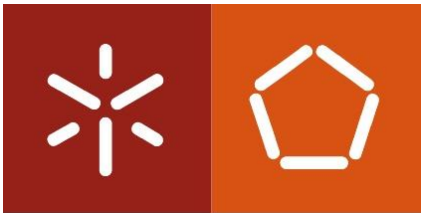

Universidade do Minho

Departamento de Informática

Computação Gráfica
Grupo 45
Trabalho Prático
Fase 3

6 de maio de 2021

Ariana Lousada (A87998)

Rui Armada (A90468)

Carolina Vila Chã (A89495)

Sofia Santos (A89615)

Conteúdo

Capítulo 1

Introdução

Um dos principais objetivos desta terceira fase do trabalho prático consiste em atualizar o *generator* concebido na anteriormente de modo a ser possível criar um novo tipo de modelo baseado em *Bezier patches*.

O outro objetivo principal desta fase é expandir os elementos de translação e rotação do *engine*, permitindo assim a criação de animações que evoluem com o passar do tempo. Estas translações animadas serão implementadas através de curvas de Catmull-Rom.

Fizemos ainda uma transição de desenhar diretamente os vértices para o uso de VBOs.

Para mostrar o correto funcionamento do nosso *engine* nesta fase, expandimos o modelo do Sistema Solar concebido na fase anterior de forma a tirar proveito destas novas funcionalidades. Incluímos ainda no modelo uma figura gerada a partir de *Bezier patches*, que representa um cometa, para mostrar que a nova funcionalidade do *generator* funciona de forma adequada.

Capítulo 2

Desenvolvimento

2.1 VBOs

O desenvolvimento desta fase começou pela transição para VBOs, visto que esta funcionalidade apenas implica alterar o que foi feito nas fases anteriores.

Na 2ª fase definimos uma estrutura "group"para cada grupo que conste do ficheiro *XML* fornecido ao *engine* e que contenha pelo menos um sólido. Dentro de cada instância desta estrutura temos um vetor com todos os pontos que constituem os sólidos desse grupo. O que fizemos para implementar VBOs foi, depois de ler o ficheiro *XML*, percorrer a lista de grupos criada após esta leitura e, para cada grupo, criar um *buffer* onde colocamos todos os vértices que fazem parte dos sólidos desse grupo. Os buffers estão armazenados num vetor, de tal forma que os seus índices correspondem aos do vetor dos grupos, ou seja, o buffer correspondente ao grupo com índice 3 também terá índice 3 no vetor dos buffers.

Foi ainda preciso alterar a função `renderScene` de forma a utilizar os buffers para desenhar os pontos. De resto, a implementação dos VBOs foi igual à que nos foi apresentada nas aulas práticas da UC, como tal não consideramos pertinente a sua descrição em detalhe, sendo possível consultar o código relevante no ficheiro relativo ao nosso *engine*.

Considerámos eliminar a estrutura que guarda os vértices de cada grupo, visto que seria possível gerar os buffers durante a leitura do ficheiro, mas desta forma somos capazes de permitir ao utilizador que escolha entre os dois modos de renderização, com ou sem VBOs. Para o fazer apenas deve premir o botão direito do rato enquanto corre o programa e aparecerá um menu com a opção de ativar/desativar os VBOs.

Com esta alteração efetuada, já temos uma base para iniciar o desenvolvimento das outras funcionalidades.

2.2 Bezier Patches

Para aplicar as curvas de Bezier desenvolvemos uma função *patch* que recebe um ficheiro XML juntamente com o número de *tesselations*. Com o número da *patch* e index a função vai adquirir os control points ao ficheiro que é fornecido. De seguida, é aplicado o algoritmo de Bezier utilizando os pontos de controlo e as inclinações (1.0 por *tessalation*). No final os pontos gerados são escritos num ficheiro ".3d".

2.3 XML Parsing e armazenamento dos dados

Para suportar as animações que vamos implementar, precisamos primeiro de alterar a forma como lemos os ficheiros *XML*, já que estas animações vão implicar uma nova variável nas translações e rotações.

Agora, quando estamos a ler uma transformação, se encontrarmos a variável `time` iremos, para além de armazenar o seu valor, alterar a forma como lemos o resto das variáveis. Para uma translação, em vez de armazenar as variáveis numa matriz, armazenamos os pontos relativos à curva de Catmull-Rom num vetor de arrays. Para uma rotação, armazenamos os valores de cada eixo num array.

Com esta alteração, perdemos a vantagem de usar apenas uma matriz para as transformações. Ao usar uma matriz, a ordem das transformações era preservada. Agora que não usamos apenas uma matriz, precisamos de criar outra estrutura de modo a preservar a ordem das transformações a efetuar. Assim, cada grupo passa a ter um vetor cujos elementos são instâncias de uma estrutura chamada `transformation`, que pode ser de três tipos. O primeiro tipo é o tipo "normal", isto é, o que já tínhamos na 2ª fase, e apenas recorre a uma matriz. O segundo tipo diz respeito a uma translação com curvas de Catmull-Rom, e precisa, para além do tempo, de um vetor com os pontos que fazem parte da mesma. O terceiro tipo inclui as rotações com valor temporal e usa uma variável para o tempo e outra para os eixos da rotação. De forma a reutilizar variáveis e assim poupar memória, a variável do tempo é igual para as translações e rotações, e os eixos da rotação ficam armazenados no vetor que armazena os pontos da translação. Para além destas variáveis, temos um array que contém os valores antigos de `y`. Para já não serve para nada, mas será útil quando formos aplicar a translação através de curvas de Catmull-Rom.

É importante realçar que, ao aplicar estas transformações, devemos percorrer o vetor da direita para a esquerda, visto que no OpenGL as transformações são aplicadas por ordem inversa.

Podemos agora implementar as animações.

2.4 Animações

Para criar as animações, a primeira coisa que fizemos foi fornecer uma "*idle function*"ao GLUT, neste caso a função `renderScene`, tal que as animações possam ser realizadas independentemente de qualquer input dos utilizadores.

Depois, usámos um algoritmo bastante semelhante ao desenvolvido nas aulas práticas da UC para implementar as curvas de Catmull-Rom. As variáveis necessárias pelo algoritmo encontram-se na estrutura "*transformation*"que mencionámos na secção anterior, incluindo a variável "prev_y", que vai ser precisa para calcular a rotação apropriada de cada modelo. Este algoritmo é implementado na função *renderScene* pois a posição dos modelos vai variar no tempo, logo precisa de ser executado de cada vez que a cena é renderizada.

Para as rotações a única coisa que fizemos foi calcular o ângulo de rotação com base no tempo através da equação

$$angle = 360.0 * (glutGet(GLUT_ELAPSED_TIME)/1000.0)/time$$

onde `time` é o tempo da rotação tal como foi lido no ficheiro *XML*. Com o ângulo calculado apenas precisamos de fazer uma rotação usando este ângulo e o vetor de rotação armazenado na estrutura relativa a esta transformação.

2.5 Cena de demonstração - Sistema Solar

Esta cena foi desenvolvida a partir da cena que concebemos para a segunda fase do trabalho. Começámos por atribuir rotações a cada planeta, sobre si próprio e sobre o Sol. É importante que a rotação sobre os próprios planetas seja efetuada antes da translação para o seu sítio no sistema solar, visto que a translação altera a origem do referencial para esses sólidos. Por outras palavras, primeiro devemos colocar os planetas a rodar sobre si próprios, depois devemos colocá-los à distância pretendida do Sol e por fim definir a sua órbita em torno da estrela.

Em relação às luas, a rotação em torno dos seus planetas deve ser feita depois da rotação em torno de si mesmas e antes da rotação em torno do Sol.

Depois de termos estas rotações definidas, colocámos um cometa, cuja forma é a figura gerada usando as curvas de Bezier, a orbitar o Sol. Esta órbita é elíptica, tal como um cometa real, e segue uma curva de Catmull-Rom. Definimos mais pontos longe do Sol de forma a que o movimento do cometa seja mais acelerado quando se aproxima do astro, mais uma vez a simular o comportamento de um cometa real. Isto funciona porque o tempo que o cometa demora a ir de um ponto a outro é independente da distância entre os pontos, apenas depende do número de pontos e do tempo definido no ficheiro *XML*.

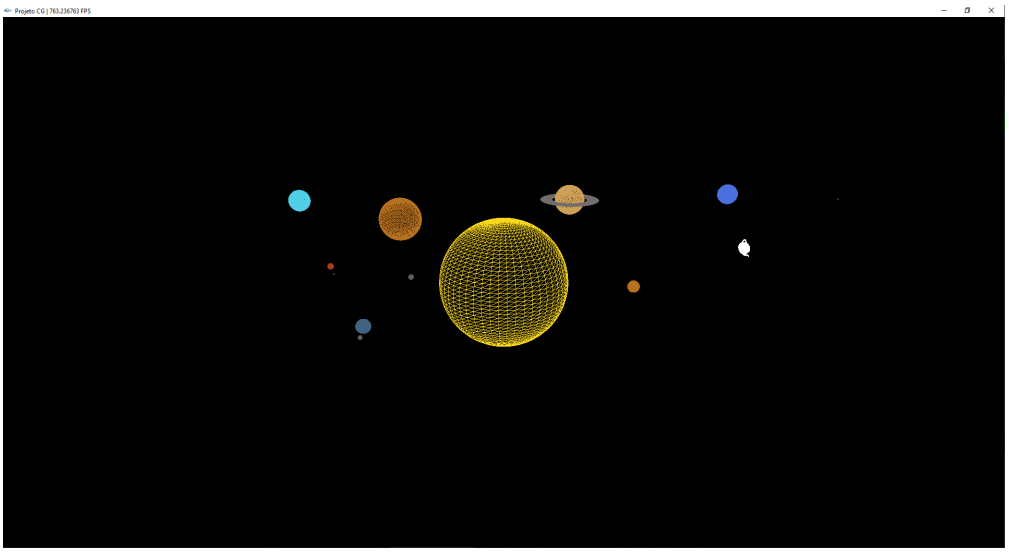

Figura 2.1: Sistema Solar. O cometa está virado para "baixo"pois a sua órbita é perpendicular à dos planetas.

Capítulo 3

Extras

Para além das funcionalidades pedidas, desenvolvemos ainda um pequeno menu que pode ser acedido ao premir o botão direito do rato, tal como foi mencionado anteriormente. Neste menu somos capazes de alternar entre duas formas de desenhar os pontos, com ou sem recurso a VBOs, e ainda trocar entre dois modos de desenho dos polígonos, um em que apenas desenhamos as suas linhas e outro em que preenchemos os triângulos que constituem os sólidos.

Para demonstrar a vantagem de preencher as figuras, desenvolvemos uma cena com o famoso boneco de neve Olaf, do filme de animação de 2013 *Frozen*. Na figura 3.1 podemos ver que, usando o modo `GL_FILL`, no qual as figuras ficam preenchidas, reconhecemos logo a personagem, enquanto que no modo `GL_LINE`, para além de ser possível ver as figuras que devem ficar parcialmente escondidas, como as pernas, o boneco fica quase irreconhecível e bastante mais feio. Num programa ou jogo real, o modo `GL_LINE` apenas seria usado para efeitos de *debugging* ou testes, sendo assim bastante útil ter uma forma de alternar entre os dois, deixando apenas o modo `GL_FILL` na aplicação final.

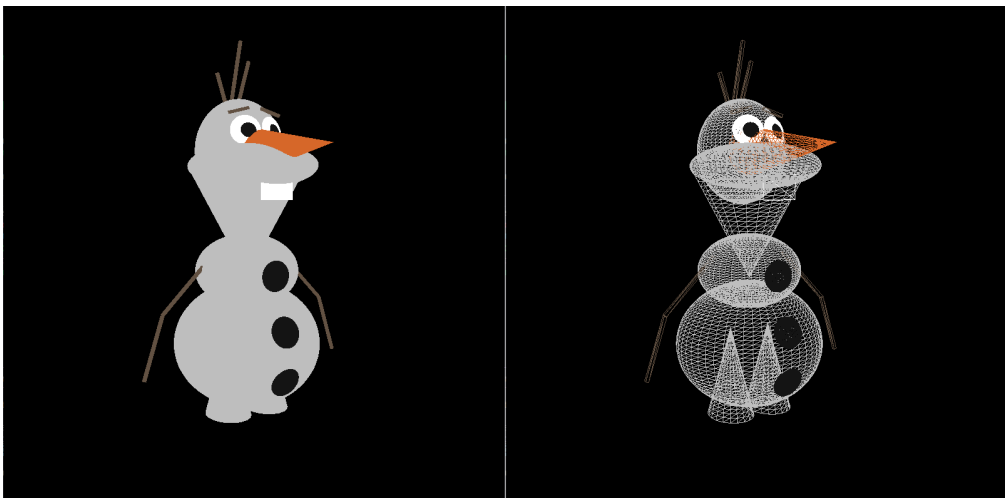

Figura 3.1: Modelo Olaf, com e sem preenchimento.xml

Fizemos ainda algumas melhorias à câmara desenvolvida na fase anterior, sendo agora possível controlá-la usando o rato. É possível consultar a lista completa de controlos ao correr o *Engine* e olhar para a linha de comandos.

Por último, definimos uma função que permite mostrar o *framerate* do programa na barra superior do mesmo, ao lado do título. Com esta funcionalidade, somos capazes de avaliar o desempenho do *engine* e de, por exemplo, comprovar a vantagem do uso de VBOs. Na imagem 2.1 verificamos que o *framerate* ronda os 750 *frames* por segundo, sendo que, na altura em que foi tirada esta captura de ecrã, a cena estava a ser renderizada recorrendo a VBOs. Desativando este modo, o *framerate* desce para valores próximos de 120 *frames* por segundo.

Capítulo 4

Conclusão

A realização desta fase do projeto permitiu-nos consolidar os conhecimentos adquiridos durante as aulas acerca de transformações de modelos 3D, patches de Bezier, VBOs e curvas de Catmull-Rom. Com isto, fomos capazes de calcular e aplicar órbitas a cada planeta assim como a rotação em torno dos seus próprios eixos no sistema solar previamente modelado. Na quarta e última fase irão ser abordados e aplicados temas como iluminação e texturas.