Computação Gráfica

Relatório

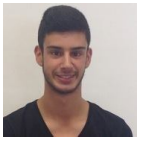
1º Fase - Primitivas gráficas

Ana Paula Carvalho
a61855

Bruno Manuel Arieira
a70565

João Miguel Palmeira
a73864

Pedro Manuel Almeida
a74301



8 de Abril de 2018

Conteúdo

Lista de Figuras

1 Introdução

No âmbito desta segunda fase de projeto introduziram-se, dispondo dos conteúdos leccionados nas aulas práticas, as transformações geométricas. Os ficheiros XML a serem utilizados encontram-se agora sujeitos a um modelo hierárquico, em árvore, na qual cada nodo possui as suas transformações, o seu conjunto de modelos e seus filhos. Qualquer transformação num nodo pertencente a esta árvore produz uma cadeia de efeitos, afetando cada um dos nodos filho – evidenciando a hierarquia das transformações na árvore.

Ao longo deste relatório, irá ser explicitado este conceito aplicando-o na elaboração de um Sistema Solar à escala.

2 Gerador

O gerador utilizado para esta fase foi essencialmente o da última – foram apenas acrescentadas funcionalidades complementares de desenho de novas formas necessárias para a construção do Sistema Solar. O ficheiro *.3d* é o componente principal do ficheiro *XML*, que contém os dados referentes aos constituintes do Sistema Solar. A única primitiva a ser usada é a esfera, que vai sofrendo redimensionamentos conforme a necessidade para formar cada um dos planetas.

3 Motor

3.1 Estruturas de Dados

Como para a realização desta nova tarefa houve a necessidade de guardar mais dados, reformulámos a nossa estrutura de dados.

3.1.1 Ponto

Esta classe define um ponto no espaço 3d, com as respetivas coordenadas x, y e z.

```
class Ponto{
public:
    float coordx;
    float coordy;
    float coordz;
};
```

3.1.2 Cor

Esta classe define as cores que utilizamos para os objetos, que é constituida por três variáveis *(red, green, blue)* do tipo float. O tipo destas variáveis foram escolhidos de forma a permitir que as variáveis possam ser passadas diretamente ás funcões do OpenGL necessárias á representação de uma cor no sistema RGB.

```
class Cor{
public:
    float r,g,b;
};
```

3.1.3 Órbita

Cada planeta tem a sua órbita (em volta do sol), assim como cada satélite tem a sua (em volta do respetivo planeta). Como a órbita representada geometricamente é uma elíptica, necessitamos de dois eixos para determinar os pontos desta, sendo que criamos dois floats que correspondem aos eixos.

```
class Orbita{
public:
    float raioX,raioZ;
};
```

3.1.4 Modelo

Esta classe já a utilizamos no trabalho prático anterior, em que se baseia numa lista de objetos *Ponto* (conjunto de vários pontos), que faz referência a uma determinada figura geométrica ou modelo, que declaramos como uma String que tem implicito o nome do ficheiro.

```
class Modelo{
public:
    string ficheiro;
    list<Ponto> pontos;
};
```

3.1.5 Specs

Classe que traduz os três tipos de transformações que usamos para representar o Sistema Solar. As primeiras três variáveis declaradas (*coordx, coordy* e *coordz*) servem de auxílio á translação que se relaciona em mover os eixos para determinada posição. De seguida, temos outro tipo de transformação designado rotação, em que nos permite mover um objeto com um determinado ângulo e direção. Por último, fazemos referência á escala, que podemos alterar as dimensões do objeto em questão.

```
class Specs{
public:
    float coordx, coordy, coordz;
    float angulo,eixox,eixoy,eixoz;
    float escala_x,escala_y,escala_z;
};
```

3.1.6 Grupo

Nesta classe está imposta todas as outras anteriores, excepto a String *tipo*, que indica ao sistema se refere um planeta ou uma órbita e a lista grupos.

Esta lista pertence a esta classe, que indica recursividade e profundidade, dando ênfase aos nodos de uma árvore, podendo haver nodos filhos num determinado grupo.

```
class Grupo{
public:
    string tipo; // planeta ou orbita
    Specs specs;
    Cor cor;
    Orbita orbita;
    list<Modelo> modelos;
    list<Grupo> grupos;
};
```

3.1.7 Figura

Classe que contém toda a informação relativa ao sistema solar, pois reúne todos os grupos existente no sistema, sendo esta considerada uma classe geral.

```
class Figura{
public:
    list<Grupo> grupos;
    void load(const char* pFilename);
};
```

Como cada grupo pode conter outros grupos, o carregamento dos ficheiros *XML* é feito recursivamente, assim como o desenho da maqueta. Este é feito para cada grupo, onde primeiro são realizadas as transformações geométricas necessárias e depois é que são desenhados os respetivos vértices.
A primitiva para desenho das órbitas é feita no motor (*main*), pela funcao *drawOrbit*, que com os devidos raios (*raioX* e *raioZ*), desenha uma elipse segundo o plano XZ, no modo *GL_LINE_STRIP*. Desta forma, com a elipse obtida é representada a informação essencial do Grupo em relação ao seu centro.

3.2 XML

Nesta fase foi necessário organizar o ficheiro XML de forma hierárquica, isto é, implementamos um conjunto de tags de maneira a que o ficheiro ficasse organizado sergundo o modelo Hierárquico em Árvore permitindo com

que, quando o ficheiro XML for lido, os seus elementos serão carregados de uma forma mais rápida e o mais eficiete possível.
Posto isto, com o intuito de apresentar a estratégia utilizada na elaboração do ficheiro XML, colocamos abaixo um excerto do ficheiro para o Sistema Solar relativo ao planeta Terra, a sua órbita e o seu satélite natural (a Lua):

```
<grupo>
    <specs angulo=20 eixoY=1 />
       <grupo>
        <specs X=0.999 />
        <cor R=0.3 G=0.3 B=1 />
        <orbita raioX=60 raioZ=59.99 />
            <!-- terra -->
            <grupo>
                <specs X=60 angulo=23.45 eixoZ=1 escala_x=0.137 escala_y=0.137
                escala_z=0.137 />
                <cor R=0.3 G=0.3 B=1 />
                <modelos>
                    <modelo ficheiro="./SistemaSolar/esferaprimitiva.3d" />
                </modelos>

                    <!-- orbita lua -->
                    <grupo>
                        <specs X=0.199 angulo=5.14 eixoZ=1 />
                        <cor R=0.7 G=0.7 B=0.7 />
                        <orbita raioX=2 raioZ=1.99 />

                        <!-- lua -->
                        <grupo>
                            <specs X=2 angulo=6.68 eixoZ=1 escala_x=0.037 escala_y=0.037
                            escala_z=0.037 />
                            <cor R=0.7 G=0.7 B=0.7 />
                            <modelos>
                                <modelo ficheiro="./SistemaSolar/esferaprimitiva.3d" />
                            </modelos>
                        </grupo>
```

```
            </grupo>
        </grupo>
    </grupo>
</grupo>
```

Apresentamos agora algumas tags utilizadas para a criação do ficheiro anterior:

- `<figura> </figura>`
- `<grupo> </grupo>`
- `<specs angulo=a eixoZ=eZ escala x=esX escala y=esY escala z=esZ />`
- `<cor R=r G=g B=b/>`
- `<orbita raioX=rx raioZ=rz/>`
- `<modelo> </modelo>`

3.3 Rendering

Após ser feita a leitura do XML e armazenada a informação nas estruturas de dados, por forma a desenhar o sistema solar, é feita, na função renderScene(), a travessia da árvore onde estes elementos estão guardados. Desta feita, iniciamos um processo recursivo na função renderScene() invocando a função renderGrupo() para os grupos (filhos).

```
void renderScene(void){
        ...
        for(list<Grupo>::iterator itg = figura.grupos.begin();
                                        itg != figura.grupos.end(); itg++)
            renderGrupo(itg);
        ...
}
```

3.3.1 renderGrupo

Esta função recebe como argumento um grupo aplicando-lhe posteriormente as suas transformações, percorrendos os seus modelos e desenhando os

respetivos pontos.

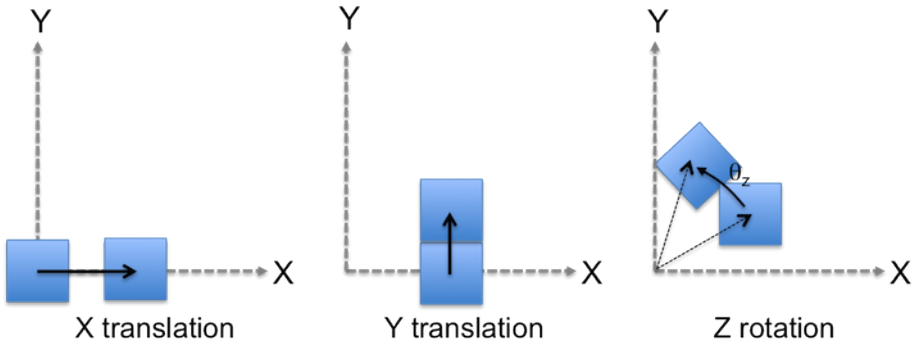


Figura 1: Diferença de transformações

Após isto, elabora uma chamada recursiva para os seus grupos - filho, isto é, é necessário que as transformações de um grupo afetem os seus grupos-filho e, para tal, realizamos um push e um pop no início da função, respetivamente (incluindo a chamada recursiva explicada anteriormente).
Através da variável tipo na classe Grupo conseguimos facilmente compreender se se vai proceder o *render* de uma órbita ou o *render* de uma planeta, uma vez que a ordem de transformações é diferente em cada um dos casos.

Implementação do render por Grupo

```
void renderGrupo(list<Grupo>::iterator g){
    glPushMatrix();

if(!strcmp(g->tipo.c_str(),"orbita")){
        glRotatef(g->specs.angulo, g->specs.eixox, g->specs.eixoy, g->specs.eixoz);
        glTranslatef(g->specs.coordx, g->specs.coordy, g->specs.coordz);
    }
    else{
        glTranslatef(g->specs.coordx, g->specs.coordy, g->specs.coordz);
        glRotatef(g->specs.angulo, g->specs.eixox, g->specs.eixoy, g->specs.eixoz);
    }
    glScalef(g->specs.escala_x, g->specs.escala_y, g->specs.escala_z);
    glColor3f(g->cor.r,g->cor.g,g->cor.b);
```

```
        //Função que desenha as elipses das orbitas
        drawOrbit(g->orbita.raioX,g->orbita.raioZ);

        for(list<Modelo>::iterator itm = g->modelos.begin();
                                        itm != g->modelos.end(); itm++){
            glBegin(GL_TRIANGLES);
        for(list<Ponto>::iterator itp = itm->pontos.begin();
                                        itp != itm->pontos.end(); itp++){
            glVertex3f(itp->coordx,itp->coordy,itp->coordz);
        }
        glEnd();
    }
        for(list<Grupo>::iterator itg = g->grupos.begin(); itg != g->grupos.end(); itg++)
            renderGrupo(itg);

    glPopMatrix();

}
```

3.4 Parsing

Como já foi falado anteriormente, nesta segunda parte foi necessário definirmos novas estratégias para satisfazer os requisitos impostos. Uma dessas novas estratégias foi realizar alterações ao nível das estruturas (principalmente com a adição de novas estruturas) o que levou à modificação da função load, isto é, elaboramos a função GRload.
Esta função percorre os filhos de cada grupo do *currentNode* e coloca os grupos carregados na lista dos grupos (lista de grupos da classe Figura). Depois de um grupo ser percorrido é feita uma chamada recursiva através do nodo desse mesmo grupo e com a lista grupo que lhe pertence. Essa chamada recursiva é realizada através da invocação da função GRload no nodo figura (o primeiro nodo do ficheiro XML).

```
void Figura::load(const char* pFilename){
    ...
    GRload(nRoot,&this->grupos);
}
```

```
void GRload(TiXmlNode *currentNode, list<Grupo> *grupos){
    //percorre os grupos
    {
        Grupo g;
        //specs
        Specs t;
        if(tNode){
            /* Parsing do Nodo */
        }
        else { t.coordx = 0; t.coordy = 0; t.coordz = 0; t.angulo=0; t.eixox=0; t.eixoy=0; t
        g.specs = t;

        //cor
        Cor c;
        /* Parsing do Nodo */
        }
        else { c.r = 1; c.g = 1; c.b = 1; }
        g.cor = c;

        //orbita
        Orbita o;
        if(oNode){
            /* Parsing do Nodo */
        }
        else { o.raioX = 0; o.raioZ = 0; g.tipo = "planeta"; }
        g.orbita = o;

        //modelos
        TiXmlNode* mNode = gNode->FirstChild("modelos");
        if(mNode){

            if(testNode){
                modElem = modElem->NextSiblingElement()){
                    Modelo m;
                    fstream f;
                    f.open(m.ficheiro.c_str());
```

```
                    if(f.is_open()){
                        string line;
                        while (getline(f,line)){
                            float coordx,coordy,coordz;
                            sscanf(line.c_str(),"\%f \%f \%f\n",&coordx,&coordy,&coordz);
                            Ponto p;
                            p.coordx = coordx; p.coordy = coordy; p.coordz =coordz;
                            m.pontos.push_back(p);
                        }
                        f.close();
                    }
                    g.modelos.push_back(m);
                }
            }
        }
        if(gNode) GRload(gNode,&g.grupos);
        grupos->push_back(g);
    }
}
```

4 Funcionalidades extra

4.1 Anel

Decidimos incluir já nesta fase os anéis para alguns planetas e, para tal, criamos a seguinte função:

```
void drawAnel(float in_r , float out_r ,int slices, char* filename){
    FILE* f;
    char* aux = (char*)malloc(sizeof(char)*64);
    strcpy(aux, filename);
    strcat(aux, ".3d");
    f = fopen(aux,"w");

    float slice = (360/slices)*(M_PI/180);
    float doisPi = 2*M_PI;
    if (f!=NULL){
    for (float i = 0; i < doisPi; i += slice){
        fprintf(f , "\%f \%f \%f \n" ,sin(i)*in_r, 0.0, cos(i)*in_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i)*out_r, 0.0, cos(i)*out_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i+slice)*in_r, 0.0,    cos(i+slice)*in_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i)*out_r, 0.0, cos(i)*out_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i+slice)*out_r, 0.0,    cos(i+slice)*out_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i+slice)*in_r, 0.0,    cos(i+slice)*in_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i)*out_r, 0.0, cos(i)*out_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i)*in_r, 0.0, cos(i)*in_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i+slice)*in_r, 0.0,    cos(i+slice)*in_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i+slice)*out_r, 0.0,    cos(i+slice)*out_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i)*out_r, 0.0, cos(i)*out_r);
        fprintf(f , "\%f \%f \%f \n" ,sin(i+slice)*in_r, 0.0,    cos(i+slice)*in_r);
        }
   }
}
```

Para o desenho do anel foi necessário limitar tanto o raio interior bem como o raio exterior e ainda o número de fatias a ser utilizadas para a criação do mesmo tal como podemos observar pelo código acima.

4.2 XML

Uma vez que na adicionamos novas estruturas tais como:

- Cor (Define uma cor a um grupo);
- Orbita (Define uma órbita elítica);
- Modelo (Define uma lista de pontos de uma figura);
- Specs (Define a posição e a escala de um Planeta);
- Grupo (Define um Planeta ou uma Órbita);

Foi necessário adicionar novos parâmetros ao *XML*:

- `<cor R=r G=g B=b />;`
- `<orbita raioX=r x raioZ=r z />;`
- `<specs X=x angulo=a eixoZ=e z escala x=9.45 escala y=e y escala z=e z />;`
- `<grupo> </grupo>;`

Posto isto, vimo-nos obrigados a adicionar novas funcionalidades a função que carregava os grupos (*GRload*) de modo que seja possível a realização do *Parsing*.

```
void GRload(TiXmlNode *currentNode,list<Grupo> *grupos){
    //percorre os grupos
    for(TiXmlNode* gNode = currentNode->FirstChild("grupo") ;
        gNode; gNode = gNode->NextSiblingElement())
    {
        (...)
        Cor c;
        TiXmlNode* cNode = gNode->FirstChild("cor");
        if(cNode)
        {
            TiXmlElement* cElem = cNode->ToElement();
            const char *R = cElem->Attribute("R"),
                       *G = cElem->Attribute("G"),
                       *B =cElem->Attribute("B");
```

```
            if(R) c.r = atof(R); else c.r = 1;
            if(G) c.g = atof(G); else c.g = 1;
            if(B) c.b = atof(B); else c.b = 1;
        }
        else { c.r = 1; c.g = 1; c.b = 1; }
        g.cor = c;
        //orbita
        Orbita o;
        TiXmlNode* oNode = gNode->FirstChild("orbita");
        if(oNode)
        {
            TiXmlElement* oElem = oNode->ToElement();
            const char *raioX = oElem->Attribute("raioX"), *raioZ = oElem->Attribute("raioZ"
            if(raioX) o.raioX = atof(raioX); else o.raioX = 0;
            if(raioZ) o.raioZ = atof(raioZ); else o.raioZ = 0;
            g.tipo = "orbita";
        }
        else { o.raioX = 0; o.raioZ = 0; g.tipo = "planeta"; }
        g.orbita = o;
        //modelos
        TiXmlNode* mNode = gNode->FirstChild("modelos");
        if(mNode)
        {
            TiXmlElement* mElem = mNode->ToElement();
            TiXmlNode* testNode = mElem->FirstChild("modelo");
            if(testNode)
            {
            (...)
            }
        }
    }
    if(gNode) GRload(gNode,&g.grupos);
    grupos->push_back(g);
    }
}
```

4.3 Órbita

Nesta fase do trabalho foi necessário foi necessário elaborar uma das componentes que a nível gráfico são das mais importantes: órbitas.

```
    void drawOrbita(float rX, float rZ){
    float slice = M_PI/180;
    float doisPi = 2*M_PI;

    glBegin(GL_LINE_STRIP);

    for(float i=0; i<=doisPi ; i+=slice)
        glVertex3f(cos(i)*rX,0,sin(i)*rZ);

    glEnd();
}
```

Por isso, implementamos a função apresentada acima, drawOrbita. Através desta função conseguimos desenhar uma elipse fornecendo os dois diferentes raios que a compõe desenhando 300 pontos no modo *GL_LINE_STRIP*.

5 Câmara

Um dos objetivos deste projeto e instalar uma camara em primeira pessoa, de modo a que o utilizador se possa mover ao longo do espaco, podendo assim visualizar melhor o que pretende ver ou da perspetiva que desejar.
Decidimos manter a mesma câmara que elaboramos para a primeira fase, uma vez que achamos que cumpre todas as necessidades que o utilizador necessita.
Funcionalidades:

- Posicionamentos da câmara: aumenta ou diminui o raio radius em 10;
- Translações: aumenta ou diminui tanto o valor do eixo do x como o valor do eixo do z em 1 unidade;
- Alteração do ângulo de visão: aumenta ou diminui tanto o valor do ângulo alpha como o valor do ângulo beta em 0.1 unidades;

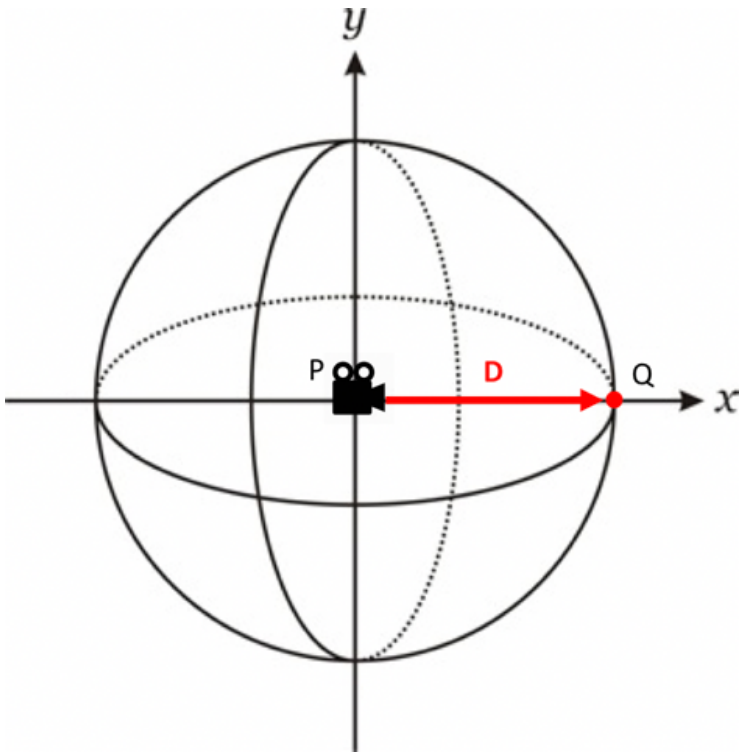


Figura 2: Esquema da posição da câmara (P) e do vetor direção D

6 Interação com o Utilizador

6.1 Teclado

- Tecla W: Aumenta o raio radius em 10, fazendo com que a câmara se movimente para a frente;
- Tecla S: Diminui o raio radius em 10, fazendo com que a câmara se movimente para trás;
- Tecla G: translação que incrementa o valor do eixo dos x em 1;
- Tecla J: decrementa valor do eixo dos x em 1;
- Tecla Y: aumenta 1 ao valor do eixo dos z;
- Tecla H: translação que dimnui 1 unidade ao valor do eixo dos z;
- Tecla ESC: Sai do sistema;
- Tecla 'UP': Permite a rotação de todos os componentes em torno do sol, de baixo para cima, aumentando o angulo beta em 0.1;
- Tecla 'DOWN': Permite a rotação de todos os componentes em torno do sol, de baixo para cima, dimnuindo o angulo beta em 0.1;
- Tecla 'LEFT': Permite a rotação das respetivas órbitas e planetas em torno do sol para a esquerda, ou seja, diminui ângulo alpha em 0.1;
- Tecla 'RIGHT': Permite a rotação das respetivas órbitas e planetas em torno do sol para a esquerda, isto é, aumenta ângulo alpha em 0.1;

6.2 Rato

Além de terem sido associadas ações ao teclado, foram também associadas ações aos botões direito e esquerdo do rato. O botão direito do rato permite ao utilizador aceder a um menu de contexto com diferentes opções de visualização do aspeto das figuras. O botão esquerdo do rato permite movimentar a câmara.

6.2.1 Botão do lado esquerdo do rato

Como visto anteriormente, a orientação da câmara pode ser controlada com as setas direcionais do teclado. Além destas opções, também é possível fazer esta alteração com o botão esquerdo do rato.
Para rodar a câmara, o utilizador deve carregar no botão esquerdo e arrastar o rato na direção que pretende, mantendo o botão pressionado.

6.3 Botão do lado direito do rato

Para proporcionar alguma flexibilidade na visualização das figuras e também de forma a aumentar a interação do utilizador com o programa, foi incluído um menu tendo como opções o FILL, o LINE e o POINT sendo cada uma das opções formas de alterar o aspeto das figuras, permitindo assim estudá-las de diversos pontos de vista.

7 Sistema Solar

Nesta fase tentamos representar o Sistema Solar da melhor maneira possível, isto é, dentro do possível tentamos representá-lo à escala. Contudo, como o tamanho do Sol comparativamente com o tamanho dos restantes planetas é muito maior, o tamanho do mesmo teve de ser ajustado de forma a apresentarmos um Sistema Solar equilibrado.
Quanto as órbitas podemos afirmar que não estão à escala, embora a sua excentricidade está de acordo com a realidade e, desta feita, colocamos uma adição de modo a que seja possível uma melhor visualização das mesmas. Tal como nas órbitas, a posição do Sol também corresponde à realidade.
Ainda quantos aos planetas, todos eles contém inclinação axial tal como as órbitas têm obliquidade em relação ao plano da orbita terrestre.

O Sistema Solar é composto por:

- Estrela : Sol, representado por uma esfera;
- Planetas : Mercúrio, Vénus, Terra, Marte, Saturno, Urânio, Neptuno;
- Satélites : Lua, Europa, Io, Ganimedes e Calisto;
- Órbitas : correspondentes a cada um dos objetos anteriores;

7.1 Resultados

Serão agora apresentados os resultados obtidos pela nossa implementação visto através de diferentes perspetivas e sobre diversos elementos presentes no mesmo. Primeiramente temos uma visão geral do sistema criado através de diferentes angulos(Figura 3 e 4) onde se observa a presença tanto do Sol como dos diferentes planetas presentes no Sistema Solar.

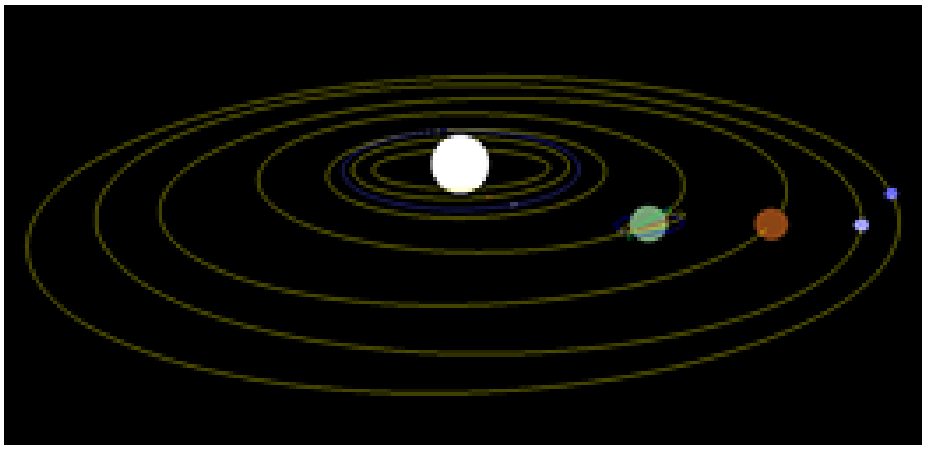


Figura 3: Perspectiva do Sistema obtido 1

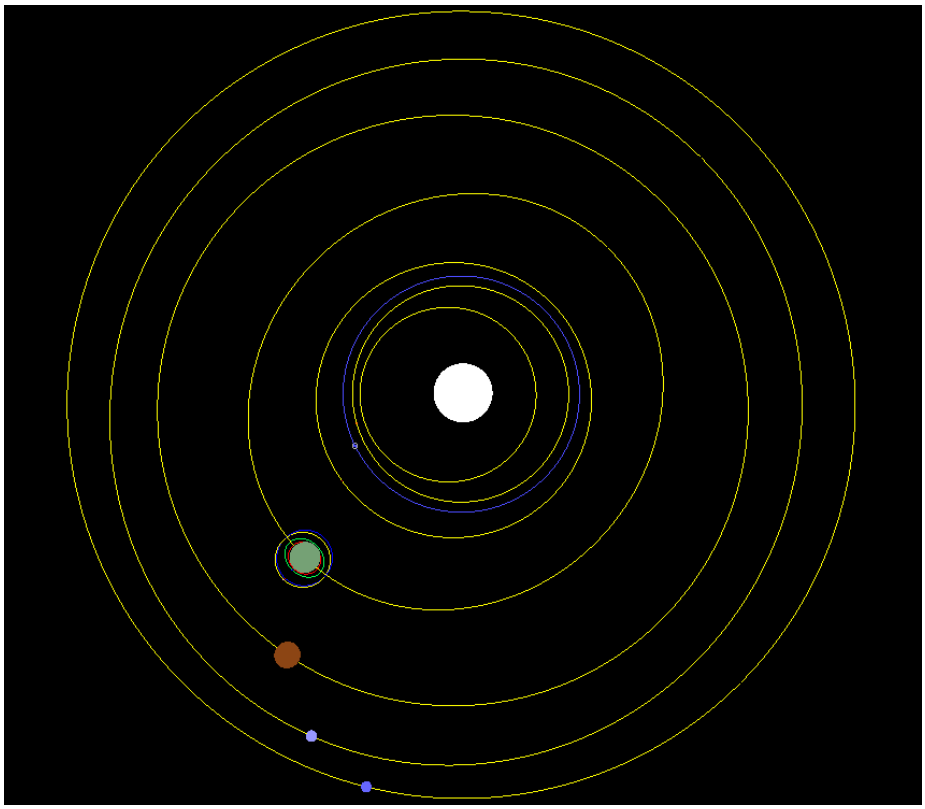


Figura 4: Perspectiva do Sistema obtido 2

Como podemos verificar, encontram-se também presentes no resultado obtido as orbitas associadas aos diferentes planetas em torno do Sol. De seguida temos um melhor visão sobre o ponto central do Sistema Solar, o Sol, onde se consegue comprovar o seu superior volume e também uma cor mais clara para representar a natureza do astro (Figura 5).

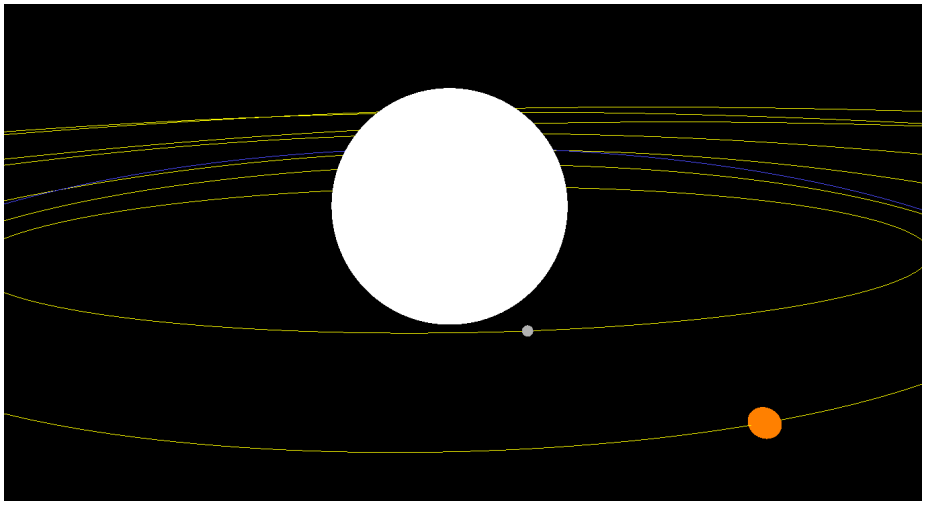


Figura 5: Visão amplificada do Sol do Sistema obtido

Podemos também observar a presença de luas em planetas como a Terra e Jupiter(Figuras 6 e 7), um dos objectivos do trabalho e onde também se comprova a existência de orbitas das luas em torno do Planeta correspondente.

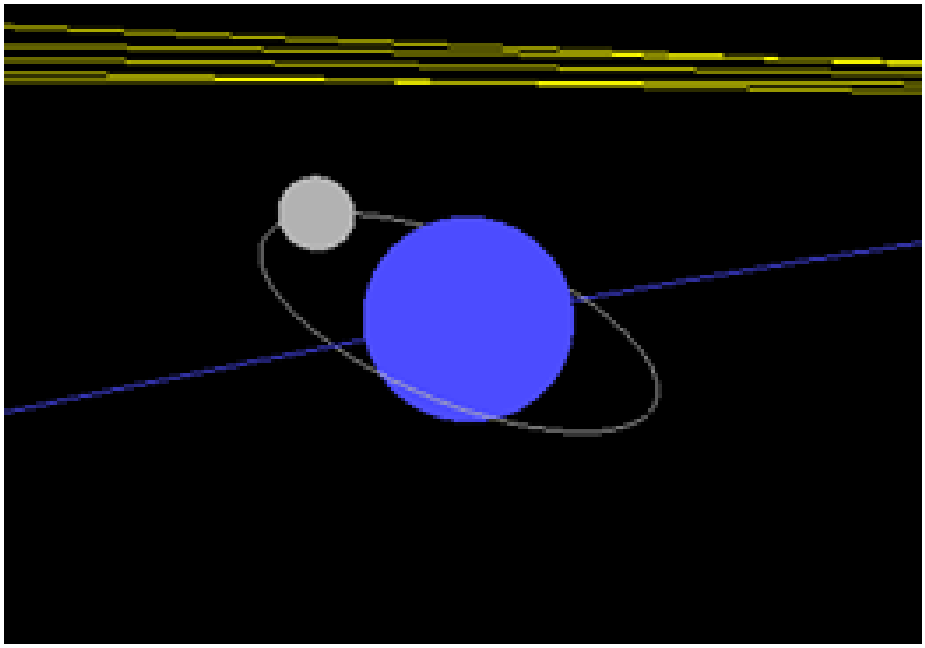


Figura 6: Visão amplificada da Terra e da Lua

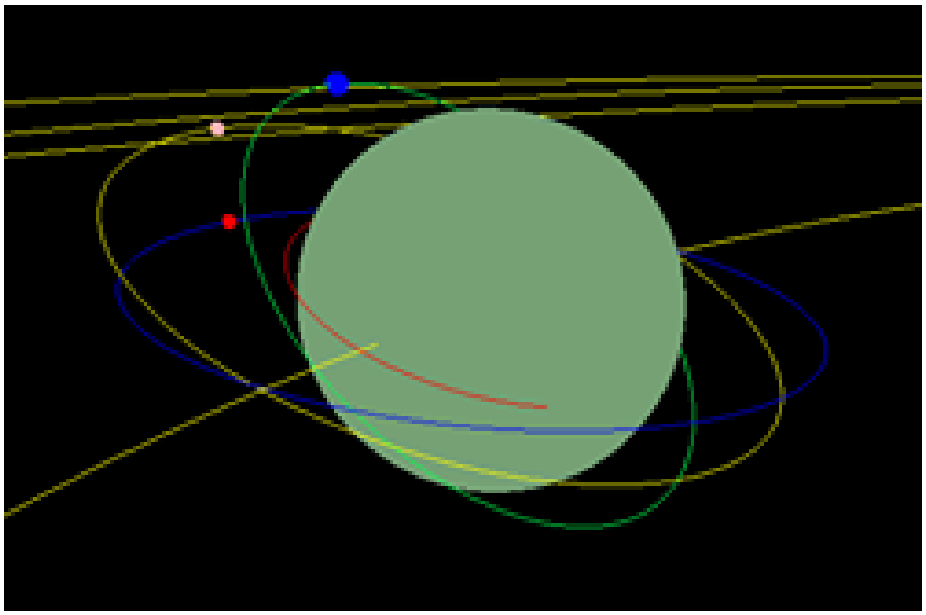


Figura 7: Visão amplificada de Jupiter e suas luas

Através das figuras apresentadas do projecto conseguimos também verificar a possibilidade de manipulação da câmara uma vez que conseguimos cobrir diferentes angulos sob o sistema.

8 Conclusão

No desfecho deste desafio, o grupo consegue claramente reconhecer que este trabalho prático consolidou, de um modo eficaz, os conteúdos leccionados até ao momento na Unidade Curricular de Computação Gráfica – fomos bem sucedidos na aplicação de transformações e modelação hierárquica e alcançou-se, como produto final, uma representação aprazível do Sistema Solar.

Considerámos a nossa implementação robusta, capaz de corresponder ao que é pedido e suficientemente escrupulosa quanto à escala do modelo.

Uma das principais dificuldades enfrentadas foi convencionar corretamente a escala do sistema solar, visto que acaba por se tornar complexo gerir distâncias tão colossais na vida real e escalá-las satisfatoriamente. Empenhámonos, no entanto, em ultrapassar este obstáculo com a maior eficiência e o mais acertadamente que foi possível.

Enumerando tudo isto, concluímos que realizámos esta atividade com sucesso e com aquisição dos conhecimentos significantes relativamente a esta matéria.