Universidade Federal da Bahia - UFBA
Instituto de Matemática - IM
Departamento de Ciência da Computação - DCC
Curso de Bacharelado em Ciência da Computação

MATA65 - Computação Gráfica Período: 2018.1 Data: 10/04/2018.
Prof. Antonio L. Apolinário Júnior
Estagiário Docente: Rafaela Alcântara

Atividade 0 – Conceitos Básicos do Three.js

Objetivos:

- Compreender de forma prática dos **conceitos básicos da biblioteca *Three.JS***
- Construir objetos gráficos simples, baseados nas primitivas geométricas disponibilizadas na biblioteca ***Three.JS***.

Conceitos básicos:

Uma das formas mais práticas para representação virtual de objetos 3D se dá a partir do uso da malha de triângulos: um conjunto de vértices e faces interconectados que permitem que um objeto 3D seja representado de acordo com a superfície de sua fronteira.

Em *Three.JS*, uma malha de triângulos pode ser facilmente construída a partir do uso dos atributos `vertices` e `faces` do tipo `THREE.Geometry.` O atributo `vertices` consiste em um vetor de vértices, sendo que cada vértice consiste em uma posição 3D do objeto, instância do tipo `THREE.Vector3`. Já o atributo `faces` consiste em um vetor de triângulos do tipo `THREE.Face3`, sendo que cada triângulo descreve como um conjunto de três vértices está conectado entre si.

Roteiro:

1. Baixe do Moodle os códigos fonte e as dependências base para esse Laboratório (Three.JS e Assets). Descompacte no diretório que será visível pelo *WebServer*.

2. Configure o servidor web[1] e execute cada um dos exemplos desse Laboratório.

3. Abra os códigos e analise cada um dos exemplos que compõe esse Laboratório. Entenda a diferença de representação dos objetos para cada exemplo.

4. Tomando como base o exemplo `TriangleFan.html/TriangleFan.js,` crie um modelo representando o **Pacman**, considerando o formato e a cor mostrados na figura ao

[1] no ambiente Windows, utilize o programa USB WebServer (http://www.usbwebserver.net/en/).
no ambiente Linux rode o comando: `python -m SimpleHTTPServer 8000` (onde 8000 é o número da porta) no diretório que será visível pelo *browser*.

lado. Execute o exemplo no servidor web e verifique se o **Pacman** criado se assemelha ao representado na figura abaixo.

5. Transforme o triângulo mostrado no exemplo **`3DFace.html/3DFace.js`** em um plano com 4 vértices e 2 faces, sendo que cada vértice deve ter uma cor diferente. Execute o exemplo no servidor web e verifique se o objeto foi criado com sucesso.

6. Modifique o exemplo **`TriangleStrip.html/TriangleStrip.js`** para geração explícita de uma malha de triângulos correspondente a um **anel**. Habilite a visualização por *wireframe* no material do objeto para visualizar a triangulação feita na criação do anel. Execute o exemplo no servidor web e verifique se o objeto foi criado com sucesso.

7. Modifique o exemplo **`Cube.html/Cube.js`** para que seja possível visualizar mais faces do cubo simultaneamente na tela, pelo menos 2 ou 3. Execute o exemplo no servidor web e verifique se o cubo foi visualizado com sucesso.

8. Analise quais alterações deveriam ser feitas no script gerado no item 7 para que a mudança de visão do cubo pudesse ser feita de forma automática, como uma animação. Implemente essas alterações de forma que possam ser aproveitadas nos próximos itens.

9. Crie um novo exemplo **`Tetrahedron.html/Tetrahedron.js`** com base no exemplo **`Cube.html/Cube.js`** que gere explícitamente uma malha de triângulos correspondente a um tetraedro. Como o cubo, cada face do tetraedro deve ter uma cor diferente. Execute o exemplo no servidor web e verifique se o objeto foi criado com sucesso.

10. Crie um novo exemplo **`Cylinder.html/Cylinder.js`** com base nos exemplos **`Cube.html/Cube.js`**, **`TriangleFan.html/TriangleFan.js`** e **`TriangleStrip.html/TriangleStrip.js`** que gere explícitamente uma malha de triângulos correspondente a um cilindro. O cilindro deve ter uma cor única. Execute o exemplo no servidor web e verifique se o objeto foi criado com sucesso.

11. Modifique o exemplo **`simpleLoadOBJ.html/simpleLoadOBJ.js`** para que cada vértice do modelo do **`Bunny`** tenha sua cor de seu vértice calculada de acordo com a suas coordenadas, normalizadas no intervalo [0.0, 1.0]. Essa coordenada normalizada poderá ser usada como valor RGB na definição da cor do objeto. Execute o exemplo no servidor web e verifique se as cores geradas estão em conformidade com o que foi proposto;

12. **Desafio**: Crie um novo padrão de cores baseado no sistema de cores **HSL** e aplique-o ao mesmo modelo **`Bunny`**.