# Motor

Nesta fase tivemos de alterar alguns aspetos do motor, nomeadamente os relativos às transformações geométricas. Para além disso e por forma a poder utilizar VBO’s tivemos também de alterar um pouco a classe Figura e também incluir algumas bibliotecas para os poder utilizar, a *GLEW e DevIL*.

## Translacao

Por forma a permitir definir as curvas de *Catmull-Rom* para as translações das figuras, decidimos que a classe translação teria um conjuntos de pontos de controlo. Para além disto, como esta translação é dependente de um tempo foi necessário guardar também o valor desse mesmo tempo. Por forma a não gerar todas as iterações os pontos da curva (de translação), e visto que a tínhamos de mostrar, então guardamos também um conjunto de todos os pontos da curva, geradas com a fórmula para as curvas de *Catmull-Rom* com um tempo a variar entre 0 e 1 com um delta igual a 0.01.

vector<Ponto\*> Translacao::getCurva(){

if(curva.size()==0){

float res[3];

float deriv[3];

for (float t = 0; t<1; t += 0.01){

getGlobalCatmullRomPoint(t, res, deriv);

Ponto\* p = new Ponto(res[0], res[1], res[2]);

curva.push\_back(p);

}

}

return curva;

}

Para além disto, era necessário ter um vetor referente ao *upVector* que é importante para a rotação do planeta segundo a trajetória consoante a derivada no ponto em questão.

class Translacao: public Operacao {

private:

float tempo;

vector<Ponto\*> pontos;

vector<Ponto\*> curva;

float upY[3];

void normalize(float\*);

void getCatmullRomPoint(float,int\*,float\*,float\*);

void getGlobalCatmullRomPoint(float,float\*,float\*);

void rodaElemento(float\*);

public:

Translacao();

Translacao(float,float,float,float,vector<Ponto\*>);

float getTempo();

void setTempo(float);

vector<Ponto\*> getPontos();

void setPontos(vector<Ponto\*>);

vector<Ponto\*> getCurva();

void aplicaOperacao();

void renderCatmullRomCurve();

string toString();

};

A única alteração em termos de algoritmos na classe foi a forma como realizamos a translação. Caso o tempo de translação seja 0 então é considerado que a translação é simples, como na fase anterior. Caso não o seja, então geramos os pontos de translação como o valor da curva de *Catmull-Rom* consoante o tempo passado (desde o início da execução) e depois aplicamos a translação consoante a posição dada pela fórmula. Por forma a aplicar a rotação segunda a derivada, temos também uma função que a aplica, invocando-a com o argumento da derivada recebida pela função que executa a fórmula das curvas de *Catmull-Rom*. Para além disso, desenhamos também a curva através da função *renderCatmullRomCurve()*

void Translacao::aplicaOperacao(){

if(tempo==0){

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

}

else{

renderCatmullRomCurve();

float tempoDecorrido = glutGet(GLUT\_ELAPSED\_TIME);

float percentagem = (int) tempoDecorrido % (int) (tempo \* 1000);

percentagem = percentagem / (tempo \* 1000);

float pos[3];

float deriv[3];

getGlobalCatmullRomPoint(percentagem,pos,deriv);

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

rodaElemento(deriv);

}

}

Para desenhar a curva, apenas chamamos a função que nos dá os pontos da curva, caso estes ainda não tenham sido calculados (daí o tamanho ser 0) e depois para cada ponto desenhamos esse ponto. Usamos a opção **GL\_LINE\_LOOP** para poder desenhar a curva.

void Translacao::renderCatmullRomCurve() {

// desenhar a curva usando segmentos de reta - GL\_LINE\_LOOP

if(curva.size()==0){

getCurva();

}

glBegin(GL\_LINE\_LOOP);

for(int i = 0; i < curva.size(); i++){

Ponto\* p = curva.at(i);

glVertex3f(p->getX(),p->getY(),p->getZ());

}

glEnd();

}

O algoritmo de seleção do tempo e da função de *Catmull-Rom* será detalhada numa seção mais à frente.