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

Lista de Figuras

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 9, 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 (endereco, nomeMtl, nomeObj) {
4           var mtlLoader = new THREE.MTLLoader();
5           var objLoader = new THREE.OBJLoader();
6           var container = new THREE.Object3D();
7
8           mtlLoader.setPath (endereco);
9           mtlLoader.load(nomeMtl, function ( materials ){
10            objLoader.setPath (endereco);
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.

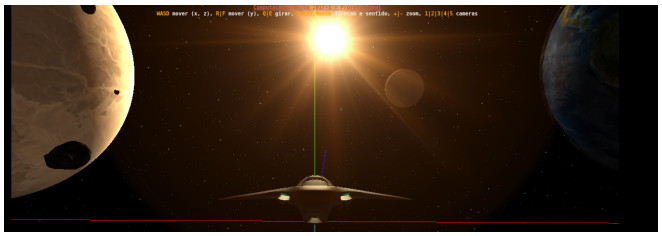

Figura 2.1: Exemplo de visualização de cinco objetos, pelo menos dois objetos carregados de arquivos e com textura em algum modelo obj

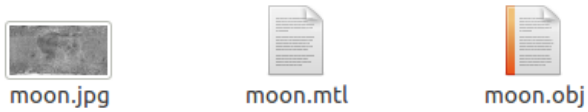

Figura 2.2: Arquivos utilizados para possibilitar a visualização da lua.

No trecho de código em 2.4 é possível visualizar no código do arquivo moon.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  mtllib 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 exibe 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)

```
// 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;
                ...
        }
}
```

- Uma curva de Bézier

  A câmera 9 viaja a cena segundo uma curva de bézier com pontos (-15000, 0, 0), (0, 15000, 0), (0, 0, 15000), (15000, 0, 0);

  O trecho de código contido em 2.4 exibe como é feita a construção da curva de Bézier no código.

```
var curva = new THREE.CubicBezierCurve3(
        new THREE.Vector3( -15000, 0, 0 ),
        new THREE.Vector3( 0, 15000, 0 ),
        new THREE.Vector3( 0, 0, 15000 ),
```

```
        new THREE.Vector3( 15000, 0, 0 )
);

bezier = new THREE.Geometry();
bezier.vertices = curva.getPoints( 500 );
```

- Shader próprio com cálculo de iluminação próprio (em shader próprio)

  Planeta Magnus (fragmentShader, vertexShader); O trecho de código contido em 2.4 exibe o uso de shader próprio no código e sua aplicação em objetos na cena. Na linha 16 é feita a criação do Shader, na linha 24 é criado o objeto que faz uso do Shader criado e na linha 26 o objeto é inserido na cena.

```
1  esfera = new THREE.SphereGeometry( 2500, 200, 200 );
2
3  uniforms = {
4
5    time:       { value: 1.0 },
6    resolution: { value: new THREE.Vector2() },
7    uvScale:    { value: new THREE.Vector2( 3.0, 1.0 ) },
8    texture1:   { value: textureLoader.load( "lava/cloud.png"
        ) },
9    texture2:   { value: textureLoader.load( "lava/
        lavatile.jpg" ) }
10
11 };
12
13 uniforms.texture1.value.wrapS =
      uniforms.texture1.value.wrapT = THREE.RepeatWrapping;
14 uniforms.texture2.value.wrapS =
      uniforms.texture2.value.wrapT = THREE.RepeatWrapping;
15
16 var material = new THREE.ShaderMaterial( {
17
18   uniforms: uniforms,
19   vertexShader: document.getElementById( 'vertexShader' ).
        textContent,
```

```
20    fragmentShader: document.getElementById( 'fragmentShader'
          ).textContent
21
22  } );
23
24  obj.push ( new THREE.Mesh( esfera, material ));
25  obj[8].position.set (-2500, 5000, -25000);
26  scene.add( obj[8] );
27
28  textura = new THREE.MeshPhongMaterial( {
29    color: 0xffaa00,
30    specular: 0x000000,
31    shininess: 50,
32    reflectivity: 1,
33    map: textureLoader.load( "Magnus/moon.png" ),
34    normalMap: textureLoader.load ("Magnus/moon.png"),
35  } );
36
37  esfera = new THREE.SphereGeometry (500, 50, 50);
38
39  obj.push (new THREE.Mesh (esfera, textura));
40  scene.add (obj[9]);
```

- Um objeto articulado (uso movimento relativo)

  Objeto articulado: Monstro no planeta Quom (na parte de "cima"); No trecho de código contido em 2.4 é feita a criação do objeto articulado, bem como sua inserção na cena.

```
1    mixer = new THREE.AnimationMixer( scene );
2
3          var loader = new THREE.JSONLoader();
4          loader.load( 'articulado/monster.js', function (
               geometry, materials ) {
5
6            var material = materials[ 0 ];
7            material.morphTargets = true;
8
```

```
9             var mesh = new THREE.Mesh( geometry, materials );
10            mesh.scale.set( .1, .1, .1 );
11
12            mesh.position.x = obj[7].position.x;
13            mesh.position.y = obj[7].position.y + 4000;
14            mesh.position.z = obj[7].position.z;
15
16            scene.add (mesh);
17
18            mixer.clipAction( geometry.animations[ 0 ], mesh )
19                .setDuration( 1 )
20                .startAt( - Math.random() )
21                .play();
22        } );
23
24
25 \end{itemize}
26
27 \section{Requisitos Extras}
28 \begin{itemize}
29     \item Mais objetos, texturas, shaders, c meras, etc ...
30 \par S o utilizados v rios objetos e texturas, sendo
      carregados 100 cometas para visualiza  o e  poss vel
       movimentar a c mera de forma que permita visualizar
      todo o cen rio de diferentes perspectivas, inserindo
      valores na faixa de $1$ a $9$ ou com as \textit{arrow
      keys}. Como  poss vel visualizar nos trechos de
      c digo em \ref{lst:alg6} est o sendo inseridos 100
      cometas em posi  es aleat rias do cen rio.
31
32 \label{lst:alg6}
33 \begin{lstlisting}[style=htmlcssjs]
34   //--------------ADICIONANDO ESTRELAS
       CENA---------------------------//
35         criaEstrelas (5000);
36         criaCometas (100);
```

...

```
1  function criaCometas (quantidade) {
2    cometa.lenght = 0;
3    for ( var i = 0; i < quantidade; i ++ ) {
4      cometa.push(objLoader ('cometa/', 'cometa.mtl', '
           cometa.obj'));
5
6      cometa[i].position.x =  ( 2.0 * Math.random() - 1.0 );
7      cometa[i].position.y = ( 2.0 * Math.random() - 1.0 );
8      cometa[i].position.z =  ( 2.0 * Math.random() - 1.0 );
9
10     cometa[i].position.normalize();
11     cometa[i].position.multiplyScalar (Math.random() * 10000
           + 1000);
12
13     cometa[i].rotation.x = Math.random() * Math.PI;
14     cometa[i].rotation.y = Math.random() * Math.PI;
15     cometa[i].rotation.z = Math.random() * Math.PI;
16
17     cometa[i].scale.multiplyScalar (20 + 80 * Math.random())
           ;
18
19     cometa.lenght++;
20     scene.add( cometa[i] );
21   }
22 }
```