Universidade do Minho

Licenciatura Engenharia Informática
Computação Gráfica
Galaxy3D

2 de Junho de 2014

Cláudia Oliveira
60987

Duarte Duarte
61001

Hélder Gonçalves
61084

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O presente trabalho enquadra-se na unidade curricular (UC) de Computação Gráfica, inserida no plano de estudos de Licenciatura Engenharia Informática da Universidade do Minho. Todos os conteúdos que vão ser apresentados refletem a maioria dos conhecimentos abordados ao longo do semestre, com especial referência as práticas laboratoriais.

O tema do projeto a desenvolver, encontra-se definido como sendo um motor 3D, onde este desenhe objetos em 3D lidos de ficheiros que lhe são fornecidos. Este exemplo é desenvolvido usando a linguagem C++ e a API OpenGL, de modo a criar uma representação gráfica 3D.

1.1 Objetivos

Com a realização deste projeto existem várias vertentes da computação gráfica que vão ser abordadas, sendo os principais objetivos definidos os seguintes:

- Desenvolver construções geométricas de primitivas e objetos básicos;
- Aplicar transformações geométricas para construir cenários 3D e visualizar os mesmos em tempo real;
- Definir luzes, matérias e normais para os objetos 3D;
- Aplicar texturas 2D sobre os objetos;

  Adicionalmente destacam-se outros objetivos que visam aprofundar os conhecimentos:

- Uso de técnicas de aceleração da renderização através de Vertex Buffer Objects(VBO's);
- Optimização com recurso a diferentes técnicas de culling, de que é exemplo o view frustum culling;

- Profiling e análise de desempenho;
- Picking.

1.2 Estrtura do Relatório

A estrutura que este relatório segue, excluindo o presente capitulo, é um capítulo sobre enquadramento dos assuntos adquiridos nesta UC (cap. 2), onde é feita uma breve descrição acerca do que é o OpenGL.

Após esse capitulo temos o (cap .3) que serve para que o leitor possa ter contexto acerca do projeto e do que é pedido para a realização do mesmo.

Passados o (cap. 2) e (cap. 3), temos então que começa a descrição propriamente deita de como é a implementação feita para cada componente necessário (cap. 4).

O (cap. 11.1) e (cap. 5) são referentes à implementação da fase 3. Por fim temos o (cap. 7) que apresenta os conceitos eleborados na fase 4.

A finalizar temos o (cap. 10) onde apresentamos algumas considerações relativas à elaboração do projeto. Temos também o (cap. 11), que possui o objetivo de consulta, para melhor consolidação de como foram implementados alguns aspetos importantes.

Capítulo 2

Enquadramento do Assunto Abordados Nesta UC

Inicialmente tinha sido referido que a implementação do motor recorria à API OpenGL[1] e à linguagem de programação C/C++. Para além "aplicativos" descritos foram também usados alguns toolkits e bibliotecas para OpenGL, como sendo:

- GLUT – que permite uma API para uma manipulação simples de janelas;
- GLEW – que fornece mecanismos de execução eficientes para determinar quais as extensões OpenGL são suportadas na plataforma destino. Por exemplo as VBO's para serem suportadas em OpenGL precisam do GLEW;
- IL – biblioteca da DeIL para o tratamento de carregar e guarda imagens.

O OpenGL utiliza o sistema de eixo descritos na figura 1 e a regra da "mão direita".

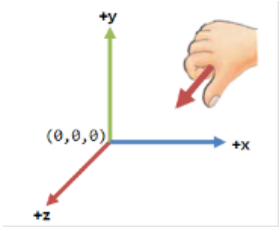

Figura 2.1: Sistema de eixos do OpenGL

Um outro aspeto que é importante referir é o facto de o OpenGL implementar uma máquina de estados, isto é, quando as definições vão alterando

[1]Pode ser consultada no site http//www.opengl.org/

o seu estado vão guardando o mesmo até este ser de novo alterado. Esta definição pode ser vista por exemplo quando falamos das cores, pois quando uma cor é definida (glColor3f()) todos os objetos depois dessa função serão desenhados com a cor definida até que uma nova cor seja definida.

Capítulo 3

Descrição do Trabalho Prático

O projeto que os alunos têm de desenvolver encontra-se divido por fases:

3.1 Fase 1 - Primitivas gráficas simples

Nesta primeira fase foi onde o motor3D foi desenvolvido. É no motor que estão definidos os módulos cpp que são relativos à leitura do ficheiros XML, as definições das câmaras que são utilizadas e main do projeto em si. O facto de este motor ser implementado deve-se que este motor mais tarde é usado para desenhar modelos 3D que são lidos de ficheiros.

Também era necessário a criação de um aplicativo que cria-se os ficheiros que mais tarde são lidos pelo motor 3D. Os objetivos que eram definidos para esta fase era a geração de:

- Áreas planas;
- Paralelepípedo;
- Esferas;
- Cones.

3.2 Fase 2 - Transformações Geométricas

Nesta segunda fase era necessário a alteração dos ficheiros para que estes incluíssem as transformações que podem ser aplicadas ao modelo 3D, como por exemplo ser possível guardar a translação de um objeto.

3.3 Fase 3 - Transformações Geométricas

Na terceira fase foi quando começamos a implementação das superfícies de Bézier. As superfícies eram geradas a partir da leitura de um ficheiro onde se encontravam os pontos de controlo e o grau de translação pretendido, dando origem a um ficheiro com lista de pontos que será usado pelas curvas.

A implementação das curvas tinham como principal objetivo definir a trajetória que os planetas, onde era necessário a noção de tempo para em vez de definir o ângulo de rotação era definido o tempo necessário para um rotação de 360º.

3.4 Fase 4 - Normais e Coordenadas de Textura

Nesta fase final do projeto encontra-se relacionada com os últimos temas abordados na UC, que são iluminação e texturas.

Relativamente ao projeto em desenvolvimento é necessário que aplicar luz ao sistema solar onde os pontos de iluminação são previamente gerados e guardados no XML, e mais tarde lidos.

As texturas são armazenadas de modo a que quando o ficheiro é lido saber o tipo de textura que o objeto vai ter.

Capítulo 4

Primitivas Geométricas

4.1 Primitivas Simples e Transformações Geométricas

Tínhamos como objetivo desenvolver primitivas simples, onde estas eram criadas pela geração de um ficheiros xml através de um módulo cpp.

Uma das preocupações que tivemos relativas a esta fase foi o facto fazermos o desenho das figuras ser o mais detalhado e abrangente possível.

As transformações geométricas foi definido para podermos ter rotações e translações nas primitivas simples.

Vamos agora apresentar os novos módulos cpp para a criação dos ficheiros .3d relativos a cada primitiva, que sustentam rotações e translações.

4.1.1 Anel

A implementação do anel que vamos explicar é a maneira imediata de criar um anel, ou seja temos que os pontos são sempre calculados quando são necessário . Temos que a função recebe os parâmetros: raio_fora, raio_dentro, fatias, aneis, ori e file.

O ficheiro de saída é o nome do ficheiro que mais tarde vai ser referido para leitura pelo motor3d. As fatias e as camadas são para definir o número de divisões verticais de triângulos que o anel vai ter. O fato de termos dois raios serve para determinarmos os círculos do anel, isto é o raio_fora é a distancia que vai desde a origem até ao anel mais distante, o raio_dentro é a distância que vai desde a origem até ao anel mais inteiror.

Figura 4.1: Diferentes anulos de vista do anel

```
1  // Anel para o modo imediato
2  void anel(float raio_fora, float raio_dentro,int fatias, int aneis, int ori, FILE
       * f){
3      float angulo=(2*M_PI)/fatias,x,y=0,l_aux,raio=(raio_fora-raio_dentro)/aneis,
           alt=0;
4
5      //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
6      fprintf(f, "%f %f %d %d %f %f\n",raio_fora,-raio_fora,0,0,raio_fora,-
           raio_fora);
7
8      //Imprime o numero de pontos que vao ser utilizados
9      fprintf(f,"%d\n",2*fatias*(aneis)*9);
10     raio_fora=raio_dentro;
11     if(ori){
12         for(;aneis>0;aneis--){
13             raio_dentro=raio_fora;
14             raio_fora+=raio;
15
16             for(l_aux=0;l_aux<fatias;l_aux++){
17                 x=y;
18                 y+=angulo;
19                 fprintf(f,"%f %f %f\n",raio_dentro*sin(x), alt, raio_dentro*cos(x
                       ));
20                 fprintf(f,"%f %f %f\n",raio_fora*sin(x), alt, raio_fora*cos(x));
21                 fprintf(f,"%f %f %f\n",raio_dentro*sin(y), alt, raio_dentro*cos(y
                       ));
22
23                 fprintf(f,"%f %f %f\n",raio_dentro*sin(y), alt, raio_dentro*cos(y
                       ));
24                 fprintf(f,"%f %f %f\n",raio_fora*sin(x), alt, raio_fora*cos(x));
25                 fprintf(f,"%f %f %f\n",raio_fora*sin(y), alt, raio_fora*cos(y));
26             }
27         }
28     }else{
29         for(;aneis>0;aneis--){
30             raio_dentro=raio_fora;
31             raio_fora+=raio;
32
33             for(l_aux=0;l_aux<fatias;l_aux++){
34                 x=y;
35                 y+=angulo;
36                 fprintf(f,"%f %f %f\n",raio_dentro*sin(x), alt, raio_dentro*cos(x
                       ));
37                 fprintf(f,"%f %f %f\n",raio_dentro*sin(y), alt, raio_dentro*cos(y
                       ));
38                 fprintf(f,"%f %f %f\n",raio_fora*sin(x), alt, raio_fora*cos(x));
39
40                 fprintf(f,"%f %f %f\n",raio_dentro*sin(y), alt, raio_dentro*cos(y
                       ));
41                 fprintf(f,"%f %f %f\n",raio_fora*sin(y), alt, raio_fora*cos(y));
42                 fprintf(f,"%f %f %f\n",raio_fora*sin(x), alt, raio_fora*cos(x));
43             }
44         }
45     }
46 }
```

Listing 4.1: Código de implementação do ficheiro3d anel

4.1.2 Cilindro

A implementação do cilindro é feita da seguinte maneira, temos que os parâmetros que o cilindro recebe são: raio, fatias, camadas, altura, anéis e ficheiro de saída.

O ficheiro de saída é o nome do ficheiro que mais tarde vai ser referido para leitura pelo motor3d. As fatias e as camadas são para definir o número de divisões verticais de triângulos que o cilindro vai ter, as camadas serve para dividir os cilindro na vertical.

A nossa implementação do cilindro passa por definir as respetivas base e o corpo do cilindro, para definir os respetivos pontos têm-se ciclos que são

efetuado o número de camadas vezes e que esse valores sejam escritos no respetivo ficheiro.

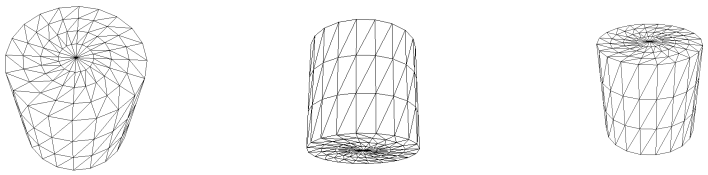

Figura 4.2: Angulos de visão de um circulo lido pelo motor3D do ficheiro circulo.3d

```
void cilindro(float raio,int fatias,int camadas,float altura,int aneis, FILE* f){
    float angulo=(2*M_PI)/fatias,x,y=0,l_aux,h_aux1,h_aux2=0, r_aux1,r_aux2;
    int aneis_aux=aneis-1;

    //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
    fprintf(f, "%f %f %f %d %f %f\n",raio, -raio,altura,0,raio,-raio);

    raio=raio/aneis;


    fprintf(f,"%d\n",(2*fatias*(aneis-1)+fatias)*9*2 + 2*fatias*camadas*9);

    for(l_aux=0;l_aux<fatias;l_aux++){
            x=y;
            y+=angulo;
            fprintf(f,"%d %f %d\n",0, altura, 0);
            fprintf(f,"%f %f %f\n",raio*sin(x), altura, raio*cos(x));
            fprintf(f,"%f %f %f\n",raio*sin(y), altura, raio*cos(y));
        }
        r_aux2=raio;
        y=0;
        for(aneis--;aneis>0;aneis--){
            r_aux1=r_aux2;
            r_aux2+=raio;

           for(l_aux=0;l_aux<fatias;l_aux++){
               x=y;
               y+=angulo;
                fprintf(f,"%f %f %f\n",r_aux1*sin(x), altura, r_aux1*cos(x));
                fprintf(f,"%f %f %f\n",r_aux2*sin(x), altura, r_aux2*cos(x));
                fprintf(f,"%f %f %f\n",r_aux1*sin(y), altura, r_aux1*cos(y));

                fprintf(f,"%f %f %f\n",r_aux1*sin(y), altura, r_aux1*cos(y));
                fprintf(f,"%f %f %f\n",r_aux2*sin(x), altura, r_aux2*cos(x));
                fprintf(f,"%f %f %f\n",r_aux2*sin(y), altura, r_aux2*cos(y));
            }
        }
    r_aux2=raio;
    y=0;
        for(l_aux=0;l_aux<fatias;l_aux++){
            x=y;
            y+=angulo;
            fprintf(f,"%d %d %d\n",0, 0, 0);
            fprintf(f,"%f %f %f\n",raio*sin(y), 0.0, raio*cos(y));
            fprintf(f,"%f %f %f\n",raio*sin(x), 0.0, raio*cos(x));
        }


        for(aneis=aneis_aux;aneis>0;aneis--){
            r_aux1=r_aux2;
            r_aux2+=raio;

            for(l_aux=0;l_aux<fatias;l_aux++){
                x=y;
                y+=angulo;
                fprintf(f,"%f %f %f\n",r_aux1*sin(x), 0.0, r_aux1*cos(x));
                fprintf(f,"%f %f %f\n",r_aux1*sin(y), 0.0, r_aux1*cos(y));
                fprintf(f,"%f %f %f\n",r_aux2*sin(x), 0.0, r_aux2*cos(x));

                fprintf(f,"%f %f %f\n",r_aux1*sin(y), 0.0, r_aux1*cos(y));
                fprintf(f,"%f %f %f\n",r_aux2*sin(y), 0.0, r_aux2*cos(y));
```

```
62                  fprintf(f,"%f %f %f\n",r_aux2*sin(x), 0.0, r_aux2*cos(x));
63              }
64          }
65
66      raio=r_aux2;
67
68      //Corpo
69      altura=altura/camadas;
70      for(;camadas>0;camadas--){
71          h_aux1=h_aux2;
72          h_aux2+=altura;
73          y=0;
74          for (l_aux=0; l_aux<fatias; l_aux++) {
75              x=y;
76              y+=angulo;
77
78              fprintf(f,"%f %f %f\n",raio*sin(x), h_aux2, raio*cos(x));
79              fprintf(f,"%f %f %f\n",raio*sin(x), h_aux1, raio*cos(x));
80              fprintf(f,"%f %f %f\n",raio*sin(y), h_aux2, raio*cos(y));
81
82              fprintf(f,"%f %f %f\n",raio*sin(x), h_aux1, raio*cos(x));
83              fprintf(f,"%f %f %f\n",raio*sin(y), h_aux1, raio*cos(y));
84              fprintf(f,"%f %f %f\n",raio*sin(y), h_aux2, raio*cos(y));
85          }
86      }
87  }
```

Listing 4.2: Código de implementação do ficheiro3d cilindro

4.1.3 Circulo

A implementação do ficheiro que quando mais tarde lido dá origem a um circulo recebe como parâmetros os seguintes: raio, fatias, anéis altura, origem e ficheiro.

O File *f é o onde o nome que o ficheiro de saída com os pontos do circulo á passado como argumento.

O raio é o raio que o circulo vai ter desde a origem até ao ponto mais afastado, as fatias é o número de vezes que nós estamos a dividir o circulo, os anéis é o número de vezes em que pequenos círculos que são criados desde a origem até atingir o raio, o ori serve para indicar se o circulo se encontra no topo ou na base.

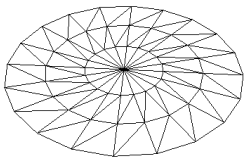

Figura 4.3: Exemplo de um circulo lido pelo motor3d do ficheiro circulo.3d

```
1  //ORI --- 1 -> BASE && 0 -> TOPO
2
3  void circulo(float raio, int fatias,int aneis, float altura,int ori, FILE* f){
4      float angulo=(2*M_PI)/fatias,x,y=0,l_aux, r_aux1,r_aux2;
5
6
7      //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
8      fprintf(f, "%f %f %f %f %f %f\n",raio, -raio,altura,altura,raio,-raio);
9
10     fprintf(f,"%d\n",(2*fatias*(aneis-1)+fatias)*9);
11     raio=raio/aneis;
12     if(ori){
13         for(l_aux=0;l_aux<fatias;l_aux++){
14           x=y;
15             y+=angulo;
16             fprintf(f,"%d %f %d\n",0, altura, 0);
17             fprintf(f,"%f %f %f\n",raio*sin(x), altura, raio*cos(x));
18             fprintf(f,"%f %f %f\n",raio*sin(y), altura, raio*cos(y));
19         }
20         r_aux2=raio;
21         y=0;
22         for(aneis--;aneis>0;aneis--){
23             r_aux1=r_aux2;
24             r_aux2+=raio;
25
26            for(l_aux=0;l_aux<fatias;l_aux++){
27                x=y;
28                y+=angulo;
29                 fprintf(f,"%f %f %f\n",r_aux1*sin(x), altura, r_aux1*cos(x));
30                 fprintf(f,"%f %f %f\n",r_aux2*sin(x), altura, r_aux2*cos(x));
31                 fprintf(f,"%f %f %f\n",r_aux1*sin(y), altura, r_aux1*cos(y));
32
33                 fprintf(f,"%f %f %f\n",r_aux1*sin(y), altura, r_aux1*cos(y));
34                 fprintf(f,"%f %f %f\n",r_aux2*sin(x), altura, r_aux2*cos(x));
35                 fprintf(f,"%f %f %f\n",r_aux2*sin(y), altura, r_aux2*cos(y));
36             }
37         }
38     }else{
39         for(l_aux=0;l_aux<fatias;l_aux++){
40           x=y;
41             y+=angulo;
42             fprintf(f,"%d %f %d\n",0, altura, 0);
43             fprintf(f,"%f %f %f\n",raio*sin(y), altura, raio*cos(y));
44             fprintf(f,"%f %f %f\n",raio*sin(x), altura, raio*cos(x));
45         }
46         r_aux2=raio;
47         y=0;
48         for(aneis--;aneis>0;aneis--){
49             r_aux1=r_aux2;
50             r_aux2+=raio;
51
52             for(l_aux=0;l_aux<fatias;l_aux++){
53                 x=y;
54                 y+=angulo;
55                 fprintf(f,"%f %f %f\n",r_aux1*sin(x), altura, r_aux1*cos(x));
56                 fprintf(f,"%f %f %f\n",r_aux1*sin(y), altura, r_aux1*cos(y));
57                 fprintf(f,"%f %f %f\n",r_aux2*sin(x), altura, r_aux2*cos(x));
58
59                 fprintf(f,"%f %f %f\n",r_aux1*sin(y), altura, r_aux1*cos(y));
60                 fprintf(f,"%f %f %f\n",r_aux2*sin(y), altura, r_aux2*cos(y));
61                 fprintf(f,"%f %f %f\n",r_aux2*sin(x), altura, r_aux2*cos(x));
62             }
63         }
64     }
65 }
```

Listing 4.3: Código que mais atrde gera o ficherio qye dará origem ao circulo

4.1.4 Cone

A implementação do cone passa por termos um circulo e depois termos a altura toda a convergir para o mesmo ponto. Na implementação do cpp que gera o ficheiro relativo ao cone temos que este recebe como parâmetros os seguintes campos:raio_base, altura, camadas, fatias, anéis e o *f;

O FILE *f onde passamos o nome que o ficheiro que vai pelo motor criar o cone. Os restantes parâmetros veem ao encontro do que temos vindo a descrever, onde o raio_base é o raio que o circulo do cone vai ter como raio, as fatias e os anéis são usados para a chamada do cpp do circulo com os parâmetros que este necessita. Mas também temos que as fatias e as camadas são usadas para definir quantos triângulos é que vamos ter a definir na altura.

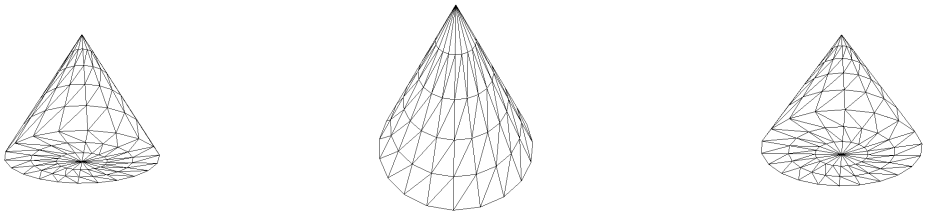

Figura 4.4: Angulos de visão de um cone lido pelo motor3D do ficheiro cone.3d

```
1  void cone(float raio_base, float altura, int fatias, int aneis, int camadas, FILE
       * f){
2      float angulo=(2*M_PI)/fatias,x,y=0,l_aux,  alt_aux1,alt_aux2=0,r_aux1,r_aux2=
           raio_base;
3
4      //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
5      fprintf(f, "%f %f %f %d %f %f\n",raio_base, -raio_base,altura,0,raio_base,-
           raio_base);
6
7
8      float i=1,factor_h=(i/camadas);
9      float raio = raio_base/aneis;
10     altura/=camadas;
11
12     fprintf(f,"%d\n",(2*fatias*(aneis-1)+fatias)*9+((camadas-1)*fatias*2+fatias)
           *9);
13
14
15     for(l_aux=0;l_aux<fatias;l_aux++){
16             x=y;
17             y+=angulo;
18             fprintf(f,"%d %f %d\n",0, 0.0, 0);
19             fprintf(f,"%f %f %f\n",raio*sin(y), 0.0, raio*cos(y));
20             fprintf(f,"%f %f %f\n",raio*sin(x), 0.0, raio*cos(x));
21         }
22         r_aux2=raio;
23         y=0;
24         for(aneis--;aneis>0;aneis--){
25             r_aux1=r_aux2;
26             r_aux2+=raio;
27
28             for(l_aux=0;l_aux<fatias;l_aux++){
29                 x=y;
30                 y+=angulo;
31                 fprintf(f,"%f %f %f\n",r_aux1*sin(x), 0.0, r_aux1*cos(x));
32                 fprintf(f,"%f %f %f\n",r_aux1*sin(y), 0.0, r_aux1*cos(y));
33                 fprintf(f,"%f %f %f\n",r_aux2*sin(x), 0.0, r_aux2*cos(x));
34
```

```
35                  fprintf(f,"%f %f %f\n",r_aux1*sin(y), 0.0, r_aux1*cos(y));
36                  fprintf(f,"%f %f %f\n",r_aux2*sin(y), 0.0, r_aux2*cos(y));
37                  fprintf(f,"%f %f %f\n",r_aux2*sin(x), 0.0, r_aux2*cos(x));
38              }
39          }
40
41
42      for(;camadas>1;camadas--){
43          alt_aux1=alt_aux2;
44          alt_aux2+=altura;
45          y=0;
46          i-=factor_h;
47          r_aux1=r_aux2;
48          r_aux2=raio_base *i;
49
50          for (l_aux=0; l_aux<fatias; l_aux++) {
51              x=y;
52              y+=angulo;
53              fprintf(f,"%f %f %f\n",r_aux2*sin(x), alt_aux2, r_aux2*cos(x));
54              fprintf(f,"%f %f %f\n",r_aux1*sin(x), alt_aux1, r_aux1*cos(x));
55              fprintf(f,"%f %f %f\n",r_aux2*sin(y), alt_aux2, r_aux2*cos(y));
56
57              fprintf(f,"%f %f %f\n",r_aux1*sin(x), alt_aux1, r_aux1*cos(x));
58              fprintf(f,"%f %f %f\n",r_aux1*sin(y), alt_aux1, r_aux1*cos(y));
59              fprintf(f,"%f %f %f\n",r_aux2*sin(y), alt_aux2, r_aux2*cos(y));
60          }
61      }
62
63      alt_aux1=alt_aux2;
64      alt_aux2+=altura;
65      y=0;
66      i-=factor_h;
67      r_aux1=r_aux2;
68      r_aux2=raio_base *i;
69
70      for (l_aux=0; l_aux<fatias; l_aux++) {
71          x=y;
72          y+=angulo;
73          fprintf(f,"%f %f %f\n",r_aux1*sin(x), alt_aux1, r_aux1*cos(x));
74          fprintf(f,"%f %f %f\n",r_aux1*sin(y), alt_aux1, r_aux1*cos(y));
75          fprintf(f,"%f %f %f\n",r_aux2*sin(y), alt_aux2, r_aux2*cos(y));
76      }
77  }
```

Listing 4.4: Código que escreve os pontos de um cone em ficheiro

4.1.5 Esfera

O desenho de uma esfera é criado tendo em contas as chamadas coordenadas polares do cilindro mas no entanto temos que à media que vamos desenhando as camadas e as fatias o raio que é usado a cada iteração é mais pequeno. Os parâmetros que são passados ao modulo que cria a esfera são: raio, camadas e fatias o nome do ficheiro de saída, onde o raio é usado para definir o tamanho máximo da esfera as camadas o número de divisórias horizontais que vão existir e as fatias as divisórias verticais.

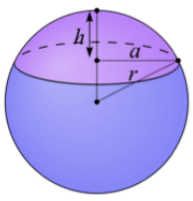

Figura 4.5: Definição do raio de uma secção da esfera a uma altura h

Figura 4.6: Angulos de visão de uma esfera lida pelo motor3D do ficheiro esfera.3d

```
#include "esfera.h"
void esfera(float raio, int camadas, int fatias, FILE* f){
    float   angulo_cir=(2*M_PI)/fatias,
            angulo_h=(M_PI)/camadas,
            x,y,l_aux,lh_aux,h_aux1,h_aux2=M_PI_2;
    h_aux1=h_aux2;
    h_aux2+=angulo_h;
    y=0;
    fprintf(f,"%d\n",2*fatias*(camadas-1)*9);

    for (l_aux=0; l_aux<fatias; l_aux++) {
        x=y;
        y+=angulo_cir;
        fprintf(f,"%f %f %f\n",raio*sin(x)*cos(h_aux2), raio*sin(h_aux2), raio*
            cos(x)*cos(h_aux2));
        fprintf(f,"%f %f %f\n",raio*sin(y)*cos(h_aux2), raio*sin(h_aux2), raio*
            cos(y)*cos(h_aux2));
        fprintf(f,"%f %f %f\n",raio*sin(y)*cos(h_aux1), raio*sin(h_aux1), raio*
            cos(y)*cos(h_aux1));
    }

    for(lh_aux=1;lh_aux<camadas-1;lh_aux++){
        h_aux1=h_aux2;
        h_aux2+=angulo_h;
        y=0;

        for (l_aux=0; l_aux<fatias; l_aux++) {
            x=y;
            y+=angulo_cir;
            fprintf(f,"%f %f %f\n",raio*sin(x)*cos(h_aux1), raio*sin(h_aux1),
                raio*cos(x)*cos(h_aux1));
            fprintf(f,"%f %f %f\n",raio*sin(x)*cos(h_aux2), raio*sin(h_aux2),
                raio*cos(x)*cos(h_aux2));
            fprintf(f,"%f %f %f\n",raio*sin(y)*cos(h_aux1), raio*sin(h_aux1),
                raio*cos(y)*cos(h_aux1));

            fprintf(f,"%f %f %f\n",raio*sin(x)*cos(h_aux2), raio*sin(h_aux2),
                raio*cos(x)*cos(h_aux2));
            fprintf(f,"%f %f %f\n",raio*sin(y)*cos(h_aux2), raio*sin(h_aux2),
                raio*cos(y)*cos(h_aux2));
            fprintf(f,"%f %f %f\n",raio*sin(y)*cos(h_aux1), raio*sin(h_aux1),
                raio*cos(y)*cos(h_aux1));
        }
    }
    h_aux1=h_aux2;
    h_aux2+=angulo_h;
    y=0;

    for (l_aux=0; l_aux<fatias; l_aux++) {
        x=y;
        y+=angulo_cir;
        fprintf(f,"%f %f %f\n",raio*sin(x)*cos(h_aux1), raio*sin(h_aux1), raio*
            cos(x)*cos(h_aux1));
        fprintf(f,"%f %f %f\n",raio*sin(x)*cos(h_aux2), raio*sin(h_aux2), raio*
            cos(x)*cos(h_aux2));
        fprintf(f,"%f %f %f\n",raio*sin(y)*cos(h_aux1), raio*sin(h_aux1), raio*
            cos(y)*cos(h_aux1));
    }

}
```

Listing 4.5: Código que escreve os pontos de uma esfera em ficheiro

4.1.6 Plano

A maneira como nós implementamos o plano foi que primeiro testamos o em que plano XoY— XoZ— ZoY e se estávamos na parte positiva ou negativa do plano.

Os parâmetro que passamos para a função do plano, onde esta depois vai escrever o ficheiro a ler, são: altura, lado, camadas, z_index, ori, e file. Têm-se então que o a altura e o lado definem respetivamente o altura e o lado do plano, temos a ori, que nos permite saber em que plano para onde vão se escritos os pontos, as fatias e a camadas para sabermos o par de triângulos necessários e o file onde é passado o ficheiro que vai conter tais pontos.

Figura 4.7: Diferentes angulos de visão do plano

```
1
2  void plano(float altura, float lado, int camadas, int fatias, float z_index, int
       ori, FILE* f, int flag){
3      int i;
4      float l_const=lado/fatias, alt_const=altura/camadas,alt_ori=-altura/2,
           lado_ori=-lado/2;
5
6      switch (ori) {
7          case 1:
8              if (flag==1){
9                  //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                       ViewFrustumCulling
10                 fprintf(f, "%f %f %f %f %f %f\n",lado/2.0f,-lado/2.0f,altura/2.0f
                       , -altura/2.0f,z_index, z_index);
11
12                 fprintf(f,"%d\n",2*camadas*fatias*9);
13             }
14             for(altura=alt_ori;camadas>0;camadas--){
15                 i=0;
16                 for(lado=lado_ori;i<fatias;i++){
17
18                     fprintf(f,"%f %f %f\n",lado,altura, z_index);
19                     fprintf(f,"%f %f %f\n",lado+l_const,altura, z_index);
20                     fprintf(f,"%f %f %f\n",lado,altura+alt_const, z_index);
21
22                     fprintf(f,"%f %f %f\n",lado+l_const,altura, z_index);
23                     fprintf(f,"%f %f %f\n",lado+l_const,altura+alt_const, z_index
                           );
24                     fprintf(f,"%f %f %f\n",lado,altura+alt_const, z_index);
25                     lado+=l_const;
26                 }
27                 altura+=alt_const;
28             }
29             break;
30         case 2:
31             if (flag==1){
32                 //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                       ViewFrustumCulling
33                 fprintf(f, "%f %f %f %f %f %f\n",lado/2.0f,-lado/2.0f,altura/2.0f
                       , -altura/2.0f,z_index, z_index);
34
35                 fprintf(f,"%d\n",2*camadas*fatias*9);
36             }
```

```
37              for(altura=alt_ori;camadas>0;camadas--){
38                  i=0;
39                  for(lado=lado_ori;i<fatias;i++){
40
41                      fprintf(f,"%f %f %f\n",lado,altura, z_index);
42                      fprintf(f,"%f %f %f\n",lado,altura+alt_const, z_index);
43                      fprintf(f,"%f %f %f\n",lado+l_const,altura, z_index);
44
45                      fprintf(f,"%f %f %f\n",lado+l_const,altura, z_index);
46                      fprintf(f,"%f %f %f\n",lado,altura+alt_const, z_index);
47                      fprintf(f,"%f %f %f\n",lado+l_const,altura+alt_const, z_index
                            );
48                      lado+=l_const;
49                  }
50                  altura+=alt_const;
51              }
52              break;
53          case 3:
54              if (flag==1){
55                  //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                        ViewFrustumCulling
56                  fprintf(f, "%f %f %f %f %f %f\n",z_index, z_index,altura/2.0f, -
                        altura/2.0f,lado/2.0f,-lado/2.0f);
57
58                  fprintf(f,"%d\n",2*camadas*fatias*9);
59              }
60              for(altura=alt_ori;camadas>0;camadas--){
61                  i=0;
62                  for(lado=lado_ori;i<fatias;i++){
63
64                      fprintf(f,"%f %f %f\n",z_index,altura, lado);
65                      fprintf(f,"%f %f %f\n",z_index,altura+alt_const, lado);
66                      fprintf(f,"%f %f %f\n",z_index,altura, lado+l_const);
67
68                      fprintf(f,"%f %f %f\n",z_index,altura, lado+l_const);
69                      fprintf(f,"%f %f %f\n",z_index,altura+alt_const, lado);
70                      fprintf(f,"%f %f %f\n",z_index,altura+alt_const, lado+l_const
                            );
71                      lado+=l_const;
72                  }
73                  altura+=alt_const;
74              }
75              break;
76          case 4:
77              if (flag==1){
78                  //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                        ViewFrustumCulling
79                  fprintf(f, "%f %f %f %f %f %f\n",z_index, z_index,altura/2.0f, -
                        altura/2.0f,lado/2.0f,-lado/2.0f);
80
81                  fprintf(f,"%d\n",2*camadas*fatias*9);
82              }
83              for(altura=alt_ori;camadas>0;camadas--){
84                  i=0;
85                  for(lado=lado_ori;i<fatias;i++){
86
87                      fprintf(f,"%f %f %f\n",z_index,altura, lado);
88                      fprintf(f,"%f %f %f\n",z_index,altura, lado+l_const);
89                      fprintf(f,"%f %f %f\n",z_index,altura+alt_const, lado);
90
91                      fprintf(f,"%f %f %f\n",z_index,altura,lado+l_const);
92                      fprintf(f,"%f %f %f\n",z_index,altura+alt_const, lado+l_const
                            );
93                      fprintf(f,"%f %f %f\n",z_index,altura+alt_const, lado);
94                      lado+=l_const;
95                  }
96                  altura+=alt_const;
97              }
98              break;
99          case 5:
100             if (flag==1){
101                 //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                        ViewFrustumCulling
102                 fprintf(f, "%f %f %f %f %f %f\n",altura/2.0f, -altura/2.0f,
                        z_index, z_index,lado/2.0f,-lado/2.0f);
103
104                 fprintf(f,"%d\n",2*camadas*fatias*9);
105             }
106             for(altura=alt_ori;camadas>0;camadas--){
107                 i=0;
108                 for(lado=lado_ori;i<fatias;i++){
109
110                     fprintf(f,"%f %f %f\n",altura,z_index, lado);
111                     fprintf(f,"%f %f %f\n",altura,z_index, lado+l_const);
```

```
112                         fprintf(f,"%f %f %f\n",altura+alt_const,z_index, lado);
113
114                         fprintf(f,"%f %f %f\n",altura,z_index,lado+l_const);
115                         fprintf(f,"%f %f %f\n",altura+alt_const,z_index, lado+l_const
                                );
116                         fprintf(f,"%f %f %f\n",altura+alt_const,z_index, lado);
117                         lado+=l_const;
118                     }
119                     altura+=alt_const;
120                 }
121                 break;
122             case 6:
123                 if (flag==1){
124                     //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                            ViewFrustumCulling
125                     fprintf(f, "%f %f %f %f %f %f\n",altura/2.0f, -altura/2.0f,
                            z_index, z_index,lado/2.0f,-lado/2.0f);
126
127                     fprintf(f,"%d\n",2*camadas*fatias*9);
128                 }
129                 for(altura=alt_ori;camadas>0;camadas--){
130                     i=0;
131                     for(lado=lado_ori;i<fatias;i++){
132
133                         fprintf(f,"%f %f %f\n",altura,z_index, lado);
134                         fprintf(f,"%f %f %f\n",altura+alt_const,z_index, lado);
135                         fprintf(f,"%f %f %f\n",altura,z_index, lado+l_const);
136
137                         fprintf(f,"%f %f %f\n",altura,z_index, lado+l_const);
138                         fprintf(f,"%f %f %f\n",altura+alt_const,z_index, lado);
139                         fprintf(f,"%f %f %f\n",altura+alt_const,z_index, lado+l_const
                                );
140                         lado+=l_const;
141                     }
142                     altura+=alt_const;
143                 }
144                 break;
145
146             default:
147                 break;
148     }
149 }
```

Listing 4.6: Códio de criação do ficheiro do plano

4.1.7 Paralelepípedo

A implementação do paralelepípedo depois de termos definido o modulo do plano o paralelepípedo na nossa implementação é a chamada do modulo do plano, mas alterando os pontos de modo a que estes fiquem ligados.

A chamada da função do plano recebe os parâmetros altura, largura o lado z, largura do lado x, camadas, fatias e file. Onde temos que o file é onde é passado o nome, lado_x define a largura definida para esse lado, o lado_z a o comprimento que é definido, as camadas e fatias servem para definir os triângulos que podemos encontrar horizontalmente e verticalmente.

Figura 4.8: Ângulos de visão de um paralelepípedo lido pelo motor3D do ficheiro

```
#include "paralelepipedo.h"

void paralelepipedo(float lado_y, float lado_x, float lado_z, int camadas, int
     fatias, int fatias_z, FILE* f){

  int flag=0;
    //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
    fprintf(f, "%f %f %f %f %f %f\n",lado_x/2.0f,-lado_x/2.0f,lado_y/2.0f,-lado_y
         /2.0f,lado_z/2.0f,-lado_z/2.0f);

  fprintf(f,"%d\n",2*(2*camadas*fatias + 2*fatias_z*camadas + 2*fatias*fatias_z)
       *9);

    plano(lado_y, lado_x, camadas, fatias, lado_z/2, 1,f, flag);
    plano(lado_y, lado_x, camadas, fatias, -lado_z/2, 2,f, flag);

    plano(lado_y, lado_z, camadas, fatias_z, lado_x/2, 3,f, flag);
    plano(lado_y, lado_z, camadas, fatias_z, -lado_x/2, 4,f, flag);

    plano(lado_x, lado_z, fatias, fatias_z, lado_y/2, 5,f, flag);
    plano(lado_x, lado_z, fatias, fatias_z, -lado_y/2, 6,f, flag);
}
```

Listing 4.7: Código que escreve os pontos de um paralelepípedo em ficheiro

4.2 Estrutura de Dados

Era necessário a criação de uma estrutura de dados, que nós desse suporte, para que quando nós estamos a desenhar a primitiva sabermos qual a primitiva a desenhar o nº de pontos necessários para a mesma.

Sendo assim definimos a seguinte estrutura que nos garante os nosso requisitos iniciais.

```
typedef struct sr_time{

    const char* nome;
    float* vertices;
    int n_pontos;

}*RTime, NRTime;
```

Listing 4.8: Estrutura de dados para armazenar as primitivas

4.3 Gerador

Para podermos criar os ficheiros, necessitávamos de um gerador que recebia os parâmetros e que cria-se um ficheiro .3d para a primitiva que lhe fosse passada.

```
if(argc<2)
    printf("ERRO!! Nenhuma 'tag' de desenho detectada!\nTem as seguintes opcoes:\
        n\t-> esfera\n\t-> circulo\n\t-> cilindro\n\t-> anel\n\t-> cone\n\t->
        plano\n\t-> paralelepipedo\n\t-> patch\n");
  else
    if(strcmp(op,"esfera")==0){
      if(argc==6 || argc==7){

        if(sscanf(argv[2],"%f",&p1)&&sscanf(argv[3],"%d",&i1)&&sscanf(argv[4],"%d
            ",&i2)){
          f=fopen(argv[5],"w");
          if(argc==7 && strcmp(argv[6],"-vbo")==0)
            esferaVBO(p1,i1,i2,f);
          else
            esfera(p1,i1,i2,f);
          fclose(f);

        }else
          printf("ERRO!! Parametros nao estao correctos!\nEx: esfera [raio] [
              camadas] [fatias] [output]\n");
      }
      else
        printf("ERRO!! Numero de argumentos errado\nEx: esfera [raio] [camadas] [
            fatias] [output]\n");
    }
```

Listing 4.9: Excerto de código do main.cpp do gerador para uma primitiva

O exemplo que apresentamos é relativo à esfera mas todas as outras primitivas seguem o mesmo raciocínio de implementação.

Abaixo apresentamos como é que o gerador deve ser chamado para cada uma das primitivas e como é que são passados os parâmetros de cada primitiva.

```
Ex: anel [raio_fora] [raio_dentro] [fatias] [aneis] [orientacao] [output]
Ex: esfera [raio] [camadas] [fatias] [output]
Ex: cilindro [raio] [fatias] [camadas] [altura] [aneis] [output]
Ex: circulo [raio] [fatias] [aneis] [orientacao] [output]
Ex: plano [altura] [lado] [camadas] [fatias] [orientacao] [output]
Ex: cone [raio_base] [altura] [fatias] [aneis] [camadas] [output]
Ex: paralelepipedo [lado_Y] [lado_X] [lado_Z] [camadas] [fatias_X] [fatias_Z] [output]
```

Figura 4.9: Exemplo de como o gerador é chamado para cada primitiva

Nota: Caso se queira criar os ficheiros usando VBO's, temos que fazer os mesmo passo que estão na imagem, mas no fim temos de acrescentar a opção -vbo.

Capítulo 5

Transformações Geométricas

Neste capítulo vamos mostrar quais as transformações que podemos encontrar e aplicar a cada um dos obejctos que fazem parte das primitivas ou no sistema solar.

5.1 Escala

Quando falamos em escala significa que estamos a alterar o tamanho do objeto em causa, isto é podemos ter um objeto pequeno e depois esse objeto ficar maior ou ao contrário, ter um objeto maior e depois ficar mais pequeno.

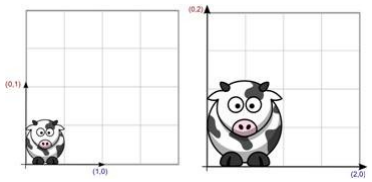

Figura 5.1: Exemplo de aplicação de escala a um objeto

5.1.1 Estrutura de Dados

A maneira que nós usamos para temos o fator de escala foi a criação de uma lista ligada onde vamos armazenar os pontos que mais tarde vão ser usados para definir o tamanho do objeto.

```
1 typedef struct sEscala{
2     float x;
3     float y;
4     float z;
5     struct sEscala *next;
6 }*Escala, NEscala;
```

Listing 5.1: Estrutura de dados escala

5.2 Rotação

Quando estamos a falar do rotação significa que temos um objeto que se encontra virado para uma determinada posição e ao lhe ser aplicado um rotação temos que este fica então virado para uma nova posição.

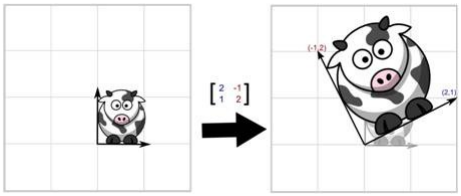

Figura 5.2: Exeplo de como é feita a rotação de um objeto

A nossa implementação de rotação está associada ao movimento de rotação dos planetas em torno do seu próprio eixo. Para termos isso o nossa estrutura de dados, teria de guardar os elementos necessários para a rotação como sendo o ângulo que o objeto sofre, o tempo que o objeto tem associado a sua rotação e a posição do objeto quando lhe é efetuada a rotação. Tivemos o cuidado de definir a estrtura de modo a que esta dê tnato para as rotações normais como as rotações em relação ao tempo. Para sabermos qual a rotação que se está a fazer temos o parametro tempo (normal se 0, 1 se relação ao tempo)

5.2.1 Estrutura de Dados

```
typedef struct sRotacao{
    int  periodo;
    float angulo;
    float x;
    float y;
    float z;
    struct sRotacao *next;
}Rotacao;
```

Listing 5.2: Estrutura de dados rotação

Ao usarmos para a escala e a rotação lista ligadas, temos que ao fazer o parsing ao ficheiro xml e ao apanhar a tag de escala os valores nessa tag são preenchidos nos respetivos campos, ficando assim guardados para que ao desenharmos o planeta sabermos onde é que este tem as suas coordenadas.

5.3 Translação

A translação serve para movimentarmos um objeto de um ponto para outro, isto é, podemos ter um objeto num ponto e ao aplicarmos um transalção este munda para uma posição diferente.

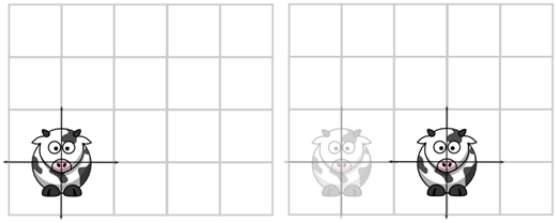

Figura 5.3: Exemplo de aplicação da translação ao objeto

```
typedef struct point {
    float coords[3] = {0.0f, 0.0f,0.0f};
    struct point *next=NULL;
} Point;

typedef struct translacao{
    float res[3];
    int point_count;
    Point *points;
    float tempo;
    float a;
    float lastTime;
    float pX;
    float pY;
    float pZ;
    struct translacao *next;
}Translacao;
```

Listing 5.3: Estrutura de dados catmull

Nós utilizamos a estrutura apresentada pois esta permite-nos armazenar o ponto inicial do planeta e o tempo que este demora para efetuar um rotação em torno do sol.

A distribuição que aplicamos aos pontos é que este se encontram todos espaçados à mesma distância, podendo assim evitar temos situações em que o planeta tem um aceleração maior do que noutros casos.

5.3.1 Catmull

Com o catmull nós definimos para podermo desenhar as linhas, que onde o respetivo planetas vai andar, pois com o catmull e a translação conseguimos definir os movimentos de rotação que os planetas efetuam relativamente ao sol. O catmull lê os pontos que depois vai usar para determinar as orbitas do planetas, e quando uma iteração é feita este define também onde estará o objeto na próxima iteração.

Os pontos que são lidos, são apenas calculados uma vez, depois disso são utilizados através do desenho de VBO's.

5.4 Superficie

Estas superfícies de Bézier [1] são definidas através da interpolação de pontos.

A aplicação da curva no nosso projeto vem ao encontro da definição das orbitas do planetas para depois podermos definir o efeito de rotação de cada planeta em torno do sol.

```
1 #ifndef CG_FORMAS_PRIMARIAS_patch_h
2 #define CG_FORMAS_PRIMARIAS_patch_h
3
4 #include <iostream>
5 #include <math.h>
6
7 void read_Patch(FILE *f_patch, FILE *f, int detail);
8 #endif
```

Listing 5.4: .h da implementação da superfícies

Temos também uma primitiva que nós criamos o nosso "asteroide"que foi criado usando superfícies de Bézier.

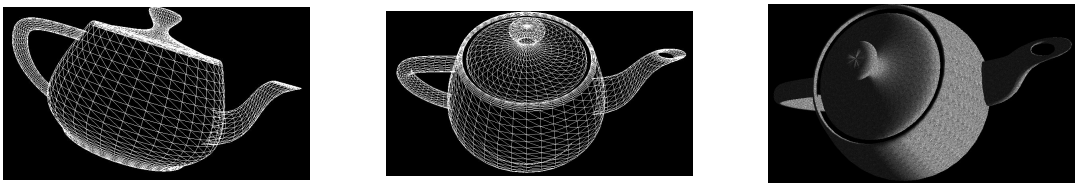

Figura 5.4: Exemplo do "asteroide"usando as superfícies de bézier

Capítulo 6

VBO's

As definições que forma apresentadas anteriormente para as primitivas são definições de modo imediato isto é cada ponto é a passado usando a função do OpenGL *glVertex3f()*. Contudo o OpenGL permite a utilização de buffer de arrays. Nestes podemos organizar os atributos dos vértices (coordenadas, coordenadas de texturas e normais). O procedimento a seguir com VBOs é o seguinte:

1. Ativar buffers (glEnableClientState(GL_VERTEX_ARRAY);
2. Alocar e preencher os arrays;
3. Gerar VBOs;
4. Preparar desenho dos VBOs;
5. Desenhar com VBOs.

6.1 Estrutura de Dados

A estrutura de dados da implementação em VBO's é semelhante à implementação das primitivas imediatas, mas existe uma pequena alteração, pois quando estamos nas primitivas imediatas temos que os pontos que vamos usar e qual a primitiva, em VBO's também temos a primitiva a implementar, mas os pontos são passados em arrays de pontos.

```
1 typedef struct sVbo{
2     const char* nome;
3     GLuint *buffers;
4     unsigned short *indices;
5     int n_indices;
6  }*Vbo, NVbo;
```

Listing 6.1: Estrutura de dados VBO's

Na secção 11.1 do capitulo Anexos pode ser consultado como é que cada primitiva foi implementada.

Capítulo 7

Normais e Coordenadas de Texturas

A definição dos vetores normais, juntamente com as texturas, possuem um fator de grande carga quando em computação gráfica, pois é através desses vetores que conseguimos dar realismo aos objetos.

Quando se fala de iluminação, tem-se então que os vetores de normais são essenciais, pois é através do ângulo entre a direção da luz a a normal que determinamos a intensidade da luz na superfície do objeto.

7.1 Texturas

Nas texturas nós temos duas vertentes, isto é, temos a aplicação de um imagem como textura, mas temos também uma vertente, que é a definição do sistema solar que não é definir as texturas como imagens, mas sim termos um espécie de material, isto é nós temos o sistema solar em que cada planeta possui um material definido por nós.

O carregar de uma textura é feito através da leitura no xml de um campo textura que indica o nome da textura a carregar, abaixo apresentamos a parte do código no ficheiro *motorXML.cpp* responsável pelo carregamento da textura.

```
1
2  }else if (strcmp(attr->Name(), "textura")==0) {
3        //Carregar textura
4        ilEnable(IL_ORIGIN_SET);
5        ilOriginFunc(IL_ORIGIN_LOWER_LEFT);
6        ilGenImages(1,&t);
7        ilBindImage(t);
8        ilLoadImage((ILstring)attr->Value());
9        tw = ilGetInteger(IL_IMAGE_WIDTH);
10       th = ilGetInteger(IL_IMAGE_HEIGHT);
11       ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
12       texData = ilGetData();
13
14       glGenTextures(1,&propModel->texID);
15       glBindTexture(GL_TEXTURE_2D,propModel->texID);
16       glTexParameteri(GL_TEXTURE_2D,   GL_TEXTURE_WRAP_S,     GL_REPEAT);
17       glTexParameteri(GL_TEXTURE_2D,   GL_TEXTURE_WRAP_T,     GL_REPEAT);
18
19       glTexParameteri(GL_TEXTURE_2D,   GL_TEXTURE_MAG_FILTER,     GL_LINEAR);
```

```
20          glTexParameteri(GL_TEXTURE_2D,   GL_TEXTURE_MIN_FILTER,        GL_LINEAR);
21
22          glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, tw, th, 0, GL_RGBA,
                GL_UNSIGNED_BYTE, texData);
23
24      }
```

Listing 7.1: Excerto do código para carregar textura

Vamos agora apresentar a aplicação das texturas a algumas primitivas simples.

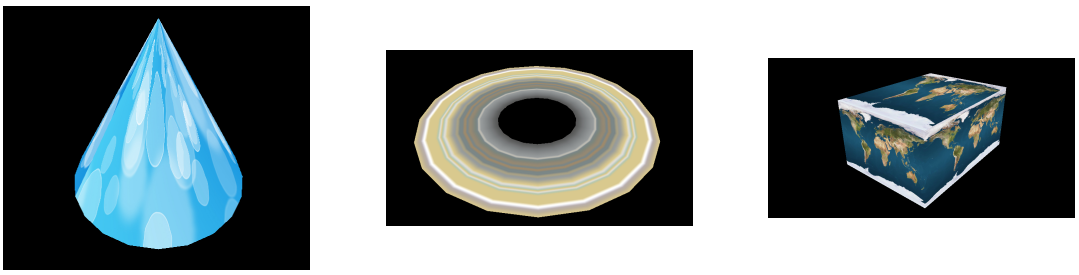

Figura 7.1: Exemplo de texturas aplicadas a primitivas

A implementação das texturas a cada primitiva foi feita da seguinte maneira:

- Anel: Repetimos a imagem em cada fatia;
- Circulo: Expandimos a imagem pelo circulo;
- Cilindro: Temos o mesmo que os circulos para as base, e depois para o corpo expandimos a imagem pelo mesmo;
- Cone: Segue o mesmo algoritmo que o cilindro, só temos de colocar o corpo a expandir para o mesmo ponto;
- Esfera: Neste caso expandimos por toda a esfera;
- Plano: Imprimimos a imagem no plano;
- Paralelipipedo: Aplica o mesmo algoritmo do plano, mas temos que o repetir pelos quator planos;
- Patch: Repete a imagem em várias direções(ideal para dar uma textura uniforme).

Relativamente ao sistema solar nós tivemos o cuidado de arranjar as texturas reais de cada um dos planetas.

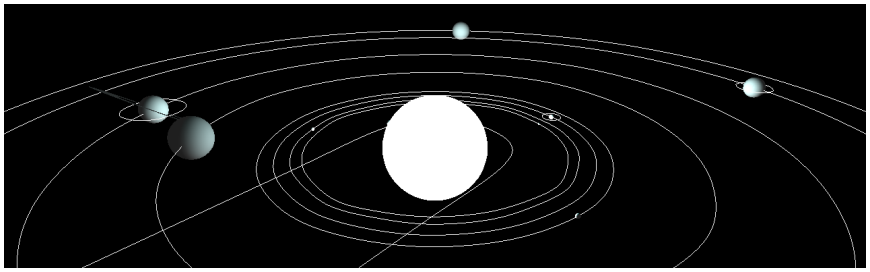

Figura 7.2: Visão do Sistema Solar sem textura

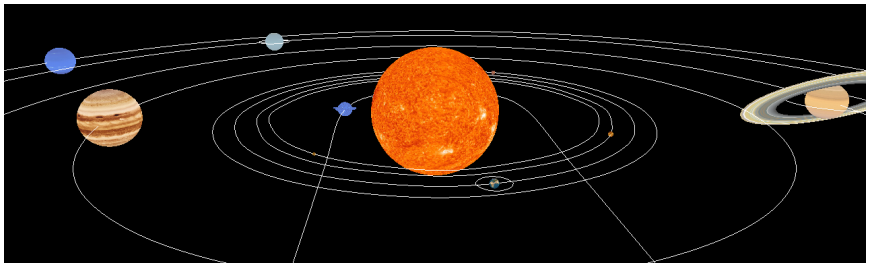

Figura 7.3: Visão do Sistema Solar com Textura

7.2 Normais e Iluminação

As normais, neste caso normais de luz são vetores perpendiculares a uma determinada superfície.

É com os vetores das normais que nós definimos como é que a luz é refletida por um determinado objeto, isto é, com a emissão da luz de determinado ponto é necessário saber se quando esta luz atinge o objeto como é que o ilumina.

Quando falamos de iluminação existem vários tipos de iluminação que podemos usar para iluminar um objeto:

- Difusa;
- Ambiente;
- Especular;
- Emissiva;
- Ambiente e Difusa.

```
1 #ifndef __Motor3D__luzes__
2 #define __Motor3D__luzes__
3 #include <iostream>
4 #include <GLUT/glut.h>
5 #include "tinyxml/tinyxml.h"
6 typedef struct sluz{
7     int luz;
8     int tipo;
```

```
9       float propriedade[4];
10      struct sluz *next;
11
12  }*Luz, NLuz;
13  void preparaLuzes(TiXmlNode* root);
14  void defineLuzes();
15  #endif /* defined(__Motor3D__luzes__) */
```

Listing 7.2: .h da definição das luzes

No nosso caso, fizemos a implementação que permite a um dado objeto ser incidido com um dos tipos de luz, para tal, temos no XMl a tag luz. Na tag luzes nós definimos os tipos de luzes que são aplicadas ao planetas quando este recebem a luz que é emitida pelo sol, pois no nosso sistema solar o sol não é "iluminado", mas sim uma fonte de luz. Para definirmos a luz no sol usamos pontos que se encontram "dentro"da área representada pelo sol no xml.

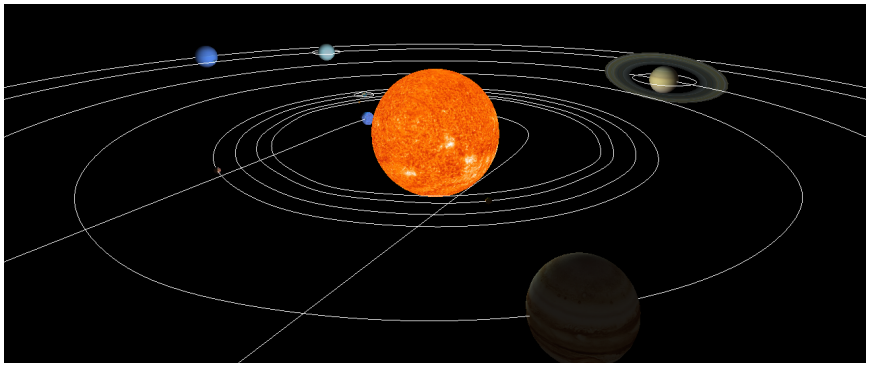

Figura 7.4: Visão de Cima do Sistema Solar com iluminação

Os materiais é algo diferente, pois com os materiais temos que cada modelo possui o seu próprio material. Para isso definimos a seguinte estrtura de dados que nos permite aramazenar as características que cada modelo tem para o material, isto é, para cada material, nós temos de ramazenar o modelo em causa, qual o material associado a esse modelo, qual a cor desse modelo e o valor de picking9.5 associadoa ao modelo.

```
1   typedef struct smodelo{
2       ViewFrustum pontos;
3       short tipo;
4       union{
5           Vbo vbo;
6           RTime rTime;
7       }u;
8       struct smodelo *next;
9   }*Modelo, NModelo;
10  typedef struct sPropModel{
11      Modelo modelo;
12      Material materiais;
13      unsigned int texID;
14      Picking picking;
15      struct sPropModel *next;
16
17  }*PropModel, NPropModel;
```

Listing 7.3: Wstrutura de dados dos materiais

Capítulo 8

Motor3D

8.1 Main

Nesta secção vamos apresentar como é que definimos o main do motor, é o main do motor o responsável por inicializar todas as funções que serão utilizadas.

Quando o main é chamado sem um ficheiro XMl, para leitura este não é executado.

Temos que caso o main sejam chamado como os parâmetros todos direitinhos, então o main é responsável por fazer a inicialização das funções do glut e ativação dos buffer's.

```
1  //Callback do GLEW - Tem de estar depois de todos os callbacks do GLUT
2              glewInit();
3              ilInit();
4
5              //Activar Buffers
6              glEnableClientState(GL_VERTEX_ARRAY);
7              glEnableClientState(GL_NORMAL_ARRAY);
8              glEnableClientState(GL_TEXTURE_COORD_ARRAY);
9
10             // alguns settings para OpenGL
11             glEnable(GL_DEPTH_TEST);
12             glEnable(GL_CULL_FACE);
13             glEnable(GL_TEXTURE_2D);
14             glClearColor(0.0f,0.0f,0.0f,0.0f);
15
16
17             //Carregar todas as estruturas para correr o Motor3D e prepara o
                   Picking
18             prepara_MotorXML(cena);
19             initPickingCena(cena);
20
21             // entrar no ciclo do GLUT
22             glutMainLoop();
```

Listing 8.1: Initiaização do glut

No main (Função) é onde estão definidos os campos que fazem parte do menu, que depois ao serem ativados entram num dos caso da função menu.

```
1
2               //MENU
3               M_Visual=glutCreateMenu(front_menu);
4               glutAddMenuEntry("GL POINT",1);
5               glutAddMenuEntry("GL LINE",2);
6               glutAddMenuEntry("GL FILL",3);
7
8               M_Camera=glutCreateMenu(front_menu);
9               glutAddMenuEntry("Modo Explorador",4);
10              glutAddMenuEntry("Modo FPS",5);
11
12              M_Luzes=glutCreateMenu(front_menu);
13              glutAddMenuEntry("Ligar",6);
14              glutAddMenuEntry("Desligar",7);
15
16              M_Texturas=glutCreateMenu(front_menu);
17              glutAddMenuEntry("Ligar",8);
18              glutAddMenuEntry("Desligar",9);
19
20              M_ViewFrustum=glutCreateMenu(front_menu);
21              glutAddMenuEntry("Ligar",10);
22              glutAddMenuEntry("Desligar",11);
23              glutAddMenuEntry("Desenhar Limites",12);
24              glutAddMenuEntry("Nao Desenhar Limites",13);
25
26              glutCreateMenu(front_menu);
27              glutAddSubMenu("Visualizacao",M_Visual);
28              glutAddSubMenu("Camera",M_Camera);
29              glutAddSubMenu("Luz",M_Luzes);
30              glutAddSubMenu("Texturas",M_Texturas);
31              glutAddSubMenu("ViewFrustumCulling",M_ViewFrustum);
```

Listing 8.2: Criaçaõ do menu

Abaixo temos um exemplo de como é que o menu é apresentado.

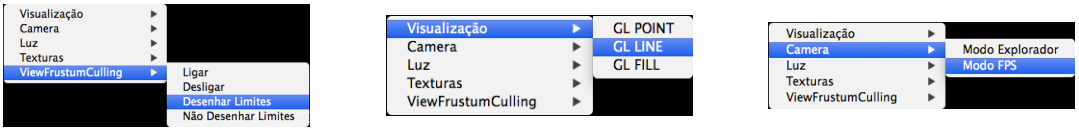

Figura 8.1: Operações que podem ser feitas usando o menu

Figura 8.2: Operações que podem ser feitas usando o menu (2)

8.2 Menu

```
1  void front_menu(int op){
2      switch (op) {
3          case 1:
4              glPolygonMode(GL_FRONT,GL_POINT);
5              break;
6          case 2:
7              glPolygonMode(GL_FRONT,GL_LINE);
8              break;
9          case 3:
10             glPolygonMode(GL_FRONT,GL_FILL);
11             break;
12         case 4:
13             glutKeyboardFunc(teclado_normal_explorador);
14             glutSpecialFunc(teclado_especial_explorador);
15             glutMouseFunc(rato_explorador);
16             glutMotionFunc(mov_rato_explorador);
17             tipo_camera=1;
18             break;
19         case 5:
20             glutKeyboardFunc(teclado_normal_fps);
21             glutSpecialFunc(teclado_especial_fps);
22             glutMouseFunc(rato_fps);
23             glutMotionFunc(mov_rato_fps);
24             tipo_camera=2;
25             break;
26         case 6:
27              glEnable(GL_LIGHTING);
28             break;
29         case 7:
30             glDisable(GL_LIGHTING);
31             break;
32         case 8:
33             glEnable(GL_TEXTURE_2D);
34             break;
35         case 9:
36             glDisable(GL_TEXTURE_2D);
37             break;
38         case 10:
39             enableViewFrustum=0;
40             break;
41         case 11:
42             enableViewFrustum=1;
43             break;
44         case 12:
45             caixasDesenho=1;
46             break;
47         case 13:
48             caixasDesenho=0;
49             break;
50
51         default:
52             break;
53     }
54     glutPostRedisplay();
55 }
```

Listing 8.3: Função menu

8.3 Render Scene

Na render scene é onde nós fazemos algumas chamadas acerca do ponto para onde estamos a olha quando a aplicação arranca , ou a chamada a algumas funções para a inicialização do processo de desenho dos objetos.

```
void renderScene(void) {

  // clear buffers
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  // set the camera
  glLoadIdentity();

    if (tipo_camera==1)
        modo_explorador();
    else
        if (tipo_camera==2)
            modo_fps();
        else{
            gluLookAt(0,3,5,
                      0.0, 0.0, 0.0,
                      0.0f, 1.0f, 0.0f);
        }

  // por instrucoes de desenho aqui

        //LUZES
    defineLuzes();

  rot_actual=rotacoes;
    tra_actual=translacoes;
    esc_actual=escalas;
    prop_actual=l_PropModel;

    n_desenhos=0;

    //Actualiza do tempo
    currentTime = glutGet(GLUT_ELAPSED_TIME);
    motor_XML(cena);

    sprintf(print, "Galaxy 3D => %d/%d desenhados\n",n_desenhos, total_desenhos);
    glutSetWindowTitle(print);

  // End of frame
  glutSwapBuffers();
}
```

Listing 8.4: Função Render scene

8.4 Motor_XML

Este motor_XMl é o responsável pela leitura e processamento do xml, passado como parâmetro.

O nosso motor está definido para ler os ficheiros xml de que são definidos pelos módulos de cpp que são gerados como os valores passados como argumento.

```
1  #ifndef Motor3D_motor_h
2  #define Motor3D_motor_h
3
4  #include <GLUT/glut.h>
5  #include <stdio.h>
6  #include "tinyxml/tinyxml.h"
7
8  void motor_XML(TiXmlNode *doc);
9
10 #endif
```

Listing 8.5: .h do motor

Capítulo 9

Extras

Os extras são objetivos que foram apresentados no primeiro capitulo, onde estes servem para melhorar a nossa atual nota do projeto.

Relativamente a esses extra tínhamos:

- Uso de técnicas de aceleração da renderização através de Vertex Buffer Objects(VBO's);
- Optimização com recurso a diferentes técnicas de culling, de que é exemplo o view frustum culling;
- Profiling e análise de desempenho;
- Picking.

No nosso projeto conseguimos implementar os atuais extras. Vamos agora apresentar os resultados relativos a implementação de cada um deles.

Relativo as VBO's temos a descrição da implementação no cap. 11.1, este capitulo apenas vamos mostrar a diferença de valores ao utilizarmos VBO's ou a ausência delas.

9.1 Cena

Neste momento foi onde começámos a definir o xml de modo a que este ao ser feito o parsing comece a construir o sistema solar.

O xml foi construido usando tags que definem o que é que vamos desenhar, temos que a definição dos planetas se encontra dentro das tags ¡grupo¿ ¡/grupo¿ e é dentro destas tags que podemos encontrar as definições de posição, translação, rotação de um planeta.

A definição da c parte pela leienatura do xml, "preencher"as respetivas estruturas, onde depois são lidos as posições de atribuição de cada planeta, os mesmos são desenhados e caso tenha sido definido no xml a translação os catmull é utilizado para desenhar as linhas onde o planeta vai efetuar o seu movimento circular.

Apresentamos abaixo a imagem do sistema solar que por nós foi implementado.

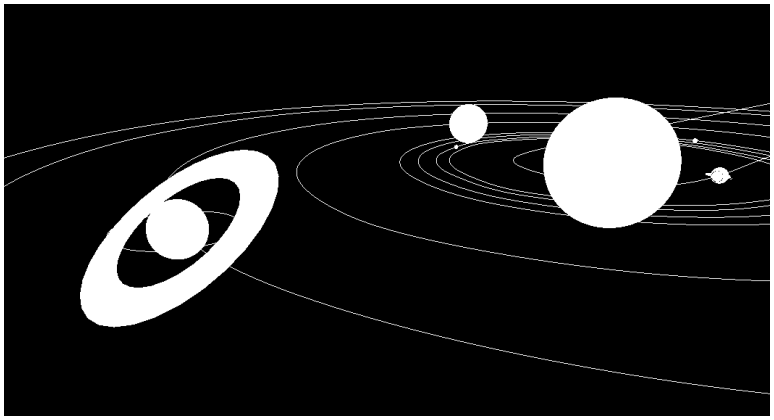

Figura 9.1: Visão do Sistema Solar

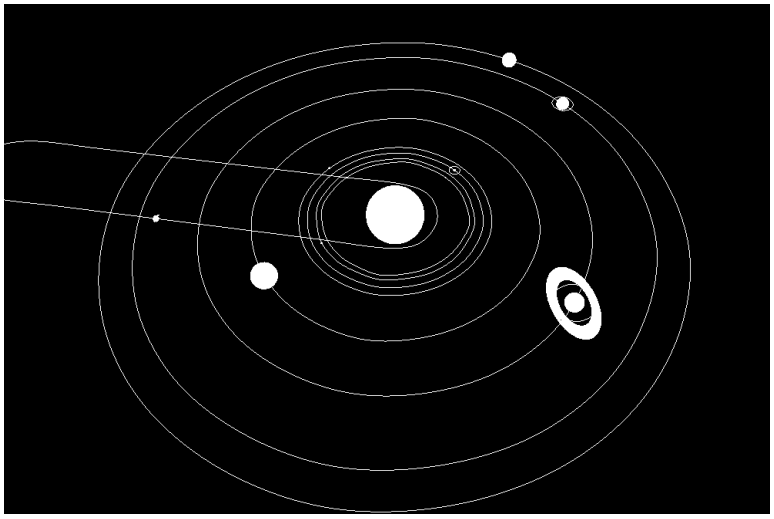

Figura 9.2: Visão de Cima do Sistema Solar

9.2 Câmaras

Temos também definido nesta etapas duas câmaras que são apresentadas como câmara_explorador e a câmara_fps.

Quando temos o modo fps temos que a câmara se movimenta para a frente e para trás e o ponto par onde ela está a olhar é definido por dois ângulo em simultâneo.

Quando temos câmara modo explorador, temos que esta está a apontar par pontos específicos definidos através de coordenadas cartesianas.

```
#ifndef Motor3D_camera_h
#define Motor3D_camera_h

#include <math.h>
#include <GLUT/glut.h>

void modo_explorador();
void rato_explorador(int botao, int estado, int x, int y);
void mov_rato_explorador(int x, int y);
void teclado_normal_explorador(unsigned char tecla,int x, int y);
void teclado_especial_explorador(int tecla,int x, int y);


#endif
```

Listing 9.1: .h da camara modo explorador

```
void modo_explorador(){
    //Camera em modo explorador
  gluLookAt(look[0]+(raio)*sin(angCam_h+angAux_h)*cos(angCam_v+angAux_v),look
      [1]+(raio)*sin(angCam_v+angAux_v),look[2]+(raio)*cos(angCam_h+angAux_h)*cos
      (angCam_v+angAux_v),
            look[0], look[1], look[2],
              0.0f, 1.0f, 0.0f);
}
```

Listing 9.2: gluLookAt do modo esploradpr

Nota: Caso não seja definida nenhuma câmara antes de se iniciar a aplicação a câmara que estará definida por omissão é a do modo explorador.

```
#ifndef Motor3D_camera_fps_h
#define Motor3D_camera_fps_h

#include <math.h>
#include <GLUT/glut.h>

void modo_fps();
void teclado_especial_fps(int tecla,int x, int y);
void teclado_normal_fps(unsigned char tecla,int x, int y);
void mov_rato_fps(int x, int y);
void rato_fps(int botao, int estado, int x, int y);

#endif
```

Listing 9.3: .h da camara modo fps

```
void modo_fps(){

    //Camera em modo explorador
  gluLookAt(px,py,pz,
            px+sin(angCamFPS_h)*cos(angCamFPS_v), py+sin(angCamFPS_v), pz+cos(
                angCamFPS_h)*cos(angCamFPS_v),
              0.0f, 1.0f, 0.0f);


}
```

Listing 9.4: gluLookAt da camara fps

9.3 View Frustum

O view frustum que é uma técnica utilizada para apenas serem desenhados os objetos que se encontram dentro dentro do raio de visão da câmara, isto é que os objetos que se encontram entre o near e o far.

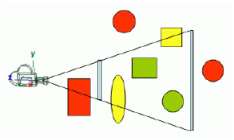

Figura 9.3: Exemplo de view frustum

Temos que na figura todos os objetos vermelhos não são desenhados pois não se encontram dentro dentro da chamada view frustum, os objetos amarelos são apenas desenhados porque estão parcialmente dentro da área e por fim os objetos verdes são desenhados pois estes encontram-se na sua totalidade dentro do view frustum.

```
1 //Tipo  --  0 => Caixa alinhada com os eixos
2 //          1 => Esfera
3
4 typedef struct sVFC{
5
6     float maxX, maxY, maxZ;
7     float minX, minY, minZ;
8
9 }*ViewFrustum, NViewFrustum;
```

Listing 9.5: Estrtura de dados do View Frustum

Esta estrutura de dados serve para quando estamos a desenhar um modelo, esse modelo é desenhado e depois quando vamos passar ao próximo temos que é feito um ”reset”que permite desenhar outro modelo sem interferir com o anterior.

Fazendo agora a analogia para o sistema solar, temos que ao aplicarmos o view frustum os planetas apenas são enviados para a placa aqueles que estão no ângulo de visão da câmara todos os outros não são desdenhados.

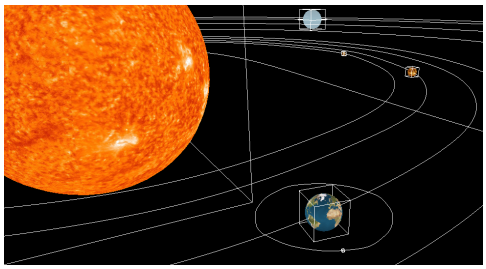

Figura 9.4: Visão do Sistema Solar View frustum

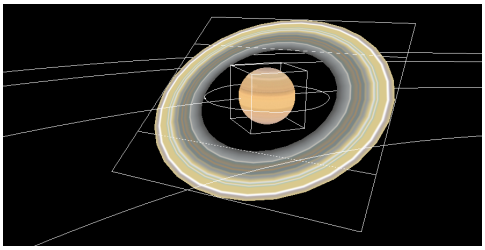

Figura 9.5: Visão de um planeta com view frustum

9.4 Profilling

Relativamente ao profilling e a análise de desempenho, temos as medidas que efetuamos quando temos a nossa aplicação a correr. Para efetuarmos estas medidas usamos os gráficos de desempenho do CPU, onde podemos concluir que quando a aplicação está a iniciar os valores do cpu são maiores, isto deve-se ao fato de "estarmos"a enviar todos os ponto para a placa, de pois disso os valores do desempenho decrescem relativamente. Os valores para os quais esta conclusão se refere são para a implementação do sistema solar usando VBO's.

Podemos verificar os valore relativos ao profilling nas próximas imagens. As imagens que nós vamos mostrar são a imagens onde temos a chamada de certas funções como por exemplo a render scene e o motor xml.

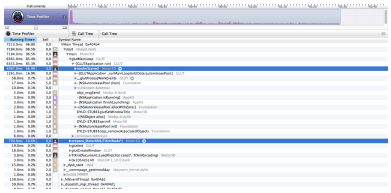

Figura 9.6: Profilling 1

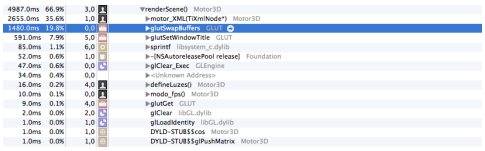

Figura 9.7: Profilling 2

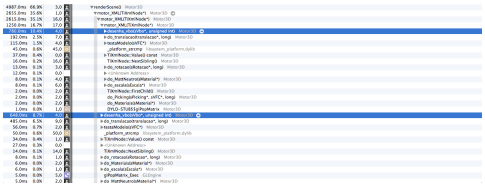

Figura 9.8: Profilling 3

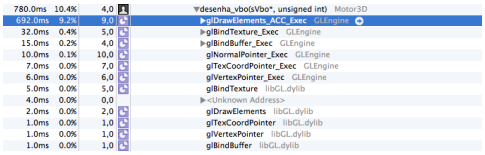

Figura 9.9: Profilling 4

9.5 Picking

A operação de picking corresponde à selecção de um objecto. Realiza-se através do mapeamentos na unidade de saída gráfica das projecções dos objectos da cena contêm o cursor. Normalmente, é escolhido o objecto que se encontre mais próximo do ponto de vista. A implementação foi feita para que quando seja selecionada um pixel que pertença a um modelo, este dê a informação que foi definida no xml.

Temos aqui um pequeno exemplo de como é que implementamos o picking, isto é, quando o botão do rato for "acionado", então é apresentada a definição que estiver atribuída a esse modelo.

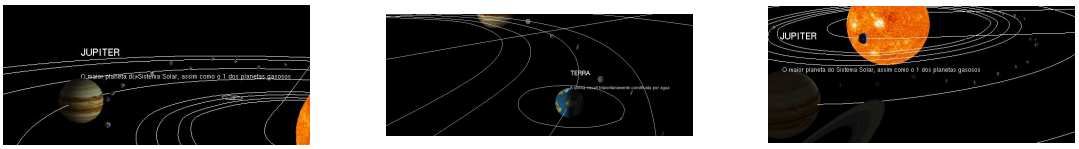

Figura 9.10: Exemplo do picking usado

A estrtura de dados que vamos agora apresentar é a que usamos par definir o picking. Para termos o picking tinhamos de ter uma estrura que armazenasse a informação que fosse encontrada no xml. Temos entã o a estrtura que guarda a cor que está associada ao modelo, o titulo correspondente, a descrição relativa a esse modelo e como é desenhado depois o modelo.

```
typedef struct sPicking{
    int cor;
    const char *descricao;
    const char *titulo;
    int desenha;
    long pressTime;
}*Picking, NPicking;
```

Listing 9.6: Estrtura de dados do Picking

9.6 XML

9.6.1 Câmara

A câmara é onde nós definimos os tipos de camaras que podem existir, podemos ter um ou mais blocos consoante o nº de camaras que queremos, neste caso temos a definição das duas camaras.

```
<!--
CAMERA (Pode ter 2 blocos para as pre-definicoes de cada tipo. A ultima a ser
     definida e a mostrada)
-->
<cameras>
<camera tipo="explorador">
<zoom raio="180" avanco="3"/>
<centro x="20" y="0" z="0"/>
<vista latitude="-0.256" longitude="1.167"/>
</camera>
<camera tipo="fps">
<velocidade avanco="3"/>
<centro x="-70.99" y="37.74" z="-60.581"/>
<vista latitude="-0.256" longitude="1.167"/>
</camera>
</cameras>
```

Listing 9.7: XML para a definição da câmara

9.6.2 Luzes

A definição das luzes no XML é definda através da indicação dos pontos que vão ter a luz e qual o tipo de iluminação que vai existir. A definição da luz encontra-se dentro da tag ¡luzes¿ ¡/luzes¿.

```
<!--
LUZES (e preciso este bloco para activar a Iluminacao)
-->
<luzes>
<!-- LUZ 0 -->
<luz tipo="ponto" y="17">
<ambiente r="0.2" g="0.2" b="0.2"/>
<difusa g="1" b="1"/>
</luz>
<!-- LUZ 1 -->
<luz tipo="ponto" y="-17">
<difusa r="1" g="1" b="1"/>
</luz>
<!-- LUZ 2 -->
<luz tipo="ponto" x="17">
<difusa r="1" g="1" b="1"/>
</luz>
<!-- LUZ 3 -->
<luz tipo="ponto" x="-17">
<difusa r="1" g="1" b="1"/>
</luz>
<!-- LUZ 4 -->
<luz tipo="ponto" z="17">
<difusa r="1" g="1" b="1"/>
</luz>
<!-- LUZ 5 -->
<luz tipo="ponto" z="-17">
<ambiente r="0.2" g="0.2" b="0.2"/>
<difusa r="1" g="1" b="1"/>
</luz>
</luzes>
```

Listing 9.8: XMl para a definição das luzes

9.6.3 Cena

A cena é onde nós definimos todo o que possa ser utilizado para um planeta, isto é dentro da cena temos um modelo qua mais tarde dá um planeta.

Dentro do modelo é onde temos os pontos que são usados para definir o planeta, as texturas que aplicamos a esse planeta.

```
<cena>
<grupo>
<!-- MERCURIO -->
<translacao tempo="10.8797">
<ponto x="44.000000" y="0" z="0.000000"/>
<ponto x="41.846489" y="0" z="13.596748"/>
<ponto x="35.596748" y="0" z="25.862551"/>
<ponto x="25.862551" y="0" z="35.596748"/>
<ponto x="13.596748" y="0" z="41.846489"/>
<ponto x="0.000000" y="0" z="44.000000"/>
<ponto x="-13.596748" y="0" z="41.846489"/>
<ponto x="-25.862551" y="0" z="35.596748"/>
<ponto x="-35.596748" y="0" z="25.862551"/>
<ponto x="-41.846489" y="0" z="13.596748"/>
<ponto x="-44.000000" y="0" z="0.000000"/>
<ponto x="-41.846489" y="0" z="-13.596748"/>
<ponto x="-35.596748" y="0" z="-25.862551"/>
<ponto x="-25.862551" y="0" z="-35.596748"/>
<ponto x="-13.596748" y="0" z="-41.846489"/>
<ponto x="-0.000000" y="0" z="-44.000000"/>
<ponto x="13.596748" y="0" z="-41.846489"/>
<ponto x="25.862551" y="0" z="-35.596748"/>
<ponto x="35.596748" y="0" z="-25.862551"/>
<ponto x="41.846489" y="0" z="-13.596748"/>
</translacao>
<escala x="0.3439" y="0.3439" z="0.3439"/>
<rotacao nome="esfera" tempo="0.785" x="0" y="1" z="0"/>
<modelo ficheiro="planeta.vbo" textura="mercurio.jpg">
<picking>
<titulo>MERCURIO</titulo>
<descricao>Planeta mais pequeno do Sistema Solar</descricao>
</picking>
</modelo>
</grupo>
<grupo>
</cena>
```

Listing 9.9: XML relativo à cena

9.6.4 Picking

Para definirmos o picking temos que no XML dentro da tag modelo, definimos uma nova tag ¡picking¿ ¡/picking¿ e dentro dessa tag definimos o que queremos, no nosso caso definimos o titulo do planeta com as tag ¡titulo¿ ¡/titulo¿ e uma breve descrição com as tag ¡descricao¿ ¡/descricao¿.

```
  <picking>
  <titulo>MERCURIO</titulo>
    <descricao> Planeta mais pequeno do Sistema Solar </descricao>
  </picking>
```

Listing 9.10: Excerto do xml para picking

Capítulo 10

Conclusões Finais

Com o presente relatório, esperemos que qualquer leitor que não tenha contacto com os conteúdos descrito em 2, entenda como é que foi feito o alinhamento, desde a fase inicial, como é que os respetivos conceitos foram aplicados.

Relativamente ao desenvolvimento de todo o projeto temos que todos os objetivos que foram proposto no capitulo inicial, podemos concluir que foram relaizados com sucesso.

Bibliografia

[1] Jesper Tveit
Bezier Curves and Surfaces
Abril
2002
http://www.gamedev.net/page/resources/_/technical/graphics-programming-and-theory/bezier-curves-and-surfaces-r1808,

[2] Tutoriais de Opengl
Maio
2014
http://www.lighthouse3d.com/

Capítulo 11

Anexos

11.1 Implementação das primitivas com VBO's

A nossa implementação de VBO's seguem todo os mesmos esquema, pois quando se fala em VBO's temos que os pontos quando são lidos para depois serem desenhados já se encontram na memória da placa gráfica sendo assim mais fácil para esta saber onde os próximos pontos a desenhar se encontram. Esta implementação permite-nos reduzir substancialmente a carga que é feita ao processador do computador.

Abaixo deixamos a nossa implementações das primitivas geométricas em VBO's.

11.1.1 Anel

```
1  // Anel para as vbo's
2  void anelVBO(float raio_fora, float raio_dentro, int fatias, int aneis, int ori,
       FILE* f){
3
4     float angulo=(2*M_PI)/fatias,y=0,l_aux,raio=(raio_fora-raio_dentro)/aneis;
5      int i=0,v=0,j,n=0,t=0,avanco;
6      float texFactor_aneis=1.0f/aneis;
7      int replic=0;
8
9      int n_pontos=((fatias+1)*(aneis+1))*3;
10     int n_indices=6*fatias*aneis;
11     int tex_pontos= (2*n_pontos)/3;
12
13
14     int *indices=(int*)malloc(n_indices*sizeof(int));
15
16     float    *vertexB=(float*)malloc(n_pontos*sizeof(float)),
17              *normalB=(float*)malloc(n_pontos*sizeof(float)),
18              *texB=(float*)malloc(tex_pontos*sizeof(float));
19
20     if(ori){
21
22         for (j=0; j<=fatias; j++) {
23              vertexB[v++]=raio_dentro*sin(y); vertexB[v++]=0; vertexB[v++]=
                    raio_dentro*cos(y);
24              normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
25              texB[t++]=replic++; texB[t++]=1;
26              y+=angulo;
27         }
28         avanco=fatias+1;
29
30         for(j=1;j<=aneis;j++){
31              raio_dentro+=raio;
32              y=0;
```

```
33              replic=0;
34              for(l_aux=0;l_aux<=fatias;l_aux++){
35
36                  vertexB[v++]=raio_dentro*sin(y); vertexB[v++]=0; vertexB[v++]=
                        raio_dentro*cos(y);
37                  normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
38                  texB[t++]=replic++;texB[t++]=1-j*texFactor_aneis;
39
40                  if(l_aux!=fatias){
41                      indices[i++]=avanco-(fatias+1)+l_aux;
42                      indices[i++]=avanco+l_aux;
43                      indices[i++]=avanco+l_aux+1;
44
45                      indices[i++]=avanco-(fatias+1)+l_aux+1;
46                      indices[i++]=avanco-(fatias+1)+l_aux;
47                      indices[i++]=avanco+l_aux+1;
48                  }
49
50                  y+=angulo;
51              }
52              avanco+=fatias+1;
53          }
54      }else{
55          for (j=0; j<=fatias; j++) {
56              vertexB[v++]=raio_dentro*sin(y); vertexB[v++]=0; vertexB[v++]=
                    raio_dentro*cos(y);
57              normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
58              texB[t++]=replic++; texB[t++]=1;
59              y+=angulo;
60          }
61          avanco=fatias+1;
62
63          for(j=1;j<=aneis;j++){
64              raio_dentro+=raio;
65              y=0;
66              replic=0;
67              for(l_aux=0;l_aux<=fatias;l_aux++){
68
69                  vertexB[v++]=raio_dentro*sin(y); vertexB[v++]=0; vertexB[v++]=
                        raio_dentro*cos(y);
70                  normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
71                  texB[t++]=replic++;texB[t++]=1-j*texFactor_aneis;
72
73                  if(l_aux!=fatias){
74                      indices[i++]=avanco-(fatias+1)+l_aux;
75                      indices[i++]=avanco+l_aux+1;
76                      indices[i++]=avanco+l_aux;
77
78                      indices[i++]=avanco-(fatias+1)+l_aux+1;
79                      indices[i++]=avanco+l_aux+1;
80                      indices[i++]=avanco-(fatias+1)+l_aux;
81                  }
82
83                  y+=angulo;
84              }
85              avanco+=fatias+1;
86          }
87      }
88
89      //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
90      fprintf(f, "%f %f %d %d %f %f\n",raio_fora,-raio_fora,0,0,raio_fora,-
            raio_fora);
91
92     //Imprimir os vertices, indices, normais e coordenadas de textura
93      fprintf(f, "%d\n",n_pontos);
94      for(i=0;i<n_pontos;i+=3)
95          fprintf(f, "%f %f %f\n",vertexB[i],vertexB[i+1],vertexB[i+2]);
96
97      fprintf(f, "%d\n",n_indices);
98      for(i=0;i<n_indices;i+=3)
99          fprintf(f, "%d %d %d\n",indices[i],indices[i+1],indices[i+2]);
100
101     for(i=0;i<n_pontos;i+=3)
102         fprintf(f, "%f %f %f\n",normalB[i],normalB[i+1],normalB[i+2]);
103
104     for(i=0;i<tex_pontos ;i+=2)
105         fprintf(f, "%f %f\n",texB[i],texB[i+1]);
106
107     free(vertexB);
108     free(normalB);
109     free(texB);
110
111 }
```

Listing 11.1: Anel versão VBO

11.1.2 Cilindro

```
1  void cilindroVBO(float raio,int fatias,int camadas,float altura,int aneis, FILE*
       f){
2      float angulo=(2*M_PI)/fatias,y=0,l_aux, r_aux,alt_aux=altura;
3      int i=0,v=0,j,n=0,t=0,avanco=0;
4      float texFactor_fatias=1.0f/fatias;
5      float texFactor_camadas=1.0f/camadas;
6      float texFactor_aneis=1.0f/aneis;
7
8
9      int n_pontos=((fatias+1)*(camadas+1)*3) + ((fatias+1)+(fatias+1)*aneis)*6;
10     int n_indices= (fatias*camadas*3*2) + (fatias*(aneis-1)*2+fatias)*6;
11     int tex_pontos=(n_pontos*2)/3;
12
13     float *vertexB=(float*)malloc(n_pontos*sizeof(float)),
14     *normalB=(float*)malloc(n_pontos*sizeof(float)),
15     *texB=(float*)malloc(tex_pontos*sizeof(float));
16     int *indices=(int*)malloc(n_indices*sizeof(int));
17
18
19
20
21     //-------------- Circulo da base -----------------//
22     r_aux=raio/aneis;
23
24     //Primeiro ponto central
25     for (l_aux=0; l_aux<=fatias; l_aux++) {
26         vertexB[v++]=0;vertexB[v++]=0;vertexB[v++]=0;
27         normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
28         texB[t++]=l_aux*texFactor_fatias;texB[t++]=1;
29     }
30     avanco+=fatias+1;
31
32     //Primeiro circulo
33     for(l_aux=0;l_aux<=fatias;l_aux++){
34
35         vertexB[v++]=r_aux*sin(y);vertexB[v++]=0;vertexB[v++]=r_aux*cos(y);
36         normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
37         texB[t++]=l_aux*texFactor_fatias; texB[t++]=1.0f-texFactor_aneis;
38         if(l_aux!=fatias){
39             indices[i++]=avanco-(fatias+1)+l_aux;
40             indices[i++]=avanco+l_aux+1;
41             indices[i++]=avanco+l_aux;
42         }
43         y+=angulo;
44     }
45     avanco+=fatias+1;
46
47     //Aneis da base
48     for(j=1;j<aneis;j++){
49         r_aux+=raio/aneis;
50         y=0;
51         for(l_aux=0;l_aux<=fatias;l_aux++){
52
53             vertexB[v++]=r_aux*sin(y); vertexB[v++]=0; vertexB[v++]=r_aux*cos(y);
54             normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
55             texB[t++]=l_aux*texFactor_fatias;texB[t++]=1.0f-((j+1)*
                   texFactor_aneis);
56
57             if(l_aux!=fatias){
58                 indices[i++]=avanco-(fatias+1)+l_aux;
59                 indices[i++]=avanco-(fatias+1)+l_aux+1;
60                 indices[i++]=avanco+l_aux+1;
61
62                 indices[i++]=avanco+l_aux;
63                 indices[i++]=avanco-(fatias+1)+l_aux;
64                 indices[i++]=avanco+l_aux+1;
65             }
66             y+=angulo;
67         }
68         avanco+=fatias+1;
69     }
70
71
```

```
72      //------------ Corpo ------------//
73      r_aux=raio;
74      for(j=0;j<=camadas;j++){
75          y=0;
76
77          for(l_aux=0;l_aux<=fatias;l_aux++){
78
79              vertexB[v++]=r_aux*sin(y); vertexB[v++]=alt_aux; vertexB[v++]=r_aux*
                    cos(y);
80              normalB[n++]=sin(y);normalB[n++]=0;normalB[n++]=cos(y);
81              texB[t++]=l_aux*texFactor_fatias;texB[t++]=j*texFactor_camadas;
82
83              if(l_aux!=fatias){
84                  if(j!=camadas){
85                      indices[i++]=avanco+l_aux;
86                      indices[i++]=avanco+(fatias+1)+l_aux;
87                      indices[i++]=avanco+l_aux+1;
88
89                      indices[i++]=avanco+(fatias+1)+l_aux;
90                      indices[i++]=avanco+(fatias+1)+l_aux+1;
91                      indices[i++]=avanco+l_aux+1;
92                  }
93              }
94              y+=angulo;
95          }
96
97          avanco+=fatias+1;
98          alt_aux-=altura/camadas;
99      }
100
101
102     //------------ Circulo do topo ------------//
103     r_aux=raio/aneis;
104
105     //Primeiro ponto central
106     for (l_aux=0; l_aux<=fatias; l_aux++) {
107         vertexB[v++]=0;vertexB[v++]=altura;vertexB[v++]=0;
108         normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
109         texB[t++]=l_aux*texFactor_fatias;texB[t++]=1;
110     }
111     avanco+=fatias+1;
112
113     //Primeiro circulo
114     for(l_aux=0;l_aux<=fatias;l_aux++){
115
116         vertexB[v++]=r_aux*sin(y);vertexB[v++]=altura;vertexB[v++]=r_aux*cos(y);
117         normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
118         texB[t++]=l_aux*texFactor_fatias; texB[t++]=1.0f-texFactor_aneis;
119         if(l_aux!=fatias){
120             indices[i++]=avanco-(fatias+1)+l_aux;
121             indices[i++]=avanco+l_aux;
122             indices[i++]=avanco+l_aux+1;
123         }
124         y+=angulo;
125     }
126     avanco+=fatias+1;
127
128     //Aneis do topo
129     for(j=1;j<aneis;j++){
130         r_aux+=raio/aneis;
131         y=0;
132         for(l_aux=0;l_aux<=fatias;l_aux++){
133
134             vertexB[v++]=r_aux*sin(y); vertexB[v++]=altura; vertexB[v++]=r_aux*
                    cos(y);
135             normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
136             texB[t++]=l_aux*texFactor_fatias;texB[t++]=1.0f-((j+1)*
                    texFactor_aneis);
137
138             if(l_aux!=fatias){
139                 indices[i++]=avanco-(fatias+1)+l_aux;
140                 indices[i++]=avanco+l_aux+1;
141                 indices[i++]=avanco-(fatias+1)+l_aux+1;
142
143                 indices[i++]=avanco+l_aux;
144                 indices[i++]=avanco+l_aux+1;
145                 indices[i++]=avanco-(fatias+1)+l_aux;
146             }
147
148             y+=angulo;
149         }
150         avanco+=fatias+1;
151     }
152
```

```
153     //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
154     fprintf(f, "%f %f %f %d %f %f\n",raio, -raio,altura,0,raio,-raio);
155
156    //Imprimir os vertices, indices, normais e coordenadas de textura
157     fprintf(f, "%d\n",n_pontos);
158     for(i=0;i<n_pontos;i+=3)
159         fprintf(f, "%f %f %f\n",vertexB[i],vertexB[i+1],vertexB[i+2]);
160
161     fprintf(f, "%d\n",n_indices);
162     for(i=0;i<n_indices;i+=3)
163         fprintf(f, "%d %d %d\n",indices[i],indices[i+1],indices[i+2]);
164
165     for(i=0;i<n_pontos;i+=3)
166         fprintf(f, "%f %f %f\n",normalB[i],normalB[i+1],normalB[i+2]);
167
168     for(i=0;i<tex_pontos ;i+=2)
169         fprintf(f, "%f %f\n",texB[i],texB[i+1]);
170
171
172     free(vertexB);
173     free(normalB);
174     free(texB);
175
176
177 }
```

Listing 11.2: Cilindro versão VBO

11.1.3 Circulo

```
1
2  void circuloVBO(float raio, int lados,int aneis, float altura,int ori, FILE *f){
3
4      float angulo=(2*M_PI)/lados,y=0,l_aux, r_aux;
5      int i=0,v=0,j=0,avanco;
6
7      raio=raio/aneis;
8      r_aux=raio;
9      int n_pontos=(1+lados*aneis)*3;
10     int n_indices=(lados*(aneis-1)*2+lados)*3;
11
12     int *indices=(int*)malloc(n_indices*sizeof(int));
13     float *vertexB=(float*)malloc(n_pontos*sizeof(float));
14
15     if(ori){
16         vertexB[v++]=0;vertexB[v++]=altura;vertexB[v++]=0;
17         for(l_aux=0;l_aux<lados;l_aux++){
18
19             vertexB[v++]=r_aux*sin(y);vertexB[v++]=altura;vertexB[v++]=r_aux*cos(
                   y);
20
21             indices[i++]=0;
22             indices[i++]=l_aux+1;
23             indices[i++]=l_aux+2;
24             y+=angulo;
25         }
26         indices[i-1]=1;
27
28         for(j++;j<aneis;j++){
29             r_aux+=raio;
30             y=0;
31             for(l_aux=0;l_aux<lados;l_aux++){
32                 avanco=j*lados+1;
33
34                 vertexB[v++]=r_aux*sin(y); vertexB[v++]=altura; vertexB[v++]=
                       r_aux*cos(y);
35
36                 indices[i++]=avanco-lados+l_aux;
37                 indices[i++]=avanco+l_aux;
38                 indices[i++]=avanco-lados+l_aux+1;
39
40                 indices[i++]=avanco+l_aux;
41                 indices[i++]=avanco+l_aux+1;
42                 indices[i++]=avanco-lados+l_aux+1;
43
44                 y+=angulo;
45             }
46             indices[i-4]=avanco-lados;
47             indices[i-2]=avanco;
```

```
48              indices[i-1]=avanco-lados;
49          }
50      }else{
51          vertexB[v++]=0;vertexB[v++]=altura;vertexB[v++]=0;
52          for(l_aux=0;l_aux<lados;l_aux++){
53
54              vertexB[v++]=r_aux*sin(y);vertexB[v++]=altura;vertexB[v++]=r_aux*cos(
                    y);
55
56              indices[i++]=0;
57              indices[i++]=l_aux+2;
58              indices[i++]=l_aux+1;
59              y+=angulo;
60          }
61          indices[i-2]=1;
62
63          for(j++;j<aneis;j++){
64              r_aux+=raio;
65              y=0;
66              for(l_aux=0;l_aux<lados;l_aux++){
67                  avanco=j*lados+1;
68
69                  vertexB[v++]=r_aux*sin(y); vertexB[v++]=altura; vertexB[v++]=
                        r_aux*cos(y);
70
71                  indices[i++]=avanco-lados+l_aux;
72                  indices[i++]=avanco-lados+l_aux+1;
73                  indices[i++]=avanco+l_aux;
74
75                  indices[i++]=avanco+l_aux;
76                  indices[i++]=avanco-lados+l_aux+1;
77                  indices[i++]=avanco+l_aux+1;
78
79                  y+=angulo;
80              }
81              indices[i-5]=avanco-lados;
82              indices[i-1]=avanco;
83              indices[i-2]=avanco-lados;
84          }
85
86      }
87
88      //Imprimir os vertices e indices
89      fprintf(f, "%d\n",n_pontos);
90      for(i=0;i<n_pontos;i+=3)
91          fprintf(f, "%f %f %f\n",vertexB[i],vertexB[i+1],vertexB[i+2]);
92
93      fprintf(f, "%d\n",n_indices);
94      for(i=0;i<n_indices;i+=3)
95          fprintf(f, "%d %d %d\n",indices[i],indices[i+1],indices[i+2]);
96  }
```

Listing 11.3: Circulo versão VBO

11.1.4 Cone

```
1   void coneVBO(float raio, float altura, int fatias, int aneis, int camadas, FILE *
        f){
2
3       float angulo=(2*M_PI)/fatias,y=0, r_aux,factor_h=(raio/camadas),alt_aux=0,
            v1[3],v2[3];
4       int i=0,v=0,j=0,n=0,avanco=0,t=0,l_aux;
5       float texFactor_fatias=1.0f/fatias;
6       float texFactor_aneis=1.0f/camadas;
7
8       //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
9       fprintf(f, "%f %f %f %d %f %f\n",raio, -raio,altura,0,raio,-raio);
10
11
12      altura/=camadas;
13      raio=raio/aneis;
14      r_aux=raio;
15      int n_pontos=(((fatias+1)+(fatias+1)*aneis)*3)  + ((camadas+1)*(fatias+1))*3;
16      int n_indices=(fatias*(aneis-1)*2+fatias)*3  +(fatias*(camadas-1)*2+fatias)
            *3;
17      int tex_pontos= (2*n_pontos)/3;
18
19
20      int* indices=(int*)malloc(n_indices*sizeof(int));
```

```
21      float *vertexB=(float*)malloc(n_pontos*sizeof(float)),
22      *normalB=(float*)malloc(n_pontos*sizeof(float)),
23      *normLado=(float*)malloc((fatias+1)*3*sizeof(float)),
24      *texB=(float*)malloc(tex_pontos*sizeof(float));
25
26
27
28
29
30
31      //--------- Base do cone ----------//
32
33      //Primeiro ponto central
34      for (l_aux=0; l_aux<=fatias; l_aux++) {
35          vertexB[v++]=0;vertexB[v++]=0;vertexB[v++]=0;
36          normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
37          texB[t++]=l_aux*texFactor_fatias;texB[t++]=1;
38      }
39      avanco+=fatias+1;
40
41      //Circulo central
42      for(l_aux=0;l_aux<=fatias;l_aux++){
43
44          vertexB[v++]=r_aux*sin(y);vertexB[v++]=0;vertexB[v++]=r_aux*cos(y);
45          normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
46          texB[t++]=l_aux*texFactor_fatias;texB[t++]=1-texFactor_aneis;
47          if(l_aux!=fatias){
48              indices[i++]=l_aux;
49              indices[i++]=avanco+l_aux+1;
50              indices[i++]=avanco+l_aux;
51          }
52          y+=angulo;
53      }
54      avanco+=fatias+1;
55
56      //Aneis da base
57      for(j=1;j<aneis;j++){
58          r_aux+=raio;
59          y=0;
60          for(l_aux=0;l_aux<=fatias;l_aux++){
61
62              vertexB[v++]=r_aux*sin(y); vertexB[v++]=0; vertexB[v++]=r_aux*cos(y);
63              normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
64              texB[t++]=l_aux*texFactor_fatias;texB[t++]=1-(j+1)*texFactor_aneis;
65
66              if(l_aux!=fatias){
67                  indices[i++]=avanco-(fatias+1)+l_aux;
68                  indices[i++]=avanco-(fatias+1)+l_aux+1;
69                  indices[i++]=avanco+l_aux;
70
71                  indices[i++]=avanco+l_aux;
72                  indices[i++]=avanco-(fatias+1)+l_aux+1;
73                  indices[i++]=avanco+l_aux+1;
74              }
75
76              y+=angulo;
77          }
78          avanco+=fatias+1;
79      }
80
81      //Calcular Normais do corpo
82      y=0;
83      for(l_aux=0;l_aux<=fatias;l_aux++){
84
85          v1[0]=r_aux*sin(y+angulo) - r_aux*sin(y-angulo);
86          v1[1]=0;
87          v1[2]=r_aux*cos(y+angulo) - r_aux*cos(y-angulo);
88          v2[0]=(r_aux-factor_h)*sin(y) - r_aux*sin(y);
89          v2[1]=altura;
90          v2[2]=(r_aux-factor_h)*cos(y) - r_aux*cos(y);
91          normal(v1, v2);
92
93          normLado[l_aux*3]=resN[0];
94          normLado[l_aux*3+1]=resN[1];
95          normLado[l_aux*3+2]=resN[2];
96
97          y+=angulo;
98      }
99
100     //---------- Corpo ----------//
101     for(j=0;j<=camadas;j++){
102
103         y=0;
104         for(l_aux=0;l_aux<=fatias;l_aux++){
```

```
105
106             vertexB[v++]=r_aux*sin(y); vertexB[v++]=alt_aux; vertexB[v++]=r_aux*
                    cos(y);
107             normalB[n++]= normLado[l_aux*3];normalB[n++]= normLado[l_aux*3+1];
                    normalB[n++]= normLado[l_aux*3+2];
108             texB[t++]=l_aux*texFactor_aneis;texB[t++]=1-j*texFactor_fatias;
109             if(l_aux!=fatias && j!=0){
110                 if(j!=camadas){
111                     indices[i++]=avanco-(fatias+1)+l_aux;
112                     indices[i++]=avanco-(fatias+1)+l_aux+1;
113                     indices[i++]=avanco+l_aux;
114
115                     indices[i++]=avanco+l_aux;
116                     indices[i++]=avanco-(fatias+1)+l_aux+1;
117                     indices[i++]=avanco+l_aux+1;
118                 }else{
119                     indices[i++]=avanco-(fatias+1)+l_aux;
120                     indices[i++]=avanco-(fatias+1)+l_aux+1;
121                     indices[i++]=avanco+l_aux;
122                 }
123             }
124
125             y+=angulo;
126         }
127
128         alt_aux+=altura;
129         r_aux-=factor_h;
130         avanco+=fatias+1;
131     }
132
133     //Imprimir os vertices, indices, normais e coordenadas de textura
134     fprintf(f, "%d\n",n_pontos);
135     for(i=0;i<n_pontos;i+=3)
136         fprintf(f, "%f %f %f\n",vertexB[i],vertexB[i+1],vertexB[i+2]);
137
138     fprintf(f, "%d\n",n_indices);
139     for(i=0;i<n_indices;i+=3)
140         fprintf(f, "%d %d %d\n",indices[i],indices[i+1],indices[i+2]);
141
142     for(i=0;i<n_pontos;i+=3)
143         fprintf(f, "%f %f %f\n",normalB[i],normalB[i+1],normalB[i+2]);
144
145     for(i=0;i<tex_pontos ;i+=2)
146         fprintf(f, "%f %f\n",texB[i],texB[i+1]);
147
148
149     free(vertexB);
150     free(normalB);
151     free(texB);
152     free(normLado);
153 }
```

Listing 11.4: Cone Versão VBO

11.1.5 Esfera

```
1  void esferaVBO(float raio, int camadas, int fatias,FILE* f){
2      float   angulo_cir=(2*M_PI)/fatias,
3      angulo_h=(M_PI)/camadas,y=0,l_aux,h_aux=M_PI_2,
4      texFactor_fatias=1.0f/fatias,
5      texFactor_camadas=1.0f/camadas;
6
7      int i=0,v=0,n=0,t=0,j,avanco;
8
9      int n_pontos=(2*(fatias+1)+(fatias+1)*(camadas-1))*3;
10     int n_indices=(fatias*(camadas-1)*2)*3;
11     int tex_pontos=(2*n_pontos)/3;
12
13     float *vertexB=(float*)malloc(n_pontos*sizeof(float)),
14     *normalB=(float*)malloc(n_pontos*sizeof(float)),
15     *texB=(float*)malloc(tex_pontos*sizeof(float));
16
17     int *indices=(int*)malloc(n_indices*sizeof(int));
18     h_aux+=angulo_h;
19
20     //Primeiro ponto central
21     for (l_aux=0; l_aux<=fatias; l_aux++) {
22         vertexB[v++]=0;vertexB[v++]=raio;vertexB[v++]=0;
23         normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
```

```
24          texB[t++]=l_aux*texFactor_fatias;texB[t++]=1;
25      }
26      avanco=fatias+1;
27
28      //Primeiro circulo da esfera
29      for (l_aux=0; l_aux<=fatias; l_aux++) {
30
31          vertexB[v++]=raio*sin(y)*cos(h_aux);vertexB[v++]=raio*sin(h_aux);vertexB[
                v++]=raio*cos(y)*cos(h_aux);
32          normalB[n++]=sin(y)*cos(h_aux);normalB[n++]=sin(h_aux);normalB[n++]=cos(y
                )*cos(h_aux);
33          texB[t++]=l_aux*texFactor_fatias;texB[t++]=1.0f-texFactor_camadas;
34          if(l_aux!=fatias){
35              indices[i++]=l_aux;
36              indices[i++]=avanco+l_aux;
37              indices[i++]=avanco+l_aux+1;
38          }
39          y+=angulo_cir;
40      }
41      avanco+=fatias+1;
42
43
44      //Corpo da esfera
45      for(j=1;j<camadas-1;j++){
46          h_aux+=angulo_h;
47          y=0;
48          for (l_aux=0; l_aux<=fatias; l_aux++) {
49
50
51              vertexB[v++]=raio*sin(y)*cos(h_aux);vertexB[v++]=raio*sin(h_aux);
                    vertexB[v++]=raio*cos(y)*cos(h_aux);
52              normalB[n++]=sin(y)*cos(h_aux);normalB[n++]=sin(h_aux);normalB[n++]=
                    cos(y)*cos(h_aux);
53              texB[t++]=l_aux*texFactor_fatias;texB[t++]=(camadas-(j+1))*
                    texFactor_camadas;
54              if(l_aux!=fatias){
55                  indices[i++]=avanco-(fatias+1)+l_aux;
56                  indices[i++]=avanco+l_aux;
57                  indices[i++]=avanco-(fatias+1)+l_aux+1;
58
59                  indices[i++]=avanco+l_aux;
60                  indices[i++]=avanco+l_aux+1;
61                  indices[i++]=avanco-(fatias+1)+l_aux+1;
62              }
63              y+=angulo_cir;
64          }
65          avanco+=fatias+1;
66      }
67
68      //Ultimo circulo
69      for (l_aux=0; l_aux<=fatias; l_aux++) {
70          vertexB[v++]=0;vertexB[v++]=-raio;vertexB[v++]=0;
71          normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
72          texB[t++]=l_aux*texFactor_fatias;texB[t++]=0;
73
74          if(l_aux!=fatias){
75              indices[i++]=avanco-(fatias+1)+l_aux;
76              indices[i++]=avanco+l_aux;
77
78              indices[i++]=avanco-(fatias+1)+l_aux+1;
79          }
80      }
81
82      //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
83      fprintf(f, "%f %f %f %f %f %f\n",raio, -raio,raio, -raio,raio,-raio);
84
85      //Imprimir os vertices, indices, normais e coordenadas de textura
86      fprintf(f, "%d\n",n_pontos);
87      for(i=0;i<n_pontos;i+=3)
88          fprintf(f, "%f %f %f\n",vertexB[i],vertexB[i+1],vertexB[i+2]);
89
90      fprintf(f, "%d\n",n_indices);
91      for(i=0;i<n_indices;i+=3)
92          fprintf(f, "%d %d %d\n",indices[i],indices[i+1],indices[i+2]);
93
94      for(i=0;i<n_pontos;i+=3)
95          fprintf(f, "%f %f %f\n",normalB[i],normalB[i+1],normalB[i+2]);
96
97      for(i=0;i<tex_pontos ;i+=2)
98          fprintf(f, "%f %f\n",texB[i],texB[i+1]);
99
100     free(vertexB);
101     free(normalB);
102     free(texB);
```

```
103
104 }
```

Listing 11.5: Esfera versão VBO

11.1.6 Plano

```
1  void planoVBO(float altura, float lado, int camadas, int fatias, float z_index,
       int ori, FILE *f){
2      int k=0,j=0,v=0, i=0, avanco,n=0,t=0;
3      float l_const=lado/fatias, alt_const=altura/camadas,alt_ori=-altura/2,
           lado_ori=-lado/2;
4      float texFactor_fatias=1.0f/fatias;
5      float texFactor_camadas=1.0f/camadas;
6
7
8      int n_pontos=(fatias+1)*(camadas+1)*3;
9      int n_indices=(2*fatias*camadas)*3;
10     int tex_pontos=(n_pontos*2)/3;
11
12     float *vertexB=(float*)malloc(n_pontos*sizeof(float)),
13     *normalB=(float*)malloc(n_pontos*sizeof(float)),
14     *texB=(float*)malloc(tex_pontos*sizeof(float));
15
16     int *indices=(int*)malloc(n_indices*sizeof(int));
17
18     switch (ori) {
19         case 1:
20             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                   ViewFrustumCulling
21             fprintf(f, "%f %f %f %f %f %f\n",lado/2.0f,-lado/2.0f,altura/2.0f, -
                   altura/2.0f,z_index, z_index);
22             for(altura=alt_ori;j<=camadas;j++){
23                 k=0;
24                 avanco=j*(fatias+1);
25                 for(lado=lado_ori;k<=fatias;k++){
26                     //Inserir Ponto
27                     vertexB[v++]=lado;vertexB[v++]=altura;vertexB[v++]=z_index;
28                     normalB[n++]=0;normalB[n++]=0;normalB[n++]=1;
29
30                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
31
32                     if(k!=fatias && j!=camadas){
33                         indices[i++]=avanco+k;
34                         indices[i++]=avanco+k+1;
35                         indices[i++]=avanco+fatias+1+k;
36
37                         indices[i++]=avanco+k+1;
38                         indices[i++]=avanco+fatias+1+k+1;
39                         indices[i++]=avanco+fatias+1+k;
40
41
42                     }
43                     lado+=l_const;
44                 }
45                 altura+=alt_const;
46             }
47             break;
48         case 2:
49             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                   ViewFrustumCulling
50             fprintf(f, "%f %f %f %f %f %f\n",lado/2.0f,-lado/2.0f,altura/2.0f, -
                   altura/2.0f,z_index, z_index);
51             for(altura=alt_ori;j<=camadas;j++){
52                 k=0;
53                 avanco=j*(fatias+1);
54                 for(lado=lado_ori;k<=fatias;k++){
55                     //Inserir Ponto
56                     vertexB[v++]=lado;vertexB[v++]=altura;vertexB[v++]=z_index;
57                     normalB[n++]=0;normalB[n++]=0;normalB[n++]=-1;
58                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
59                     if(k!=fatias && j!=camadas){
60                         indices[i++]=avanco+k;
61                         indices[i++]=avanco+fatias+1+k;
62                         indices[i++]=avanco+k+1;
63
64                         indices[i++]=avanco+k+1;
65                         indices[i++]=avanco+fatias+1+k;
66                         indices[i++]=avanco+fatias+1+k+1;
```

```
67                      }
68                      lado+=l_const;
69                  }
70                  altura+=alt_const;
71              }
72              break;
73
74          case 3:
75              //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                    ViewFrustumCulling
76              fprintf(f, "%f %f %f %f %f %f\n",z_index, z_index,altura/2.0f, -
                    altura/2.0f,lado/2.0f,-lado/2.0f);
77              for(altura=alt_ori;j<=camadas;j++){
78                  k=0;
79                  avanco=j*(fatias+1);
80                  for(lado=lado_ori;k<=fatias;k++){
81                      //Inserir Ponto
82                      vertexB[v++]=z_index;vertexB[v++]=altura;vertexB[v++]=lado;
83                      normalB[n++]=1;normalB[n++]=0;normalB[n++]=0;
84                      texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
85                      if(k!=fatias && j!=camadas){
86                          indices[i++]=avanco+k;
87                          indices[i++]=avanco+fatias+1+k;
88                          indices[i++]=avanco+k+1;
89
90                          indices[i++]=avanco+k+1;
91                          indices[i++]=avanco+fatias+1+k;
92                          indices[i++]=avanco+fatias+1+k+1;
93                      }
94                      lado+=l_const;
95                  }
96                  altura+=alt_const;
97              }
98              break;
99          case 4:
100             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                    ViewFrustumCulling
101             fprintf(f, "%f %f %f %f %f %f\n",z_index, z_index,altura/2.0f, -
                    altura/2.0f,lado/2.0f,-lado/2.0f);
102             for(altura=alt_ori;j<=camadas;j++){
103                 k=0;
104                 avanco=j*(fatias+1);
105                 for(lado=lado_ori;k<=fatias;k++){
106                     //Inserir Ponto
107                     vertexB[v++]=z_index;vertexB[v++]=altura;vertexB[v++]=lado;
108                     normalB[n++]=-1;normalB[n++]=0;normalB[n++]=0;
109                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
110                     if(k!=fatias && j!=camadas){
111                         indices[i++]=avanco+k;
112                         indices[i++]=avanco+k+1;
113                         indices[i++]=avanco+fatias+1+k;
114
115                         indices[i++]=avanco+k+1;
116                         indices[i++]=avanco+fatias+1+k+1;
117                         indices[i++]=avanco+fatias+1+k;
118                     }
119                     lado+=l_const;
120                 }
121                 altura+=alt_const;
122             }
123             break;
124         case 5:
125             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                    ViewFrustumCulling
126             fprintf(f, "%f %f %f %f %f %f\n",altura/2.0f, -altura/2.0f,z_index,
                    z_index,lado/2.0f,-lado/2.0f);
127             for(altura=alt_ori;j<=camadas;j++){
128                 k=0;
129                 avanco=j*(fatias+1);
130                 for(lado=lado_ori;k<=fatias;k++){
131                     //Inserir Ponto
132                     vertexB[v++]=altura;vertexB[v++]=z_index;vertexB[v++]=lado;
133                     normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
134                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
135                     if(k!=fatias && j!=camadas){
136                         indices[i++]=avanco+k;
137                         indices[i++]=avanco+k+1;
138                         indices[i++]=avanco+fatias+1+k;
139
140                         indices[i++]=avanco+k+1;
141                         indices[i++]=avanco+fatias+1+k+1;
142                         indices[i++]=avanco+fatias+1+k;
143                     }
144                     lado+=l_const;
```

```
145                 }
146                 altura+=alt_const;
147             }
148             break;
149         case 6:
150             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                    ViewFrustumCulling
151             fprintf(f, "%f %f %f %f %f %f\n",altura/2.0f, -altura/2.0f,z_index,
                    z_index,lado/2.0f,-lado/2.0f);
152             for(altura=alt_ori;j<=camadas;j++){
153                 k=0;
154                 avanco=j*(fatias+1);
155                 for(lado=lado_ori;k<=fatias;k++){
156                     //Inserir Ponto
157                     vertexB[v++]=altura;vertexB[v++]=z_index;vertexB[v++]=lado;
158                     normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
159                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
160                     if(k!=fatias && j!=camadas){
161                         indices[i++]=avanco+k;
162                         indices[i++]=avanco+fatias+1+k;
163                         indices[i++]=avanco+k+1;
164
165                         indices[i++]=avanco+k+1;
166                         indices[i++]=avanco+fatias+1+k;
167                         indices[i++]=avanco+fatias+1+k+1;
168                     }
169                     lado+=l_const;
170                 }
171                 altura+=alt_const;
172             }
173             break;
174
175
176     }
177
178
179     //Imprimir os vertices, indices, normais e coordenadas de textura
180     fprintf(f, "%d\n",n_pontos);
181     for(i=0;i<n_pontos;i+=3)
182         fprintf(f, "%f %f %f\n",vertexB[i],vertexB[i+1],vertexB[i+2]);
183
184     fprintf(f, "%d\n",n_indices);
185     for(i=0;i<n_indices;i+=3)
186         fprintf(f, "%d %d %d\n",indices[i],indices[i+1],indices[i+2]);
187
188     for(i=0;i<n_pontos;i+=3)
189         fprintf(f, "%f %f %f\n",normalB[i],normalB[i+1],normalB[i+2]);
190
191     for(i=0;i<tex_pontos ;i+=2)
192         fprintf(f, "%f %f\n",texB[i],texB[i+1]);
193
194     free(vertexB);
195     free(normalB);
196     free(texB);
197
198 }
```

Listing 11.6: Plano Versão VBO

11.1.7 Paralelepípedo

```
1  void paralelepipedoVBO(float lado_y, float lado_x, float lado_z, int camadas, int
       fatias_x, int fatias_z, FILE *f){
2      int      k=0,j=0,v=0, i=0, avanco=0,n=0,t=0;
3      float    change_x=lado_x/fatias_x,
4               change_z=lado_z/fatias_z,
5               change_y=lado_y/camadas,
6               begin_y=-lado_y/2,
7               begin_x=-lado_x/2,
8               begin_z=-lado_z/2,
9               altura_aux,lado_aux;
10     float texFactor_fatias_x=1.0f/fatias_x;
11     float texFactor_fatias_z=1.0f/fatias_z;
12     float texFactor_fatias_y=1.0f/camadas;
13
14
15     int n_pontos=(2*(fatias_x+1)*(camadas+1) + 2*(fatias_z+1)*(camadas+1) + 2*(
           fatias_x+1)*(fatias_z+1))*3;
```

```
16      int n_indices=(2*fatias_x*camadas + 2*fatias_z*camadas + 2* fatias_x*fatias_z
            )*3*2;
17      int tex_pontos=(n_pontos*2)/3;
18
19      float *vertexB=(float*)malloc(n_pontos*sizeof(float)),
20      *normalB=(float*)malloc(n_pontos*sizeof(float)),
21      *texB=(float*)malloc(tex_pontos*sizeof(float));
22
23      int *indices=(int*)malloc(n_indices*sizeof(int));
24
25      //1
26      for(altura_aux=begin_y;j<=camadas;j++){
27          k=0;
28
29          for(lado_aux=begin_x;k<=fatias_x;k++){
30              //Inserir Ponto
31              vertexB[v++]=lado_aux;vertexB[v++]=altura_aux;vertexB[v++]=fabs(
                    begin_z);
32              normalB[n++]=0;normalB[n++]=0;normalB[n++]=1;
33              texB[t++]=k*texFactor_fatias_x;texB[t++]=j*texFactor_fatias_y;
34              if(k!=fatias_x && j!=camadas){
35                  indices[i++]=avanco+k;
36                  indices[i++]=avanco+k+1;
37                  indices[i++]=avanco+fatias_x+1+k;
38
39                  indices[i++]=avanco+k+1;
40                  indices[i++]=avanco+fatias_x+1+k+1;
41                  indices[i++]=avanco+fatias_x+1+k;
42              }
43              lado_aux+=change_x;
44          }
45          avanco+=fatias_x+1;
46          altura_aux+=change_y;
47      }
48
49      //2
50      j=0;
51      for(altura_aux=begin_y;j<=camadas;j++){
52          k=0;
53          for(lado_aux=begin_x;k<=fatias_x;k++){
54              //Inserir Ponto
55              vertexB[v++]=lado_aux;vertexB[v++]=altura_aux;vertexB[v++]=begin_z;
56              normalB[n++]=0;normalB[n++]=0;normalB[n++]=-1;
57              texB[t++]=k*texFactor_fatias_x;texB[t++]=j*texFactor_fatias_y;
58              if(k!=fatias_x && j!=camadas){
59                  indices[i++]=avanco+k;
60                  indices[i++]=avanco+fatias_x+1+k;
61                  indices[i++]=avanco+k+1;
62
63                  indices[i++]=avanco+k+1;
64                  indices[i++]=avanco+fatias_x+1+k;
65                  indices[i++]=avanco+fatias_x+1+k+1;
66              }
67              lado_aux+=change_x;
68          }
69          avanco+=fatias_x+1;
70          altura_aux+=change_y;
71      }
72
73      //3
74      j=0;
75      for(altura_aux=begin_y;j<=camadas;j++){
76          k=0;
77          for(lado_aux=begin_z;k<=fatias_z;k++){
78              //Inserir Ponto
79              vertexB[v++]=fabs(begin_x);vertexB[v++]=altura_aux;vertexB[v++]=
                    lado_aux;
80              normalB[n++]=1;normalB[n++]=0;normalB[n++]=0;
81              texB[t++]=k*texFactor_fatias_z;texB[t++]=j*texFactor_fatias_y;
82              if(k!=fatias_z && j!=camadas){
83                  indices[i++]=avanco+k;
84                  indices[i++]=avanco+fatias_z+1+k;
85                  indices[i++]=avanco+k+1;
86
87                  indices[i++]=avanco+k+1;
88                  indices[i++]=avanco+fatias_z+1+k;
89                  indices[i++]=avanco+fatias_z+1+k+1;
90              }
91              lado_aux+=change_z;
92          }
93          avanco+=fatias_z+1;
94          altura_aux+=change_y;
95      }
96
```

```
97      //4
98      j=0;
99      for(altura_aux=begin_y;j<=camadas;j++){
100         k=0;
101         for(lado_aux=begin_z;k<=fatias_z;k++){
102             //Inserir Ponto
103             vertexB[v++]=begin_x;vertexB[v++]=altura_aux;vertexB[v++]=lado_aux;
104             normalB[n++]=-1;normalB[n++]=0;normalB[n++]=0;
105             texB[t++]=k*texFactor_fatias_z;texB[t++]=j*texFactor_fatias_y;
106             if(k!=fatias_z && j!=camadas){
107                 indices[i++]=avanco+k;
108                 indices[i++]=avanco+k+1;
109                 indices[i++]=avanco+fatias_z+1+k;
110
111                 indices[i++]=avanco+k+1;
112                 indices[i++]=avanco+fatias_z+1+k+1;
113                 indices[i++]=avanco+fatias_z+1+k;
114
115             }
116             lado_aux+=change_z;
117         }
118         avanco+=fatias_z+1;
119         altura_aux+=change_y;
120     }
121
122     //5
123     j=0;
124     for(altura_aux=begin_x;j<=fatias_x;j++){
125         k=0;
126         for(lado_aux=begin_z;k<=fatias_z;k++){
127             //Inserir Ponto
128             vertexB[v++]=altura_aux;vertexB[v++]=fabsf(begin_y);vertexB[v++]=
                    lado_aux;
129             normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
130             texB[t++]=k*texFactor_fatias_z;texB[t++]=j*texFactor_fatias_x;
131             if(k!=fatias_z && j!=fatias_x){
132                 indices[i++]=avanco+k;
133                 indices[i++]=avanco+k+1;
134                 indices[i++]=avanco+fatias_z+1+k;
135
136                 indices[i++]=avanco+k+1;
137                 indices[i++]=avanco+fatias_z+1+k+1;
138                 indices[i++]=avanco+fatias_z+1+k;
139
140             }
141             lado_aux+=change_z;
142         }
143         avanco+=fatias_z+1;
144         altura_aux+=change_x;
145     }
146
147     //6
148     j=0;
149     for(altura_aux=begin_x;j<=fatias_x;j++){
150         k=0;
151         for(lado_aux=begin_z;k<=fatias_z;k++){
152             //Inserir Ponto
153             vertexB[v++]=altura_aux;vertexB[v++]=begin_y;vertexB[v++]=lado_aux;
154             normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
155             texB[t++]=k*texFactor_fatias_z;texB[t++]=j*texFactor_fatias_x;
156             if(k!=fatias_z && j!=fatias_x){
157                 indices[i++]=avanco+k;
158                 indices[i++]=avanco+fatias_z+1+k;
159                 indices[i++]=avanco+k+1;
160
161                 indices[i++]=avanco+k+1;
162                 indices[i++]=avanco+fatias_z+1+k;
163                 indices[i++]=avanco+fatias_z+1+k+1;
164             }
165             lado_aux+=change_z;
166         }
167         avanco+=fatias_z+1;
168         altura_aux+=change_x;
169     }
170
171
172     //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o ViewFrustumCulling
173     fprintf(f, "%f %f %f %f %f %f\n",lado_x/2.0f,-lado_x/2.0f,lado_y/2.0f,-lado_y
            /2.0f,lado_z/2.0f,-lado_z/2.0f);
174
175     //Imprimir os vertices, indices, normais e coordenadas de textura
176     fprintf(f, "%d\n",n_pontos);
177     for(i=0;i<n_pontos;i+=3)
178         fprintf(f, "%f %f %f\n",vertexB[i],vertexB[i+1],vertexB[i+2]);
```

```
179
180     fprintf(f, "%d\n",n_indices);
181     for(i=0;i<n_indices;i+=3)
182         fprintf(f, "%d %d %d\n",indices[i],indices[i+1],indices[i+2]);
183
184     for(i=0;i<n_pontos;i+=3)
185         fprintf(f, "%f %f %f\n",normalB[i],normalB[i+1],normalB[i+2]);
186
187     for(i=0;i<tex_pontos ;i+=2)
188         fprintf(f, "%f %f\n",texB[i],texB[i+1]);
189
190     free(vertexB);
191     free(normalB);
192     free(texB);
193
194 }
```

Listing 11.7: Paralelepipedo versão VBO

11.1.8 Pacth

```
1  void planoVBO(float altura, float lado, int camadas, int fatias, float z_index,
       int ori, FILE *f){
2      int k=0,j=0,v=0, i=0, avanco,n=0,t=0;
3      float l_const=lado/fatias, alt_const=altura/camadas,alt_ori=-altura/2,
           lado_ori=-lado/2;
4      float texFactor_fatias=1.0f/fatias;
5      float texFactor_camadas=1.0f/camadas;
6
7
8      int n_pontos=(fatias+1)*(camadas+1)*3;
9      int n_indices=(2*fatias*camadas)*3;
10     int tex_pontos=(n_pontos*2)/3;
11
12     float *vertexB=(float*)malloc(n_pontos*sizeof(float)),
13     *normalB=(float*)malloc(n_pontos*sizeof(float)),
14     *texB=(float*)malloc(tex_pontos*sizeof(float));
15
16     int *indices=(int*)malloc(n_indices*sizeof(int));
17
18     switch (ori) {
19         case 1:
20             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                   ViewFrustumCulling
21             fprintf(f, "%f %f %f %f %f %f\n",lado/2.0f,-lado/2.0f,altura/2.0f, -
                   altura/2.0f,z_index, z_index);
22             for(altura=alt_ori;j<=camadas;j++){
23                 k=0;
24                 avanco=j*(fatias+1);
25                 for(lado=lado_ori;k<=fatias;k++){
26                     //Inserir Ponto
27                     vertexB[v++]=lado;vertexB[v++]=altura;vertexB[v++]=z_index;
28                     normalB[n++]=0;normalB[n++]=0;normalB[n++]=1;
29
30                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
31
32                     if(k!=fatias && j!=camadas){
33                         indices[i++]=avanco+k;
34                         indices[i++]=avanco+k+1;
35                         indices[i++]=avanco+fatias+1+k;
36
37                         indices[i++]=avanco+k+1;
38                         indices[i++]=avanco+fatias+1+k+1;
39                         indices[i++]=avanco+fatias+1+k;
40
41
42                     }
43                     lado+=l_const;
44                 }
45                 altura+=alt_const;
46             }
47             break;
48         case 2:
49             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                   ViewFrustumCulling
50             fprintf(f, "%f %f %f %f %f %f\n",lado/2.0f,-lado/2.0f,altura/2.0f, -
                   altura/2.0f,z_index, z_index);
51             for(altura=alt_ori;j<=camadas;j++){
52                 k=0;
```

```
53                  avanco=j*(fatias+1);
54                  for(lado=lado_ori;k<=fatias;k++){
55                      //Inserir Ponto
56                      vertexB[v++]=lado;vertexB[v++]=altura;vertexB[v++]=z_index;
57                      normalB[n++]=0;normalB[n++]=0;normalB[n++]=-1;
58                      texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
59                      if(k!=fatias && j!=camadas){
60                          indices[i++]=avanco+k;
61                          indices[i++]=avanco+fatias+1+k;
62                          indices[i++]=avanco+k+1;
63
64                          indices[i++]=avanco+k+1;
65                          indices[i++]=avanco+fatias+1+k;
66                          indices[i++]=avanco+fatias+1+k+1;
67                      }
68                      lado+=l_const;
69                  }
70                  altura+=alt_const;
71              }
72              break;
73
74          case 3:
75              //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                    ViewFrustumCulling
76              fprintf(f, "%f %f %f %f %f %f\n",z_index, z_index,altura/2.0f, -
                    altura/2.0f,lado/2.0f,-lado/2.0f);
77              for(altura=alt_ori;j<=camadas;j++){
78                  k=0;
79                  avanco=j*(fatias+1);
80                  for(lado=lado_ori;k<=fatias;k++){
81                      //Inserir Ponto
82                      vertexB[v++]=z_index;vertexB[v++]=altura;vertexB[v++]=lado;
83                      normalB[n++]=1;normalB[n++]=0;normalB[n++]=0;
84                      texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
85                      if(k!=fatias && j!=camadas){
86                          indices[i++]=avanco+k;
87                          indices[i++]=avanco+fatias+1+k;
88                          indices[i++]=avanco+k+1;
89
90                          indices[i++]=avanco+k+1;
91                          indices[i++]=avanco+fatias+1+k;
92                          indices[i++]=avanco+fatias+1+k+1;
93                      }
94                      lado+=l_const;
95                  }
96                  altura+=alt_const;
97              }
98              break;
99          case 4:
100             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                    ViewFrustumCulling
101             fprintf(f, "%f %f %f %f %f %f\n",z_index, z_index,altura/2.0f, -
                    altura/2.0f,lado/2.0f,-lado/2.0f);
102             for(altura=alt_ori;j<=camadas;j++){
103                 k=0;
104                 avanco=j*(fatias+1);
105                 for(lado=lado_ori;k<=fatias;k++){
106                     //Inserir Ponto
107                     vertexB[v++]=z_index;vertexB[v++]=altura;vertexB[v++]=lado;
108                     normalB[n++]=-1;normalB[n++]=0;normalB[n++]=0;
109                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
110                     if(k!=fatias && j!=camadas){
111                         indices[i++]=avanco+k;
112                         indices[i++]=avanco+k+1;
113                         indices[i++]=avanco+fatias+1+k;
114
115                         indices[i++]=avanco+k+1;
116                         indices[i++]=avanco+fatias+1+k+1;
117                         indices[i++]=avanco+fatias+1+k;
118                     }
119                     lado+=l_const;
120                 }
121                 altura+=alt_const;
122             }
123             break;
124         case 5:
125             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                    ViewFrustumCulling
126             fprintf(f, "%f %f %f %f %f %f\n",altura/2.0f, -altura/2.0f,z_index,
                    z_index,lado/2.0f,-lado/2.0f);
127             for(altura=alt_ori;j<=camadas;j++){
128                 k=0;
129                 avanco=j*(fatias+1);
130                 for(lado=lado_ori;k<=fatias;k++){
```

```
131                     //Inserir Ponto
132                     vertexB[v++]=altura;vertexB[v++]=z_index;vertexB[v++]=lado;
133                     normalB[n++]=0;normalB[n++]=1;normalB[n++]=0;
134                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
135                     if(k!=fatias && j!=camadas){
136                         indices[i++]=avanco+k;
137                         indices[i++]=avanco+k+1;
138                         indices[i++]=avanco+fatias+1+k;
139
140                         indices[i++]=avanco+k+1;
141                         indices[i++]=avanco+fatias+1+k+1;
142                         indices[i++]=avanco+fatias+1+k;
143                     }
144                     lado+=l_const;
145                 }
146                 altura+=alt_const;
147             }
148             break;
149         case 6:
150             //Imprimir maxX, minX, maxY, minY, maxZ, minZ para o
                      ViewFrustumCulling
151             fprintf(f, "%f %f %f %f %f %f\n",altura/2.0f, -altura/2.0f,z_index,
                      z_index,lado/2.0f,-lado/2.0f);
152             for(altura=alt_ori;j<=camadas;j++){
153                 k=0;
154                 avanco=j*(fatias+1);
155                 for(lado=lado_ori;k<=fatias;k++){
156                     //Inserir Ponto
157                     vertexB[v++]=altura;vertexB[v++]=z_index;vertexB[v++]=lado;
158                     normalB[n++]=0;normalB[n++]=-1;normalB[n++]=0;
159                     texB[t++]=k*texFactor_fatias;texB[t++]=j*texFactor_camadas;
160                     if(k!=fatias && j!=camadas){
161                         indices[i++]=avanco+k;
162                         indices[i++]=avanco+fatias+1+k;
163                         indices[i++]=avanco+k+1;
164
165                         indices[i++]=avanco+k+1;
166                         indices[i++]=avanco+fatias+1+k;
167                         indices[i++]=avanco+fatias+1+k+1;
168                     }
169                     lado+=l_const;
170                 }
171                 altura+=alt_const;
172             }
173             break;
174
175
176     }
177
178
179     //Imprimir os vertices, indices, normais e coordenadas de textura
180     fprintf(f, "%d\n",n_pontos);
181     for(i=0;i<n_pontos;i+=3)
182         fprintf(f, "%f %f %f\n",vertexB[i],vertexB[i+1],vertexB[i+2]);
183
184     fprintf(f, "%d\n",n_indices);
185     for(i=0;i<n_indices;i+=3)
186         fprintf(f, "%d %d %d\n",indices[i],indices[i+1],indices[i+2]);
187
188     for(i=0;i<n_pontos;i+=3)
189         fprintf(f, "%f %f %f\n",normalB[i],normalB[i+1],normalB[i+2]);
190
191     for(i=0;i<tex_pontos ;i+=2)
192         fprintf(f, "%f %f\n",texB[i],texB[i+1]);
193
194     free(vertexB);
195     free(normalB);
196     free(texB);
197
198 }
```

Listing 11.8: Patch versão VBO