

# Trabalho Prático - Grupo 32 Fase 3 - Curvas, Superfícies Cúbicas e VBOs

# Computação Gráfica

Bruno Filipe de Sousa Dias A89583 Luís Enes Sousa A89597 Pedro Miguel de Soveral Pacheco Barbosa A89529







4 de abril de 2021

# Conteúdo

1	Intr	rodução	
<b>2</b>	Alt	Alterações na Estrutura do Projeto	
	2.1	Generator	
		2.1.1 generator.cpp	
	2.2	Engine (Dynamic Translate e Dynamic Rotate)	
		2.2.1 engine.cpp	
		2.2.2 group.cpp e group.h	
	2.3	Engine $(VBOs)$	
	2.4	engine.cpp	
	2.5	group.h	
	2.6	parser.cpp	
	2.7	model.h	
	2.8	Parsing	
3	Alterações na Estrutura do Ficheiro $XML$		
4	Alterações no Cenário Final		
5	Cor	nclusão e Trabalho Futuro	

# 1 Introdução

Nesta fase do trabalho prático, dando continuidade às duas fases anteriores, foi nos proposto alterar o Generator de maneira a ser capaz de criar um modelo com base em *Bezier patches*.

Também foi necessário alterar o Engine, de forma a suportar translações e rotações dinâmicas, usando curvas de *Catmull-Rom* para o primeiro caso e, ainda, desenhar os modelos usando VBOs.

O cenário final desta fase é um modelo dinâmico do Sistema Solar, incluindo um cometa cuja trajetória é definida por uma curva de *Catmull-Rom*.

## 2 Alterações na Estrutura do Projeto

Nesta secção, serão abordadas as alterações que foram efetuadas na estrutura do projeto, para alcançar os objetivos finais.

#### 2.1 Generator

Nesta fase, o Generator foi alterado de forma a ser capaz de converter um ficheiro com *Bezier patches* (.patch) e convertê-lo num ficheiro com a lista de pontos necessários para criar os triângulos para desenhar a figura (.3d), seguindo um dado nível de tesselação.

## ${\bf 2.1.1} \quad generator.cpp$

Neste ficheiro, foi adicionada a opção descrita anteriormente.

Para tal, começamos por fazer o parsing do ficheiro, guardando os índices dos patches num map, que tem um indicativo como chave e um vector de índices como valor. Quanto aos pontos de controlo, estes são guardados num vector.

De seguida, calculamos os pontos da superfície de Bezier, percorrendo os vários patches guardados. Começamos por obter os pontos de controlo. De seguida, inicializamos as matrizes de Bezier para estes pontos de controlo. Para criar a grelha final, demos valores às variáveis u e v, incrementando de acordo com o nível de tesselação, e usamos as matrizes calculadas anteriormente para obter os pontos finais. Com esta grelha calculada, podemos dividir cada quadrado em dois triângulos e guardar os seus pontos num vector, que depois será escrito num ficheiro .3d.

#### 2.2 Engine (Dynamic Translate e Dynamic Rotate)

Nesta fase do projeto, tivemos de suportar translações dinâmicas, usando curvas cúbicas de *Catmull-Rom*. Para tal, criamos uma classe *DynamicTranslate*, capaz de guardar as informações necessárias para criar a curva e calcular o ponto atual da translação.

Também foi necessário adaptar o nosso projeto para suportar rotações dinâmicas. Neste caso apenas precisamos de calcular constantemente o tempo decorrido, para calcular o ângulo de rotação nesse momento.

#### 2.2.1 engine.cpp

Neste ficheiro, apenas tivemos de alterar a função drawGroup(), tornandoa capaz de indentificar estas duas novas transformações. Se uma delas for encontrada, as respetivas funções são invocadas, aplicando a transformação aos modelos correspondentes.

## 2.2.2 group.cpp e group.h

Nestes ficheiros, foram adicionadas duas classes: DynamicTranslate e DynamicRotate.

A classe *DynamicTranslate* guarda o tempo total para percorrer a curva e os pontos de controlo de *Catmull-Rom*. Com estes pontos calculamos os pontos da curva, seguindo um certo nível de tesselação, para depois podermos desenhar a curva. Depois, quando queremos aplicar a transformação num dado momento, calculamos o tempo passado desde o início da curva e calculamos o ponto correspondente na curva, de forma a posicionar o *teapot* corretamente. É também calculada a derivada do ponto, para podermos orientar o *teapot* de acordo com a curva.

Quanto à classe *DynamicRotate*, esta guarda o tempo necessário para executar uma rotação completa e o eixo de rotação. Em cada frame desenhada é calculado o novo ângulo, de acordo com o tempo passado desde o início. Depois executamos uma rotação com esse mesmo ângulo e sobre o eixo guardado.

## 2.3 Engine (VBOs)

Nesta fase do trabalho, foi-nos exigido que usasemos *VBOs* para desenhar os modelos, ao invés do modo imediato.

#### 2.4 engine.cpp

Neste ficheiro, quando queremos desenhar os modelos, deixamos de ir buscar os pontos a um vector. Agora acedemos à VBO correspondente e usamos as funções disponibilizadas para desenhar os vértices.

### 2.5 group.h

Neste ficheiro, a estrutura da classe Group foi alterada. O vector que guarda os modelos do cenário, passa a ter elementos da classe Model e não da classe Object.

#### 2.6 parser.cpp

Neste ficheiro, a função que carrega os pontos de um ficheiro .3d foi alterada. Antes retornava um vector com os pontos lidos. Agora guarda os vértices numa VBO e retorna um Model que guarda a informação necessária para os desenhar depois.

#### $2.7 \quad model.h$

Este ficheiro foi adicionado ao projeto, substituindo o ficheiro object.h e guarda as informações necessárias para desenhar um model guardado numa VBO, nomeadamente o índice desta e o número de vértices.

## 2.8 Parsing

De forma a estruturar melhor o nosso código decidimos criar dois ficheiros, parser.cpp e parser.h, onde estão concentradas todas as funções de parsing, quer dos ficheiros .xml quer dos ficheiros .3d.

## 3 Alterações na Estrutura do Ficheiro XML

Nesta fase, e de forma a suportar as novas funcionalidades, tivemos que alterar a estrutura do ficheiro XML.

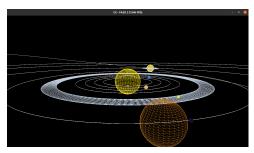
No elemento *rotate* podemos ter o atributo *time* em substituição do atributo *angle*, indicando o tempo necessário para completar uma rotação completa.

No elemento translate o atributo angle também pode ser substituído pelo atributo time. Neste caso, este representa o tempo necessário para percorrer a curva inteira. Os atributos X, Y e Z também desaparecem, sendo substituídos por uma sequência de pontos que constroem a curva de Catmull-Rom. Este elemento ainda possui um atributo closed, que indica se os pontos apresentados formam uma curva fechada.

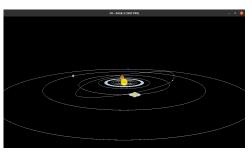
## 4 Alterações no Cenário Final

Em relação à fase anterior, a principal diferença no cenário é o facto de haver movimento dos modelos, em vez de estarem estáticos. Neste caso em específico, conseguimos ver os planetas a orbitar à volta do Sol, bem como as suas luas à volta do respetivo planeta.

Podemos, ainda, observar um cometa e a sua trajetória.



Cenário final - perto



Cenário final - longe

## 5 Conclusão e Trabalho Futuro

Concluída a terceira fase do projeto, achamos que fomos capazes de pôr em prática os conceitos de curvas e superfícies cúbicas, abordados nas aulas teóricas. Tivemos, ainda, que aplicar *VBOs* no nosso trabalho, algo que já tinha sido feito num guião duma aula prática.

Na próxima fase esperamos poder completar o Sistema Solar com iluminação e texturas. Mais uma vez, sentimos que mantivemos o código organizado, facilitando o trabalho futuro.