Estruturas

Nesta fase necessitamos de mais estruturas das usadas na fase anterior. Por forma a realizar as transformações geométricas e como estas são de 3 diferentes tipos necessitamos de uma classe **Operação** que corresponde a uma transformação geométrica. No entanto esta terá de ser abstracta pois nunca pode ser concretizada, ou seja, uma operação é um caso geral, mas a transformação em si é um caso particular desta, pelo que criamos 3 subclasses da classe **Operação**: **Rotação, Translação, Escala**, que correspondem às transformações que necessitamos. Para além desta, e como o ficheiro *XML* está organizado por grupos decidimos também ter uma classe **Grupo**, que corresponde a um *group* do ficheiro *XML*. Por forma a ter uma câmera móvel, decidimos também implementar uma classe **Camera.**

# Operação

A classe operação corresponde a uma transformação geométrica, pelo que é abstracta (como referido anteriormente). Esta classe apenas tem 3 variáveis do tipo *float* que correspondem respetivamente aos valores relativos aos eixos **x,y** e **z** pelo que têm exatamente esse nome (todas as transformações contém estas variáveis). Para além disto necessitamos de criar um método para que a transformação fosse criada, pelo que temos o método **aplicaOperação** (método abstracto que apenas é realizado nas subclasses).

class Operacao {

float x;

float y;

float z;

public:

Operacao();

Operacao(float,float,float);

float getX();

float getY();

float getZ();

void setX(float);

void setY(float);

void setZ(float);

virtual void aplicaOperacao() = 0;

virtual string toString() = 0;

}

## Rotação

Esta classe é uma subclasse da **Operacao** e corresponde a uma rotação, pelo que necessita de mais uma variável, o **angulo** da rotação. Para além disto a função **aplicaOperacao** realiza uma rotação (*glRotate*) segundo as variáveis da classe.

class Rotacao: public Operacao {

private:

float angulo;

public:

Rotacao();

Rotacao(float,float,float,float);

float getAngulo();

void setAngulo(float);

void aplicaOperacao();

string toString();

}

void Rotacao::aplicaOperacao(){

glRotatef(angulo,getX(),getY(),getZ());

}

## Translação

Esta classe é uma subclasse da **Operacao** e corresponde a uma translação, pelo que não necessita de mais nenhuma variável das correspondentes ao **x, y** e **z**. A função **aplicaOperacao** realiza uma translação (*glTranslate*) segundo as variáveis da classe.

void Translacao::aplicaOperacao(){

glTranslatef(getX(),getY(),getZ());

}

## Escala

Esta classe é uma subclasse da **Operacao** e corresponde a uma escala, pelo que não necessita de mais nenhuma variável das correspondentes ao **x, y** e **z**. Ainda, a função **aplicaOperacao** realiza uma escala (*glScale*) segundo as variáveis da classe.

void Escala::aplicaOperacao(){

glScalef(getX(),getY(),getZ());

}

# Grupo

A classe **Grupo** corresponde a um *group* presente no ficheiro *XML*. Como identificamos no ficheiro, um grupo pode conter várias operações (ordenadas), várias figuras e vários grupos (filhos), pelo que esta classe contém:

* um *vector* de apontadores para **Operacao**, sendo que cada operação é inserida no fim desse vector (por causa da ordem)
* um *vector* de apontadores para **Figura**, correspondente às figuras
* um *vector* de apontadores para **Grupo**, correspondendo aos filhos

Para além disto, para identificar um grupo decidimos ter um inteiro que é o **id** do mesmo.

class Grupo {

private:

int id;

vector<Operacao\*> operacoes;

vector<Figura\*> figuras;

vector<Grupo\*> grupos;

public:

Grupo(int);

vector<Operacao\*> getOperacoes();

vector<Figura\*> getFiguras();

vector<Grupo\*> getGrupos();

void adicionaOperacao(Operacao\*);

void adicionaFigura(Figura\*);

void adicionaGrupo(Grupo\*);

string toString();

}

void Grupo::adicionaOperacao(Operacao\* a){

operacoes.push\_back(a);

}

Outros casos são semelhantes.

# Câmera

POR AQUI A VOSSA EXPLICAÇÃO!