Computação Gráfica: Etapa Final

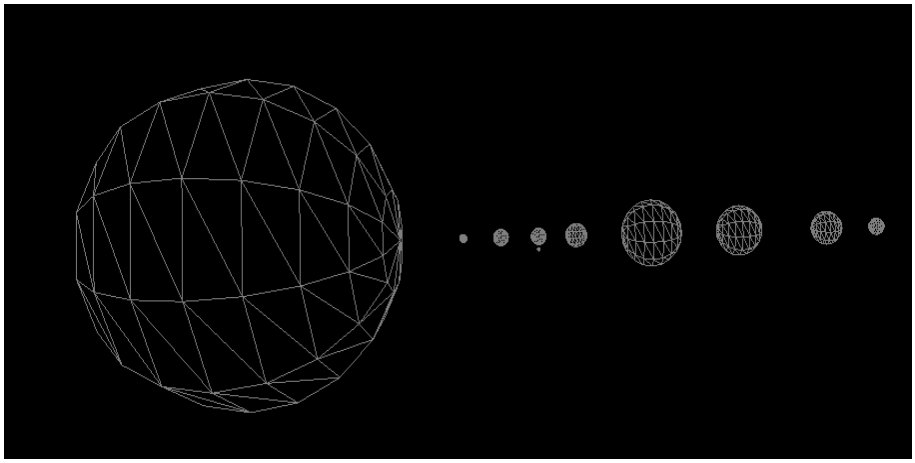
Bernardo Rodrigues
a79008@alunos.uminho.pt

César Silva
a77518@alunos.uminho.pt

Pedro Faria
a82725@alunos.uminho.pt

Rui Silva
a77219@alunos.uminho.pt

Universidade do Minho — 7 de Maio de 2019



Resumo

A **Computação Gráfica**, uma das muitas áreas da informática, assume um papel fulcral em interações homem-máquina e na visualização de dados. O seu espectro de aplicações varia desde geração de imagens e animações até a simulação do mundo real.

O presente relatório refere-se às diferentes fases de entrega da componente prática da Unidade Curricular de **Computação Gráfica** enquadrada no curso de *Ciências da Computação* da *Universidade do Minho*.

Conteúdo

1 Introdução

Na primeira fase, foram desenvolvidas duas aplicações. Um **Gerador** de modelos, que aceite argumentos a partir do terminal, com a função de gerar os pontos de uma primitiva gráfica desejada pelo utilizador, imprimindo estes para um ficheiro. E a última, um **Motor** que interpreta os pontos gerados anteriormente de acordo com um ficheiro de configuração dado. Com base nos tópicos enunciados e ferramentas exploradas nas aulas implementamos o **Gerador** e o **Motor** na linguagem **C++**. Em particular esta última faz uso de biblioteca **TinyXML2** para que a leitura dos documentos que lhe são dados com input seja feita de forma simples e consistente. E por fim, utilizamos a API fornecida pelo **OpenGl** para dar vida aos nossos modelos.
Na segunda, adicionamos features ao parser de ficheiros de configuração de **scenes**, nomeadamente, o reconhecimento de **scenes** dispostas hierarquicamente usando transformações geométricas (translações, rotações e de escala).// Na terceira fase, **gerador** passa a usar patches de **Bezier**, melhoramos as transformações do **motor** com curvas de **Catmull-Rom** e os modelos passam a ser desenhados com **VBO's**. E na última fase, implementamos texturas e diferentes modos de iluminação.

2 Primeira Fase

2.1 Gerador

O nosso **Gerador** é um pequeno programa em **C++**, cuja implementação é disponibilizada em anexo, que depois de compilado, o correspondente executável escreve para um ficheiro (um por linha) os vértices da primitiva gráfica desejada como iremos ilustrar nas seguintes secções.

2.1.1 Caixa

Para geração desta primitiva o utilizador deverá invocar a aplicação a com o nome do executável seguido dos comprimentos dos lados de um paraleloipipedo no eixos com X's, Y's e Z's respetivamente e por fim o nome do ficheiro destino. A sintaxe é demonstrada no exemplo abaixo:

Command Line

```
$ g++ -o gerador gen.cpp
$ ./gerador caixa 3 4 5 osmeusvertices
$ ls
$ gen.cpp            gerador            osmeusvertices.txt
```

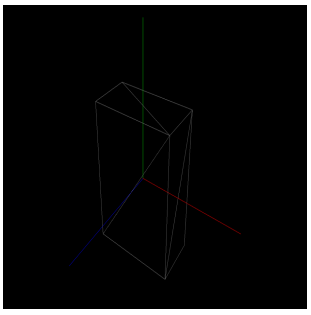


Figura 1: Caixa

Notice: Todas as faces são desenhadas com dois triângulos apontados para o exterior pela norma da mão direita mencionada nas aulas. Um zoom excessivo, ou a escolha de dimensões superiores à distância da câmara à origem(onde este é centrado) pode levar ao aparente desaparecimento do modelo.

2.1.2 Cone

O **Cone** recebe como argumentos o raio da base, a sua altura, o número de slices e stacks.
A construção deste começa por fixar um uma *slice* calculando os vértices da base correspondente, de seguida todas as *stacks* relativas, sendo o última stack - a do bico - um caso especial.
O algoritmo usa noções como semelhança de triângulos para cálculo dos sucessivos raios das circuferências formadas pelas *stacks*.

Info: Apresentamos o significado das variáveis usadas no programa que gera os pontos do **Cone**:

stkd - diferença entre stacks consecutivas

slcd - diferença entre slices consecutivas

raiod - diferença entre o raio de duas stacks consecutivas

stk - stack atual

slc - slice atual

nslc - próxima slice

nstk - próxima stack

nr - próximo raio

r - raio atual

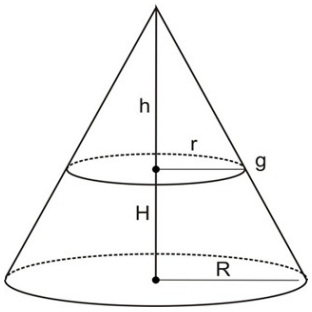
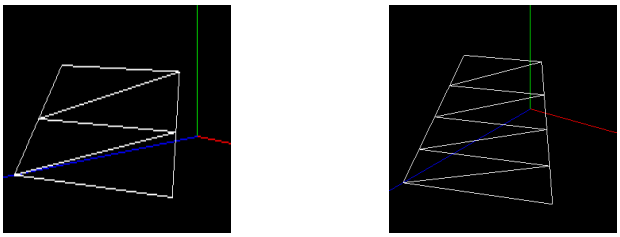


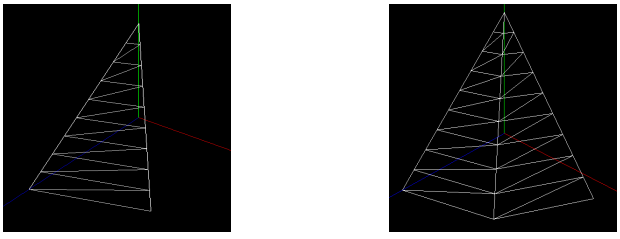
Figura 2: Semelhança de triângulos num cone

Apresentamos algumas imagens que ilustram a criação do cone.



(a) 2 stacks (b) 4 stacks

Figura 3: Progessão das stacks do cone



(a) 1 slice (b) 2 slices

Figura 4: Progessão das slices do cone

Finalmente obtemos:

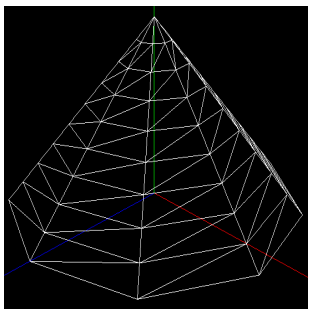


Figura 5: Cone

2.1.3 Esfera

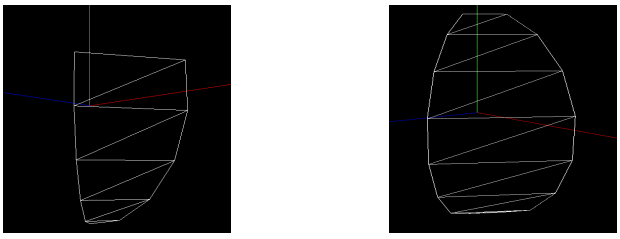
A **Esfera** recebe como argumentos o seu raio, o número de slices e stacks. A sua construção usa coordenadas esféricas usando o raio dado como argumentos e manipulando 2 ângulos *Alfa* e *Beta*. *Alfa* é dado por:

$$alfa = \frac{2 \times \Pi}{slices}$$

Este nos programas é usado juntamente com o raio para calcular os pontos de *slices* consecutivas. De seguida *beta* é dado por:

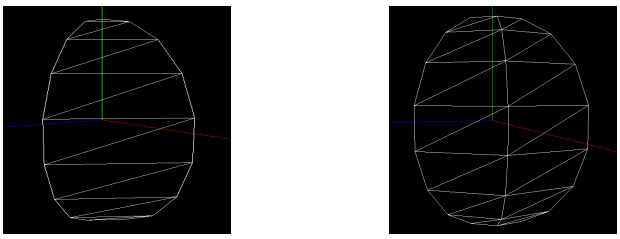
$$beta = \frac{\Pi \div 2}{stacks}$$

Este desenpenha uma função igual ao anterior considerando *stacks*.



(a) 4 stacks (b) 8 stacks

Figura 6: Progessão das stacks da esfera



(a) 1 slice (b) 2 slices

Figura 7: Progessão das slices da esfera

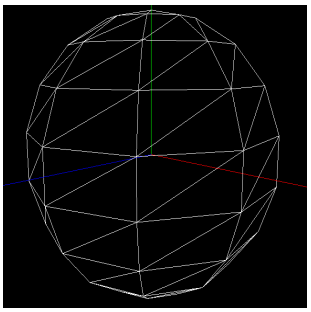


Figura 8: Esfera

2.1.4 Plano

O requisito estabelecido no guião do trabalho veio em muito simplificar a representação do plano XZ ao ponto de para a sua computação seja apenas necessário um único argumento que representa o tamanho da porção visível desejada.

Command Line

```
$ g++ -o gerador gen.cpp
$ ./gerador plano 5 pontos
$ ls
$ gen.cpp            gerador            pontos.txt
```

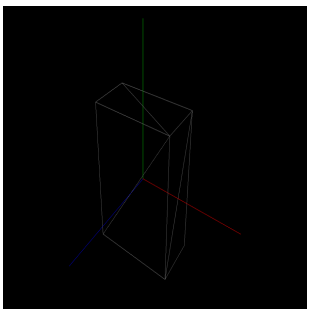


Figura 9: Caixa

Notice: Os planos são infinitos a referência ao tamanho é feita apenas para facilitar a observação do mesmo. São desenhados 4 triângulos, em vez de dois, para que esta primitiva seja visível de todas as perspectivas.

2.2 Motor

O motor é a parte do nosso trabalho que faz o linking de todas as outras partes que foram realizadas. O motor, lê um ficheiro xml (conf.xml). Este ficheiro contém uma estrutura bastante simples, dividida em scenes.

conf.xml

```
        <scene>
                <model file="esfera.txt" />
                <model file="cone.txt" />
                <model file="caixa.txt" />
        </scene>
```

Cada ficheiro que está referenciado nas tags **<model>** é um ficheiro de texto criado pelo nosso gerador e contém todos os pontos das figuras que queremos representar. Como o motor lê os ficheiros de pontos a partir dum ficheiro **XML** utilizamos um parser xml para **C++** chamado **tinyxml-2**.

main.cpp

```
...
tinyxml2::XMLDocument doc;

doc.LoadFile("./conf.xml");

tinyxml2::XMLNode *scene = doc.FirstChild();

tinyxml2::XMLElement* model;

while(scene) {
        for(model = scene->FirstChildElement();
        model != NULL;
        model = model->NextSiblingElement()) {
                const char * file;
                file = model->Attribute("file");
                guardaPontos(file);
        }
        scene = scene->NextSiblingElement();
}
...
```

Com este snippet de código, criamos um objeto do tipo XMLDocument. De seguida, com a função **LoadFile**, abrimos o ficheiro de configuração **conf.xml**, e começamos a manipular o seu conteúdo. Criamos um objeto do tipo **XMLNode** e associamos-lhe a primeira **tag** do ficheiro conf.xml que é a raiz da estrutura do nosso ficheiro. No ciclo while, percorremos todos o ChildElements de scene, que são as tags que guardam os nossos ficheiros das figuras. Cada ficheiro de figura tirado dos **models** é passado à função **guardaPontos**, que será explicada de seguida.

main.cpp

```
...
struct Pontos {
    float a;
    float b;
    float c;
};

std::vector<Pontos> pontos;

void guardaPontos(std::string ficheiro) {
        std::ifstream file;
        std::string s = "./";
        s.append(ficheiro.c_str());
        file.open(s.c_str());
        float a,b,c;
        while(file >> a >> b >> c) {
                Pontos aux;
                aux.a = a;
                aux.b = b;
                aux.c = c;
                pontos.push_back(aux);
        }
}
...
```

Criamos uma estrutura **Pontos** que tem como campos 3 *floats*, que servem para registar as coordenadas destes. De seguida usamos um vector que utiliza a estrutura enunciada para os guardar. À função **guardaPontos** são passados nomes de ficheiros. A função abre os ficheiros e lê pontos linha a linha, guardando cada coordenada *x, y e z* num **Ponto** e de seguida inserindo-o no vector.

Info: A instrução:

```
file >> a >> b >> c
```

associa a cada uma das variaveis *a, b* e *c*, as coordenadas da linha que está a ser lida em cada iteração do ciclo.

Por ultimo temos a função **printPontos**:

main.cpp

```
...
void printPontos(std::vector<Pontos> pontos) {
  for(int i = 0; i < pontos.size(); i++) {
                glVertex3f(pontos[i].a,
                        pontos[i].b,
                        pontos[i].c);
        }
}
...
```

Esta função é chamada na renderScene e geras todos os pontos percorrendo o vector.

3 Segunda Fase

Após ponderarmos o enunciado, o grupo, decidiu implementar uma classe, em **C++**, inspirada numa estrutura de dados largamente conhecida na área, denominada por **Scene Graph**.
A origens desta estrutura de dados remonta aos primórdios dos primeiros jogos de vídeo sobre simulação de voo mas actualmente é vulgarmente incluída qualquer aplicação que lide com **Graphic Rendering**.
Esta estrutura de dados foi revolucionária pelo facto de reduzir drasticamente a memória necessária em cálculos sistemáticos sobre o mundo que está a tentar representar. Os cálculos passaram a ser feitos **in place** na estrutura dispensando na totalidade a necessidade de memória auxiliar.

3.1 A classe Scene Graph

Apesar de disponibilizarmos a implementação em anexo iremos contemplar alguns pormenores neste capítulo.
Face a como os ficheiros **XML** são organizados via uma *árvore - DOM Tree* - implementamos uma classe que usa os princípios de essa mesma estrutura para guardar os dados de tudo o que é exposto na cena.
Por exemplo se quiséssemos desenhar um cavalo e o seu cavaleiro, não de maneira independente, mas como se o cavalo fosse uma extensão do seu cavaleiro. O **Scene Graph** correspondente teria no nodo **Cavalo** "pendurado" no nodo **Cavaleiro**.
Cada nodo, é um **Group** e nele guardamos as transformações geométricas que a ela lhe dizem respeito assim como um vector de pontos do **model** que queremos desenhar, e por fim, um array de outros **Scene Graphs** que representam o próximo nível de descendentes.

main.cpp

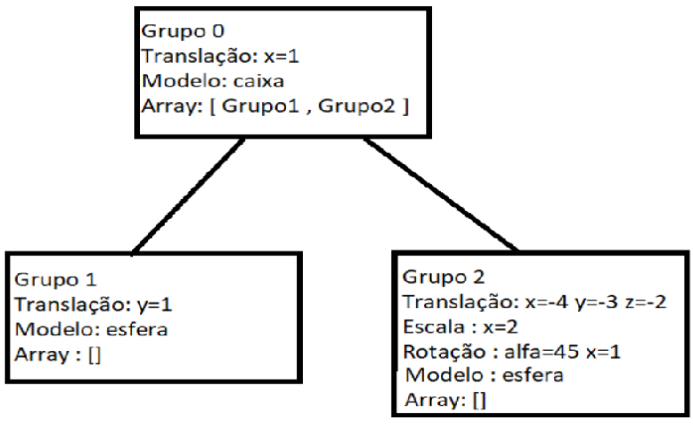
```
class SceneGraph {

    array<float, 3> scale;
    array<float, 3> trans;
    array<float, 4> rot;
    vector<vector<Ponto>> modelos;
    vector<SceneGraph> filhos;

}
```

Estamos então perante a definição de uma árvore de **Scene Graphs**. Para facilitar a visualização deste conceito dispomos o seguinte exemplo.

```
<scene>
    <group>
        <translate x="1" />
        <models>
            <model file="caixa.txt" />
        </models>
        <group>
            <translate y="1" />
            <models>
                <model file="esfera.txt" />
            </models>
        </group>
        <group>
            <translate x="-4" y="-3" z="-2" />
            <scale x="2" />
            <rotate ang="45" x="1" />
            <models>
                <model file="esfera.txt"/>
            </models>
        </group>
    </group>
</scene>
```



Info:

- Todos os nodos filhos herdam as transformações dos pais.
- Cada nodo pode ter um qualquer número de filhos, mas apenas um pai, i.e., não existe herança múltipla.
- Da travessia desde a raiz, até a uma folha, resultam todas as dependências que um certo objecto tem na **scene** em questão.

Fixada a estrutura e comportamento do classe. Basta, agora, expor o algoritmo de travessia que é efectuada quando a nossa modesta aplicação lê um ficheiro de configuração. O resto da **API** é disponibilizada em anexo.

main.cpp

```
void SceneGraph::draw() const {

        glPushMatrix();

        glScalef(scale[0], scale[1], scale[2]);
        glRotatef(rot[0], rot[1], rot[2], rot[3]);
        glTranslatef(trans[0], trans[1], trans[2]);

        glBegin(GL_TRIANGLES);

        for( vector<Pontos> const &pnts :
                this->modelos ) {
                for( Pontos const &p : pnts ) {
                        glVertex3f(p.a, p.b, p.c);
                }
        }

        glEnd();

        for( SceneGraph const &tmp :
                this->filhos ) {
                tmp.draw();
        }

        glPopMatrix();

}
```

Info: Com base nos conteúdos abordados nas aulas teóricas decidimos adotar a convenção de ordenar as transformações geométricas **glRotatef()**,**glTranslatef()** e **glSclasef()** pela sequência exposta no código acima.

3.2 Motor

De maneira semelhante à primeira fase, o motor continua a fazer leituras de um ficheiro xml (conf.xml). Por outro lado, este ficheiro apresenta uma complexidade superior ao anterior, contendo groups, models e ainda tags de translação (**<translate>**), rotação (**<rotate>**) e mudança de escala (**<scale>**).

conf.xml

```
<scene>
    <group>
        <translate x="1" />
        <models>
            <model file="caixa.txt" />
        </models>
        <group>
            <translate y="1" />
            <models>
                <model file="esfera.txt" />
            </models>
        </group>
        <group>
            <translate x="-4" y="-3" z="-2" />
            <scale x="2" />
            <rotate ang="45" x="1" />
            <models>
                <model file="esfera.txt"/>
            </models>
        </group>
    </group>
</scene>
```

Cada ficheiro que está referenciado nas tags **<model>** é um ficheiro de texto criado pelo nosso gerador e contém todos os pontos das figuras que queremos representar. Como o motor lê os ficheiros de pontos a partir dum ficheiro **XML** utilizamos um parser xml para **C++** chamado **tinyxml-2**.

main.cpp

```
...
tinyxml2::XMLDocument doc;
//new file
doc.LoadFile("./conf.xml");
tinyxml2::XMLNode *scene = doc.FirstChild();
if (scene == nullptr) perror("Erro de Leitura.\n");

s_gg = doGroup(scene->FirstChildElement("group"));
...
```

Com este snippet de código, criamos um objeto do tipo XMLDocument. De seguida, com a função **LoadFile**, abrimos o ficheiro de configuração **conf.xml**, Criamos um objeto do tipo **XMLNode** e associamos-lhe a primeira **tag** do ficheiro conf.xml que é a raiz da estrutura do nosso ficheiro. Através da função doGroup, começamos a manipular o seu conteúdo.

engine.cpp

```
SceneGraph doGroup(tinyxml2::XMLElement* group) {
    SceneGraph s_g;
    tinyxml2::XMLElement* novo =
        group->FirstChildElement();
    for(novo; novo != NULL;
        novo = novo->NextSiblingElement()) {
        //printf("%s\n", novo->Name());
        if(!strcmp(novo->Name(), "group")) {
            s_g.addFilho(doGroup(novo));
        } else if(!strcmp(novo->Name(), "models")) {
            s_g.setModelo(doModels(novo));
        } else if(!strcmp(novo->Name(),
                "translate")) {
            s_g.setTrans(doTranslate(novo));
        } else if(!strcmp(novo->Name(), "rotate")) {
            s_g.setRot(doRotate(novo));
        } else if(!strcmp(novo->Name(), "scale")) {
            s_g.setScale(doScale(novo));
        } else {
            perror("Formato␣XML␣Incorreto.\n");
        }
    }
return s_g;
}
```

No ciclo *for*, percorremos todos o ChildElements da group, que são as tags que guardam os nossos ficheiros das figuras. Dentro de cada tag **group** pode haver outra tag **group**, ativando a recursão da função doGroup. Mas, há outras tags que podem aparecer, tais como **models <translate>**, **<rotate>** e **<scale>**. Caso a tag lida seja **<translate>**, a função doGroup evoca a função doTranslate:

engine.cpp

```
array<float,3> doTranslate(
        tinyxml2::XMLElement* translate) {

    array<float, 3> trans;

    const char * x;
    const char * y;
    const char * z;
    x = translate->Attribute("x");
    y = translate->Attribute("y");
    z = translate->Attribute("z");

    x == nullptr ? trans[0] = 0 : trans[0] = atof(x);
    y == nullptr ? trans[1] = 0 : trans[1] = atof(y);
    z == nullptr ? trans[2] = 0 : trans[2] = atof(z);

    return trans;
}
```

Caso a tag lida seja **<rotate>**, a função doGroup evoca a função doRotate:

engine.cpp

```
array<float,4> doRotate(tinyxml2::XMLElement* rotate) {

    array<float,4> rot;

    const char * x;
    const char * y;
    const char * z;
    const char * ang;
    x = rotate->Attribute("x");
    y = rotate->Attribute("y");
    z = rotate->Attribute("z");
    ang = rotate->Attribute("angle");

    ang == nullptr ? rot[0] = 0 : rot[0] = atoi(ang);
    x == nullptr ? rot[1] = 0 : rot[1] = atof(x);
    y == nullptr ? rot[2] = 0 : rot[2] = atof(y);
    z == nullptr ? rot[3] = 0 : rot[3] = atof(z);

    return rot;
}
```

Caso a tag lida seja **<scale>**, a função doGroup evoca a função doScale:

engine.cpp

```
array<float,3> doScale(tinyxml2::XMLElement* scale) {

    array<float,3> sca;

    const char * x;
    const char * y;
    const char * z;

    x = scale->Attribute("x");
    y = scale->Attribute("y");
    z = scale->Attribute("z");

    x == nullptr ? sca[0] = 1 : sca[0] = atof(x);
    y == nullptr ? sca[1] = 1 : sca[1] = atof(y);
    z == nullptr ? sca[2] = 1 : sca[2] = atof(z);

    return sca;
}
```

Qualquer uma destas 3 funções, retorna um array com os argumentos que serão passados às funções Glut que executarão as transformações: a função doTranslate, retorna um array com 3 argumentos para a função **glTranslate**, a função doRotate, retorna um array com 4 argumentos para a função **glRotate** e a função doScale, retorna um array com 3 argumentos para a função **glScale**.

Por outro lado, ainda pode aparecer uma tag **<models>**, ou seja é chamada a função doModels que significa que, dentro dessas tags, vai haver uma tag **<model file=** "**nome do ficheiro.txt**», ficheiro este que tem todos os pontos gerados pelo gerador.

engine.cpp

```
std::vector<std::vector<Pontos>>
        doModels(tinyxml2::XMLElement* models) {
    std::vector<std::vector<Pontos>> pPontos;
    tinyxml2::XMLElement* novo =
        models->FirstChildElement();
    for(novo; novo != NULL;
        novo = novo->NextSiblingElement()) {
        const char * file;
        file = novo->Attribute("file");
        pPontos.push_back(guardaPontos(file));
    }
    return pPontos;
}
```

A função doModels chama a função **guardaPontos** que, usando a estrutura **Pontos**, regista as coordenadas de cada ponto presente no ficheiro. A estrutura tem campos 3 *floats*, que servem para guardar as coordenadas dos pontos. De seguida usamos um vector que utiliza a estrutura enunciada para os guardar. A função abre os ficheiros e lê pontos linha a linha, guardando cada coordenada *x, y e z* num **Ponto** e de seguida inserindo-o no vector.

engine.cpp

```
...
struct Pontos {
    float a;
    float b;
    float c;
};

std::vector<Pontos> pontos;

void guardaPontos(std::string ficheiro) {
        std::ifstream file;
        std::string s = "./";
        s.append(ficheiro.c_str());
        file.open(s.c_str());
        float a,b,c;
        while(file >> a >> b >> c) {
                Pontos aux;
                aux.a = a;
                aux.b = b;
                aux.c = c;
                pontos.push_back(aux);
        }
}
...
```

Este array de pontos que a guardaPontos retorna, preenche o array pPontos instanciado na doModels, que por sua vez é retornado por esta função. Na função doGroup, uma SceneGraph é passada como objeto aos Setters definidos na estrutura SceneGraph, que chamam as funções acima faladas que, retornandos os arrays de argumentos e os pontos das figuras, fazem uma atualização da estrutura, que na **main.cpp** é passada à renderScene.

4 Terceira Fase

Os requisitos para esta fase são:

- O gerador tem agora de ser capaz de criar um novo modelo baseado em **patches** de **Bezier**;
- O motor terá que enriquecer a sua definicão de duas transformações geométricas. As *translações* poderam agora receber um conjunto de pontos (com um mínimo de 4 elementos), sendo estes os pontos de controlo de uma curva de **Catmull-Rom**, e um tempo (em segundos) para percorrer a curva. O objetivo é realizar animações baseadas nestas curvas. As *rotações* podem agora receber um tempo (novamente em segundos) em vez de um ângulo, este server para descrever o tempo para se realizar uma rotação de 360º. Estas também com o objetico de animar objetos no sistema;
- Por fim, os modelos têm agora de ser desenhados usando **VBO**'s.

Os requisitos levaram a atualização de alguns conceitos e a criação de novos. Descrevemos todas estas mudanças nos capítulos que se seguem.

4.1 Link para um streamable com a demonstrar animação

LINK - `https://streamable.com/k2906`

4.2 Modificações à classe SceneGraph

A classe **SceneGraph**, introduzida na **Etapa 2** sofreu alguma restruturação. Os *arrays* que eram usados para guardar as várias *tranformações geométricas* passaram agora a ser classes com o seu próprio comportamento. Foram criados dois novo objetos **TranslacaoC** e **RotacaoT** para lidar com os novos requisitos referentes às translações e rotações. O objeto **SceneGraph** passou a ter uma variável de instância *vbo*, esta, como o nome indica, é usada para desenhar os modelos usando **VBO**'s. As mundanças são apresentadas em maior detalhe nos capítulos seguintes.

4.2.1 Redefinição de Conceitos

Arrays

Como mencionado acima os **arrays** foram convertidos para classes. Esta mudança veio a propósito de tornar mais fácil o trabalho a desenvolver com esta, face ao aumento em número das suas variáveis de instância e com isto, a sua complexidade. Estas apresentam a mesma funcionalidade da etapa anterior.
O *array* que codificava uma rotação simples passou a ser definido por:

conversao.cpp

```
array<float, 4> rot

                //para

// RotacaoV = Rotacao Vetorial
class RotacaoV {

        array<float, 4> rot;

        public:
                RotacaoV();
                void setRot( array<float, 4> );
                void aplica();
};
```

Os métodos a que esta responde (e de forma similar todas as outras transformações estáticas) são um *setter* para mudar o seu estado interno e a função *aplica* que será usada na altura de desenhar para aplicar a transformação aos modelos. As restantes classes mais simples apresentam uma conversão similar, pelo que escusámos de as apresentar todas aqui.

Modelos

Os modelos guardados passaram agora a ser uma *vector* de *float*'s em vez de um da estrutra *Pontos* para facilitar a sua impressão via **VBO**'s.

conversao.cpp

```
...
vector<Pontos> modelos;
...
                  //para
...
vector<float> modelos;
...
```

A existência de métodos como *data*, *size* e ainda *insert* facilitam a transição para **VBO**'s.

Método draw

O método encarregue de desenhar os modelos sofreu como de esperar algumas mudanças.

drawantigo.cpp

```
void SceneGraph::draw() const {

    glPushMatrix();

    glScalef(scale[0], scale[1], scale[2]);
    glRotatef(rot[0], rot[1], rot[2], rot[3]);
    glTranslatef(trans[0], trans[1], trans[2]);

    glBegin(GL_TRIANGLES);

    for( vector<Pontos> const &pnts :
        this->modelos ) {
        for( Pontos const &p : pnts ) {
                glVertex3f(p.a, p.b, p.c);
            }
    }
    glEnd();
    for( SceneGraph const &tmp : this->filhos ) {
        tmp.draw();
    }
    glPopMatrix();
}
```

Este passa agora a chamar os métodos *aplica* em vez de explicitamente fazer as transformações, sendo posteriormente feito o *bind* a partir da variável *vbo*. Os argumentos da função *VertexPointer* e *glDrawArrays*. Também importante notar o *booleano* usado para sinalizar quando as transformações dinâmicas têm de ser aplicadas(n vezes por segundo).

drawnovo.cpp

```
void SceneGraph::draw( bool updt ) {

    glPushMatrix();

    this->scale.aplica();
    this->rot.aplica();
    this->trans.aplica();

    this->curva.aplica( updt );
    this->eixo.aplica( updt );

    glBindBuffer(GL_ARRAY_BUFFER, this->vbo);
    glVertexPointer(3,GL_FLOAT, 0, 0);

    glDrawArrays(GL_TRIANGLES, 0,
        this->modelos.size() / 3);

    for( SceneGraph &tmp : this->filhos ) {
        tmp.draw( updt );
    }

    glPopMatrix();

}
```

Info: De notar a ordem com que as transformações são aplicadas. As estáticas são priorizadas (mantendo a ordem da etapa anterior), sendo das dinâmicas aplicada primeiro a translação e por fim a rotação.

4.2.2 Novas Funcionalidades

Para satisfazer os requisitos, foram criadas as classes:

- **TranslacaoC** - para codificar a animação a partir de uma curva de **CatmullRom**;
- **RotacaoT** - para codificar a animação de um planeta em torno de um eixo dado um fator temporal.

Da classe **TranslacaoC**, apresentamos o seu método principal:

transC.cpp

```
void TranslacaoC::setCurva( vector<Pontos> pntCnt,
        int tempoVolta ) {

    Pontos aux;
    float tseg = 1.0f / tempoVolta;

    if( !pntCnt.empty() ) {
        for(int i = 0; i < tempoVolta; i++) {
            getGlobalCatmullRomPoint(i * tseg,
            pntCnt.data(), pntCnt.size(), &aux);
            this->pntsCurva.push_back(aux);
        }
    }
}
```

Que aceita um *vector* de pontos de controlo da curva e o tempo para dar uma volta. Esta calcula os pontos por onde o objeto passa a cada segundo da volta, guardando-os.

De seguida, da classe **RotacaoT** apresentamos os métodos que gerem o estado interno.

RotacaoT.cpp

```
void RotacaoT::setRot( array<float, 3> axs ) {
    this->rot[1] = axs[0];
    this->rot[2] = axs[1];
    this->rot[3] = axs[2];
}

void RotacaoT::setGraus( int tempoVolta ) {
    this->rot[0] = 360.0f / tempoVolta;
    this->segundos = tempoVolta;
}
```

4.3 A classe TimedSG

Esta classe utiliza os objetos **SceneGraph** e **Cronometro** para conseguir trazer animações ao motor.

Cronometro.cpp

```
bool Cronometro::updateTime() {

    int aux = glutGet(GLUT_ELAPSED_TIME);

    if(aux - this->basetime >= 25) {
        this->basetime = aux;
        return true;
    }

    return false;

}
```

A classe **Cronometro** apresenta apenas um método, este compara um tempo previamente registado ao tempo atual e se este for maior que um delta atualiza o seu valor interno e devolve *true*, caso contrário devolve *false*. Este boleano é usado na classe seguinte para determinar se as trasformações dinâmicas têm de movimentar os seus objetos.

timedsg.cpp

```
TimedSG::TimedSG() {

}

void TimedSG::setSG( SceneGraph novoSG ) {

    this->sg = novoSG;

}

void TimedSG::prep() {

    this->sg.prep();

}

void TimedSG::draw() {

    bool aux = this->tmp.updateTime();
    this->sg.draw( aux );
}
```

4.4 Atualizações ao Motor

A nível do parse do ficheiro .xml, é criado um **XMLNode** com a primeira tag **gorup**, e passando a mesma à função **doGroup** que trata todas as tags desse groupo. O que difere da fase anterior é que as informações são passadas por objetos, sendo que vai haver uma função associada a cada tag lida, que mantém a funcionalidade registada na fase anterior. À semelhança da fase anterior, caso a tag lida seja group, é chamada a doGroup de novo de maneira recursiva.
Porém, nesta nova fase foi introduzida a noção de curvas e tempo. O que significa que tivemos que fazer mudanças na função doGroup para puder receber estas tags com **Atribute time**.

engine.cpp

```
SceneGraph doGroup(tinyxml2::XMLElement* group) {
    SceneGraph s_g;
    tinyxml2::XMLElement* novo = group->FirstChildElement();
    for(novo; novo != NULL; novo = novo->NextSiblingElement()) {
        //printf("%s\n", novo->Name());
        if(!strcmp(novo->Name(), "group")) {
            s_g.addFilho(doGroup(novo));
        } else if(!strcmp(novo->Name(), "models")) {
            s_g.setModelo(doModels(novo));
        } else if(!strcmp(novo->Name(), "translate")) {
            if(novo->Attribute("time") == nullptr) {
                s_g.setTrans(doTranslate(novo));
            } else {
                s_g.setCurva(doTimeTranslate(novo));
            }
        } else if(!strcmp(novo->Name(), "rotate")) {
            if(novo->Attribute("time") == nullptr) {
                s_g.setRot(doRotate(novo));
            } else {
                s_g.setEixo(doTimeRotate(novo));
            }
        } else if(!strcmp(novo->Name(), "scale")) {
            s_g.setScale(doScale(novo));
        } else {
            perror("Formato␣XML␣Incorreto.\n");
        }
    }
    return s_g;
}
```

As novas tags podem ser **translate time** e **rotate time** e para isso foram criadas duas novas funções: **doTimeTranslate** e **doTimeRotate** para tratar cada uma das tags, respetivamente.

engine.cpp

```
        TranslacaoC doTimeTranslate(tinyxml2::XMLElement* translate) {

            TranslacaoC t;

            int tempo;
            tempo = atoi(translate->Attribute("time"));

            std::vector<Pontos> pontos;

            tinyxml2::XMLElement* point = translate->FirstChildElement();
            for(point; point != NULL; point = point->NextSiblingElement()
                Pontos aux;
                aux.a = atof(point->Attribute("x"));
                aux.b = atof(point->Attribute("y"));
                aux.c = atof(point->Attribute("z"));
                pontos.push_back(aux);
            }
            if(pontos.size() < 4) {
                perror("Sao necessarios no minimo 4 pontos");
            } else {
                t.setCurva(pontos, tempo);
            }
            return t;
        }
```

engine.cpp

```
        RotacaoT doTimeRotate(tinyxml2::XMLElement* rotate) {

            RotacaoT rotation;

            std::array<float, 3> xyz;

            int tempo;

            xyz[0] = atof(rotate->Attribute("x"));
            xyz[1] = atof(rotate->Attribute("y"));
            xyz[2] = atof(rotate->Attribute("z"));

            tempo = atoi(rotate->Attribute("time"));

            rotation.setRot(xyz);
            rotation.setGraus(tempo);

            return rotation;
        }
```

4.5 Atualizações ao Gerador

Não contemplamos mudanças ao gerador nesta fase. (Bezier patches não implementadas).

5 Quarta fase

5.1 Gerador

O **Gerador**, nesta fase, tem de também de imprimir no ficheiro alvo, que á semelhança das fases anteriores é escolhido pelo utilizador, para cada vertice, a normal e as coordenadas no espaço textura.
Por uma questão de organização adoptamos a convenção na primiera linha do ficheiro consta sempre o numero vertices, seguidos das componentes das coordenadas dos vertices, depois das normais e por fim das texturas (uma por linha).

5.1.1 Calculo Normais

As normais dos pontos de um objecto servem para que o **OpenGl** possa calcular o comportamento da luz ao longo das superficies da primitiva. Todos estes valores são armazenados na sua forma normalizada para garantir eficiencia e comportamentos menos desejados.

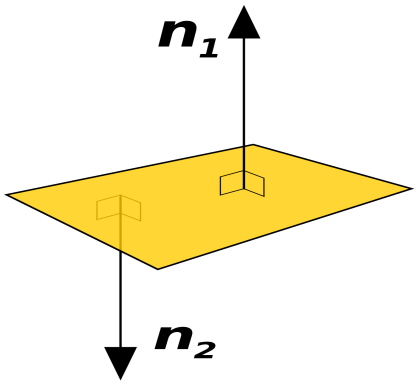
Info: Frequentemente acontece de haver um ponto que pertence a mais que uma fase:

- O topo do da primitiva Cone
- O vertices da Caixa
- O todos os vertices da esfera

Ora as normais sao calculadas por fase por isso frequentemente existem mais que uma normal assciada a pontos nestas condições

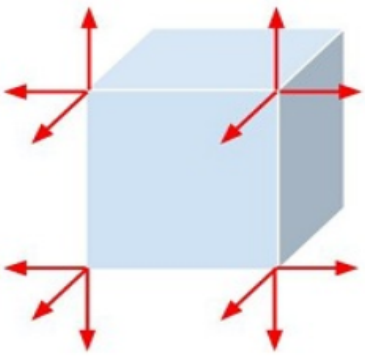
5.1.2 Plano

No caso de um plano XZ é trivialmente verdade que a normal é a mesma em todos os pontos do plano, em particular nas extremidades, e que é igual a (0,1,0). No entanto, para que o comportamento da luz seja equivalente na frente eno verso desta primitiva, associamos a normal (0,-1,0) aos pontos do verso do plano.



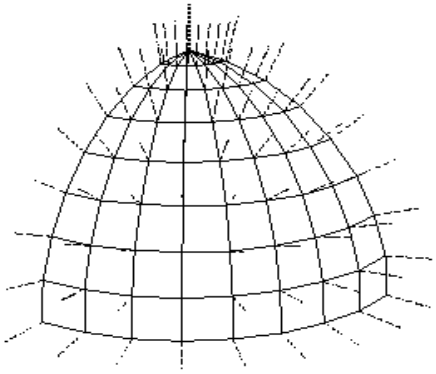
5.1.3 Caixa

Se subdividirmos esta primitiva nas fases que a constituem caimos no caso da primitiva anterior. De emidiato vem que para as fases paralelas a XY as normais são (0,0,1) e (0,0,-1), a XZ (0,1,0) e (0,-1,0) e por fim as YZ (1,0,0) e (-1,0,0).



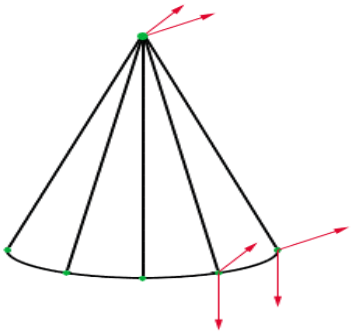
5.1.4 Esfera

No caso desta primitiva, como já trabalhamos com coordenadas polares para gerar os vertices, as normais de cada ponto podem ser facilmente obtidas se considerarmos uma esfera de raio 1.



5.1.5 Cone

A base do cone esta em XZ, é novamente o caso do plano, portanto, a sua normal é (0,-1,0). Já para os pontos pertencentes ás fases laterais, a sua normal é dada pela expressão (r * sin(alpha), stk ou nstk , r * cos(alpha)), onde **stk** e **nstk** são as variaveis que armazenam a altura da atual ou proxima **stack** a que o vertice pertence.

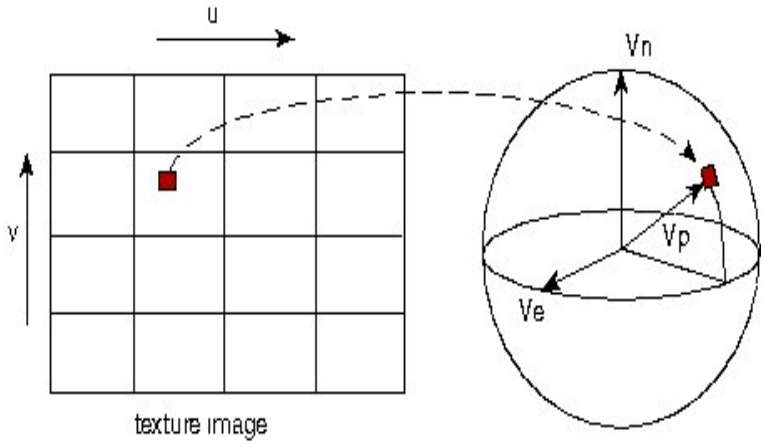


5.1.6 Mapeamento de Texturas

As coordenadas de textura servem para que se possa atribuir uma textura a uma primitiva. Cada ponto terá um par de coordenadas x e y, que variam entre 0 e 1. Estas coordenadas são utilizadas pelo **OpenGL** para saber que ponto na imagem a ser utilizada como textura corresponde ao ponto da primitiva em questão.

6 Esfera

Sejam alpha e beta os angulos que variam, respetivamente [0..2*PI] e [-PI/2 ... PI /2], facilmente se mapeam para espaço textura (u e v ambos a variar entre 0 e 1) com a tranformação u = a / (2 * PI) e v = (b / PI) + 0.5 .



7 Box

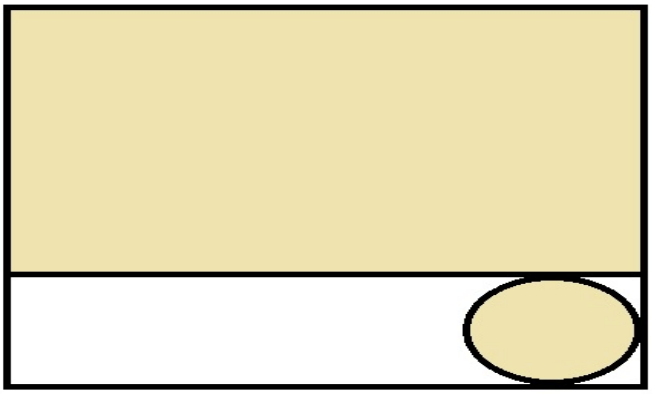
Sejam i e j as variaveis que iteram as subdivisoes da caixa (ndivs) nos eixos XX e ZZ respetivamente, mapeamos u = i / ndivs e v = j / ndivs.

7.0.1 Plano

Neste caso basta mapear as extremidades do plano para a extremidade correspondente da imagem, isto é , canto superior direito do plano para (1,1), canto superior esquerdo do plano para (-1,1), canto inferior direito do plano para (1,-1) e o canto inferior esquerdo do plano para (-1,-1) .

7.0.2 Cone

Para mapear esta primitiva para o seu correspondente espaço textura, desenvolvemos o atlas representado pela figura abaixo.



8 Motor

Nesta face o sistema de classes desenvolvido anteriormente para o **motor** foi expandido para suportar **texture mapping** e diferentes modos de **lightning**.

8.1 Lightning

Atraves das tags e atributos relativamente as luzes consegue-se, facilmente, representar um dos tres tipos de luz em OpenGL a partir dos nossos ficheiros de configuração **XML**. As diferentes luzes afetam os objetos pertencentes a um grupo e todos os objetos pertencentes a grupos de hierarquias inferiores. Internamente o **motor** guarda as diferentes configurações e tipos de luz na classe **Luz**, que tem três outras descendentes a **pontual**, **direcional** e de **foco?**.
Todas elas partilham um posição no espaço, e por esta razão, posemos em evidencia este campo para a classe Mãe, de modo a que seja herdado para as suas filhas.
Assim podemos ter coleções com diferentes tipos luzes a obedecerem todas ao mesmo metodo **aplica**. Este comportamento polimorfico, apresenta um grande ganho na simplificação do codigo.

8.1.1 Pontual

Os objetos deste tipo possuem um comportamento semelhante ao de uma estrela. No sentido em que a partir deste ponto são propagados feixes de luz em todas as direções.

8.1.2 Direcional

8.1.3 Spot ?

8.2 Texuring

9 Conclusão

Também nos ajudou a possuir mais discernimento sobre a geometria e cálculos matemáticos por detrás de todo um esquema geométrico em 3 dimensões.
Durante a realização dos geradores conseguimos perceber que a propagação dos erros nos cálculos dos ângulos pode ter impacto na apresentação das figuras geométricas.
Apresentou-nos também mais uma oportunidade de aprender e melhorar as nossas capacidades de programação em C++, no uso de LaTex e de ficheiros XML.
Todas as aptidões aqui aprendidas e/ou desenvolvidas, não só a nível escolar mas como a nível de cooperação e de trabalho de equipa. vão-nos permitir uma melhor realização de projetos futuros.
A realização desta fase, ajudou-nos a ganhar um conhecimento mais profundo no âmbito das curvas e superfícies de Bezier e do cálculo por detrás do algoritmo de Catmull-Rom, conhecimento sobre rotações e translações face ao tempo. À medida que vamos avançando no projeto, a complexidade do mesmo vai

aumentando, o que também nos trás a oportunidade de melhorar as nossas capacidades de programação em C++, e do uso das funções das bibliotecas leccionadas nas aulas.
Está secção poderá ser modificada ao longo das fases de entrega.

A Biblioteca CatmullRomMath

```
#ifndef CATMULLROMMATH__H
#define CATMULLROMMATH__H

#define _USE_MATH_DEFINES
#include "ponto.h"
#include <math.h>

#define tvector(t) {powf(t, 3.0f), powf(t, 2.0f), t, 1.0f}

#define mult1441(a, b) a[0] * b[0] + a[1] * b[1] + a[2] * b[2] + a[3] * b[3]

void multMatrixVector(float *m, float *v, float *res);


void getCatmullRomPoint(float t, Pontos p0, Pontos p1, Pontos p2, Pontos p3, Pontos


void getGlobalCatmullRomPoint(float gt, Pontos *pontos, int nmrpontos, Pontos *pos);

#endif

#include "catmullmath.h"


void multMatrixVector(float *m, float *v, float *res) {

        for (int j = 0; j < 4; ++j) {
                res[j] = 0;
                for (int k = 0; k < 4; ++k) {
                        res[j] += v[k] * m[j * 4 + k];
                }
        }

}

void getCatmullRomPoint(float t, Pontos p0, Pontos p1, Pontos p2, Pontos p3, Pontos


        // matriz 4x4 de Catmull em formato linha
        float mCat[16] = { -0.5f,  1.5f, -1.5f,  0.5f,
                            1.0f, -2.5f,  2.0f, -0.5f,
                           -0.5f,  0.0f,  0.5f,  0.0f,
                            0.0f,  1.0f,  0.0f,  0.0f};

        float pts[3][4] = { { p0.a, p1.a, p2.a, p3.a },
                            { p0.b, p1.b, p2.b, p3.b },
                            { p0.c, p1.c, p2.c, p3.c } };

        float ts[4] = tvector(t);
        float aux[4];
        float posaux[3];

        for(int i = 0; i < 3; i++) {
                multMatrixVector(mCat, pts[i], aux);
                posaux[i] = mult1441(ts, aux);
```

```
        }

        pos->a = posaux[0];
        pos->b = posaux[1];
        pos->c = posaux[2];

}

void getGlobalCatmullRomPoint(float gt, Pontos *pontos, int nmrpontos, Pontos *pos)

        float t = gt * nmrpontos;
        int index = floor(t);
        int indices[4];

        t = t - index;
        indices[0] = (index + nmrpontos - 1) % nmrpontos;
        indices[1] = (indices[0] + 1) % nmrpontos;
        indices[2] = (indices[1] + 1) % nmrpontos;
        indices[3] = (indices[2] + 1) % nmrpontos;

        getCatmullRomPoint(t, pontos[indices[0]], pontos[indices[1]], pontos[indices

}
```

B Biblioteca Cronometro

```
#ifndef CRONOMETRO__H
#define CRONOMETRO__H

#include <GL/glew.h>
#include <GL/glut.h>

class Cronometro {

        int basetime;

        public:
                Cronometro();
                bool updateTime();
};

#endif

#include "cronometro.h"

Cronometro::Cronometro() {
        this->basetime = 0;
}

bool Cronometro::updateTime() {

        int aux = glutGet(GLUT_ELAPSED_TIME);

        if(aux - this->basetime >= 25) {
                this->basetime = aux;
                return true;
        }

        return false;

}
```

C Biblioteca Engine

```
std::vector<Pontos> guardaPontos(std::string ficheiro);

std::vector<std::vector<Pontos>> doModels(tinyxml2::XMLElement* models);

array<float,3> doTranslate(tinyxml2::XMLElement* translate);

array<float,4> doRotate(tinyxml2::XMLElement* rotate);

array<float,3> doScale(tinyxml2::XMLElement* scale);

SceneGraph doGroup(tinyxml2::XMLElement* group);

#include "tinyxml2.h"
#include <stdio.h>
#include <string.h>
#include "./timedsg.h"
#include <fstream>
#include <iostream>



std::vector<float> guardaPontos(std::string ficheiro) {

    std::vector<float> pontos;

    std::ifstream file;
    //change this to your folder's path.
    std::string s = "./";
    s.append(ficheiro.c_str());
    file.open(s.c_str());
    float a,b,c;
    while(file >> a >> b >> c) {
        pontos.push_back(a);
        pontos.push_back(b);
        pontos.push_back(c);
    }
    return pontos;
}

std::vector<float> doModels(tinyxml2::XMLElement* models) {
    std::vector<float> pPontos;
    std::vector<float> savedPoints;

    tinyxml2::XMLElement* novo = models->FirstChildElement();
    for(novo; novo != NULL; novo = novo->NextSiblingElement()) {
        const char * file;
        file = novo->Attribute("file");
        savedPoints = guardaPontos(file);
        pPontos.insert(pPontos.begin(), savedPoints.begin(), savedPoints.end());
    }
    return pPontos;
}

TranslacaoV doTranslate(tinyxml2::XMLElement* translate) {
```

```
    TranslacaoV transl;

    array<float, 3> trans;

    const char * x;
    const char * y;
    const char * z;
    x = translate->Attribute("x");
    y = translate->Attribute("y");
    z = translate->Attribute("z");

    x == nullptr ? trans[0] = 0 : trans[0] = atoi(x);
    y == nullptr ? trans[1] = 0 : trans[1] = atoi(y);
    z == nullptr ? trans[2] = 0 : trans[2] = atoi(z);

    transl.setTrans(trans);

    return transl;
}

RotacaoV doRotate(tinyxml2::XMLElement* rotate) {

    RotacaoV rotation;

    array<float,4> rot;

    const char * x;
    const char * y;
    const char * z;
    const char * ang;
    x = rotate->Attribute("x");
    y = rotate->Attribute("y");
    z = rotate->Attribute("z");
    ang = rotate->Attribute("angle");

    ang == nullptr ? rot[0] = 0.0f : rot[0] = atoi(ang);
    x == nullptr ? rot[1] = 0.0f : rot[1] = atof(x);
    y == nullptr ? rot[2] = 0.0f : rot[2] = atof(y);
    z == nullptr ? rot[3] = 0.0f : rot[3] = atof(z);

    rotation.setRot(rot);

    return rotation;
}

Escala doScale(tinyxml2::XMLElement* scale) {

    Escala escala;

    array<float,3> sca;

    const char * x;
    const char * y;
    const char * z;

    x = scale->Attribute("x");
    y = scale->Attribute("y");
```

```
    z = scale->Attribute("z");

    x == nullptr ? sca[0] = 1.0f : sca[0] = atof(x);
    y == nullptr ? sca[1] = 1.0f : sca[1] = atof(y);
    z == nullptr ? sca[2] = 1.0f : sca[2] = atof(z);

    escala.setAxis(sca);

    return escala;
}

TranslacaoC doTimeTranslate(tinyxml2::XMLElement* translate) {

    TranslacaoC t;

    int tempo;
    tempo = atoi(translate->Attribute("time"));

    std::vector<Pontos> pontos;

    tinyxml2::XMLElement* point = translate->FirstChildElement();
    for(point; point != NULL; point = point->NextSiblingElement()) {
        Pontos aux;
        aux.a = atof(point->Attribute("x"));
        aux.b = atof(point->Attribute("y"));
        aux.c = atof(point->Attribute("z"));
        pontos.push_back(aux);
    }
    if(pontos.size() < 4) {
        perror("Sao␣necessarios␣no␣minimo␣4␣pontos");
    } else {
        t.setCurva(pontos, tempo);
    }
    return t;
}

RotacaoT doTimeRotate(tinyxml2::XMLElement* rotate) {

    RotacaoT rotation;

    std::array<float, 3> xyz;

    int tempo;

    xyz[0] = atof(rotate->Attribute("x"));
    xyz[1] = atof(rotate->Attribute("y"));
    xyz[2] = atof(rotate->Attribute("z"));

    tempo = atoi(rotate->Attribute("time"));

    rotation.setRot(xyz);
    rotation.setGraus(tempo);

    return rotation;
}

SceneGraph doGroup(tinyxml2::XMLElement* group) {
```

```
    SceneGraph s_g;
    tinyxml2::XMLElement* novo = group->FirstChildElement();
    for(novo; novo != NULL; novo = novo->NextSiblingElement()) {
        //printf("%s\n", novo->Name());
        if(!strcmp(novo->Name(), "group")) {
            s_g.addFilho(doGroup(novo));
        } else if(!strcmp(novo->Name(), "models")) {
            s_g.setModelo(doModels(novo));
        } else if(!strcmp(novo->Name(), "translate")) {
            if(novo->Attribute("time") == nullptr) {
                s_g.setTrans(doTranslate(novo));
            } else {
                s_g.setCurva(doTimeTranslate(novo));
            }
        } else if(!strcmp(novo->Name(), "rotate")) {
            if(novo->Attribute("time") == nullptr) {
                s_g.setRot(doRotate(novo));
            } else {
                s_g.setEixo(doTimeRotate(novo));
            }
        } else if(!strcmp(novo->Name(), "scale")) {
            s_g.setScale(doScale(novo));
        } else {
            perror("Formato␣XML␣Incorreto.\n");
        }
    }
    return s_g;
}
```

D Biblioteca Escala

```
#ifndef ESCALA__H
#define ESCALA__H

#include <array>
#include <GL/glew.h>
#include <GL/glut.h>

using namespace std;

class Escala {

        array<float, 3> escala;

        public:
                Escala();
                void setAxis( array<float, 3> );
                void aplica();
};

#endif

#include "escala.h"

using namespace std;

Escala::Escala() {
        this->escala.fill(1.0f);
}

void Escala::setAxis( array<float, 3> newAxs ) {
        this->escala = newAxs;
}

void Escala::aplica() {
        glScalef(this->escala[0], this->escala[1], this->escala[2]);
}
```

E Biblioteca RotacaoT

```
#ifndef ROTACAOT__H
#define ROTACAOT__H

#include <array>
#include <GL/glew.h>
#include <GL/glut.h>

using namespace std;

class RotacaoT {

        array<float, 4> rot;
        int segundos;
        int voltas;

        public:
                RotacaoT();
                void setRot( array<float, 3> );
                void setGraus( int );
                void aplica( bool );

};

#endif

#include "rotacaoT.h"

RotacaoT::RotacaoT() {
        this->rot.fill(0.0f);
        this->segundos = 1;
        this->voltas = 0;
}

void RotacaoT::setRot( array<float, 3> axs ) {
        this->rot[1] = axs[0];
        this->rot[2] = axs[1];
        this->rot[3] = axs[2];
}

void RotacaoT::setGraus( int tempoVolta ) {
        this->rot[0] = 360.0f / tempoVolta;
        this->segundos = tempoVolta;
}

void RotacaoT::aplica( bool updt ) {

        if( updt ) {
                this->voltas = (this->voltas + 1) % this->segundos;
        }

        glRotatef(this->voltas * rot[0], rot[1], rot[2], rot[3]);
}
```

F Biblioteca RotacaoV

```
#include <array>
#include <GL/glew.h>
#include <GL/glut.h>

#ifndef ROTACAOV__H
#define ROTACAOV__H

using namespace std;

class RotacaoV {

        array<float, 4> rot;

        public:
                RotacaoV();
                void setRot( array<float, 4> );
                void aplica();
};

#endif

#include "rotacaoV.h"

using namespace std;

RotacaoV::RotacaoV() {
        this->rot.fill(0.0f);
}

void RotacaoV::setRot( array<float, 4> newrot ) {
        this->rot = newrot;
}

void RotacaoV::aplica() {
        glRotatef(this->rot[0], this->rot[1], this->rot[2], this->rot[3]);
}
```

G Biblioteca SceneGraph

```
#ifndef SG__H
#define SG__H

#include <vector>
#include <string>
#include <array>
#include <GL/glew.h>
#include <GL/glut.h>
#include "ponto.h"
#include "escala.h"
#include "rotacaoT.h"
#include "rotacaoV.h"
#include "translacaoC.h"
#include "translacaoV.h"
#include "cronometro.h"

using namespace std;

class SceneGraph {

        // variaveis
        // transformacoes nao baseadas em tempo
        Escala scale;
        TranslacaoV trans;
        RotacaoV rot;

        //transformacoes baseadas em tempo
        TranslacaoC curva;
        RotacaoT eixo;

        // modelos guardados na scenegraph
        vector<float> modelos;

        // inteiro responsavel pelos VBOS
        GLuint vbo;

        // Descendencia do SceneGraph
        vector<SceneGraph> filhos;

        public:
                // Construtores
                SceneGraph();

                // Setters
                void setScale( Escala );
                void setTrans( TranslacaoV );
                void setRot( RotacaoV );
                void setModelo( vector<float> );
                void setCurva( TranslacaoC );
                void setEixo( RotacaoT );

                // Funcoes adicionais
                void addFilho( SceneGraph );

                // Funcao que trata de inicializar os VBOS
```

```
                void prep();

                // Funcao responsavel por desenhar a estrutura
                void draw( bool );
};

#endif

#include "sg.h"

using namespace std;

// SceneGraph
// Construtor

SceneGraph::SceneGraph() {

}

// Setters

void SceneGraph::setScale( Escala novaesc ) {

        this->scale = novaesc;

}

void SceneGraph::setTrans( TranslacaoV novatrans ) {

        this->trans = novatrans;

}

void SceneGraph::setRot( RotacaoV novorot ) {

        this->rot = novorot;

}

void SceneGraph::setModelo( vector<float> pontos ) {

        this->modelos = pontos;

}

void SceneGraph::setCurva( TranslacaoC novacurva ) {

        this->curva = novacurva;


}

void SceneGraph::setEixo( RotacaoT novoeixo ) {

        this->eixo = novoeixo;
}
```

```
// Funcoes Adicionais

void SceneGraph::addFilho( SceneGraph c ) {

        this->filhos.push_back( c );

}

void SceneGraph::prep() {

        glGenBuffers(1, &(this->vbo));
        glBindBuffer(GL_ARRAY_BUFFER, this->vbo);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * this->modelos.size(), this->mo

        for( SceneGraph &tmp : this->filhos ) {
                tmp.prep();
        }
}

// Funcao responsavel por desenhar a estrutra
void SceneGraph::draw( bool updt ) {

        glPushMatrix();

        this->scale.aplica();
        this->rot.aplica();
        this->trans.aplica();

        this->curva.aplica( updt );
        this->eixo.aplica( updt );

        glBindBuffer(GL_ARRAY_BUFFER, this->vbo);
        glVertexPointer(3,GL_FLOAT, 0, 0);

        glDrawArrays(GL_TRIANGLES, 0, this->modelos.size() / 3);

        for( SceneGraph &tmp : this->filhos ) {
                tmp.draw( updt );
        }

        glPopMatrix();

}
```

H Biblioteca TimedSG

```
#ifndef TIMEDSG__H
#define TIMEDSG__H

#include "sg.h"
#include "cronometro.h"

class TimedSG {

        SceneGraph sg;
        Cronometro tmp;

        public:
                TimedSG();
                void setSG( SceneGraph );
                void prep();
                void draw();
};

#endif

#include "timedsg.h"


TimedSG::TimedSG() {

}

void TimedSG::setSG( SceneGraph novoSG ) {

        this->sg = novoSG;

}

void TimedSG::prep() {

        this->sg.prep();

}

void TimedSG::draw() {

        bool aux = this->tmp.updateTime();
        this->sg.draw( aux );
}
```

I Biblioteca TranslacaoC

```
#ifndef TRANSLACAOC__H
#define TRANSLACAOC__H

#include <vector>
#include "ponto.h"
#include <GL/glew.h>
#include <GL/glut.h>
#include "catmullmath.h"

using namespace std;

class TranslacaoC {

        vector<Pontos> pntsCurva;
        int voltas;

        public:
                TranslacaoC();
                void setCurva( vector<Pontos>, int);
                void aplica( bool );

};

#endif

#include "translacaoC.h"

using namespace std;

TranslacaoC::TranslacaoC() {
        this->voltas = 0;
}

void TranslacaoC::setCurva( vector<Pontos> pntCnt, int tempoVolta ) {


        Pontos aux;
        float tseg = 1.0f / tempoVolta;

        if( !pntCnt.empty() ) {
                for(int i = 0; i < tempoVolta; i++) {
                        getGlobalCatmullRomPoint(i * tseg, pntCnt.data(), pntCnt.siz
                        this->pntsCurva.push_back(aux);
                }
        }
}

void TranslacaoC::aplica( bool updt ) {

        Pontos aux;
        if( !this->pntsCurva.empty() ) {
                if( updt ) {
                        this->voltas = ( this->voltas + 1 ) % this->pntsCurva.size()
                }
                aux = this->pntsCurva[this->voltas];
```

```
                glTranslatef(aux.a, aux.b, aux.c);
        }
}
```

J Biblioteca TranslacaoV

```
#include <array>
#include <GL/glew.h>
#include <GL/glut.h>

#ifndef TRANSLACAOV__H
#define TRANSLACAOV__H

using namespace std;

class TranslacaoV {

        array<float, 3> trans;

        public:
                TranslacaoV();
                void setTrans( array<float, 3> );
                void aplica();

};

#endif

#include "translacaoV.h"

using namespace std;

TranslacaoV::TranslacaoV() {
        this->trans.fill(0.0f);
}

void TranslacaoV::setTrans( array<float, 3> trans ) {
        this->trans = trans;
}

void TranslacaoV::aplica() {
        glTranslatef(trans[0], trans[1], trans[2]);
}
```

K Código da Main

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glew.h>
#include <GL/glut.h>
#endif

#define _USE_MATH_DEFINES
#include <math.h>

#include <iostream>
#include "./Deps/tinyxml2.h"
#include <string>
#include <vector>
#include <fstream>
#include <list>
#include "./Deps/timedsg.h"
#include "./Deps/engine.h"


using namespace tinyxml2;

TimedSG ts_g;
SceneGraph s_gg;

#define GROWF 0.01

double alfa = M_PI / 4;
double beta = M_PI / 4;
float raio = 350.0f;

extern int nrModels;


void changeSize(int w, int h) {

        // Prevent a divide by zero, when window is too short
        // (you cant make a window with zero width).
        if(h == 0)
                h = 1;

        // compute window's aspect ratio
        float ratio = w * 1.0 / h;

        // Set the projection matrix as current
        glMatrixMode(GL_PROJECTION);
        // Load Identity Matrix
        glLoadIdentity();

        // Set the viewport to be the entire window
    glViewport(0, 0, w, h);

        // Set perspective
        gluPerspective(45.0f ,ratio, 1.0f ,1000.0f);
```

```
        // return to the model view matrix mode
        glMatrixMode(GL_MODELVIEW);
}


void renderScene(void) {

        // clear buffers
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // set the camera
    glLoadIdentity();
    gluLookAt(raio * cos(beta) * cos(alfa), raio * cos(beta) * sin(alfa), raio * sin
              0.0,75.0,0.0,
              0.0f,0.0f,1.0f);

    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    glColor3f(1.0f,1.0f,1.0f);

    ts_g.draw();

        // End of frame
        glutSwapBuffers();
}


void processSpecialKeys(int key, int xx, int yy) {


// put code to process special keys in here
    switch(key) {
        case GLUT_KEY_LEFT:
            alfa += GROWF;
            break;
        case GLUT_KEY_RIGHT:
            alfa -= GROWF;
            break;
        case GLUT_KEY_UP:
            beta += GROWF;
            break;
        case GLUT_KEY_DOWN:
            beta -= GROWF;
            break;
        default:
            ;
    }

    glutPostRedisplay();


}

void init() {
    glewInit();

    // OpenGL Settings
```

```
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    glEnableClientState(GL_VERTEX_ARRAY);

    ts_g.prep();
}



int main(int argc, char **argv) {



    tinyxml2::XMLDocument doc;
    //new file
    doc.LoadFile("./conf.xml");
    tinyxml2::XMLNode *scene = doc.FirstChild();
    if (scene == nullptr) perror("Erro de Leitura.\n");

    s_gg = doGroup(scene->FirstChildElement("group"));

    ts_g.setSG(s_gg);

// init GLUT and the window
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(800,800);
        glutCreateWindow("CG@DI-UM");

// Required callback registry
        glutDisplayFunc(renderScene);
        glutIdleFunc(renderScene);
        glutReshapeFunc(changeSize);

// Callback registration for keyboard processing
        glutSpecialFunc(processSpecialKeys);

// Init function Call
    init();

// enter GLUT's main cycle
        glutMainLoop();

        return 1;
}
```

L Codigo do Gerador

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <fstream>
#include <vector>
#include <iostream>
#define F_PI (float)M_PI
#define cp1(r, a)     (r * sin(a))
#define cp2(r, a)     (r * cos(a))
#define ce1(r, a, b)  (r * cos(b) * sin(a))
#define ce2(r, b)     (r * sin(b))
#define ce3(r, a, b)  (r * cos(b) * cos(a))
#define norma(a,b,c)  ( sqrt(a*a + b*b + c*c) )
#define texX(a)       (a/(2.0*F_PI))
#define texY(b)       ((b/F_PI)+0.5)
#define crossX( ay, bz, az, by) (ay * bz + az * by )
#define crossY( az, bx, ax, bz) (az * bx + ax * bz )
#define crossZ( ax, by, ay, bx) (ax * by + ay * bx )

using namespace std;

struct Ponto {
    float x;
    float y;
    float z;
};



void printSphere(float radius, int slices, int stacks, FILE* f){

    // variaveis que vem o numero da stack atual
    vector<float> vertices;
    vector<float> normais ;
    vector<float> textura ;
    float stkd, slcd;
    float slc, stk, nslc, nstk;
    int j;
    // delta dos angulos por stack
    stkd = M_PI / stacks;
    // delta dos angulos por slice
    slcd = 2 * M_PI / slices;

    //base
    for(int i = 0; i < slices; ++i) {

        j = 1;
        slc = i * slcd;
        nslc = (i+1) * slcd;
        stk = -M_PI_2 + j * stkd;

        vertices.push_back(0.0);
        vertices.push_back(-radius);
        vertices.push_back(0.0);
```

```
normais.push_back(0.0);
normais.push_back(-1.0);
normais.push_back(0.0);

textura.push_back( 0.0 );
textura.push_back( 0.0 );

vertices.push_back(ce1(radius, nslc, stk));
vertices.push_back(ce2(radius, stk));
vertices.push_back(ce3(radius, nslc, stk));

normais.push_back( ce1( 1.0, nslc, stk) );
normais.push_back( ce2( 1.0, stk) );
normais.push_back( ce3( 1.0, nslc, stk) );

textura.push_back( texX(nslc) );
textura.push_back( texY(stk) );

vertices.push_back( ce1(radius, slc, stk));
vertices.push_back( ce2(radius, stk));
vertices.push_back( ce3(radius, slc, stk));

normais.push_back( ce1(1.0, slc, stk) );
normais.push_back( ce2(1.0, stk) );
normais.push_back( ce3(1.0, slc, stk) );

textura.push_back( texX(slc) );
textura.push_back( texY(stk) );

for(; j < stacks-1; ++j) {

    slc = i * slcd;
    nslc = (i+1) * slcd;
    stk = -M_PI_2 + j * stkd;
    nstk = -M_PI_2 + (j+1) * stkd;

    vertices.push_back( ce1(radius, nslc, nstk) );
    vertices.push_back( ce2(radius, nstk) );
    vertices.push_back( ce3(radius, nslc, nstk) );

    normais.push_back( ce1(1.0, nslc, nstk) );
    normais.push_back( ce2(1.0, nstk) );
    normais.push_back( ce3(1.0, nslc, nstk) );

    textura.push_back( texX(nslc) );
    textura.push_back( texY(nstk) );

    vertices.push_back( ce1(radius, slc, stk));
    vertices.push_back( ce2(radius, stk));
    vertices.push_back( ce3(radius, slc, stk));

    normais.push_back( ce1(1, slc, stk) );
    normais.push_back( ce2(1, stk));
    normais.push_back( ce3(1, slc, stk) );

    textura.push_back( texX(slc) );
```

```
    textura.push_back( texY(stk) );

    vertices.push_back( ce1(radius, nslc, stk));
    vertices.push_back( ce2(radius, stk));
    vertices.push_back( ce3(radius, nslc, stk));

    normais.push_back( ce1(1, nslc, stk) );
    normais.push_back( ce2(1, stk));
    normais.push_back( ce3(1, nslc, stk) );

    textura.push_back( texX(nslc) );
    textura.push_back( texY(stk) );

    vertices.push_back( ce1(radius, slc, nstk));
    vertices.push_back( ce2(radius, nstk));
    vertices.push_back( ce3(radius, slc, nstk));

    normais.push_back( ce1(1, slc, nstk) );
    normais.push_back( ce2(1, nstk) );
    normais.push_back( ce3(1, slc, nstk) );

    textura.push_back( texX(slc) );
    textura.push_back( texY(nstk) );

    vertices.push_back( ce1(radius, slc, stk));
    vertices.push_back( ce2(radius, stk));
    vertices.push_back( ce3(radius, slc, stk));

    normais.push_back( ce1(1, slc, stk));
    normais.push_back( ce2(1, stk));
    normais.push_back( ce3(1, slc, stk) );

    textura.push_back( texX(slc) );
    textura.push_back( texY(stk) );

    vertices.push_back( ce1(radius, nslc, nstk)) ;
    vertices.push_back( ce2(radius, nstk) );
    vertices.push_back( ce3(radius, nslc, nstk) );

    normais.push_back( ce1(1, nslc, nstk) );
    normais.push_back( ce2(1, nstk) );
    normais.push_back( ce3(1, nslc, nstk) );

    textura.push_back( texX(nslc) );
    textura.push_back( texY(nstk));

}

slc = i * slcd;
nslc = (i+1) * slcd;
stk = -M_PI / 2 + j * stkd;

vertices.push_back(0.0);
vertices.push_back(radius);
vertices.push_back(0.0);

normais.push_back(0.0);
```

```
        normais.push_back(1.0);
        normais.push_back(0.0);

        textura.push_back(1.0);
        textura.push_back(1.0);

        vertices.push_back( ce1(radius, slc, stk) );
        vertices.push_back( ce2(radius, stk) );
        vertices.push_back( ce3(radius, slc, stk) );

        normais.push_back( ce1(1, slc, stk) );
        normais.push_back( ce2(1, stk) );
        normais.push_back( ce3(1, slc, stk) );

        textura.push_back( texX(slc) );
        textura.push_back( texY(stk) );

        vertices.push_back( ce1(radius, nslc, stk) );
        vertices.push_back( ce2(radius, stk) );
        vertices.push_back( ce3(radius, nslc, stk) );

        normais.push_back( ce1(1, nslc, stk) );
        normais.push_back( ce2(1, stk) );
        normais.push_back( ce3(1, nslc, stk) );

        textura.push_back( texX(nslc) );
        textura.push_back( texY(stk) );

    }
    //numero de vertices
    fprintf(f,"%lu\n", vertices.size() );

    for(float i:vertices){
        fprintf(f,"%f\n",i);
    }

    for(float i:normais){
        fprintf(f,"%f\n",i);
    }

    for(float i:textura){
        fprintf(f,"%f\n",i);
    }

}

void printBox ( int ndivs, float xx , float yy , float zz , FILE *f ){

    vector<float> vertices;
    vector<float> normais ;
    vector<float> textura ;

    float xxdelta , yydelta , zzdelta ;
    xxdelta = xx /ndivs;
    yydelta = yy /ndivs;
    zzdelta = zz /ndivs;
```

```
float texdelta = 1.0 / ndivs;
//front
for (int i = 0; i < ndivs; i++)
{
    for (int j = 0; j < ndivs; j++)
    {
        vertices.push_back( i*xxdelta-xx/2);
        vertices.push_back( j*yydelta-yy/2);
        vertices.push_back( zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(1.0);

        textura.push_back( i*texdelta );
        textura.push_back( j*texdelta );

        vertices.push_back( (i+1)*xxdelta-xx/2);
        vertices.push_back( (j+1)*yydelta-yy/2);
        vertices.push_back( zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(1.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back( i*xxdelta-xx/2);
        vertices.push_back( (j+1)*yydelta-yy/2);
        vertices.push_back( zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(1.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back(i*xxdelta-xx/2);
        vertices.push_back(  j*yydelta-yy/2);
        vertices.push_back( zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(1.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back((i+1)*xxdelta-xx/2);
        vertices.push_back( j*yydelta-yy/2);
        vertices.push_back( zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(1.0);
```

```
        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back( (i+1)*xxdelta-xx/2);
        vertices.push_back( (j+1)*yydelta-yy/2);
        vertices.push_back( zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(1.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

    }
}
//back
for (int i = 0; i < ndivs; i++)
{
    for (int j = 0; j < ndivs; j++)
    {

        vertices.push_back( i*xxdelta-xx/2);
        vertices.push_back(  j*yydelta-yy/2);
        vertices.push_back( -zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(-1.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back( (i+1)*xxdelta-xx/2);
        vertices.push_back((j+1)*yydelta-yy/2);
        vertices.push_back( -zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(-1.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back( i*xxdelta-xx/2);
        vertices.push_back( (j+1)*yydelta-yy/2);
        vertices.push_back( -zz/2);

        normais.push_back(0.0);
        normais.push_back(0.0);
        normais.push_back(-1.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back( i*xxdelta-xx/2);
```

```
            vertices.push_back( j*yydelta-yy/2);
            vertices.push_back( -zz/2);

            normais.push_back(0.0);
            normais.push_back(0.0);
            normais.push_back(-1.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back( (i+1)*xxdelta-xx/2);
            vertices.push_back( j*yydelta-yy/2);
            vertices.push_back( -zz/2);

            normais.push_back(0.0);
            normais.push_back(0.0);
            normais.push_back(-1.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back( (i+1)*xxdelta-xx/2);
            vertices.push_back( (j+1)*yydelta-yy/2);
            vertices.push_back( -zz/2);

            normais.push_back(0.0);
            normais.push_back(0.0);
            normais.push_back(-1.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

        }
    }
    //top
    for (int i = 0; i < ndivs; i++)
    {
        for (int j = 0; j < ndivs; j++)
        {
            vertices.push_back( i*xxdelta-xx/2);
            vertices.push_back( yy/2);
            vertices.push_back( j*zzdelta-zz/2);

            normais.push_back(0.0);
            normais.push_back(1.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back( (i+1)*xxdelta-xx/2);
            vertices.push_back( yy/2);
            vertices.push_back( (j+1)*zzdelta-zz/2 );

            normais.push_back(0.0);
            normais.push_back(1.0);
            normais.push_back(0.0);
```

```
            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back( i*xxdelta-xx/2);
            vertices.push_back( yy/2);
            vertices.push_back((j+1)*zzdelta-zz/2);

            normais.push_back(0.0);
            normais.push_back(1.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back( i*xxdelta-xx/2);
            vertices.push_back( yy/2);
            vertices.push_back( j*zzdelta-zz/2);

            normais.push_back(0.0);
            normais.push_back(1.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back( (i+1)*xxdelta-xx/2);
            vertices.push_back( yy/2);
            vertices.push_back( j*zzdelta-zz/2);

            normais.push_back(0.0);
            normais.push_back(1.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back( (i+1)*xxdelta-xx/2);
            vertices.push_back( yy/2);
            vertices.push_back( (j+1)*zzdelta-zz/2);

            normais.push_back(0.0);
            normais.push_back(1.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );


        }
    }

    //bottom
    for (int i = 0; i < ndivs; i++)
    {
        for (int j = 0; j < ndivs; j++)
        {
```

```
vertices.push_back(i*xxdelta-xx/2);
vertices.push_back(-yy/2);
vertices.push_back(j*zzdelta-zz/2);

normais.push_back(0.0);
normais.push_back(-1.0);
normais.push_back(0.0);

textura.push_back(i*texdelta );
textura.push_back(j*texdelta );

vertices.push_back((i+1)*xxdelta-xx/2);
vertices.push_back(-yy/2);
vertices.push_back((j+1)*zzdelta-zz/2);

normais.push_back(0.0);
normais.push_back(-1.0);
normais.push_back(0.0);

textura.push_back(i*texdelta );
textura.push_back(j*texdelta );

vertices.push_back(i*xxdelta-xx/2);
vertices.push_back(-yy/2);
vertices.push_back((j+1)*zzdelta-zz/2);

normais.push_back(0.0);
normais.push_back(-1.0);
normais.push_back(0.0);

textura.push_back(i*texdelta );
textura.push_back(j*texdelta );

vertices.push_back(i*xxdelta-xx/2);
vertices.push_back(-yy/2);
vertices.push_back(j*zzdelta-zz/2);

normais.push_back(0.0);
normais.push_back(-1.0);
normais.push_back(0.0);

textura.push_back(i*texdelta );
textura.push_back(j*texdelta );

vertices.push_back((i+1)*xxdelta-xx/2);
vertices.push_back(-yy/2);
vertices.push_back(j*zzdelta-zz/2);

normais.push_back(0.0);
normais.push_back(-1.0);
normais.push_back(0.0);

textura.push_back(i*texdelta );
textura.push_back(j*texdelta );

vertices.push_back((i+1)*xxdelta-xx/2);
vertices.push_back(-yy/2);
```

```
        vertices.push_back((j+1)*zzdelta-zz/2);

        normais.push_back(0.0);
        normais.push_back(-1.0);
        normais.push_back(0.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );


    }
}

//righty
for (int i = 0; i < ndivs; i++)
{
    for (int j = 0; j < ndivs; j++)
    {
        vertices.push_back(xx/2);
        vertices.push_back(i*yydelta-yy/2);
        vertices.push_back(j*zzdelta-zz/2);

        normais.push_back(1.0);
        normais.push_back(0.0);
        normais.push_back(0.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back(xx/2);
        vertices.push_back((i+1)*yydelta-yy/2);
        vertices.push_back((j+1)*zzdelta-zz/2);

        normais.push_back(1.0);
        normais.push_back(0.0);
        normais.push_back(0.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back(xx/2);
        vertices.push_back(i*yydelta-yy/2);
        vertices.push_back((j+1)*zzdelta-zz/2);

        normais.push_back(1.0);
        normais.push_back(0.0);
        normais.push_back(0.0);

        textura.push_back(i*texdelta );
        textura.push_back(j*texdelta );

        vertices.push_back(xx/2);
        vertices.push_back(i*yydelta-yy/2);
        vertices.push_back(j*zzdelta-zz/2);

        normais.push_back(1.0);
        normais.push_back(0.0);
```

```
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back(xx/2);
            vertices.push_back((i+1)*yydelta-yy/2);
            vertices.push_back(j*zzdelta-zz/2 );

            normais.push_back(1.0);
            normais.push_back(0.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back(xx/2);
            vertices.push_back((i+1)*yydelta-yy/2);
            vertices.push_back((j+1)*zzdelta-zz/2 );

            normais.push_back(1.0);
            normais.push_back(0.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );
        }
    }
    //lefty
    for (int i = 0; i < ndivs; i++)
    {
        for (int j = 0; j < ndivs; j++)
        {
            vertices.push_back(-xx/2);
            vertices.push_back(i*yydelta-yy/2);
            vertices.push_back(j*zzdelta-zz/2);

            normais.push_back(-1.0);
            normais.push_back(0.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back(-xx/2);
            vertices.push_back((i+1)*yydelta-yy/2);
            vertices.push_back((j+1)*zzdelta-zz/2);

            normais.push_back(-1.0);
            normais.push_back(0.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back(-xx/2);
            vertices.push_back(i*yydelta-yy/2);
```

```
            vertices.push_back((j+1)*zzdelta-zz/2);

            normais.push_back(-1.0);
            normais.push_back(0.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back(-xx/2);
            vertices.push_back(i*yydelta-yy/2);
            vertices.push_back(j*zzdelta-zz/2);

            normais.push_back(-1.0);
            normais.push_back(0.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back(-xx/2);
            vertices.push_back((i+1)*yydelta-yy/2);
            vertices.push_back(j*zzdelta-zz/2 );

            normais.push_back(-1.0);
            normais.push_back(0.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

            vertices.push_back(-xx/2);
            vertices.push_back((i+1)*yydelta-yy/2);
            vertices.push_back((j+1)*zzdelta-zz/2 );

            normais.push_back(-1.0);
            normais.push_back(0.0);
            normais.push_back(0.0);

            textura.push_back(i*texdelta );
            textura.push_back(j*texdelta );

        }
    }

    fprintf(f,"%lu\n", vertices.size() );

    for(float i:vertices){
        fprintf(f,"%f\n",i);
    }

    for(float i:normais){
        fprintf(f,"%f\n",i);
    }

    for(float i:textura){
        fprintf(f,"%f\n",i);
```

```
    }

}

void printCone(float radius, float altura, int slices, int stacks, FILE *f) {


    vector<float> vertices;
    vector<float> normais ;
    vector<float> textura ;

    float stkd, slcd, raiod;
    float stk, slc, nslc, nstk, nr, r;
    float  nX , nY, nZ , nn, nX2 , nY2, nZ2 , nn2;

    stkd = altura / stacks;
    slcd = 2 * M_PI / slices;
    raiod = radius / stacks;

    float texslc, texstk;

    texslc = 1 / slices;
    texstk = 0.625 / stacks;

    int j;

    for(int i = 0; i < slices; i++) {

        // codigo responsavel por gerar uma slice da base

        slc = i * slcd;
        nslc = (i+1) * slcd;

        vertices.push_back(0.0);
        vertices.push_back(0.0);
        vertices.push_back(0.0);

        normais.push_back(0.0);
        normais.push_back(-1.0);
        normais.push_back(0.0);

        textura.push_back(0.8125);
        textura.push_back(0.18);

        vertices.push_back(cp1(radius, nslc));
        vertices.push_back(0.0);
        vertices.push_back(cp2(radius, nslc));

        normais.push_back( 0.0  );
        normais.push_back( -1.0 );
        normais.push_back( 0.0 );

        textura.push_back( 0.8125 + cp1(0.1875, nslc));
        textura.push_back( 0.18 + cp2(0.1875, nslc));

        vertices.push_back( cp1(radius, slc) );
        vertices.push_back( 0.0 );
```

```
vertices.push_back( cp2(radius, slc) );

normais.push_back( 0.0 );
normais.push_back( -1.0 );
normais.push_back( 0.0 );

textura.push_back( 0.8125 + cp1(0.1875, slc));
textura.push_back( 0.18 + cp2(0.1875, slc));

    j=stacks;
    slc = i * slcd;
    nslc = (i+1) * slcd;
    stk = (stacks - j) * stkd;
    nstk = (stacks - (j-1)) * stkd;
    r = j * raiod;
    nr = (j - 1) * raiod;


    nX = crossX(  stk, cp2(nr,slc), cp2(r, nslc), nstk );
    nY = crossY(cp2(r,nslc), cp1(nr,slc), cp1(r,nslc), cp2(nr,slc));
    nZ = crossZ ( cp1(r,nslc), nstk, stk, cp1(nr,slc));

    nn = norma(nX, nY, nZ);

    nX /= nn ;
    nY /= nn ;
    nZ /= nn ;

    nX2 = crossX(nstk, cp2(r,slc), cp2(nr,nslc), stk);
    nY2 = crossY(cp2(nr,nslc), cp1(r,slc), cp1(nr,nslc), cp2(r,slc));
    nZ2 = crossZ(cp1(nr,nslc), stk, nstk, cp1(r,slc));

    nn2 = norma(nX2,nY2,nZ2);

    nX2 /= nn2 ;
    nY2 /= nn2 ;
    nZ2 /= nn2 ;

// codigo responsavel por gerar as slices laterais
for(j = stacks ; j > 1; j--) {

    slc = i * slcd;
    nslc = (i+1) * slcd;
    stk = (stacks - j) * stkd;
    nstk = (stacks - (j-1)) * stkd;
    r = j * raiod;
    nr = (j - 1) * raiod;


    vertices.push_back(cp1(nr, nslc));
    vertices.push_back(nstk);
    vertices.push_back(cp2(nr, nslc));

    normais.push_back( nX2 );
    normais.push_back( nY2 );
    normais.push_back( nZ2 );
```

```
textura.push_back( (i+1) * texslc );
textura.push_back( (j+1) * texstk );

vertices.push_back(cp1(r, slc));
vertices.push_back(stk);
vertices.push_back( cp2(r, slc));

normais.push_back( nX );
normais.push_back( nY );
normais.push_back( nZ );

textura.push_back( i * texslc);
textura.push_back( j * texstk);

vertices.push_back(cp1(r, nslc));
vertices.push_back(stk);
vertices.push_back(cp2(r, nslc));

normais.push_back( nX2 );
normais.push_back( nY2 );
normais.push_back( nZ2 );

textura.push_back( (i+1) * texslc );
textura.push_back( j * texstk );

vertices.push_back(cp1(nr, slc));
vertices.push_back(nstk);
vertices.push_back(cp2(nr, slc));

normais.push_back( nX );
normais.push_back( nY );
normais.push_back( nZ );

textura.push_back( i * texslc );
textura.push_back( (j+1) * texstk );

vertices.push_back(cp1(r, slc));
vertices.push_back(stk);
vertices.push_back(cp2(r, slc));

normais.push_back( nX );
normais.push_back( nY );
normais.push_back( nZ );

textura.push_back( i * texslc );
textura.push_back( j * texstk );

vertices.push_back(cp1(nr, nslc));
vertices.push_back(nstk);
vertices.push_back(cp2(nr, nslc));

normais.push_back( nX2 );
normais.push_back( nY2 );
normais.push_back( nZ2 );

textura.push_back( (i+1) * texslc );
textura.push_back( (j+1) * texstk );
```

```
    }
    // codigo responsavel por gerar a slice do topo
    slc = i * slcd;
    nslc = (i+1) * slcd;
    stk = (stacks - j) * stkd;
    r = j * raiod;

    vertices.push_back(0.0);
    vertices.push_back(altura);
    vertices.push_back(0.0);

    normais.push_back(0.0);
    normais.push_back(1.0);
    normais.push_back(0.0);

    textura.push_back( 0.5 );
    textura.push_back( 1 );

    vertices.push_back(cp1(r, slc));
    vertices.push_back(stk);
    vertices.push_back( cp2(r, slc));

    normais.push_back( nX );
    normais.push_back( nY );
    normais.push_back( nZ );

    textura.push_back( i * texslc );
    textura.push_back( j * texslc );

    vertices.push_back(cp1(r, nslc));
    vertices.push_back(stk);
    vertices.push_back(cp2(r, nslc));

    normais.push_back(nX2);
    normais.push_back(nY2);
    normais.push_back(nZ2);

    textura.push_back( (i+1) * texslc);
    textura.push_back( j * texstk );
}

fprintf(f,"%lu\n", vertices.size() );

for(float i:vertices){
    fprintf(f,"%f\n",i);
}

for(float i:normais){
    fprintf(f,"%f\n",i);
}

for(float i:textura){
    fprintf(f,"%f\n",i);
}
```

```
}

void printPlano(float tam,FILE *f){

    vector<float> vertices;
    vector<float> normais ;
    vector<float> textura ;


    vertices.push_back(tam/2);
    vertices.push_back(0.0);
    vertices.push_back(tam/2);

    normais.push_back(0.0);
    normais.push_back(1.0);
    normais.push_back(0.0);

    textura.push_back(1.0);
    textura.push_back(0.0);

    vertices.push_back(-tam/2);
    vertices.push_back(0.0);
    vertices.push_back(tam/2);

    normais.push_back(0.0);
    normais.push_back(1.0);
    normais.push_back(0.0);

    textura.push_back(0.0);
    textura.push_back(0.0);

    vertices.push_back(tam/2);
    vertices.push_back(0.0);
    vertices.push_back(-tam/2);

    normais.push_back(0.0);
    normais.push_back(1.0);
    normais.push_back(0.0);

    textura.push_back(1.0);
    textura.push_back(1.0);

    vertices.push_back(tam/2);
    vertices.push_back(0.0);
    vertices.push_back(tam/2);

    normais.push_back(0.0);
    normais.push_back(1.0);
    normais.push_back(0.0);

    textura.push_back(1.0);
    textura.push_back(0.0);

    vertices.push_back(tam/2);
    vertices.push_back(0.0);
    vertices.push_back(-tam/2);
```

```
normais.push_back(0.0);
normais.push_back(1.0);
normais.push_back(0.0);

textura.push_back(1.0);
textura.push_back(1.0);

vertices.push_back(-tam/2);
vertices.push_back(0.0);
vertices.push_back(tam/2);

normais.push_back(0.0);
normais.push_back(1.0);
normais.push_back(0.0);

textura.push_back(0.0);
textura.push_back(0.0);

vertices.push_back(tam/2);
vertices.push_back(0.0);
vertices.push_back(-tam/2);

normais.push_back(0.0);
normais.push_back(1.0);
normais.push_back(0.0);

textura.push_back(1.0);
textura.push_back(1.0);

vertices.push_back(-tam/2);
vertices.push_back(0.0);
vertices.push_back(tam/2);

normais.push_back(0.0);
normais.push_back(1.0);
normais.push_back(0.0);

textura.push_back(0.0);
textura.push_back(0.0);

vertices.push_back(-tam/2);
vertices.push_back(0.0);
vertices.push_back(-tam/2);

normais.push_back(0.0);
normais.push_back(1.0);
normais.push_back(0.0);

textura.push_back(0.0);
textura.push_back(1.0);

vertices.push_back(tam/2);
vertices.push_back(0.0);
vertices.push_back(-tam/2);

normais.push_back(0.0);
normais.push_back(1.0);
```

```
    normais.push_back(0.0);

    textura.push_back(1.0);
    textura.push_back(1.0);

    vertices.push_back(-tam/2);
    vertices.push_back(0.0);
    vertices.push_back(-tam/2);

    normais.push_back(0.0);
    normais.push_back(1.0);
    normais.push_back(0.0);

    textura.push_back(1.0);
    textura.push_back(0.0);

    vertices.push_back(-tam/2);
    vertices.push_back(0.0);
    vertices.push_back(tam/2);

    normais.push_back(0.0);
    normais.push_back(1.0);
    normais.push_back(0.0);

    textura.push_back(0.0);
    textura.push_back(0.0);

    fprintf(f,"36\n");

    for( float i: vertices ){
        fprintf(f,"%f\n", i);
    }

    for( float i: normais ){
        fprintf(f,"%f\n", i);
    }

    for( float i: textura ){
        fprintf(f,"%f\n", i);
    }

}

int main( int i ,char *args[] ) {

    if ( !strcmp("cone", args[1]) ){
        FILE *f = fopen( args[6],"w");
        printCone( atof(args[2]), atof(args[3]), atoi(args[4]), atoi(args[5]), f);
        fclose(f);
    }
    else if ( !strcmp("caixa", args[1]) ){
        FILE *f = fopen( args[6],"w");
        printBox( atof(args[2]), atof(args[3]), atof(args[4]), atof(args[5]), f);
        fclose(f);
    }
    else if ( !strcmp("esfera", args[1]) ){
        FILE *f = fopen( args[5],"w");
```

```
        printSphere( atof( args[2] ), atoi( args[3] ), atoi( args[4] ), f);
        fclose(f);
    }
    else if ( !strcmp("plano", args[1]) ){
        FILE *f = fopen( args[3],"w");
        printPlano( atof( args[2] ),f );
        fclose(f);
    }

    return 0;
}
```

M XML usado para a animação

```
<scene>
    <group>
        <group>
            <!-- Sol -->
            <translate y="-3" />
            <scale x="20" y="20" z="20" />
            <models>
                <model file="esfera.txt"/>
            </models>
        </group>
        <group>
            <!-- Mercurio -->
            <translate time="1500">
                <point x="0" y="55" z="0"/>
                <point x="-110" y="0" z="0"/>
                <point x="0" y="-110" z="0"/>
                <point x="110" y="0" z="0"/>
            </translate>
            <scale x="0.5" y="0.5" z="0.5"/>
            <models>
                <model file="esfera.txt"/>
            </models>
        </group>
        <group>
            <!-- Venus -->
            <translate time="1500">
                <point x="0" y="55" z="0"/>
                <point x="-110" y="0" z="0"/>
                <point x="0" y="-110" z="0"/>
                <point x="110" y="0" z="0"/>
            </translate>
            <scale x="1" y="1" z="1"/>
            <models>
                <model file="esfera.txt"/>
            </models>
        </group>
        <group>
            <!-- Terra -->
            <translate time="1500">
                <point x="0" y="55" z="0"/>
                <point x="-110" y="0" z="0"/>
                <point x="0" y="-110" z="0"/>
                <point x="110" y="0" z="0"/>
            </translate>
            <rotate time="500" x="0" y="1" z="0" />
            <scale x="1" y="1" z="1"/>
            <models>
                <model file="esfera.txt"/>
            </models>
            <group>
                <!-- Lua -->
                <translate x="25" />
                <scale x="0.2" y="0.2" z="0.2"/>
                <models>
                    <model file="esfera.txt"/>
```

```
        </models>
    </group>
</group>
<group>
    <!-- Marte -->
    <translate time="1500">
        <point x="0" y="55" z="0"/>
        <point x="-110" y="0" z="0"/>
        <point x="0" y="-110" z="0"/>
        <point x="110" y="0" z="0"/>
    </translate>
    <scale x="1.4" y="1.4" z="1.4"/>
    <models>
        <model file="esfera.txt" />
    </models>
</group>
<group>
    <!-- Jupiter -->
    <translate time="1500">
        <point x="0" y="55" z="0"/>
        <point x="-110" y="0" z="0"/>
        <point x="0" y="-110" z="0"/>
        <point x="110" y="0" z="0"/>
    </translate>
    <scale x="4" y="4" z="4"/>
    <models>
        <model file="esfera.txt" />
    </models>
</group>
<group>
    <!-- Saturno -->
    <translate time="1500">
        <point x="0" y="55" z="0"/>
        <point x="-110" y="0" z="0"/>
        <point x="0" y="-110" z="0"/>
        <point x="110" y="0" z="0"/>
    </translate>
    <scale x="3" y="3" z="3"/>
    <models>
        <model file="esfera.txt" />
    </models>
</group>
<group>
    <!-- Urano -->
    <translate time="1500">
        <point x="0" y="55" z="0"/>
        <point x="-110" y="0" z="0"/>
        <point x="0" y="-110" z="0"/>
        <point x="110" y="0" z="0"/>
    </translate>
    <scale x="2" y="2" z="2"/>
    <models>
        <model file="esfera.txt" />
    </models>
</group>
<group>
    <!-- Neptuno -->
```

```
            <translate time="1500">
                <point x="0" y="55" z="0"/>
                <point x="-110" y="0" z="0"/>
                <point x="0" y="-110" z="0"/>
                <point x="110" y="0" z="0"/>
            </translate>
            <scale x="1" y="1" z="1"/>
            <models>
                <model file="esfera.txt" />
            </models>
        </group>
    </group>
</scene>
```