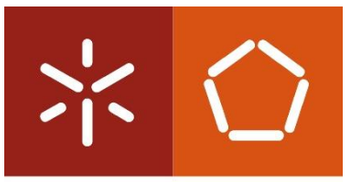

Universidade do Minho

Computação Gráfica
Fase 3

Benjamim Coelho, Henrique Neto, Leonardo Marreiros e
Júlio Alves

e-mail: {a89616,a89618,a89537,a89468}@alunos.uminho.pt

1 de maio 2021

1 Introdução

Para esta fase tínhamos objetivos para o engine e para o generator, ao contrário da fase 2, em que só foi necessário fazer alterações no engine. No engine, tinhamos de redefinir a forma como a translação e a rotação eram feitas para permitir a implementação da curva cúbica de Catmull-Rom, assim como definir quantos segundos se demora a percorrer toda a curva, sendo que o objetivo primário é fazer animações baseadas nessas curvas. No generator, temos de o modificar de forma a permitir que a aplicação seja capaz de criar um novo tipo de modelo baseado nos patches de Bezier. Para esta fase também temos como objetivo desenhar os modelos com VBOs (Vertex Buffer Objects), ao contrário das fases anteriores.

2 Arquitetura do projeto

2.1 Engine

2.1.1 Display

O ficheiro Display que foi criada nesta fase não contém nenhuma informação nova. Foi criada apenas com o propósito de termos uma melhor estruturação do projeto e com as classes devidamente separadas, pois nas fases anteriores estas funções estavam na Main, porém tornavam a mesma desnecessariamente confusa e desorganizada.

2.1.2 Models

Como nesta fase pretendemos ler os modelos utilizando VBOs, tivemos de proceder a alterações nesta classe. Desta forma a classe Models evoluiu para uma classe abstracta que obriga que cada modelo defendido implemente dois métodos. Um é responsável por ler e inicializar os objetos em memória e o outro é responsável por desenhar o modelo na cena,

Atualmente esta classe possui duas subclasses, **BulkModel** e **IndexedModel**. Ambas as classes fazem partido de VBOs, permitido reduzir drasticamente o número de pedidos feitos à placa gráfica, fazendo com que tenhamos uma melhoria significativa de desempenho comparado com as fazes anteriores.

BulkModel

A classe BulkModel permite ler os modelos previamente criados nas outras fazes de maneira a permitir compatibilidade com versões anteriores.

IndexedModel

A classe IndexedModel que nos permite ler e desenhar os ficheiros .3d com o novo formato que implementamos, que graças a um simples sistemas de índices permite-nos ter ficheiros .3d mais compactos e assim facilitar a sua reprodução no ecrã.

2.1.3 Groups

A classe Groups teve uma ligeira mudança relativamente à ultima fase. Cada grupo responsável por representar modelos apenas contém um índice dos modelos em vez dos objetos em si. Isto deve-se ao facto que os modelos agora são guardados nos objetos das cenas e não na árvore de instruções em si.

2.1.4 Scenes

A classes representativas da cena foram mudadas drasticamente de forma a poderem implementar as novas funcionalidades de maneira eficiente. Desta forma a classe *StaticScene* foi completamente substituída pela classe *Scene*.

A classe *Scene* possui um objeto *Group* que contém a informação de como desenhar a cena na sua árvore. Adicionalmente também instancia a lista de modelos que intervém na cena. A partir disto é possui montar a cena em memória (através do método *assemble*) e sucessivamente desenhar as respetivas imagens a partir do método *draw*. Adicionalmente também permite ao utilizador ativar e desativar as animações e desenhar as rotas estabelecidas no ficheiro XML.

2.1.5 Transformations

A superclasse *Transformation* não sofreu alterações significativas, apenas foram implementadas duas novas subclasses a sua hierarquia, a classe *Spin* e a classe *CatmullRomRoute*.

Spin

Esta classe implementa as funcionalidades de rotação contínua ao longo do tempo e de um eixo de rotação. Com isto, o método *apply()* (herdado da superclasse *Transformation*) limita-se a aplicar uma rotação no eixo instanciado com base no tempo de rotação e no relógio do sistema.

CatmullRomRoute

Esta classe permite percorrer uma trajectória definida por uma *spline* de *Catmull-Rom* durante um período de tempo predefinido. Desta forma a partir de 4 ou mais pontos é possível fazer um grupo de objetos movimentar-se ao longo do tempo. Esta classe faz partido da classe *CatmullRoomCurve*, definida na biblioteca *utils* para calcular a translação a cada instante.

2.1.6 Parser XML

Para facilitar a construção das novas cenas, cada função do parser agora recebe um objeto constante da uma nova classe *BuildingInfo*. De momento esta classe é apenas utilizada para identificar e registar os modelos lidos de forma assegurar que o mesmo modelo não é instanciado mais do que uma vez em memória, mesmo se este for desenhado mais do que uma vez na cena.

Adicionalmente também extendidas as funcionalidades de *Translate* e *Rotate*.

Translate

Agora a translação pode receber um conjunto de pontos que definem a curva cúbica de Catmull-Rom, tal como os segundos que o objeto deverá demorar a percorrer a mesma. Para além disso, podemos passar o parâmetro fwd="true"para ativar a rotação do objeto de acordo com a curva que está a percorrer, fazendo com que este esteja sempre virado para a "frente".

```
<translate time="10" fwd="true">
    <point x="60" z="60"/>
    <point x="0" z="60"/>
    <point x="-60" z="60"/>
    <point x="-60" z="38"/>
    <point x="-60" y="14.142" z="21"/>
    <point x="-60" y="14.142" z="-5"/>
    <point x="-60" z="-21"/>
    <point x="-60" z="-60"/>
    <point x="0" z="-60"/>
    <point x="60" z="-60"/>
    <point x="60" z="0"/>
</translate>
```

Figura 2.1: Exemplo do uso do translate

Rotate

Quanto à funcionalidade de rotação, em vez de receber o ângulo de rotação, pode receber o tempo de rotação de 360^{o} em volta do eixo especificado.

```
<rotate time="10" axisX="0" axisY="1" axisZ="0"/>
```

Figura 2.2: Exemplo do uso do rotate

2.2 Generator

O generator passou a suportar novas primitivas com recurso a *Patches de Bezier*. Também procedemos a alterar a forma como os ficheiros .3d são criados de forma a que a sua estrutura seja mais eficiente.

2.2.1 Novo formato do ficheiro 3d

Nesta fase, mudámos completamente o modo como o *generator* cria ficheiros 3d para os vários modelos (primitivas e output de *patch de bezier*). Anteriormente, o generator escrevia no ficheiro 3d todas as coordenadas de todos os triângulos que compõem o modelo. Ora, esta abordagem tem a vantagem de ser simples de implementar, no entanto tem a desvantagem de esrever pontos repetidos no ficheiro 3d, o que posteriormente, diminui a eficiência e desempenho do engine. Agora, o generator tem o cuidado de escrever apenas uma vez cada ponto. Para tal, ao calcular todos os pontos necessários para o modelo, atribui a cada um, um índice. Com estes índices, sempre que é calculado um triângulo, o generator guarda a sequência de índices correspondentes aos pontos que o formam. No final, começa por escrever o número total de pontos, seguido de uma nova linha com o número total de índices, de modo a facilitar a leitura do ficheiro no engine. De seguida, escreve todos os pontos únicos calculados, seguidos de todas as sequências de índices dos triângulos necessários ao desenho do modelo (um por linha). Com esta nova abordagem, o engine guarda menos informação repetida em memória e pode utilizar VBO's com índices de modo a melhorar ainda mais o desempenho do nosso programa.

2.2.2 Patch

As superfícies de Bezier são compostas por conjuntos de 16 pontos denominados de patches, pontos esses que na verdade não são pontos, mas sim indices que referenciam os pontos de controlo da superficie. Esses 16 pontos de controlo são divididos em grupos de 4 que descrevem os arcos da superfície indicada por cada patch. Após a definição de cada arco, recorremos ao nível de tecelagem que indica quantos pontos são necessários criar nessa curva, e desta forma, indica o nível de precisão com que a primitiva irá ser desenhada.

2.2.3 Curvas de Bezier

Nesta fase, o generator é capaz de criar um novo tipo de modelo baseado em patches de Bezier. Para isso, o utilizador tem de passar como argumentos ao generator o nome do ficheiro onde os pontos de controlo de Bezier estão definidos, o nível de tesselação pretendida e, por último, o nome do ficheiro de output que conterá todas as coordenadas de todos os triângulos necessários para desenhar a superfície, tal como todos os índices.

Um patch de uma superfície de bezier é uma matriz 4x4 de 16 pontos de controlo que descrevem a superfície bicúbica G_B, oferecendo uma definição matemática compacta da superfície sem ser necessário armazenar cada um dos pontos interpolados. É uma generalização das curvas de Bezier, em que cada uma das 4 linhas dos pontos de controlo podem ser consideradas como uma curba de Bezier em duas dimensões. A representação das curvas de Bezier em patches é extremamente poderosa para a descrição analítica da superficie, que pode ser facilmente manipulada.

Os pontos de fora de G_B representam pontos nas bordas da superfície interpolada, enquanto que os pontos interiores (qualquer ponto que não esteja na primeira linha, quarta linha, primeira coluna ou quarta coluna) são pontos intermédios que especificam indiretamente os vetores tangentes à superfície. As direções x,y, e z da superfície

são calculados independentemente. Assim para um patch de uma superfície em três dimensões, existem matrizes geométricas separadas G_{Bx}, G_{By} e G_{Bz}, uma para cada direção. Deste modo, a matriz geométrica G_{Bk} é dada por:

$$G_{Bk} = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

onde $P_{i,j}$ são as coordenadas na direção k.

Cada ponto da superfície depende de dois parâmetros u e v que variam entre 0 e 1, tal como nas curvas de Bezier em duas dimensões. Deixando u controlar a variação ao longo do patch de cima a baixo(de P_{0X} a P_{3X}) e v da esquerda para a direita (de P_{X0} a P_{X3}). Quando u ou v são 0 ou 1, o ponto descrito fica precisamente na borda da superfície. Daí u = v = 0 fica no ponto P_{00} e u = v = 1 no ponto de controlo P_{33}.

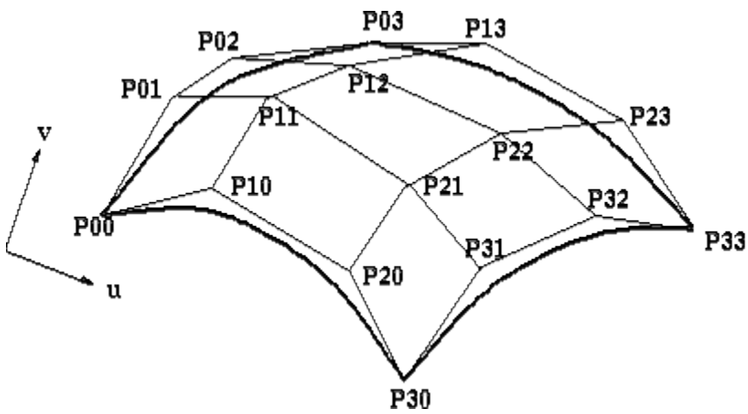

Figura 2.3: Exemplo de uma superficie de Bezier

Um ponto da superficie pode ser calculada da seguinte maneira. Para cada coordenada x,y e z:

$$x(u,v) = U * M_B * G_{Bx} * M_B^T * V^T$$
$$y(u,v) = U * M_B * G_{By} * M_B^T * V^T$$
$$z(u,v) = U * M_B * G_{Bz} * M_B^T * V^T$$

Onde M_B é a matriz base de Bezier tal como é utilizada no cálculo de curvas Bezer em duas dimensões:

$$M_{Bk} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

e os vetores U e V dependem dos parametros u e v, que variam entre 0 e 1:

$$U = [u^3 u^2 u 1]$$
$$V = [v^3 v^2 v 1]$$

2.3 Utils

2.3.1 Interpolation

Neste fase, o ficheiro *Interpolation* contém apenas a classe *CatmullRomCurve* que fornece os métodos e funcionalidades necessários para trabalhar com estas splines.

CatmullRomCurve

A classe *CatmullRomCurve* calcula os pontos e as tangentes aos vários pontos pertencentes a curva definida. Para se construir o objeto é necessário fornecer pelo menos 4 pontos para se definir a curva.

A partir dos pontos dados, o objecto permitirá computar os pontos e as tangentes de cada segmento da curva segundo as expressões:

$$P(t) = (t^3, t^2, t, 1) \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5f \\ 1.0 & -2.5 & 2.0 & -0.5f \\ -0.5 & 0.0 & 0.5 & 0.0f \\ 0.0 & 1.0 & 0.0 & 0.0f \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$P'(t) = (3t^2, 2t, 1, 0) \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5f \\ 1.0 & -2.5 & 2.0 & -0.5f \\ -0.5 & 0.0 & 0.5 & 0.0f \\ 0.0 & 1.0 & 0.0 & 0.0f \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

onde $P_{\{0,1,2,3\}}$ são os pontos adjacentes ao segmento e t é o valor de deslocação do mesmo.

Tendo em conta que teremos de efectuar estes cálculos várias vezes para o mesmo segmento, estas operações encontram-se distribuídas em três métodos diferentes para que em conjunto com quatro variáveis de instância($index0, v1, v2, v3$) exista uma diminuição grande de cálculos.

Primeiramente, o utilizador deve chamar o método *getLocalT(globalT)*, que com base num deslocamento global na curva *globalT* calcula o deslocamento local no segmento correspondente e prossegue por comparar o seu estado interno com o estado calculado através da variável $index0$ (que contém o índice do primeiro usado na interpolação). Caso estes não correspondam ao mesmo segmento as operações matriciais presentes em ambas as formulas são calculadas e o resultado é escrito nos objetos $v1, v2$ e $v3$. Finalmente, o método devolve a deslocação local previamente calculada.

Com isto o deslocamento local pode ser usado para chamar os métodos *getPoint* e *getTangent* que em conjunto com as variáveis de instância $v1, v2$ e $v3$ calculam respetivamente o ponto e a tangente á curva na posição pedida.

2.3.2 Point

Devido à necessidade de implementar Catmull-Rom e os patches de Bezier, tínhamos de ser capazes de fazer operações com matrizes, sendo que as alterações nesta classe prendem-se exatamente com essa necessidade, sendo que agora somos capazes de rodar, transpor e multiplicar matrizes com vetores.

2.3.3 Vector

De modo a sermos capazes de fazer operações com vetores mais facilmente, criámos a classe Vector. Cada objeto desta classe tem como atributos 4 floats (x,y,z,w) o que nos permite distinguir entre pontos e vetores. Para além disto, implementámos vários métodos para nos permitirem realizar operações sobre vetores tais como: *cross product*, *dot product* e a norma de um vetor.

2.3.4 Matrix

A classe Matrix foi criada de modo a facilitar o cálculo de operações que envolvem matrizes. Tem, como atributos, um array de floats bidimensional 4x4, que representa, obviamente, uma matriz de floats 4x4. Para além disso, esta classe disponibiliza um conjunto de métodos para facilitar cálculos que envolvem matrizes tais como: multi-

plicar matrizes, calcular a transposta e criar uma matriz a partir de vetores.

3 Construção de cenas

3.1 Sistema Solar

Fazendo uso das novas funcionalidades implementadas, é possível agora melhorar o sistema solar apresentado na fase anterior. Os planetas passam agora a ter um período de translação em volta do sol e rotação própria. Foi também adicionado um cometa, utilizando o modelo do teapot que foi gerado com patches de Bezier pelo generator, que executa também um movimento de translação em volta do Sol mas este definido por curvas de Catmull-Rom.

É de notar que o tamanho dos planetas e o seu tempo de órbita estão à escala.

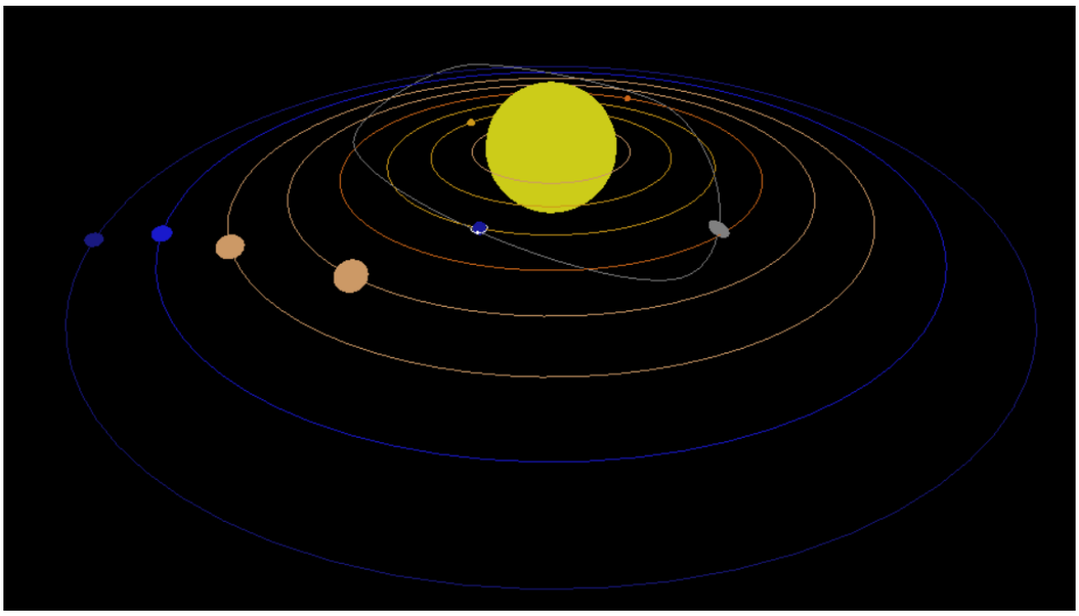

Figura 3.1: Sistema solar com as trajetórias visíveis

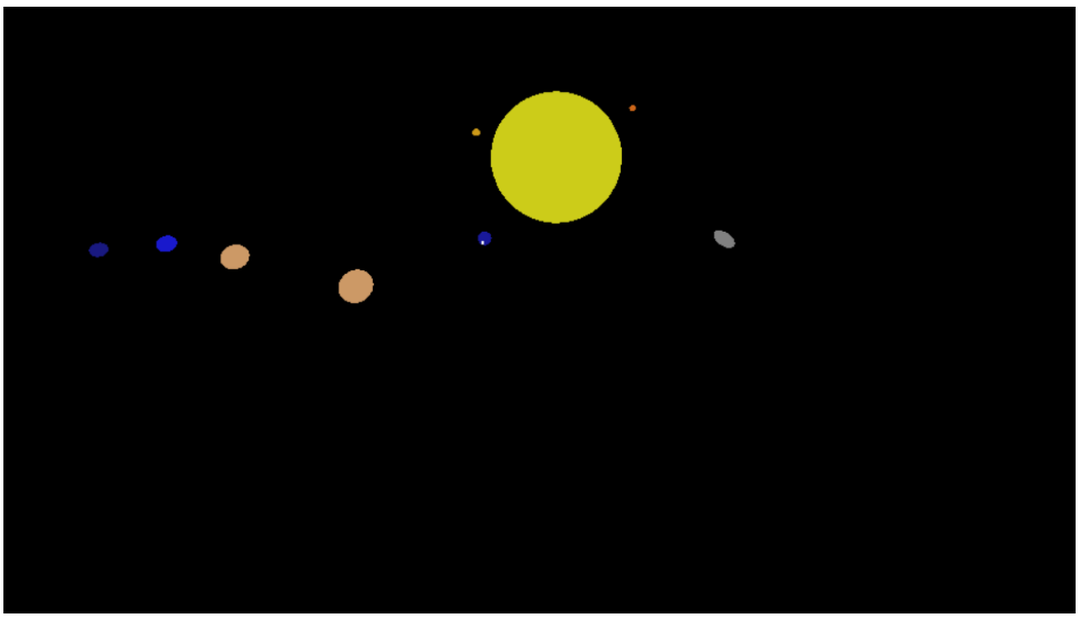

Figura 3.2: Sistema Solar sem as trajetórias visíveis

3.2 Pista de Natal

A cena da árvore de natal da fase anterior evoluiu agora para uma pista de corrida onde três *teapots* competem entre si numa pista com elevações e curvas. Para esta cena em específico fundimos duas cenas, a cena da pista com os teapots a percorrer a mesma com a cena da árvore de Natal, utilizando um comando no ficheiro XML chamado *input*, que nos permite juntar diferentes cenas numa só.

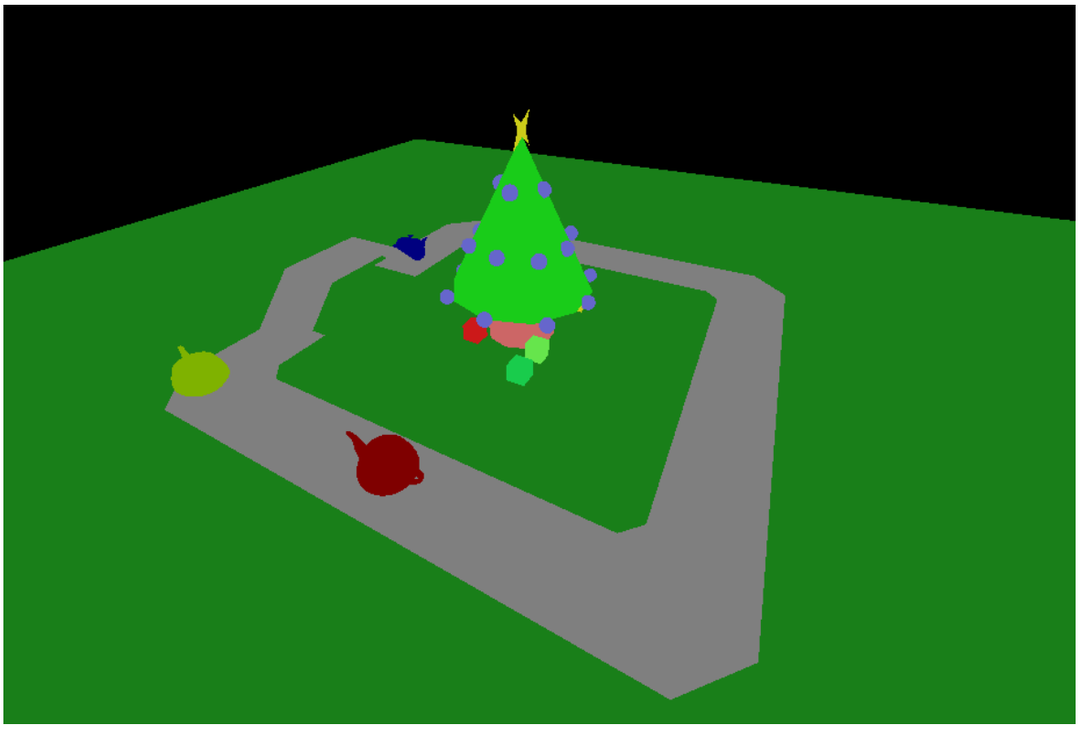

Figura 3.3: Pista de corrida de Natal

4 Conclusão

Com a terceira fase, foi possível implementar animações em translações com o uso de curvas de Catmull-Rom que representaram um método simples para o cálculo de todos os pontos que traduzem uma animação; e animações em rotações.

Além disso, conseguimos agora construir primitivas de maior complexidade como o *teapot* com auxilio das superfícies cúbicas de Bezier.

Quanto à eficiência, o desenho de primitivas com recurso a VBOs e a nova forma como os ficheiros são escritos com índices permitiram um aumento significativo neste aspeto.

Finalmente, podemos concluir que foram atingidos todos os objetivos esperados para esta fase onde foram aplicados todos os conhecimentos lecionados acerca de computação gráfica. Como trabalho futuro pretendemos implementar iluminação e texturas mas também acrescentar mais extras e cenas.