Universidade do Minho
Escola de Engenharia

COMPUTAÇÃO GRÁFICA

PHASE 2 - GEOMETRIC TRANSFORMS

GRUPO 30 - ABRIL 2021

Luís Miguel Pinto A89506

Ana Luísa Carneiro A89533

Pedro Almeida Fernandes A89574

Ana Rita Peixoto A89612

ÍNDICE

INTRODUÇÃO

O objetivo da 2ª fase do trabalho prático consistiu em desenvolver um cenário gráfico 3D representativo do sistema solar através da leitura e interpretação de um ficheiro XML e do uso da primitiva gráfica, esfera, criada na primeira fase do projeto.

Nesta fase foi necessário atualizar a aplicação Engine criada na fase anterior de forma que esta interpretasse os novos constituintes do ficheiro XML, armazenando os dados lidos numa estrutura adequada e, em seguida, desenhar todos os modelos e transformações em 3D.

Além dos objetivos propostos no enunciado, achamos por bem acrescentar algumas funcionalidades e características ao nosso projeto, como a rotação da câmara, a criação de uma nova primitiva, torus, a adição dos anéis de saturno e das cinturas de asteroides e de *Kuiper*. Assim, tornamos a representação do sistema solar mais completa e personalizada.

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.

FICHEIRO XML

De modo a cumprir com os requisitos da segunda fase, foi necessário criar um ficheiro XML capaz de representar toda a informação pretendida para o desenho da cena. Através da hierarquização de cenas e com o auxílio de transformações geométricas, é possível criar inúmeros cenários com diferentes elementos.

ESTRUTURA DO FICHEIRO

Qualquer ficheiro XML usado para representar uma cena, tem na sua essência uma estrutura em árvore, onde cada nodo contém um conjunto de transformações geométricas e/ou um conjunto de modelos (ficheiros.3d). Além disso, cada nodo pode ainda ter nodos filho aninhados que herdam as transformações dos nodos antecessores.

No que diz respeito às transformações, existem no total três transformações geométricas que devemos aplicar de forma ordenada para verdadeiramente atingir o resultado pretendido (*rotate*, *translate* e *scale*, nesta ordem) e ainda uma transformação para alteração da cor dos modelos (*color*).

Em seguida, apresentamos um exemplo demonstrativo de um ficheiro XML:

```
<group>
    <rotate angle="172" x="0" y="1" z="0"/>
    <translate x="3.1" y="0" 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>
        <translate x="1.5" y="0.5" z="0"/>
        <scale x="0.33" y="0.33" z="0.33"/>
        <color x="0.9568627450980392156" y="0.93333333333333333" z="0.92156862745098039"/>
        <models>
            <model file="sphere.3d"/>
        </models>
    </group>
</group>
```

Figura 1: Exemplo da formatação XML

Este excerto do ficheiro XML criado tem como funcionalidade representar as diretrizes necessárias para o desenho da terra e da lua.

Como podemos verificar ainda na figura acima, cada nodo corresponde a um ***group*** que é definido por um conjunto de elementos, quer sejam transformações, modelos ou subgrupos.

A **primitiva *rotate*** permite estabelecer a rotação de um objeto através de um ângulo (angle) e um vetor de rotação (definido por x, y e z).

A **primitiva *translate*** tem como função estabelecer a translação de um objeto através de um vetor (definido por x, y e z).

A **primitiva *scale*** estabelece a escala de um objeto através de escalares (x, y e z).

O **elemento *color*** tem como propósito estabelecer a cor de um objeto dos seus parâmetros RGB (definido por x, y e z, respetivamente).

O **elemento *models*** indica quais os modelos a desenhar.

ENGINE

Nesta fase do projeto foi necessário atualizar o programa *Engine* para que este interprete e processe o novo ficheiro XML uma única vez. Para isso, implementou-se na classe *Engine* um vetor com elementos da classe *Group*. Esta classe denota uma estrutura constituída por vetores de várias outras classes de forma a armazenar a informação do ficheiro. Esta estrutura vai ser povoada nas funções *parseDocument* e *parseGroup* e vai ser posteriormente utilizada para desenhar a *scene* armazenada nessa estrutura.

ESTRUTURA IMPLEMENTADA

Tal como foi mencionado anteriormente, o XML é constituído por diversos grupos que contêm transformações, primitivas (ficheiros 3d) e até outros subgrupos. Nesse sentido, a estrutura implementada para o armazenamento dos dados vai refletir a formatação desse ficheiro. Assim, a estrutura permite a organização da informação pelos grupos através do uso de um vetor *groups* onde cada elemento é da classe *Group*.

A **classe Group** é constituída por toda a informação possível que podemos encontrar nos grupos do ficheiro XML, isto é, transformações, primitivas e outros subgrupos.

Para as transformações implementou-se um vetor *transformations* onde cada elemento é da **classe *Transformations***. Esta classe implementa variáveis que permitem identificar quais as transformações em questão e as suas características, através da criação de 4 variáveis *float* e uma variável do tipo *string* que identifica o tipo de transformação, isto é, *scale*, *translate*, *rotate* ou *color*.

As primitivas existentes são armazenadas num vetor *primitives* que utiliza elementos da **classe *Primitive*** já utilizada na fase 1 do projeto. Nesta classe estão armazenados todos os pontos existentes nos ficheiros 3d através de um vetor *vertices* que contém elementos da classe *Point*.

Finalmente, para representar os subgrupos utilizou-se um vetor da classe *Group*. Assim, cada elemento desse vetor terá a mesma constituição do grupo representando, desta forma, um subgrupo.

Na figura a seguir está representada esquematicamente a estrutura completa que foi implementada.

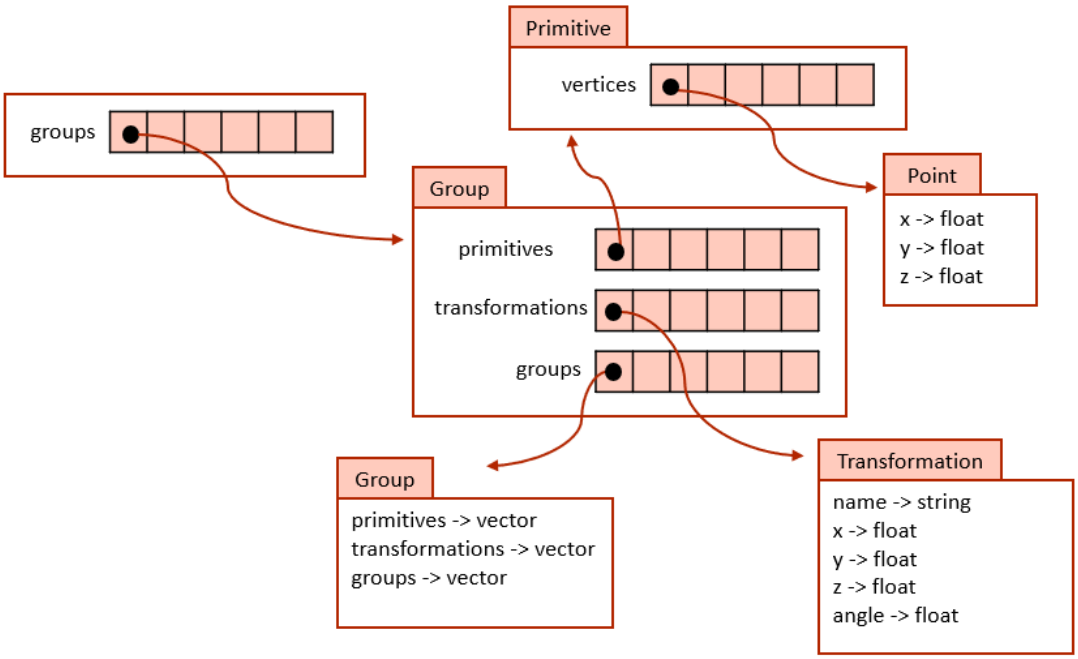

Figura 2: Estrutura Implementada

LEITURA DO FICHEIRO XML

Para a segunda fase do projeto, foi necessário alterar o modo de leitura do ficheiro XML, não só devido ao aumento da sua complexidade, mas também pela alteração da sua estrutura.

Para concretizar tal feito, recorreu-se à utilização da biblioteca tinyXML2 e às funções que esta contém. Deste modo, começou-se pela função **parseDocument** que contém o nome do ficheiro XML em questão. Esta função trata de abrir o ficheiro XML e de obter o *firstChild*/primeiro nodo, que deverá ser identificado com a *string* "*scene*". De seguida e de modo análogo, lê-se o nodo "*group*" do XML e invoca-se a função *parseGroup* que irá tratar de todo a informação presente nesse grupo. É importante realçar também que devido à recursividade da função *parseGroup*, foi necessário acrescentar um argumento identificador de modo a distinguir entre o grupo pai e os grupos filhos.

A função ***parseGroup*** contém um ciclo exterior que itera pelos "*group*" do XML, através de um apontador. Internamente a este ciclo, existe um outro que irá proceder de modo semelhante para os elementos de cada *group*, ou seja, à medida que lê cada elemento presente naquele grupo do XML, vai verificando de qual se trata e armazenando na estrutura. Caso o elemento encontrado seja um subgrupo, a função irá proceder à chamada recursiva dela própria, com a diferença da variável *father* que irá ter o valor 1, para denotar que se trata de um subgrupo.

Por fim, após a leitura do grupo, estamos em posição de o armazenar no vetor de *Group* presente na classe *Engine*, caso este seja um grupo pai.

ParseDocument

```
bool parseDocument() {
        // get MODELS path
        // open XML file (doc)
        XMLNode* scene = doc.FirstChild();

        if (scene == nullptr) {
                cout << "ERRO\n";
                return false;           //in case of error
        }

        XMLElement *group = scene->FirstChildElement("group");

        parseGroup(group,0);            // First evocation of parseGroup
        return true;                    //in case of success
}
```

ParseGroup

```
Group parseGroup(XMLElement *group, int father) {

do {
        // create group g
        // element can be a : transformation, model or subgroup
        // element = get first elemento

        while (element != nullptr) {
                if( element equals model ) {
                        // get model file
                        while (file != nullptr) {
                                //get file path
                                Primitive primitive = readFile(path);
                                g.addPrimitivas(primitive);
                                // file = next file
                        }
                }
                else if ( element equals scale ) {
                        // get scale components
                        Transformation t = Transformation(scale,x,y,z,0);
                        g.addTransformacoes(t);

                } else if ( element equals translate ) {
                        // get translate components
                        Transformation t = Transformation("translate",x,y,z,0);
                        g.addTransformacoes(t);
```

```
            } else if ( element equals rotate ) {
                    // get rotate components
                    Transformation t = Transformation("rotate",x,y,z,angle);
                    g.addTransformacoes(t);

            } else if ( element equals color) {
                    // get color components
                    Transformation t = Transformation("color",red,green,blue,0);
                    g.addTransformacoes(t);

            } else {  // subgroup
                    g.addGrupos(parseGroup(element,1));          // recursive call
            }
            //element = next element
        }
        if( first call of parseGroup) {
                // add group to vector groups
        }
        If (subgroup) return group;
        // group = next group

} while(group != nullptr);
return group;
}
```

DESENHO DO FICHEIRO XML

Para desenhar a *scene* armazenada na estrutura *groups*, foi necessário reescrever a função *drawPrimitive* criada na fase 1 do projeto para que esta percorra a estrutura e desenhe não só as transformações como também as primitivas e todos os subgrupos. A *drawPrimitive* vai ser chamada na função *renderScene* e vai receber como parâmetro uma lista de *Group* que inicialmente será o vetor *groups*.

A função *drawPrimitive* começa por percorrer todos os elementos do vetor que recebe como argumento. Como pretendemos que as transformações de cada grupo principal sejam particulares a esse mesmo grupo, utilizou-se o método *glPushMatriz* para armazenar a matriz de desenho antes que sejam realizadas transformações sobre os eixos. De seguida, percorre-se as transformações e as primitivas de forma que estas sejam desenhadas sobre os eixos alterados.

Para desenhar os subgrupos, caso estes existam, usa-se a função *drawPrimitive* de forma recursiva. Finalmente, usa-se o método *glPopMatriz* para que tanto as transformações como as primitivas do elemento seguinte sejam desenhadas sobre os eixos iniciais.

drawPrimitive

```
void drawPrimitives(vector<struct Group> vector) {

        for ( each group ){
                // push matrix

                for ( each transformation ) {     // geometric or color
                        // get transformation
                        if ( transformation equals translate ) {
                                glTranslatef(x, y, z);
                        } else if ( transformation equals scale )  {
                                glScalef(x, y, z);
                        } else if ( transformation equals rotate )  {
                                glRotated(angle, x, y, z);
                        } else if ( transformation equals color )  {
                                glColor3f(r, g, b);
                        }
                }


                for ( each primitive )  {
                        glBegin(GL_TRIANGLES);
                        for ( each point ) {
                                glVertex3f(x, y, z);
                        }
                        glEnd();
                }
                drawPrimitives( subgroup );     // recursive call
                // pop matrix
        }
}
```

EXTRAS

De modo a complementar o trabalho elaborado nesta fase, consideramos conveniente implementar alguns extras, tais como: rotação da câmara, a criação da nova primitiva Torus no programa Generator e a implementação de luas, cinturas de asteroides e de *Kuiper* e anéis de Saturno.

ROTAÇÃO DA CAMARA

A rotação da câmara foi uma funcionalidade muito útil para a observação da cena em diferentes perspetivas e ângulos. A sua implementação passou por acrescentar duas novas teclas à função *mover*, criada na fase 1, que reage a eventos de teclado alterando as características da câmara como ângulos e distâncias. Ao premir a tecla *PG UP* a câmara afasta-se da *scene* e com o *PG DOWN* a câmara aproxima-se desta. Juntamente com as funcionalidades já implementadas na fase anterior, podemos reposicionar a câmara através das seguintes teclas:

Rotação Direita	Rotação Esquerda	Rotação Cima	Rotação Baixo	Afastar	Aproximar
KEY_RIGHT	KEY_LEFT	KEY_UP	KEY_DOWN	PG_UP	PG_DOWN

Tabela 1: Comandos para rotação da câmara

SPACE RING

De modo a conseguir projetar um anel de poeiras espaciais, implementamos um torus ligeiramente cortado tendo como inspiração código encontrado na biblioteca openGL. A partir desta nova funcionalidade foi possível representar os **anéis de saturno**, **a cintura de asteróides** e até mesmo a **cintura de *Kuiper***. Para isso, recorreu-se às já mencionadas transformações geométricas, translações, escalas e rotações, de modo a posicionar cada primitiva corretamente. Em adição, também foi utilizada uma escolha de cores representativa dos elementos em questão. A concretização desta primitiva ocorreu no *Generator*, cujo código se encontra em anexo.

SISTEMA SOLAR

De forma a cumprir com o propósito desta fase do trabalho, implementamos um modelo estático do sistema solar através de um ficheiro XML personalizado. Este modelo tem em consideração as dimensões relativas de cada planeta de forma que haja um balanço entre uma representação realista e visualmente apelativa e percetível. O ficheiro XML do modelo está organizado conforme a distância dos planetas ao sol, começando pelo sol e indo em direção Neptuno. Além disso, no caso do planeta Terra também foi representado o seu satélite natural, a Lua. Em relação aos planetas Marte, Jupiter, Saturno, Urano e Neptuno também foram

implementadas algumas das suas luas. Finalmente, desenhou-se também o anel de saturno, a cintura de asteróides e a cintura de *Kuiper*.

Na imagem a seguir encontra-se o desenho em 3D do conteúdo presente no ficheiro XML, solarSystem.xml (em anexo).

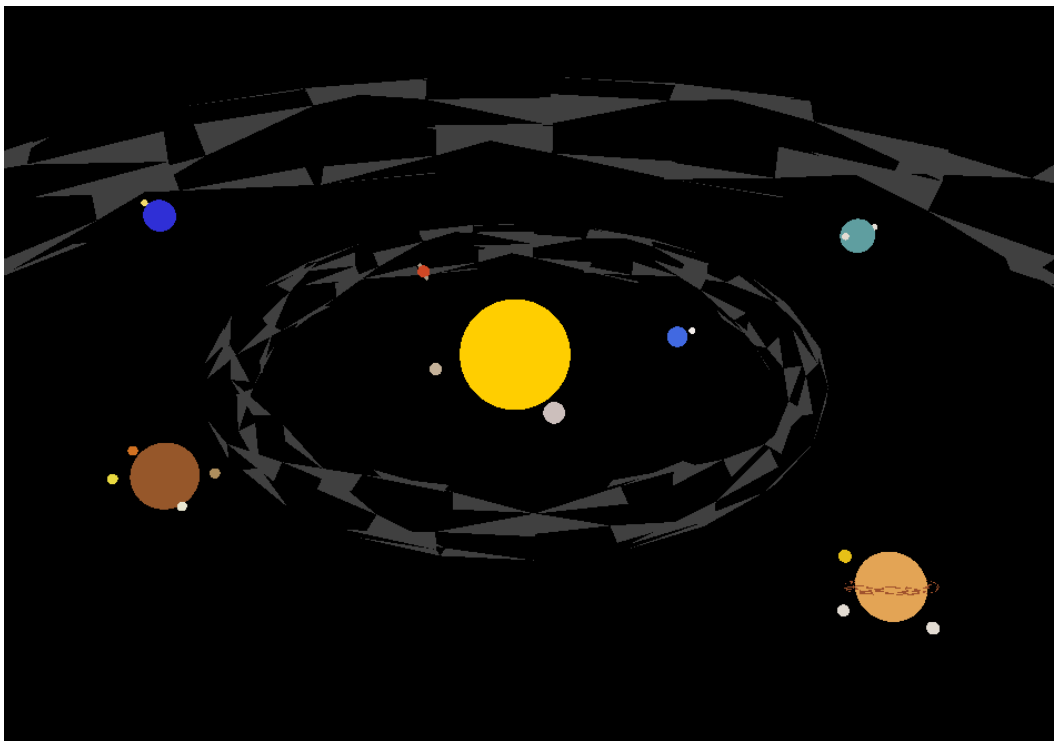

Figura 3: Sistema Solar

VALORES IMPLEMENTADOS

Para concretizar este modelo, recorreu-se à representação em XML de funções do *glut* tais como *rotate*, *scale* e *translate*. Em complemento, também existe uma transformação extra *color* cuja finalidade é denotar as cores de cada planeta. Deste modo foi possível desenhar cada planeta, através de esferas criadas no *Generator*, aproximando a sua cor à realidade com o propósito de aumentar o realismo da cena. As transformações aplicadas tiveram em consideração diversos fatores. No caso do *translate*, foi necessário verificar quais as distâncias entre cada planeta. Para o *scale*, tivemos em conta as dimensões de cada planeta relativas ao sol. No que toca ao *rotate*, a sua utilização teve como propósito colocar os planetas em diferentes fases da sua órbita em torno do sol.

Ao gerar o ficheiro XML, foi necessário também ter em consideração uma aproximação às cores correspondentes a cada planeta. Para isso, recorreu-se ao código de cores RGB e efetuou-se a divisão dos valores desejados por 255 de modo a obter compatibilidade com o *openGL*. Na tabela abaixo encontram-se as cores utilizadas para cada um dos astros implementados.

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**	#5F9EA0 (95,158,160)
Lua	#F4EEEB (244,238,235)	**Neptuno**	#2F2FD6 (47,47,214)
EXTRA	**COR**	**EXTRA**	**COR**
Cinturas	#414141 (65,65,65)	**Anel de saturno**	#A0522D (160,82,45)

Tabela 2: Palete de cores

CONCLUSÃO

Dada por concluída a fase 2 do projeto, consideramos relevante efetuar uma análise crítica do trabalho realizado.

No espetro positivo, consideramos conveniente destacar o correto funcionamento do nosso programa. Além disso, as estruturas implementadas estão em concordância com a estrutura do XML o que permite uma visualização mais clara daquilo que está a ser armazenado. A implementação de funcionalidades adicionais no movimento da câmara também se revelou um aspeto útil.

Por outro lado, também existiram algumas dificuldades, tais como a familiarização com as funções do *tinyXML2* e a implementação de funções recursivas. Apesar disso, as dificuldades foram ultrapassadas.

Para concluir, consideramos que houve um balanço positivo do trabalho realizado dado que as dificuldades sentidas foram superadas e foram cumpridos todos os requisitos.

ANEXOS

solarSystem.xml

```
<scene>
    <group>
        <translate x="0" y="0" z="0"/>
        <color x="1" y="0.80784313725490196" z="0"/>
        <models>
            <model file="sphere.3d"/>
        </models>
    </group>
    <group>
        <translate x="1.5" y="0" z="0"/>
        <scale x="0.11" y="0.11" z="0.11"/>
        <color x="0.77254901960784" y="0.698039215686" z="0.6"/>
        <models>
            <model file="sphere.3d"/>
        </models>
    </group>
    <group>
        <rotate angle="87" x="0" y="1" z="0"/>
        <translate x="2.1" y="0" 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>
        <rotate angle="172" x="0" y="1" z="0"/>
        <translate x="3.1" y="0" z="0"/>
        <scale x="0.19" y="0.19" z="0.19"/>
        <color x="0.254901960" y="0.41176470" z="0.882352941"/>
        <models>
            <model file="sphere.3d"/>
        </models>
        <group>
```

```
            <translate x="1.5" y="0.5" z="0"/>
            <scale x="0.33" y="0.33" z="0.33"/>
            <color x="0.956862745" y="0.933333333" z="0.921568627"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
    <group>
        <rotate angle="277" x="0" y="1" z="0"/>
        <translate x="4.1" y="0" z="0"/>
        <scale x="0.13" y="0.13" z="0.13"/>
        <color x="0.815686274" y="0.28235294117647" z="0.145098039"/>
        <models>
            <model file="sphere.3d"/>
        </models>
            <group>
            <translate x="1.5" y="0.5" z="0"/>
            <scale x="0.33" y="0.33" z="0.33"/>
            <color x="0.650392154" y="0.596078449" z="0.5215686274"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
            <group>
                  <rotate angle="173" x="0" y="1" z="0"/>
            <translate x="1.5" y="-0.5" z="0"/>
            <scale x="0.33" y="0.33" z="0.33"/>
            <color x="0.650980392" y="0.59607843" z="0.521568627"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
    <group>
        <rotate angle="90" x="1" y="0" z="0"/>
        <scale x="5" y="5" z="5"/>
        <color x="0.250980392156" y="0.2509803921568" z="0.250980"/>
```

```
        <models>
            <model file="ring.3d"/>
        </models>
    </group>
    <group>
        <rotate angle="15" x="0" y="1" z="0"/>
        <translate x="6.5" y="0" z="0"/>
        <scale x="0.5" y="0.5" z="0.5"/>
        <color x="0.5882" y="0.3411764705882" z="0.16470588"/>
        <models>
            <model file="sphere.3d"/>
        </models>
            <group>
            <translate x="1.5" y="0.5" z="0"/>
            <scale x="0.15" y="0.15" z="0.15"/>
            <color x="0.91764705" y="0.854901960" z="0.250980"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
            <group>
                  <rotate angle="87" x="0" y="1" z="0"/>
            <translate x="1.5" y="-0.2" z="0"/>
            <scale x="0.14" y="0.14" z="0.14"/>
            <color x="0.91764705" y="0.905882352941" z="0.8196078"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
            <group>
                  <rotate angle="179" x="0" y="1" z="0"/>
            <translate x="1.5" y="-0.5" z="0"/>
            <scale x="0.16" y="0.16" z="0.16"/>
            <color x="0.6823529411" y="0.55686274509" z="0.36078431"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
```

```
            <group>
                  <rotate angle="287" x="0" y="1" z="0"/>
            <translate x="1.5" y="0.3" z="0"/>
            <scale x="0.15" y="0.15" z="0.15"/>
            <color x="0.82352941176" y="0.4470588" z="0.133333333"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
    <group>
        <rotate angle="107" x="0" y="1" z="0"/>
        <translate x="8" y="0" z="0"/>
        <scale x="0.45" y="0.45" z="0.45"/>
        <color x="0.890196078" y="0.643137" z="0.333"/>
        <models>
            <model file="sphere.3d"/>
        </models>
        <group>
            <rotate angle="90" x="1" y="0" z="0.7"/>
            <scale x="1.2" y="1.2" z="1.2"/>
            <color x="0.6274509803" y="0.321568" z="0.17647"/>
            <models>
                <model file="ring.3d"/>
            </models>
        </group>
            <group>
            <translate x="1.5" y="-0.5" z="0"/>
            <scale x="0.18" y="0.18" z="0.18"/>
            <color x="0.8901960784" y="0.86274509" z="0.8274509803"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
            <group>
                  <rotate angle="190" x="0" y="1" z="0"/>
            <translate x="1.5" y="0.3" z="0"/>
            <scale x="0.19" y="0.19" z="0.19"/>
```

```
            <color x="0.9058823529" y="0.7529411764" z="0.090196078"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
            <group>
                 <rotate angle="250" x="0" y="1" z="0"/>
            <translate x="1.5" y="-0.5" z="0"/>
            <scale x="0.17" y="0.17" z="0.17"/>
            <color x="0.890196078" y="0.8627450980" z="0.8274509809"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
    <group>
        <rotate angle="195" x="0" y="1" z="0"/>
        <translate x="9.5" y="0" z="0"/>
        <scale x="0.38" y="0.38" z="0.38"/>
        <color x="0.37254901" y="0.6196078" z="0.6274509803"/>
        <models>
            <model file="sphere.3d"/>
        </models>
            <group>
            <translate x="1.5" y="0.2" z="0"/>
            <scale x="0.19" y="0.19" z="0.19"/>
            <color x="0.890196078" y="0.86274509803" z="0.827450"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
            <group>
                 <rotate angle="190" x="0" y="1" z="0"/>
            <translate x="1.5" y="0.4" z="0"/>
            <scale x="0.20" y="0.20" z="0.20"/>
            <color x="0.8901960" y="0.862745098039" z="0.82745098"/>
            <models>
                <model file="sphere.3d"/>
```

```
            </models>
        </group>
    </group>
    <group>
        <rotate angle="300" x="0" y="1" z="0"/>
        <translate x="10.75" y="0" z="0"/>
        <scale x="0.37" y="0.37" z="0.37"/>
        <color x="0.18431372549" y="0.184313" z="0.839215"/>
        <models>
            <model file="sphere.3d"/>
        </models>
            <group>
            <translate x="1.5" y="0.5" z="0"/>
            <scale x="0.21" y="0.21" z="0.21"/>
            <color x="0.9568627" y="0.85490196" z="0.4235294117"/>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
    </group>
    <group>
        <rotate angle="90" x="1" y="0" z="0"/>
        <scale x="14" y="14" z="14"/>
        <color x="0.25098039215" y="0.250980392" z="0.2509803"/>
        <models>
            <model file="ring.3d"/>
        </models>
    </group>
</scene>
```

CÓDIGO SPACE RING

```
bool creatRing(vector<string> params) {
    int numc, numt;
    numc = stod(params[0]);
    numt = stod(params[1]);

    int i, j, k;
    double s, t, x, y, z, twopi;

    string res;

    twopi = 2 * M_PI;
    for (i = 0; i < numc; i++) {
        for (j = 0; j <= numt; j++) {
            for (k = 1; k >= 0; k--) {
                s = (i + k) % numc + 0.5;
                t = j % numt;

                x = (1 + .1 * cos(s * twopi / numc)) * cos(t * twopi /
numt);
                y = (1 + .1 * cos(s * twopi / numc)) * sin(t * twopi /
numt);
                z = .1 * sin(s * twopi / numc);

            res = res + to_string(x) + "," + to_string(y) + "," +
            to_string(z) + "\n";
            }
        }
    }

    int tam = count(res.begin(), res.end(), '\n');
    string resultado = to_string(tam) + "\n" + res;

    writeInFile(resultado, params[2]);

    return true;
}
```