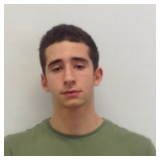
Computação Gráfica
Trabalho Prático Fase II
MIEI
Grupo 27

Gonçalo Pereira

A74413

António Silva



A73827

André Diogo

A75505

31 de Março de 2017

Conteúdo

Capítulo 1

Introdução

Esta segunda fase consiste na leitura e rendering de cenas hierárquicas com translações, rotações e escalas a partir de ficheiros *XML* e de modelos gerados externamente (usar-se-á para esta fase o mesmo formato de modelos utilizado na fase anterior). Para tal é necessário expandir o motor codificado aquando da primeira fase para ler e guardar dados deste tipo de cenas, definindo estruturas de dados úteis para o efeito e algoritmos eficazes para tratar do rendering destes dados.

Nas seguintes seções detalhar-se-á o processo para expandir o motor e demonstrar-se-á o motor em funcionamento com uma cena modelada do sistema solar.

Capítulo 2

Estruturas de Dados

2.1 Parsing de XML

Tomando o formato geral de um ficheiro *XML*, indentifica-se claramente uma estrutura em árvore no que toca às suas etiquetas, sendo que dentro de cada etiqueta é possível um número discreto e potencialmente infinito de etiquetas.

Similarmente, no enunciado do problema, identifica-se também uma estrutura em árvore para a aplicação de translações, rotações e escalas e para o rendering dos modelos. Tem-se etiquetas de grupo que podem conter outras etiquetas de grupo dentro de si e possíveis translações, rotações e escalas aplicadas a cada grupo e aos seus grupos internos recursivamente (modelam relações de parentagem).

Como tal, definiram-se duas classes *SceneTree* e *GroupComponent* que contêm ambos um vetor de *Component*, classe que generaliza os grupos e os modelos e que permite assim recorrência de grupos, criando uma estrutura para a cena em árvore n-ária.

SceneTree é a classe que representa a raíz da cena e que despoleta o parsing do XML e trata de conter o primeiro nível da hierarquia no seu vetor. Ao encontrar uma etiqueta de grupo cria um novo *GroupComponent* que trata de, durante a sua criação retomar o parsing do XML no seu nível da hierarquia e conter as eventuais translação, rotação e escala, assim como os grupos e modelos que aparecem no seu nível hierárquico. Caso encontre uma etiqueta de grupo cria também um novo *GroupComponent* tornando o processo exaustivo e recursivo.

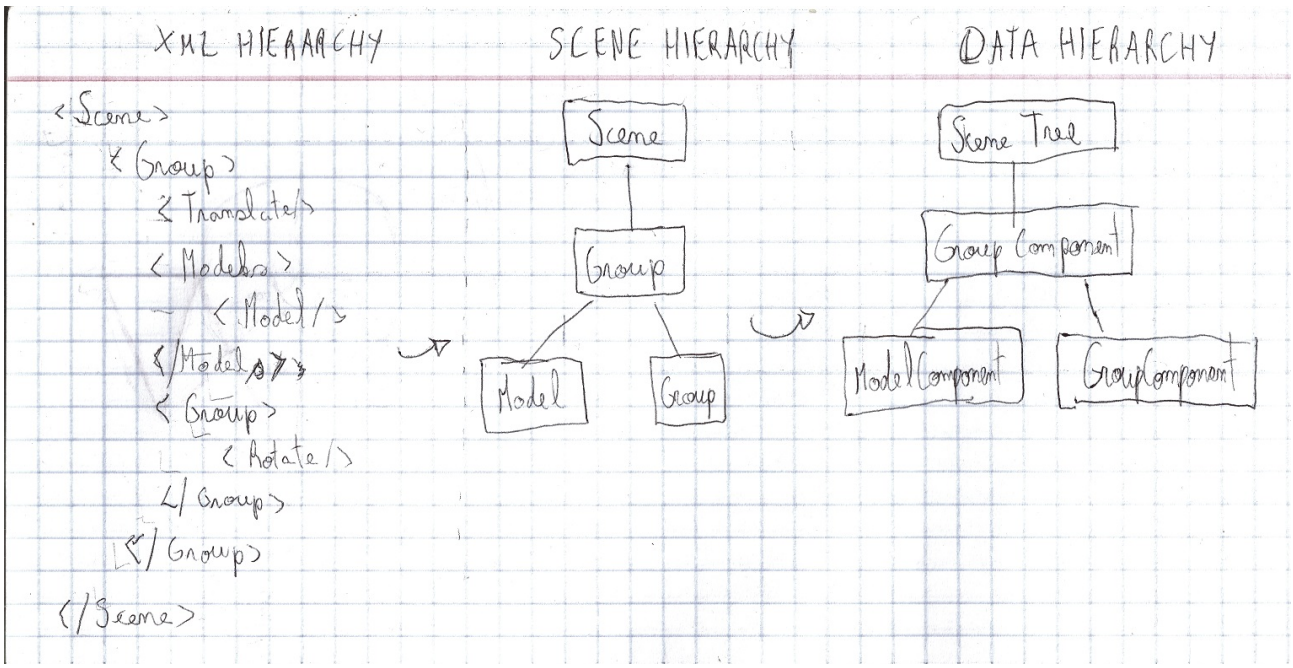


Figura 2.1: Hierarquias identificadas.

2.2 Modelos

Para armazenar os modelos de forma eficiente, construiu-se um dicionário, *modelmap*, onde se guardam apontadores para todos os modelos lidos, de forma única. Quando aparecem repetidos na hierarquia da cena são apenas apontadores para estes guardados no dicionário.

modelmap é então uma variável global que permite duas funcionalidades importantes. É um dicionário que relaciona o caminho no sistema de ficheiros para um modelo com um *ModelComponent*, o outro dos componentes, que guarda o número de vértices requirido para o render desse modelo e um array desses vértices, além de guardar um índice para um *GL buffer* a que está associado para usufruir dos *VBOs*. Permite assim a deduplicação de modelos, isto é, não reler um modelo já lido, assim como fácil incorporação de VBOs (um buffer associado a cada modelo).

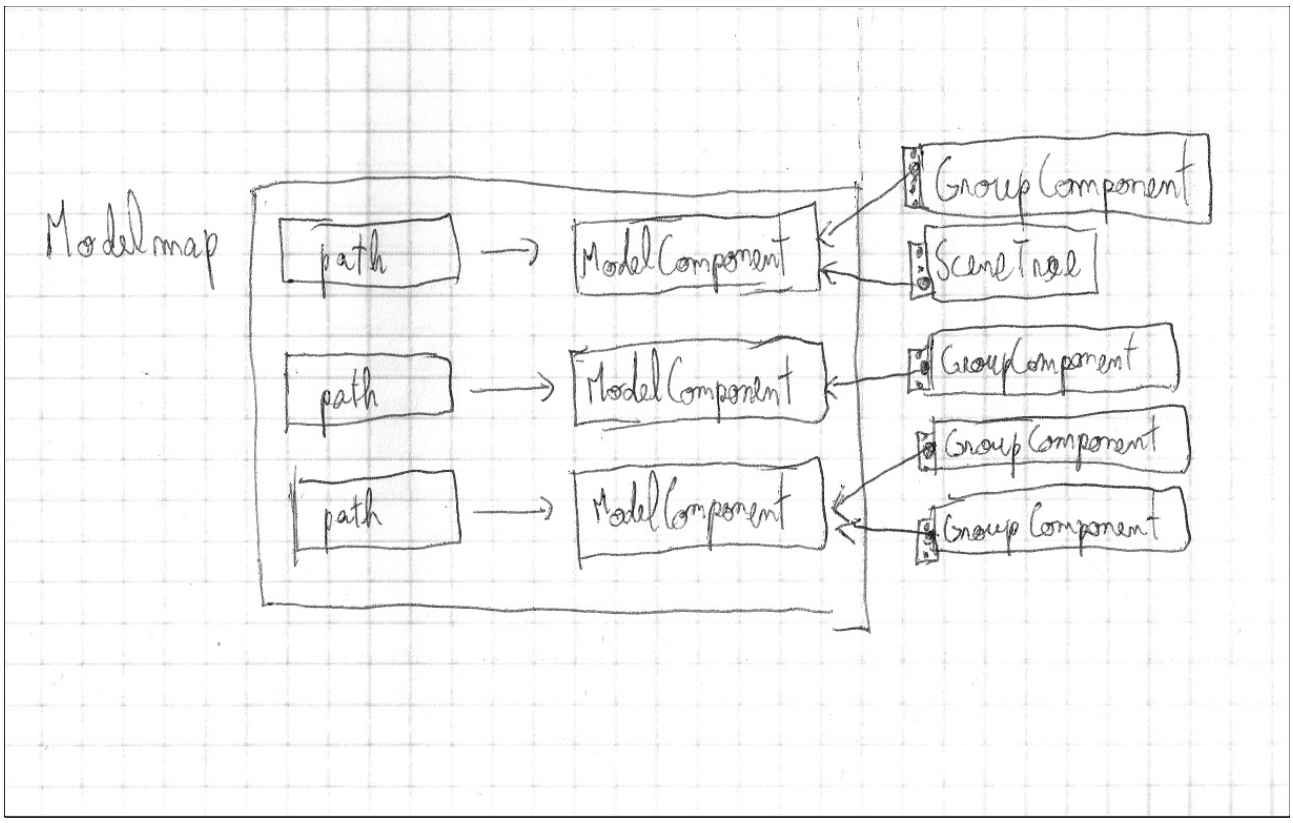


Figura 2.2: *modelmap* e a sua relação com os outros componentes.

2.3 Grupos

Cada *GroupComponent* possui um *Vector3D* (estrutura que armazena coordenadas float 3d) de translação, um de escala e um de rotação (com um angulo de rotação associado em separado) assim como o já referenciado vetor de componentes que são seus filhos e aos quais serão aplicadas estas operações.

Em cada *GroupComponent* associado a um grupo na cena hierárquica podem ocorrer 6 combinações diferentes de ordem de operações aplicadas porque a ordem das translações, rotações e escalas é relevante. Além destas 6 combinações temos ainda todas as restantes permutações de não ter de aplicar qualquer uma destas operações. Para resolver este problema, associa-se a cada operação um valor numa enumeracao e guarda-se um vetor de ordem com tamanho para 3 valores. Temos 4 possíveis valores, uma operação nula (ID), uma translação (TR), uma rotação(RT) e uma escala(SC). Quando se efetua o render basta iterar por este vetor de ordem, comparar cada operação com um dos 4 possíveis valores. No caso de um ser nulo não é preciso continuar a iterar pois as operações que não nulas estarão sempre presentes nas posições anteriores em virtude de o vetor exprimir a ordem.

2.4 Vista geral das estruturas de dados

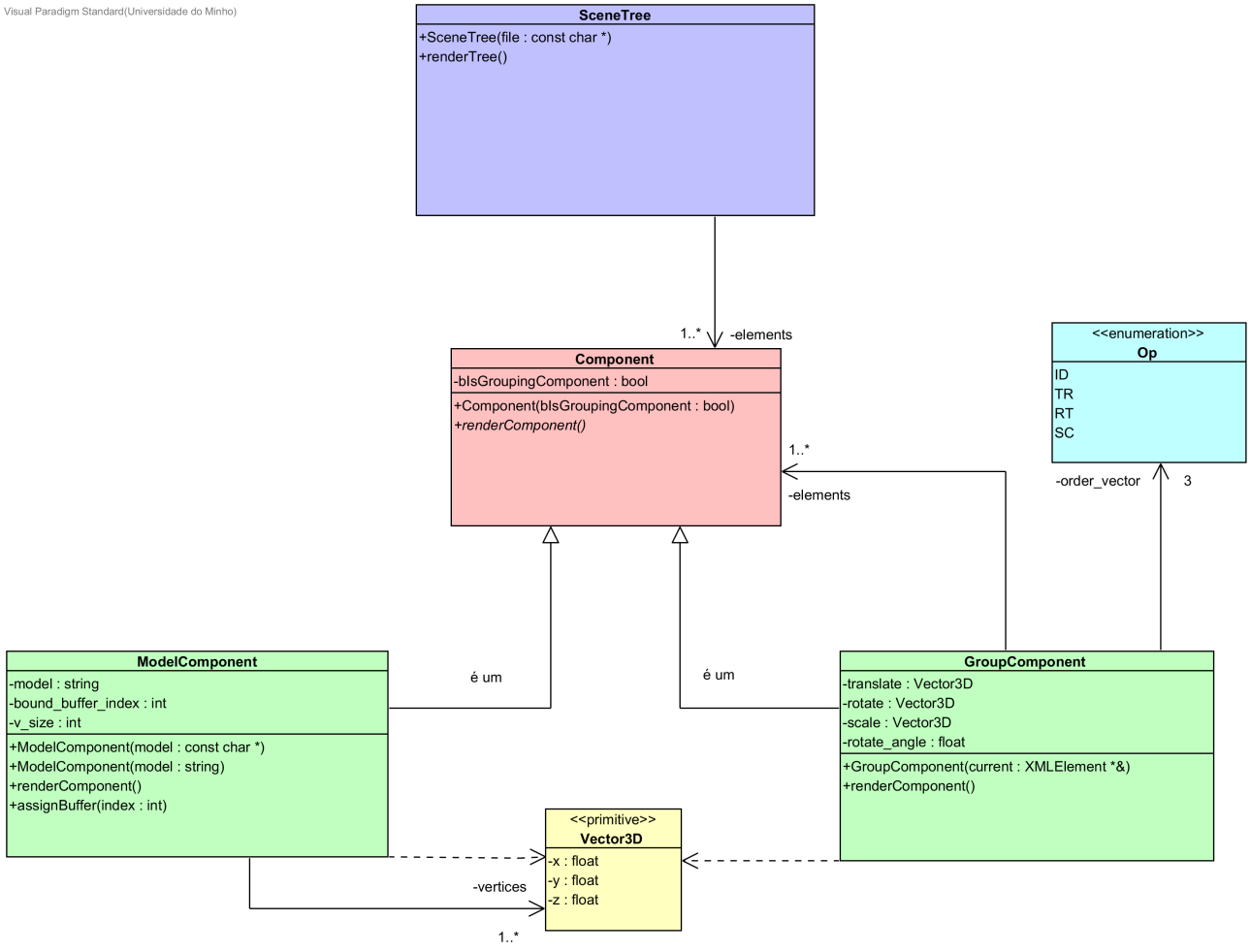


Figura 2.3: Diagrama de classes e estruturas de dados relevantes do motor.

Capítulo 3

Algoritmos

3.1 Parsing de XML

Como o *tinyxml2* não disponibiliza uma forma de saber onde se encontra na hierarquia (profundidade em número de etiquetas abertas) um dado elemento *XML*, começa-se por, na travessia, quer na raiz da cena, quer nos grupos, recorrer ao método *nextSiblingElement()*, que devolve apenas o próximo elemento num dado nível. A travessia dá-se por terminada quando se ultrapassa o último elemento, momento em que o próximo é o *nullptr*. A travessia é feita recursivamente na presença de grupos (está aqui implícita a utilização da stack do programa para manter os dados sobre o nível em que se está a qualquer momento).

```
Algoritmo de travessia XML:

|
|  Se existem filhos para um dado grupo ou para a cena:
|
| |
| |  elem_XML_currente = primeiro_elem_XML filho
| |
| | Enquanto há elem_XML_currente:
| |
| | |
| | | tratar_elemento_XML
| | |
| | | elem_XML_currente = próximo_elem_XML no nível
| | |
| |
|

Algoritmo de tratar_elemento_XML (e tratar_ultimo_elemento_XML)
para a cena (só pode haver grupos ou modelos):

|
|  Se elem_XML_currente é um grupo:
|
| |
| |  cria_novo_grupo
| |
|
| Se elem_XML_currente tem modelos e ainda
| não se trataram modelos:
|
```

```
| |
| | tratar_modelos
| |
|
| Senão:
|
| |
| | imprime erro tag não reconhecida
| |
|
```

Algoritmo de tratar_elemento_XML para um grupo:

```
|
|  Se elem_XML_current é um grupo:
|
| |
| |  cria_novo_grupo
| |
|
| Se elem_XML_current tem modelos e ainda
| não se trataram modelos:
|
| |
| | tratar_modelos
| |
|
| Se elem_XML_current é uma translação e ainda
| não se registou nenhuma:
|
| |
| | regista_translação
| |
|
| Se elem_XML_current é uma rotação e ainda
| não se registou nenhuma:
|
| |
| | regista_rotação
| |
|
| Se elem_XML_current é uma escala e ainda
| não se registou nenhuma:
|
| |
| | regista_escala
| |
|
| Senão:
|
| |
| | imprime erro tag não reconhecida
| |
```

```
|
|

Algoritmo de tratar_modelos:

|
|  Se existem modelos:
|
| |
| |  elem_XML_currente = primeiro_elem_XML_filho
| |
|
| Enquanto há elem_XML_corrente
| e elem_XML_corrente é modelo:
|
| |
| | trata_modelo
| |
| | elem_XML_corrente = próximo_elem_XML no nível
| |
|
| Se o elem_XML_corrente não é modelo:
|
| |
| | erro
| |
|
|
```

3.2 Grupos

Durante o parsing do XML num grupo ocorre o possível registo de uma translação, escala e rotação, sendo a ordem relevante. Para tal, é preciso guardar um contador que mantém quantas das possíveis três já apareceram. Ao registar uma delas, escreve-se para o *array* de ordem a operação corresponde no índice do contador, e de seguida incrementa-se o contador. Este *array* de ordem é inicializado com a operação nula, ID, para todas as suas posições.

```
Algoritmo de regista_translação:

|
| regista coordenadas para o Vetor3D de translação
|
| vetor de ordem, posição do indice atual = TR
|
| incrementa indice atual
|

Algoritmo de regista_rotação:

|
| regista coordenadas para o Vetor3D de rotação
|
| regista o angulo da rotação
|
```

```
| vetor de ordem, posição do indice atual = RT
|
| incrementa indice atual
|

Algoritmo de regista_escala:

|
| regista coordenadas para o Vetor3D de escala
|
| vetor de ordem, posição do indice atual = SC
|
| incrementa indice atual
|
```

3.3 Modelos

Os modelos são lidos a partir dos ficheiros gerados pelo gerador. Quando acabar a fase de leitura do XML e a cena já está construída, geram-se então tantos *GLuint* quanto o número de modelos únicos lidos (tamanho do mapa de modelos). De seguida associam-se a cada um desse modelos um *GL_ARRAY_BUFFER* com o tamanho do modelo lido (número de vértices multiplicado pelo tamanho de abarcar 3 floats).

```
Algoritmo de tratar_modelo

|
| Se o caminho para o modelo existe no mapa de modelos:
|
| |
| | guarda em vetor do grupo ou cena o modelo
| | que está no mapa já lido
| |
|
| Senão:
|
| |
| | cria novo componente de modelo
| | (lê o modelo de ficheiro para memória)
| |
| | insere componente no mapa de modelos
| |
| | Se inserir componente falhar:
| |
| | |
| | | erro de chaves duplicadas
| | |
| |
| | Senão:
| |
| | |
| |  | insere o componente no vetor do grupo ou cena
| | |
| |
|
```

```
Algoritmo de associar buffers a modelos

|
| Para cada modelo em mapa de modelos:
|
| |
| | indice do modelo é o indice atual do ciclo
| |
| | bind do array de indice atual
| | no vetor dos GLuints global
| |
| | definir tamanho e começo com
| | o vetor de vertices lido no modelo
| |
|
```

3.4 Render

Para evitar dispender recursos desnecessários do sistema, apenas se refazem renderizações da cena quando nela ocorrem mudanças expressas, através do *glutPostRedisplay()*.

3.4.1 Cena

Para finalmente desenhar a cena, simplesmente começa-se pelo vetor de componentes na *SceneTree* e aplica-se a cada um o método de render a que respondem de diferentes modos.

3.4.2 Modelos

Para desenhar os modelos, aproveita-se o facto de estarem registados para desenhar com *VBOs* e aplicam-se apenas as instruções glBindBuffer ao indíce do vetor de GLuints global que está guardado para o modelo em causa, glVertexPointer, para desenhar a passo de três *floats* de cada vez e finalmente glDrawArrays no modo de triângulos para o número de vértices do modelo, também nele guardado.

Deixou-se uma opção de pré-processamento desativada para retomar o desenho em modo imediato caso seja desejado.

3.4.3 Grupos

Para desenhar os grupos, aproveita-se o facto de os grupos serem essencialmente iguais à cena, e basta desenhar todos os componentes no vetor de componentes que contém, de igual modo.

No entanto, um grupo pode conter entre uma e três operações geométricas a aplicar a todos os seus filhos. Face a isto é preciso então testar o vetor de ordem, para saber que transformações estão presentes e aplicá-las antes de começar o desenho dos componentes filhos.

Como um grupo aplica operações geométricas, é necessário antes de proceder a desenhar os filhos e fazer a aplicação das operações que os vão afetar, preservar a matriz atual do modo *MODELVIEW* para que os grupos seguintes sejam independentes como esperado. Para este fim, depois de se desenhar os filhos, vai-se buscar de novo esta matriz do topo da stack do openGL (usamos *glPushMatrix() e glPopMatrix()*).

```
Algoritmo para desenhar um grupo:

|
| empurrar a matriz atual para stack do openGL
|
| Desde a primeira à terceira operação
|  mas apenas se não for uma operação nula:
|
| |
| | Se esta operação é uma translação:
| |
| | |
| | | aplica translacao com coordenadas lidas
| | |
| |
| | Se esta operação é uma rotação:
| |
| | |
| | | aplica rotacao com angulo e coordenadas lidas
| | |
| |
| | Se esta operação é uma escala:
| |
| | |
| | | aplica escala com coordenadas lidas
| | |
| |
|
| Para cada filho do grupo:
|
| |
| | pede ao filho que se desenhe
| |
|
| tira a matriz que pôs anteriormente da stack do openGL
|
```

3.5 Câmaras

3.5.1 Primeira Pessoa

Para a câmara de primeira pessoa, para facilitar os cálculos, fixou-se a coordenada Y na origem, e permite-se apenas a movimentação do ângulo de visão e da posição da câmara nos eixos xx e zz. Como o sistema solar de demonstração é desenhado estáticamente sobre a origem na coordenada Y, permite a sua exploração.

A implementação desta câmara foi feita de acordo com o que foi leccionado recentemente.

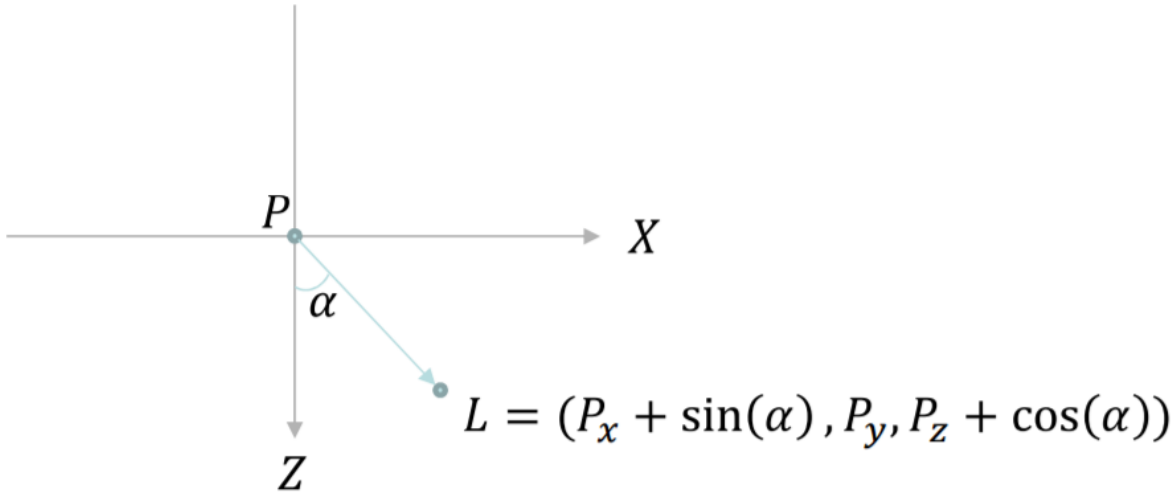
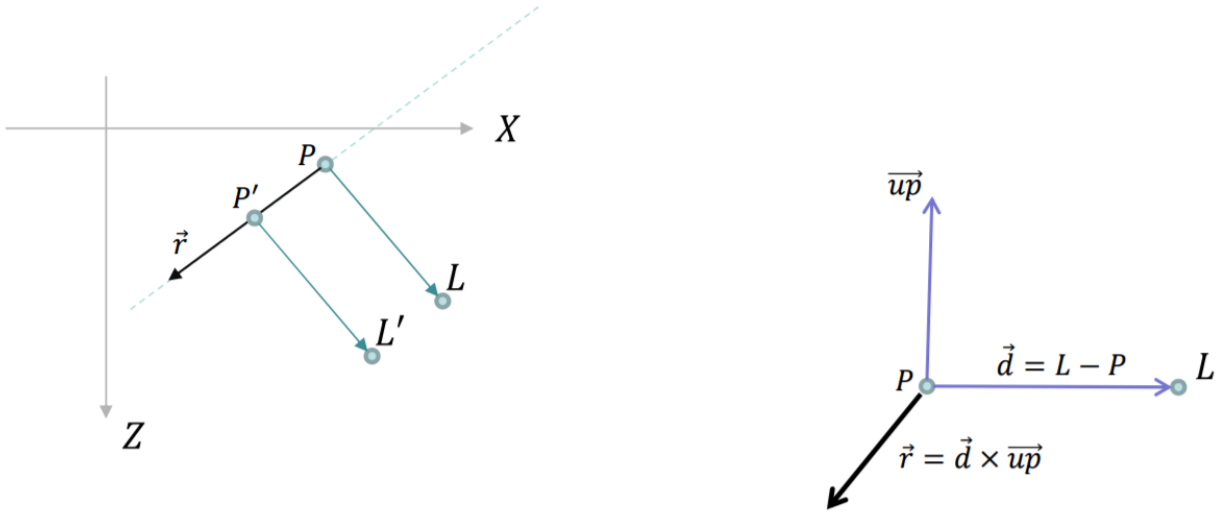


Figura 3.1: Cálculo da direção.

$$\vec{d} = L - P = (L_x - P_x, 0, L_z - P_z)$$
$$P' = P + k\vec{d}$$
$$L' = L + k\vec{d}$$

Figura 3.2: Cálculo da movimentação em linha com a direção da câmara.



$$P' = P + k\vec{r}$$
$$L' = L + k\vec{r}$$

Figura 3.3: Cálculo da movimentação na lateral.

3.5.2 Fixa a olhar o Sol

Esta câmara foi clonada a partir da utilizada na aula prática de desenho dos cowboys e índios. É controlada pelo rato.

Esta câmara utiliza coordenadas esféricas, pelo que o cálculo das coordenadas X,Y e Z varia com a variação de um ângulo alfa (responsável pela movimentação nos eixos do xx e zz ao longo da superfície esférica) e do ângulo beta (que controla a variação na superfície esférica no eixo dos yy.) e do raio, caso se use o segundo botão do rato. As variações a aplicar ao ângulo alfa e beta são extraídas do movimento no eixo xx e yy do rato na janela projetada, respetivamente.

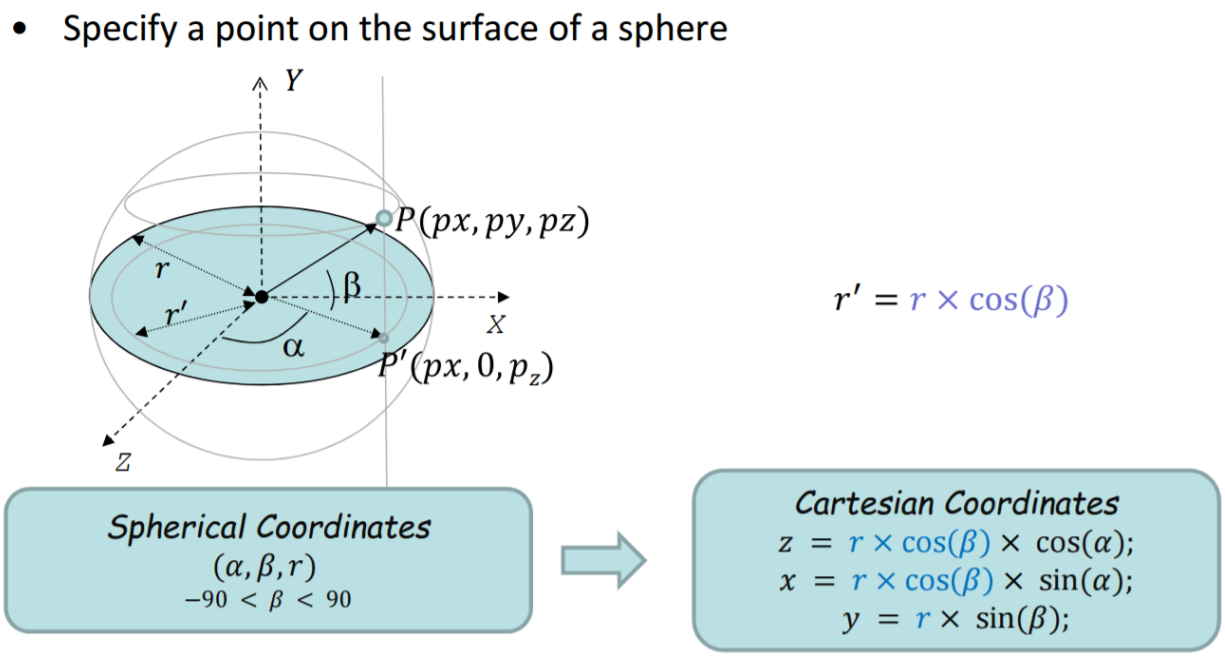


Figura 3.4: Equações de determinação de pontos numa superfícies esférica

.

3.5.3 Alternância entre as câmaras

Para alternar entre as câmaras definiu-se uma variável que diz qual a câmara atual, que é comparada quando se atualiza estados das câmaras para não se alterar o estado duma enquanto não está ativa. Utilizou-se o F1 e F2 para mudar ativar uma ou a outra, saltando-se instantaneamente de uma para a outra.

3.6 Gerador de Discos

Para permitir desenhar discos quer à volta dos planetas que têm anéis, quer para possivelmente representar órbitas, criou-se um gerador de discos planos. O disco consiste numa coroa circular com um dado número de cortes, desenhado como duas coroas circulares, uma com os triângulos virados para o eixo positivo perpendicular e a outra para o eixo negativo perpendicular, de modo a que se veja de todas as perspetivas, mesmo com o *culling* de faces traseiras ligado.

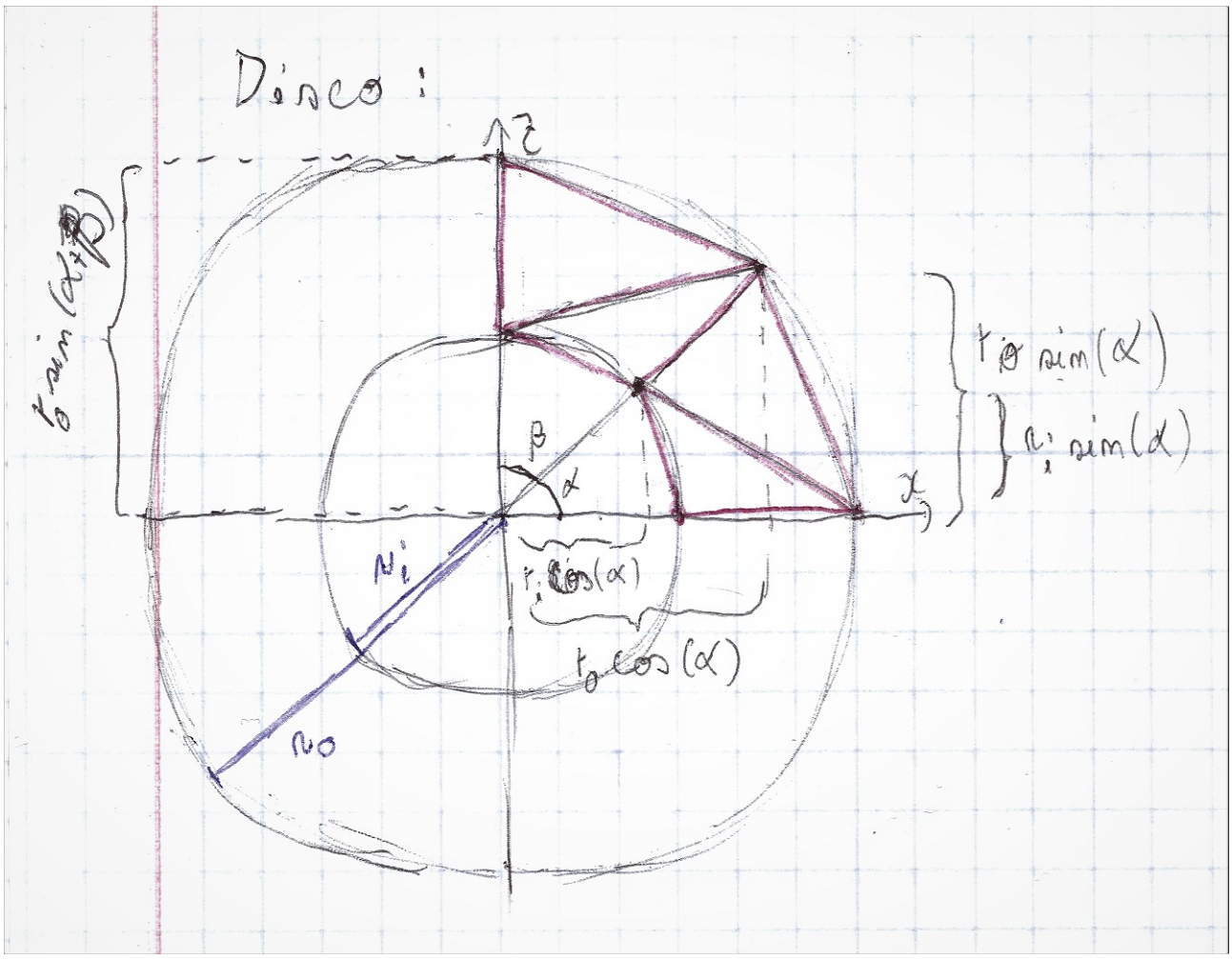


Figura 3.5: Lógica e equações por trás da geração do disco.

Capítulo 4

Modelo do sistema solar

Para a criação do modelo do sistema solar, utilizou-se valores reais das distâncias e diâmetros dos planetas, que se foi buscar a um site sobre factos astronómicos (`http://www.enchantedlearning.com/subjects/astronomy/planets/`).

No entanto, visto que as distâncias entre planetas e o Sol são na ordem das centenas de milhoes de km, enquanto os planetas são infimamente pequenos em comparação, na ordem dos milhares de km de raio, decidiou-se aplicar uma escala de 3:20000000 para as distâncias, e de 1:100000 para os tamanhos dos astros, respeitando-se assim a ordem topográfica, mas de forma a permitir que se visualizasse os planetas facilmente a partir duma câmara (com distâncias tão elevadas em relação ao tamanho dos astros seria perfeitamente incomportável visualizar seja o que for, se feito completamente à escala).

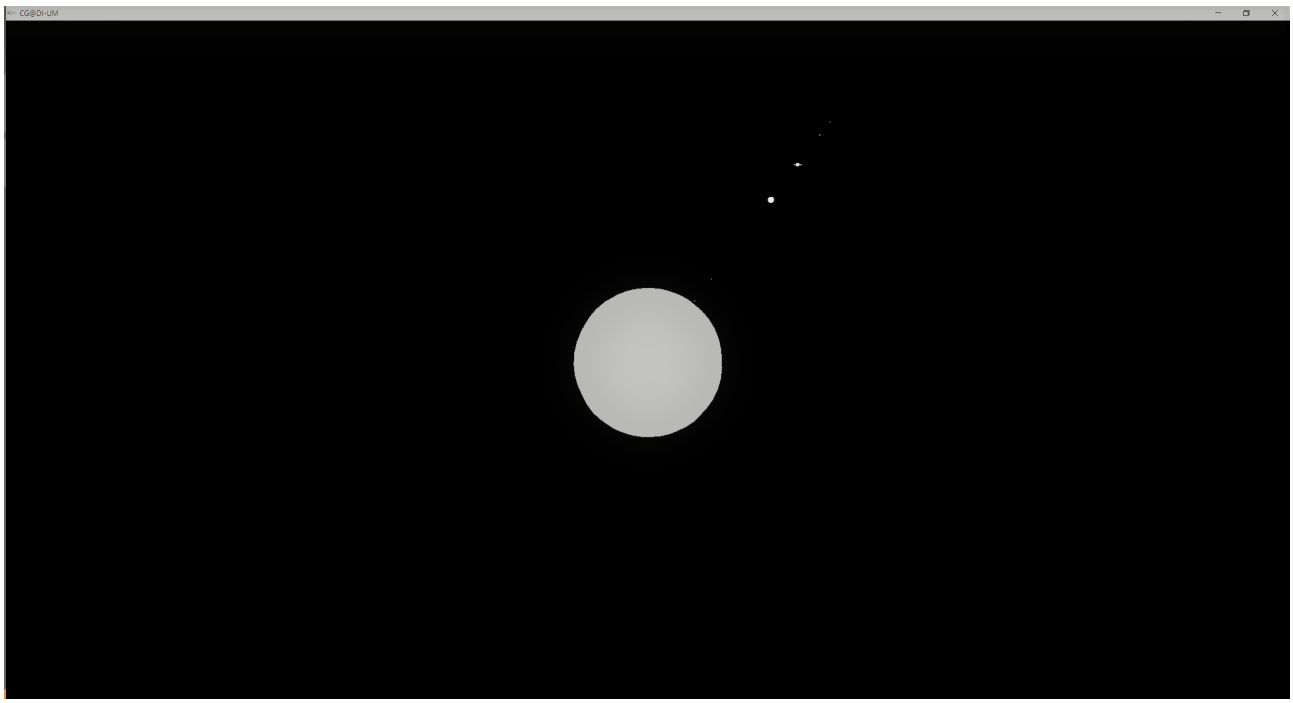


Figura 4.1: Demonstração do Sistema solar.