

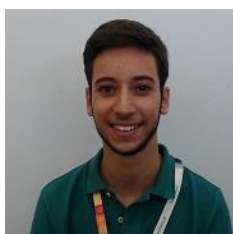


Universidade do Minho
Escola de Engenharia

COMPUTAÇÃO GRÁFICA

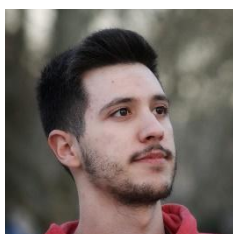
PHASE 3 – CURVES, CUBIC SURFACES AND VBOs

GRUPO 30 - MAIO 2021



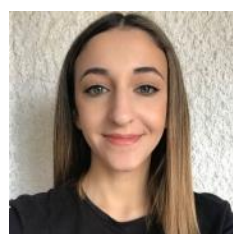
Luís Miguel Pinto A89506

Ana Luísa Carneiro A89533



Pedro Almeida Fernandes A89574

Ana Rita Peixoto A89612



ÍNDICE

INTRODUÇÃO	3
GENERATOR	4
BEZIER PATCHES	4
LEITURA DOS PATCHES	4
FORMAÇÃO DOS PONTOS	5
PSEUDOCÓDIGO	7
EXTRAS	8
CINTURAS ASTERÓIDES E KUIPER	8
ENGINE	9
LEITURA DO FICHEIRO XML	9
DESENHO DO FICHEIRO XML	10
DrawPrimitives	10
RenderScene	10
IMPLEMENTAÇÃO VBOs	10
PRIMITIVAS	10
CINTURAS DE ASTERÓIDES E KUIPER	11
TRANSFORMAÇÕES COM TEMPO	11
ROTAÇÃO	12
PSEUDO-CÓDIGO ROTATE TIME	12
TRANSLAÇÃO	13
PSEUDO-CÓDIGO RENDER CATMULL	13
EXTRAS	14
MOVIMENTO DA CÂMARA	14
CINTURAS DE ASTERÓIDES E DE KUIPER	14
SISTEMA SOLAR	15
DEMO SCENES	15
PALETE CORES	17
CONCLUSÃO	19
ANEXOS	20
solarSystem.xml	20

ÍNDICE DE ILUSTRAÇÕES

Figura 1: estrutura que organiza o conteúdo dos patches	4
Figura 2: Matriz <i>grid</i>	6
Figura 3: triangulação da grid.....	6
Figura 4: Exemplo ficheiro 3d com pontos.....	8
Figura 5: Teapot com tesselação.....	15
Figura 6: Teapot e Derivadas de Catmull	15
Figura 7: teapot com outra orientação e derivadas.....	16
Figura 8: cinturas de asteróides e kuiper	16
Figura 9: Sistema Solar	17

ÍNDICE DE TABELAS

Tabela 1 : Palete de cores - Adições fase 3	17
Tabela 2: Palete de cores – Fase 2	18

INTRODUÇÃO

A 3ª fase deste projeto teve como objetivo obter uma representação dinâmica do sistema solar, tornando-o mais realista. De modo a concretizar este objetivo, foi necessário implementar curvas, superfícies cúbicas e VBOs.

No *Generator*, foi proposta a conversão de um dado ficheiro em ficheiro 3d, a partir de um conjunto de *patches* de *bezier* e o nível de tecelagem. Além disso, no âmbito das funcionalidades extra também foi implementado um mecanismo gerador de pontos aleatórios de modo a permitir a representação das cinturas.

Quanto ao *Engine*, a implementação passou pelo acréscimo de transformações com tempo. Sendo que no caso da translação podemos definir uma animação através de curvas de *Catmull-Rom*, ao passo que na rotação é possível efetuar uma rotação de 360 graus em torno de um dado eixo. Nas duas transformações podemos definir um tempo para as efetuar. Em adição, foi também alterado o modo de desenho de primitivas para VBOs. Como funcionalidade extra, decidimos implementar o movimento da câmara com o rato.

No presente relatório apresentamos explicações detalhadas de todas as etapas que sustentaram esta fase do projeto, acompanhadas por pseudocódigo e imagens de forma a ilustrar o raciocínio usado e manter uma documentação exemplificativa do projeto.

GENERATOR

No programa *Generator* foi necessário efetuar algumas alterações e acréscimos de modo a efetuar a leitura dos *patches* de *Bezier* e a sua transformação em pontos que formam triângulos. Além disso, foram também efetuadas adições relativas às funcionalidades extra.

BEZIER PATCHES

Nesta fase foi proposta a implementação de um teapot utilizando *patches* de *Bezier* que será, posteriormente, utilizado para a criação de um cometa no sistema solar. Para isso implementou-se uma nova primitiva, cometa, no *Generator* materializado na função *creatComet*. Esta função recebe como parâmetro um ficheiro onde se encontram os *patches* de *Bezier* (teapot.patch), um nível de tesselação que determina o detalhe do *teapot* e um ficheiro *3d* onde armazenamos os pontos finais do *teapot*.

LEITURA DOS PATCHES

Para a leitura do ficheiro teapot.patch e de forma a seguir com a estrutura do ficheiro começou-se por percorrer todas as linhas que representam cada *patch*, que são representados pelos vários índices dos pontos de controlo. Assim, decidiu-se armazenar cada linha lida do ficheiro num vetor de *strings*, *patchesIndices*, onde cada *string* representa um *patch* do *teapot*. De seguida armazenaram-se os vários pontos de controlo num vetor de Pontos, *points*, onde cada ponto é representado pela classe *Point* que é constituída por 3 floats que representam as coordenadas x, y e o z.

Para finalizar a leitura do ficheiro associa-se a cada *patch* um conjunto de pontos de controlo. Para isso associa-se a cada conjunto de índices armazenados no vetor *patchesIndices* os respetivos pontos armazenados no vetor *points*. Cada *patch* é constituído por 16 pontos que formam 4 curvas de *bezier*, e por sua vez cada curva é constituída por 4 pontos. Desta forma, decidiu-se armazenar esses pontos num vetor *patches* onde cada elemento representa um *patch*, e cada *patch* é constituído por um vetor que armazena a representação das várias curvas, tal como está na figura 1.

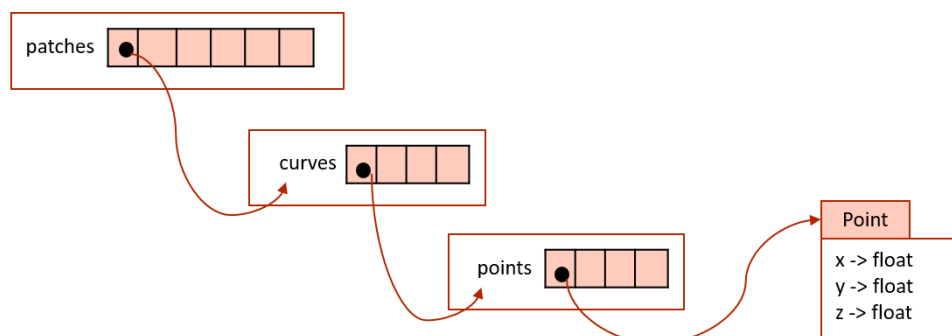


FIGURA 1: ESTRUTURA QUE ORGANIZA O CONTEÚDO DOS PATCHES

FORMAÇÃO DOS PONTOS

Para formar os pontos constituintes do *teapot* começou-se por percorrer cada *patch* formando em cada iteração a matriz, *matrix*, que é calculada através da expressão abaixo.

$$matrix = M \cdot \begin{pmatrix} P_{00} & P_{10} & P_{20} & P_{30} \\ P_{01} & P_{11} & P_{21} & P_{31} \\ P_{02} & P_{12} & P_{22} & P_{32} \\ P_{03} & P_{13} & P_{23} & P_{33} \end{pmatrix} \cdot M^T$$

A matriz central será alterada em cada iteração à medida que se percorre o vetor *patches* e é constituída pelas várias curvas de *bezier* que representam um *patch*. Assim, cada linha representa uma curva e cada elemento representa um objeto da classe *Point*.

A representação da matriz M é dada pela expressão abaixo mantendo-se constante ao longo de toda a formação de pontos. Como a matriz M é simétrica então a sua transposta é igual a ela mesma.

$$M = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Para a multiplicação destas matrizes implementou-se as funções *matrixMultiplication* e *matrix2Multiplication* que multiplicam a matriz M pela matriz do *patch* e que multiplica a matriz do *patch* pela transposta da matriz M, respetivamente.

De seguida, calculou-se cada ponto de *bezier*, $p(u, v)$, através da expressão seguinte:

$$p(u, v) = (u^3 \quad u^2 \quad u \quad 1) \cdot matrix \cdot \begin{pmatrix} v^3 \\ v^2 \\ v \\ 1 \end{pmatrix}, \quad u, v \in [0, 1]$$

Para cada ponto formado, os parâmetros u e v vão aumentando de acordo com o nível de tesselação recebido, seguindo a expressão:

$$x_{i+1} = x_i + \frac{1}{tesselagem}, \quad i \in [0, tesselação] \wedge x \in \{u, v\}$$

Através das funções *multiplyUbyMatrix* (que multiplica o vetor u pela matriz *matrix*) e *multiplyVectors* (que multiplica o resultado da função *multiplyUbyMatrix* pelo vetor v) obtemos um ponto de *bezier* que será armazenado na matriz *grid*. Esta matriz é constituída por todos os pontos de *bezier* formados a partir de um *patch* como podemos ver de seguida:

$P(0, 0)$	$P(0.25, 0)$	$P(0.5, 0)$	$P(0.75, 0)$	$P(1, 0)$
$P(0, 0.25)$	$P(0.25, 0.25)$	$P(0.5, 0.25)$	$P(0.75, 0.25)$	$P(1, 0.25)$
$P(0, 0.5)$	$P(0.25, 0.5)$	$P(0.5, 0.5)$	$P(0.75, 0.5)$	$P(1, 0.5)$
$P(0, 0.75)$	$P(0.25, 0.75)$	$P(0.5, 0.75)$	$P(0.75, 0.75)$	$P(1, 0.75)$
$P(0, 1)$	$P(0.25, 1)$	$P(0.5, 1)$	$P(0.75, 1)$	$P(1, 1)$

FIGURA 2: MATRIZ GRID

A matriz grid, no final das iterações u e v , contém todos os pontos de *bezier* que constituem um *patch* que formam uma grelha de quadrados. Para que se consiga representar o *teapot* através de triângulos é necessário recorrer à triangulação da grelha, tal como está representado na figura seguinte:

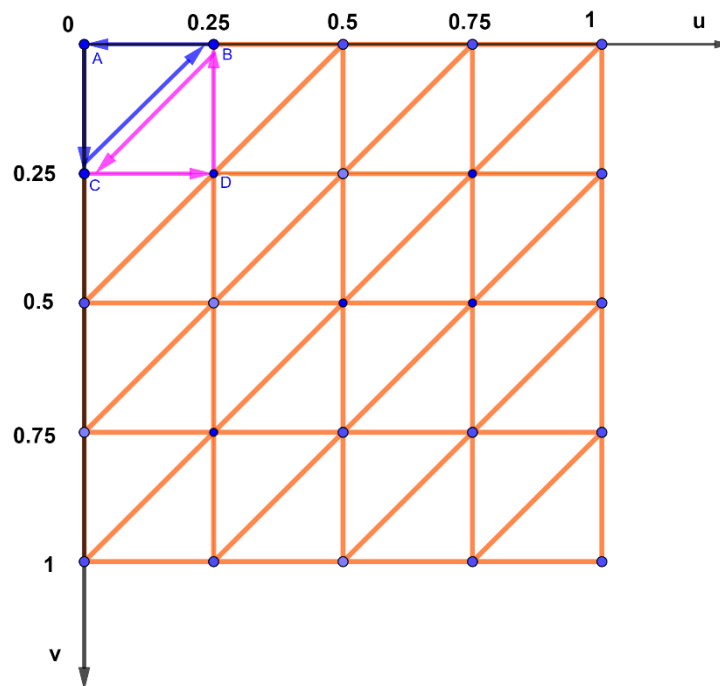


FIGURA 3: TRIANGULAÇÃO DA GRID

Para que cada quadrado seja constituído por dois triângulos é necessário percorrer a grelha de forma que os pontos dos quadrados formem triângulos no sentido anti-horário. Cada quadrado será armazenado no vetor de pontos (*triangles*) sobre a forma de dois triângulos. Contudo é essencial que os pontos desses triângulos sejam inseridos na ordem A-C-B seguido de C-D-B de modo a respeitar a regra da mão direita.

Finalmente, o vetor *triangles* será transformando numa *string* e posteriormente colocado no ficheiro .3d que recebemos como parâmetros

PSEUDOCÓDIGO

De seguida encontra-se o pseudocódigo da formação dos pontos tal como foi explicado anteriormente.

```
vector<Point> buildPointsComet (vector<vector<vector<Point>>> patches, int tessellation) {

    // init m, grid, triangles

    for ( each patch ){
        // calculate matrix matrix

        for ( each v tessellation ( j ) ) {
            // calculate vector v

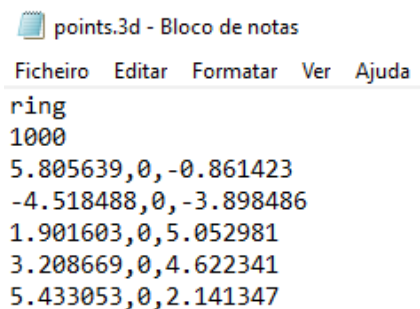
            for ( each u tessellation ( k ) ) {
                // calculate vector u
                // calculate point p
                grid[j][k] = p;
            }
        }
        for ( j < tessellation ) {
            for ( k < tessellation ) {
                triangles.push_back( grid[j][k] );           // A
                triangles.push_back( grid[j+1][k] );       // C
                triangles.push_back (grid[j][k+1] );       // B
                triangles.push_back( grid[j+1][k] );       // C
                triangles.push_back( grid[j+1][k+1] );     // D
                triangles.push_back( grid[j][k+1] );       // B
            }
        }
    }
}
```


EXTRAS

Para esta fase, consideramos pertinente melhorar as cinturas de asteroides e de *Kuiper* de modo a obter uma representação do Sistema Solar visualmente mais apelativa e realista.

CINTURAS ASTERÓIDES E KUIPER

Para a implementação das cinturas, foi criada uma nova função que gera pontos aleatórios entre duas circunferências. A função geradora dos pontos é a *creatPoints* que recebe como argumento um *vector* que contém informações como o raio da circunferência mais interna, o raio da circunferência mais externa, o número de pontos a gerar e o nome do ficheiro 3D. Os pontos são gerados aleatoriamente a partir das funções *srand* e *rand*. Caso o ponto gerado não se encontre entre as duas circunferências, é gerado um novo ponto até cumprir essa restrição. Por fim, é efetuada a escrita no ficheiro 3D. Este ficheiro possui uma estrutura particular, na medida em que possui um parâmetro extra no início, ou seja, a primeira linha irá conter a *string* “ring” como indicador de que se trata de um ficheiro de pontos. A seguinte figura é representativa desta estrutura:



```

points.3d - Bloco de notas
Ficheiro Editar Formatar Ver Ajuda
ring
1000
5.805639,0,-0.861423
-4.518488,0,-3.898486
1.901603,0,5.052981
3.208669,0,4.622341
5.433053,0,2.141347
  
```

FIGURA 4: EXEMPLO FICHEIRO 3D COM PONTOS

De modo a gerar o ficheiro 3D das cinturas, pode-se efetuar um comando com a seguinte estrutura:

```
points raioInterno raioExterno nrPontos nomeFicheiro
```

A título de exemplo, de forma a gerar 1000 pontos entre uma circunferência de raio 5 e uma circunferência de raio 6, armazenando os pontos no ficheiro *points.3d*, poderíamos executar o comando `points 5 6 1000 points.3d`.

ENGINE

Nesta fase do projeto foi necessário atualizar o programa *Engine* com o principal objetivo de permitir translações e rotações com tempo e de desenhar as primitivas com recurso a VBOs. Para isso, foi necessário efetuar alterações tanto na leitura do ficheiro XML, como no desenho das primitivas. Além disso, também foi necessário efetuar alterações ligeiras de modo a suportar as funcionalidades extra.

LEITURA DO FICHEIRO XML

De modo a suportar a extensão das transformações *rotate* e *translate*, foi necessário alterar a leitura do ficheiro XML. A função *parseGroup*, que trata da análise deste documento e do populamento das estruturas de dados, possui agora capacidade de ler e armazenar rotações e translações com tempo. Dado que em cada grupo existirá, no máximo, uma translação com tempo, foi necessário a adição de um novo *vector* de *Point* à classe *Group*. Este *vector* permitiu armazenar os pontos provenientes do *translate time* do XML, que irão originar a curva de *Catmull*. Adicionalmente, de forma a auxiliar a implementação das rotações com tempo, considerou-se útil acrescentar uma variável *angle* à classe *Group*. Além disso, na classe *Transformation* foi necessário adicionar uma variável *time*, de forma a permitir o registo do tempo relativo ao *rotate* ou *translate*. Deste modo, ao ser efetuada a leitura do ficheiro XML todas as estruturas serão preenchidas corretamente e todos os dados do XML serão tidos em consideração.

Por fim, é de notar, que existiram algumas adições na estrutura do ficheiro XML, de modo a permitir a existência de transformações com tempo. A estrutura destas transformações é a seguinte:

```
<rotate time="60" x="0" y="1" z="0" />
<translate time="5">
    <point x="-1" y="0" z="-4"/>
    <point x="-5" y="0" z="4"/>
    <point x="5" y="5" z="4"/>
    <point x="5" y="0" z="0"/>
</translate>
```

DESENHO DO FICHEIRO XML

DrawPrimitives

Anteriormente, a execução da **drawPrimitives** traduzia-se em, para cada grupo, executar inicialmente as transformações contidas no ficheiro XML, de seguida desenhar as primitivas e por fim atuar recursivamente caso esse grupo tivesse subgrupos associados.

Nesta fase, as mudanças recaem sobre as transformações aplicadas e o desenho das primitivas. Existe agora uma separação entre grupos com transformações com tempo e grupos com transformações sem tempo. No tratamento do grupo haverá, portanto, uma fase inicial onde apuramos a ocorrência das novas transformações. Caso estas existam, as transformações e o desenho das primitivas será executado numa função auxiliar **rotateTime**, no caso das rotações, e **renderCatmull**, no caso das translações. Caso não existam, o procedimento a adotar pela função **drawPrimitives** é igual ao da fase 2. Assim foi criado um procedimento mutuamente exclusivo, onde ou se trata tempo ou então não se trata tempo. Finalmente, será efetuada a chamada recursiva para subgrupos aninhados.

RenderScene

A principal mudança na **renderScene** deve-se à funcionalidade extra implementada – a cintura de asteroides e de *Kuiper*. O passo dado é simples, basta apenas desenhar todos os pontos que constituem os anéis presentes num vetor de pontos pré-carregado durante a leitura do XML. A motivação da mudança é puramente visual, mas consideramos o resultado final bastante mais apelativo.

IMPLEMENTAÇÃO VBOs

Para esta fase, foi necessário implementar o desenho dos modelos utilizando VBOs. Esta alteração permite aumentar a eficiência do programa e renderização das cenas. O utilizou-se VBO's tanto para as primitivas construídas a partir de triângulos, como para os pontos que representam as cinturas.

PRIMITIVAS

De modo a concretizar o requisito anteriormente referido, procedeu-se à criação de um *vector* de *float* onde será armazenada cada coordenada de cada ponto do ficheiro 3D, *vertexB*. Dado que a leitura de todas as primitivas é efetuada para um único VBO, foi necessário acrescentar uma variável de controlo que verifica qual a zona do VBO a desenhar em cada momento, *ptr*. O processo de escrita no VBO é efetuado aquando da leitura de cada ficheiro 3D. A função encarregue da leitura, *readFile*, efetua a operação de *push_back* de cada coordenada para o *vertexB*. O VBO relativo a este *vector* foi criado na função *initGlut*, a partir da função

glGenBuffers que cria o VBO, e das funções *glBindBuffer* e *glBufferData* que efetuam a cópia do *vector* para a memória gráfica.

No que toca ao desenho do conteúdo presente no VBO, foi necessário utilizar as funções *glBindBuffer* de forma a indicar qual o VBO ativo, *glVertexPointer* para informar que um vértice é constituído por 3 *floats* e, por último, a função *glDrawArrays* utiliza o modo *GL_TRIANGLES* para desenhar os triângulos presentes no VBO, tendo em conta o vértice inicial e o número de pontos a desenhar. Possuímos informação acerca da zona do VBO a desenhar em cada momento devido às primitivas armazenadas nas estruturas de dados, e também devido à variável global *ptr* anteriormente referida.

CINTURAS DE ASTERÓIDES E KUIPER

Analogamente ao procedimento efetuado para as primitivas, também foram aplicados VBOs para o desenho das cinturas (funcionalidade extra). Procedeu-se à criação de um *vector* de *float* onde será armazenada cada coordenada de cada ponto do ficheiro 3D, *ringVBO*. Dado que a leitura de todos os pontos é efetuada para um único VBO, foi necessário acrescentar uma variável de controlo que verifica a zona do VBO a desenhar em cada momento, *ptrRing*. O processo de escrita no VBO é efetuado aquando da leitura de cada ficheiro 3D. A função encarregue da leitura, *readFile*, efetua a operação de *push_back* de cada coordenada para o *ringVBO*, caso se trate de um ficheiro de pontos que não formam triângulos. O VBO relativo a este *vector* foi criado na função *initGlut*, a partir da função *glGenBuffers* que cria o VBO, e das funções *glBindBuffer* e *glBufferData* que efetuam a cópia do *vector* para a memória gráfica.

No que toca ao desenho do conteúdo presente no VBO, foi necessário utilizar as funções *glBindBuffer* de forma a indicar qual o VBO ativo, *glVertexPointer* para informar que um vértice é constituído por 3 *floats* e, por último, a função *glDrawArrays* utiliza o modo *GL_POINTS* para desenhar os pontos presentes no VBO, tendo em conta o vértice inicial e o número de pontos a desenhar. Possuímos informação acerca da zona do VBO a desenhar em cada momento devido às primitivas armazenadas nas estruturas de dados, e também devido à variável global *ptrRing* anteriormente referida.

TRANSFORMAÇÕES COM TEMPO

Antes de apresentar em maior detalhe as metodologias por detrás da implementação de transformações com tempo, achamos sensato enunciar os pressupostos adotados :

- Para cada grupo apenas poderá existir um tipo de translação e um tipo de rotação, isto é, ou com a noção de tempo ou sem a noção de tempo.
- A base de toda a noção de tempo usará a função *gluGet(GLUT_ELAPSED_TIME)*, tal como foi recomendado no guião da fase.
- A classe Group tem agora novas variáveis e métodos de acordo com a informação do tempo.

ROTAÇÃO

De modo a implementar a rotação com a noção de tempo foram necessárias 2 alterações fundamentais: na função *drawPrimitives* (mencionado no capítulo Desenho do Ficheiro XML) e na classe *Group* (mencionado no capítulo Leitura do Ficheiro XML); e uma nova função auxiliar chamada *rotateTime*.

A principal responsável da rotação com tempo é a função *rotateTime*, cujo resultado permite atualizar o ângulo de rotação para cada grupo, para que de facto se processe uma rotação na cena. Tendo recebido como argumentos o tempo total da rotação em segundos, o vetor que define o eixo de rotação, as primitivas a desenhar e as respetivas transformações a executar e os estado atual do ângulo do grupo. Poderá então, internamente, executar o seu “modus operandi”, ou seja, efetuar o *rotate*, seguido de todas as transformações desse grupo e do desenho das primitivas. Em seguida apresentamos a fórmula de atualização do ângulo:

$$\text{angle} = ((\text{float})\text{glutGet}(\text{GLUT_ELAPSED_TIME}) * 360 / 1000) / ((\text{float})\text{time});$$

Esta função *rotateTime* permite tanto a rotação dos planetas em torno do sol, como a rotação dos planetas em torno de si próprios. Este objetivo está intrinsecamente relacionado com o formato do ficheiro XML. Se quisermos simular o movimento de translação em torno do sol, bastará efetuar um grupo com um *rotate time*. Para incluir o movimento de rotação em torno de si próprio, será necessário definir um grupo aninhado no grupo anteriormente mencionado, com outro *rotate time*.

Posto isto, conseguimos agora atingir o objetivo proposto inicialmente, ou seja, traçar movimentos de translação para : os planetas (em torno do sol), as cinturas de asteroides (em torno do sol) e as luas (em torno do seu planeta); e ainda definir movimentos de rotação para o sol, as luas e os planetas, tendo sempre em conta o tempo definido para a rotação.

PSEUDO-CÓDIGO ROTATE TIME

```
vector<float> rotateTime(int time, float x, float y, float z, vector<Primitive> primitives,
vector<Transformation> transformations, float angle) {

    glRotatef(angle, x, y, z);

    for (each transformation) {
        // apply transformation
    }
    for (each primitive) {
        // draw primitive with VBO
    }

    angle = ((float)glutGet(GLUT_ELAPSED_TIME) *360/1000) /((float)time); //update angle

    glutPostRedisplay();
    vector<float> res;
    res.push_back(angle);
    return res;
}
```

TRANSLAÇÃO

No caso das translações com tempo, foi necessário recorrer a curvas de *Catmull* de modo a delinear a trajetória da primitiva. Além disso, existe também um tempo definido para percorrer a curva.

De modo a respeitar os requisitos anteriormente referidos, foi necessário considerar um conjunto de pontos, no mínimo 4. Estes pontos são provenientes do ficheiro XML e são armazenados aquando da leitura do ficheiro num *vector* de elementos *Point*, *p*, que se trata de uma variável global do programa.

O movimento da primitiva propriamente dito, é efetuado com recurso à função *renderCatmull*. Esta função é invocada se for detetado um *translate time* e recebe como argumentos o tempo de translação, as primitivas que serão alvo do movimento e as restantes transformações que lhes deverão ser aplicadas (por exemplo, *scale* ou *color*). Inicialmente é efetuada a operação de *glPushMatrix()* de modo a armazenar o estado da matriz. Para efeitos de *debug* e visualização do percurso efetuado, foi criada a função *renderCatmullRomCurve* que desenha a curva de *Catmull* e respetivas derivadas de cada ponto calculado, com recurso aos modos de desenho *GL_LINE_LOOP* e *GL_LINES*. Por um prisma semelhante, implementou-se a função *getGlobalCatmullRomPoint* que, a partir do valor de *t* (calculado com recurso à função *gluGet(GLUT_ELAPSED_TIME)*) é capaz de determinar o próximo ponto e a sua derivada. A partir destes pontos, são efetuadas operações de normalização e cálculo vetorial, recorrendo às funções *normalize* e *cross*, respetivamente. Além disso, também é efetuado o cálculo de uma matriz de rotação, recorrendo à função *buildRotMatrix* que irá efetuar a rotação da primitiva de modo a acompanhar o movimento da curva. Após efetuar as diversas alterações ao referencial, estamos em posição de aplicar as restantes transformações, que foram recebidas como argumento. Seguidamente, podemos desenhar as primitivas em questão que terão em conta todas as alterações anteriormente aplicadas ao sistema de eixos. Por fim, é efetuado um *glPopMatrix()* que permitirá voltar à matriz anterior.

PSEUDO-CÓDIGO RENDER CATMULL

```
t = ((float)glutGet(GLUT_ELAPSED_TIME) / 1000) / ((float)time);

getGlobalCatmullRomPoint(t, pos, deriv); // calculates point and derivative

glTranslatef(pos[0], pos[1], pos[2]);

for (int i = 0; i < 3; i++) x[i] = deriv[i]; // the x axis is equal to the derivative

normalize(x) // normalize x vector
cross(x, yBefore, z); // right hand rule to get z vector
normalize(z); // normalize z vector
cross(z, x, yNew); // right hand rule to get yNew
normalize(yNew); // normalize yNew vector

for (int i = 0; i < 3; i++) yBefore[i] = yNew[i]; // update yBefore
```

```

    buildRotMatrix(x, yNew, z, m); // build rotation matrix m
    glMultMatrixf(m);             // apply rotation matrix m

    for (each transformation) {
        // apply transformation
    }

    for (each primitive) {
        // draw primitive with VBO
    }

```

EXTRAS

MOVIMENTO DA CÂMARA

De forma a registar os movimentos efetuados com o rato na cena e a permitir uma melhor experiência de navegação pela mesma, decidimos implementar as funções *processMouseMotion* e *processMouseButtons*.

CINTURAS DE ASTERÓIDES E DE KUIPER

Nesta fase, foi otimizado o extra de desenhar as cinturas de asteróides e de *Kuiper*. Na fase anterior tínhamos uma implementação a partir de um *torus* que não dava uma perspetiva tão realista da cena. Consequentemente, consideramos conveniente implementar de maneira diferente e mais realista. Para isso, foi necessário adaptar o *Engine* de modo a capacitá-lo da leitura de ficheiros 3D cujos pontos não formam triângulos. Deste modo, foi necessário adicionar uma variável à classe *Primitive* que denota se se trata de uma primitiva formada a partir de triângulos ou não. O populamento das primitivas constituídas por pontos que não formam triângulos é equivalente às primitivas normais. Tal como foi mencionado anteriormente, a partir do indicador “ring” no ficheiro 3d é possível efetuar a leitura correta dos pontos e posterior desenho.

SISTEMA SOLAR

DEMO SCENES

A partir da leitura do ficheiro dos *patches* de *bezier* no *Generator*, foi possível gerar pontos que serão posteriormente desenhados e visualizados no programa *Engine*, originando um *teapot*. Na seguinte figura está presente o desenho do *teapot* onde é possível observar com clareza a existência de tesselação.

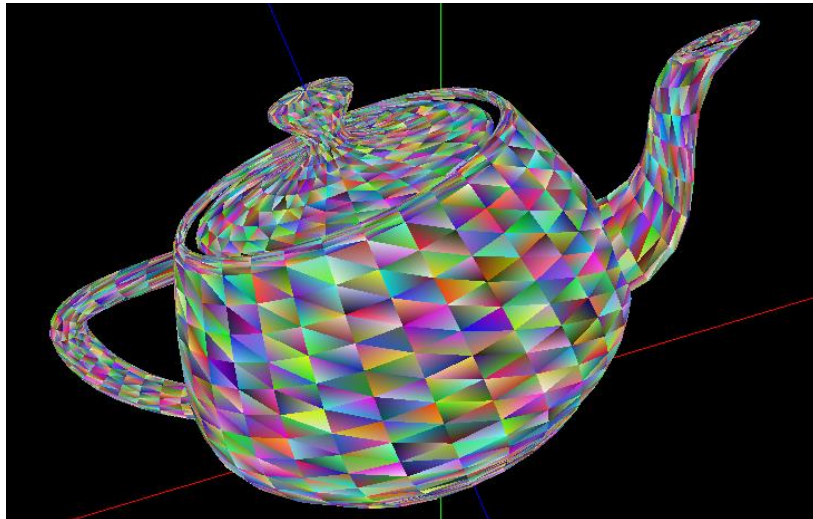


FIGURA 5: TEAPOT COM TESSELAÇÃO

De modo a oferecer uma visualização do trabalho realizado, foram registadas algumas cenas representativas de certas partes do projeto. De seguida está presente uma imagem do *teapot* formado a partir dos *patches* de *bezier* a descrever uma trajetória formada através das curvas de *Catmull*. Além disso, também é possível observar as derivadas de cada ponto que forma a curva.

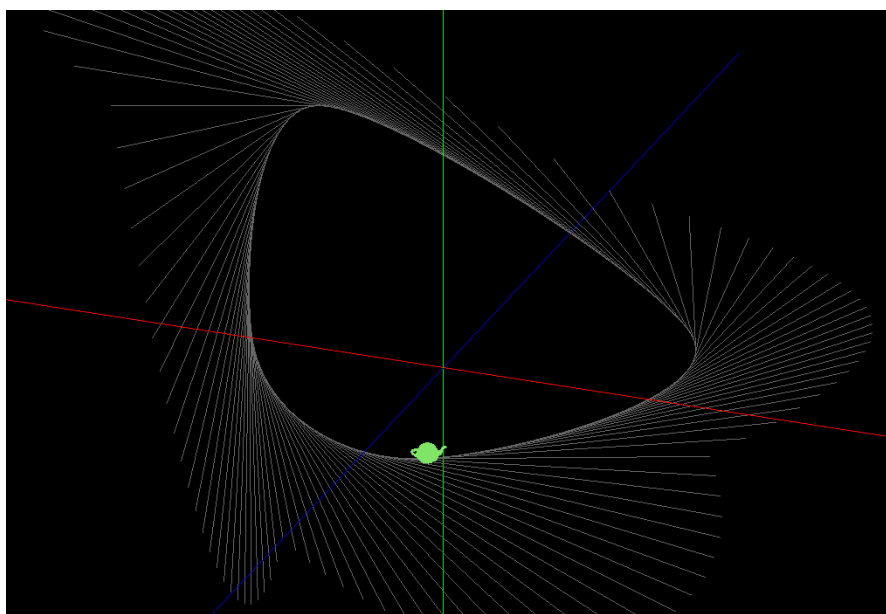


FIGURA 6: TEAPOT E DERIVADAS DE CATMULL

Na figura abaixo é possível observar o *teapot* numa outra perspectiva que denota o seu movimento ao longo da curva e a sua orientação que acompanha o movimento.

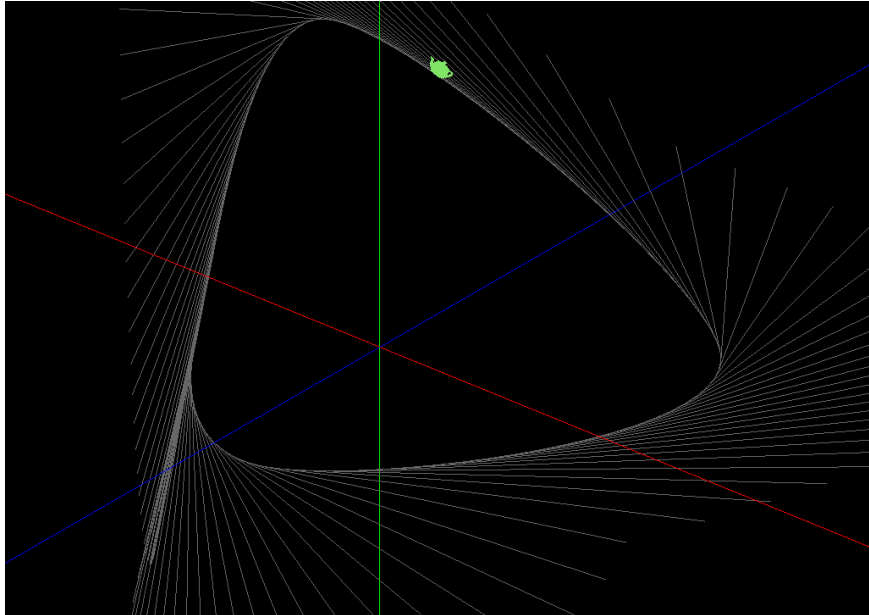


FIGURA 7: TEAPOT COM OUTRA ORIENTAÇÃO E DERIVADAS

Além disso, dado que também foram implementadas cinturas de asteróides e de *Kuiper* no Sistema Solar, consideramos conveniente observar a sua representação individualmente, tal como é possível observar na seguinte figura:

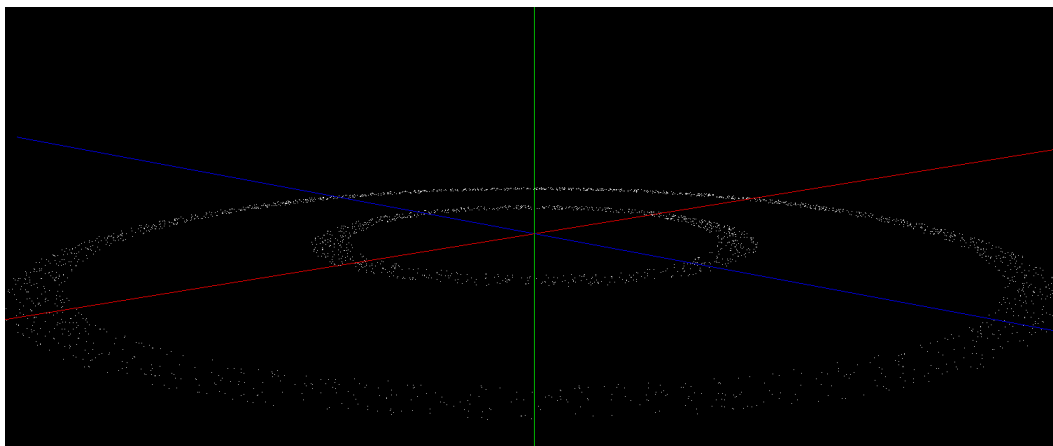


FIGURA 8: CINTURAS DE ASTERÓIDES E KUIPER

Por último, mas não menos importante, apresentamos uma imagem representativa do Sistema Solar. O Sistema Solar implementado inclui o sol, os planetas de Mercúrio a Neptuno e algumas das suas respectivas luas, a cintura de asteróides e por fim a cintura de *Kuiper*. Apesar de não ser possível visualizar na imagem, todas as primitivas presentes possuem movimento. É de notar que os planetas Vénus e Urano rodam em sentido horário, ao contrário dos outros planetas.

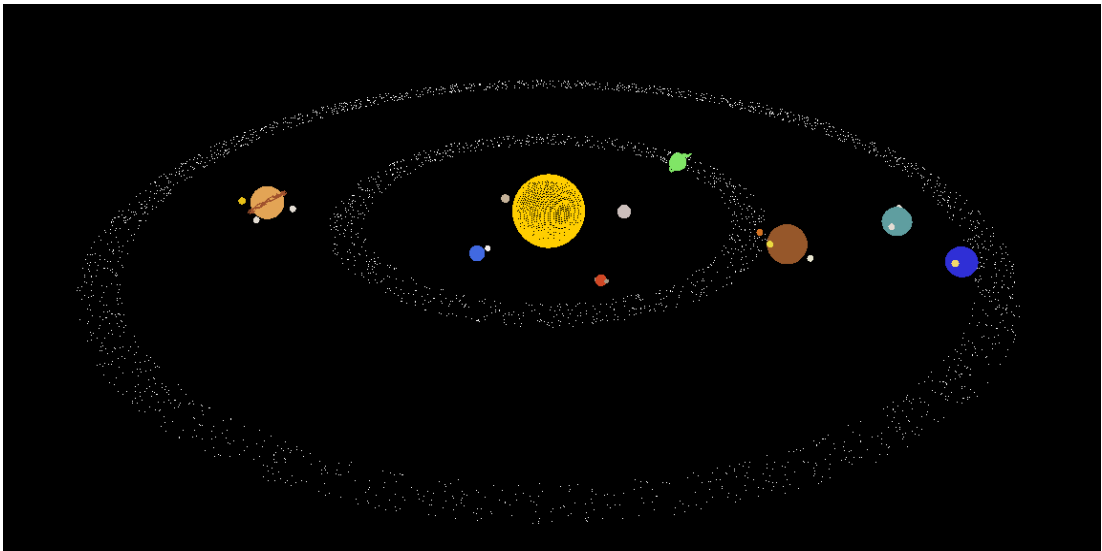


FIGURA 9: SISTEMA SOLAR

PALETE CORES

Na seguinte tabela está presente a paleta de cores com os respectivos códigos utilizados para cada modelo. Podemos observar as adições efetuadas na 3ª fase no topo da tabela.

FASE 3	COR	FASE 3	COR
Cintura Asteroide	<div>#FFFFFF (255,255,255)</div>	Anel de saturno	<div>#A0522D (160,82,45)</div>
Cometa	<div>#80E666 (128,230,102)</div>	Cintura Kuiper	<div>#FFFFFF (255,255,255)</div>

TABELA 1 : PALETE DE CORES - ADIÇÕES FASE 3




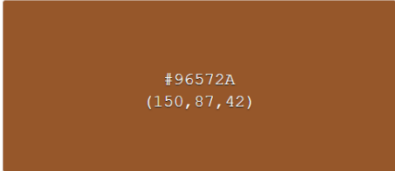

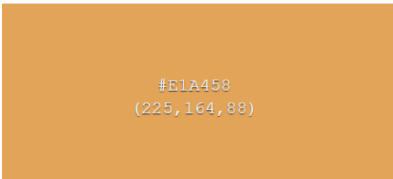

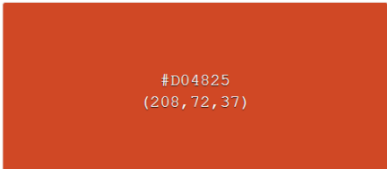


ASTRO	COR	ASTRO	COR
Sol	 #FFCE00 (255, 206, 0)	Marte	 #D04825 (208, 72, 37)
Mercúrio	 #CCBFBC (204, 191, 188)	Júpiter	 #96572A (150, 87, 42)
Vénus	 #C7B465 (199, 180, 101)	Saturno	 #E1A458 (225, 164, 88)
Terra	 #4169E1 (65, 105, 225)	Úrano	 #D04825 (208, 72, 37)
Lua	 #F4EEEE (244, 238, 235)	Neptuno	 #2F2FD6 (47, 47, 214)

TABELA 2: PALETE DE CORES – FASE 2

CONCLUSÃO

Finalizada a terceira fase do projeto, passamos agora a discussão crítica do trabalho realizado.

No espectro positivo, consideramos importante destacar o correto funcionamento do programa, atendendo aos requisitos mencionados. Além disso, os VBOs introduziram um ganho de eficiência ao trabalho, as novas cinturas de asteroides e de Kuiper tornam o resultado final mais apelativo visualmente e o movimento da camara através do rato adicionado nesta fase facilita a compreensão da cena.

Por outro lado, sentimos que poderíamos melhorar a orbita dos planetas, isto é, aproximando-a mais do real ao adotar por uma órbita mais elíptica invés de circular.

Em geral, consideramos que o balanço do trabalho é positivo, as dificuldades sentidas foram superadas, os requisitos propostos cumpridos, os extras são relevantes e os aspetos a melhor podem ser facilmente cumpridos na próxima fase.

ANEXOS

solarSystem.xml

```
<scene>
  <group>
    <rotate time="1" x="0" y="1" z="0" />
    <color x="1" y="0.80784313725490196" z="0"/>
    <models>
      <model file="sphere.3d"/>
    </models>
  </group>
  <group>
    <rotate time="2" x="0" y="1" z="0" />
    <translate x="1.5" y="0" z="0"/>
    <group>
      <rotate time="30" x="0" y="1" z="0" />
      <scale x="0.11" y="0.11" z="0.11"/>
      <color x="0.772549019607843137"
y="0.6980392156862745" z="0.6"/>
      <models>
        <model file="sphere.3d"/>
      </models>
    </group>
  </group>
  <group>
    <rotate time="6" x="0" y="-1" z="0" />
    <translate x="2.1" y="0" z="0"/>
    <group>
      <rotate time="80" x="0" y="-0.9" z="0" />
      <scale x="0.18" y="0.18" z="0.18"/>
      <color x="0.8" y="0.7490196078" z="0.737254902"/>
      <models>
        <model file="sphere.3d"/>
      </models>
    </group>
  </group>
</scene>
```

```

</group>
<group>
  <rotate time="12" x="0" y="1" z="0" />
  <translate x="3.1" y="0" z="0"/>
  <group>
    <rotate time="10" x="0" y="0.7" z="0" />
    <scale x="0.19" y="0.19" z="0.19"/>
    <color      x="0.2549019607843"      y="0.411764705882"
z="0.8823529411764"/>
    <models>
      <model file="sphere.3d"/>
    </models>
  </group>
  <group>
    <rotate time="16" x="0" y="1" z="0" />
    <translate x="1.5" y="0.5" z="0"/>
    <group>
      <rotate time="9" x="0" y="1" z="0" />
      <scale x="0.33" y="0.33" z="0.33"/>
      <color      x="0.95686274509803921568"
y="0.9333333333333333" z="0.92156862745098039"/>
      <models>
        <model file="sphere.3d"/>
      </models>
    </group>
  </group>
</group>
<group>
  <rotate time="18" x="0" y="1" z="0" />
  <translate x="4.1" y="0" z="0"/>
  <group>
    <rotate time="10" x="0" y="0.7" z="0" />
    <scale x="0.13" y="0.13" z="0.13"/>
    <color      x="0.8156862745098039"      y="0.28235294117647"
z="0.14509803921568"/>
    <models>

```

```

        <model file="sphere.3d"/>
    </models>
    <group>
        <rotate time="15" x="0" y="1" z="0" />
        <translate x="1.5" y="0.5" z="0"/>
        <group>
            <rotate time="11" x="0" y="1" z="0" />
            <scale x="0.33" y="0.33" z="0.33"/>
            <color
x="0.65098039215686274509803921568627"
y="0.5960784313725490196078431372549"
z="0.52156862745098039215686274509804"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
    <group>
        <rotate time="19" x="1" y="1" z="0" />
        <translate x="1.5" y="-0.5" z="0"/>
        <group>
            <rotate time="11" x="0" y="1" z="0" />
            <scale x="0.33" y="0.33" z="0.33"/>
            <color
x="0.65098039215686274509803921568627"
y="0.5960784313725490196078431372549"
z="0.52156862745098039215686274509804"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
</group>
<group>
    <rotate time="60" x="0" y="1" z="0" />
    <color x="1" y="1" z="1"/>

```

```

    <models>
        <model file="points.3d"/>
    </models>
</group>
<group>
    <rotate time="36" x="0" y="1" z="0" />
    <translate x="6.5" y="0" z="0"/>
    <group>
        <rotate time="6" x="0" y="1" z="0" />
        <scale x="0.5" y="0.5" z="0.5"/>
        <color      x="0.5882352941176"      y="0.3411764705882"
z="0.16470588235294"/>
        <models>
            <model file="sphere.3d"/>
        </models>
        <group>
            <rotate time="22" x="0" y="0" z="1" />
            <translate x="1.5" y="0.5" z="0"/>
            <group>
                <rotate time="10" x="0" y="1" z="0" />
                <scale x="0.15" y="0.15" z="0.15"/>
                <color
x="0.91764705882352941176470588235294"
y="0.85490196078431372549019607843137"
z="0.25098039215686274509803921568627"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
</group>
<group>
    <rotate time="21" x="1" y="0" z="0" />
    <translate x="1.5" y="-0.2" z="0"/>
    <group>
        <rotate time="10" x="0" y="1" z="0" />
        <scale x="0.14" y="0.14" z="0.14"/>

```



```

x="0.91764705882352941176470588235294"
y="0.90588235294117647058823529411765"
z="0.81960784313725490196078431372549"/>

    <models>
        <model file="sphere.3d"/>
    </models>
</group>
</group>
<group>
    <rotate time="18" x="1" y="1" z="0" />
    <translate x="1.5" y="-0.5" z="0"/>
    <group>
        <rotate time="10" x="0" y="1" z="0" />
        <scale x="0.16" y="0.16" z="0.16"/>
    </group>
    <color
x="0.68235294117647058823529411764706"
y="0.55686274509803921568627450980392"
z="0.36078431372549019607843137254902"/>
    <models>
        <model file="sphere.3d"/>
    </models>
</group>
</group>
<group>
    <rotate time="20" x="0" y="1" z="0" />
    <translate x="1.5" y="0.3" z="0"/>
    <group>
        <rotate time="10" x="0" y="1" z="0" />
        <scale x="0.15" y="0.15" z="0.15"/>
    </group>
    <color
x="0.82352941176470588235294117647059"
y="0.44705882352941176470588235294118"
z="0.13333333333333333333333333333333"/>
    <models>
        <model file="sphere.3d"/>
    </models>
</group>

```

```

        </group>
    </group>
</group>
<group>
    <rotate time="48" x="0" y="1" z="0" />
    <translate x="8" y="0" z="0"/>
    <group>
        <rotate time="5" x="0" y="1" z="0" />
        <scale x="0.45" y="0.45" z="0.45"/>
        <color
            x="0.890196078431372549"
y="0.64313725490196078" z="0.3333333333333333"/>
        <models>
            <model file="sphere.3d"/>
        </models>
    </group>
    <group>
        <rotate angle="90" x="1" y="0" z="0.7"/>
        <scale x="1.2" y="1.2" z="1.2"/>
        <color
            x="0.62745098039215"
y="0.321568627450980" z="0.17647058823529411"/>
        <models>
            <model file="ring.3d"/>
        </models>
    </group>
    <group>
        <rotate time="25" x="0" y="1" z="1" />
        <translate x="1.5" y="-0.5" z="0"/>
        <group>
            <rotate time="7" x="0" y="1" z="0" />
            <scale x="0.18" y="0.18" z="0.18"/>
            <color
x="0.89019607843137254901960784313725"
y="0.86274509803921568627450980392157"
z="0.82745098039215686274509803921569"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>

```

```

</group>
  <group>
    <rotate time="22" x="0" y="1" z="0" />
    <translate x="1.5" y="0.3" z="0"/>
    <group>
      <rotate time="7" x="0" y="1" z="0" />
      <scale x="0.19" y="0.19" z="0.19"/>
      <color
x="0.90588235294117647058823529411765"
y="0.75294117647058823529411764705882"
z="0.09019607843137254901960784313725"/>
      <models>
        <model file="sphere.3d"/>
      </models>
    </group>
  </group>
  <group>
    <rotate time="18" x="1" y="1" z="0" />
    <translate x="1.5" y="-0.5" z="0"/>
    <group>
      <rotate time="7" x="0" y="1" z="0" />
      <scale x="0.17" y="0.17" z="0.17"/>
      <color
x="0.89019607843137254901960784313725"
y="0.86274509803921568627450980392157"
z="0.82745098039215686274509803921569"/>
      <models>
        <model file="sphere.3d"/>
      </models>
    </group>
  </group>
</group>
<group>
  <rotate time="60" x="0" y="-1" z="0" />
  <translate x="9.5" y="0" z="0"/>
</group>

```

```

    <rotate time="7" x="0" y="0" z="1" />
    <scale x="0.38" y="0.38" z="0.38"/>
    <color                                x="0.372549019607843137"
y="0.61960784313725490" z="0.62745098039215686"/>
    <models>
        <model file="sphere.3d"/>
    </models>
    <group>
        <rotate time="21" x="0" y="1" z="1" />
        <translate x="1.5" y="0.2" z="0"/>
        <group>
            <rotate time="10" x="0" y="1" z="0" />
            <scale x="0.19" y="0.19" z="0.19"/>
            <color
x="0.89019607843137254901960784313725"
y="0.86274509803921568627450980392157"
z="0.82745098039215686274509803921569"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
    <group>
        <rotate time="20" x="0" y="1" z="0" />
        <translate x="1.5" y="0.4" z="0"/>
        <group>
            <rotate time="10" x="0" y="1" z="0" />
            <scale x="0.20" y="0.20" z="0.20"/>
            <color
x="0.89019607843137254901960784313725"
y="0.86274509803921568627450980392157"
z="0.82745098039215686274509803921569"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
</group>

```

```

</group>
<group>
  <rotate time="78" x="0" y="1" z="0" />
  <translate x="10.75" y="0" z="0"/>
  <group>
    <rotate time="6" x="0" y="1" z="0" />
    <scale x="0.37" y="0.37" z="0.37"/>
    <color x="0.184313725490196" y="0.184313725490196"
z="0.839215686274509"/>
    <models>
      <model file="sphere.3d"/>
    </models>
    <group>
      <rotate time="20" x="0" y="1" z="0" />
      <translate x="1.5" y="0.5" z="0"/>
      <group>
        <rotate time="4" x="0" y="1" z="0" />
        <scale x="0.21" y="0.21" z="0.21"/>
        <color
x="0.95686274509803921568627450980392"
y="0.85490196078431372549019607843137"
z="0.42352941176470588235294117647059"/>
        <models>
          <model file="sphere.3d"/>
        </models>
      </group>
    </group>
  </group>
</group>
<group>
  <rotate time="60" x="0" y="1" z="0" />
  <color x="1" y="1" z="1"/>
  <models>
    <model file="pointsKuiper.3d"/>
  </models>
</group>
<group>

```

```
<translate time="5">
    <point x="-1" y="0" z="-4"/>
    <point x="-5" y="0" z="4"/>
    <point x="5" y="5" z="4"/>
    <point x="5" y="0" z="0"/>
</translate>
<scale x="0.1" y="0.1" z="0.1"/>
<color x="0.5" y="0.9" z="0.4"/>
<models>
    <model file="comet.3d"/>
</models>
</group>
</scene>
```