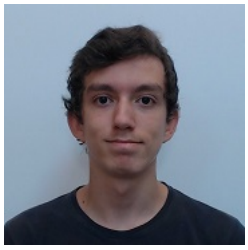
Trabalho Prático - fase 3
Curvas, Superficies cúbicas e VBOs

Catarina Cruz
A84011

Jorge Mota
A85272



Maria Silva
A83840

Mariana Marques
A85171

06 de Março de 2020

Universidade do Minho

Conteúdo

1 Introdução

Para esta terceira fase do projeto, concentramo-nos em implementar três funcionalidades principais.

A primeira tarefa realizada foi trocar todas as formas de modelos realizadas anteriormente para VBO's, com o intuito de otimizar o projeto realizado até ao momento.

A segunda funcionalidade implementada foi a geração de um novo tipo de primitiva baseado em Bezier patches, segundo uma sintaxe própria fornecida e escrita num ficheiro passado como argumento.

Por último, criamos ainda um tipo de translate e rotate animado que realiza a transformação durante um intervalo de tempo. O translate animado segue uma animação de acordo com os pontos descritos por uma curva de Catmull-Rom.

Depois destas funcionalidades implementadas inseriu-se um cometa com uma trajetória no sistema solar. Este cometa foi construído a partir do gerador usando patches de Bezier (teapot).

2 VBOs

Sendo VBO's uma das funcionalidades requeridas e uma grande melhoria para o sistema em geral, todos os modelos desenhados passaram a utilizá-los.

Para economizar o código e facilitar a utilização de vbo's criou-se um módulo **vbo** para facilitar a não só a sua manipulação com também reaproveitar o módulo **color**.

Este módulo define a class `vbo`, o construtor e método de desenho `draw()`.

3 Adições ao gerador

3.1 Patches de Bezier

Para ser possível gerar modelos mais complexos a partir de superfícies cúbicas teve de se ter em conta um novo tipo de ficheiros, aonde são definidos os patches que são utilizados. Estes ficheiros (`name.patch`) seguem uma sintaxe própria, descrita de seguida.

A primeira linha contém o número de patches. As linhas seguintes, uma para cada patch, contêm o índice dos pontos de controlo, havendo 16 pontos para cada patch. A linha posterior possui o número de pontos de controlo e por último os próprios pontos de controlo, um por linha.

O gerador lê o ficheiro `.patch`, extrai os dados e começa o processo de criação dos vértices.

De seguida, aplicando o que foi aprendido nas aulas teóricas e acompanhando os devidos apontamentos disponibilizados com as fórmulas, o gerador, para cada patch, vai criando os pontos (segundo um fator de aproximação) para os triângulos do modelo.

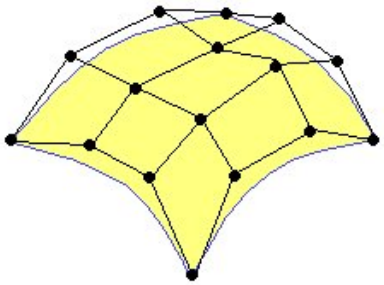


Figura 1: Uma figura simbólica de uma superficie cúbica de Bezier.

Para cada patch é utilizada a seguinte formula:

$$p(u,v) = [u^3 \quad u^2 \quad u \quad 1]M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

Figura 2: Formula de cálculo de um ponto na superficie cúbica

P00 a P33 identifica os 16 pontos necessários para calcular os pontos dessa superficie cúbica

No fim da geração de todo o modelo, os vértices são guardados em ficheiro, tal como é feito para as outras primitivas.

Para mostrar o efeito da geração de modelos com patches de Bezier temos as seguintes imagens:

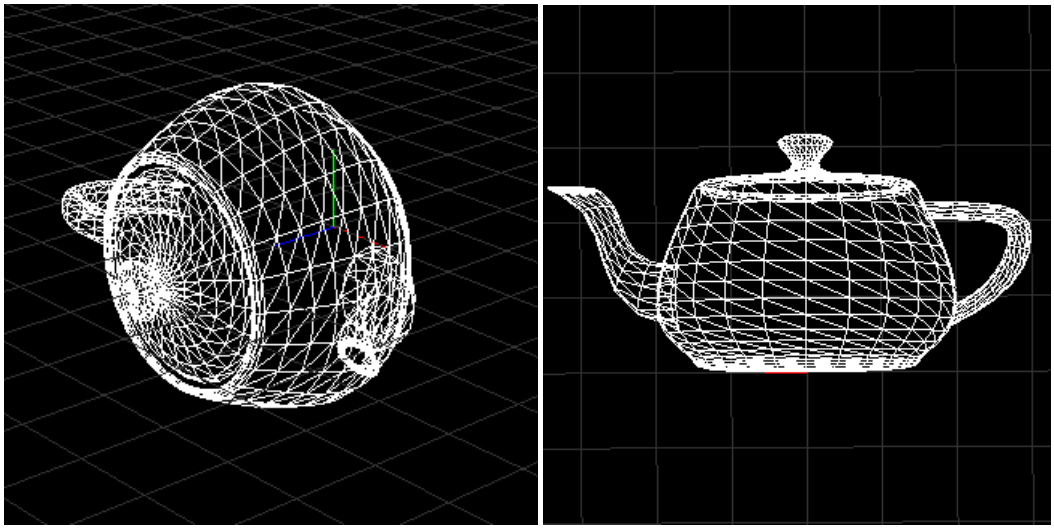


Figura 3: Teapot gerado de lado a partir de `teapot.patch`

Nota: Este teapot foi gerado com um fator de aproximação de 6

3.2 Torus

Esta primitiva apenas será utilizada numa fase mais avançada do projeto
A geração desta primitiva foi adicionada como funcionalidade extra e será usada para criar cenários com diversas geometrias com o intuito de ganharem

mais detalhe.

A forma como os vértices dos triângulos são calculados tem semelhanças ao processo feito pela geração do cilindro.

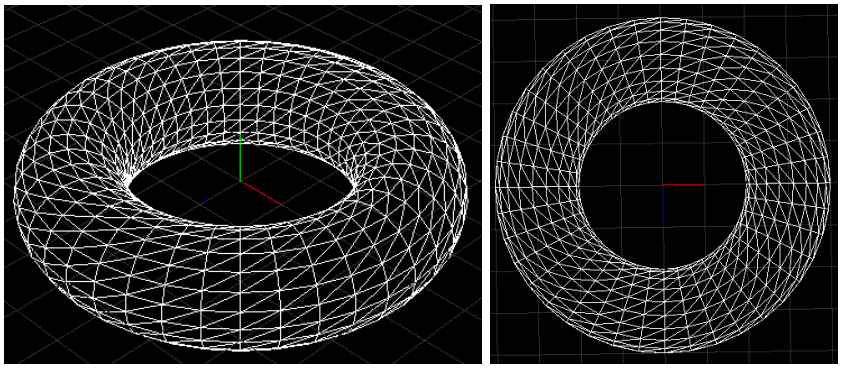


Figura 4: Torus gerado com raio 4, raio cilindrico 1, slices 40 e slices cilindricas 20

3.3 Ring

Para criar os anéis de saturno sentimos necessidade de criar uma primitiva que produz discos.

A forma como os vértices dos triângulos são calculados, tal como no torus, é muito semelhante ao processo feito pela geração do cilindro.

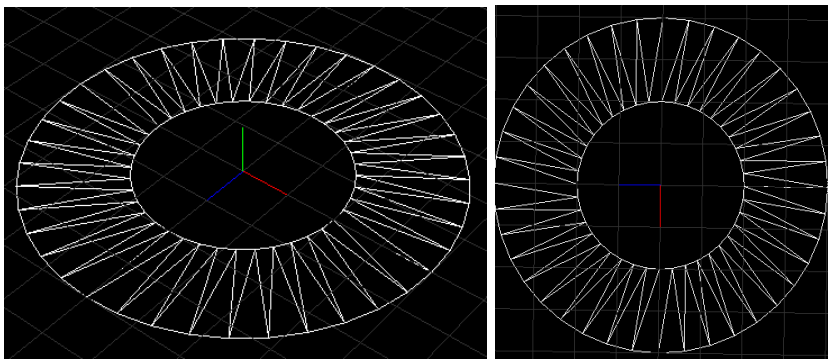


Figura 5: Anel gerado com raio interior 2, raio exterior 4, slices 40

4 Transformações em tempo

É um tópico dedicado para descrever a implementação das transformações dinâmicas tendo em conta o tempo.

4.1 Translate em tempo

Para realizar translações suaves num intervalo de tempo foi necessário implementar translações com curvas. Neste caso seguimos a implementação por curvas de Catmul-Rom.

Simplificando o funcionamento destas curvas, para descrever uma trajetória de P1 a P2 seleciona-se 4 pontos. Dois desses pontos (P0 e P4) são pontos para aproximar mais adequadamente a curva entre P1 e P2:

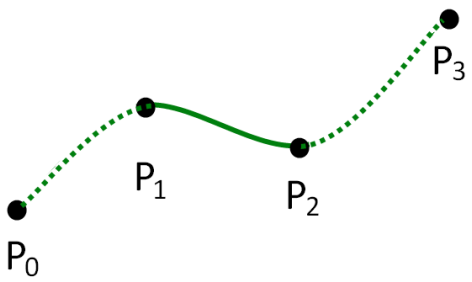


Figura 6: Uma figura simbólica de uma curva de Catmul-Rom

Para obter os pontos de P1 a P2 usa-se a variável `t` de domínio [0,1]

Como queremos que a translação / trajetória do objeto siga um caminho não limitado a apenas um ponto de início e fim, usa-se uma estratégia de seleção de pontos para o cálculo da curva.
Esta estratégia tem por base o código exemplo fornecido num dos guiões das aulas práticas embora tenha algumas alterações para simplificação dos cálculos e suporte de 2 tipos de seleção de pontos

Esses dois tipos de seleção de pontos são:

- Seleção de pontos para uma trajetória fechada.
- Seleção de pontos para uma trajetória aberta.

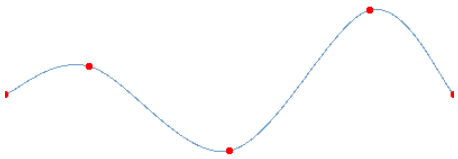


Figura 7: Exemplo de trajetória aberta

Para uma trajetória aberta, assumindo , por exemplo , 5 pontos P0, P1, P2, P3 e P4 em que queremos passar por **todos** os pontos descritos, a seleção de pontos para o cálculo da curva vai ser:

- `[ P0, P0, P1, P2 ]`
- `[ P0, P1, P2, P3 ]`
- `[ P1, P2, P3, P4 ]`
- `[ P2, P3, P4, P4 ]`

o `t` tem um domínio "virtual"de `[0,N-1]` e este valor é normalizado para `[0,1]` quando é passado para apenas os 4 pontos selecionados de cada vez

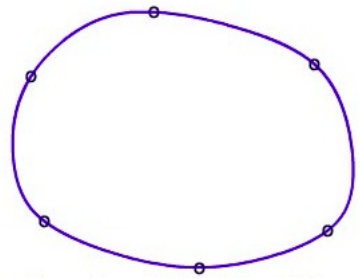


Figura 8: Exemplo de trajetória fechada

Para uma trajetória fechada, assumindo , por exemplo , 6 pontos P0, P1, P2, P3, P4 e P5 em que queremos passar por **todos** os pontos descritos e ainda voltar ao ponto inicial, a seleção de pontos para o cálculo da curva vai ser:

- `[ P5, P0, P1, P2 ]`
- `[ P0, P1, P2, P3 ]`
- `[ P1, P2, P3, P4 ]`
- `[ P2, P3, P4, P5 ]`

- `[ P3, P4, P5, P0 ]`
- `[ P4, P5, P0, P1 ]`

O `t` tem um domínio "virtual"de `[0,N]` e este valor é normalizado para `[0,1]` quando é passado para apenas os 4 pontos selecionados de cada vez

Para cada 4 pontos selecionados é utilizada uma forma simplificada da seguinte fórmula:

$$[x(u) \quad y(u) \quad z(u)] = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_o \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Figura 9: Formula de cálculo de um ponto na curva Catmul-Rom

Nota: O formato xml para este tipo de transformações requer passar como argumento um tempo em segundos que serve para descrever em quanto tempo o objeto deve fazer essa transformação, quando uma diferença de tempo é passada como argumento para recalcular a nova posição do objeto na trajétória esse valor passa por uma normalização para o devido `t` requerido.

4.2 Rotate em tempo

Em contraste com a translação num intervalo de tempo, a rotação não precisa de nenhum método matemático sofisticado.

Para um tempo `time` (em segundos) o objeto roda em torno do(s) eixo(s) especificado(s) em 360º.

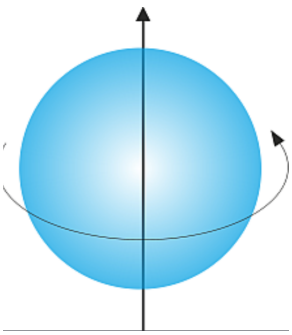


Figura 10: Imagem ilustrativa de como um objeto roda em torno do eixo especificado

5 Funcionalidades adicionais

5.1 Melhorias do formato XML

Com o progresso no parser XML, fomos notando a necessidade de acrescentar algumas funcionalidades que ajudam na construção de cenários.

5.1.1 Parâmetros costumizáveis do timed translate

Os parâmetros que adicionamos foram:

- `scaling`

O valor deste campo é usado para multiplicar por todos os pontos da trajetória pretendida. Esta funcionalidade é desejável para caso não se queira especificar várias trajétórias semelhante mas que mudam a escala, como por exemplo o movimento dos planetas.
Ex: Point = Point * scaling

- `closed`

Valor que pode ser "true"ou "false"se queremos a trajetória do objeto fechada ou aberta (pormenores descritos no tópico: 4.1)
Exemplo de XML:

```
<translate time="2" scaling="8" closed="true">
    <point X="1" Y="0" Z="0"/>
    <point X="0.7071" Y="0" Z="-0.7071"/>
    <point X="0" Y="0" Z="-1"/>
    <point X="-0.7071" Y="0" Z="-0.7071"/>
    <point X="-1" Y="0" Z="0"/>
    <point X="-0.7071" Y="0" Z="0.7071"/>
    <point X="0" Y="0" Z="1"/>
    <point X="0.7071" Y="0" Z="0.7071"/>
</translate>
```

Nota: Como scaling e closed são argumentos opcionais, os valores default são 1.0 e "true"respetivamente. Portanto não é necessário adicionar o closed = "true"no exemplo acima.

5.1.2 Keywords reservadas para `color`

Para cada modelo é possível especificar uma cor, como já tinha sido explicado na fase 2 do trabalho. Para esta fase adicionamos outra keyword 'rgb' que dá uma cor especial aos modelos, essa cor especial é no fundo uma cor que está sempre a variar pelo espetro de cores.

De seguida apresentam-se 3 imagens para se ter uma ideia geral da representação da animação feita:

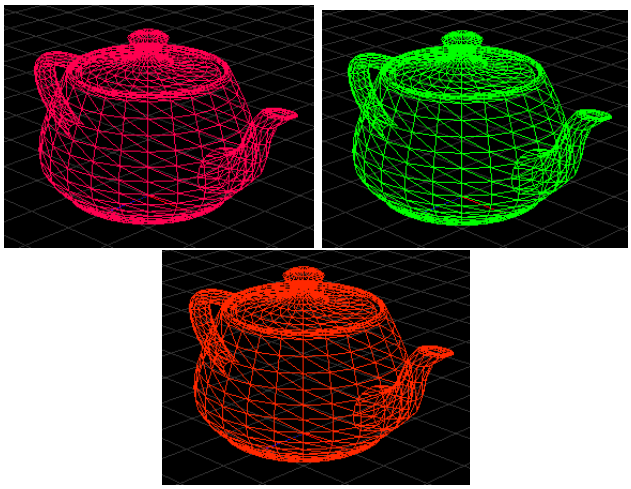


Figura 11: Captura de ecrãs do mesmo cenário em tempos diferentes

`Nota:` No outro relatório já tinhamos especificado outra keyword que era o 'random' que dava uma cor aleatória ao modelo.

5.1.3 XML loader errors

A fim de ter mensagens de erro relevantes, o loader de cenários agora imprime o texto que for necessário para se compreender com mais clareza o erro no ficheiro que se está a tentar ler.

5.1.4 Atributo `command`

Outra funcionalidade que achavamos crucial adicionar é a habilidade de gerar modelos dentro do ficheiro XML sem ter a necessidade de invocar o gerador.

Para realizar esta funcionalidade acrescentamos uma alternativa ao atributo `file` que é o `command`. O `command` tem como valor uma string de um comando gerador de primitivas, a sintaxe deste comando é semelhante à do gerador visto que usa diretamente as mesmas funções que este. Com isto, aproveita-se os headers do gerador e ainda se evita a dependência de ficheiros `.3d` pois os vértices são gerados durante o carregamento do cenário.

Comandos

- `cylinder [radius] [height] [slices]`
- `sphere [radius] [slices] [stacks]`
- `plane [width]`
- `box [width_x] [width_y] [width_z] [divisions]`
- `cone [radius] [height] [slices]`
- `ring [inner_radius] [outter_radius] [slices]`
- `torus [radius] [cyl_radius] [slices] [cyl_slices]`
- `patch [patch_file] [detail]`

Exemplo:
Para o XML seguinte:

```
<scene>
    <camera posX="15" posY="15" posZ="15"/>
    <group>
        <model command="sphere 2 20 20"/>
    </group>
</scene>
```

É gerado:

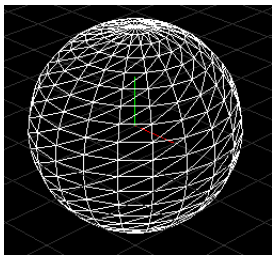


Figura 12: Esfera gerada pelo XML descrito em cima

5.1.5 Controlo sobre o tempo

A fim de controlar as animações inserimos controlos que modificam o valor da diferença de tempo do frame anterior com o frame atual, multiplicando-o por um valor de tempo que é ajustado de acordo com o pressionar dos controles.

Para aumentar o tempo usa-se a tecla 'E' e para diminuir o 'Q'.

Ao diminuir, o valor de tempo pode passar a ser inferior a 0, o que permite viagens no tempo...

5.2 Modo debug

Este "modo debug"é uma continuação e modificação do funcionamendo do renderizador em fases passadas.

Quando é precionado o 'Space' em vez de apenas mostrar o cenário carregado do ficheiro XML, todas as trajetorias aparecem traçadas como uma linha e ao invés dos objetos sólidos é desenhado a 'Mesh' dos mesmos. Neste modo também é desenhado uma grelha sobre o plano xOz para o utilizador ter uma percepção do espaço/distâncias

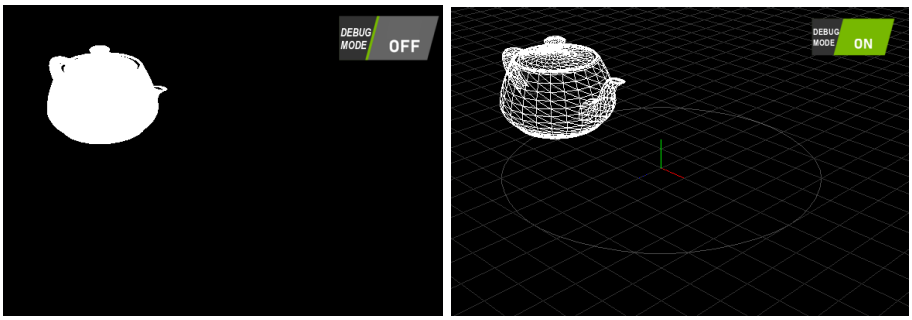


Figura 13:

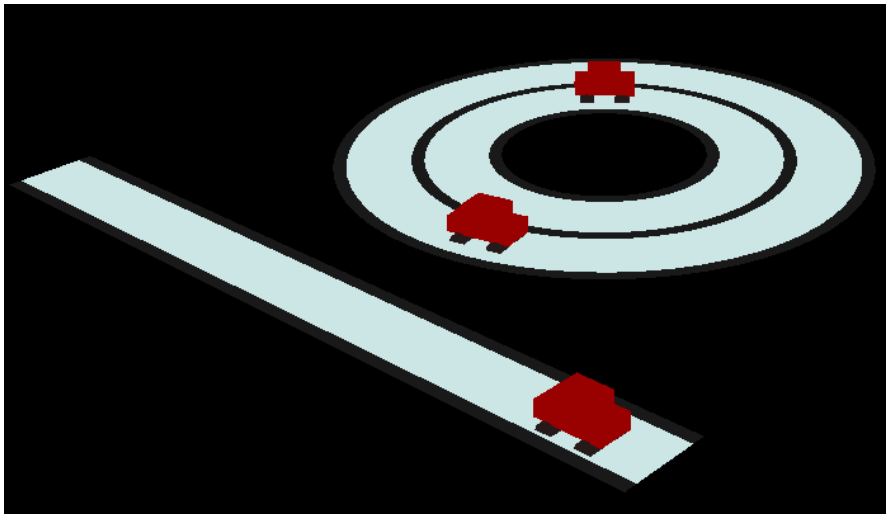
5.3 Controles

Em baixo apresenta-se os controles do teclado para esta fase:

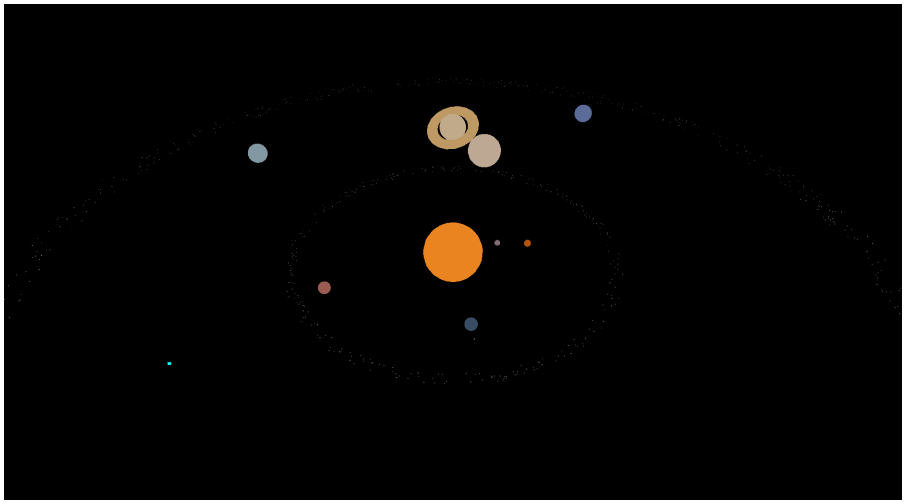
- '`Arrow Keys`' Movimento da posição da camera relativa ao ponto fixo (origem)
- '`Space`' Debug Mode
- '`Q`' Diminuir o valor do tempo
- '`E`' Aumentar o valor do tempo
- '`+`' Zoom in
- '`-`' Zoom in

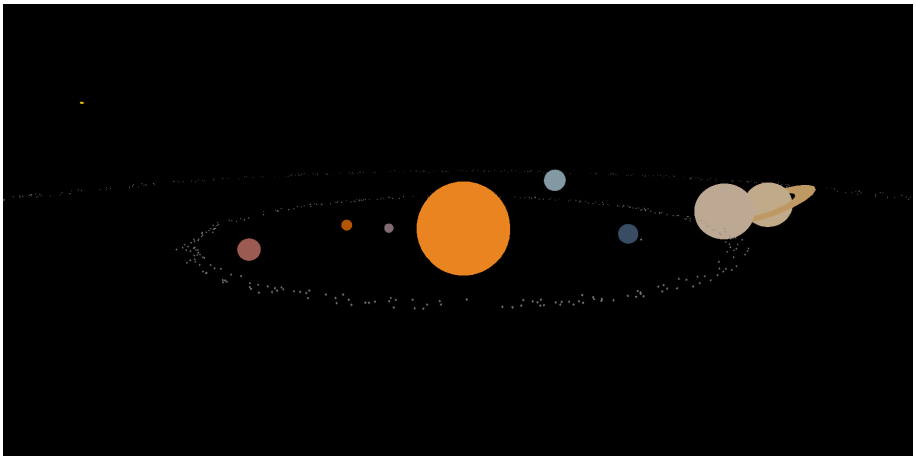
6 Cenários

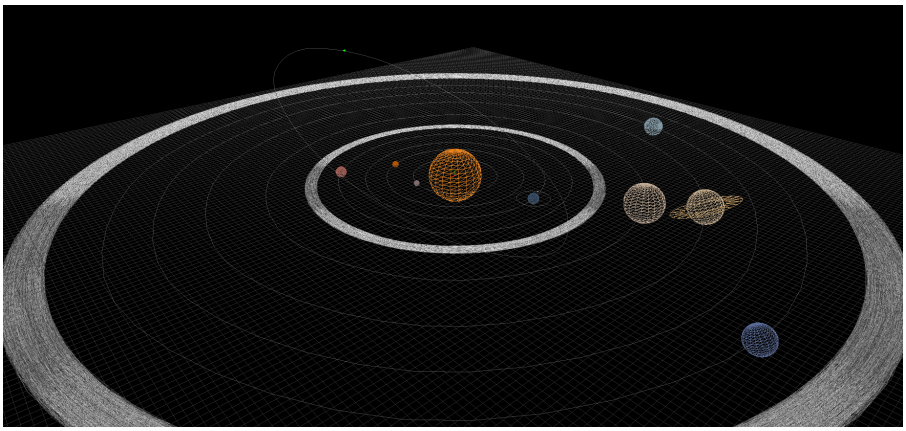
6.1 Tráfego



6.2 Sistema solar







7 Conclusão

Com esta fase deu para consolidar melhor as curvas de Catmul-Rom e as superficies bicúbicas de Bezier. Seguindo a evolução progressiva deste projeto, podemos refletir sobre as funcionalidades que queremos para futuras fases, algumas destas funcionalidades incluem (para além dos objetivos propostos pelo enunciado):

- **Skybox** - Uma skybox texturizada

- **Free Camera** - Vai ser o modo de navegação pelo espaço livre

- **Third Person Camera Movement** - Em vez da câmera ficar fixa na origem, vai poder ter a perspetiva em 3ª pessoa mas em qualquer ponto do plano xOz (em debug mode vai aparecer um ponto que indica onde é que a câmera está a olhar)

- **Doctor Who TARDIS scene** - Ao falar de "viagens no tempo"em 5.1.5 despertou o nosso interesse de aproveitar as funcionalidades do projeto para fazer um cenário apróximado da "cabine"da série

- **Index Buffers** - Usar uma estratégia já planeada para introduzir index buffers e aproveitar os vértices repetidos