Computação Gráfica (3º ano de LCC)

3ª Fase

Relatório de Desenvolvimento

Grupo 18

Eduardo Pereira	Diogo Coelho	Pedro Oliveira	João Barbosa
(A70619)	(A100092)	(A97686)	(A100054)

27 de abril de 2025

Resumo

Este relatório descreve o desenvolvimento da terceira fase de um projeto, desenvolvido no âmbito da Unidade Curricular de Computação Gráfica, do 3º ano da Licenciatura em Ciências da Computação, na Universidade do Minho.

Este projeto consiste na implementação de duas grandes componentes: um **generator** e um **engine**, que em conjunto produzem figuras geométricas visíveis num espaço 3D.

O **generator** é responsável pela geração dos vértices relativos a cada figura, enquanto o **engine** trata da renderização dos objetos no espaço.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

No âmbito da Unidade Curricular de Computação Gráfica, foi-nos proposto, pelo docente, a implementação de um sistema capaz de desenhar figuras num espaço 3D.

Esta terceira fase do projeto tem como objetivo dar continuidade ao trabalho desenvolvido nas fases anteriores, introduzindo agora a implementação de `patches de Bezier`, curvas `Catmull-Rom` e ainda a conversão da forma de como os modelos são renderizados, até agora de forma imediata, para o uso de `VBO´s`. Nesta etapa evoluímos também o Sistema Solar de teste que passou de estático para dinámico. Conseguimos alcançar este passo, devido às alterações anteriormente citadas.

Neste relatório será apresentada a nossa interpretação do problema, assim como as abordagens adotadas ao longo da implementação deste projeto.

1.1 Estrutura do Relatório

De forma a facilitar a leitura e compreensão deste documento, nesta secção será explicada a sua estrutura e o conteúdo de cada um dos capítulos, resumido e explicado.

- **Capítulo 1** - Definição do Sistema, que consiste em: Introduzir e descrever a contextualização do projeto desenvolvido, assim como os objetivos a atingir com o mesmo.
- **Capítulo 2** - Análise e Especificação: Descrever informalmente o problema, indicando o que se espera ser possível fazer.
- **Capítulo 3** - Descrição da implementação do `Generator`.
- **Capítulo 4** - Descrição da implementação do `Engine`.
- **Capítulo 5** - Apresentação e explicação da implementação do Sistema Solar Dinámico.
- **Capítulo 6** - Instruções de Utilização: Instruções para compilar e executar os programas do projeto
- **Capítulo 7** - Testes: Neste capítulo são apresentados diversos testes realizados sobre o nosso programa
- **Capítulo 8** - Conclusão: Contém as últimas impressões acerca do projeto desenvolvido e oportunidades de trabalho futuro.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Nesta fase do projeto foi trabalhada tanta a parte do `Engine` tanto a parte do `Generator`. O `Engine` sofreu alterações para suportar a extensão dos elementos de translação e de rotação, que vai permitir por exemplo, a navegação de um objeto pelo trajeto de uma curva de `Catmull-Rom`.
Agora o `Engine` renderiza as figuras recorrendo a `VBO`, i.e , `Vertex Buffer Objects`, que vai permitir aumentar a performance e otimização do nosso renderizador. O `Generator` foi alterado para permitir `Patches de Bezier`, uma extensão das curvas de `Bezier`.
Tal como nas outras fases, o **Generator** gera os ficheiros com os vértices a serem utilizados pelo **Engine**.
As primitivas gráficas que temos até este ponto, são:

- **Plano**: Um quadrado no eixo **XZ**, centrado na origem, subdividido nas direções **X** e **Z**.
- **Caixa**: Requer **dimensão** e o número de **slices** por aresta, e está centrada na origem.
- **Esfera**: Requer um **raio**, **slices** e **stacks**, e está centrada na origem.
- **Cone**: Requer um **raio** para a base, **altura**, **slices** e **stacks**, e a base do cone está no eixo **XZ**.
- **Torus**: Novo modelo geométrico adicionado, definido por **dois raios** e o número de **rings** e **sides**.

O `Engine` foi ampliado para suportar animações temporizadas baseadas em curvas `Catmull-Rom`, com novas funcionalidades no ficheiro `XML` e integração de um módulo auxiliar (`catmullrom.cpp`) para cálculos matemáticos.
Ambos os programas foram desenvolvidos em `C++`.

Capítulo 3

Generator

3.1 Implementação do Suporte a Superfícies de Bezier

Nesta fase do projeto, foi adicionada ao `Generator` a capacidade de criar modelos baseados em `Patches` de `Bezier`, permitindo a geração de formas e figuras mais complexas a partir de um conjunto de pontos de controlo.

Esta nova funcionalidade vai permitir expandir o lote de figuras disponíveis para modelar no `Generator`, como por exemplo uma chaleira (`teapot`).

O ponto de partida para a geração das superfícies de Bezier é a leitura de um ficheiro de `patches`, fornecido à partida para esta fase, onde estão presentes o número de `patches` a processar, para cada `patch` uma lista de 16 índices que referenciam os pontos de controlo, e uma lista de todos os pontos de controlo, definidos cada um por 3 coordenadas (x,y,z).
O método para calcular a posição de cada ponto na superfície, foi o algoritmo de `De Casteljau`. O algoritmo foi aplicado de forma bidimensional, primeiro para interpolar ao longo de uma direção v, gerando uma curva intermédia, e depois ao longo de outra direção u, obtendo assim o ponto na superfície.
O nível de detalhe na superfície é controlado pelo parámetro de tesselação, que determina quantas subdivisões são feitas em cada direção do `patch`. Para cada célula da `grid` resultante, são gerados 2 triângulos. A implementação faz um pré-cálculo dos pontos da `grid` para otimizar o desempenho, evitando cálculos repetidos.
O resultado final da tesselação é escrita num ficheiro `.3d`, onde cada linha corresponde a um vértice de um triângulo, seguindo o formato padrão utilizado pelo `generator`.

Capítulo 4

Engine

O **Engine** foi mais uma vez a componente mais alterada nesta fase da implementação. Mais uma vez, esta processa os modelos tridimensionais a partir das especificações de um ficheiro `XML`, aplica as transformações esperadas e efetua o desenho da cena.
A renderização é realizada pela função **`renderScene()`**, que limpa o ecrã, configura a câmera, desenha os modelos em **wireframe** e troca os **buffers** para exibir uma cena nova.

4.1 Implementação de Curvas Catmull-Rom

As curvas Catmull-Rom foram implementadas para definir trajetórias contínuas, que vão servir para a simulação do sistema solar, como por exemplo, as órbitas dos planetas. A abordagem matemática baseia-se na interpolação de pontos de controle utilizando polinómios cúbicos. O ficheiro `catmull-rom.cpp` contém a matriz base `Catmull-Rom`, que define os coeficientes polinomiais para calcular posições e vetores tangentes ao longo da curva.

A função `getCatmullRomPoint` calcula um ponto específico num segmento da curva utilizando quatro pontos de controlo, enquanto que a função `getGlobalCatmullRomPoint` estende o cálculo para curvas fechadas, garantindo continuidade através do `wrap-around` dos índices, ou seja, conecta o último ponto ao primeiro, criando por isso um `loop`.

No contexto do XML, foi adicionado o atributo `time` em elementos do tipo `<translate>` permitindo definir o tempo total (em segundos) para percorrer uma trajetória.
Por exemplo, `` `<translate time="30"align="True»` `` especifica que o objeto completará o percurso em 30 segundos. Durante a renderização, o `Engine` calcula o tempo normalizado com base no tempo decorrido, utilizando a função `glutGet(GLUTELAPSEDTIME)`, e aplica a translação correspondente recorrendo à função `glTranslatef`.

O alinhamento de objetos em relação à curva é alcançado através do cálculo de vetores tangentes, derivadas, que definem a direção do movimento. Uma matriz de rotação é construída utilizando os vetores tangente, up global e produto cruzado, garantindo que o objeto se oriente corretamente durante a translação.

Foram adicionados também parámetros adicionais, tais como o `align` e `draw` que permitem, respetivamente, alinhar a direção do objeto à direção tangente da curva e ocultar a renderização da curva no motor gráfico.

4.2 Conversão do modo de Renderização: Imediato para o uso de VBO's

Para otimizar o desempenho, os modelos geométricos são armazenados em `VBOs` na `GPU`. O processo inicia-se com a função `loadModel`, que lê os vértices de um ficheiro (ex: $sphere\_10\_20\_20.3d$) e envia-os para a memória da `GPU` com recurso à função `glBufferData`. Cada modelo é associado a um identificador único (`vbo`), gerado pela função `glGenBuffers`. Durante a renderização, os VBOs são ativados pela função `glBindBuffer`, e os vértices são desenhados utilizando a função `glDrawArrays`. Esta abordagem reduz a sobrecarga da comunicação entre o processador e a placa gráfica, especialmente em cenas mais complexas como o sistema solar, onde múltiplas figuras são renderizadas simultaneamente.

4.3 Hierarquia de Transformações e da Cena

A estrutura hierárquica definida no `XML` é processada recursivamente pela função `trenderGroup`. Cada grupo pode conter transformações geométricas (translação, rotação, escala) e modelos, com transformações aplicadas na ordem específica. Por exemplo, um planeta orbita o Sol (translação), gira sobre si mesmo (rotação) e pode ter luas como objetos filhos, que herdam as suas transformações. Para translações temporizadas, o motor verifica a presença do atributo `time` e calcula a posição atual na curva Catmull-Rom. Se o parámetro (align=”True”), então a matriz de rotação é atualizada para alinhar o objeto à tangente da curva.
As transformações são acumuladas pelas funções `glPushMatrix` e `glPopMatrix`, funcionando como uma `stack`, garantindo assim que as operações em objetos filhos não afetem os pais.
Exemplo: Sistema Solar Dinámico No ficheiro ‘dynamicsystem.xml‘, cada planeta é definido como um grupo com:

1. `Translação temporizada`: Uma curva `Catmull-Rom` com oito pontos, simulando órbitas.
2. `Rotação contínua`: Com recurso ao parámetro `<rotate time="X»`, onde ‘X‘ é o período de rotação em segundos.
3. `Hierarquia de modelos`: Luas são grupos filhos com trajetórias e transformações próprias.

O cometa, por exemplo, utiliza uma curva aberta com quatro pontos e um modelo de `patch` de `Bézier` ($bezier\_10.3d$), o que implica diversos requisitos pedidos nesta fase do projeto, demonstrando a flexibilidade do sistema para diferentes tipos de trajetórias e geometrias.

Capítulo 5

Sistema Solar Dinámico

No enunciado deste projeto, também foi pedida uma demonstração de um Sistema Solar dinámico, com os vários planetas, bem como as suas luas, expandindo do modelo estático.

Para este efeito, foi desenvolvido um ficheiro XML com as informações necessárias ao desenho:

- **Transformações geométricas:** Translações, rotações ou escalas
- **Modelos utilizados:**
  - **Sol:** sphere_10_20_20.3d
  - **Planetas:** sphere_10_20_20.3d
  - **Luas:** sphere_10_20_20.3d
  - **Anel de Saturno:** torus_15_14_20_20.3d

5.1 XML

No contexto deste projeto, foi desenvolvido um ficheiro `XML` que descreve um Sistema Solar dinámico, onde cada planeta e as respetivas luas executam movimentos ao longo de curvas `Catmull-Rom`. A estrutura hierárquica do `XML` reflete a organização natural do sistema solar: o Sol ocupa o centro da cena, servindo de nó pai para todos os planetas (filhos) , e cada planeta pode ter as suas luas como nós-filho. As órbitas são parametrizadas através do atributo `time`, que define o período de translação de cada corpo , enquanto o parâmetro `(align="True")` garante que os objetos se mantêm orientados segundo a direção do movimento. Para maior clareza visual, é ainda possível ocultar a representação gráfica de certas órbitas com o atributo `(draw="false")`.

5.2 Exemplo Sistema Solar

Em baixo está presente uma imagem da nossa interpretação do Sistema Solar pedido. Na pasta source do código está presente um pequeno vídeo que demonstra as translações e rotações dos planetas ao longo das órbitas definidas pelas curvas de `Catmull-Rom`.

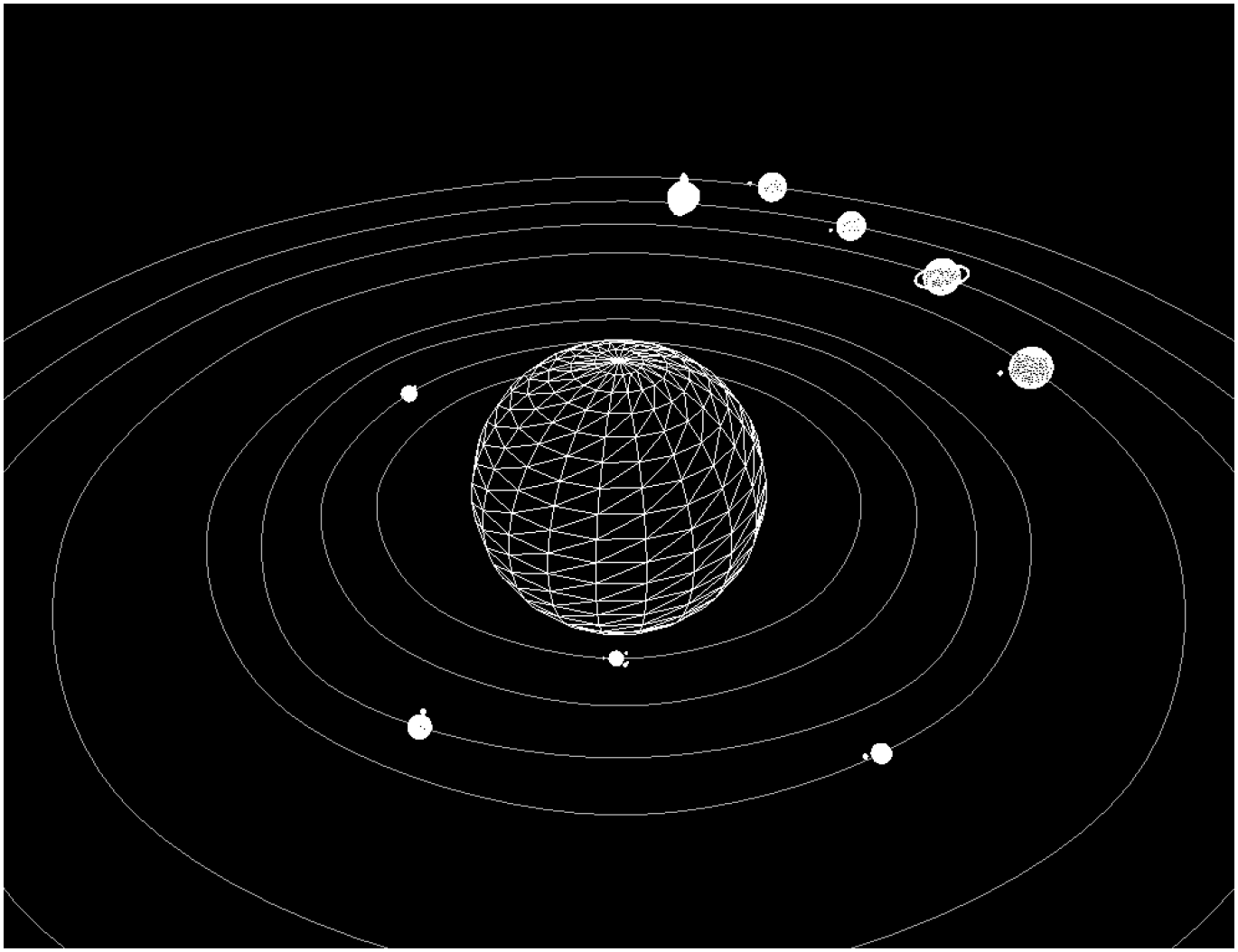

Figura 5.1: Exemplo Sistema Solar Dinámico

Capítulo 6

Instruções de Utilização

Para compilar e executar o nosso programa, existem duas formas:

6.1 Manualmente

1. Posicionar-se na pasta `build` com o comando:

   ```
   cd build
   ```

2. Executar os seguintes comandos:

   ```
   cmake ..
   ```

   seguido de

   ```
   make
   ```

3. Posicionar-se na pasta `generator` e executar o seguinte comando:

   ```
   ./generator nome_figura <parametros> nome_figura.3d
   ```

4. Sair da pasta `generator` com o comando:

   ```
   cd ..
   ```

5. Posicionar-se na pasta `build` e executar o seguinte comando:

   ```
   ./engine ../engine/configs/test_file_pretendido.xml
   ```

6.2 Script

Para simplificar o processo, criámos um **script** que executa automaticamente todas as instruções mencionadas. Para executar o **script**, siga os seguintes passos:

1. Tornar o **script** executável com o comando:

```
chmod +x run.sh
```

2. Executar o **script** com:

```
./run.sh
```

Após executar o **script**, será apresentado um menu que permite realizar diversas operações:

- Na primeira utilização, recomenda-se executar um `make clean` e recompilar o projeto. Para isso, seleciona-se a opção **1**.
- Criar objetos 3D utilizando a opção **2**.
- Visualizar os objetos já criados com a opção **3**.
- Executar os ficheiros de teste através da opção **4**.

Dentro da pasta do projeto, incluímos uma pequena demonstração para facilitar a compreensão do processo.

Capítulo 7

Testes

Nesta secção, serão apresentados alguns exemplos com as figuras renderizadas conforme as configurações dos arquivos de teste no formato **.XML**.

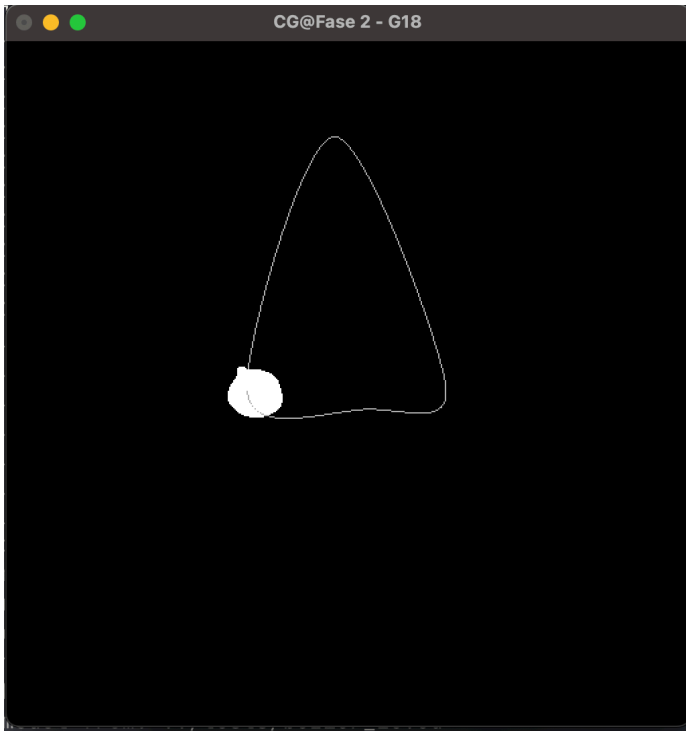

Figura 7.1: Teste 1

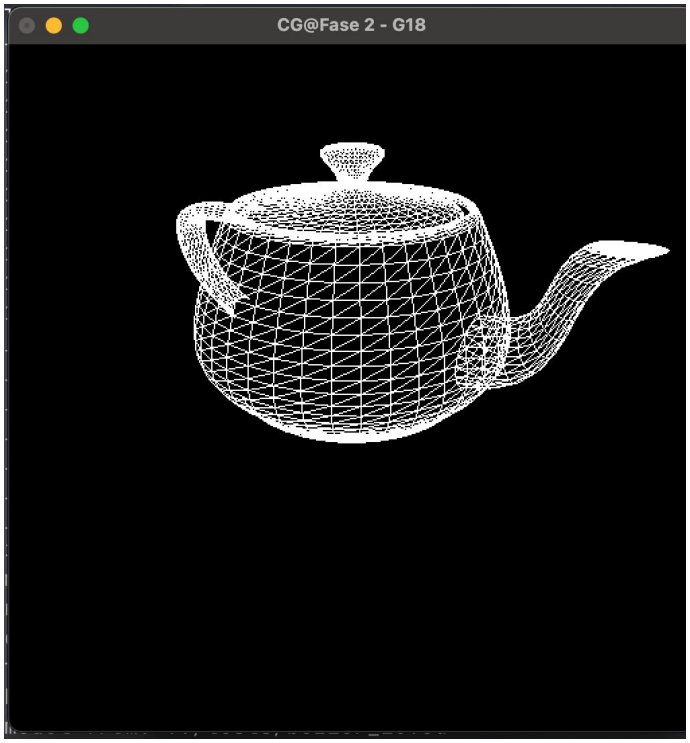

Figura 7.2: Teste 2

Capítulo 8

Conclusão

A terceira fase deste projeto representou um avanço técnico significativo em relação às anteriores, ao introduzir novas funcionalidades no nosso motor gráfico, elevando a complexidade e o realismo das cenas geradas.

A implementação das superfícies de Bézier permitiu-nos criar geometrias mais suaves e detalhadas, como exemplificado no `teapot` requirido nos dois ficheiros de teste fornecidos. Consideramos esta funcionalidade um marco importante, pois trouxe um novo paradigma na construção dos modelos: em vez de apenas figuras primitivas, passámos a gerar superfícies complexas a partir de controlos geométricos.

Outro aspeto de grande relevância foi a extensão das transformações de translação e rotação para suportar movimentos ao longo de curvas Catmull-Rom e rotações contínuas no tempo. Esta evolução foi fundamental para dar vida ao nosso sistema solar: conseguimos, por exemplo, representar trajetórias dinâmicas como a órbita do cometa, de forma natural e fluída, e fazer rotações mais realistas dos planetas.

A obrigatoriedade de usar Vertex Buffer Objects (VBOs) em vez do modo imediato também trouxe ganhos evidentes em termos de desempenho do motor gráfico.

Sentimos que atingimos com sucesso os principais objetivos propostos para esta fase, conseguindo integrar de forma funcional as novas `features`. Contudo, reconhecemos que existe espaço para melhorias, nomeadamente na organização do nosso código. Com o aumento da complexidade, em especial pela gestão de VBOs e pelas novas transformações temporizadas, tornou-se ainda mais evidente a necessidade de modularizar melhor a implementação do nosso código.

Em síntese, esta fase não só expandiu as capacidades técnicas do projeto, como também consolidou muitos conceitos fundamentais.