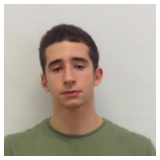
Computação Gráfica
Trabalho Prático Fase III
MIEI
Grupo 27

Gonçalo Pereira

A74413

António Silva



A73827

André Diogo

A75505

1 de Maio de 2017

Conteúdo

Capítulo 1

Introdução

Esta terceira fase consiste na leitura e rendering de cenas hierárquicas com translações e rotações animadas (as translações através de curvas de catmull-rom e as rotações através de simples rotações em função do tempo) a partir de um ficheiro *XML* com etiquetas específicas para o efeito e de modelos gerados externamente, agora com suporte para criação de modelos a partir de superfícies cúbicas de bezier (num formato específico para o efeito .patch). Usar-se-á para esta fase o mesmo formato de modelos utilizado nas fases anterior.

Para cumprir estas novas metas é necessário expandir o motor codificado aquando da segunda fase para ler e guardar pontos das curvas e registar o tempo global em cada frame, definindo estruturas de dados úteis para o efeito e algoritmos eficazes para tratar do rendering destes dados.

Para o mesmo efeito é necessário também expandir o programa gerador de modelos para reconhecer e tratar ficheiros com superfícies de bezier neles codificados.

Nas seguintes secções detalhar-se-á o processo para expandir o motor e o gerador e demonstrar-se-á o motor em funcionamento com uma cena modelada do sistema solar.

Capítulo 2

Motor de renderização

2.1 Parsing de XML

Agora para esta fase, estando presentes variações em algumas etiquetas, nomeadamente nas etiquetas *translate* e *rotate*, teve-se a necessidade de para cada grupo guardar um novo componente, responsável por guardar os pontos dados para a translação: *AnimationComponent*.

Tal sucede da necessidade de guardar agora um conjunto de pontos numa curva para efetuar uma animação segundo a curva de *Catmull-Rom* definida por esses pontos e fazê-lo num determinado intervalo de tempo.

A segunda variação dá-se na etiqueta de *rotate*, em que pode aparecer um atributo de tempo, em substituição de um ângulo fixo, que determina o tempo até efetuar uma rotação de 360º em torno dos eixos especificados.

Mais à frente detalhar-se-à como lidar com estas variações fazendo algumas alterações no nosso algoritmo de parsing e quais as alterações às estruturas de dados para lidar com estas variações.

2.2 Modelos

Os modelos são tratados da mesma forma referida no relatório da fase anterior. São guardados num dicionário de forma a evitar duplicação e garante-se o *binding* de cada modelo único a um *buffer* do *OpenGL* de modo a usufruir de *VBOs* em vez do uso do modo immediato de desenho de triângulos imediatamente antes de iniciar o ciclo de desenho do *GLUT*.

2.3 Grupos

Em cada *GroupComponent* associado a um grupo na cena hierárquica, devido às variações introduzidas nas etiquetas de *rotate* e *translate*, podem ocorrer novas combinações diferentes de ordem de operações aplicadas porque associa-se agora a estas duas etiquetas duas novas possíveis operações: uma translação segundo uma curva (ANT) e uma rotação em função do tempo (ANR), valores novos na enumeração de operações possíveis, em acréscimo às anteriores: uma operação nula (ID), uma translação (TR), uma rotação(RT) e uma escala(SC). Quando se efetua o render basta iterar pelo vetor de ordem, comparar cada operação com uma das 6 possíveis. No caso das novas duas detalhar-se-à na secção dos algoritmos a forma de as tratar.

2.4 Vista geral das estruturas de dados

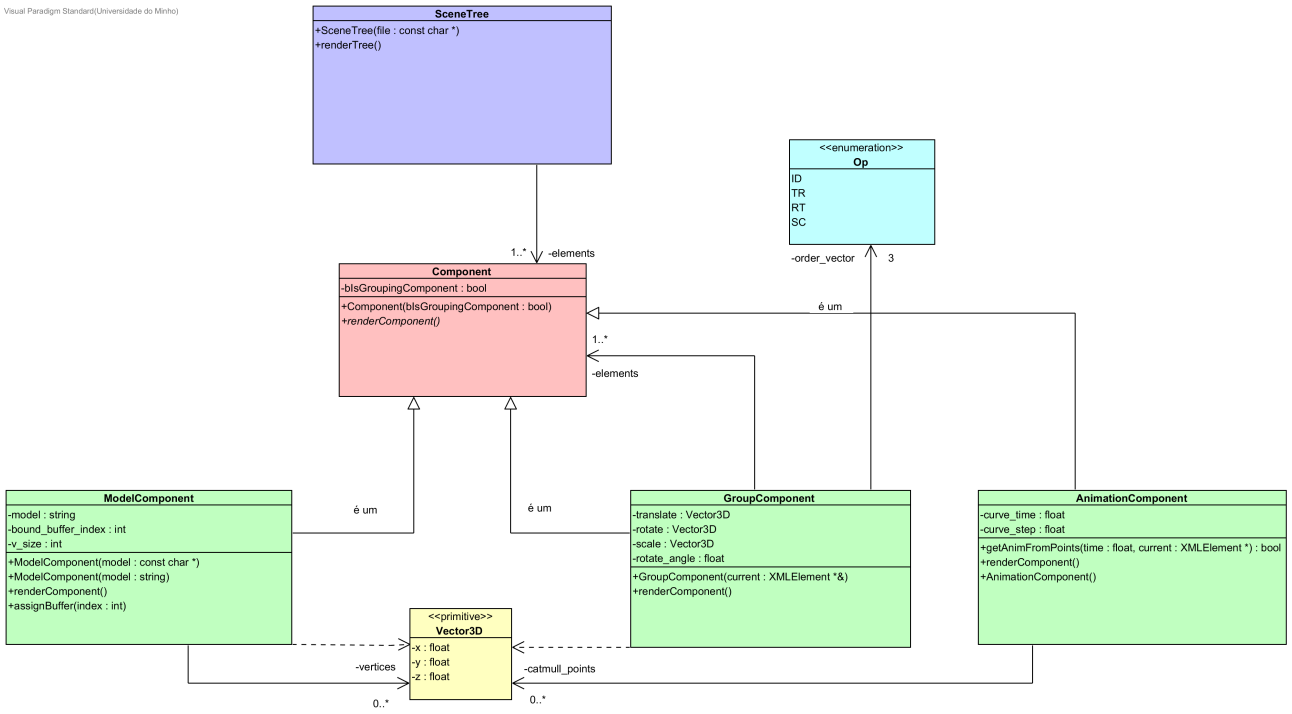


Figura 2.1: Diagrama de classes e estruturas de dados relevantes do motor.

Capítulo 3

Gerador a partir de ficheiros *patch*

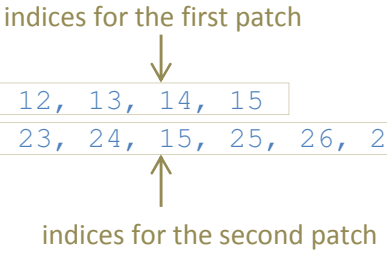
Para tratar o novo caso de geração de um modelo a partir de superfícies cúbicas de *bezier* adicionou-se ao já construído gerador uma nova opção *surface* para gerar um modelo *.3d* que recebe como parâmetro um ficheiro *.patch*. Estes ficheiros têm o seguinte formato:

Patch file format

Text file which contains the description of a set of Bezier patches. The first line contains the number of patches. The following lines, one for each patch, contain the indices of the control points (16 for each patch). The next line contains the number of control points, and afterwards the control points themselves, one per line.

Example:

```
2 <- number of patches
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
3, 16, 17, 18, 7, 19, 20, 21, 11, 22, 23, 24, 15, 25, 26, 27
28 <- number of control points
1.4, 0, 2.4 <- control point 0
1.4, -0.784, 2.4 <- control point 1
0.784, -1.4, 2.4 <- control point 2
0, -1.4, 2.4
1.3375, 0, 2.53125
1.3375, -0.749, 2.53125
0.749, -1.3375, 2.53125
0, -1.3375, 2.53125
1.4375, 0, 2.53125
1.4375, -0.805, 2.53125
0.805, -1.4375, 2.53125
0, -1.4375, 2.53125
1.5, 0, 2.4
1.5, -0.84, 2.4
0.84, -1.5, 2.4
0, -1.5, 2.4
-0.784, -1.4, 2.4
-1.4, -0.784, 2.4
-1.4, 0, 2.4
-0.749, -1.3375, 2.53125
-1.3375, -0.749, 2.53125
-1.3375, 0, 2.53125
-0.805, -1.4375, 2.53125
-1.4375, -0.805, 2.53125
-1.4375, 0, 2.53125
-0.84, -1.5, 2.4 <- control point 26
-1.5, -0.84, 2.4 <- control point 27
```



Face a este formato torna-se claro que aquando da geração de um modelo com esta opção é necessário ler este ficheiro e produzir estruturas de dados adequadas para guardar a informação nele embebido.

3.1 Estruturas de Dados

Para tal começa-se por guardar o número de *patches* e criar uma vetor dos indices todos seguidos *patch* a *patch* de tamanho dessasseis, número de indíces por *patch*, por número de *patches*. Guarda-se em seguida o número de pontos de controlo e um vetor com os pontos de controlo todos seguidos, de tamanho três, número de coordenadas por ponto, por número de pontos de controlo.

Guarda-se também um vetor v, com todos os vetores U/V (são iguais), usados no cálculo dos vértices segundo a fórmula de cálculo de um ponto numa superfície de bezier em função de dois pontos (u,v), a usar nos triângulos finais, tantos mais quanto o nível de tesselação.

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

Figura 3.1: Fórmula de cálculo de uma coordenada de um vértice em função de dois pontos u e v.

Finalmente guardam-se a matriz de *bezier* (constante e global ao programa para fácil acesso) e uma matriz onde se carregam os pontos de controlo de um *patch*, apenas uma pois esta é reutilizada *patch* a *patch*.

Para calcular os triângulos para cada *patch* aplicou-se um algoritmo sem memória, que calcula 4 vértices de cada vez em função de um vetor U, vetor U+1, vetor V e vetor V+1, tirados do vetor v, e em seguida imprime para o ficheiro .3d os vértices de modo a descrever os dois triângulos no quadrado formado por esses 4 vértices.

Capítulo 4

Algoritmos

4.1 Parsing de XML

Surgem algumas alterações ao algoritmo de tratamento de um elemento XML para lidar com a introdução das variações às duas etiquetas *translate* e *rotate*.

```
Algoritmo de tratar_elemento_XML dentro de um grupo:

|
|   Se elem_XML_currente é um grupo:
|
|   |
|   |   cria_novo_grupo
|   |
|
|   Se elem_XML_currente tem modelos e ainda
|   não se trataram modelos:
|
|   |
|   |   tratar_modelos
|   |
|
|   Se elem_XML_currente é uma translação e ainda
|   não se registou nenhuma:
|
|   |
|   |   regista_translação
|   |
|
|   Se elem_XML_currente é uma rotação e ainda
|   não se registou nenhuma:
|
|   |
|   |   regista_rotação
|   |
|
|   Se elem_XML_currente é uma escala e ainda
|   não se registou nenhuma:
|
|   |
|   |   regista_escala
|   |
```

```
|
|   Senão:
|
|   |
|   |   imprime erro tag não reconhecida
|   |
|
|
```

4.2 Grupos

Durante o parsing do XML num grupo ocorre o possível registo de uma translação segundo uma curva, translação normal, escala, rotação ou rotação em função do tempo, sendo a ordem relevante. Para tal, é preciso guardar um contador que mantém quantas das possíveis três já apareceram. Ao registar uma delas, escreve-se para o *array* de ordem a operação corresponde no índice do contador, e de seguida incrementa-se o contador. Este *array* de ordem é inicializado com a operação nula, ID, para todas as suas posições.

```
Algoritmo de regista_translação:

|
|   Se translação tem atributo tempo:
|
|   |
|   |   regista tempo e pontos de curva
|   |   para dentro do componente de animação
|   |
|   |   vetor de ordem, posição do índice atual = ANT
|   |
|
|   Senão:
|
|   |
|   |   regista coordenadas para o Vetor3D de translação
|   |
|   |   vetor de ordem, posição do índice atual = TR
|   |
|
|   incrementa índice atual
|

Algoritmo de regista_rotação:

|
|   regista coordenadas para o Vetor3D de rotação
|
|   Se rotação tem atributo tempo:
|
|   |
|   |   regista o tempo da rotação
|   |
|   |   vetor de ordem, posição do índice atual = ANR
|   |
|
```

```
|   Senão:
|
|   |
|   |   regista o ângulo da rotação
|   |
|   |   vetor de ordem, posição do índice atual = RT
|   |
|
|   incrementa índice atual
|
```

```
Algoritmo de regista_escala:

|
|   regista coordenadas para o Vetor3D de escala
|
|   vetor de ordem, posição do índice atual = SC
|
|   incrementa índice atual
|
```

4.3 Render

Para permitir animações fluídas, a cena é redesenhada sempre que houver disponibilidade de recursos. Tal é conseguido registando a função de desenho da cena para ser chamada, sempre que possível, com *glutIdleFunc()*.

Para garantir que as animações são cumpridas de acordo com o tempo especificado no *XML* lido, no ínicio de cada chamada de desenho da cena, é registado o tempo decorrido desde o começo do ciclo principal do glut através da função *gluGet(GLUT_ELAPSED_TIME)* para uma variável global timestamp.

4.3.1 Cena

Para finalmente desenhar a cena, simplesmente começa-se pelo vetor de componentes na *SceneTree* e aplica-se a cada um o método de render a que respondem de diferentes modos.

4.3.2 Grupos

Para desenhar os grupos, aproveita-se o facto de os grupos serem essencialmente iguais à cena, e basta desenhar todos os componentes no vetor de componentes que contém, de igual modo.

No entanto, um grupo pode conter entre uma e três operações geométricas a aplicar a todos os seus filhos. Face a isto é preciso então testar o vetor de ordem, para saber que transformações estão presentes e aplicá-las antes de começar o desenho dos componentes filhos.

Como um grupo aplica operações geométricas, é necessário antes de proceder a desenhar os filhos e fazer a aplicação das operações que os vão afetar, preservar a matriz atual do modo *MODELVIEW* para que os grupos seguintes sejam independentes como esperado. Para este fim, depois de se desenhar os filhos, vai-se buscar de novo esta matriz do topo da stack do openGL (usamos *glPushMatrix() e glPopMatrix()*).

```
Algoritmo para desenhar um grupo:
```

```
|
|   empurrar a matriz atual para stack do openGL
|
|   Desde a primeira à terceira operação
|   mas apenas se não for uma operação nula:
|
|   |
|   |   Se esta operação é uma translação (TR):
|   |
|   |   |
|   |   |   aplica translação com coordenadas lidas
|   |   |
|   |
|   |
|   |   Se esta operação é uma translação segundo uma curva (ANT):
|   |
|   |   |
|   |   |   calcula_translação_em_função_do_tempo
|   |   |
|   |
|   |   Se esta operação é uma rotação (RT):
|   |
|   |   |
|   |   |   aplica rotação com ângulo e coordenadas lidas
|   |   |
|   |
|   |   Se esta operação é uma rotação em função do tempo (ANR):
|   |
|   |   |
|   |   |   calcula_rotação_em_função_do_tempo
|   |   |
|   |
|   |   Se esta operação é uma escala (SC):
|   |
|   |   |
|   |   |   aplica escala com coordenadas lidas
|   |   |
|   |
|
|   Para cada filho do grupo:
|
|   |
|   |   pede ao filho que se desenhe
|   |
|
|   tira a matriz que pôs anteriormente da stack do openGL
|


    Algoritmo para calcula_translação_em_função_do_tempo:

|
|   número_segmentos = número_pontos_na_curva - 2
|
```

```
|   tempo_no_ciclo_atual = tempo_até_ao_momento - (número_ciclos * tempo_ciclo)
|
|   indice_segmento = trunca(tempo_no_ciclo_atual / tempo_de_segmento)
|
|   tempo_no_segmento = tempo_no_ciclo_atual - (indice_segmento * tempo_de_segmento)
|
|   tempo_0_a_1 = tempo_no_segmento / tempo_de_segmento
|
|   Resultado = calcula_ponto_na_curva
|
|   glTranslatef(Resultado.x,Resultado.y,Resultado.z)
```

Algoritmo para calcula_ponto_na_curva
(\textbf{NOTA}: indice_segmento%número_pontos_na_curva impede apenas
erro de saída fora do vetor devido às aproximações de floats):

```
|
|   pontos = Pontos na curva de catmull-rom lidos no translate
|
|   M = matriz de Catmull-Rom
|
|   P = [pontos[indice_segmento%número_pontos_na_curva],
|        pontos[(indice_segmento+1)%número_pontos_na_curva],
|        pontos[(indice_segmento+2)%número_pontos_na_curva],
|        pontos[(indice_segmento+3)%número_pontos_na_curva],
|       ]
|
|   T = [tempo_0_a_1^3, tempo_0_a_1^2, tempo_0_a_1, 1]
|
|   A = M * P
|
|   Resultado = T * A
|
```

Algoritmo para calcula_rotação_em_função_do_tempo:

```
|
|   tempo_no_ciclo_atual = tempo_até_ao_momento - (número_ciclos * tempo_ciclo)
|
|   angulo_da_rotação = tempo_no_ciclo_atual / tempo_ciclo * 360.0f
|
|   glRotatef(angulo_da_rotacao, rotacao.x, rotacao.y, rotacao.z)
|
```

4.4 Gerador de superfícies

Algoritmo de cálculo de vértices para patches:

```
|
|   Para cada patch:
|
```

```
|   |
|   |   constroi_matriz_de_vértices_de_controlo
|   |
|   |   gera_vertices_para_o_patch
|   |
|
```

Algoritmo de construção da matriz de vértices de controlo:

```
|
|   Para cada linha da matriz:
|
|   |
|   |   Para cada coluna da matriz:
|   |
|   |   |
|   |   |   indice_vértice_atual = patches[começo_do_patch_atual + linha * 4 + coluna]
|   |   |
|   |   |   Para cada coordenada do vértice:
|   |   |
|   |   |   |
|   |   |   |   matriz_de_vértices_de_controlo[coordenada][linha][coluna] =
|   |   |   |       pontos_de_controlo[indice_vértice_atual * 3 + coordenada]
|   |   |   |
|   |   |
|   |
|
```

Algoritmo de construção do vetor v que
contém todos os vetores U/V para cálculo
dos vértices em função do nível de tesselação:

```
|   De 0 até (nível de tesselação) -> considerando nível atual x:
|
|   |
|   |   t_de_0_a_1 = x / (nível de tesselação - 1)
|   |
|   |   v[4*x] = t_de_0_a_1^3
|   |
|   |   v[4*x+1] = t_de_0_a_1^2
|   |
|   |   v[4*x+2] = t_de_0_a_1
|   |
|   |   v[4*x+3] = 1
|   |
```

Algoritmo de geração de vértices para um patch:

```
|
|   De 0 até (nível de tesselação -1) -> considerando nível atual x:
|
```

```
|   |
|   |   De 0 até (nível de tesselação -1) -> considerando nível atual y:
|   |
|   |   |
|   |   |   U = v+(x*4) -> buscar vetor U a v
|   |   |                  em função do ponto atual de tesselação x
|   |   |
|   |   |   V = v+(y*4) -> buscar vetor V a v
|   |   |                  em função do ponto atual de tesselação y
|   |   |
|   |   |   calcula_quadrado com U e V
|   |   |
|   |   |   desenha triângulo superior com canto esquerdo
|   |   |
|   |   |   desenha triângulo inferior com canto direito
|   |   |
|   |
|
```

Algoritmo de cálculo de um quadrado:

```
|
|   aplica_fórmula_de_superfície_de_bezier a U e V
|
|   aplica_fórmula_de_superfície_de_bezier a U+1 e V
|
|   aplica_fórmula_de_superfície_de_bezier a U e V+1
|
|   aplica_fórmula_de_superfície_de_bezier a U+1 e V+1
|
```

Algoritmo de aplicação da fórmula de superfície de Bezier:

```
|
|   M = matriz de Bezier
|
|   P = matriz_de_vértices_de_controlo
|
|   vertice_resultado = U * M * P * M * V
|
```

Capítulo 5

Modelo do sistema solar dinâmico

Os diâmetros dos planetas mantiveram-se desde a fase anterior, tendo sido usado o seguinte site para a recolha de informações: (`http://www.enchantedlearning.com/subjects/astronomy/planets/`).

Para criar as órbitas com proporções aproximadas, recorremos aos periélios (ponto da órbita mais próximo do Sol) e afélios (ponto da órbita mais afastado do Sol) de cada um dos planetas, como apresentado nos fact sheets da *NASA* (`https://nssdc.gsfc.nasa.gov/planetary/factsheet/`).

Decidimos que 20 pontos para a curva de Catmull Rom corresponderia a uma resolução suficiente para o efeito pretendido. Para o cálculo das posições dos pontos, criamos círculos no Blender e afastamos um dos pontos de forma proporcional até obtermos o afélio da órbita.

Orbital parameters

	Mercury
Semimajor axis (10^6 km)	57.91
Sidereal orbit period (days)	87.969
Tropical orbit period (days)	87.968
Perihelion (10^6 km)	46.00
Aphelion (10^6 km)	69.82

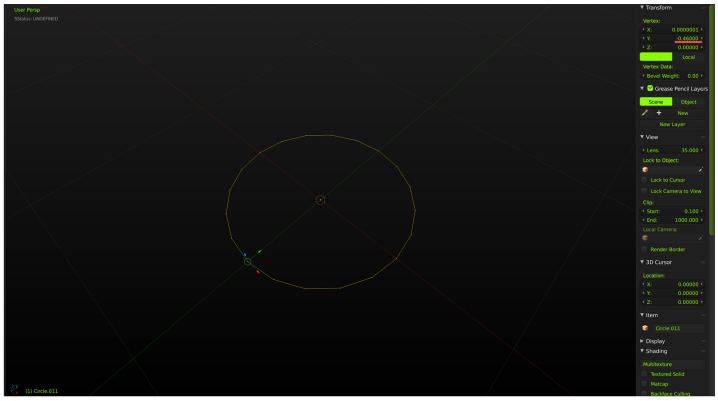


Figura 5.1: Criação da órbita como um círculo com 20 vértices no Blender

Orbital parameters

	Mercury
Semimajor axis (10^6 km)	57.91
Sidereal orbit period (days)	87.969
Tropical orbit period (days)	87.968
Perihelion (10^6 km)	46.00
Aphelion (10^6 km)	69.82

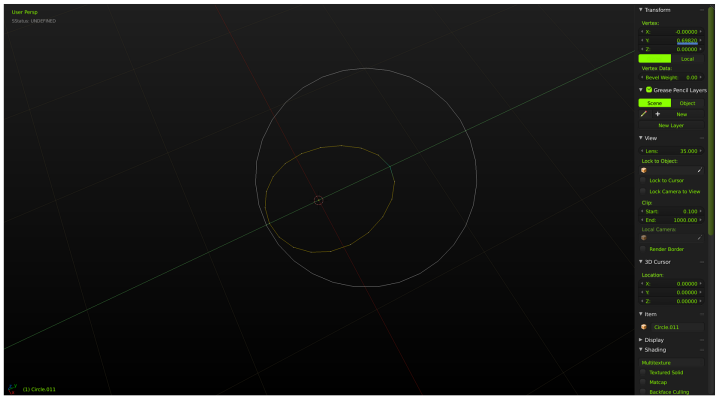


Figura 5.2: Posicionamento do vértice representativo do afélio

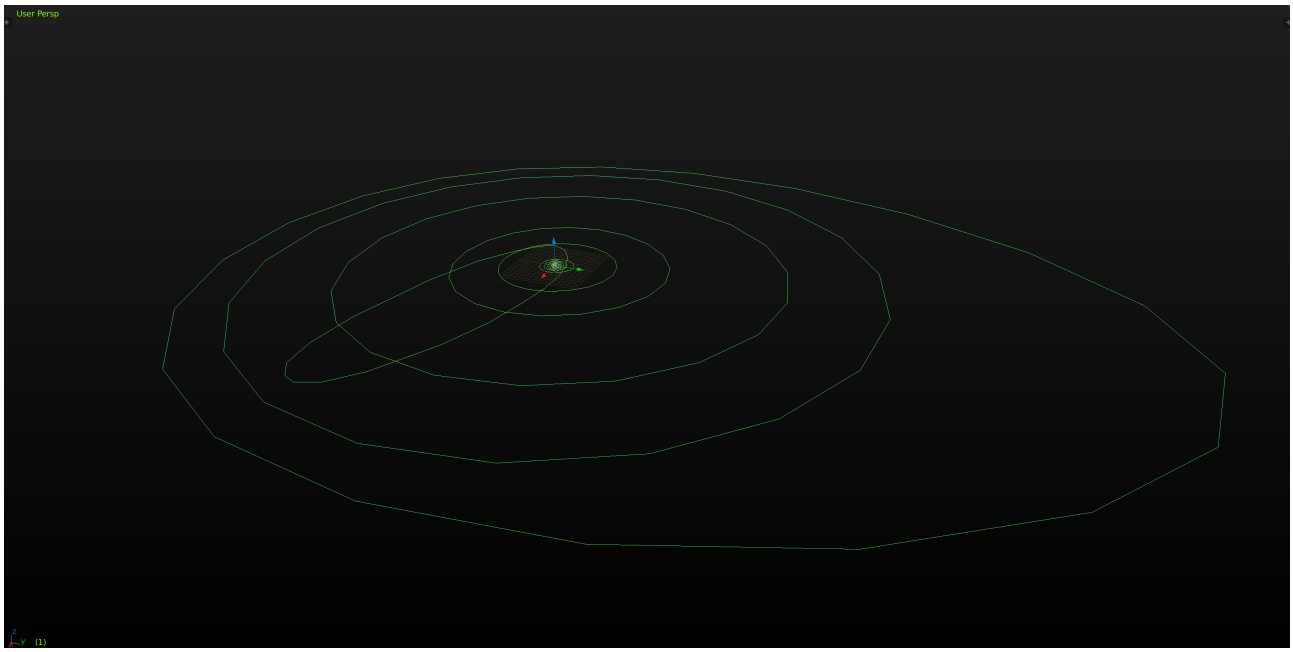


Figura 5.3: Todas as órbitas do Sistema Solar

Depois de geradas todas as órbitas, inserimos as posições dos vértices (com ligeiras modificações) no XML a ser lido. As relações entre velocidades de planetas à volta das órbitas estão também de acordo com as informações presentes no site da *NASA*, mas criamos ainda outro XML com outros valores para ser possível visualizar melhor os movimentos, visto que no original a diferença de velocidades entre planetas é bastante grande.

```
<group>
  <translate time='8.7969'>
    <point X='0' Y='0' Z='12.5657' />
    <point X='2.5586' Y='0' Z='11.9197' />
    <point X='4.8668' Y='0' Z='10.1422' />
    <point X='6.6986' Y='0' Z='7.5342' />
    <point X='7.8747' Y='0' Z='4.4401' />
    <point X='8.2800' Y='0' Z='1.2044' />
    <point X='7.8748' Y='0' Z='-1.8613' />
    <point X='6.6986' Y='0' Z='-4.4992' />
    <point X='4.8668' Y='0' Z='-6.5144' />
    <point X='2.5586' Y='0' Z='-7.7741' />
    <point X='0' Y='0' Z='-8.2021' />
    <point X='-2.5586' Y='0' Z='-7.7741' />
    <point X='-4.8668' Y='0' Z='-6.5144' />
    <point X='-6.6986' Y='0' Z='-4.4992' />
    <point X='-7.8748' Y='0' Z='-1.8613' />
    <point X='-8.2800' Y='0' Z='1.2044' />
    <point X='-7.8747' Y='0' Z='4.4401' />
    <point X='-6.6986' Y='0' Z='7.5342' />
    <point X='-4.8668' Y='0' Z='10.1422' />
    <point X='-2.5586' Y='0' Z='11.9197' />
    <point X='0' Y='0' Z='12.5657' />
    <point X='2.5586' Y='0' Z='11.9197' />
  </translate>
  <scale X='0.02439' Y='0.02439' Z='0.02439' />
  <models>
    <model file="sphere.3d" />
  </models>
</group>
```

Figura 5.4: Secção do XML com informação dos pontos da curva da órbita de Mercúrio

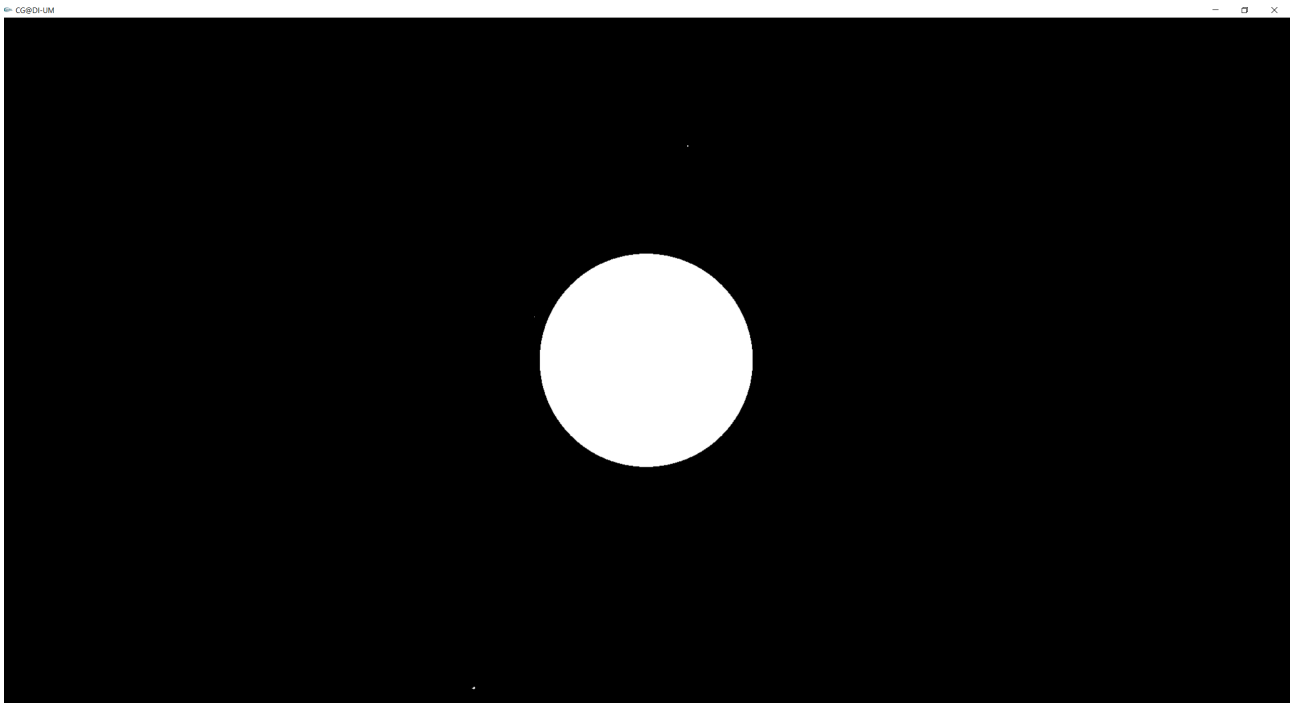


Figura 5.5: Demonstração do Sistema solar - Parte I.

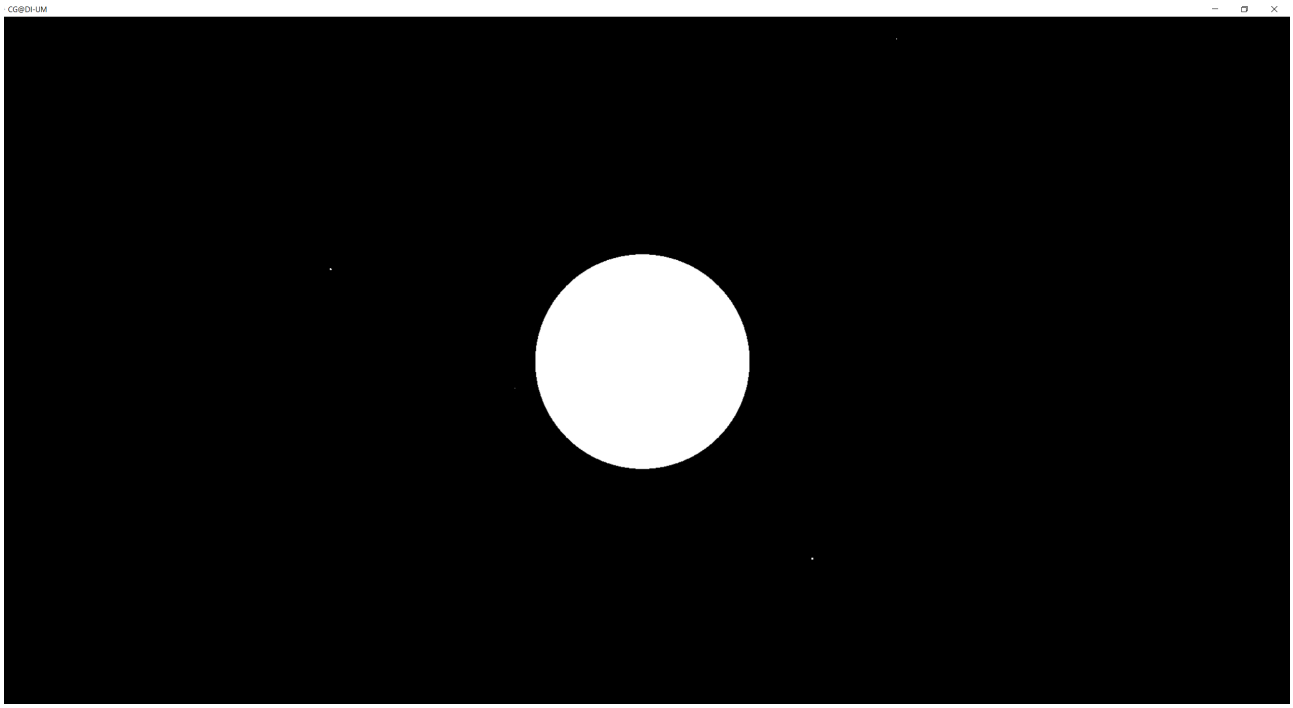


Figura 5.6: Demonstração do Sistema solar - Parte II.

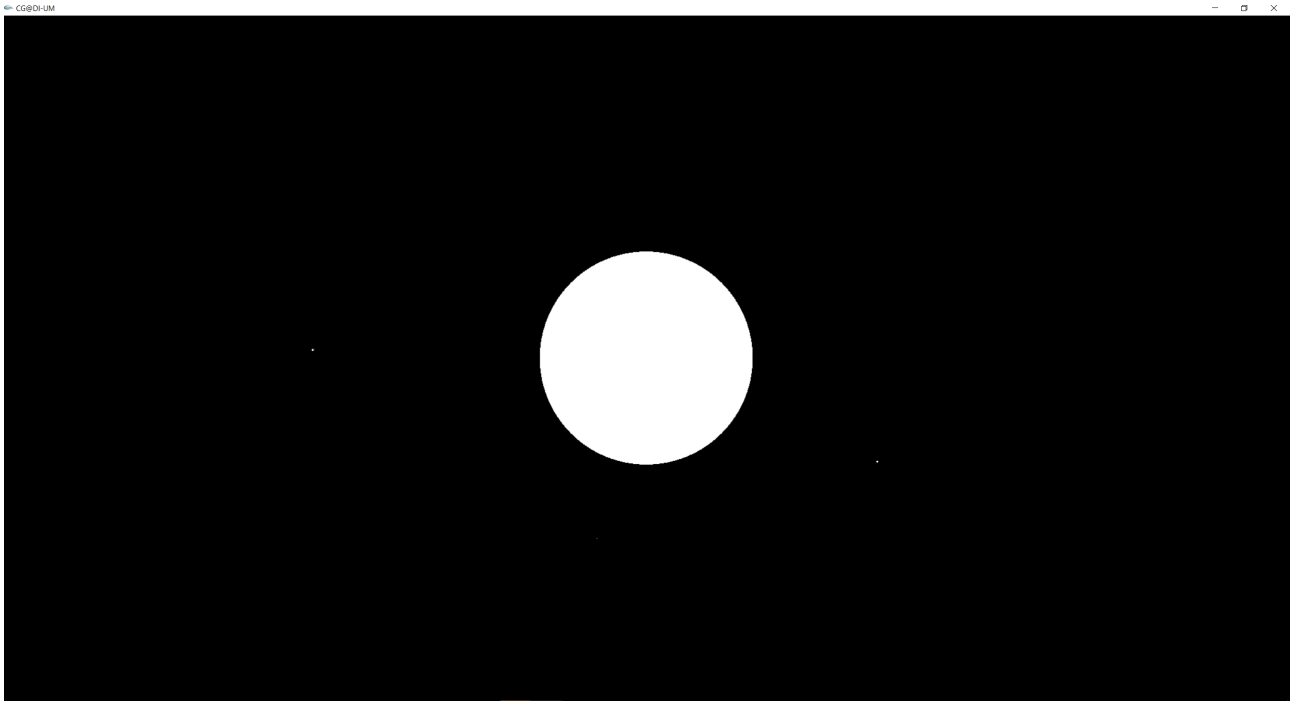


Figura 5.7: Demonstração do Sistema solar - Parte III.

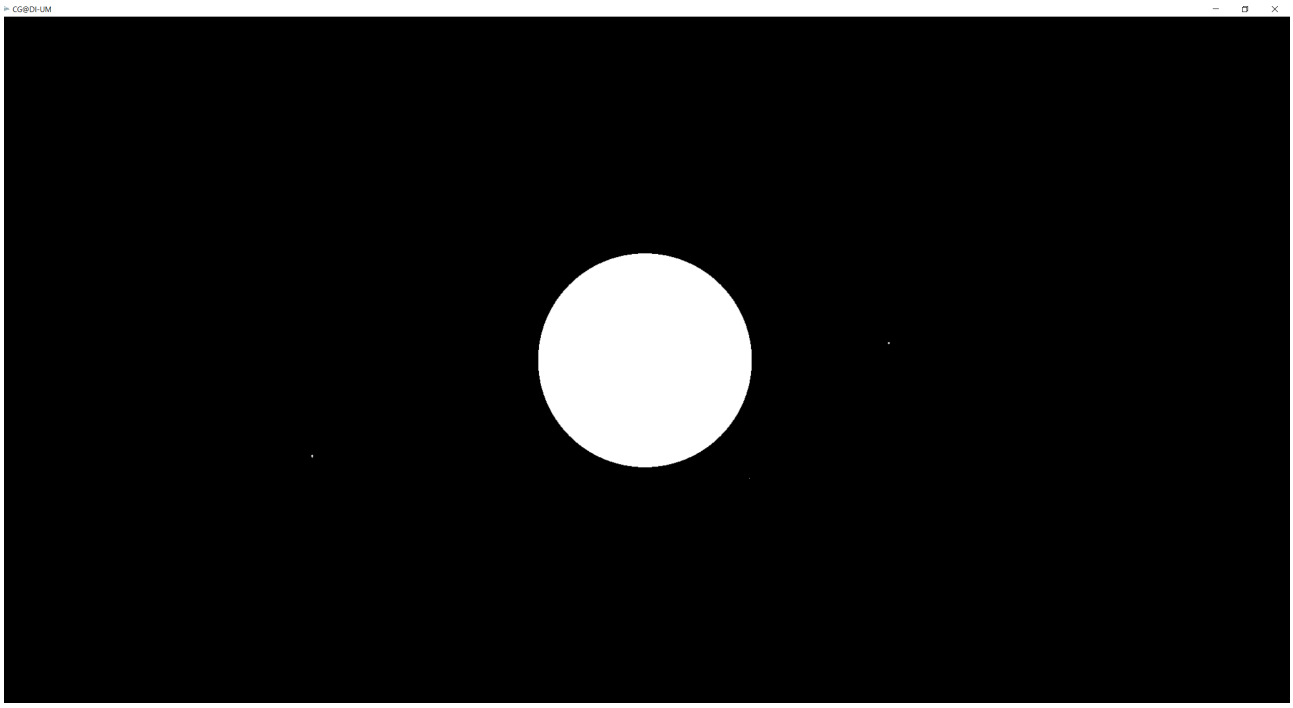


Figura 5.8: Demonstração do Sistema solar - Parte IV.