Universidade do Minho

3º ano do Mestrado Integrado em Engenharia Informática

Computação Gráfica

Ano lectivo 2016/2017

Fase 2 - Transformações Geométricas

Autores:
Diana Oliveira (a67652)
Manuel Moreno (a67713)
Xavier Francisco (a67725)

Braga, 2 de Abril de 2017

Conteúdo

1 Introducao

Continua-se o desenvolvimento de um motor de renderização. Nesta etapa, permitimos a descrição de um cenário hierárquico.

Duas outras features que também adicinamos ao projecto foi a possibilidade de renderizar ficheiros .stl e de movimentar para alêm do método orbital, também como um FPS (como DOOM, só movimenta paralelos aos eixos).

Neste documento estão contidos todos os passos relevantes e decisões tomadas para o desenvolvimento deste projeto.

2 Introdução ao problema

O objetivo desta fase é criar cenas hierárquicas usando transformações geométricas. Um cenário é definida como uma árvore onde cada nodo contém um conjunto de tranformações geométricas como a translação, a rotação e a escala. Cada nodo pode ter também nodos filhos. A cena demo para esta fase é um modelo estático do sistema solar, incluindo o sol, os planetas e as luas definidas na hierarquia.

2.1 Projecção da solução

Para cumprir os objctos desta fase, óptamos por melhor estruturar o nosso projecto. Criamos uma classe que guardará o estado global do nosso motor. Este estado seria composto por uma sequência de comandos (derivada do cenário representado no ficheiro `XML`) e por uma câmera.

Dividimos os comandos necessários para a renderização de um cenário nos seguintes:

- Translação (glTranslatef)
- Rotate (glRotatef)
- Scale (glScalef)
- DrawModel (glVertex3f*)
- EnterContext (glPushMatrix)
- ExitContext (glPopMatrix)

A sequência de comandos é dada por uma travesia *depth-first inorder* da árvore scene no XML. Esta traversia tem a propriedade de returnar a mesma sequência pela qual eles aparecem no ficheiro.

Assume-se que em cada grupo a sequêmcia de elementos comece com as suas transformações geométricas (translação, rotação, scale).

3 Leitura e análise de XML

Como sabemos, é a partir do ficheiro `XML` que o cenário será lido, pois lá contém os elementos a serem desenhadas e a suas respectivas transformações.

Como referimos anteriormente, a nossa representação do cenário é a sequência dada pela traversia inorder da árvore grupos.

A ordem dessa sequência é a mesma ordem pela qual aparece no ficheiro. Deste principio o nosso método consiste em transformar os elementos, à medida que vamos lendo, nos respectivos comandos e adicionando-o ao estado.

O parser é um parser descendente recursivo, escrito sem bibliotecas e implementa a seguinte gramática

```
scene <- elemento*
elemento <- '<' word (sep word)* '>'
word <- [a-zA-Z]* | '"' word '"'
sep <- '=' | ' '
```

Ao encontrar um elemento completo, o parser, lê o elemeento e adiciona o comando correspondente (se houver algum). Para o comando DrawModel, é também necessário ler o ficheiro correspondente, para guardar no próprio comando. É de notar que ainda existe redudância nos ficheiros guardados. Sempre que o motor encontra um `model` elemento, ele lê o ficheiro que o elemento especifica, mesmo se já tiver sido referido.

4 Comandos

A classe abstracta comando define a implementação mínima de um comando. O método que cada comando tem que implementar é o `apply`, que chama as funções OpenGL necessárias para aplicar o comando.

4.1 Translate

A translação é o movimento que um objeto realiza de um ponto para outro.

4.1.1 Fundamentos

Figura 1: Tranlação

Pode ser vista num sistema de equações:

$$\begin{cases} x' = x + \Delta x \\ y' = y + \Delta y \\ z' = z + \Delta z \end{cases}$$

Ou então como uma matriz:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

4.1.2 Implementação

A estrutura tem três variáveis de instância: X, Y, Z. O método `apply` simplesmente chama o `glTranslatef` com os argumentos correspondentes.

4.2 Rotation

4.2.1 Fundamentos

A rotação é o movimento circular de um objeto ao redor de um centro ou ponto de rotação. Dado um ponto (x,y,z) e um ângulo θ a sua rotação sobre o eixo Y pode ser visto da seguinte forma:

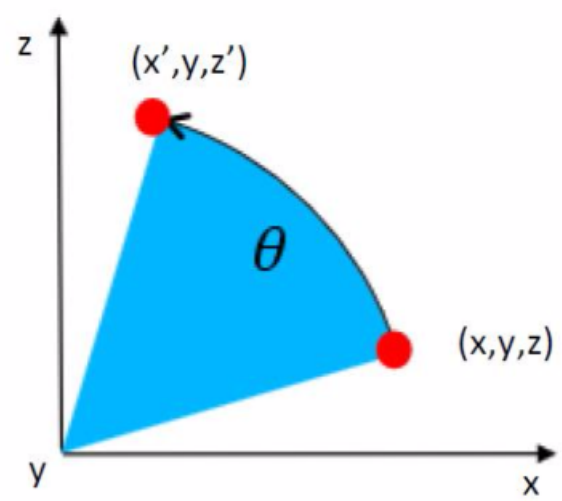

Figura 2: Rotação de um ponto

Pode ser vista num sistema de equações:

$$\begin{cases} x' = x\cos\theta + y\sin\theta \\ y' = y \\ z' = -x\sin\theta + y\cos\theta \end{cases}$$

Ou então como uma matriz:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

4.2.2 Implementação

A estrutura tem quatro variáveis de instância: angle, axisX, axisY, axisZ. O método `apply` simplesmente chama o `glRotatef` com os argumentos correspondentes.

4.3 Scale

4.3.1 Fundamentos

As escalas permitem variar o tamanho de um objeto multiplicando cada ponto por um escalar.

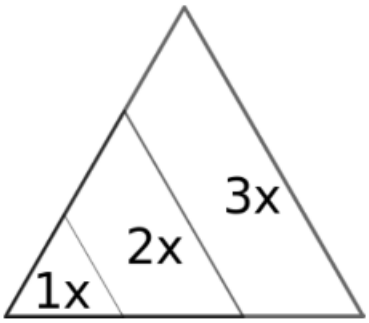

Figura 3: Escala de uma imagem

Pode ser vista num sistema de equações:

$$\begin{cases} x' = \lambda_x x \\ y' = \lambda_y y \\ z' = \lambda_z z \end{cases}$$

Ou então como uma matriz:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda_x & 0 & 0 & 0 \\ 0 & \lambda_y & 0 & 0 \\ 0 & 0 & \lambda_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

4.3.2 Implementação

A estrutura tem três variáveis de instância: X, Y, Z. O método `apply` simplesmente chama o `glScalef` com os argumentos correspondentes.

4.4 DrawModel

4.4.1 Implementação

A estrutura tem apenas uma variável de instância: um vector com todos os pontos. Como o nosso motor ainda não suporta `VBO's`, o método `apply` chama glVertex a todos os pontos guardados.

4.5 EnterContext e ExitContext

4.5.1 Fundamentos

O glPushMatrix e o glPopMatrix são operações que nos ajudam a renderizar um cenário hieárquico como é o nosso caso. Isso permite-nos, renderizar um objecto (filho) em relação a outro, sem conhecimento da posição e orientação do pai.

4.5.2 Implementação

Os dois comandos chamam, respectivamente, glPushMatrix e glPopMatrix.

4.6 Descrição do ciclo de rendering

O ciclo de rendering tem apenas duas funções principais. Mover a câmera de acordo com as teclas pressionadas, com a função move da câmera. Aplicar sequencialmente os comandos com o método `applyCommands`, pertencente ao estado.

5 Conclusão e Trabalho Futuro

Nesta fase do projeto houve uma mudança de maneira de lidar mais com o C++ em relação à fase anterior.
Consideramos que cumprimos os requisitos para esta fase.