Universidade do Minho

Departamento de Informática

Computação Gráfica

Grupo 32

2ªFase



Afonso Magalhães- A95250 Artur Leite- A97027 Diana Teixeira- A97516

Índice

1. Introdução e Objetivos

Com o Generator e o Engine já feitos na 1º fase, passamos agora para a fase 2, onde iremos modificar o Engine, criando cenas hierárquicas usando transformações geométricas, como é o caso das translações, rotações e escalas.

Sendo então o objetivo desta fase, seguir os exemplos fornecidos no enunciado, e criar uma cena, onde cada nó contém um conjunto de transformações geométricas e, opcionalmente, um conjunto de modelos e até nós filhos.

A cena em questão trata-se de uma representação do Sistema Solar. Esta pode ser obtida com o auxílio de um *script* escrito em *Python,* que irá incluir o Sol, os planetas e os respetivos satélites naturais correspondentes.

2. Engine

2.1. Transformações

Com o intuito de garantir uma boa implementação das diferentes possíveis transformações que iremos necessitar nesta fase, optamos por criar uma interface denominada de Transform, que irá disponibilizar o método *apply*. Em seguida, três classes (Rotation, Translation e Scale) irão implementar essa interface, definindo o método *apply* para a sua respetiva transformação. De notar que, como a ordem em que estas transformações ocorrem é relevante, decidimos colecioná-las ordenadamente para que o interpretador XML as processe de ordem correta. Para tal, a nossa classe Group terá um vetor de Transforms, juntamente com um vetor de *groups* que corresponde aos seus filhos. Finalmente, o método *draw* cria uma matriz através da função *glPushMatrix,* onde serão aplicadas cada uma das transformações na matriz em questão, de seguida desenhará todos os seus modelos, bem como os seus grupos filho, destruindo a sua matriz com a função *glPopMatrix.* De realçar a importância de realizar este último passo apenas depois de desenhar todos os grupos, de forma que as suas transformações também sejam aplicadas neles.

2.2. Camera

Nesta fase, optamos por implementar a movimentação da camara através do input do utilizador. Tendo, portanto, finalizado o modo explorador e começado por definir o modo FPS, não estando finalizado. Para isso, foi necessário adicionar três importantes variáveis à classe Camera, sendo estes o α, ângulo formado entre os eixos Z e X, o β, ângulo formado entre o plano X0Z e o eixo Y, e o raio, que se trata da distância entre a origem do referencial e o local da camera. O cálculo destas variáveis será feito aquando do arranque do programa, onde estão fixados o ponto incial da Camera e a origem do referencial.

Estes ângulos α e β irão ser manipulados em função do input que o utilizador fornece. Caso o utilizador queira rodar para a direita, o valor de α será incrementado, sendo que para o outro sentido, o processo é inverso. Caso o utilizador queira movimentar a camera para cima ou para baixo, o ângulo β será incrementado ou decrementado, respetivamente.

A posição da camera será calculada através das seguintes expressões:

(raio*cos (β)* sen(α); raio* sen(β); raio*cos (β)* cos(α))=> (Px;Py;Pz)

2.3 XML

Tendo em conta que o *parser* já tinha sido desenvolvido na primeira fase do projeto tendo em vista a hierarquia do XML, necessitávamos apenas de adicionar a possibilidade de interpretação de transformações geométricas. Para tal, adicionamos uma função capaz de executar o *parsing* em grupo.

3. Sistema

3.1. Script

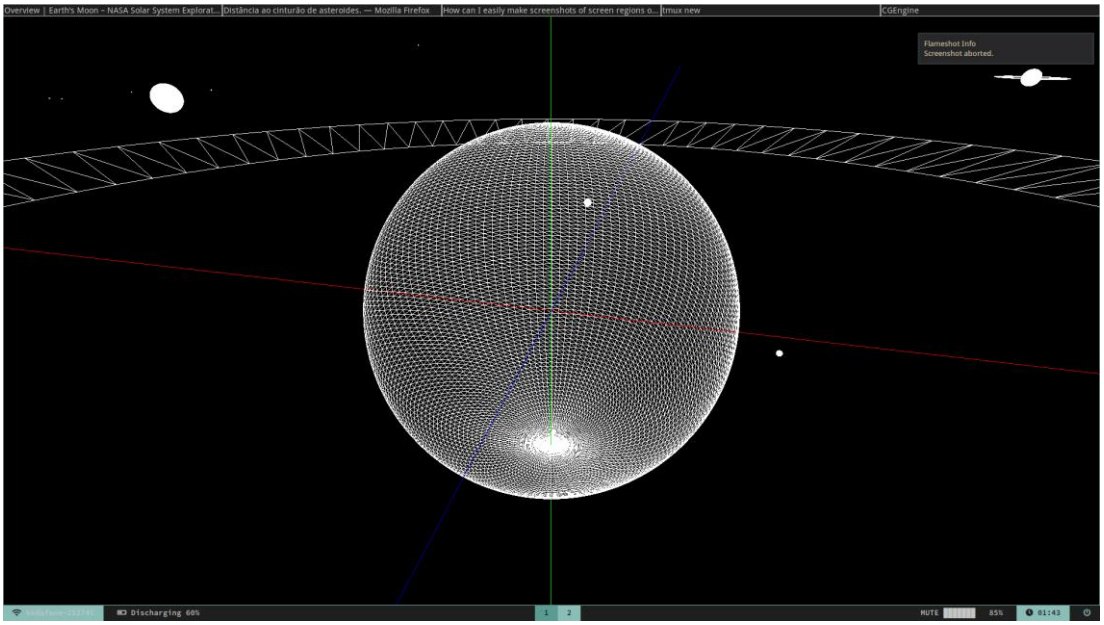
Com o intuito de automatizar o processo da criação do ficheiro XML, utilizamos uma script em Python com os respetivos grupos e modelos necessários para o Sistema Solar. Para garantir que seria proporcionada uma visualização dos elementos do modelo organizada, aplicamos uma escala, tanto na dimensão dos próprios astros, como também na distância entre eles. Desta forma, com recurso a algumas *libraries* de *Python,* foi possível a indentação típica de um ficheiro XML.

Com o intuito de aproximar a escala do nosso modelo às dimensões reais, começamos por atribuir um valor ao raio da esfera do sol. Vamos denotar este valor como Ω. Após isso, aferimos a dimensão de alguns astros em relação ao sol. Por exemplo, a largura da terra é 109 vezes inferior ao sol. Multiplicamos então o raio Ω por esse rácio, atribuindo o resultado ao raio da terra. De seguida, multiplicamos este valor por 4 para garantir que estes astros sejam visíveis no modelo.

Adicionalmente, optamos por incluir os satélites naturais e os planetas no mesmo *group.* Desta forma, conseguimos simular o movimento conjunto que os planetas partilham com as suas luas. Sejam estas translações, rotações, etc... Garantindo sempre a órbita natural.

Para representar os aneís de Saturno e Neptuno, implementamos uma class r*ing,* que irá inserir um Torus com uma altura nula por volta destes planetas. Garantindo também, tal como mencionamos anteriormente, que estejam agrupados de forma a realizar as mesmas transformações do respetivo planeta. Utilizamos esta estrutura ainda para representar o anel de asteróides entre Marte e Júpiter.

Para facilitar a visibilidade das Luas e dos Planetas, aplicamos uma rotação sobre o eixo dos Ys, de forma que estes não se encontrem todos em linha, sendo mais fácil visualizá-los.



4. Conclusão

Tendo em conta o trabalho que realizamos, achamos que fomos capazes de ir de acordo com os requisitos presentes no enunciado. Acreditamos ainda que aproximamos de forma relativamente semelhante à escala real do Sistema Solar. Apesar disso, encontramos algumas dificuldades no que toca a definir as distâncias entre os astros que compõem o nosso modelo. Temos ainda intenção de melhorar o desempenho do nosso *engine* no futuro, de forma a otimizar o processo de visualização.