Universidade do Minho
Escola de Engenharia

COMPUTAÇÃO GRÁFICA

FASE 3

Alexandre Martins A77523

André Vieira A78322

Eduardo Rocha A77048

Ricardo Neves A78764

ÍNDICE

INTRODUÇÃO

Neste relatório iremos apresentar e discutir o trabalho realizado pelo grupo, no âmbito da Unidade Curricular de Computação Gráfica, do 3º ano do Mestrado Integrado em Engenharia Informática.

Nesta terceira fase do trabalho prático, tivemos como objetivo a criação de uma cena dinâmica, onde cada planeta gira à volta do Sol, enquanto que um cometa percorre um determinado percurso. Para a construção deste modelo dinâmico, foi necessário estudar e implementar as Curvas de Catmull-Rom e os Bezier Patches (para o desenho do cometa).

De salientar que todos os modelos foram desenhados com recurso aos VBOs, ao contrário das duas fases anteriores. Ao longo deste documento, iremos detalhar cada uma destas noções especificadas.

Portanto, neste relatório, iremos apresentar detalhadamente todo o processo realizado, de modo a cumprir os objetivos previamente estabelecidos pelo docente da Unidade Curricular.

Para isto, apresentamos também neste documento, algumas linhas de pensamento que o grupo seguiu, ferramentas usadas, e excertos de código fonte utilizado (e respetiva explicação), de modo a suportar a perceção do trabalho realizado.

De relembrar que, na segunda fase, o grupo desenvolveu um sistema solar estático, com todos os planetas, o Sol e algumas luas. A cada modelo desenhado está associada uma cor, de modo a traduzir, o mais precisamente, a realidade, sendo que, também, cada planeta do Sistema contém a sua órbita desenhada.

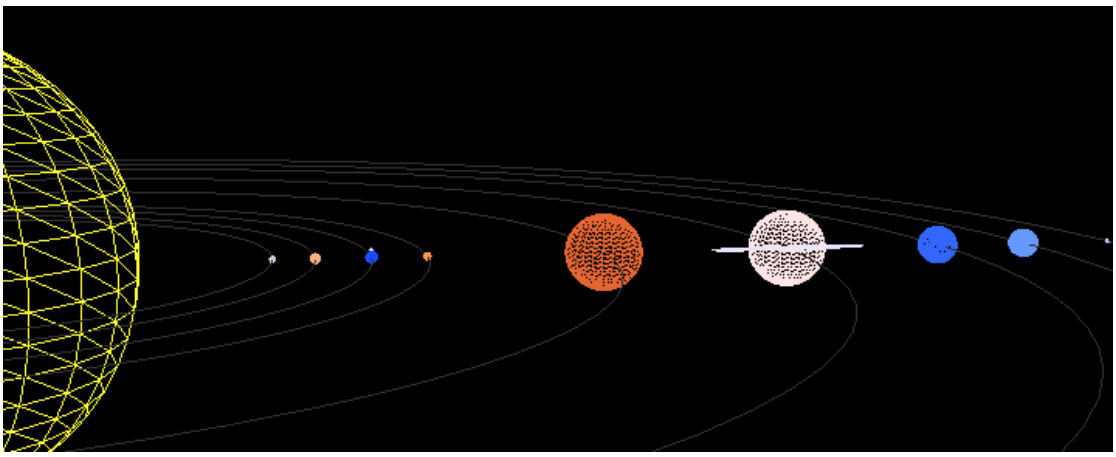


Figura 1 - Modelo da Fase 2

PARSER DO XML

Tal como na fase anterior, o Parser do XML sofreu algumas adições ao seu algoritmo, de modo a acolher uma maior quantidade de informação proveniente do ficheiro de configuração do Sistema Solar.

Nesta fase, o Parser terá de ser capaz de reconhecer e guardar os pontos do Sistema por onde um certo planeta terá de passar, de modo a formar a sua órbita à volta do Sol.

Deste modo, o grupo achou por bem criar uma nova estrutura, denominada Translation, que guarda o tempo da transformação e os pontos constituintes da órbita a ser percorrida.

```
typedef struct translation {
    double time;
    vector<coords> points;
} Translation;
```

Figura 2 - Estrutura de dados "Translation"

Agora que temos uma forma de armazenar os dados facilmente, temos de preencher esta estrutura. Sempre que o Parser encontrar o atributo “time”, significa que irá se tratar de uma transformação dinâmica. Depois disto, percorremos todos os pontos disponibilizados no XML, guardando um a um no seu espaço correspondente. No final, temos uma estrutura onde estão armazenados todos os modelos, as suas transformações e, se for o caso, os pontos da translação dinâmica. Não esquecendo que a rotação de um modelo também pode ser efetuada ao longo do tempo, sendo que esse caso também foi coberto.

Temos a reforçar que, tal como explicado na segunda fase, todas estas estruturas se encontram aninhadas. Neste caso, a estrutura de dados “Translation” faz parte da estrutura “Second”, detalhada, como já dissemos, no relatório da Fase 2. Isto traduz-se numa maior organização e hierarquia dos dados, muito importante para quando for necessário aplicar todas estas transformações mais ou menos complexas.

No caso de se tratar de uma translação dinâmica, o "time" disponibilizado corresponde ao tempo, em segundos, que o modelo demorará a percorrer uma curva de Catmull-Rom.

Aqui, temos um pedaço do algoritmo que é responsável por adicionar os pontos da curva à estrutura de dados correta, guardando consigo, também, o tempo indicado. A função addPoint() ali indicada apenas adiciona o ponto de coordenadas (x, y, z) ao vetor Points.

```
if(group->Attribute("time")) {

    char* timeString = (char *) group->Attribute("time");
    time = atof(timeString);

    for(const XMLElement* translate = group->FirstChildElement();
        translate; translate = translate->NextSiblingElement()){

        char* pointX = (char*) translate->Attribute("X");
        if(pointX != NULL) {
            x = atof(pointX);
        }

        char* pointY = (char*) translate->Attribute("Y");
        if(pointY != NULL) {
            y = atof(pointY);
        }

        char* pointZ = (char*) translate->Attribute("Z");
        if(pointZ != NULL) {
            z = atof(pointZ);
        }

        addPoint(x, y, z, points);

    }

    translation = new Translation();
    translation->time = time;
    translation->points = points;

}
```

Figura 3 - Algoritmo para translação dinâmica

Assim, podemos afirmar que o Parser não sofreu alterações muito profundas, mas sim que foi necessário a adição de alguns casos de transformações dinâmicas, pelo que esta tarefa não apresentou grandes dificuldades ao grupo de trabalho.

EXTRAÇÃO DAS COORDENADAS

No que toca a toda a extração das coordenadas dos ficheiros dos modelos criados, este algoritmo não foi modificado. Isto deve-se a que os modelos dos planetas e outros constituintes do Sistema Solar já estavam criados da fase anterior.

OPENGL

VBOs

VBO (Vertex Buffer Object) é uma característica do OpenGL que oferece um ganho de performance substancial sobre o método utilizado até esta fase do projeto. Isto deve-se ao facto de a informação ser agora armazenada na memória da placa gráfica ao invés da memória do próprio sistema.

Sendo assim, é necessário preencher os dados VBOs com os triângulos constituintes dos vários modelos do sistema. Este processo foi bastante rápido uma vez que, nas fases anteriores, o grupo teve que efetuar a mesma tarefa.

Aqui, também é preciso verificar a passagem de um modelo para outro, fazendo uso, de novo, da função checkEnding().

```
while(i != triangles.end()){

    First* first = *iFirst;
    vbo[x] = (double*) malloc(sizeof(double) * first->npoints*3);

    flag = false;
    pts = 0;

    while(!flag){

        coords one = *i;
        i++;
        coords two = *i;
        i++;
        coords three = *i;
        i++;

        if(!checkEnding(one, two, three)){

            vbo[x][pts++] = get<0>(one);
            vbo[x][pts++] = get<1>(one);
            vbo[x][pts++] = get<2>(one);

            vbo[x][pts++] = get<0>(two);
            vbo[x][pts++] = get<1>(two);
            vbo[x][pts++] = get<2>(two);

            vbo[x][pts++] = get<0>(three);
            vbo[x][pts++] = get<1>(three);
            vbo[x][pts++] = get<2>(three);

        }
```

Figura 4 - Preenchimento dos VBOs

No fim deste preenchimento, temos uma matriz cheia com os pontos a desenhar. Cada linha da matriz corresponde a um modelo diferente, de modo a ser mais fácil aplicar as transformações numa fase posterior.

Transformações

Neste momento, temos de verificar se a transformação é estática ou dinâmica. Se se verificar a existência da estrutura de dados "Translation", podemos chegar à conclusão imediatamente que se trata de uma transformação dinâmica.

Em baixo, temos o exemplo do algoritmo explicado para a transformação de translação.

```
if (strcmp(second->transf, "translate") == 0) {

    if(second->translation == nullptr)
        glTranslated(second->params[0], second->params[1], second->params[2]);

    else
        renderCatmullRomCurve(x, second);

}
```

Figura 5 - Translação estática ou dinâmica

Curvas de Catmull-Rom

Como já dissemos, as órbitas dos planetas e o percurso do cometa foram construídas a partir do conceito das curvas de Catmull-Rom.

Para isto, utilizamos o formulário disponibilizado pelo docente da UC, disponível na plataforma ELearning. Daqui, extraímos a fórmula a aplicar:

$$[t^3 \quad t^2 \quad t \quad 1]\begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -0.5t^3 + t^2 - 0.5t \\ 1.5t^3 - 2.5t^2 + 1 \\ -1.5t^3 + 2t^2 + 0.5t \\ 0.5t^3 - 0.5t^2 \end{bmatrix}^T$$

Figura 6 - Fórmula de Catmull-Rom

Com isto, e com o trabalho realizado nas aulas práticas, foi mais fácil a implementação das curvas.

```
double matrixT[1][4] = { { powf(t,3), powf(t,2), t, 1 } };
```

Figura 7 - Matriz T

```
// Matrix Catmull-Rom
double m[4][4] = { {-0.5f,  1.5f, -1.5f,  0.5f},
                   { 1.0f, -2.5f,  2.0f, -0.5f},
                   {-0.5f,  0.0f,  0.5f,  0.0f},
                   { 0.0f,  1.0f,  0.0f,  0.0f} };
```

Figura 8 - Matriz de Catmull-Rom

Com isto, criamos as matrizes da fórmula, efetuando os cálculos posteriormente, fazendo uso de 3 funções multiplicativas, cada uma com argumentos diferentes, dependendo, claro, do caso a ser tratado no momento. Dito isto, o grupo seguiu a seguinte metodologia de cálculo de Catmull-Rom (também disponibilizada nos slides da disciplina), utilizando as matrizes e as funções criadas anteriormente.

```
void getCatmullRomPoint(float t,
                        float *p0, float *p1, float *p2, float *p3,
                        float *pos, float *deriv) {

      // catmull-rom matrix
      float m[4][4] = {   {-0.5f,  1.5f, -1.5f,  0.5f},
                          { 1.0f, -2.5f,  2.0f, -0.5f},
                          {-0.5f,  0.0f,  0.5f,  0.0f},
                          { 0.0f,  1.0f,  0.0f,  0.0f}};

      // Compute A = M * P
      // for component x P is the vector (p0[0], p1[0], p2[0],p3[0]
      // Compute pos = T * A
      // compute deriv = T' * A
      // ...
}
```

Figura 9 - Algoritmo de Catmull-Rom

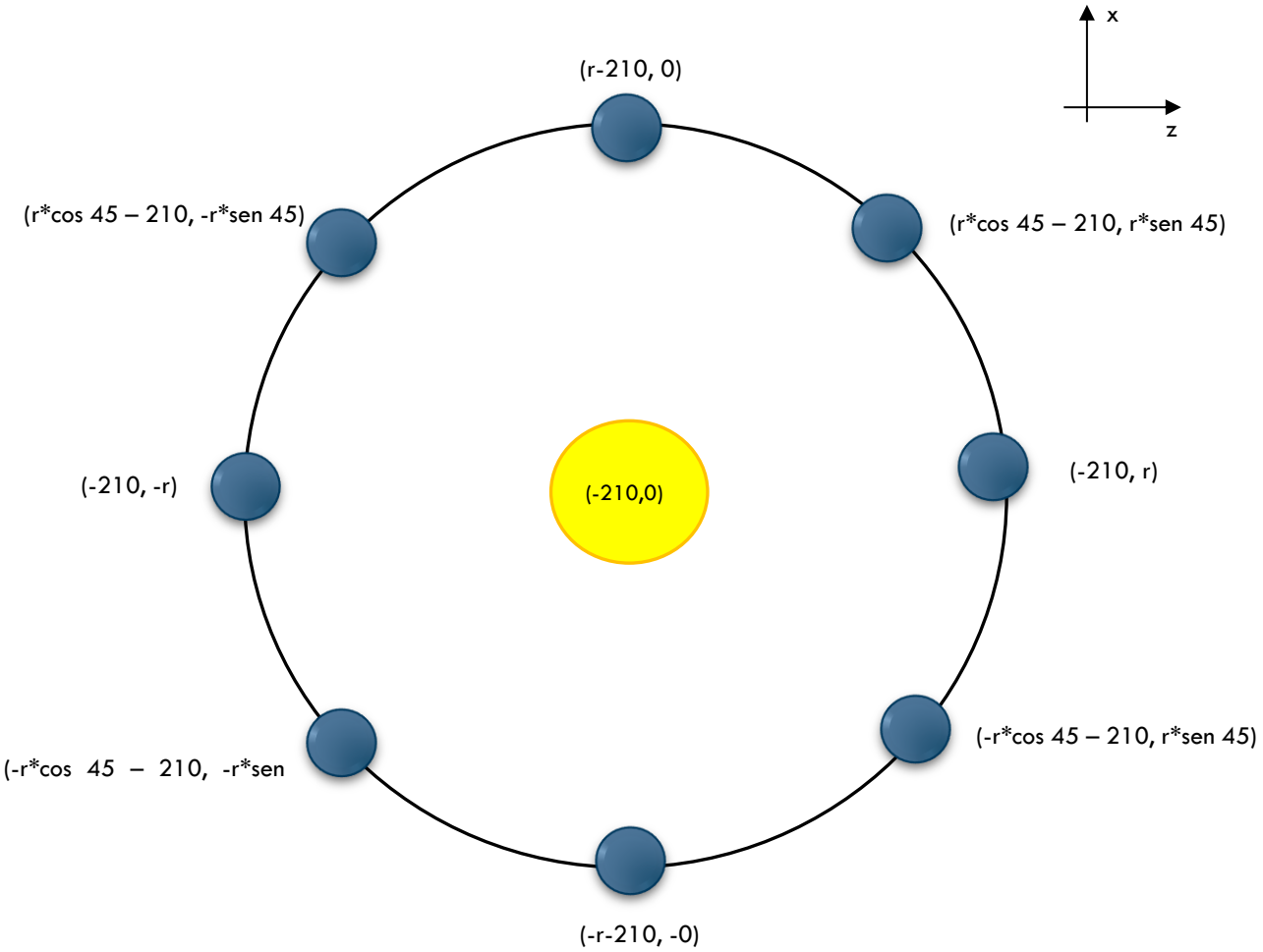
CÁLCULO DAS ÓRBITAS

Planetas

Para o cálculo das órbitas do planeta, tivemos em consideração as distâncias já calculadas na segunda fase.

Para a definição da órbita de um planeta, foram calculados 8 pontos, de modo a formar essa mesma órbita circular à volta do Sol. Assim, conhecendo, à partida, o raio da órbita de cada planeta em relação ao Sol e que a coordenada Y é sempre igual a 0, construímos o seguinte esquema para facilitar o cálculo dos 8 pontos da órbita.

r é a distância do planeta ao sol.



Aproveitamos para informar que, na segunda fase, os desenhos das várias órbitas dos planetas foram desenhados de uma forma bruta, "hard coded", sendo que cada uma tem o seu pedaço de código.

Agora, nesta terceira fase, estas órbitas foram desenhadas com a ajuda, também, das curvas de Catmull-Rom, com uma cor ténue que não interferisse em demasia com o modelo final.

```
glBegin(GL_LINE_LOOP);

for (int i = 0; i < npoints; i++) {

    getGlobalCatmullRomPoint((double) i/npoints, pos, deriv, curve, size);
    glColor3f(0.2, 0.2, 0.2);
    glVertex3d(pos[0], pos[1], pos[2]);

}
```

Figura 10 - Ciclo das orbitas

Para certificar que toda a informação estava a ser lida e aplicada corretamente, decidimos verificar, através de um output, esses mesmos dados sobre os modelos e as suas transformações. Como vemos, tudo está a ser lido com precisão.

```
File: ../sphere.3d
   Transformation: translate
   Time: 3
   Number of points: 8
       :: X = -210 Y = 0 Z = -165
       :: X = -93.3 Y = 0 Z = -116.7
       :: X = -45 Y = 0 Z = 0
       :: X = -93.3 Y = 0 Z = 116.7
       :: X = -210 Y = 0 Z = 165
       :: X = -326.7 Y = 0 Z = 116.7
       :: X = -375 Y = 0 Z = 0
       :: X = -326.7 Y = 0 Z = -116.7

   Transformation: scale
   X = 0.3 Y = 0.3 Z = 0.3 Angle = 0

   Transformation: color
   X = 0.1 Y = 0.3 Z = 1 Angle = 0

----------------------------------------
```

Figura 11 - Output de um modelo

Assim, já temos os planetas a seguir o caminho das suas órbitas individuais, bem como as suas luas, se for o caso.

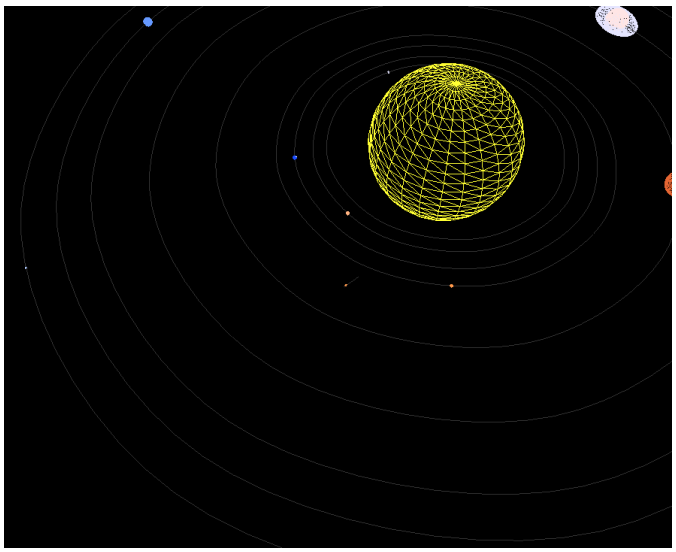


Figura 12 - Órbitas dos planetas

Cometa

Um dos pontos negativos a salientar foi a não implementação dos Patches de Bezier, aspeto que pretendemos melhorar na quarta e última fase do projeto.

Assim, decidimos usar o modelo da "box" criada na primeira fase. Este cometa contém uma trajetória associada a 4 pontos, o mínimo para qualquer curva de Catmull-Rom.

Para além desta trajetória, podemos ver através da análise do XML disponibilizado, o cometa contém um movimento de rotação dinâmico sobre os eixos X, Y e Z, o que imprime uma maior sensação de movimento de queda livre no espaço, por parte do modelo.

```
<group>
    <translate time="2" >
        <point X="0" Y="20" Z="-20" />
        <point X="0" Y="10" Z="-15" />
        <point X="0" Y="0" Z="-10" />
        <point X="0" Y="-10" Z="-5" />
    </translate>
    <rotate time="0.5" X="10" Y="10" Z="10"/>
    <scale X="0.2" Y="0.2" Z="0.2" />
    <color R="1" G="0.6" B="0.3" />
    <models>
        <model file="box.3d" /> -- Comet
    </models>
</group>
```

Figura 13 - XML do cometa

Aqui apresentamos um frame da trajetória do cometa, com forma de uma caixa, e a sua trajetória. Infelizmente, não percebemos, a partir desta imagem, o movimento de rotação que o modelo está a efetuar.

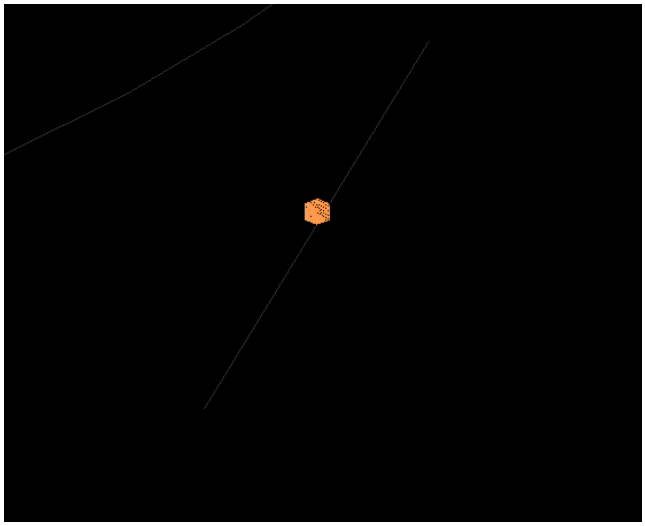


Figura 14 - Cometa do modelo

SISTEMA SOLAR FINAL

Por fim, o conjunto de todos os modelos criados, associados aos seus movimentos, traduz-se no seguinte Sistema Solar:

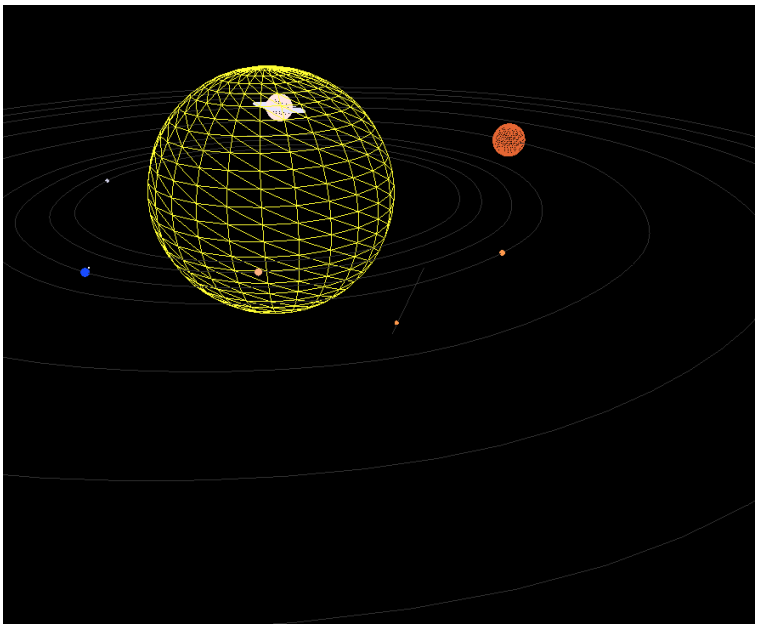


Figura 15 - Modelo do Sistema Solar

CONCLUSÃO E TRABALHO FUTURO

Chegando ao final deste documento, podemos afirmar que esta terceira fase foi bastante demorosa, tendo sido aquela em que o grupo sentiu mais dificuldades devido à quantidade e complexidade de pontos a abordar.

Nesta fase, conseguimos cumprir o objetivo da implementação dos VBOs, das transformações dinâmicas, incluindo as curvas de Catmull-Rom. No entanto, o cometa presente no modelo final não foi concebido com os Bezier Patches, o que conseideramos um ponto negativo a ter em conta, procurando melhor isto mesmo na última fase do trabalho prático.

No entanto, podemos afirmar que o resultado final foi satisfatório entre os membros,

Por isto, podemos dizer que a nossa prática em C++ foi um pouco melhorada com este projeto até então, bem como os conhecimentos em OpenGL.

Para a próxima fase, iremos debruçar-nos sobre a implementação de texturas nos modelos do Sistema Solar, e os vários tipos de luzes.

Com o trabalho realizado até aqui, sentimo-nos capazes de continuar na implementação do projeto, sendo que os objetivos propostos foram cumpridos e consolidados.

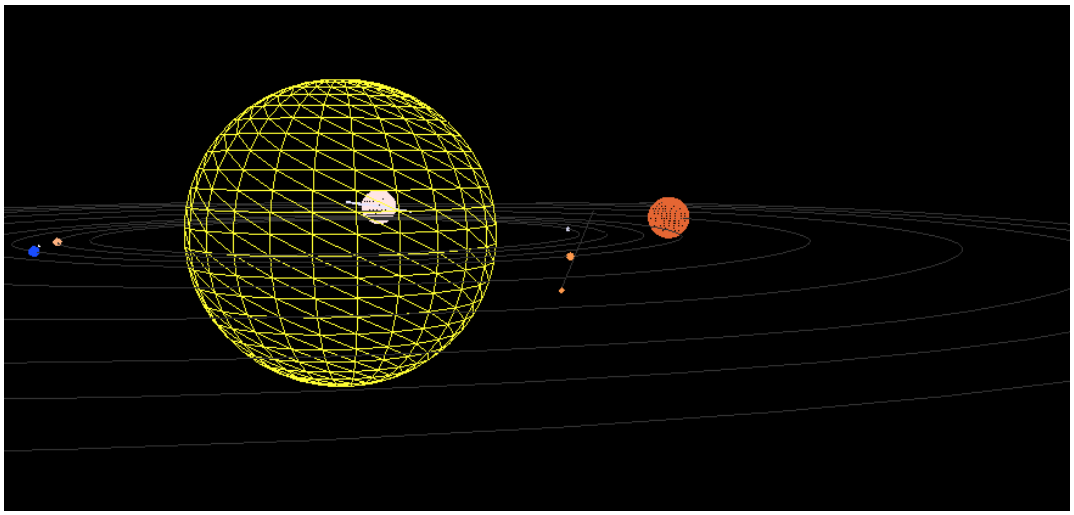


Figura 16 - Modelo do Sistema Solar