

Trabalho Final de Computação Gráfica

Antonio Jorge Medeiros, RA: 620521

Carlos Alberico Bezerra de Andrade, RA: 613827

Evelin Priscila Ferreira Soares, RA: 489832

Professor: Prof.: Dr. Mario Augusto de Souza Liziér

Centro de Ciências e Gestão em Tecnologia

Universidade Federal de São Carlos

Sumário

1	Apresentação do Trabalho	1
1.1	Objetos	1
2	Desenvolvimento	2
2.1	Visualização	2
2.2	Dificuldades encontradas	2
2.3	Bibliotecas utilizadas	2
2.4	Requisitos	3
2.5	Requisitos Extras	7

Lista de Figuras

2.1	Exemplo de visualização de cinco objetos, pelo menos dois objetos carregados de arquivos e com textura em algum modelo obj	6
2.2	Arquivos utilizados para possibilitar a visualização da lua.	7

1. *Apresentação do Trabalho*

Nesta avaliação prática da Disciplina de Computação Gráfica, implementamos uma visualização do espaço com controle de uma nave pelo espaço, em um planeta, com uma lua o orbitando, um cinturão de cometas e uma sistema solar provedor de luz.

1.1 Objetos

- Nave: Um objeto carregado de um arquivo, de fonte aberta.
- Planeta: Também um objeto carregado de um arquivo, de fonte aberta.
- Sol: Objeto provedor de luz.
- Cometas: Outro objeto carregado de um arquivo, de fonte aberta, estão espalhados ao redor do cenário.

2. *Desenvolvimento*

Quanta ao desenvolvimento, aproveitamos boa parte do código das atividades anteriores, como ObjLoader, Curva de Bézier, manipulação da Câmera e interação com o usuário.

2.1 Visualização

Inicialmente tem-se a nave posicionada no centro do cenário, para visualizar o cenário a partir de outro ponto de visualização é possível movimentar a nave com os comandos WASD, e mudar o foco da câmera de visualização com as *arrow keys*. Também é possível mudar a câmera escolhendo valores na faixa de 1 a 5, onde é possível visualizar o cenário de uma outra perspectiva.

2.2 Dificuldades encontradas

Como não tínhamos muita experiência com JavaScript e absolutamente nenhum contato prévio com WebGL, tivemos dificuldade em implementar as funcionalidades, pois se tratava de uma linguagem que não possuíamos domínio, tivemos que procurar por tutoriais e tirar dúvidas em relação a linguagem em fóruns de internet.

2.3 Bibliotecas utilizadas

Optamos por desenvolver o trabalho utilizando WebGL, (Web Graphics Library) que é uma API em JavaScript, disponível a partir do novo elemento canvas da HTML5,

que oferece suporte para renderização de gráficos 2D e gráficos 3D, que pode ser implementado em uma aplicação web sem a necessidade de plug-ins no navegador, pois achamos mais prático fazer uso das bibliotecas disponíveis para auxiliar no desenvolvimento do trabalho. Fizemos bastante uso da biblioteca Three.js que possibilita uso de diversos métodos já implementados, o que nos ajudou em manipular recursos que não possuíamos muito conhecimento em como implementar.

2.4 Requisitos

- Cinco objetos no total, dois objetos carregados de arquivos.

No trecho de código em 2.4 cinco objetos são enviados como parâmetro para o método *objLoader* que irá carregar esses objetos e aplicar a textura nos respectivos objetos.

E na figura 2.4 é possível visualizar os cinco objetos.

//CARREGA CINCO OBJETOS DIFERENTE USANDO O MÉTODO OBJLOADER

```
obj.push (objLoader ('quad/', 'quad.mtl', 'quad.obj'));
obj[0].rotateY (Math.PI);
obj[0].scale.multiplyScalar (1/50);
scene.add (obj[0]);

obj.push (objLoader ('earth/', 'earth.mtl', 'earth.obj'));
obj[1].position.set (10000, 2500, -10000);
obj[1].scale.multiplyScalar (10);
scene.add (obj[1]);

obj.push (objLoader ('moon/', 'moon.mtl', 'moon.obj'));
obj[2].scale.multiplyScalar (300);
obj[2].rotateX (-Math.PI/2);
scene.add (obj[2]);
```

```

var tamanhoAstronauta = 10;
obj.push(objLoader ('astronaut/', 'astronaut.mtl', 'astronaut.obj'));
obj[3].scale.multiplyScalar(tamanhoAstronauta);
scene.add (obj[3]);

obj.push(objLoader ('astronaut/', 'astronaut.mtl', 'astronaut.obj'));
obj[4].scale.multiplyScalar(tamanhoAstronauta);
scene.add (obj[4]);

obj.push(objLoader ('astronaut/', 'astronaut.mtl', 'astronaut.obj'));
obj[5].scale.multiplyScalar(tamanhoAstronauta);
scene.add (obj[5]);

```

- Uma forma simples, Textura em algum objeto simples

É possível visualizar um objeto de forma simples, sendo este objeto o planeta Quom, o trecho de código contido em 2.4 exibe como é feita a inserção do planeta no cenário, bem como o carregamento da textura desse objeto. Na linha 7 do trecho de código é feito o carregamento da textura a partir do arquivo *Quom.png* e na linha 13 o objeto é inserido na cena.

```

1
2  //-----FORMA SIMPLES COM TEXTURA (Planeta
   Quom)-----//
3      var textura = new THREE.MeshPhongMaterial( {
4          specular: 0x020201,
5          shininess: 5,
6          reflectivity: .01,
7          map: textureLoader.load( "Quom/Quom.png" ),
8      } );
9
10     var esfera = new THREE.SphereGeometry( 4000, 200,
        200 );
11     obj.push(new THREE.Mesh( esfera, textura ));

```



```
12         obj[7].position.set (-10000, 2500, -10000);  
13         scene.add( obj[7] );
```

- Textura em algum modelo obj

Para carregar os objetos criamos um OBJLoader e um MTLLoader diferente do das etapas anteriores, esse funciona de forma mais abstrata, passando como parâmetro o endereço, nome do MTL e nome do OBJ. Conforme é possível ver no trecho de código em 2.4

```
1  
2  //CARREGA UM OBJETO E A TEXTURA CORRESPONDENTE USANDO  
   OBJLOADER E MTLLOADER  
3  function objLoader (endereço, nomeMtl, nomeObj) {  
4      var mtlLoader = new THREE.MTLLoader();  
5      var objLoader = new THREE.OBJLoader();  
6      var container = new THREE.Object3D();  
7  
8      mtlLoader.setPath (endereço);  
9      mtlLoader.load(nomeMtl, function ( materials ){  
10         objLoader.setPath (endereço);  
11         materials.preload();  
12         objLoader.setMaterials( materials );  
13         objLoader.load(nomeObj, function ( object ) {  
14             container.add (object);  
15         });  
16     });  
17     return container;  
18 }  
19 }
```

Na figura 2.4 é possível visualizar cinco objetos distintos e suas respectivas texturas sendo exibidas.

Na figura em 2.4 é possível ver que a Lua, assim como Nave, Terra e Cometas, possui sua própria textura, todas retiradas de fontes abertas da internet.

No trecho de código em 2.4 é possível visualizar no código do arquivo moon.obj

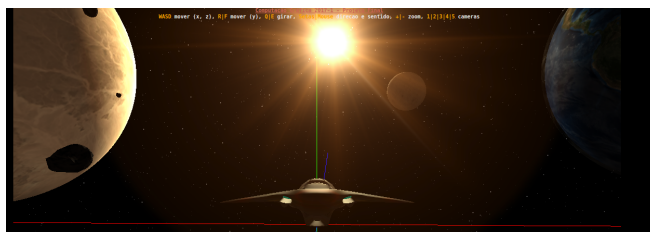


Figura 2.1: Exemplo de visualização de cinco objetos, pelo menos dois objetos carregados de arquivos e com textura em algum modelo obj

na linha 1 uma referência para o arquivo moon.mtl, e no trecho de código em 2.4 é possível visualizar no código do arquivo moon.mtl nas linha 11 e 12 uma referência para o arquivo .jpg, fazendo com que assim, seja possível visualizar o arquivo com sua respectiva textura ao carregá-lo na função *objLoader*

```
1  mllib moon.mtl
2  o Plane
3  v -0.000000 1.048728 -2.606873
4  v -0.000000 1.048728 -2.606873
```

```
1  newmtl moon
2
3  Ka .2 .2 .2
4  Kd .3 .3 .3
5  Ks .05 .05 .05
6  Ns 1.000000
7  Ni 1.000000
8  d 1.000000
9  illum 2
10
11 map_Ka moon.jpg
12 map_Kd moon.jpg
```

O trecho de código contido em 2.4 exhibe como é feita a alternância da câmera em relação a escolha do usuário.

- Duas posições distintas de câmeras, Alguma iteração do usuário (teclado ou mouse)

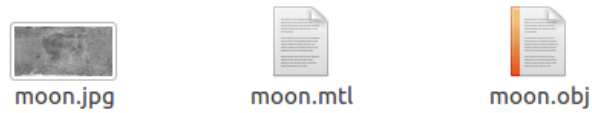


Figura 2.2: Arquivos utilizados para possibilitar a visualização da lua.

```
// ALTERA A CÂMERA DE ACORDO COM A ESCOLHA DO USUÁRIO
function alteraCamera () {
  document.onkeydown = function(e) {
    switch (e.keyCode) {
      case 107:
        cameras[0].position.multiplyScalar (2);
        break;
      case 109:
        cameras[0].position.multiplyScalar (1/2);
        break;
      ...
    }
  }
}
```

2.5 Requisitos Extras

- Mais objetos, texturas, shaders, câmeras, etc ...

São utilizados vários objetos e texturas, sendo carregados 100 cometas para visualização e é possível movimentar a câmera de forma que permita visualizar todo o cenário de diferentes perspectivas, inserindo valores na faixa de 1 a 5 ou com as *arrow keys*