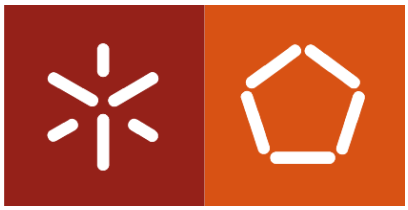

UNIVERSIDADE DO MINHO
Mestrado Integrado em Engenharia Informática
Computação Gráfica

TRABALHO PRÁTICO: 4º FASE

Coordenadas de texturas e vetores normais

Grupo 3:
Bruno Pereira — 72628

Braga, 21 de Maio de 2017

Conteúdo

Lista de Figuras

Introdução

Para esta fase deste projeto é requerido que se implementem texturas, luzes e componentes de cor da luz e da reflexão da luz nos materiais, ou componentes virtuais da reflexão do mundo real.

Pede-se que se calcule vetores normais às superfícies de todas as geometrias criadas, bem como coordenadas de textura no *Generator*, mas com particular interesse no desenvolvimento de um modelo de sistema solar, com transformações geométricas animadas, texturas dos planetas e outros objetos do sistema solar, com a implementação de uma luz dentro do Sol, como a luz que irradia deste objeto celeste.

1 Gerador

O `Generator` tem, como função, recebendo parâmetros como comprimento, largura, raio, etc., gerar ficheiros de texto com a extensão `.3d`, cujo conteúdo é a informação sobre as figuras a criar.

Nesta secção ir-se-á descrever o processo de desenvolvimento das figuras necessárias do sistema solar. As figuras pertinentes a desenhar são a esfera (para os planetas e sol) e um disco (para alguns planetas que os tenham, como por exemplo Saturno) e um bule que servirá de cometa.

1.1 Esfera

1.1.1 Vértices da geometria

Para a construção da esfera teve-se que ter em conta coordenadas esféricas modificadas para o referencial rodado com Y para cima, Z como eixo das abcissas e X como eixo das ordenadas, como demonstra a Equação 1.

$$\begin{cases} x = \cos(\phi) * \sin(\theta) * \rho \\ y = \sin(\phi) * \rho \\ z = \cos(\phi) * \cos(\theta) * \rho \end{cases} \tag{1}$$

Na Equação 1, ρ representa o raio, ϕ o ângulo polar sendo $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, θ representa o ângulo azimutal sendo $\theta \in [0, 2\pi]$.

1.1.2 Vetores normais à superfície da geometria

Para a aplicação de luzes no *Engine* é necessário efetuar o cálculo dos vetores normais. O cálculo pode ser efetuado de diversas formas, tanto computacionais, como o cálculo de derivadas com o cálculo do vetor normal dessas derivadas (devidamente normalizado), podem ser obtidas por interpolação de vértices, e se a geometria já for conhecida, analiticamente. A normalização é necessária, uma vez que, a fórmula para o cálculo da intensidade da luz da lei de *Lambert*, que postula que a intensidade refletida por um material puramente difuso é proporcional ao cosseno do ângulo entre a normal da superfície e a direção da luz de entrada. Ora para computar o cosseno, pode-se usa-se o produto externo, e tendo vetores unitários a computação é mais eficiente. Outra questão é o facto de diferentes vetores em vértices de um mesmo triângulo terem magnitudes diferentes, e o OpenGL computa as normais com a interpolação, criando uma cópia dos vetores normais. Se os vetores estiverem com a mesma magnitude, esse problema não se coloca, uma vez que estão normalizados.

Assim, para o cálculo das normais de uma esfera sabemos as direções de cada ponto, e para serem coordenadas de um vetor unitário basta o raio ser 1. Assim a Equação 2 representa o cálculo de uma normal de um vértice da esfera.

$$\begin{cases} N_x = \cos(\phi) * \sin(\theta) \\ N_y = \sin(\phi) * \rho \\ N_z = \cos(\phi) * \cos(\theta) \end{cases} \tag{2}$$

1.1.3 Coordenadas de texturas para a geometria

Usando uma textura bidimensional, sabendo que as coordenadas da mesma do OpenGL variam entre 0 e 1, pode-se cobrir um geometria descrevendo as coordenadas em x e y desse referencial. Pretende-se construir a esfera de cima para baixo, desde $\frac{\pi}{2}$ a $-\frac{\pi}{2}$ do ângulo ϕ. Se considerarmos uma textura orientada como um retrato, a primeira coordenada de x será 0 e a primeira coordenada de y será 1, e pretende-se que o valor de x aumente da esquerda para direita, e diminua de cima para baixo, e o valor de y diminua de cima para baixo.

1.1.4 Diagrama

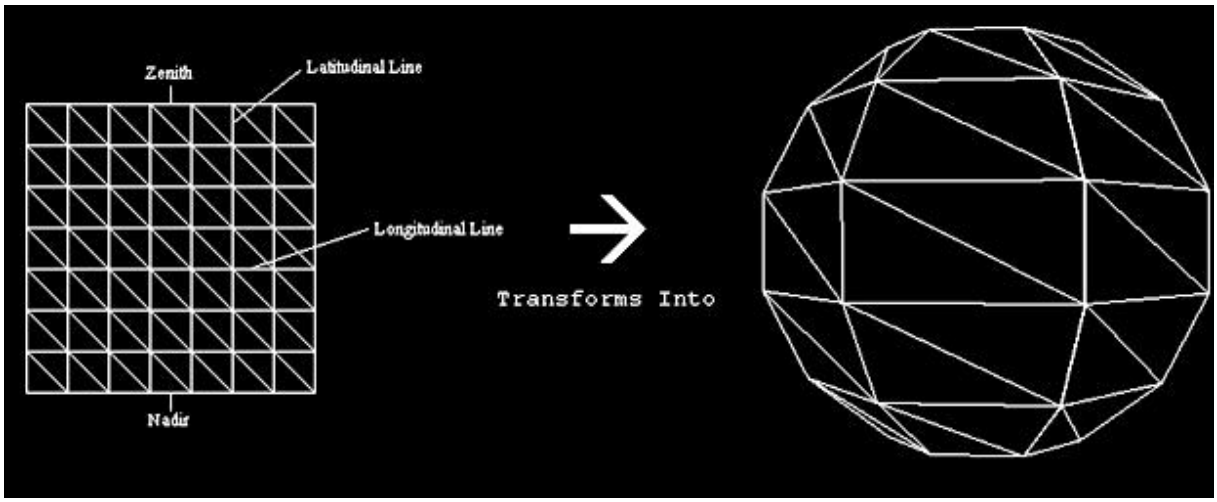

Figura 1: Objetivo do algoritmo de construção de esfera

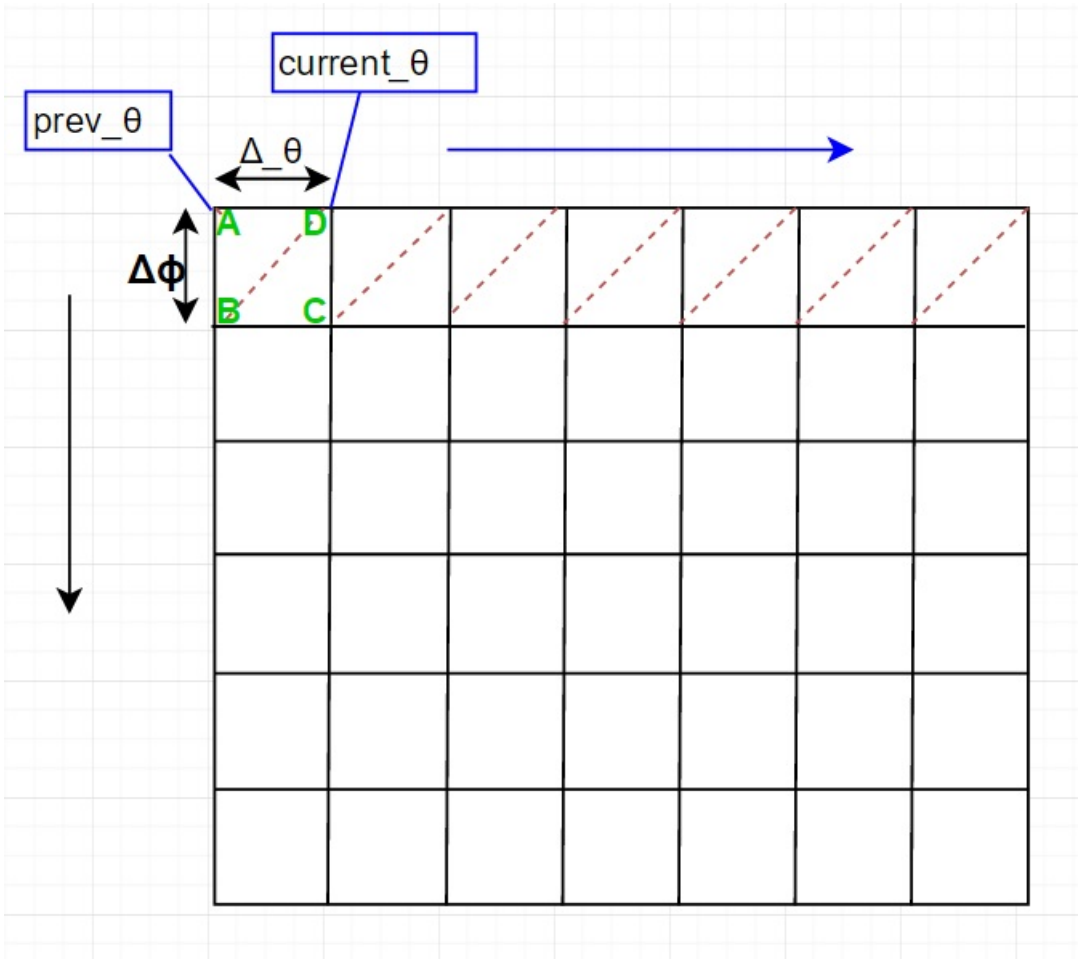

Figura 2: Diagrama de representativo de construção de esfera

No *Figura 2* pode-se ver uma matriz, que representa a esfera nos graus de ϕ e θ para

6 *stacks* e 7 *slices*. Assim como um mapa representativo da Terra, pretende-se mostrar os pontos se a esfera fosse aplanada (ver *Figura 1*).

Em cada quadrícula são calculados 4 pontos iniciais, com base nos cálculos apresentados pelo fórmula anterior. Note-se que, se usou duas variáveis para guardar o ϕ anterior e o ϕ corrente, e θ anterior e θ corrente. Adicionalmente é calculada a diferença de graus entre *slices* e *stacks*, representados por $\Delta\phi$ e $\Delta\theta$, respetivamente. Adicionalmente serão calculados os vetores normais, um por vértice, como descrito na Equação 2.

A intenção é calcular cada quadricula para cada linha e coluna, com auxilio das diferenças dos ângulos e à medida que se avança em cada quadricula, guardar o último grau calculado (ϕ e θ) e calcular nos pontos com o incremento nestes ângulos. Assim desloca-se para a direita na matriz, conforme θ avança de 0 para 2π e para baixo, conforme ϕ avança de $\frac{\pi}{2}$ para $-\frac{\pi}{2}$ (sentido dos ponteiros do relógio). Note-se que a Figura 2 pode representar de igual modo, a elaboração das coordenadas das texturas, tendo em conta, que no algoritmo, cada coluna tem que refletir o valor de x e cada linha y e, como se pretende aplicar a textura de cima para baixo, terá que se subtrair cada valor de linha ou coluna a 1, para as coordenadas decrescerem.

1.2 Disco

Nesta secção descreve os procedimentos usados para desenvolver um disco. A motivação para o desenvolvimento desta figura provém da necessidade de representar os anéis que rodeiam os planetas Saturno e Úrano.

1.2.1 Vértices da geometria

Para criar um modelo do sistema solar é necessário criar discos para aplicar em Saturno e, eventualmente, em Urano. Note-se que, todos os gigantes de gás possuem anéis, no entanto, dado que os anéis de Júpiter e Neptuno são pequenos não se considera a construção do mesmo.

Com efeito, requer-se para este projeto que se criem discos de vários tamanhos para os anéis de Saturno e Urano. Note-se que cada anel tem que ter alguma espessura, uma vez que, num plano, no *OpenGL* não se consegue ver o objeto. Assim cada disco terá duas circunferências, uma interior e outra exterior, com raio interno e externo respetivamente. Assim, as duas circunferências têm os mesmos pontos *xx* e *zz* mas com uma distancia fixa no eixo *yy*.

A fórmula para desenhar uma circunferência está representada na *Equação 3*, podendo acrescentar a componente em y para criar um cilindro com uma altura pequena, mas suficiente para se poder distinguir.

$$\begin{cases} x = \sin(\theta) * r \\ z = \cos(\theta) * r \end{cases} \tag{3}$$

Nesta secção apresentam-se diagramas que explicam o processo de criação de uma disco.

A *Figura 3* representa a forma como a iteração será feita, bem como apresenta de lado a espessura do disco.

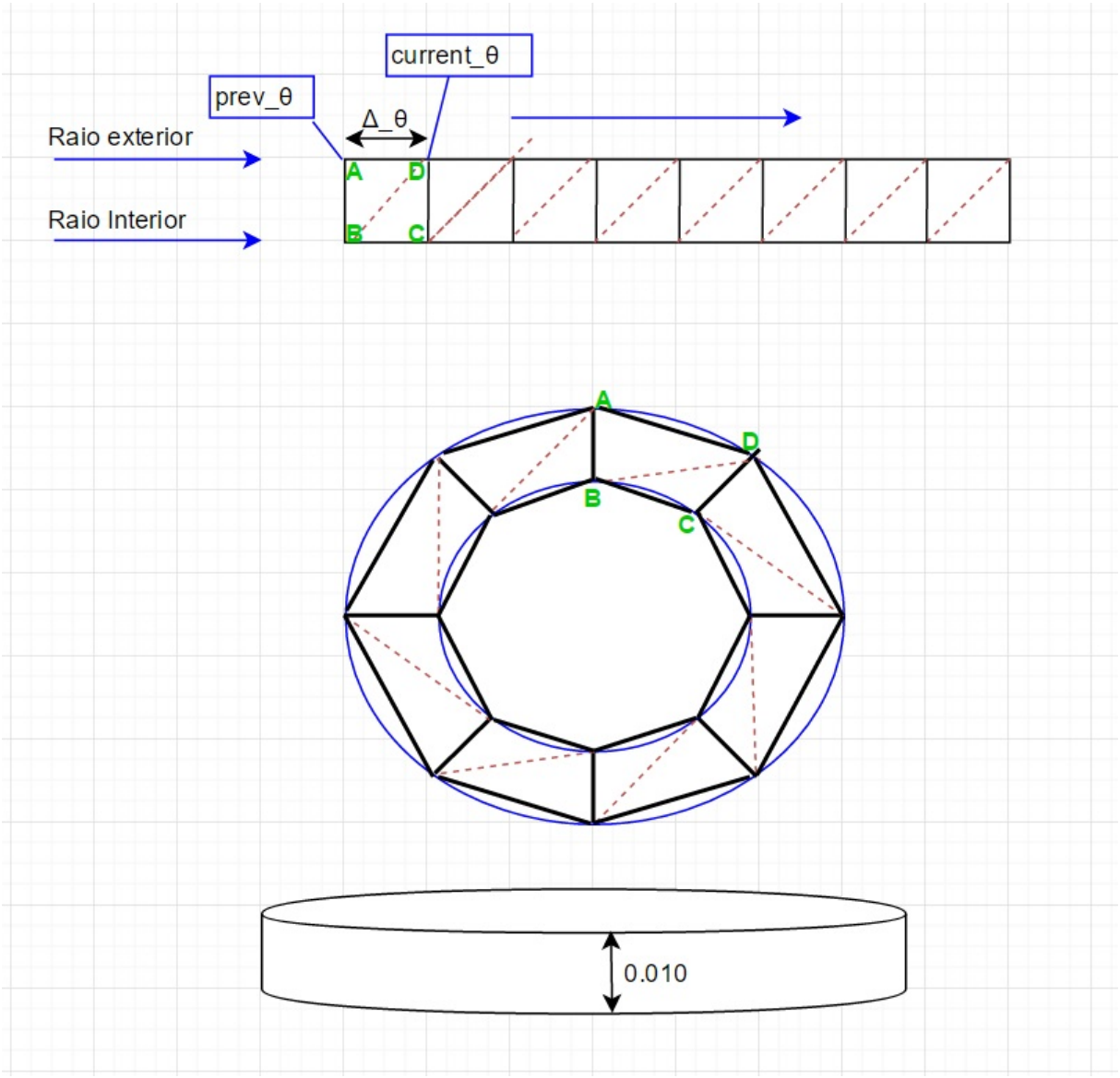

Figura 3: Diagrama Disco

Como se pode verificar o diagrama é relativamente semelhante ao da esfera. A matriz aqui observada apenas tem uma linha porque não se consideram *stacks* na representação do disco. As 8 colunas que representam as 8 *slices* (estas 8 slides servem meramente para propósitos exemplificativos).

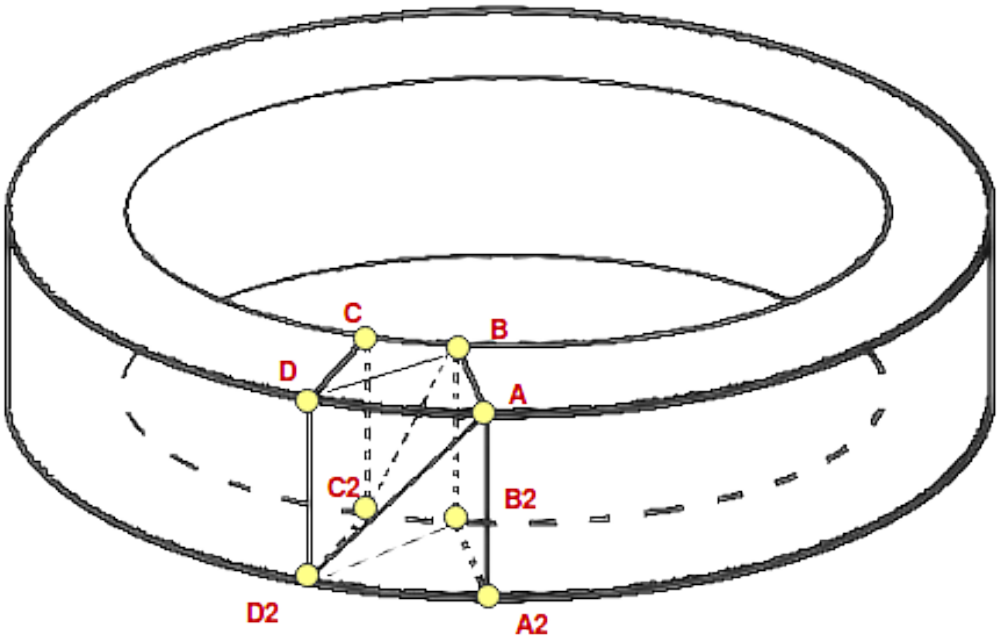

Figura 4: Pormenor dos vértices para desenho de um disco

Tal como na esfera, pretende-se criar quadricula a quadricula, criando os triângulos na orientação necessário, para a parte interior, exterior e os topos.

1.2.2 Vetores normais à superfície da geometria

À semelhança da esfera, podem-se obter os vetores normais do disco de forma analítica, sendo que no topo superior o vetor normal terá as coordenadas (0,1,0) e para o topo inferior terá as coordenadas (0,-1,0).

Para o cálculo dos vetores normais da circunferência exterior, pode-se aplicar o mesmo principio da esfera, e usar as coordenadas da circunferência com raio 1. Para o lado interior, apenas é necessário mudar a direção dos vetores normais do exterior. A Equação 4 mostra a fórmula para as normais exteriores.

$$\begin{cases} N_x = \sin(\theta) \\ N_z = \cos(\theta) \end{cases} \tag{4}$$

1.2.3 Coordenadas de texturas para a geometria

A textura que se pretende aplicar é uma faixa horizontal, sendo que o lado de fora do disco é o lado direito, e lado interior é o lado esquerdo, e que se pretende repetir. Assim para as coordenadas de textura apenas é necessário definir as coordenadas entre 0 e 1 para os topos, podendo aplicar-se uma pequena porção de cada lado, para tapar a espessura do disco.

Assim, as coordenadas para os pontos definidos na Figura 4 são:

- $A(1,0)$
- $B(0,0)$
- $C(0,1)$
- $D(1,1)$
- $A2(1,0)$
- $B2(0,0)$
- $C2(0,1)$
- $D2(1,1)$

1.3 *Patch* de Bézier baseado em ficheiro de pontos de controlo

1.3.1 Vértices da geometria

Para este projeto, requere-se que se obtenha pontos de controlo de uma superfície de Bézier, através de uma ficheiro *teapot.patch*, para a construção de um bule de chá, para a representação de um cometa. O formato de ficheiro é o que se segue:

1. Número de *patches*;
2. Índices para *patches* (16 por linha, tantas linhas como nº de *patches*);
3. Número de pontos de controlo;
4. Ponto de controlo (coord. x, y, z) por linha, tantas linhas como nº de pontos de controlo;

A função `drawPatch`, trata do processamento de leitura de transformação dos valores lidos em ficheiro num vetor de `MatrixP`, onde por último, para cada matriz armazenada no vetor anterior, para um valor de i e um valor de j dividido pela tecelagem, iterando cada um desses valor entre 0 e o valor da tecelagem, cria-se 4 pontos da superfície de Bézier, para cada iteração, onde são retirados os triângulos e armazenados num ficheiro 3d, todos os vértices calculados. Para calcular cada ponto da superfície de Bézier, é utilizada a função `getBezierPatchPoint`, que recebe um valor de u, v e uma matriz de pontos de controlo.

Se se tiver um *array* bi-dimensional de pontos $P_{i,j}$, com $i \in [0, m]$, e $j \in [0, n]$, então pode-se construir uma superfície Bézier da mesma forma usando um método similar a uma curva cúbica de Bézier. Neste caso, ao invés de um parâmetro, existem dois (u e v), onde $u \in [0, 1]$ e $v \in [0, 1]$. Uma superfície de Bézier bi-cúbica (m=n=3), dado que tem por base curvas cúbicas de Bézier definidas por 4 pontos de controlo, uma vez que é bi-dimensional, é definida por 16 pontos de controlo $P_{i,j}$, e representa-se pela *Equação 5*. Um exemplo de uma superfície de Bézier está na *Figura 5*

$$B(u, v) = \sum_{j=0}^{3} \sum_{i=0}^{3} B_i(u) P_{i,j} Bj(v) \tag{5}$$

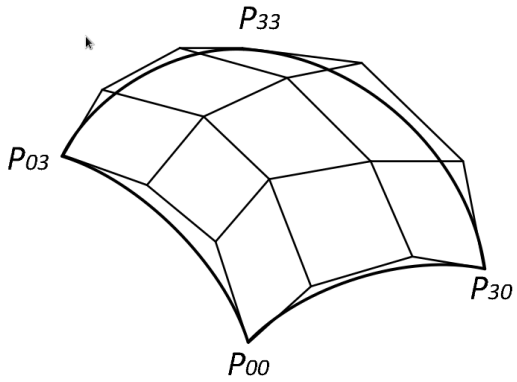

Figura 5: Superfície de Bézier bi-cúbica

Temos que:

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

O parâmetros u e v estão entre 0 e 1, à semelhança do parâmetro t da função para uma curva de Bézier, onde M é a matriz dos coeficientes obtida da *Equação* **??**. A explicação para a obtenção de um ponto numa superfície de Bézier, e, grosso modo, análoga à obtenção de um ponto numa curva de Bézier, sendo que neste caso, estão dois parâmetros a variar entre 0 e 1, criando uma malha de com curvas de Bézier, conforme está na *Figura 5*. Derivando a *Equação 5* obtém-se a representação matricial na *Equação 6*.

$$B(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} \quad (6)$$

1.3.1.1 Função `getBezierPatchPoint` Antes de descrever a função `getBezierPatchPoint` é necessário descrever algumas funções cridas par utilizar nessa função.

Para cálculos com valores escalares de vírgula flutuante, nomeadamente multiplicação de matrizes, criou-se a função `multMatrix` que multiplica duas matrizes. Para obter um cálculo correto, são obtidas as dimensões das matrizes (nº de linhas e nº de colunas), tal que, as dimensões para uma dada matriz M1 são m $\times$ n, e as dimensões para uma dada matriz M2 são p $\times$ q. Para obtenção da matriz resultado, tem-se em conta o nº de linhas da matriz M1 (m) e o nº de colunas da matriz M2 (q), onde a matriz resultante terá um dimensão m $\times$ q. Note-se que os valores n — nº de colunas da matriz M1 — e p — nº de linhas da matriz M2 têm que ser iguais. No entanto, essa verificação não é feita no código, no entanto, assume-se que o programador sabe da especificação desta função. Para guardar o valor da multiplicação de matrizes (somatório da multiplicação das linhas com as colunas), usa-se um acumulador de resultado, fazendo variar um índice k, entre 0 e p (que poderia ser n).

A função `matrixPointToScalar` obtém as coordenadas de x, ou de y, ou de z, da matriz de pontos de controlo e armazena esses valores numa matriz de escalares.

A função `getBezierPatchPoint` utiliza a *Equação 6*, sendo o código amigável na sua leitura e interpretação, uma vez que abstrai detalhes de implementação através dos tipos de matrizes já mencionados. Em primeiro lugar, a função calcula a matriz U e a matriz V, com os parâmetros u e v respetivos, conforme a equação. A segunda parte do algoritmo é multiplicar a matriz U por M, e matriz M^T por V e guarda cada resultado numa matriz. Note-se que, $M = M^T$, então a matriz M é reutilizada.

Para calcular o ponto da superfície de Bézier, é necessário obter cada coordenada dos pontos de controlo, sendo que a matriz de escalares da coordenada x será para calcular a coordenada x do ponto da superfície, a matriz de escalares da coordenada y será para calcular a coordenada y do ponto da superfície e a matriz de escalares da coordenada z será para calcular a coordenada z do ponto da superfície. Cada matriz de escalares é multiplicada pelo resultado de UM, e o resultado desta, com o resultado de M^TV ou MV. As matrizes resultantes destas operações são matrizes com dimensão 1×1, com cada valor da coordenada x, y e z do ponto da superfície. Essas coordenadas são guardadas num `Point3d` que é retornado pela função.

Os resultados da aplicação do algoritmo para uma tecelagem de 50 podem ser vistos nas figuras abaixo.

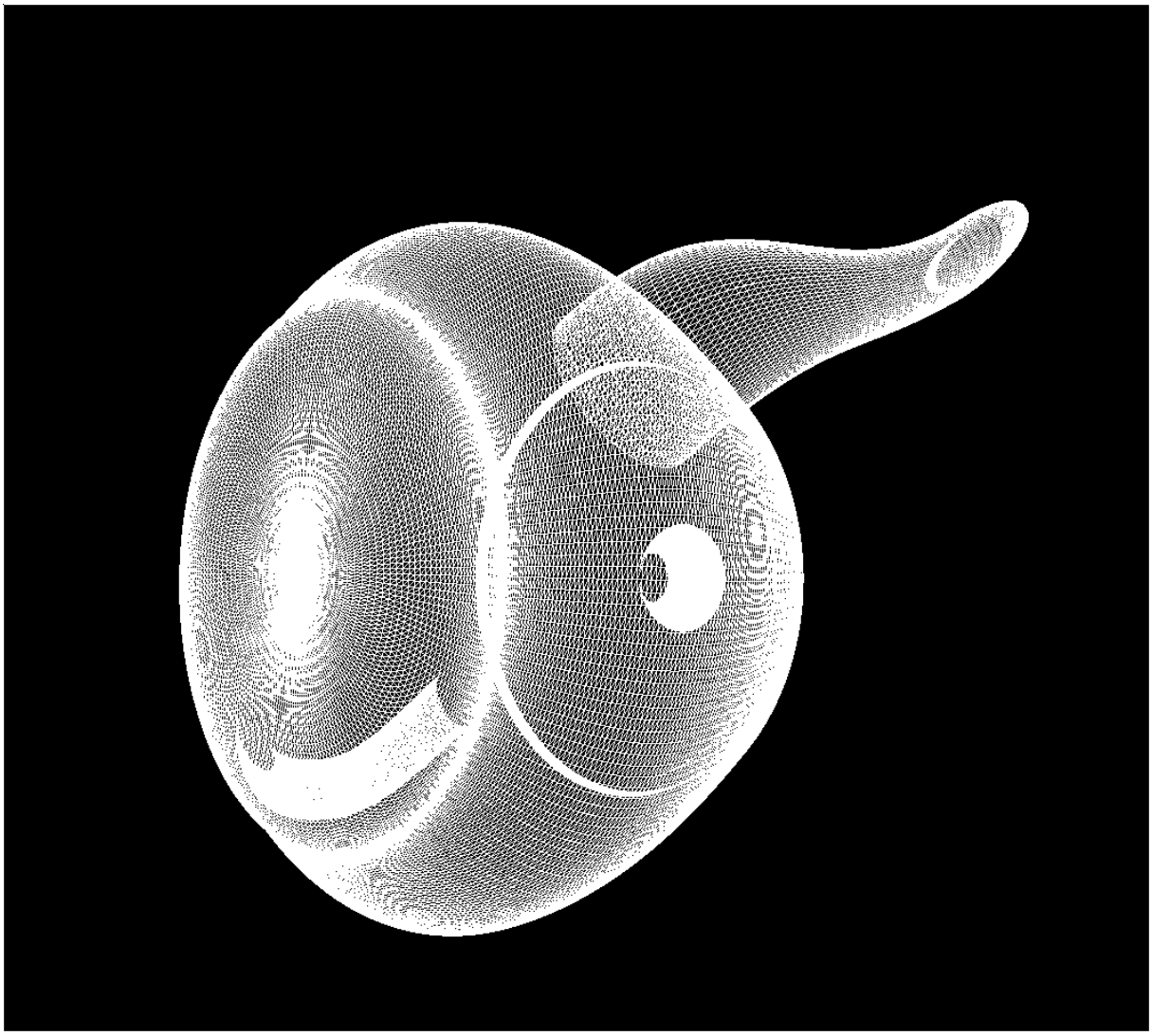

Figura 6: Bule visto de baixo

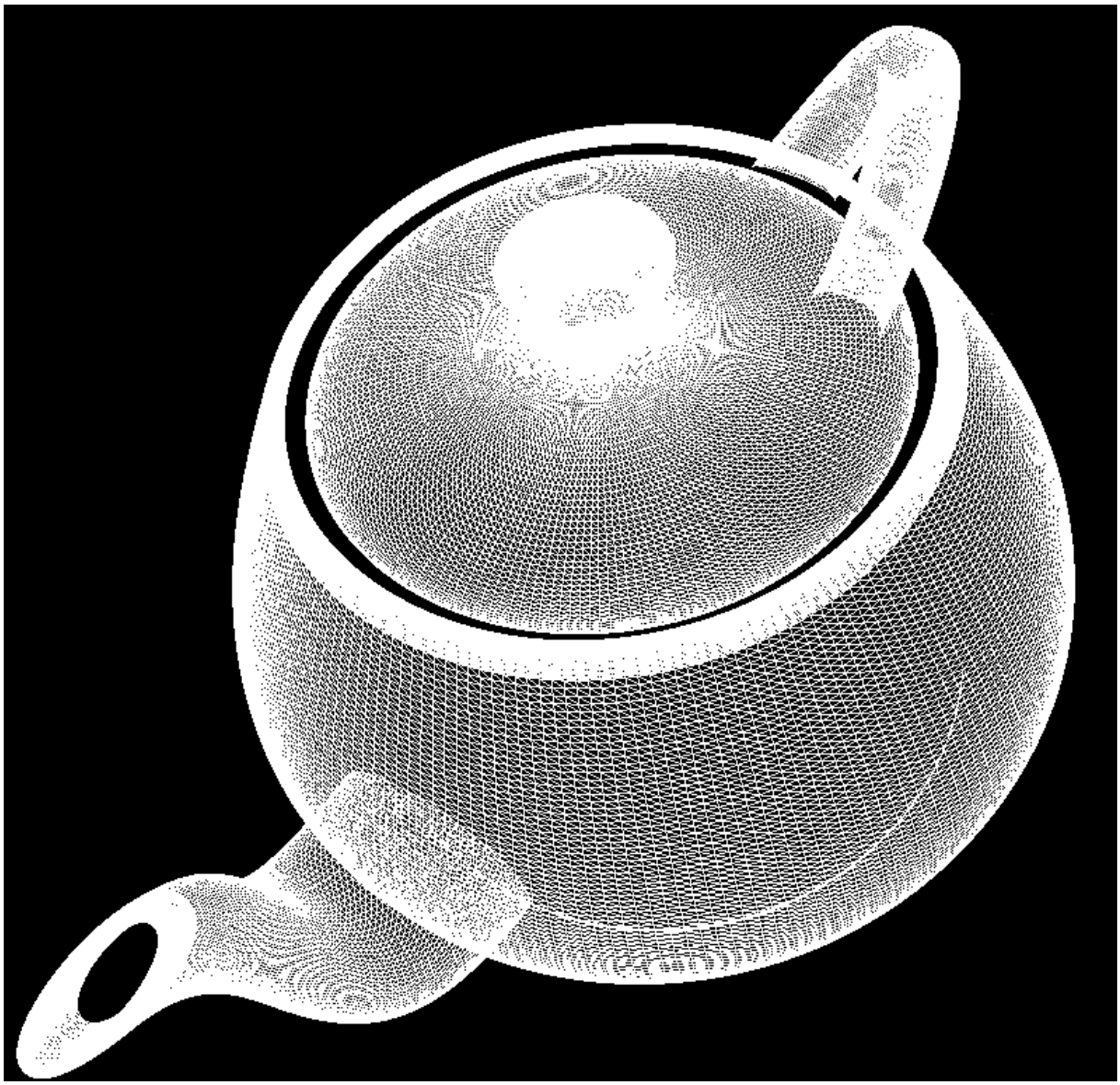

Figura 7: Bule visto de cima

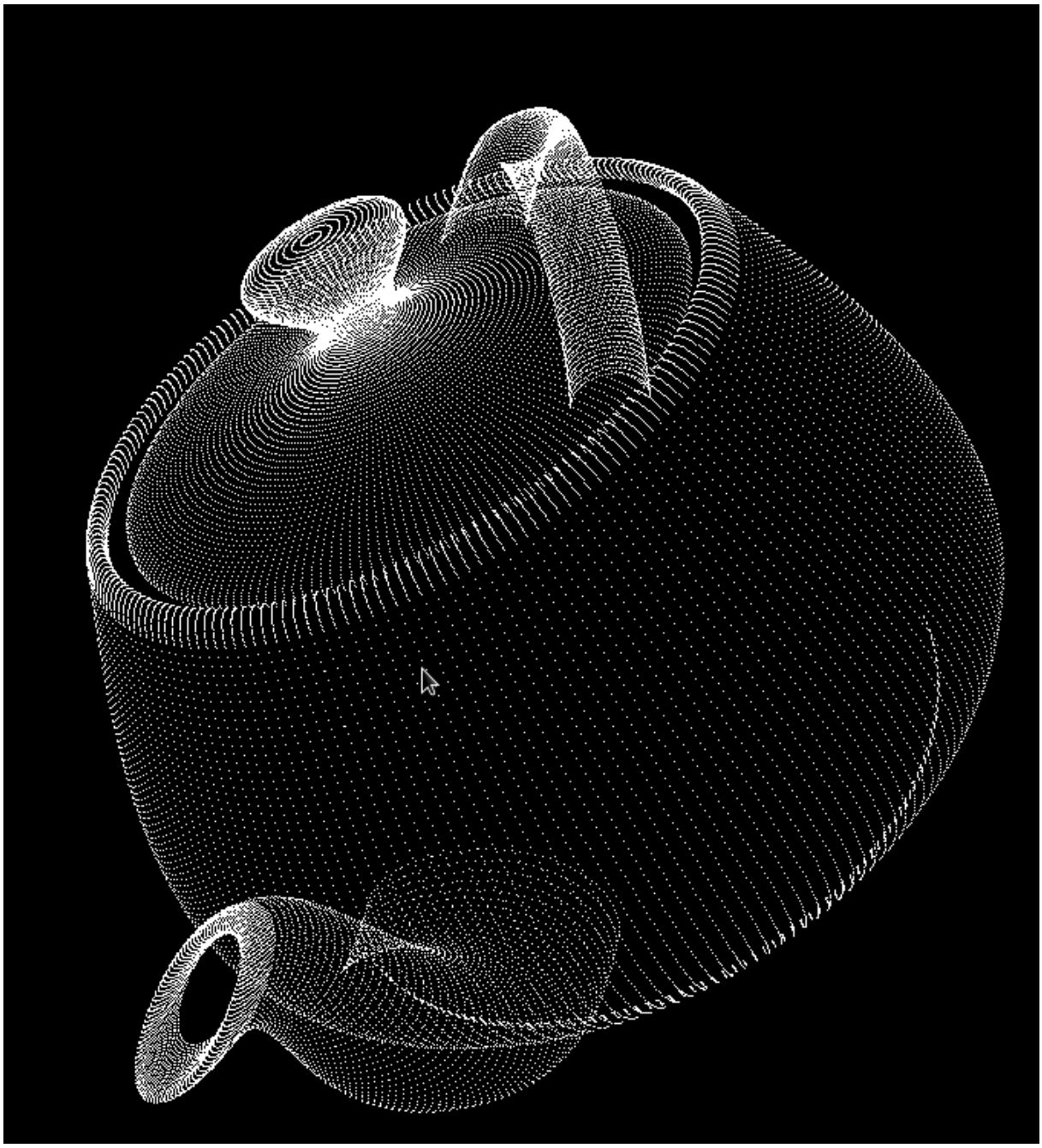

Figura 8: Pontos da superfície do bule

1.3.2 Vetores normais à superfície da geometria

Para obter os vetores normais à superfície é preciso obter primeiro as tangentes da superfície relativamente ao parâmetro u e ao parâmetro v, ou seja é preciso obter as derivadas parciais da superfície. Cada derivada representa uma direção, ou um vetor, e representa a taxa de crescimento da função em relação àquele parâmetro. As derivadas são tangentes à superfície. Para obter cada derivada utiliza-se a Equação 7 e a Equação 8.

Em seguida é necessário calcular o produto externo para obter o vetor diretor da superfície no ponto da superfície, que se obtém a partir de u e v com as funções anteriormente descritas. Este vetor não se encontra normalizado, pelo é necessário dividir o vetor pela norma para obter um vetor unitário.

$$\frac{\partial B(u,v)}{\partial u} = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} \tag{7}$$

$$\frac{\partial B(u,v)}{\partial v} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} 3v^2 \\ 2v \\ 1 \\ 0 \end{bmatrix} \tag{8}$$

A função `getPartialDerivativeBezierPatchPointParamU` aplica a Equação 7 e a função `getPartialDerivativeBezierPatchPointParamV` aplica a Equação 8 e o algoritmo destas duas funções é similar à função `getBezierPatchPoint`, exceto nos valores dos vetores da derivada U e V, e no tipo de dados que devolve. A função `getNormalBezierPatchPoint` aplica as duas funções mencionadas, calcula o o produto externo, normalizando depois o vetor. Note-se que, pode ocorrer que o vetor seja nulo, devido a descontinuidades, ou onde a derivada é 0, ou as derivas têm o mesmo valor. Daí que não se normalize esses vetores, pelo que é devolvido o vetor nulo.

1.3.3 Coordenadas de texturas para a geometria

Para as coordenadas de textura usou-se os valores de u e v para definir as mesmas.

2 Motor

Nesta secção pretende-se descrever o funcionamento do Motor *Engine*, e para tal pretende-se abordar vários aspetos desde as estruturas de dados auxiliares, estruturas de dados para os diferentes tipos de objetos que compõem o motor, descrição do processo de leitura e do processo de *rendering*. Para este projeto usaram-se *vertex array objets* para o *rendering* das geometrias, tanto vértices, vetores normais, como também coordenadas de textura.

2.1 Estruturas de dados auxiliares

Para poder manipular dos dados, tanto dos vértices, normais, coordenadas de textura, como também informação sobre os *vertex array objets — VBOs*, luzes com as suas propriedades, identificadores de texturas e ainda a raiz da estrutura de dados para armazenar informação do ficheiro XML, criou-se um tipo de dados *Models*.

Este tipo de dados é composto por vários `vector`, dos quais um para os vértices, outro para os vetores normais e outro para coordenadas de textura. Note-se que estes vetores contêm os identificadores para a inicialização e *rendering* dos VBO's, sendo que a informação para o *rendering* dos VBO's se encontra num tipo de dados homónimo, com um índice para os `vector` para e o total de vértices. De salientar que o valor de índice e do número de vértices é partilhado pelos diferentes VBO's, de vértices, normais e texturas, uma vez que o número de ocorrência de vértices, normais e pontos da textura são os mesmos, e consequentemente, o acesso por índice a cada um dos *vector* de identificadores é igual. Porém, na função de inicialização, no preenchimento do *buffer* com os valores para cada tipo de objeto foi multiplicado o número de componentes (coordenadas) de cada vértices. Assim para normais e vértices o número de vértices foi multiplicado por três, e para as coordenadas das textura, o número de vértices foi multiplicado por 2.

Para cada ficheiro com informação sobre a geometria, existe um e um só `VBO` (o tipo de dados), uma vez que a mesma geometria pode ser usadas várias vezes. Assim existe uma tabela com o nome do ficheiro como chave, tendo associado o tipo `VBO` a essa chave. Para as texturas, a estrutura é similar, só que para cada nome do ficheiro da imagem está associado um número inteiro não negativo para o identificador da textura.

Relativamente, à informação de luzes e suas componentes existe um vetor do tipo `Light*`, cujo o tipo contém informação sobre as luzes, e posteriormente neste documento será melhor documentado. Ainda para as luzes, dado que o OpenGL, na definição de propriedades materiais de cada geometria, essa propriedade é partilhada de forma global até ser novamente alterada, criou-se um vetor do tipo `Materials*` para armazenamento dos valores por defeito da propriedades de materiais. Assim, após a definição de uma propriedade de uma material de uma geometria, este vetor é percorrido aplicando as propriedades materiais por defeito.

Por último, ainda existe uma variável do tipo inteiro par controlo da atribuição do índice

do identificador do VBO's, e raiz para a estrutura de dados com a informação me memória do ficheiro XML.

2.2 Estruturas de dados — classes

As classes definidas neste projeto servem o propósito de abstrair os conceitos de computação gráfica como transformações geométricas, definição de luzes e suas propriedades, bem como classes auxiliares para representação de tipos geométricos como pontos em 3 dimensões e 4 dimensões e valores RGBA (*red, green, blue, alpha*).

Para definir a componente de luz material de cada geometria, para além das normais é necessário definir as quatro características materiais de reflexão da luz na geometria, tanto componente físicas reais (componente difusa e especular), como virtuais (ambiente e emissiva). As componentes ambiente e emissiva são virtuais, uma vez que, não é possível representar a luz ambiente de uma cena, nem se um objeto emite luz (componente emissiva) e, como tal, não dependem dos valores dos vetores normais à superfície da geometria.

Para todas as características dos materiais foram definidas classes, numa hierarquia, e à exceção da componente especular, apenas aplicam um conjunto de valores representativo dos valores RGBA. No entanto, o contexto da sua aplicação é diferente, uma vez que é invocado a função `glMaterialfv` com a respetiva componente (`GL_EMISSIVE`, `GL_DIFFUSE`, `GL_AMBIENT` e `GL_SPECULAR`) e é aplicado aquando da invocação do método `applyProperties` ou a partir da superclasse *Materials* ou partir de uma instância da própria classe. A componente especular possui ainda o brilho da componente especular (*shininess*).A Figura **??** apresenta parte da hierarquia de classes mencionada.

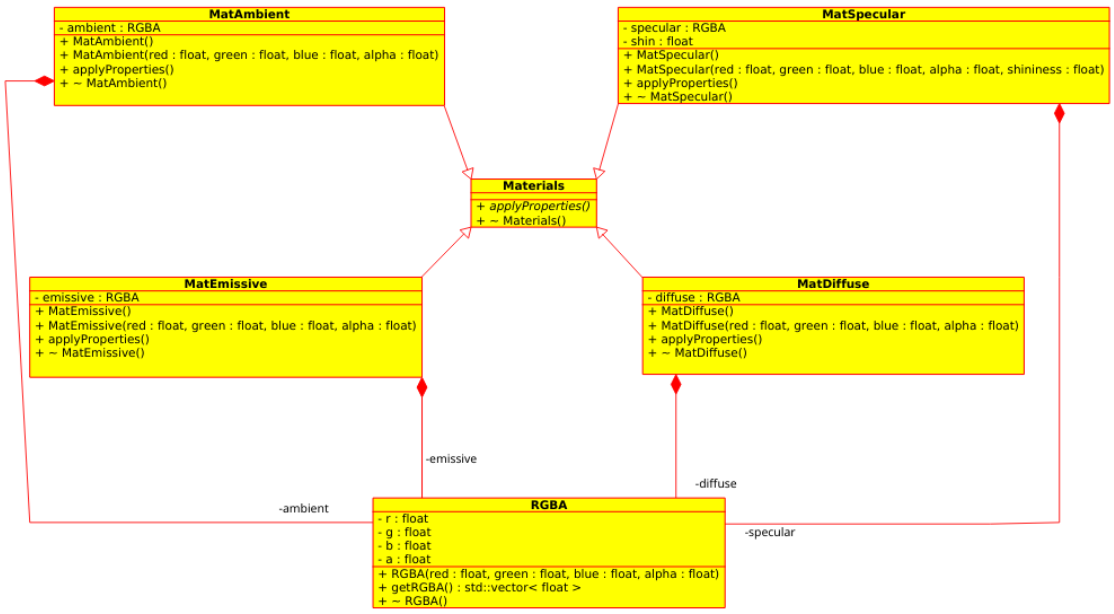

Figura 9: Hierarquia de classes de *Materials*

Para definir as luzes, podem ser definidas até um conjunto de 8 luzes (assumindo uma abordagem de luzes fixas ao espaço global), podendo estas luzes posicionais (`FixedLight`), direcionais (`DirectionalLight`) ou *spotlight* `Spolight`. Cada um desde tipos de luz pode ter várias componentes, tais como as geometrias, neste caso apenas sendo três: difusa, ambiente e especular.

Tal como para as caraterísticas materiais, cada classe possui um conjunto de valores RGBA, sendo cada característica da luz aplicada no seu contexto com a função `glLight`, com o identificador da luz a que estão associadas, com a componente correspondente (`GL_DIFFUSE`, `GL_AMBIENT` e `GL_SPECULAR`) através da superclasse `LightProperty` ou duma instância da própria classe, à semelhança da hierarquia anterior. Note-se que, aqui não faz sentido colocar o brilho da reflexão.

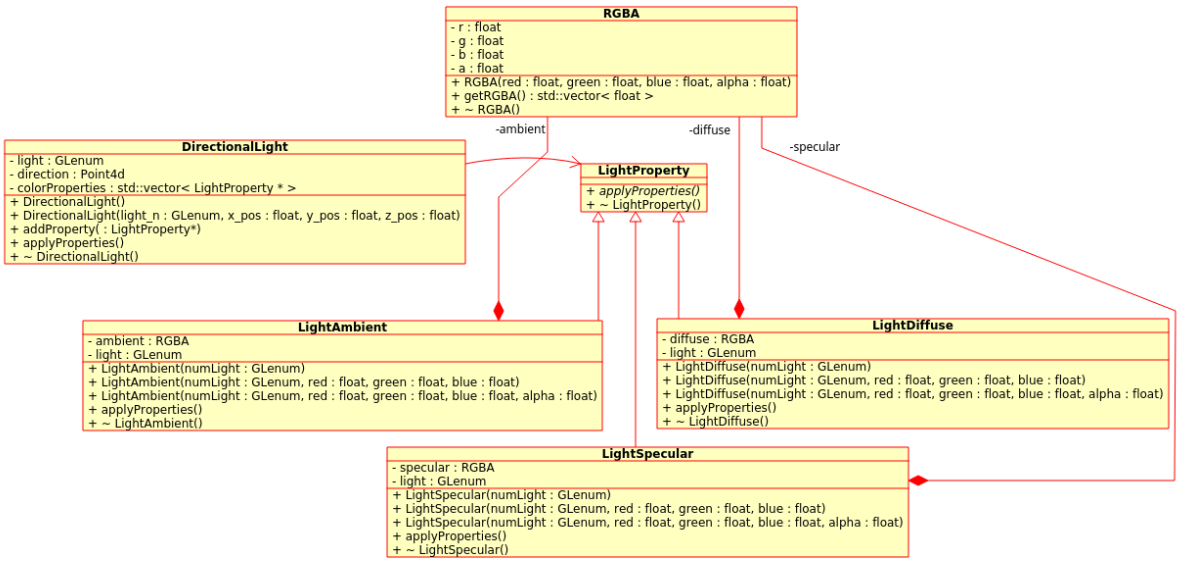

Figura 10: Hierarquia de classes de *LightProperty*

Relativamente ao tipo de luzes, estes podem ser de três tipos: direcional, posicional e *spotlight*. Cada um deste tipos possui um número variado de parâmetros, que caraterizam cada tipo de luz.

A luz direcional apenas é parametrizada pelo identificador da luz e pela direção da luz, sendo 3 coordenadas do referencial ortonormado, mais a coordenada w (coordenadas homogéneas), com o valor 0, o que representa um vetor. Uma vez que uma luz direcional pode ser considerada com uma luz infinitamente longe, esta não possui parâmetros de atenuação, no entanto, possui as propriedades da luz mencionadas atrás.

Relativamente à luz posicional, à semelhança da luz direcional, para além do identificador

da luz, recebe as coordenadas homogêneas, no entanto a coordenada w tem um valor de 1, dado que é um ponto. Nesta luz já se podem definir valores de atenuação, sendo estes valor da atenuação constante, linear ou quadrática. O inverso da soma deste valores, sendo o valor de atenuação quadrática multiplicado pelo quadrado da distância entre o ponto de luz e o vértice de uma geometria e valor da atenuação linear multiplicado pela distância, é o fator de atenuação. Esta relação está representada na Equação 9.

$$fatordeatenuao = \frac{1}{k_c + k_l d + k_q d^2} \tag{9}$$

Na Equação 9 k_c é a atenuação constante, k_l é a atenuação linear, k_q é a atenuação quadrática e d é a distância entre a posição da luz e o vértice da geometria. No OpenGL a característica da atenuação, pode ser definida pela função `glLightf`, com o identificador da luz, podendo ser `GL_CONSTANT_ATENUATION`, `GL_LINEAR_ATENUATION` e `GL_QUADRATIC_ATENUATION`, para atenuação constante, linear e quadrática respetivamente.

Por último, temos a luz do tipo *spotlight*. Este tipo de luz, para além do identificador da luz e da posição da mesma, possui uma direção para onde o cone de luz está apontado e o ângulo de abertura do cone, estando este último a 180º por defeito, podendo ser alterado para um valor entre 0 e 45º. Adicionalmente, existe um expoente de concentração de luz. Do mesmo modo, que as anteriores luzes, possui contribuições do tipo especular, ambiente e difuso.

As coordenadas homogéneas são representadas pela classe `Point4d`. À semelhança as outras hierarquias de classes já descritas, existe uma superclasse (`Light`) sendo as características de cada tipo de luz, aplicadas pela função `applyProperties` no contexto desta superclasse, ou numa instância de cada uma das subclasses. Cada uma das classes implementa a função `glLightsfv`, aplica as contribuições e ativa a luz, no método `applyProperties`, nesta ordem. Adicionalmente, existe uma método `addProperty`, que adiciona as contribuições de natureza especular, difusa ou ambiente. A hierarquia de classes pode ser vista em maior detalhe na Figura 11.

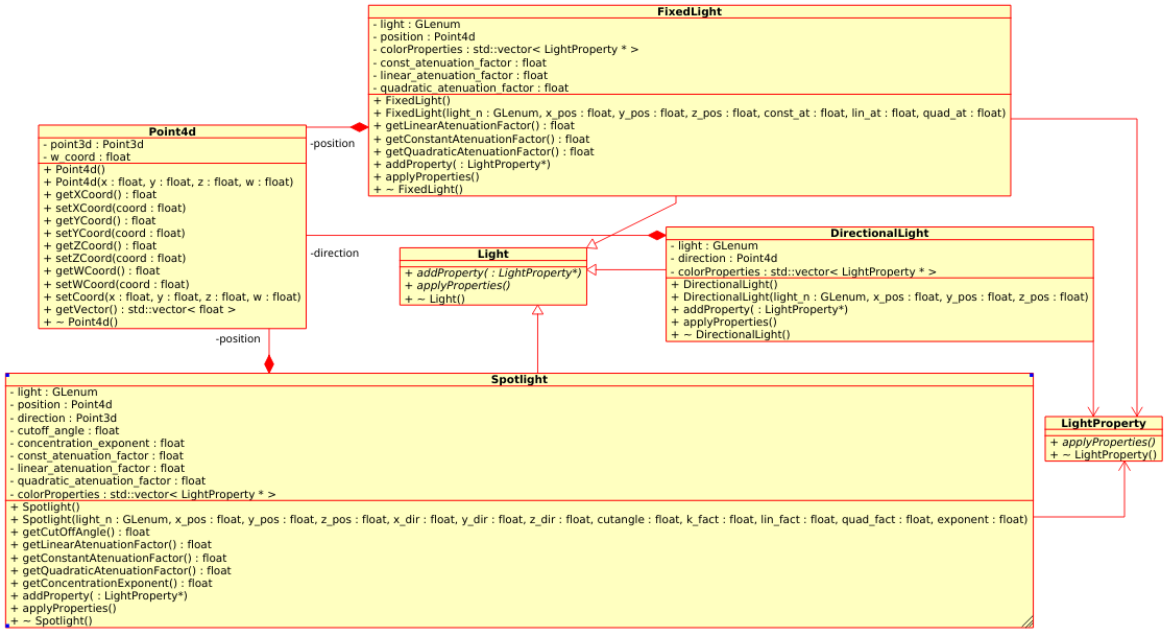

Figura 11: Hierarquia de classes de *Light*

2.3 *Vertex Array Objects*

O OpenGL possibilita dois modos de renderização: o modo imediato e o *vertex buffer objects* (VBOs). Os VBOs permitem um ganho substancial de performance, uma vez que os dados são logo enviados para a memória da placa gráfica onde residem, e por isso podem ser renderizados diretamente da placa gráfica. O modo imediato usa a memória do sistema onde os dados são inseridos *frame* a *frame*, usando uma API de renderização, o que causa peso computacional sobre o processador.

Para a utilização das VBO's é necessário recorrer a criação de *arrays* com os dados para renderizar geometria, com vetores normais para as luzes e coordenadas de textura, para uma eventual aplicação desta. Com efeito, é necessário, em primeiro lugar, ativar os *arrays* com os diferentes tipos de dados e colocar os dados num *buffer object*. Estes *arrays* são acedidos pelo seu endereço individual da sua localização em memória, sendo então desenhadas as figuras geométricas dos respetivos conteúdos dos *arrays*.

Note-se que, no OpenGl, qualquer inteiro sem sinal pode ser usado como um identificador de *buffer objecto*. Estes identificadores podem-se armazenados numa estrutura, sendo necessário, em seguida, gerar *buffers* para os vértices, para os vetores normais e coordenadas de textura (um *buffer* para cada *array* — vértices, normais e coordenadas de textura) `glGenBuffers` e ativar cada *buffer* pelo seu identificador (`glBindBuffer`) e preencher o *buffer* com os dados de cada *array* previamente mencionado.

Para desenhar, a figura geométrica é necessário definir a semântica, ou seja, definir o

offset relativo ao inicio do buffer consoante o tipo de dados, fazer o bind do objeto apropriado para fazer a renderização dos *arrays* de vértices, normais e coordenadas de textura usando a função adequada (`glDrawArrays` ou `glDrawElements`).De notar que, cada *bind* é seguido da semântica. A semântica para os vértices é feita usando a função `glVertexPointer`, definido o valor 3 o número de elementos do tipo valor de vírgula flutuante, correspondente às coordenadas de cada vértice. Relativamente à semântica para os vetores normais, é utilizada a função `glNormalPointer`. Nesta função não é necessário definir o número de coordenadas, uma vez que a própria função já o faz. No entanto, é necessário definir o tipo, como valor de vírgula flutuante. Por último, para as coordenadas das texturas é necessário definir a semântica com a função `glTexCoordPointer`, onde o número de elementos para cada vértice será dois, dado que as textura são bidimensionais. Note-se também, que é necessário fazer o *bind* da textura pelo identificador obtido pela função `loadTexture` antes de proceder ao *rendering* da geometria, sendo invocada logo de seguida o *bind* para identificador 0, para evitar que continua a desenhar a mesma textura. Este processo é feito na função `drawElement`, que será descrito mais à frente. A inicialização dos *buffers* é feita pela função `initBuffers` e para o processo de *rendering* utilizou-se a função `drawVBO`, que faz o *bind* e define a semântica, que é aplicada na função `drawElement` que aplica outros elementos para o desenho da geometria, como o *bind* da textura mencionado acima e aplicação de cores dos materiais, *reset* do mesmos.

2.4 Descrição do processo de leitura

A função de leitura identifica *tags* correspondentes às transformações geométricas, sejam elas animadas ou estáticas, e à medida que se vai lendo o ficheiro XML, vai se construindo uma árvore n-ária do tipo `Group`, como demonstra a Figura 12. No entanto, existem dois tipos de elementos do XML com interesse particular neste projeto, que são os `models` e as `lights`.

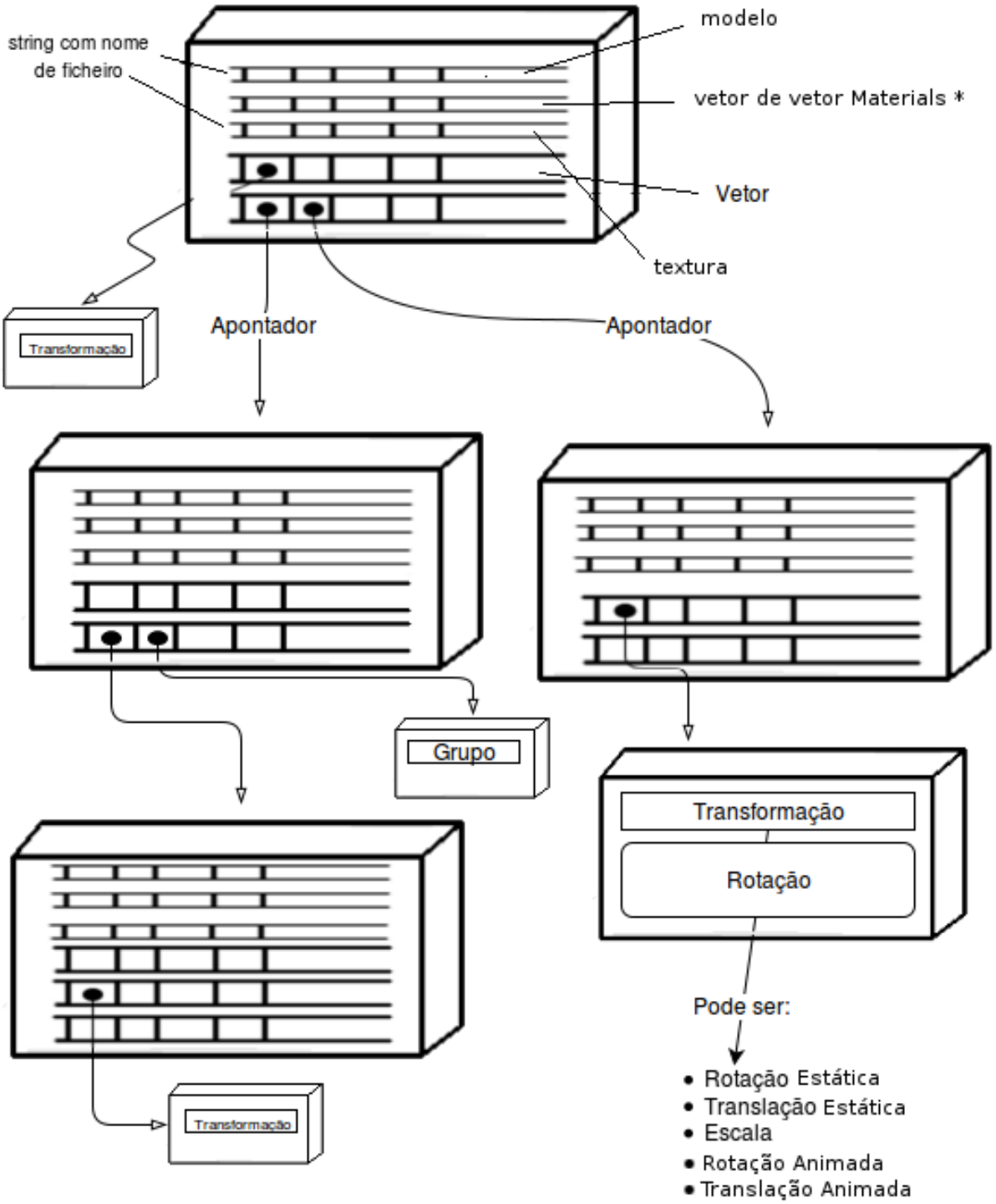

Figura 12: Árvore *n-ária* para armazenamento de grupos

2.4.1 Modelos

Para o caso de ocorrência da *tag* `models`, para cada ocorrência de uma *tag* filha `model`, a função `readXMLFromRootElement` obtém o nome do ficheiro de vértices, normais e coorde-

nadas de textura. Se a *tag* `model` também tiver um atributo `texture`, para cada propriedade material, cria a propriedade material de acordo com a ocorrência (`diffuse`, `ambient`, `specular` e `emissive`). Note-se que, se alterou o ficheiro XML para conter *tags* desta forma, dado que: é mais legível; mais fácil de tratar cada caso na leitura. Em seguida, é guardado o nome do ficheiro 3d no vetor de nomes de modelos no tipo `Group`, bem como o conjunto de propriedades materiais e nome do ficheio de textura. Note-se que, pode não existir uma ocorrência de uma textura, ou de propriedades materiais. No entanto, por uma questão de manter a representação ordenada para ser acedida por índice, caso não haja uma ocorrência de um ficheiro de textura, é adicionada a *string* vazia ou o vetor vazio.

Caso o nome da textura existir, isto é, não for uma *string* vazia, é carregada a textura com a função `loadTexture`, que devolve um identificador depois de fazer o *bind* da textura, que ocorre na função `loadTexture`. O nome do ficheiro da textura e identificador são guardados num par chave/valor na estrutura `Models` mencionada no início do capítulo. Em seguida é invocada a função `readFile`, que procederá a leitura do ficheiro 3d, inserirá um par chave/valor na tabela para o efeito em `Models`, com o nome do ficheiro 3d e o tipo `VBO`, que por sua vez possui o índice dos vetores de identificadores de *buffers* (vértices, normais e coordenadas de textura) — um índice para os três vetores, como já foi explicado —, e inicializará os *vertex array objects*.

2.4.2 Luzes

Relativamente às luzes, neste modelo são descritas fora de um grupo, no entanto, a função de leitura permite a flexibilidade de ter luzes dentro de um grupo. Mesmo assim, pretende-se que as luzes sejam declaradas no inicio do ficheiro do documento XML, uma vez que os atributos das luzes e seus tipos são armazenados numa estrutura à parte (`Models`), para serem invocadas na `renderScene` após o `gluLookAt`, antes de quaisquer transformações geométricas, de forma as luzes ficarem fixas no espaço global.

Com efeito, para cada ocorrência da *tag* `lights`, para cada elemento `light` filho, se o tipo (`type`) for do valor `DIR`, cria uma luz direcional, se for do valor `POINT` cria uma luz posicional e se for do tipo `SPOT`, cria um *spotlight*. Para cada uma desta ocorrências, se existirem propriedades da luz, à semelhança da propriedades materiais, cada componente é criado conforme a ocorrência (`diffuse`, `specular` e `ambient`). Estas componentes são adicionadas à luz, no entanto, podem não existir componentes da luz, e nesse caso nada é adicionado. A luz é adicionada ao vetor de `Light*` em `Models`.

2.5 Descrição do ciclo de *rendering*

Para fazer o *rendering* da estrutura de dados em memória, implementou-se uma função de travessia da árvore, colocada na função `renderScene`, após a função `glLoadIndentity` e `gluLookAt`, nesta sequência.

Com efeito, a primeira instrução é a `glPushMatrix`, uma vez que se pretende colocar uma matriz para aplicação das transformações no topo da *stack* de matrizes do *OpenGL*. Em seguida, para cada transformação contida num `Group`, é invocada a função `applyTrasformation`, que aplica as transformações as transformações geométricas.

Em seguida, é invocada a função `drawElement`. Esta função especifica a primitiva para que será criada com os vértices, normais e coordenadas de textura em memória, usando a função `drawVBO`. Adicionalmente, itera pelas as estruturas de `Group` (vetores de nomes de ficheiros 3d, vetor de vetor de `Materials*`) e procura pela chave do nome do ficheiro, 3d ou textura, o valor do tipo `VBO` e identificador da textura, respetivamente, nas tabelas em `Models`. Caso o nome do ficheiro 3d existir é que invoca, a função `drawVBO`, e antes disso, verifica se o nome da textura existe, e só se existir é que faz o *bind* da textura. A primeira estrutura a ser iterada é o vetor de vetores de *Materials**, sendo que para cada valor de vetor de *Materials**, as propriedades são aplicadas conforme o contexto, pela a invocação do método da superclasse `applyProperties`. Após o *rendering* dos *vertex array objects*, todas as propriedades materiais por defeito do OpenGL são aplicadas, para evitar que propriedades de diferentes geometrias, sejam aplicadas de forma transversal, até que sejam alteradas novamente, que a aplicação dos valores por defeito faz.

Como já foi mencionado, a aplicação das luzes acontece na `renderScene`, após o `gluLookAt` e antes da função `traverseTree` que possui a descrição do modelo XML em memória. Para a aplicação das luzes aplica-se o mesmo principio de utilização da hierarquia de classes que se usou até aqui, e para cada ocorrência de uma instância de `Light*` no vetor em `Models` é invocada a função `applyProperties` de `Light`, sendo a mesma função aplicada, no seu devido contexto nas subclasses.

3 Resultados

Os resultados obtidos estão nas imagens seguintes.

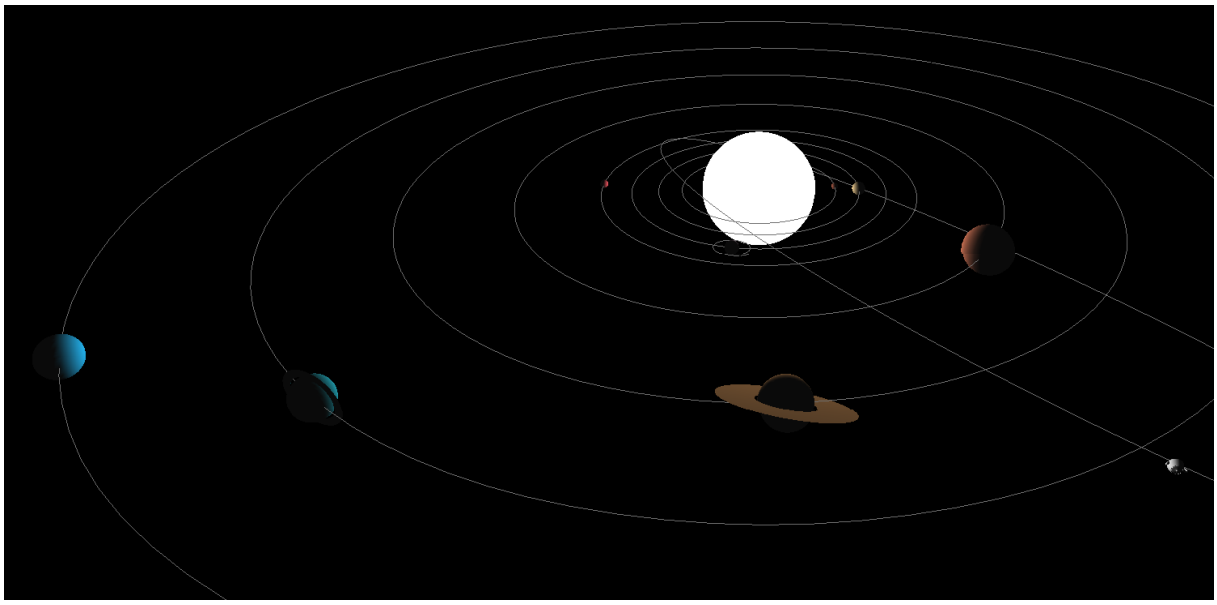

Figura 13: *Rendering* do modelo com cores e um luz posicional

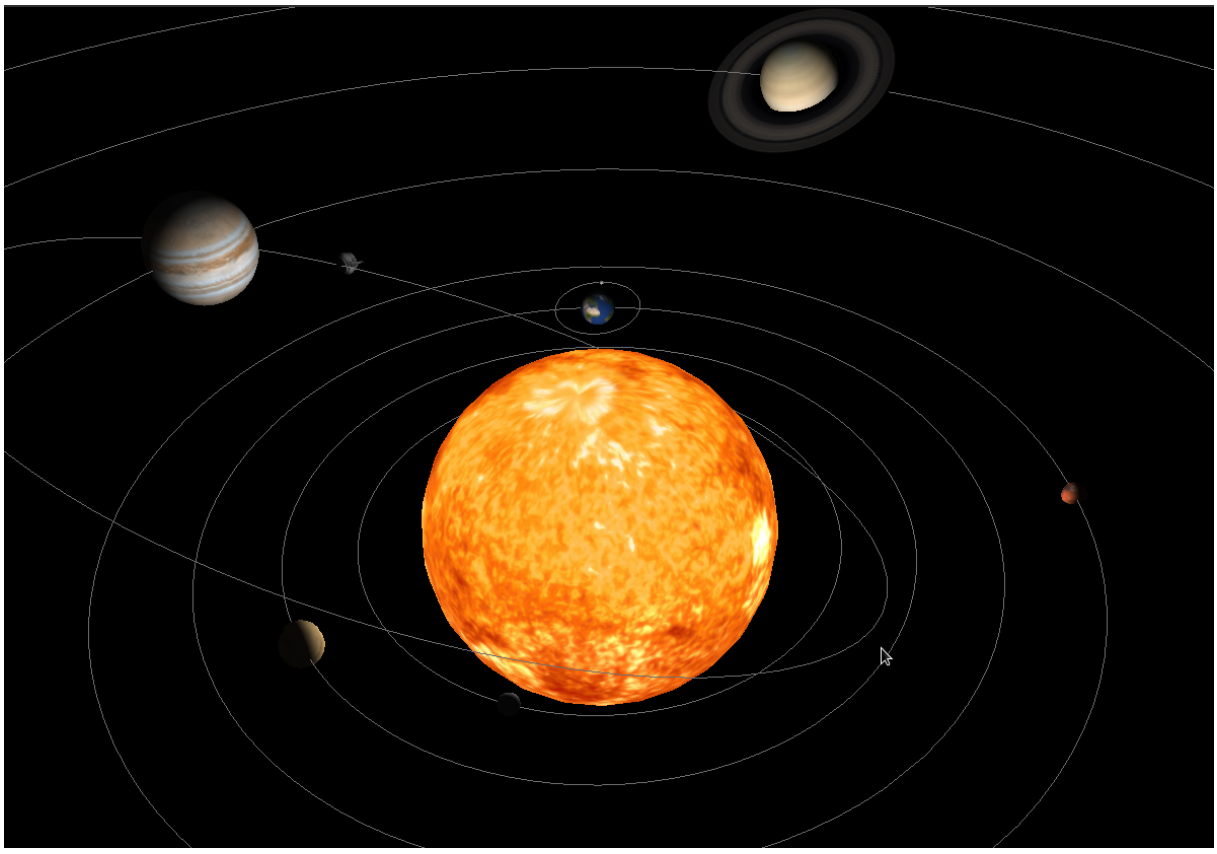

Figura 14: *Rendering* do modelo com texturas e um luz posicional

A *Figura 15* mostra o modelo com cometa.

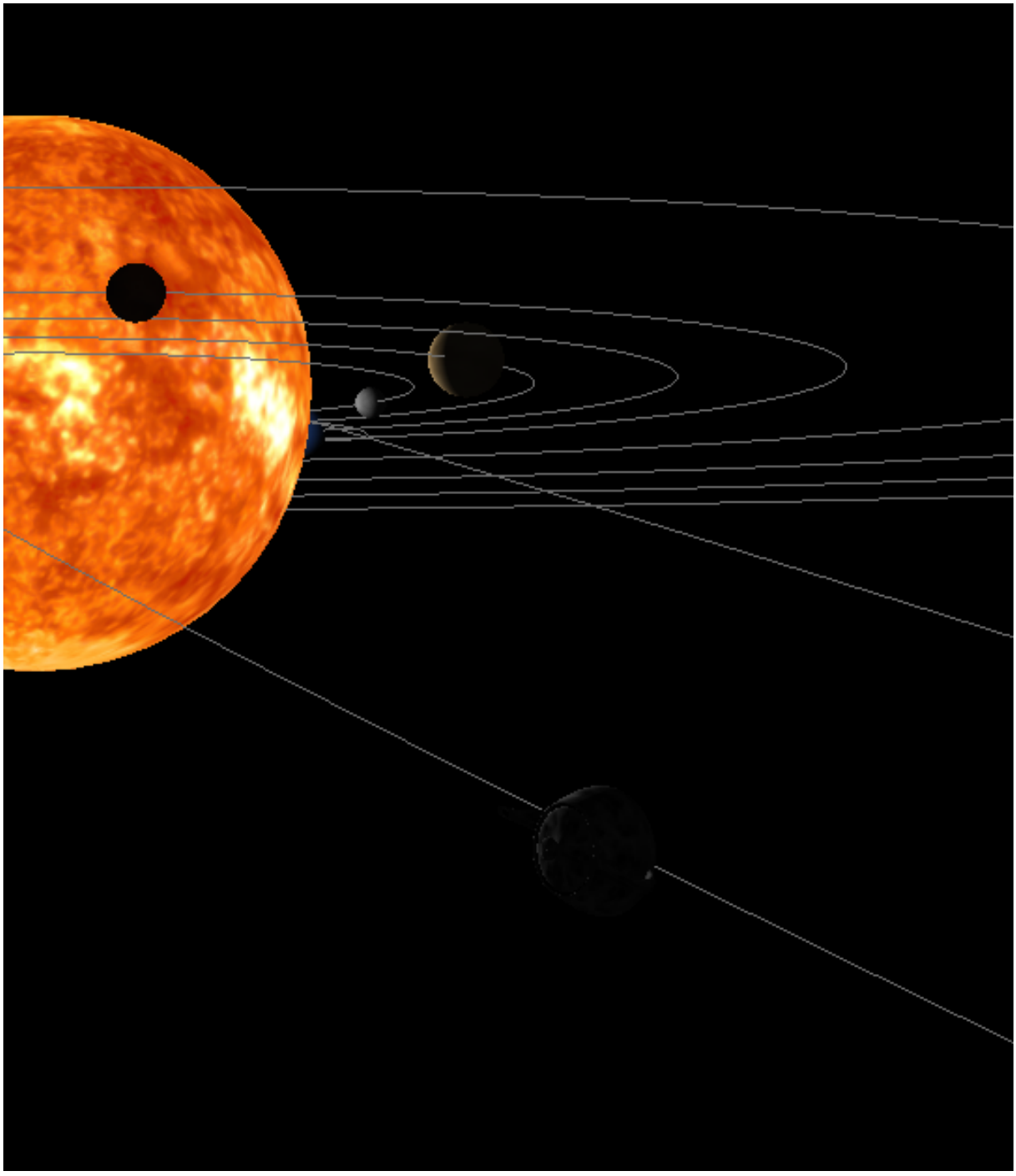

Figura 15: *Rendering* do modelo com foco no cometa

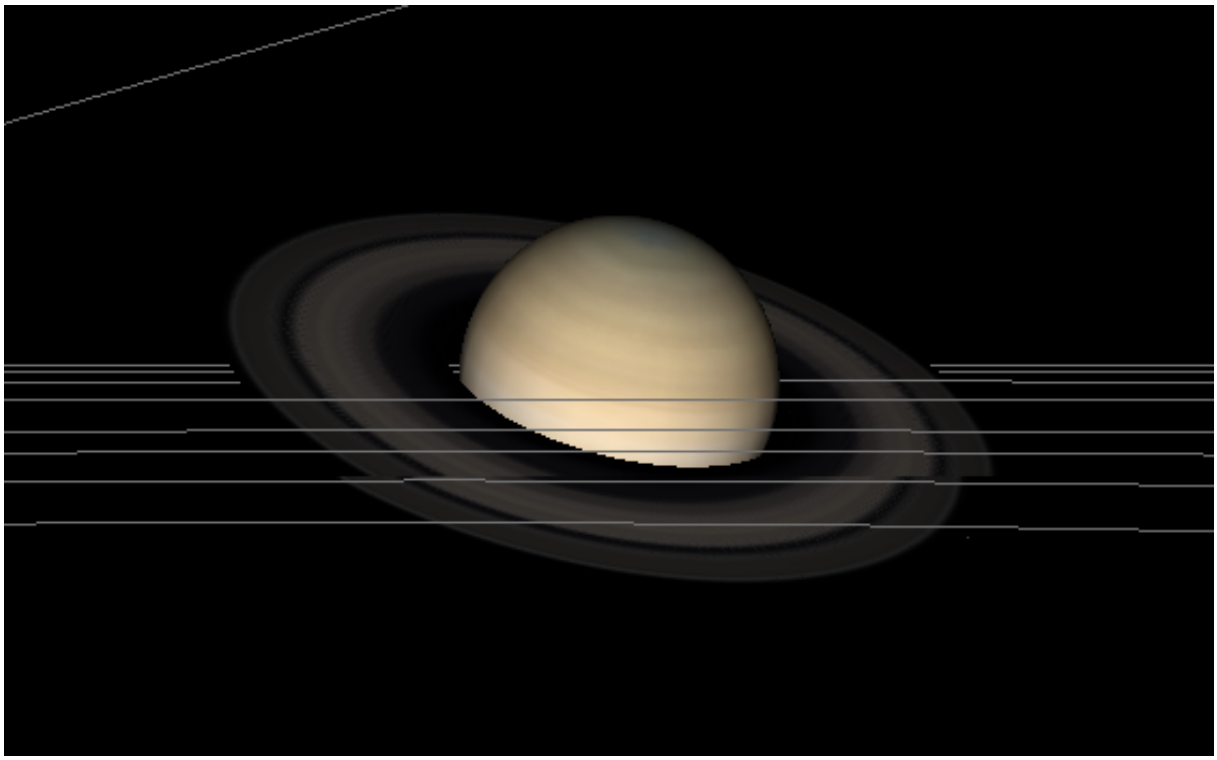

Figura 16: *Rendering* do modelo com foco em Saturno

Conclusão

Em suma, podemos concluir que o projeto foi um sucesso parcial, dado que não foi possível iluminar os topos do bule, e que possivelmente poderiam ter sido efetuados mais testes. Não foram geradas coordenadas de textura para a caixa, cone e plano, nem vetores normais destas geometrias. No entanto, ficaram implementadas os vários tipos de luzes, com as suas propriedades de cor, bem como as propriedades materiais da geometria, assim como a aplicação de texturas.

Como trabalho futuro, sugere-se implementar o cálculo dos vetores normais e coordenadas de textura para as geometrias que faltam, bem tentar o cálculo de vetores normais através de interpolação para o bule, tentando diminuir o tamanho de cada triângulo, para introduzir o efeito MACH ou usar diretamente o modelo *flat* sabendo das desvantagens de quaisquer um destes métodos.

Referências

[1] A. Boreskov and E. Shikin, *Computer Graphics: From Pixels to Programmable Graphics Hardware*, 2013.

[2] F. Dunn and I. Parberry, *3D Math Primer for Graphics and Game Development*, 2nd ed. Wordware Pub, 2002.

[3] B. Eckel, *Thinking in C++*. Prentice Hall, 2000.

[4] ——, *Thinking in C++*. Prentice Hall, 2000.

[5] A. Koenig and B. E. Moo, *Accelerated C++ : Practical Programming by Example*. Addison-Wesley, 2000.

[6] E. Lengyel, *Mathematics for 3D Game Programming and Computer Graphics*, 3rd ed. Course Technology PTR, 2012.

[7] S. B. Lippman, J. Lajoie, and B. E. Moo, *C++ Primer*, 5th ed. Addison-Wesley, 2013.

[8] A. Ramires, "GLUT Tutorial," 2011. [Online]. Available: http://www.lighthouse3d.com/tutorials/glut-tutorial/

[9] P. Shirley and S. Marschner, *Fundamentals of Computer Graphics*. CRC Press, 2015.

[10] D. Shreiner and Khronos OpenGL ARB Working Group., *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, 7th ed. Addison-Wesley, 2010.

[11] B. Stroustrup, *A Tour Of C++*. Addison-Wesley, 2014.

ANEXOS

A Modelo do Sistema Solar

```
<?xml version="1.0"?>
<scene>
<lights>
<light type="POINT" posX="0" posY="0" posZ="0" />
</lights>
  <!-- Haley's comet -->
  <group>
    <rotate angle="162.26" axisX="1"/>
    <translate Z="100"/>
    <group>
      <translate time="81.45">
        <point X="0" Y="0" Z="133"/>
        <point X="12.6286" Y="0" Z="122.876"/>
        <point X="23.3345" Y="0" Z="94.0452"/>
        <point X="30.488" Y="0" Z="50.8969"/>
        <point X="33" Y="0" Z="0"/>
        <point X="30.488" Y="0" Z="-50.8969"/>
        <point X="23.3345" Y="0" Z="-94.0452"/>
        <point X="12.6286" Y="0" Z="-122.876"/>
        <point X="0" Y="0" Z="-133"/>
        <point X="-12.6286" Y="0" Z="-122.876"/>
        <point X="-23.3345" Y="0" Z="-94.0452"/>
        <point X="-30.488" Y="0" Z="-50.8969"/>
        <point X="-33" Y="0" Z="0"/>
        <point X="-30.488" Y="0" Z="50.8969"/>
        <point X="-23.3345" Y="0" Z="94.0452"/>
        <point X="-12.6286" Y="0" Z="122.876"/>
      </translate>
      <group>
        <rotate time="81.45" axisY="1"/>
        <scale X="1" Y="1" Z="1"/>
        <models>
          <model file="teapotLight2.3d"  texture="comet.jpg">
            <diffuse R="1" G="1" B="1"/>
            <specular R="1" G="1" B="1" shin="100"/>
          </model>
        </models>
      </group>
    </group>
  </group>
  <group>
    <!-- Sun -->
    <group>
      <rotate angle="7.25" axisZ="1"/>
      <scale X="17" Y="17" Z="17"/>
      <models>
```

```
    <model file="sphereLight.3d" texture="sun.jpg">
      <emissive R="1" G="1" B="1"/>
    </model>
  </models>
</group>
<!-- Mercury -->
<group>
  <translate time="6.69">
    <point X="0" Y="0" Z="23"/>
    <point X="8.8017" Y="0" Z="21.2492"/>
    <point X="16.2635" Y="0" Z="16.2635"/>
    <point X="21.2492" Y="0" Z="8.8017"/>
    <point X="23" Y="0" Z="0"/>
    <point X="21.2492" Y="0" Z="-8.8017"/>
    <point X="16.2635" Y="0" Z="-16.2635"/>
    <point X="8.8017" Y="0" Z="-21.2492"/>
    <point X="0" Y="0" Z="-23"/>
    <point X="-8.8017" Y="0" Z="-21.2492"/>
    <point X="-16.2635" Y="0" Z="-16.2635"/>
    <point X="-21.2492" Y="0" Z="-8.8017"/>
    <point X="-23" Y="0" Z="0"/>
    <point X="-21.2492" Y="0" Z="8.8017"/>
    <point X="-16.2635" Y="0" Z="16.2635"/>
    <point X="-8.8017" Y="0" Z="21.2492"/>
  </translate>
  <rotate angle="0.03" axisZ="1"/>
  <rotate time="15.85" axisY="1"/>
  <models>
    <model file="sphereLight.3d" texture="mercury.jpg">
      <diffuse R="1" G="1" B="1"/>
    </model>
  </models>
</group>
<!-- Venus -->
<group>
  <translate time="11.02">
    <point X="0" Y="0" Z="30"/>
    <point X="11.4805" Y="0" Z="27.7164"/>
    <point X="21.2132" Y="0" Z="21.2132"/>
    <point X="27.7164" Y="0" Z="11.4805"/>
    <point X="30" Y="0" Z="0"/>
    <point X="27.7164" Y="0" Z="-11.4805"/>
    <point X="21.2132" Y="0" Z="-21.2132"/>
    <point X="11.4805" Y="0" Z="-27.7164"/>
    <point X="0" Y="0" Z="-30"/>
    <point X="-11.4805" Y="0" Z="-27.7164"/>
    <point X="-21.2132" Y="0" Z="-21.2132"/>
    <point X="-27.7164" Y="0" Z="-11.4805"/>
    <point X="-30" Y="0" Z="0"/>
    <point X="-27.7164" Y="0" Z="11.4805"/>
    <point X="-21.2132" Y="0" Z="21.2132"/>
    <point X="-11.4805" Y="0" Z="27.7164"/>
  </translate>
```

```
    <scale X="2" Y="2" Z="2"/>
    <rotate angle="180" axisX="1"/>
    <rotate angle="2.64" axisZ="1"/>
    <rotate time="29.54" axisY="1"/>
    <models>
     <model file="sphereLight.3d" texture="venus.jpg">
        <diffuse R="1" G="1" B="1"/>
      </model>
    </models>
  </group>
  <!-- Earth -->
  <group>
    <translate time="13.46">
      <point X="0" Y="0" Z="38"/>
      <point X="14.542" Y="0" Z="35.1074"/>
      <point X="26.8701" Y="0" Z="26.8701"/>
      <point X="35.1074" Y="0" Z="14.542"/>
      <point X="38" Y="0" Z="0"/>
      <point X="35.1074" Y="0" Z="-14.542"/>
      <point X="26.8701" Y="0" Z="-26.8701"/>
      <point X="14.542" Y="0" Z="-35.1074"/>
      <point X="0" Y="0" Z="-38"/>
      <point X="-14.542" Y="0" Z="-35.1074"/>
      <point X="-26.8701" Y="0" Z="-26.8701"/>
      <point X="-35.1074" Y="0" Z="-14.542"/>
      <point X="-38" Y="0" Z="0"/>
      <point X="-35.1074" Y="0" Z="14.542"/>
      <point X="-26.8701" Y="0" Z="26.8701"/>
      <point X="-14.542" Y="0" Z="35.1074"/>
    </translate>
    <group>
      <scale X="2" Y="2" Z="2"/>
      <rotate angle="6.68" axisX="1"/>
      <rotate time="2" axisY="1"/>
      <models>
       <model file="sphereLight.3d" texture="earth.jpg">
        <diffuse R="1" G="1" B="1"/>
      </model>
      </models>
    </group>
  </group>
  <!-- Moon -->
  <group>
    <translate time="13.46">
      <point X="0" Y="0" Z="38"/>
      <point X="14.542" Y="0" Z="35.1074"/>
      <point X="26.8701" Y="0" Z="26.8701"/>
      <point X="35.1074" Y="0" Z="14.542"/>
      <point X="38" Y="0" Z="0"/>
      <point X="35.1074" Y="0" Z="-14.542"/>
      <point X="26.8701" Y="0" Z="-26.8701"/>
      <point X="14.542" Y="0" Z="-35.1074"/>
      <point X="0" Y="0" Z="-38"/>
```

```
      <point X="-14.542" Y="0" Z="-35.1074"/>
      <point X="-26.8701" Y="0" Z="-26.8701"/>
      <point X="-35.1074" Y="0" Z="-14.542"/>
      <point X="-38" Y="0" Z="0"/>
      <point X="-35.1074" Y="0" Z="14.542"/>
      <point X="-26.8701" Y="0" Z="26.8701"/>
      <point X="-14.542" Y="0" Z="35.1074"/>
    </translate>
    <group>
      <rotate angle="5.14" axisZ="1"/>
      <translate time="50">
        <point X="0" Y="0" Z="5"/>
        <point X="1.9134" Y="0" Z="4.6194"/>
        <point X="3.5355" Y="0" Z="3.5355"/>
        <point X="4.6194" Y="0" Z="1.9134"/>
        <point X="5" Y="0" Z="0"/>
        <point X="4.6194" Y="0" Z="-1.9134"/>
        <point X="3.5355" Y="0" Z="-3.5355"/>
        <point X="1.9134" Y="0" Z="-4.6194"/>
        <point X="0" Y="0" Z="-5"/>
        <point X="-1.9134" Y="0" Z="-4.6194"/>
        <point X="-3.5355" Y="0" Z="-3.5355"/>
        <point X="-4.6194" Y="0" Z="-1.9134"/>
        <point X="-5" Y="0" Z="0"/>
        <point X="-4.6194" Y="0" Z="1.9134"/>
        <point X="-3.5355" Y="0" Z="3.5355"/>
        <point X="-1.9134" Y="0" Z="4.6194"/>
      </translate>
      <scale X="0.27" Y="0.27" Z="0.27"/>
      <rotate angle="6.68" axisY="1"/>
      <models>
       <model file="sphereLight.3d" texture="moon.jpg">
        <diffuse R="1" G="1" B="1"/>
      </model>
      </models>
    </group>
  </group>
  <!-- Mars -->
  <group>
    <translate time="16.4">
      <point X="0" Y="0" Z="47"/>
      <point X="17.9861" Y="0" Z="43.4223"/>
      <point X="33.234" Y="0" Z="33.234"/>
      <point X="43.4223" Y="0" Z="17.9861"/>
      <point X="47" Y="0" Z="0"/>
      <point X="43.4223" Y="0" Z="-17.9861"/>
      <point X="33.234" Y="0" Z="-33.234"/>
      <point X="17.9861" Y="0" Z="-43.4223"/>
      <point X="0" Y="0" Z="-47"/>
      <point X="-17.9861" Y="0" Z="-43.4223"/>
      <point X="-33.234" Y="0" Z="-33.234"/>
      <point X="-43.4223" Y="0" Z="-17.9861"/>
      <point X="-47" Y="0" Z="0"/>
```

```
      <point X="-43.4223" Y="0" Z="17.9861"/>
      <point X="-33.234" Y="0" Z="33.234"/>
      <point X="-17.9861" Y="0" Z="43.4223"/>
    </translate>
    <scale X="1.25" Y="1.25" Z="1.25"/>
    <rotate angle="25.19" axisZ="1"/>
    <rotate time="2.72" axisY="1"/>
    <models>
     <model file="sphereLight.3d" texture="mars.jpg">
        <diffuse R="1" G="1" B="1"/>
      </model>
    </models>
  </group>
  <!-- Jupiter -->
  <group>
    <translate time="36.6">
      <point X="0" Y="0" Z="72"/>
      <point X="27.5532" Y="0" Z="66.5193"/>
      <point X="50.9117" Y="0" Z="50.9117"/>
      <point X="66.5193" Y="0" Z="27.5532"/>
      <point X="72" Y="0" Z="0"/>
      <point X="66.5193" Y="0" Z="-27.5532"/>
      <point X="50.9117" Y="0" Z="-50.9117"/>
      <point X="27.5532" Y="0" Z="-66.5193"/>
      <point X="0" Y="0" Z="-72"/>
      <point X="-27.5532" Y="0" Z="-66.5193"/>
      <point X="-50.9117" Y="0" Z="-50.9117"/>
      <point X="-66.5193" Y="0" Z="-27.5532"/>
      <point X="-72" Y="0" Z="0"/>
      <point X="-66.5193" Y="0" Z="27.5532"/>
      <point X="-50.9117" Y="0" Z="50.9117"/>
      <point X="-27.5532" Y="0" Z="66.5193"/>
    </translate>
    <scale X="7" Y="7" Z="7"/>
    <rotate angle="3.13" axisZ="1"/>
    <rotate time="1.69" axisY="1"/>
    <models>
      <model file="sphereLight.3d" texture="jupiter.jpg">
        <diffuse R="1" G="1" B="1"/>
      </model>
    </models>
  </group>
  <!-- Saturn -->
  <group>
    <translate time="54.6">
      <point X="0" Y="0" Z="105"/>
      <point X="40.1818" Y="0" Z="97.0074"/>
      <point X="74.2462" Y="0" Z="74.2462"/>
      <point X="97.0074" Y="0" Z="40.1818"/>
      <point X="105" Y="0" Z="0"/>
      <point X="97.0074" Y="0" Z="-40.1818"/>
      <point X="74.2462" Y="0" Z="-74.2462"/>
      <point X="40.1818" Y="0" Z="-97.0074"/>
```

```
      <point X="0" Y="0" Z="-105"/>
      <point X="-40.1818" Y="0" Z="-97.0074"/>
      <point X="-74.2462" Y="0" Z="-74.2462"/>
      <point X="-97.0074" Y="0" Z="-40.1818"/>
      <point X="-105" Y="0" Z="0"/>
      <point X="-97.0074" Y="0" Z="40.1818"/>
      <point X="-74.2462" Y="0" Z="74.2462"/>
      <point X="-40.1818" Y="0" Z="97.0074"/>
    </translate>
    <scale X="6" Y="6" Z="6"/>
    <rotate angle="26.63" axisZ="1"/>
    <rotate time="1.83" axisY="1"/>
    <models>
     <model file="sphereLight.3d" texture="saturn.jpg">
        <diffuse R="1" G="1" B="1"/>
      </model>
     <model file="ringLight.3d" texture="saturn_ring.png">
        <diffuse R="1" G="1" B="1"/>
      </model>
    </models>
  </group>
  <!-- Uranus -->
  <group>
    <translate time="90.02">
      <point X="0" Y="0" Z="140"/>
      <point X="53.5757" Y="0" Z="129.3431"/>
      <point X="98.9949" Y="0" Z="98.9949"/>
      <point X="129.3431" Y="0" Z="53.5757"/>
      <point X="140" Y="0" Z="0"/>
      <point X="129.3431" Y="0" Z="-53.5757"/>
      <point X="98.9949" Y="0" Z="-98.9949"/>
      <point X="53.5757" Y="0" Z="-129.3431"/>
      <point X="0" Y="0" Z="-140"/>
      <point X="-53.5757" Y="0" Z="-129.3431"/>
      <point X="-98.9949" Y="0" Z="-98.9949"/>
      <point X="-129.3431" Y="0" Z="-53.5757"/>
      <point X="-140" Y="0" Z="0"/>
      <point X="-129.3431" Y="0" Z="53.5757"/>
      <point X="-98.9949" Y="0" Z="98.9949"/>
      <point X="-53.5757" Y="0" Z="129.3431"/>
    </translate>
    <scale X="5" Y="5" Z="5"/>
    <rotate angle="82.23" axisZ="1"/>
    <rotate time="2.34" axisY="1"/>
    <models>
     <model file="sphereLight.3d" texture="uranus.jpg">
        <diffuse R="1" G="1" B="1"/>
      </model>
      <model file="ring2Light.3d" texture="uranus_ring.png">
        <diffuse R="1" G="1" B="1"/>
      </model>
    </models>
  </group>
```

```
    <!-- Neptune -->
    <group>
      <translate time="121.51">
        <point X="0" Y="0" Z="180"/>
        <point X="68.883" Y="0" Z="166.2983"/>
        <point X="127.2792" Y="0" Z="127.2792"/>
        <point X="166.2983" Y="0" Z="68.883"/>
        <point X="180" Y="0" Z="0"/>
        <point X="166.2983" Y="0" Z="-68.883"/>
        <point X="127.2792" Y="0" Z="-127.2792"/>
        <point X="68.883" Y="0" Z="-166.2983"/>
        <point X="0" Y="0" Z="-180"/>
        <point X="-68.883" Y="0" Z="-166.2983"/>
        <point X="-127.2792" Y="0" Z="-127.2792"/>
        <point X="-166.2983" Y="0" Z="-68.883"/>
        <point X="-180" Y="0" Z="0"/>
        <point X="-166.2983" Y="0" Z="68.883"/>
        <point X="-127.2792" Y="0" Z="127.2792"/>
        <point X="-68.883" Y="0" Z="166.2983"/>
      </translate>
      <scale X="5" Y="5" Z="5"/>
      <rotate angle="28.32" axisZ="1"/>
      <rotate time="2.28" axisY="1"/>
      <models>
       <model file="sphereLight.3d" texture="neptune.jpg">
          <diffuse R="1" G="1" B="1"/>
        </model>
      </models>
    </group>
  </group>
</scene>
```

Listing 1: Código XML com parãmetros para o sistema solar