Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática

Computação Gráfica

Geometric Transforms

Luís Capa a81960
Moisés Antunes a82263
Pedro Capa a83170
Vítor Gomes a75362

Conteúdo

1 Introdução

O trabalho da Unidade Curricular de Computação Gráfica está dividido em 4 partes, sendo este relatório referente à 2ª fase. O objetivo desta fase era apenas desenvolver o programa engine realizado na 1ª fase. Enquanto na 1ª fase o engine apenas desenhava as figuras geométricas, nesta fase estão associadas transformações às figuras.

2 Engine

O motor, ou "engine", recebe como argumento um ficheiro de configuração, escrito em *XML*, após os vértices do modelo serem gerados. Por agora, este ficheiro *XML* contém apenas a indicação de ficheiros previamente gerados pelo programa descrito anteriormente, o *generator*, e uma série de transformações geométricas associadas a cada ficheiro. Estes ficheiros contêm os pontos dos triângulos que serão desenhados com auxilio da biblioteca *GLUT* e da biblioteca *Glew*. Este programa, depois de recolher os vértices e as transformações geométricas aplica as transformações aos vértices correspondentes e são desenhados.

2.1 Estruturas de Dados

Para que o programa fosse mais fácil, quer no seu desenvolvimento, quer na sua compreensão foram criadas algumas estruturas de dados.

2.1.1 Transformacao

Foi criada uma estrutura de dados abstrata que englobava todas as transformações geométricas. Esta classe tinha as variáveis:

- **float x**; que guarda o valor da transformação no eixo dos x.
- **float y**; que guarda o valor da transformação no eixo dos y.
- **float z**; que guarda o valor da transformação no eixo dos z.

2.1.2 Scale

Esta estrutura de dados não tinha variáveis definidas uma vez que todas as variáveis necessárias estavam definidas na super classe *Transformacao*. Como era subclasse de *Transformacao* era obrigada a definir o método *aplicaTransformacao*. Este método apenas invoca a função $glScalef(x, y, z)$

2.1.3 Translacao

Esta estrutura de dados não apresenta variáveis definidas pelo mesmo motivo da estrutura anterior, no entanto o método *aplicaTransformacao* varia, pois invoca a função $glTranslate(x, y, z)$.

2.1.4 Rotacao

Além de apresentar as variáveis da super classe *Transformacao*, esta tem a variável:

- **float angle**; que guarda o valor do ângulo da rotação.

O método *aplicaTransformacao* chama a função de rotação do *glut*, $glRotatef(angle, x, y, z)$.

2.1.5 Grupo

Esta estrutura de dados tem como variáveis:

- $vector < Transformacao >$ **trans**; que guarda todas as transformações geométricas associadas ao grupo.
- $vector < float >$ **pontos**; que guarda todos os pontos que estão nos ficheiros associados ao *group*. Esta variável apenas fará a *"ponte"* para a matriz que desenha os pontos utilizando VBO's.
- $vector < Grupo >$ **grupos**; que guarda todos os *grupo* filhos. É necessário ter esta variável uma vez que as transformações geométricas associadas a este *group* também são associadas aos filhos. Desta forma é mais eficiente uma vez que não se tem que guardar informação repetida.
- *GLuint* **buffer**; Este buffer é um apontador para uma determinada posição de um array de *GLuint* que vai ser usado para desenhar os *pontos* deste grupo.

2.2 Leitura do ficheiro xml

Para a leitura do documento foi usada a biblioteca *tinyxml*2, devido aos métodos já implementados para processar este tipo de ficheiro. Este ficheiro tinha os nodos *scene* que marcam o inicio e o fim do documento, o *group* e este poderia conter *group* filhos, transformações geométricas e/ou modelos. O nodo *scale*, *translate* e o *rotate* correspondem às transformações geométricas. Os nodos *models* contêm nodos *model* que guardam o nome de um ficheiro previamente gerado pelo *generator*.

Neste processamento vai ser usado um vector *Grupo*, que contem as transformações do nodo *group*, os vértices que estavam no ficheiro correspondente a esse *group* e *group* filhos.

2.2.1 Parser da função tratagroup

Como todos os nodos, à exceção do *scene*, estão dentro de um *group* então foi decidido criar uma função que trata do conteúdo de um *group*. Nesta função iria ter uma variável auxiliar que é um *vector* < *Transformacao* > que contém todas as transformações geométricas do *group* atual. Dentro do *group* vão ser verificados todos os nodos que estejam dentro do mesmo. No caso do nodo ser *rotate*, *translate* ou *scale* este será adicionado ao *vector* < *Transformacao* >. Caso seja um model o programa vai obter todos os pontos dentro desses ficheiros e vai adicionar os pontos ao *vector* < *float* > *pontos*. Caso seja um *group*, chama-se recursivamente a função. O resultado da função é um *group* que será adicionado ao *vector* < *Grupo* > do *group* atual.

2.2.2 Exemplo

Para exemplificar o processo vai ser usado o ficheiro da figura 1. Inicialmente é criado o *vector* < *Grupo* > vazio. Ao entrar no primeiro nodo *group*, é chamada a função *tratagroup*. Neste momento é criada uma variável temporária denominada de *atual*, que é um *vector* < *Transformacao* > vazio. Como o próximo nodo é um *rotate*, então vai ser adicionada ao vetor *atual*. Como o próximo nodo é um *models*, é criado um *vector* < *float* > que vai guardar os pontos dos ficheiros *sphere*.3*d* e *plane*.3*d*. Como o nodo a seguir é um *group*, então vai ser, chamada a função *tratagroup*. São adicionados as duas transformações à variável *atual* do grupo filho e processado o ficheiro *sphere*.3*d*. Como o nodo a seguir é *null*, então a função acaba e regressa-se à função que a invocou. O *group* filho é adicionado aos filhos do *grupo* pai. A seguir, é processado o seguinte *group* com processo descrito anteriormente para o *group* filho. Todo o processo é repetido para o *group* seguinte, o que tem o file "../cone.3d".

```
<scene>
    <group>
        <rotate angle="90" X="1" Y="1" Z="0" />
        <models>
            <model file="../sphere.3d" />
            <model file="../plane.3d" />
        </models>
        <group>
            <rotate angle="90" X="1" Y="1" Z="0" />
            <scale X="12" Y="20" Z="2" />
            <models>
                <model file="../sphere.3d" />
            </models>
        </group>
        <group>
            <translate X="2" Y="2" Z="2" />
            <rotate angle="90" X="1" Y="0" Z="1" />
            <scale X="2" Y="2" Z="2" />
            <models>
                <model file="../plane.3d" />
            </models>
        </group>
    </group>
    <group>
        <translate X="15" Y="0" Z ="10" />
        <models>
                <model file="../cone.3d" />
            </models>
    </group>
</scene>
```

Figura 1: Exemplo de um ficheiro xml

3 Desenho com VBO

Após a extração de todos os dados, foram inicializados os buffers usados nos VBO e só depois, é que as figuras foram desenhadas. A fase de inicialização do VBO baseia-se nos seguintes passo:

- inicialização dos VBO com a função *GLEnableClientState*(*GL_VERTEX_ARRAY*)
- declaração da variável $GLuint * buffers$, como uma variável global, que

mais tarde foi alocada memória, para que houvesse um buffer para cada grupo sem desperdiçar memória

- inicialização do $buffers$ com a função $glGenBuffers(tam, buffers)$ em que tam é o número de grupos previamente calculado
- $glBindBuffer(GL\_ARRAY\_BUFFER, buffers[j])$;, associa um vetor de float $pontos$ à posição j de $buffers$
- $glBufferData(GL\_ARRAY\_BUFFER, points.size()*4, points.front(), GL\_STATIC\_DRAW)$, insere todos os valores de $pontos$ na posição j do $buffers$
- $group->setBuffer(buffers[j])$ insere o apontador da posição j de $buffers$, na variável de instância $buffer$ do $group$

Para desenhar as figuras, na função *renderScene*, para todos os grupos era aplicado o método *draw*. O método *draw* pode ser dividido nos seguintes passos:

- $glPushMatrix()$ para guardar na stack a última configuração
- para todas as $Transformacao$ em $trans$ aplica o método $aplicaTransformacao$
- para desenhar os vértices do $buffer$ do grupo são usados as seguintes funções:
  - $glBindBuffer(GL\_ARRAY\_BUFFER, buffer)$
  - $glVertexPointer(3, GL\_FLOAT, 0, 0)$
  - $glDrawArrays(GL\_TRIANGLES, 0, pontos.size()/3)$
- para todos os grupos "filhos"em $grupos$ é aplicada a função $draw$ recursivamente
- $glPopMatrix()$ voltar à configuração que estava no topo da stack

Desta forma, as transformações geométricas eram apenas aplicadas uma vez, ou seja, uma transformação quando era aplicada a um grupo, todos os filhos desse grupo também sofriam essa transformação, sem ser necessário aplicar novamente essas transformações.

4 Sistema Solar

Como era necessário um exemplo que demonstrasse o funcionamento do nosso engine com uma hierarquia que obrigasse a fazer uso de translações, rotações e grupos "filho", foi pedido que se criasse um modelo estático do sistema solar com o Sol, os oito planetas principais e as suas luas. Para isso, foi criado um grupo cujos filhos eram os grupos com o Sol e os planetas principais, sendo que de cada vez que se quisesse adicionar uma lua, criava-se um grupo filho para o grupo do planeta correspondente a essa lua.

```
    </group>
    <group>         <!-- TERRA -->
        <translate X="45"     />
        <group>         <!-- TERRA -->
            <rotate angle="23" axisX="1" axisY="0" axisZ="0" />
            <scale X="2" Y="2" Z="2" />
            <models>
                <model file="../sphere.3d" />
            </models>
        </group>
        <group>         <!-- LUA -->
            <translate Z="4" />
            <rotate angle="5" axisX="1" axisY="0" axisZ="0" />
            <scale X="0.5" Y="0.5" Z="0.5" />
            <models>
                <model file="../sphere.3d" />
            </models>
        </group>
    </group>
```

Figura 2: Código XML da Terra e da Lua na representação do Sistema Solar

O Sol assumia o centro do plano e era aplicado a todos os outros planetas um *translate* que movia esse planeta para uma distância relativa (se bem que não completamente à escala) à sua distância real ao Sol, de maneira a que os planetas ficassem dispostos pela ordem correta. As luas, que já tinham sido afetadas pela translação que também afetou os seus respetivos planetas (e que era realizada em ordem ao eixo X), sofreram outra translação, desta vez em ordem a Z, de maneira a que estas não fossem confundidas com planetas principais mas a sua posição não coincidisse com a de algum planeta.

Foi executada também um *rotate* em cada corpo celestial de maneira a representar os seus eixos de rotação. A cada planeta foi atribuído um ângulo e foi explicitado que esse ângulo está determinado em relação ao eixo Y, já que é em relação ao eixo vertical que foram determinados esses ângulos cientificamente. Esses ângulos foram todos arredondados às unidades porque a diferença entre o ângulo verdadeiro e aquele que foi atribuído é desprezável. Isto leva a que haja casos (como Mercúrio) em que o ângulo arredonda para 0^o e não seja necessário aplicar a rotação. Para as luas, foram usados os ângulos em relação ao Sol, já que este é uma espécie de referencial na nossa representação e também porque os ângulos em relação aos planetas podem variar.

Após a rotação, foi dada uma escala (utilizando o *scale*) aos planetas de acordo com os seus tamanhos relativos, com ligeiros arredondamentos (por exemplo, nesta representação, a Terra tem exatamente o mesmo tamanho que Vénus, apesar de ser ligeiramente maior na vida real). As luas de Júpiter não recebem este *scale* porque se recebessem daria um resultado aproximadamente igual ao resultado obtido ao não usá-lo. O Sol foi diminuído propositadamente em relação aos outros planetas de maneira a que este não ocupasse o espaço todo, tendo ao mesmo tempo os planetas a um tamanho que permitisse ver as

suas diferenças.

Por fim, foi dado um ficheiro a cada corpo celestial como modelo da sua representação. Como todos, apesar de serem achatados ou até irregulares em termos de forma, se aproximam de um formato esférico, todos eles receberam o ficheiro "sphere.3d"criado na primeira fase do projeto.

Quanto aos corpos representados, apenas criamos os mais necessários ou notáveis: O Sol, os oito planetas principais, a Lua (lua da Terra), Fobos e Deimos (luas de Marte), Ganimedes e Europa (as duas maiores luas de entre as dezenas de luas de Júpiter).

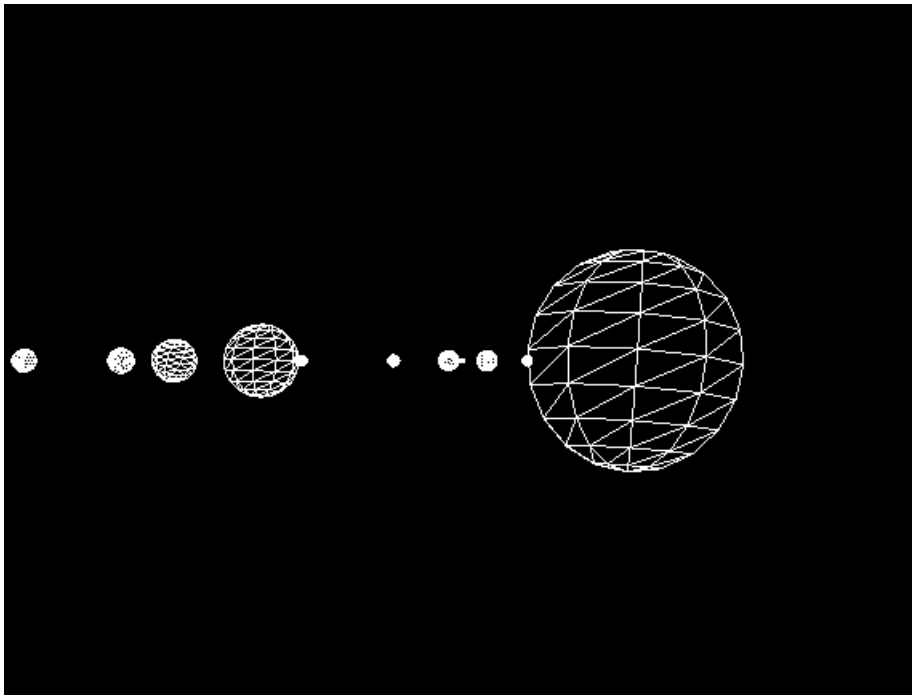

Figura 3: Representação gráfica do Sistema Solar

5 Conclusão

Esta fase do trabalho tinha como objetivo a modificação do engine de maneira a serem associadas às figuras transformações, como translações e rotações, exemplificadas através de um modelo do sistema solar. Apesar de não ser um objetivo desta fase, as figuras foram desenhadas com VBO. Concluída esta tarefa, pode-se prosseguir para as próximas etapas, que certamente necessitarão destas alterações no engine como uma base para o resultado final deste projeto.