Universidade do Minho

Mestrado Integrado em Engenharia Informática

Computação Gráfica

2ª Fase

Grupo 17

Daniela Fernandes	José Gomes	Ricardo Costa	Tiago Rodrigues
A73768	A82418	A85851	A84276

4 de abril de 2021

Conteúdo

1 Introdução

Este documento, elaborado para a segunda fase do Projeto Prático na unidade curricular de Computação Gráfica, serve para retratar todas decisões tomadas para a resolução do problema desenvolvido em C/C++.

Para a vertente prática da unidade curricular utiliza-se como base a utilização do *OpenGL*, recorrendo à biblioteca *GLUT*, para a construção de modelos 3D.

Nesta fase foi-nos pedido a criação de *hierarchical scenes*, tendo em conta as Transformações Geométricas. Para isto, foi-nos pedido um sistema solar estático.

2 Descrição do problema

Para esta fase foi-nos dado como objetivo o desenvolvimento de novas funcionalidades aplicadas ao Motor criado na fase anterior. Deve processar transformações geométricas, como translações, escalas e rotações, descritas no ficheiro XML(ficheiro de configuração). O ficheiro deve apresentar um modelo estático do sistema solar, com o sol, planetas e satélites.

Os requisitos básicos desta fase são os seguintes:

- **Gerador:** tem como parâmetros os tipos de primitiva gráfica, parâmetros relativos ao modelo e o nome do ficheiro onde vão ser guardados os vértices do modelo:
  - ring in_radius out_radius slices file.3d : é o anel que se encontra no plano xz, centrado na origem e desenhado no sentido positivo e negativo do y.
- **Motor 3D:** lê de um ficheiro XML que contêm os modelos das primitivas, carrega e armazena os vértices dos modelos em memória. Depois de carregados os ficheiros, desenha, utilizando triângulos, as primitivas dos modelos
- **Modelo Sistema Solar:** modelo estático que representa o sistema solar.
  - para a representação do sol, planetas e satélites vai utilizar a esfera e o anel como primitivas.
  - tem uma escala calculada previamente para servir como base nas transformações geométricas a ser aplicadas.

3 Resolução

Para a resolução do problema vamos utilizar o Gerador e Motor criado na fase anterior mas vão ser modificados para cumprir os requisitos pretendidos.

3.1 Gerador

Para esta fase e para desenhar corretamente o sistema solar, foi necessário atualizar o Gerador da fase anterior. Para isso, criamos uma primitiva que calcula os pontos necessários para desenhar um anel (coroa circular) que é necessário para o planeta Saturno.

3.1.1 Anel

Para desenhar o anel são necessários triângulos desenhados no sentido positivo e negativo do eixo dos Y. Esta primitiva vai estar centrada na origem.

O anel vai ser composto por um raio interior, que depois de aplicado a *scale* deverá ser ligeiramente maior que o raio da esfera que representa Saturno, e por um raio exterior maior que o raio interior. Estes raios são denominados respetivamente por *in_radius* e *out_radius*. Além disso vai precisar do número de divisões que vão ser feitas, as *slices*, semelhante ao que se já tinha feito na fase anterior para as primitivas da esfera e do cone.

Os pontos que vão constituir o anel vão ser gerados através de coordenadas polares. Cada *slice* vai ser constituída por 2 triângulos que vão formar um trapézio (dado que devido ao raio interior ser menor os 2 pontos do interior de uma *slice* vão estar mais próximos que os 2 pontos exteriores). Assim, para cada *slice* irão ser gerados 4 pontos através das coordenadas polares que são os 4 pontos de um trapézio. O conjunto dos vários trapézios irá formar um objeto semelhante a um anel.

Para o anel foi necessário, tal como para o plano na fase anterior, desenhar os 2 triângulos de cada *slice* duas vezes, 2 para a parte de cima e 2 para a parte de baixo. Para isto é apenas necessário inverter a ordem pela qual os pontos são escritos para o ficheiro, de acordo com a regra da mão direita. Na figura em baixo é possível ver os triângulos a formar um anel:

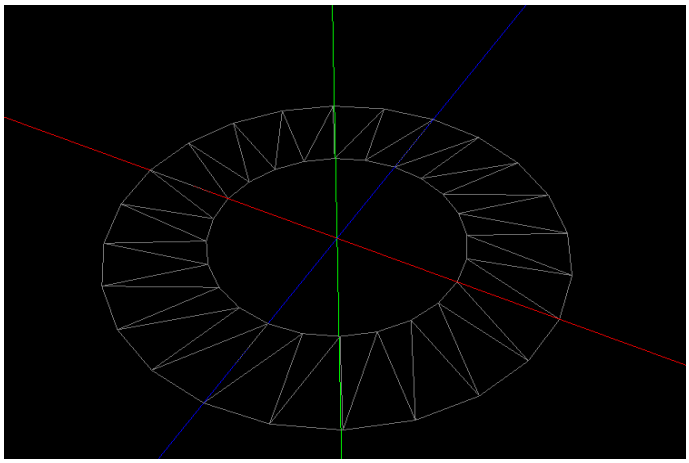

Figura 1: Anel na aplicação engine com 24 *slices* centrado na origem

3.2 Motor 3D

Para esta fase teve que haver uma adaptação porque o ficheiro XML é diferente e por isso fez-se um adaptação para a leitura do mesmo.

Também se teve que armazenar mais informação além dos vértices e dos modelos, para isso teve que se criar novas estruturas de dados.

A estrutra *Geometric_Transf* representa o conjunto de transformações: translação, rotação e escala que podem ser aplicadas a um grupo.

De seguida, criou-se a estrutura *Group* que está relacionada com os grupos existentes no ficheiro XML. Nesta estrutura encontra-se uma instância da estrutura *Geometric_Transf* e 3 vetores: para os modelos, para os sub-grupos de um grupo e para o conjunto dos vértices dos modelos presentes no vetor *models* de um grupo.

```
struct Point {
    float x;
    float y;
    float z;
};

struct Geometric_Transf{
    float t_x, t_y, t_z;            //Translate
    float angle, r_x, r_y, r_z;     //Rotate
    float s_x, s_y, s_z;            //Scale
};

struct Group{
    Geometric_Transf transform;
    vector<string> models;
```

```
    vector<Group> subgroups;
    vector<Point> vertices;
};
```

3.2.1 Leitura do Ficheiro XML

Na segunda fase do projeto vamos continuar a usar a blibioteca *TinyXML-2* que serve para realizar o *parsing* do documento. Contrariamente ao que foi realizado na fase anterior, que consistia apenas em descobrir que modelos desenhar, nesta fase é necessário compreender a constituição de cada grupo que está descrito no ficheiro XML que vai ser passado como argumento ao Motor 3D.

Para realizar a leitura do documento, foi implementada uma função ***load_XML_file*** que vai guardar todas as informações que vão ser lidas de todos os grupos num vetor : *vector<Group> global_groups*. Para isto foram implementadas algumas funções que vão auxiliar este processo:

- **load_Groups:** é chamada por cada grupo encontrado através da função *load_XML_file* para fazer o seu *parsing*. Isto corresponde a preencher os componentes de um grupo através do uso de outras funções que vão ser descritas a seguir e, além disso, a dar *parse* de todos os sub-grupos presentes no grupo (chamando a função recursivamente) e a inseri-los no vetor *subgroups*.

- **init_Geometric_Transf** inicializa os valores da estrutura *Geometric_Transf* relativas a um grupo (todos a 0 exceto os valores relativos à escala que são inicializados a 1).

- **load_Transforms** faz *parsing* das tranformações geométricas: *translate*, *rotate* e *scale*, inserindo os valores respetivos, caso existam, na instância *Geometric_Transf* de um grupo.

- **load_Models** faz *parsing* dos modelos que constituem um grupo e armazena-os no vetor *models* do respetivo grupo.

3.2.2 Preenchimento dos vetores *vertices* de cada grupo

Depois de feita a leitura do ficheiro XML e inserido no vetor *global_groups* as informações presentes no ficheiro, ou seja, as transformações geométricas que deverão ser aplicadas, os modelos presentes e os sub-grupos de cada grupo, falta ainda preencher os vetores *vertices* de cada grupo e respetivos sub-grupos. Para isso são utilizadas duas funções, uma delas a mesma da fase anterior: ***load_model_files*** e ***load_model_file_aux***.

A função **load_model_files** vai percorrer os grupos todos presentes no vetor *global_groups* e para cada grupo vai percorrer o seu vetor *models*. Para cada modelo vai chamar a função

auxiliar ***load_model_file_aux*** que vai percorrer todas as linhas do ficheiro ".3d" correspondente ao modelo e vai inserir os pontos no vetor *vertices* do respetivo grupo. Além disto, a função ***load_model_files*** é também chamada recursivamente para cada grupo do vetor global dos grupos para realizar o mesmo processo para todos os sub-grupos possivelmente presentes num dado grupo.

3.2.3 Desenho das primitivas

Para o desenho das primitivas foi criada uma função **draw_Groups** que tem como argumento um vetor de grupos *vector_groups*. A função é chamada pela primeira vez com o vetor *global_groups* onde se encontram os grupos todos do ficheiro XML.

A função vai começar por percorrer esse vetor, e para cada grupo encontrado vai fazer *push* das tranformações geométricas encontradas. Para fazer isto vai ter de utilizar as funções **glPushMatrix**, **glTranslatef**, **glRotatef** e **glScalef**.

De seguida vai desenhar os triângulos necessários para cada modelo relativos aos pontos que estão guardados no vetor *vertices*.

Por fim, vai-se invocar novamente a função **draw_Groups** recursivamente para os sub-grupos, fazendo com que os sub-grupos herdem as tranformações do grupo em que estão e de seguida a função **glPopMatrix** para estas transformações serem anuladas. Assim as transformações de um grupo não influenciam os outros grupos.

Por fim esta função, juntamente com a função *draw_referencial* (já presente na fase anterior) são chamadas na *renderScene*.

3.2.4 Câmara

Nesta segunda fase, foi implementada uma câmara que se direciona segundo ângulos às posições horizontais e verticais na janela.

Para isto, utilizamos a função **gluLookAt** que nos permite definir três parâmetros que são a posição da câmara, a posição do ponto que a câmara está a focar e direção do vetor.

```
gluLookAt(px, py, pz,
              px + dx, py + dy, pz + dz,
              0.0f,1.0f,0.0f);
```

Como dito anteriormente, as variáves *px*, *py* e *pz* indicam a posição em que a câmara se encontra, já as variáveis *dx*, *dy* e *dz* indicam a direção do vetor, isto é, a direção do "olhar" e também do movimento da câmara, se estivermos a nos movimentar para a frente ou o inverso se estivermos a nos movimentar para trás. Daí o segundo argumento da função *gluLookAt* (que corresponde ao ponto para o qual a câmara está a olhar) ser *px + dx, py + dy, pz + dz*,

já que isto representa a posição atual da câmara mais o vetor "d" que representa a direção para a qual estámos a olhar, resultado por isso no ponto para o qual estámos a olhar.

Inicialmente definiram-se os seguintes valores para as variáveis:

```
float alpha = -2.7, beta = -0.7;      // ângulos que definem a direção do olhar
float px = 10, py = 20, pz = 20;      // posição da câmara
float dx, dy, dz;                     // vetor direção do olhar
```

Dados os ângulos α e β vamos concluir que a direção do olhar da câmara, ou seja o vetor 'd', vai ser calculada da seguinte forma (coordenadas esféricas sem o raio):

```
void sphericalCartesian() {
    dx = cos(beta) * sin(alpha);
    dy = sin(beta);
    dz = cos(beta) * cos(alpha);
}
```

Não é necessário utilizar o raio neste caso, já que apenas a direção importa e não a distância (é como se o raio fosse 1). Assim o vetor "d" é um vetor unitário. Os valores destes ângulos são alterados da seguinte forma:

```
    case GLUT_KEY_LEFT :
        alpha += 0.05;
        break;

    case GLUT_KEY_RIGHT :
        alpha -= 0.05;
        break;

    case GLUT_KEY_UP :
        if (beta <= 1.5f)
            beta += 0.05f;
        break;

    case GLUT_KEY_DOWN :
        if (beta >= -1.5f)
            beta -= 0.05f;
        break;
```

Para conseguirmos movimentar a câmara para direita e para a esquerda, vamos criar um vetor *rigth* que resulta do produto externo entre o vetor direção e o vetor up(0,1,0) (que usando a regra da mão direita resulta num vetor a apontar para a direita da direção do vetor "d") que é calculado na seguinte função:

```
void right_vector(){
    right_x = -dz;
    right_y = 0;
    right_z = dx;
}
```

Para se encontrar uma nova posição do ponto *px*, *py* e *pz* , utiliza-se o vetor "d" (para a frente e trás) ou o vetor *right* (para a esquerda e direita) e um valor "dist" que é a distância que a câmara se movimenta com cada clique da tecla:

```
        case 'w' :
            px += (dx * dist);
            py += (dy * dist);
            pz += (dz * dist);
            break;

        case 's' :
            px -= (dx * dist);
            py -= (dy * dist);
            pz -= (dz * dist);
            break;

        case 'a' :
            right_vector();
            px -= (right_x * dist);
            py -= (right_y * dist);
            pz -= (right_z * dist);
            break;

        case 'd' :
            right_vector();
            px += (right_x * dist);
            py += (right_y * dist);
            pz += (right_z * dist);
            break;
```

Com estas fórmulas consegue-se mover a câmara em qualquer direção, de acordo com o vetor "d" e o *right*.

3.2.5 Teclado

Como foi dito anteriormente, a câmara tem a possibilidade de se movimentar com a utilização da função **gluLookAt**. A posição da câmara e do ponto de referência são definidos pelas variáveis *px*, *py*, *pz*, *dx*, *dy* e *dz*.

Posto isto, para alterar a posição da câmara foram utilizadas as teclas 'w', 's', 'a' e 'd'. Cada vez que a tecla 'w' for premida, faz com que a câmara se movimente *dist* unidades de acordo com a direção 'd', enquanto que se for premida a tecla 's', a câmara vai movimentar-se *dist* unidades contrariamente a essa mesma direção. Para movimentar perpendicularmente à direção 'd' são utilizadas as teclas 'a' e 'd' (esquerda e direita, respetivamente). É utilizada a função **regular_keys** para definir estes movimentos, como foi apresentado no tópico da câmara anterior.

Para além destas foram criadas algumas diretrizes que foram especificadas anteriormente, *UP*, *DOWN*, *LEFT* e *RIGHT*, que permitem alterar a direção para a qual a câmara está a olhar, alterando os ângulos *alpha* e *beta*. Isto é feito na função **special_keys**.

3.3 Modelo Sistema Solar

No ficheiro XML que criamos acerca do sistema solar, temos representados o Sol, Mercúrio, Vénus, Terra e Lua, Marte e Fobos, Júpiter e Europa, Saturno (e anel) e Titã, Urano e Titânia, Neptuno e Tritão. Representamos portanto, o Sol, os 8 planetas e os Satélites mais importantes dos planetas que possuem satélites.

Para a representação do modelo utilizamos a esfera e o anel.

Como as primitivas que vamos utilizar (esfera e anel) possuem uma dimensão e posição fixas, vamos ter que utilizar transformações geométricas para conseguirmos representar o sistema corretamente. As primitivas que vamos utilizar são, a translação, rotação e escala. Foi também necessário utilizarmos uma escala para mantermos as dimensões dos elementos do sistema o mais realistas possíveis, apesar de haver uma grande diferença entre os elementos.

Tal como falamos anteriormente, os subgrupos herdam as características dos grupos e portanto, temos que ter em atenção que vão herdar as transformações que lhes sejam aplicadas.

- A translação de um grupo vai ser somado há translação do seu subgrupo
- A escala do grupo vai ser multiplicada há escala do subgrupo
- O valor da escala de um grupo multiplica com a translação do subgrupo
- A rotação do grupo vai ser somada há rotação do seu subgrupo

3.3.1 Rotações

Para podermos simular a inclinação dos planetas utilizamos a transformação geométrica rotação e de acordo com os dados da tabela abaixo, foram aplicadas as seguintes rotações aos planetas. De notar que a rotação de um grupo é herdada pelo sub-grupo, logo tivemos isso em consideração relativamente às luas dos planetas, e que as rotações foram efetuadas relativamente ao eixo dos 'Z' de acordo com a inclinação dos planetas. Como este ângulo é positivo para o lado esquerdo, é necessário aplicar uma rotação inversa do valor da inclinação. Na tabela seguinte estão representados os valores utilizados para as rotações:

Tabela 1: Rotações para Planeta/Satélite do modelo.

Nome	Inclinação(graus)	*Rotate*
Mercúrio	0.1	-0.1
Vénus	1.77	-177
Terra	23.5	-23.5
Lua	5.15	18.35
Marte	25	-25
Fobos	1	24
Júpiter	3	-3
Europa	0.2	2.8
Saturno	27	-27
Titã	0.33	26.67
Urano	98	-98
Titânia	0.14	97.86
Neptuno	30	-30
Tritão	157.35	-127.35

3.3.2 Escalas

Relativamente à escala tivemos por base o valor do raio dos planetas e da distância do último planeta ao Sol para conseguirmos calcular uma escala. Devido às dimensões dos planetas ficarem tão desajustadas (uns demasiado grandes, outros demasiado pequenos) foram feitas alterações à mão a esta escala para se melhor poder visualizar o Sistema Solar, tentando manter minimamente aproximado à realidade. Não esquecer a escala aplicada a um planeta é também aplicada aos seus satélites, já que se encontram dentro do grupo do planeta, por isso foi necessário ter isto em consideração nas escalas dos satélites. Os valores utilizados estão representados em baixo:

Tabela 2: Escalas para Estrela/Planeta/Satélite do modelo.

Nome	Raio(km)	Escala	*Scale*
Sol	69 600 000	3.10165	8
Mercúrio	2 440	0.000109	0.22
Vénus	6 052	0.00027	0.54
Terra	6371	0.000284	0.57
Lua	1 737	0.000077	0.35
Marte	3 390	0.000151	0.35
Fobos	11.1	0.000000494	0.2
Júpiter	69 911	0.0031	2.4
Europa	2 634	0.000117	0.1
Saturno	58 232	0.002595	2
Titã	2 575	0.000115	0.12
Urano	25 362	0.00113	1.2
Titânia	789	0.000035	0.15
Neptuno	24 622	0.001097	1.1
Tritão	1 353	0.00006	0.2

3.3.3 Translação

Relativamente às translações, foi considerado uma distância máxima de 150 (relativa ao último planeta do Sol) e tendo em conta os valores reais das distâncias dos planetas foi criada uma escala. Esta escala teve de ser alterada "à mão" porque o Sistema Solar não ficava percetível. Não esquecer que os valores das rotações e escalas aplicados ao planetas também afetam os seus satélites (além dos das translações), logo foi necessário ter isto em consideração nas translações dos satélites. Os valores finais usados nas translações encontram-se na tabela seguinte:

Tabela 3: Translações para Estrela/Planeta/Satélite do modelo.

	Distância (km)	Escala	*Translate*
Mercúrio-Sol	56 847 190	2.53	X=2.5
Vénus-Sol	107 710 466	4.8	X=5
Terra-Sol	149 597 870	6.67	X=9
Lua-Terra	394 400	0.017	X=-0.6485 Y=1.96
Marte-Sol	227 388 763	10.13	X=15
Fobos-Marte	9 380	0.0004	X=-0.82 Y=1.85
Júpiter-Sol	777 908 927	34.67	X=40
Europa-Júpiter	1 070 400	0.477	X=-0.16 Y=2.43
Saturno-Sol	1 421 179 771	63.3	X=70
Titã-Saturno	1 221 870	0.0545	X=1.8 Z=1.8
Urano-Sol	2 872 279 117	127.99	X=110
Titânia-Urano	436 300	0.0194	X=-2.01 Y=-0.28
Neptuno-Sol	4 487 936 121	200	X=150
Tritão-Neptuno	354 800	0.0158	X=1.62 Y=-1.23

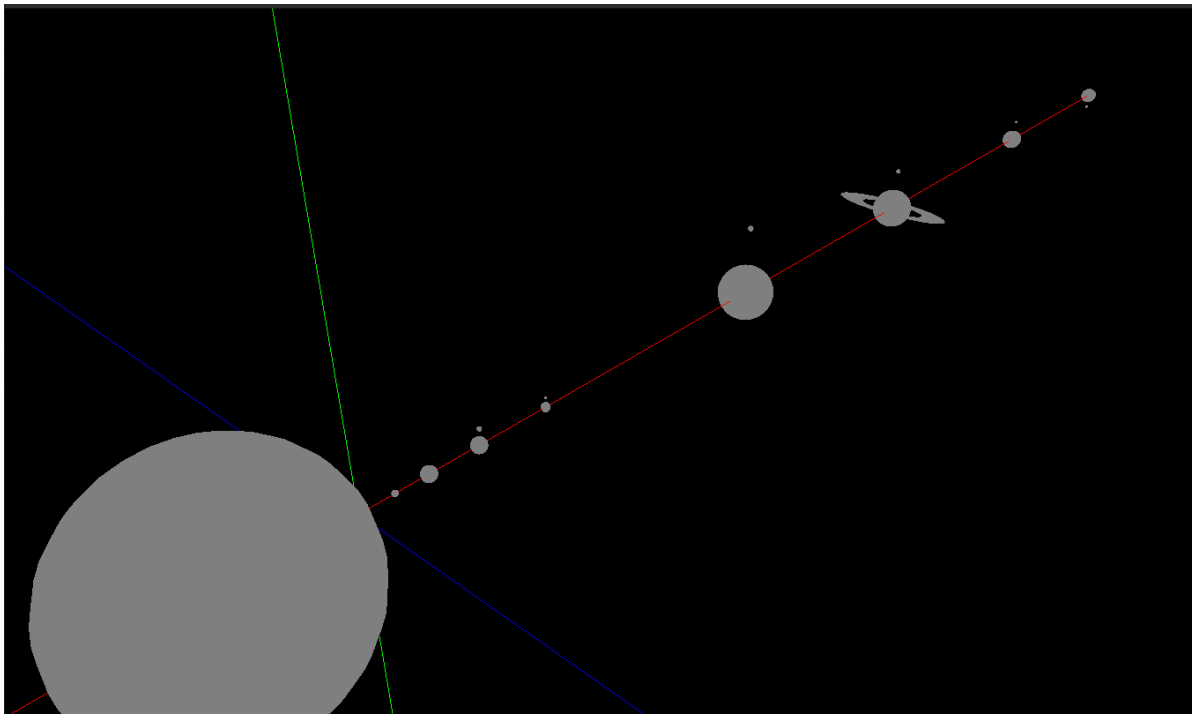

Figura 2: Modelo estático do Sistema Solar

4 Conclusão

Consideramos que esta segunda fase foi bem sucedida e que conseguimos cumprir todos os requisitos propostos inicialmente.

Nesta fase, conseguimos implementar uma câmara para ver o modelo elaborado de diferentes pontos. Além disto, conseguimos movimentar a câmara apartir do teclado.

Concluímos que a resolução desta fase foi importante para a consolidação dos temas abordados nas aulas práticas e teóricas da Unidade Curricular. Neste projeto abordamos essencialmente as tranformações geométricas, translação, rotação e escala, e também o *parsing* do ficheiro XML .