Universidade do Minho

Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Unidade Curricular de Computação Gráfica

Relatório do Trabalho Prático

2016/2017

GRUPO 35

Pedro Cunha A73958, José Silva A74601, Luís Fernandes A74748, João Coelho A74859

1 de maio de 2017

Índice

Introdução

A terceira fase do trabalho prático continua a incidir sobre transformações geométricas, dando agora uma atenção especial à translação e rotação com animação. Fazendo uso de um determinado tipo de curvas, pretende-se animar o modelo. Para além disto, a estratégia, no que diz respeito à maneira como os modelos são desenhados, passa a ser distinta da fase anterior e devidamente descrita mais à frente. Finalmente, criam-se as funcionalidades capazes de criar modelos baseados num novo tipo de ficheiro, capaz de representar curvas de Bezier.

Particularizando, distintamente da fase anterior, o objetivo recai agora na criação de uma cena baseada num sistema solar dinâmico, com a representação do Sol, planetas e luas, definidos com hierarquia, e um cometa capaz de definir uma trajetória especifica baseada numa determinada curva.

Fase 3

1) Descrição do processo de leitura

Quanto ao processo de leitura (*parser XML*), as alterações relativamente à fase anterior consistiram na inclusão do tempo para as rotações e translações, para que cada planeta rodasse em torno do Sol à velocidade que lhe é característica, bem como em torno de si próprio.
De notar ainda a adição da função *getPoints* que é usada nas translações para que os planetas rodem em torno do Sol segundo uma curva de *Catmull-Rom*.

Sempre que é encontrado um *translate* ou um *rotate* com tempo, são criados objetos específicos, que, apesar de serem na mesma *transformations*, diferem dos objetos criados para um *translate* ou *rotate* ditos normais.

No caso específico do *translate*, o objeto referido anteriormente é criado já com os pontos obtidos pela função *getPoint*, função esta que obtém todas as *tags* filho que provêm da *tag translate* onde estão os pontos.

2) Bezier Patches

Nesta terceira fase do projeto prático foi proposta a inclusão de novos tipos de modelos baseados em 'Bezier patches'. Consequentemente, foi necessário incluir no generator uma funcionalidade capaz de receber um ficheiro do tipo patch (com vértices de controlo e conjuntos de índices), em conjunto com o valor de "tessellation level", e gerar uma lista com os vários pontos necessários (que formam triângulos) capazes de definir a superfície.

O processo dividiu-se em duas partes distintas. Numa primeira parte, foi necessário implementar um parser capaz de realizar a leitura do ficheiro patch e guardar toda a informação necessária numa estrutura de dados.

Através da análise do documento exemplificativo da estrutura de um ficheiro patch, tornou-se bastante mais fácil executar a primeira parte do processo descrito anteriormente. Dado que o ficheiro patch contém a indicação do número de patches e do número de vértices de controlo existentes no documento, guardando essa informação é possível facilmente identificar os vários campos do ficheiro. Utilizando um simples inteiro como iterador, facilmente se percebeu que quando o seu valor é:

- 0 – Está-se perante a linha correspondente ao número de patches;
- Maior do que 0 e menor ou igual ao número de patches – Está-se perante a linha correspondentes aos vários índices para determinada patch;
- Igual ao número de patches mais um (dado que o iterador começa em 0) – Está-se perante o número de pontos de controlo;

Caso contrário, está-se perante os vários pontos de controlo (X,Y,Z), um em cada linha, até ao final do ficheiro. Tanto neste caso, como no caso correspondente aos índices para determinada patch, foram realizadas duas funções auxiliares capazes de ler a informação presente na linha para um:

- std::vector<std::vector<int>>, caso o parsing esteja a ser realizado sobre os índices;
- std::vector<std::vector<float>>, caso o parsing esteja a ser realizado sobre os pontos de controlo;

É utilizado um vector de vectores, dado que é necessário fazer a distinção entre os vários patches, no caso dos índices, e a distinção entre aquilo que são os componentes dos pontos de controlo e os vários pontos, no segundo caso.

Ainda no que diz respeito às funções auxiliares, alguns tópicos devem ser esclarecidos. Para os dois casos descritos anteriormente, as funções auxiliares são bastante semelhantes. Estabelecem o delimitador composto por uma vírgula seguida de um espaço, que, quando encontrado na linha, guarda a posição em que se encontra. Guardando a string que começa no início da linha e termina na posição guardada, conseguimos retirar a informação que pretendemos. O próximo passo passa apenas por transformar a string num inteiro/float e inseri-lo na estrutura desejada. Eliminando a string da linha lida, o processo prossegue. Existe apenas um caso extraordinário correspondente ao final da linha, onde não se verifica o delimitador descrito acima. Sendo tratado como um caso à parte, já na parte final da função (fora do ciclo), o processo é basicamente o mesmo (só que agora é tratada a

informação respeitante ao que resta da linha e não a que fez matching antes do aparecimento do delimitador).

Em conclusão, pode-se caracterizar o processo como sendo um ciclo de leitura das várias linhas do ficheiro, que, juntamente com os processos descritos acima, retiram toda a informação necessária para o próximo passo.

Já com o parse efetuado e todos os pontos guardados, passa-se então à segunda fase, que consiste em gerar os vértices dos triângulos da superfície em questão. Percorre-se todos os patches e para cada patch são obtidos todos os pontos de controlo correspondentes. De seguida, estes pontos são guardados em matrizes px, py e pz, estando em cada uma delas os valores da coordenada correspondente. Tendo em conta o valor de tessellation, obtém-se o "tamanho" de cada nível(1/tessellation). Posto isto, para cada par u,v(1>u>=0 & 1>v,>=0) tem-se o seguinte:

-Ciclo para u(de 0 até 1 incrementando tamanhoNivel).

-Ciclo aninhado para v(também de 0 até 1 e incrementando tamanhoNivel).

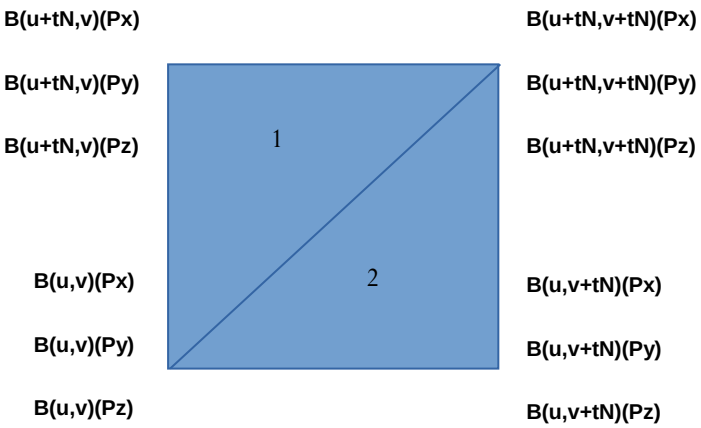
-Obter pontos para u,v; -Obter pontos para u+tamanhoNivel, v+tamanhoNivel;

-Obter pontos u+tamanhoNivel,v; (Forma-se 1º triangulo)

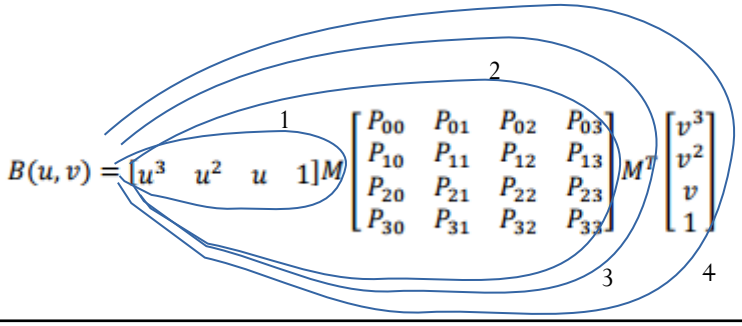
-Obter pontos para u,v; -Obter pontos para u, v+tamanhoNivel;

-Obter pontos u+tamanhoNivel,v+tamanhoNivel; (Forma-se 2º triangulo)

Esquema:

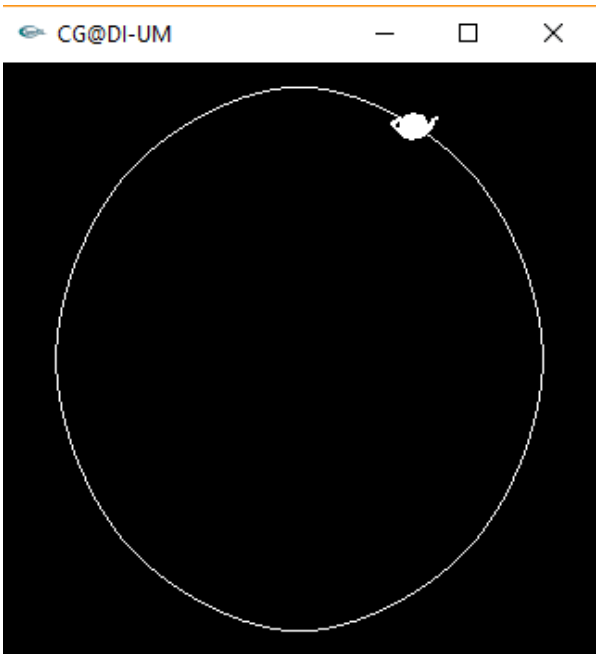


Estes pontos são obtidos seguindo a fórmula na figura seguinte, para cada matriz P(px,py,pz). Existe uma função que recebe como argumentos u, v, a matriz M e uma matriz P. O cálculo é feito em 4 fases representadas também na figura. Inicialmente multiplica-se U por M, de seguida (resultado da última fase)*P, mais uma vez (resultado da última fase)*MT(matriz transposta) e por último (resultado da ultima fase)*V.



3) Curvas de Catmull-Rom

Como expresso no enunciado, a translação dos planetas do sistema solar passa a ser expressa sob a forma de um conjunto de pontos, que reunidos permitem calcular curvas de Catmull-Rom. Reaproveitando o material da aula alusiva a estas curvas, mas alterando o número de pontos, e naturalmente as suas coordenadas, para que a curva se parecesse mais com uma órbita planetária, criou-se a seguinte curva com os pontos da tabela.



Point	Coordinates
P1	{0,0,-1.9}
P2	{-1.2,0,-1.3}
P3	{-1.7,0,0}
P4	{-1.2,0,1.3}
P5	{0,0,1.9}
P6	{1.2,0,1.3}
P7	{1.7,0,0}
P8	{1.2,0,-1.3}

Utilizando os pontos dessa curva, fez-se uma escala utilizando a distância dos planetas em relação à posição do Sol, considerada na fase anterior, para obter as curvas correspondentes à órbita de cada planeta:

Planet	Scale	Coordinates
Mercury	25	{0,0,-47.5} {-30,0,-32.5} {-42.5,0,0} {-30,0,32.5} {0,0,47.5} {30,0,32.5} {42.5,0,0} {30,0,-32.5}
Venus	30.5	{0,0,-57.95} {-36.6,0,-39.65} {-51.85,0,0} {-36.6,0,39.65} {0,0,57.95} {36.6,0,39.65} {51.85,0,0} {36.6,0,-39.65}
Earth	36	{0,0,-68.4} {-43.2,0,-46.8} {-61.2,0,0} {-43.2,0,46.8} {0,0,68.4} {43.2,0,46.8} {61.2,0,0} {43.2,0,-46.8}
Mars	43	{0,0,-81.7} {-51.6,0,-55.9} {-73.1,0,0} {-51.6,0,55.9} {0,0,81.7} {51.6,0,55.9} {73.1,0,0} {51.6,0,-55.9}
Jupiter	75.5	{0,0,-143.45} {-90.6,0,-98.15} {-128.35,0,0} {-90.6,0,98.15} {0,0,143.45} {90.6,0,98.15} {128.35,0,0} {90.6,0,-98.15}

Saturn	101	{0,0,-191.9} {-121.2,0,-131.3} {-171.7,0,0} {-121.2,0,131.3} {0,0,191.9} {121.2,0,131.3} {171.7,0,0} {121.2,0,-131.3}
Uranius	172	{0,0,-326.8} {-206.4,0,-223.6} {-292.4,0,0} {-206.4,0,223.6} {0,0,326.8} {206.4,0,223.6} {292.4,0,0} {206.4,0,-223.6}
Neptune	252	{0,0,-478.8} {-302.4,0,-327.6} {-428.4,0,0} {-302.4,0,327.6} {0,0,478.8} {302.4,0,327.6} {428.4,0,0} {302.4,0,-327.6}

Nota para o facto de estas órbitas estarem desenhadas no plano xOz. Equacionou-se desenhá-las com ligeira inclinação, porém a maioria das imagens analisadas, resultantes de uma pesquisa na web, mostraram que essa inclinação apenas é considerável no caso da órbita de Plutão, que atualmente já não faz parte do sistema solar e, como tal, não surge representado neste trabalho.

O enunciado desta fase pressupõe também que seja passado no XML o tempo que o planeta (objeto) necessita para percorrer a curva na totalidade. Explicar-se-á de seguida o tratamento a dar a este tempo de input, mas antes apresenta-se uma imagem exemplo do XML relativo às translações:

```
<!-- Earth -->
<group>
    <translate time="1000">
        <point X="0" Y="0" Z="-68.4"/>
        <point X="-43.2" Y="0" Z="-46.8"/>
        <point X="-61.2" Y="0" Z="0"/>
        <point X="-43.2" Y="0" Z="46.8"/>
        <point X="0" Y="0" Z="68.4"/>
        <point X="43.2" Y="0" Z="46.8"/>
        <point X="61.2" Y="0" Z="0"/>
        <point X="43.2" Y="0" Z="-46.8"/>
    </translate>
```

Para dar uso aos dados de input da imagem anterior criou-se uma classe com vários métodos conhecidos das aulas, com funções diversas como o cálculo do produto escalar, o cálculo do produto externo, a normalização de um vetor ou o produto de uma matriz por um vetor, para além das funções responsáveis pelo desenho das curvas de Catmull-Rom propriamente ditas.

Estas funções, de um modo geral, calculam a posição na curva utilizando quatro pontos de controlo e um índice que representa a distância entre os dois pontos de controlo mais próximos. Há também o cálculo da derivada no ponto, para efeitos de orientação do objeto sobre a reta. Nota para o facto de serem usados, como pontos de controlo, apenas quatro dos pontos passados através do XML, sendo os seus índices calculados.

Passando à explicação das animações de translação e rotação, estabeleça-se primeiro a comparação entre o tratamento dado ao tempo lido do XML em ambos os casos, que é praticamente em tudo igual, desde a utilização de glutGet(GLUT_ELAPSED_TIME), para medir o tempo em milissegundos desde o glutInit, ao cálculo do resto entre este valor e o tempo do XML, em milissegundos, e à divisão desse resto pelo mesmo tempo convertido do XML. A diferença está no facto de, para as translações, o processo parar aqui e o valor calculado é utilizado nas funções de Catmull-Rom como a distância entre os dois pontos de controlo mais próximos, enquanto que nas rotações este valor é multiplicado por 360, obtendo assim o ângulo a rodar em cada frame. Com estes valores, no caso da rotação, o próximo e último passo consiste na aplicação da rotação sobre o corpo, através de um glRotate. Já no caso da translação, seguem-se alguns cálculos utilizando a derivada calculada pelas funções de Catmull-Rom, referidas anteriormente, e um vetor que se supõe definir o alinhamento inicial do objeto em relação à curva. Estes e outros cálculos foram retirados dos slides da aula prática, de onde é originária a seguinte imagem que surge a propósito dos cálculos mencionados:

Assuming an initial specification of an $\overrightarrow{up}$ vector, to place and align the object with the curve, we need to build a transformation matrix for the object:

$$\vec{X} = P'(t)$$
$$\vec{Z} = X \times \overrightarrow{up}$$
$$\overrightarrow{up} = \vec{Z} \times \vec{X}$$

$$M = \begin{bmatrix} X_x & up_x & Z_x & p_x \\ X_y & up_y & Z_y & p_y \\ X_z & up_z & Z_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

`glMultMatrixf(float *m)`

- Current matrix gets multiplied by `m`

4) VBO’s

Como indicado no enunciado e realizado como nas aulas práticas, foi posto em prática o uso de VBO's (Vertex Buffer Object) de forma a guardar os vértices num buffer que irá residir na memória de vídeo em vez de residir na memória do sistema, podendo assim ser renderizado de maneira não imediata.

Existe GLuint buffers[3], em que cada índice conta com vértices, normais e texturas, em cada File carregado(file de vértices). Quando se carrega um ficheiro novo executa-se logo glGenBuffers, é apenas gerado um buffer(glGenBuffers(1, buffers);), uma vez que nesta fase não existem normais, nem texturas. Também se executa glBindBuffer e preenche-se o buffer executando glBufferData com os vértices correspondentes ao File em questão. Sempre que se pretende desenhar um determinado file, chama-se a função draw que executa glbindBuffer, glVertexPointer e glDrawArrays.

5) Informações adicionais

Relativamente à fase anterior, foi também incluída a First Person Camera, para facilitar a visualização e navegação entre planetas, seguindo a implementação abordada nas aulas práticas.

Conclusão

Nesta terceira fase do projeto foi possível aprimorar o modelo já iniciado na fase anterior, mais concretamente no que diz respeito a questões de performance e de inserção de novas funcionalidades.

Se, por um lado, a utilização de VBOs (Vertex Buffer Object) veio permitindo reduzir o número de chamadas a funções e o uso redundante de vértices partilhados, a definição da curva de Catmull-Rom trouxe consigo a capacidade de transformar o modelo estático da fase anterior num modelo dinâmico, mais aproximado da realidade. Para além disto, a inclusão de ficheiros patch, como ponto de partida para a criação do modelo, veio permitir uma representação mais concisa de objectos curvos.

De um modo geral, faz-se uma apreciação positiva desta terceira fase do trabalho, dado que o modelo desenvolvido do sistema solar corresponde àquelas que foram as exigências descritas no enunciado. De salientar a importância da base desenvolvida na fase anterior para a versão bastante satisfatória desenvolvida na presente fase.