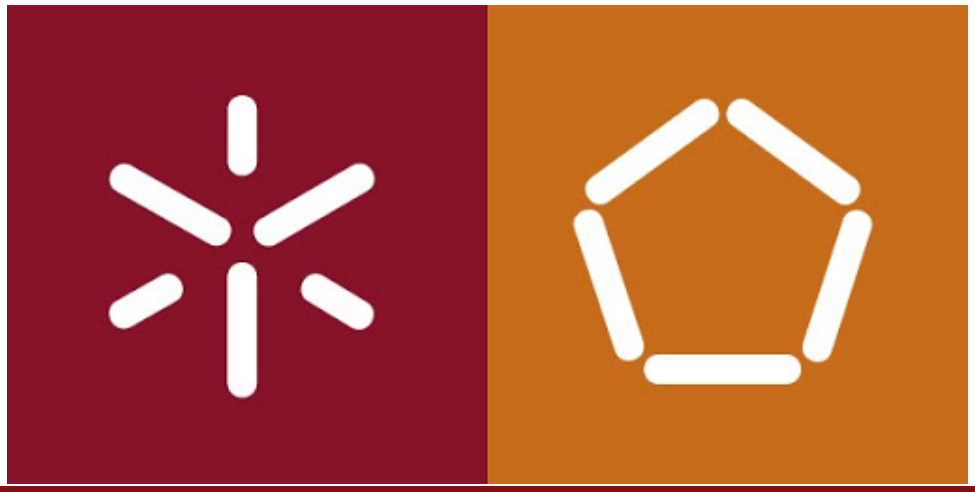

Computação Grafica
Universidade do Minho

Henrique Paz (a84372), João Queiros (a82422), José Santos (a84288), Pedro Gomes (a84220)

23 de Julho de 2020

Conteúdo

1 Introdução

Para a terceira fase do trabalho, foi necessária a alteração da geração e criação dos vários modelos, de modo a permitir o uso de VBO's, sendo esta uma forma mais rápida e eficiente de os gerar.

Foi ainda necessário alterar algumas das estruturas das tranformações geométricas, nomeadamente da translação e da rotação, para permitir a criação de animações recorrendo a curvas Catmull-Rom.

2 Generator

Nesta fase foi necessário atualizar a forma como os vértices das várias figuras eram gerados, de modo a permitir o uso de VBO's, sendo que era necessário gerar os vértices sem repetições, assim como gerar os índices de cada vértice. Assim, foi também alterada a estrutura do ficheiro gerado, sendo que, para além dos vértices, o ficheiro contém ainda os índices, que são colocados depois dos vértices.

2.1 Plano

Para gerar o plano com o centro na origem dividimos os lados por 2 e assim obtivemos os valores das coordenadas x e z,visto que y é sempre constante,0. Depois é so ir mudando os sinais da variavel x e z dos 4 vertoces e ir desenhando na ordem correta tendo em conta o sentido da mao direita. De seguida escrevemos os indices que relacionam esses mesmos vertices.

2.2 Caixa

Para obter a caixa com o centro na origem sao utiliazdos 4 parametros, comprimento, altura,lagura e o numero de divisoes.

Deste modo usamos 2 ciclos for que tem em conta o numero de divisoes e consuante a face em questao escrevem para ficheiro os pontos dos vertices.Repetimos o processo para as restantes 5 faces da caixa sendo que em cada uma o calculo do x,y,z variavel consuante a face em questao

2.3 Cone

Para gerar o cone a estrategia que usamos foi baseado no numero de slices e de stacks. Em primeiro lugar calculamos o vertice que está no centro da base.Depois para cada slice o numero de vertices calculado depende do numero de stacks que é dada como imput.

Em relaçao aos indices o algoritmo que usamos foi um que nos permite interligar vertices da slice inferior com a superior.Para os triangulos da base do cone os indices sao calculados para cada slice.

2.4 Esfera

Para obter os vertices da esfera usamos um algoritmo que usa dois ciclos um para as stacks da esfera e outro para as slices em que calcula os pontos com base nos parametros alpha e beta que sao calculados com base na slice/stack em que se encontra. Os indices sao calculados tambem com dois ciclos um para as stacks e outro para as slices e sao guardados no ficheiro depois dos vertices.

2.5 Patches de Bezier

De maneira a conseguirmos gerar um modelo a partir dos patches de bezier é necessario processar o ficheiro com os pontos de controlo,teapot.patch, esse mesmo ficheiro tem uma estrutura propria em que contem um numero na primeira linha que indica o numero de patches e no fim dos patches outro numero que indica a quantidade de pontos de controlo no ficheiro. Visto isto decidimos criar uma estrutura para

armazenar tal informaçao quando fazemos o parse do ficheiro.

```
struct bezier {
std::vector<Point> * pontos;
std::vector<int> * indices;
int numOfPatches;
int numOfCtrPoints;
};
```

Esta estrutura é composta por inteiro,numOfPatches, que corresponde ao numero na primeira linha do ficheiro, outro inteiro,numOfCtrPoints, que corresponde ao numero de pontos de controlo e depois 2 vetores um de pontos e outro de inteiros que correspondem aos vectores que vao armazenar os pontos de controlo e os indices respetivamente. Esta estrutura é preenchida na funçao initParser que percorre o ficheiro todo e popula a estrutura e da retorna da mesma para ser possivel no futuro gerar o modelo de bezier.

Para gerar o modelo de bezier é usada a funçao mkBezierModel que a partir da estrutura ja populada anteriomente gera em ficheiro os pontos. Para isso o algoritmo usado consiste em ciclos, um que é em relaçao ao tamanho do vector de indices .O primeiro incrementa 16 de cada vez visto que existe em cada patch 16 indices. De seguida sao copiados os componentes de cada vertice no patch e posteriomente sao calculados os numeros de vertices que vao fazer parte da superficie.Depois de ter os vertices calculados calculamos os indices que relacionam os vertices para podermos desenhar a figura.

3 Engine

3.1 Estruturas de Dados

Para permitir a inclusão de animações, assim como a criação de modelos com VBO's, foi necessário alteral algumas das estruturas existentes, de modo a incluir a nova informação necessária

3.2 Modelo

No modelo alteramos a estrutura do mesmo para incorporar o desenho das figuras com VBOs. Mudamos o tipo de dados do vector de vertices para um vector de floats visto que fica mais pratico porque quando preparamos o buffer do VBO de vertices ele necessita de um vector de floats. Alem disso adicionamos um vector de indices para guardar os indices. Ambos estes vetores sao preenchidos no inicio do programa quando fazemos parse do ficheiro xml do sistema solar. De seguida os buffers,iBuff e vBuff, que sao usados para desenhar nos VBOs sao preparados na funçao fillALLbuff() que prepara os buffers dos indices e dos vertices de todos os modelos e assim ficam prontos a desenhar.

```
typedef struct Model {
    vector<float> vertexes;
    vector<unsigned int> indices;
    GLuint vBuff[1];
    GLuint iBuff[1];
} model;
```

3.2.1 Translação

A estrutura que guarda as informações para realizar translações foi atualizada, de modo a conter, para além de três floats correspondentes a translação estática se for o caso disso, um vetor de pontos que corresponde ao pontos chave para a definicao da curva de Catmull-Rom. Para alem disso tem tambem um float que corresponde à duração da trajetória, assim como um vetor OldY que guarda as coordenadas do vetor Y.

No caso de ser uma translaçao dinamica esta é feita com as curvas de catmull, em que para calcular o ponto que no dado instante o modelo se encontra usamos a funçao getCurvePoint que dado o instante de tempo, calcula a posicao e a derivada do modelo. De modo a obtermos o instante de tempo atual usamos glutGet(GLUT_ELAPSED_TIME) que depois é divido pelo tempo que extraimos do ficheiro xml na translaçao.

```
typedef struct translation{
    float x;
    float y;
    float z;
    vector<Vertex>* pontos;
    float time;
    float* oldY;
} *Translation;
```

3.2.2 Rotação

Na rotaçao foi adicinada uma variavel do tempo que representa o valor que o modelo vai demorar a fazer uma rotaçao sobre o eixo indicado. A rotaçao dasse pela expressao glRotatef(time * 360/ r->angle, getX(r->o), getY(r->o), getZ(r->o)) em que r é a estrutura da rotaçao, e o r->angle representa o numero retirado do xml e o time o tempo no dado instante.

```
typedef struct rotation {
float angle;
Operation3f o;
}*Rotation;
```

4 Sistema Solar

Nesta fase, todos os planetas do sistema solar possuem a sua própria órbita em torno do sol, sendo que também foi adicionado ao mesmo um cometa com a forma do teapot em que a sua orbita consiste em andar entre marte e jupiter.

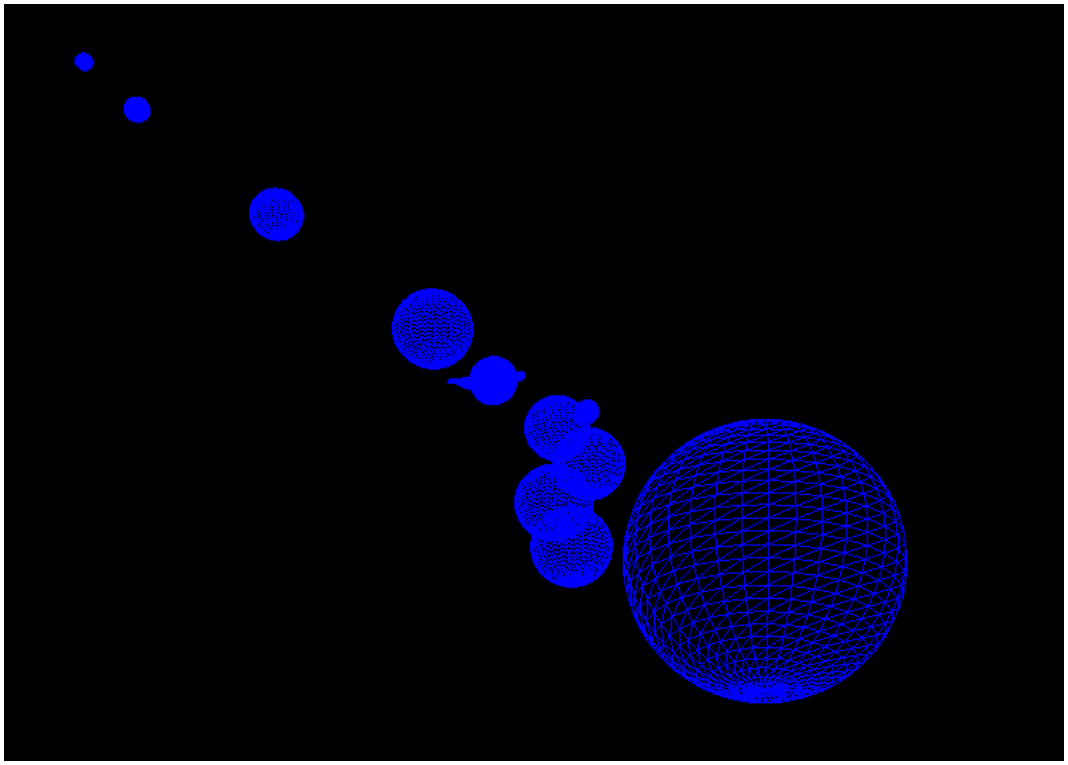

Figura 1: Sistema solar com a lua e cometa teapot

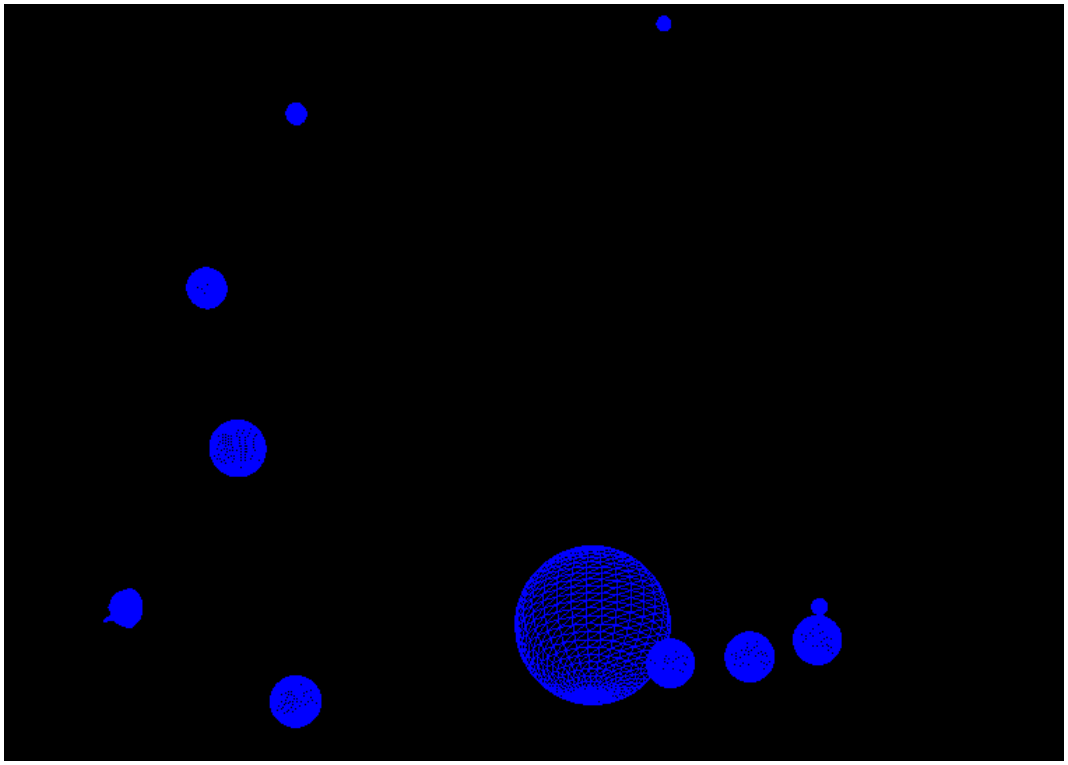

Figura 2: Sistema solar com a lua e cometa teapot