

**Universidade do Minho**

Escola de Engenharia

Mestrado Integrado em Engenharia Informática

**Unidade Curricular de**

**Computação Gráfica**

Ano Lectivo de 2016/2017

|  |
| --- |
|  |
| 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 2](#_Toc481413342)

[Fase 3 3](#_Toc481413343)

[Conclusão 6](#_Toc481413344)

### Introdução

A terceira fase do trabalho prático da unidade curricular de “Computação Gráfica” continua a incidir sobre transformações geométricas, dando agora uma atenção especial à translação e rotação. 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, definidas 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 ao da 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.

1. Descrição das estruturas de dados
2. **Bezier Patches**

Nesta terceira fase do projeto prático foi proposto 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 “tressellation 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 percebemos que quando o seu valor é:

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

Caso contrário, estamos 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, podemos 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.

1. **Informações adicionais**

### Conclusão

Nesta terceira fase do projeto foi possível, ao grupo de trabalho, 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 vieram permitir uma representação mais concisa de objectos curvos.

De um modo geral, fazemos 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.