desenhar as texturas o ângulo de ringAng começa com um offset de $\pi/2$.



5.3. Cilindro

Agora o generator permite a criação de um novo tipo de modelo a partir da primitiva cilindro.

Para calcular os pontos desta primitiva é necessário o raio do cilindro(r) a altura(h), o número de stacks e slices.

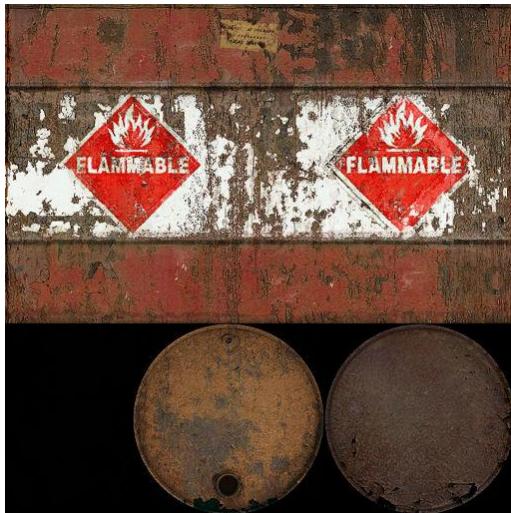
```
~/build ./generator cylinder 5 12 64 64 cylinder.3d
```

Para começar são adicionados todos os pontos da base, as coordenadas x e z destes pontos são calculadas a partir do ângulo da slice em que estão inseridos sendo que a altura é 0, as normais destes pontos são sempre (0,-1,0) e as coordenadas de textura são referentes ao círculo da direita na imagem da textura, estas também são calculadas a partir do ângulo da slice. Este processo é repetido para os pontos da base no final sendo que a altura é igual a h, normais igual a (0,1,0) e as coordenadas de textura são referentes ao círculo da esquerda na imagem da textura.

Para calcular os pontos da área lateral do cilindro é necessário saber qual a altura do ponto em questão esta pode ser calculada sabendo qual a stack em que o ponto está inserido e dividindo o seu valor pela altura total.

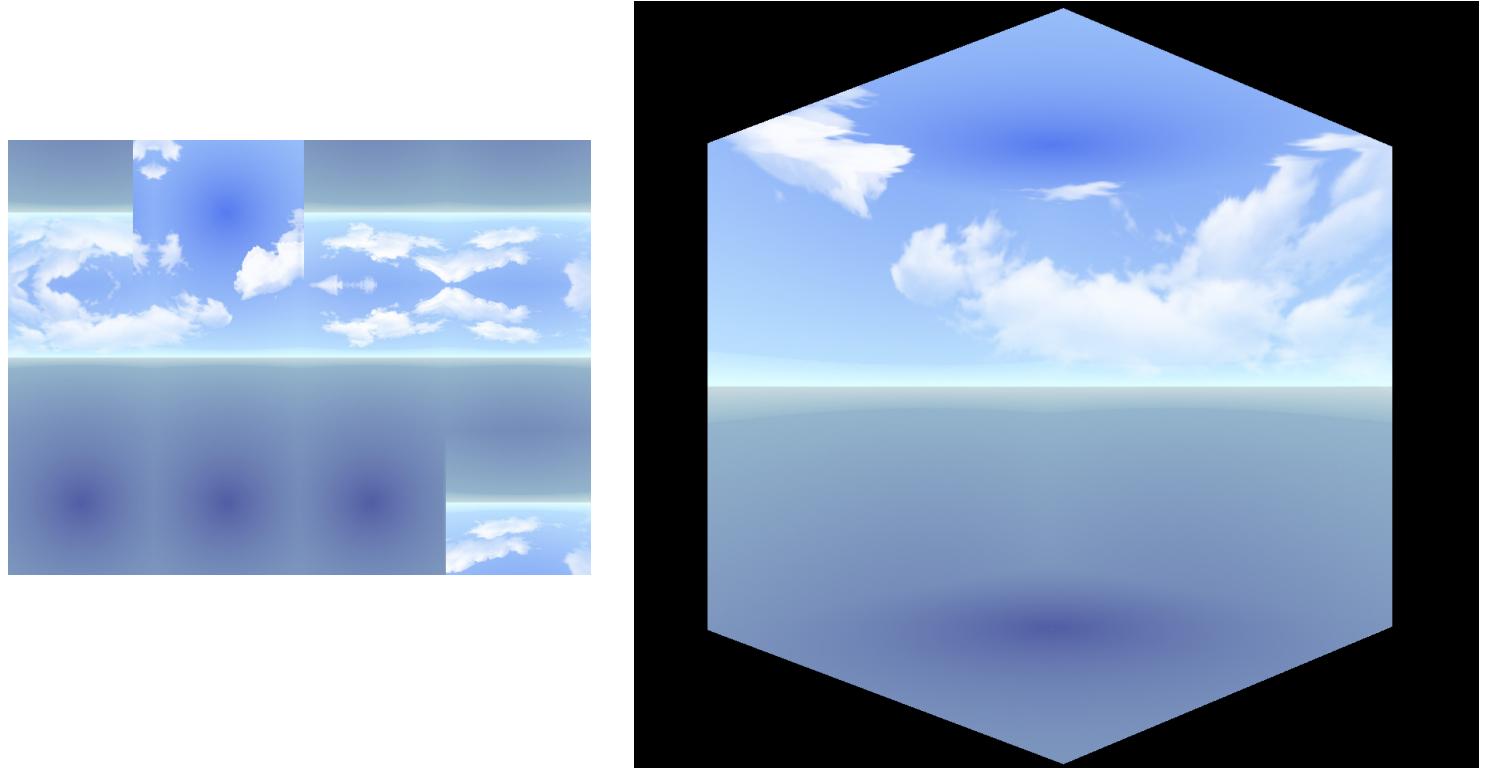
As normais destes pontos são calculadas a partir do ângulo da slice em que estão inseridos.

As coordenadas de textura são referentes ao retângulo no topo da textura.



5.4. Caixa Invertida

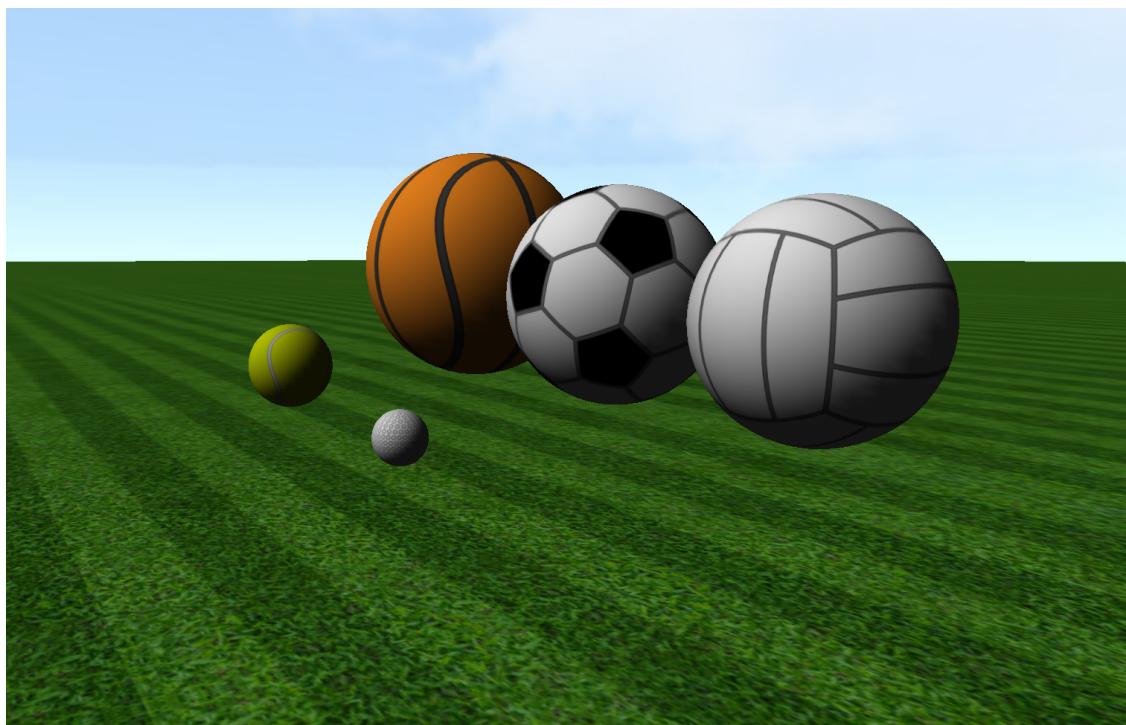
Foi feito um modelo novo a partir do modelo da caixa já estabelecido, para obter o efeito de uma caixa invertida simplesmente alteramos a ordem pela qual os pontos eram adicionados ao ficheiro e invertemos as normais dos pontos.



6. Cenas

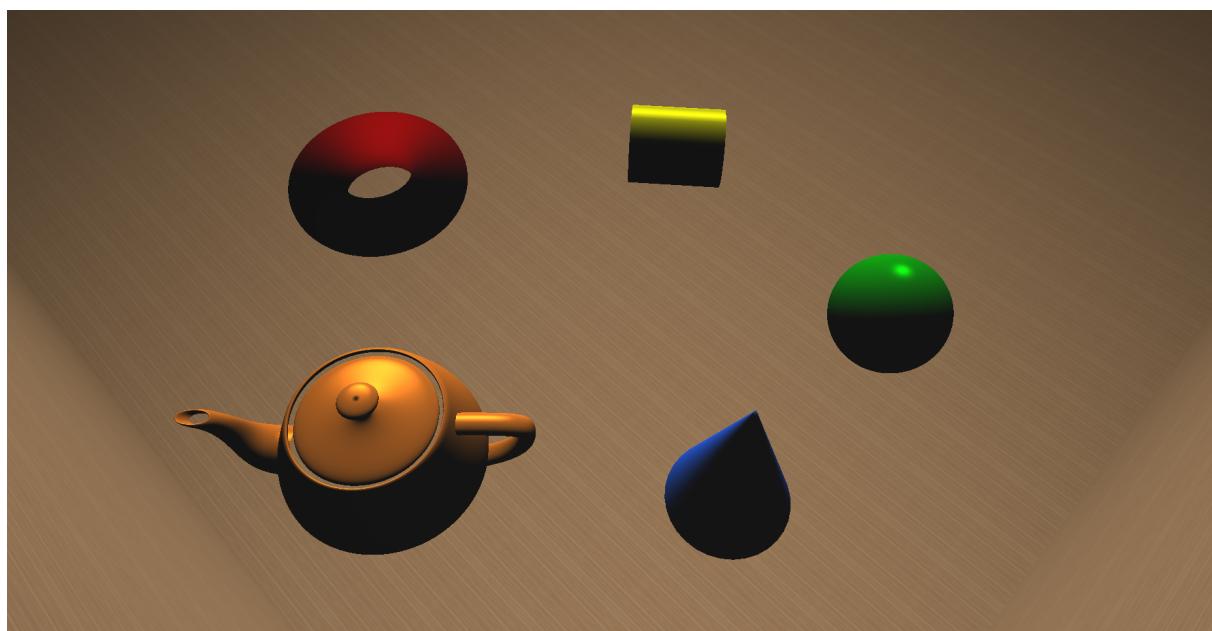
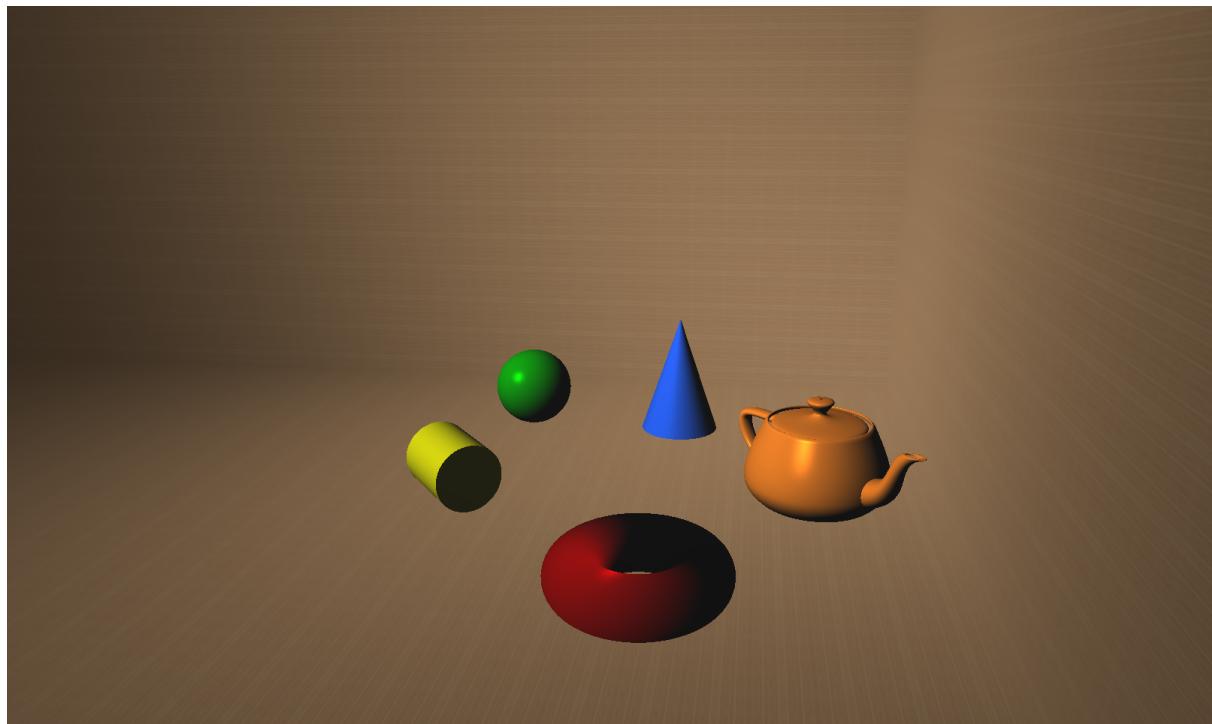
6.1. Bolas de Desporto

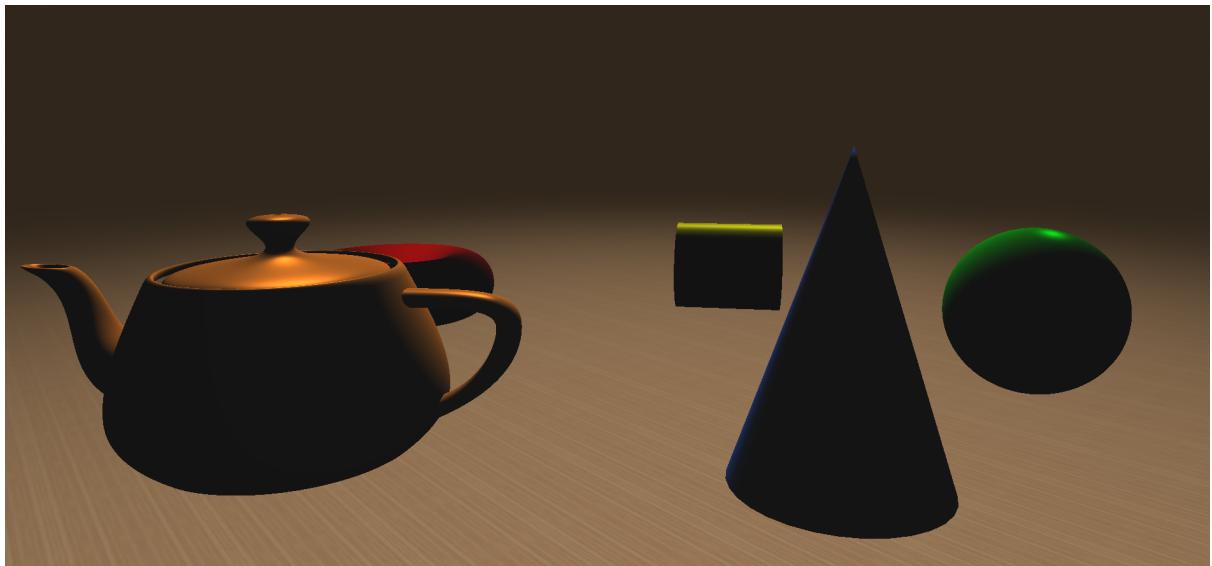
Nesta cena simplesmente adicionamos esferas com vários tamanhos e textura, usamos uma skybox para o céu, uma caixa para o chão e uma luz com direção $(0,1,1)$.



6.2. Lanterna

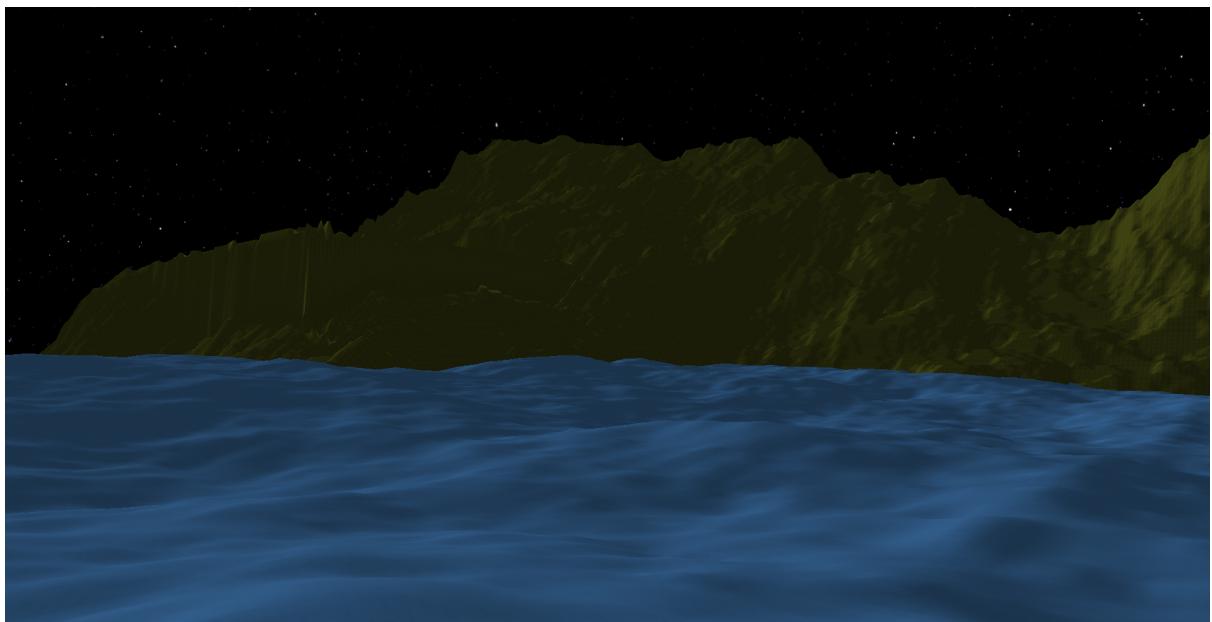
Nesta cena colocamos vários objetos no canto de uma caixa invertida que está a ser iluminada por uma lanterna no canto oposto da caixa.

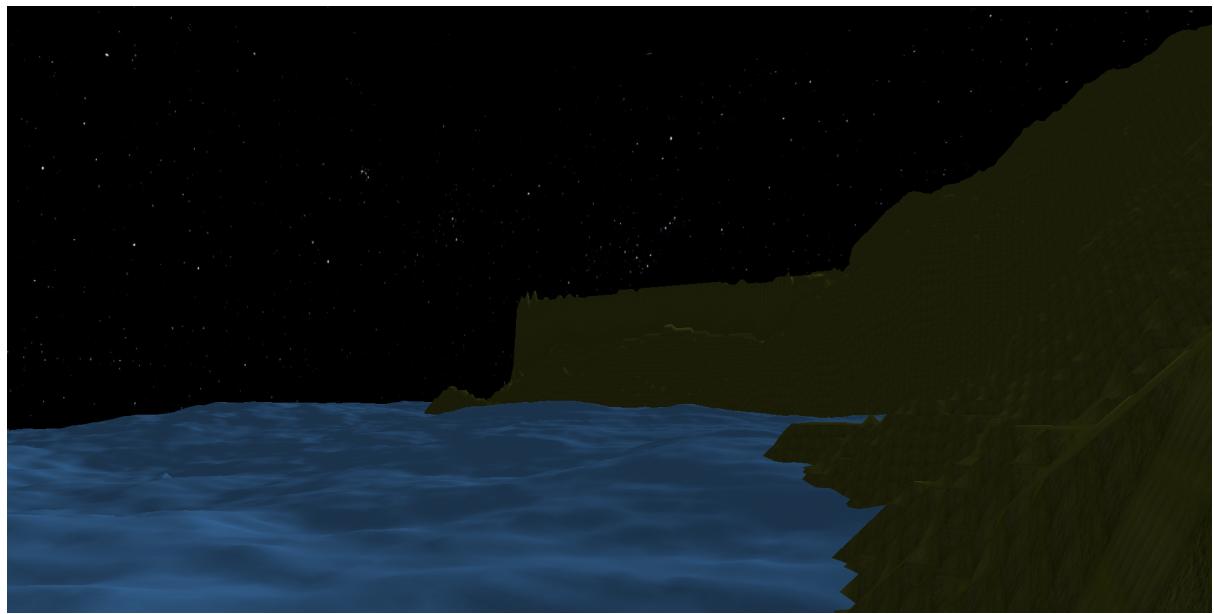




6.3. Noite Estrelada

Nesta cena colocamos dois terrenos sendo que um deles tem um fator de multiplicação das alturas muito alto, para simular as ondas do mar, também foi colocado uma skyBox para simular o céu de uma noite estrelada.





6.4. Luzes Coloridas

Nesta cena colocamos uma luz vermelha a apontar na direção $(1,0,1)$ e uma luz verde a apontar na direção $(-1,0,1)$ de modo a se intercetarem no centro do objeto branco que se encontra na cena, é assim esperado conseguir ver um bocadinho de luz amarela no centro deste objeto.



7. Controladores

Na figura abaixo, apresentamos os controladores que deverão ser utilizados para explorar as cenas.

Key	Effect
mouse	moves camera
c	changes camera between fps and centered
mouse wheel	zooms camera (centered)
w	moves camera foward (fps)
a	moves camera left (fps)
s	moves camera backwards (fps)
d	moves camera right (fps)
q	increases camera speed (fps)
e	decreases camera speed (fps)
t	toggles xyz axis
m	toggles rendering of catmull-rom curves
n	changes the type of rendering of catmull-rom curves (points or lines)
l	changes between GL_LINES and GL_FILL

8. Conclusão

Na primeira fase do projeto, implementamos o *Generator*, capaz de gerar os pontos de várias primitivas, como do plano, caixa, esfera, cone, Torus, entre outros. Também foi implementada a primeira versão da *Engine*, capaz de ler ficheiros XML, que contém a informação dos ficheiros .3d gerados que deve ler, e em seguida desenhar as primitivas.

Na fase seguinte, foram implementadas transformações geométricas. Foi apenas necessário alterar o *Engine*, para suportar a rotação, translação e escala e que fosse capaz de ler ficheiros XML que apresentem a noção de hierarquia.

Em seguida, na fase três, foram implementadas diversas técnicas para enriquecer o tipo de cenas a construir, foi alterado o *generator* para permitir a criação de modelos baseados em *patches* de Bézier e foram implementados novos modos para rotação e translação tirando proveito de curvas de Catmull-Rom.

Nesta última fase, foi implementada a geração de normais e de coordenadas de textura, o que permitiu adicionar vários tipos de iluminação e ainda textura aos modelos, foi ainda adicionada a possibilidade de definir materiais para as figuras, .

Depois da implementação de todas as fases foi possível a criação de demo scenes complexas.

Ao longo das várias fases fomos testando o programa com vários modelos para além de irmos atualizando o modelo do sistema solar para ir incluindo as novas funcionalidades implementadas em cada uma das fases.

Consideramos que o facto deste trabalho estar dividido em quatro fases e cada uma delas ter acompanhado os assuntos que iam sendo dados nas aulas, nos facilitou a implementação do programa do que se tivesse sido tudo feito de uma só vez. Visto que permitiu que a implementação fosse um processo incremental.