1)Introdução:

O objetivo deste livro é ensinar como construir e exibir uma cena 3d em uma página web usando os recursos do html5 como a canvas3d e a biblioteca three.js.

1.1)Convenções

Terminal: quando no texto houver referência a terminal isso significa usar o powershell se o seu sistema for windows, bash se linux ou mac.

1.2) Pré-requisitos

Conhecimento de Javascript e HTML. Assume-se que quem está lendo saiba o básico de programação em Javascript e de criação de páginas HTML
Servidores HTTP: É preciso saber que existem servidores HTTP, que é com servidores HTTP que os navegadores interagem. Mais do que isso não é necessário.

2)Hello World

O hello world é dividido em duas etapas: preparação do ambiente e exibição de uma cena mínima e serve para garantir que tudo está funcionando. O ambiente mínimo necessário é um editor de texto adequado, um sistema de controle de versão e um servidor web para servir as páginas. O controle de versão não é necessário mas é recomendado para ajudar a não perder o progresso.

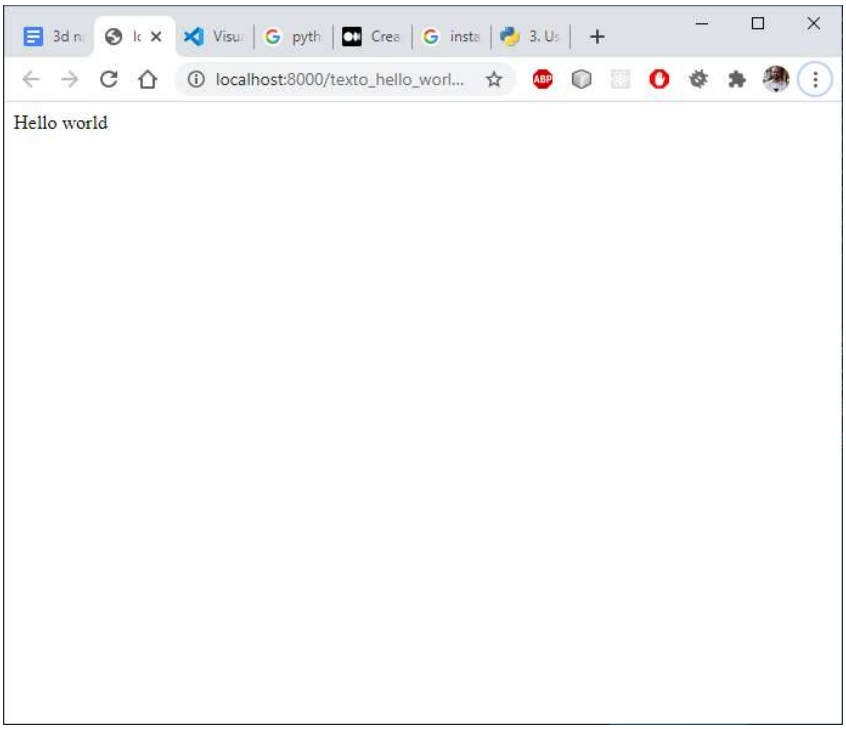
2.1)Editor de texto

O recomendado é o Visual Studio Code (https://code.visualstudio.com/). Baixe-o e instale-o.

2.2)Um servidor

- O que é realmente necessário é um servidor http. Escolhi o python e seus servidores http simples pela praticidade de usá-los mas se você estiver mais confortável com outro servidor não fará diferença, desde que você saiba como servir páginas web em seu servidor http.
- Instale o Python, caso o seu sistema operacional já não o tenha. Para verificar se o seu sistema operacional já tem python abra o terminal e execute o comando `python --version` . Se houver python será mostrada a versão do python instalado. Se o seu sistema operacional for linux ou mac abra o terminal e execute o comando `python --version`.

- Caso o python não tenha sido instalado em sua máquina siga os procedimentos específicos de cada sistema para a instalação: (https://docs.python-guide.org/starting/install3/linux/ , https://docs.python.org/3/using/windows.html).
- Uma vez que a questão do python tenha sido resolvida ele pode ser usado para criar um servidor web com o conteúdo de uma pasta usando o comando `python -m SimpleHTTPServer 8000` (python 2.x) ou `python -m http. server 8000.` (python 3.x) no terminal. Com esse comando o conteúdo da pasta onde ele é executado é servido via http e disponível para acesso nos navegadores em http://localhost:8000.
- Para testar a criação do servidor web crie uma nova pasta, copie o arquivo texto_hello_world.html para esse diretório e use o comando de inicialização do servidor apropriado para a versão do python instalada em sua máquina. Uma vez inicializado o servidor, abra o endereço http://localhost:8000/texto_hello_world.html em um navegador. O resultado deve ser uma página:



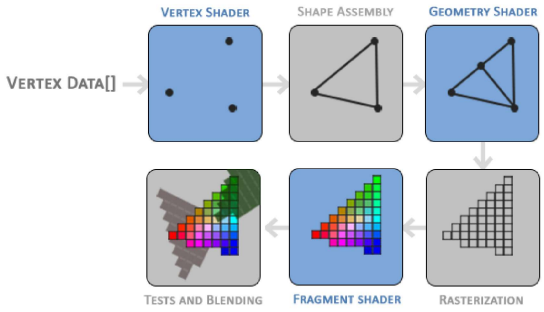
- Se a página não aparecer é porque algo bloqueou o servidor, verifique permissão de administrador e firewalls. Se tudo deu certo, nós temos um servidor web pronto para servir nossa página web com uma cena 3d.

2.3)OpenGL e ThreeJS

- A Three.js é uma biblioteca criada para simplificar o uso do OpenGL em páginas web. O HTML5 tem a canvas3d, um elemento de tela que cria um contexto OpenGL-ES. A Three.js serve para facilitar o uso do OpenGL.
- O que é o OpenGL? O OpenGL é um padrão de comandos e estruturas de dados para desenho 3d. O OpenGL-ES é a implementação para dispositivos móveis. O OpenGL

define as funções e estruturas de dados para o desenho 3d mas nada diz sobre onde exibir. Para isso existem os chamados contextos. A Canvas3d do HTML provê um contexto para o OpenGL em páginas web.

- O funcionamento básico do OpenGL é mostrado nos diagramas abaixo. O OpenGL funciona baseado em programas que são executados na placa de vídeo chamados de Shaders. Estes programas são escritos em algum dialeto da GLSL (GL Shader Language) e recebem como parâmetros Atributos e Valores Uniformes. Atributos são coisas como vértices, coordenadas de texturas, vetores normais, que variam de vértice para vértice. Valores uniformes são constantes no processo de renderização em andamento. Atributos e uniformes são fornecidos ao vertex shader, que os transforma usando operações de álgebra linear e o resultado do vertex shader são valores interpolados. O vertex shader é executado uma única vez para cada vértice. Esses valores interpolados são fornecidos para o fragment shader, junto com os valores uniformes. O fragment shader os transforma em cores na tela e é executado uma única vez para cada fragmento, que em situações normais corresponde a um pixel na tela. O resultado final do processo são os pixels na tela coloridos mostrando a cena 3d. Efeitos como profundidade, sombras, cores, reflexos, são programados nos shaders. Existem outros shaders mas para os propósitos deste livro só serão usados os vértices e fragment shaders.



- A ThreeJS torna o uso do OpenGL eficiente ao operar em um nível de abstração superior. O OpenGL só sabe como desenhar pontos, linhas e triângulos, a ThreeJS sobe o nível de abstração ao ter conceitos como cenas, luzes, sombras, materiais, texturas e ter matemática 3d. Sem a ThreeJS seria necessário que nós escrevêssemos tudo isso nós mesmos.

2.4)Hello ThreeJS

- Vamos agora criar um primeiro projeto da ThreeJS. Ele mostrará um cubo na cena.
- Criemos uma pasta chamada 01_hello_world.

- Nosso projeto não está usando o sistema de módulos do javascript moderno. Para adicionar a ThreeJS baixaremos a versão compilada em https://threejs.org/build/three.js e a colocaremos no diretório. Se estivéssemos usando os sistemas modernos como NPM teríamos que instalar os módulos da Three.
- Criaremos a index.html nesse diretório. No momento ela é só uma página html:

```
<html>
    <head>
    </head>
    <body>
        <div>Hello world</div>
    </body>
</html>
```

- Se ativarmos o servidor aqui (ver seção 2.2) e abrirmos no navegador veremos a página em sua forma inicial.
- Adicionaremos a three.js à página e o bloco onde colocaremos nosso javascript. O bloco de script está no final para garantir que os elementos da página existam quando a execução dos scripts começar. Se estiver no começo comandos da família getElementBy* falharão pois os componentes ainda não existem.

```
    <body>
        <div>Hello world</div>
        <script src="three.js"></script>
        <script>
        </script>
    </body>
```

- Para exibir algo na ThreeJS precisamos de uma câmera, uma cena e um renderer. A câmera tem as instruções de projeção dos vértices para a tela, encapsulando a matemática do 3d que projeta e transforma os pontos de acordo com as propriedades da câmera. A Cena é um grafo que organiza os objetos de um mundo 3d, determinando relações de pai-filho que afetam as operações de álgebra linear como rotação e translação. O renderer é o contexto do OpenGL, a implementação do OpenGL que recebe os dados e os transforma em imagem na tela. Para exibirmos a cena criaremos uma div de destino onde a ThreeJS criará o contexto de OpenGL por baixo dos panos e a cena será exibida. Esse código cria uma cena, uma câmera e um renderer. O resultado é uma tela preta. Não é muito impressionante mas é o primeiro uso da ThreeJS:

```
   <body>
      <div>HELLO THREE!</div>
        <div id="target_div" style="width: 500px; height: 500px;"></div>
        <script src="three.js"></script>
        <script>
            const targetDiv = document.getElementById("target_div");
```

```
        const scene = new THREE.Scene();
        const camera = new THREE.PerspectiveCamera(75,
            targetDiv.clientWidth / targetDiv.clientHeight, 0.1, 1000);
        const renderer = new THREE.WebGLRenderer();
        renderer.setSize(targetDiv.clientWidth, targetDiv.clientHeight);
        targetDiv.appendChild(renderer.domElement);
    </script>
</body>
```

- Indo por partes: A target_div é a div onde o ThreeJS irá criar o contexto OpenGL. Ela é guardada na variável targetDiv ao ser pega da página com getElementById. É importante que tanto a largura quanto a altura do componente estejam definidos de alguma forma. No código acima, pela div estar vazia eu tenho que definir sua altura pois uma div vazia tem altura zero por padrão. Se ela tivesse algum componente dentro dela ela teria a altura do componente.
- Scene é a nova cena de ThreeJS criada.
- Camera é a nova câmera criada. Existem vários tipos de câmera e eu escolhi criar uma do tipo perspectiva. Seu construtor pede os atributos necessários para definir a matriz de projeção perspectiva: o primeiro parâmetro é o Field-of-View em graus, que é a abertura do campo de visão da câmera em graus, o segundo parâmetro é a razão entre a largura e altura da câmera, o terceiro e quarto parâmetros são distâncias de corte. Tudo que estiver a uma distância menor que o terceiro e maior que o quarto parâmetro não será renderizado pela câmera. Para mais informações sobre a matemática da projeção consulte http://www.songho.ca/opengl/gl_projectionmatrix.html
- Renderer é o novo renderer. Depois de criado nós determinamos o tamanho dele em pixels como sendo iguais à largura e altura da div e então adicionamos o elemento de DOM criado pela Three no renderer aos filhos da div

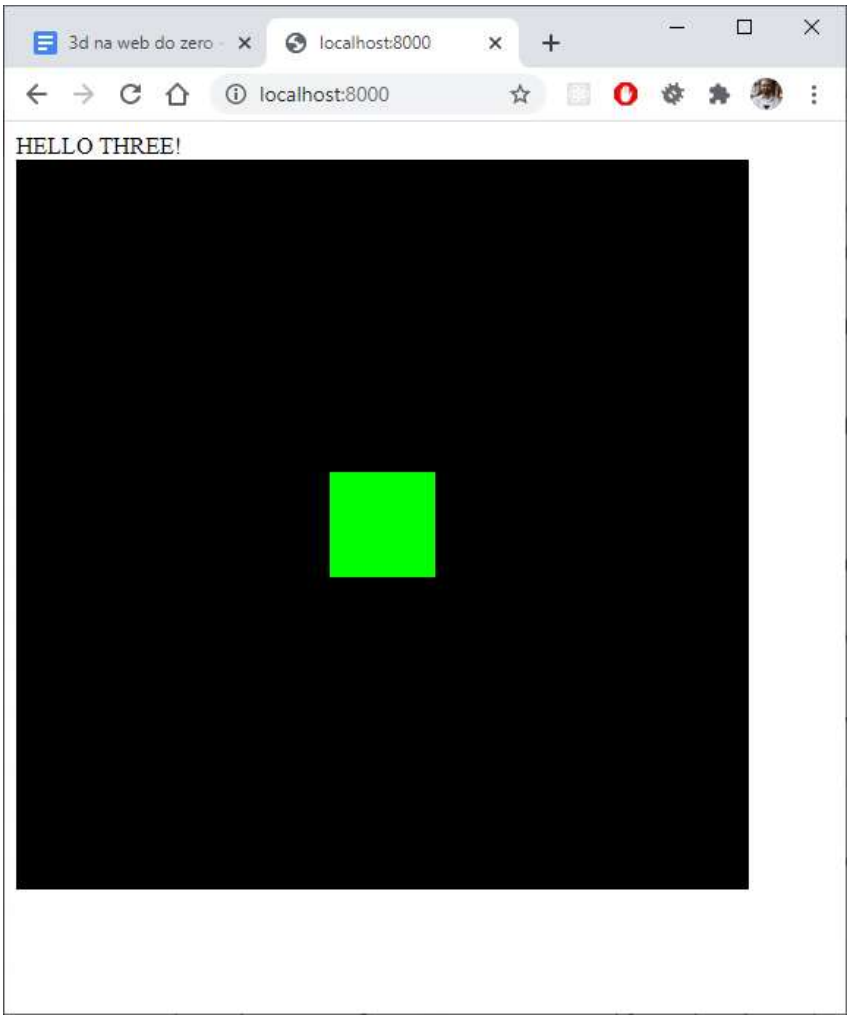
- Quando a página rodar o resultado será uma tela preta:

- Se algo deu errado e estivermos usando o Chrome (recomendado) podemos entrar no modo desenvolvedor pressionando F12. Na aba sources temos o código fonte e podemos por breakpoints para execução passo-a-passo, ver os erros que ocorreram, ver o valor das variáveis caso a execução esteja parada, etc.
- Agora é a criação do cubo e do loop de renderização, a ser posto depois de targetDiv.appendChild(renderer.domElement):

```
        const geometry = new THREE.BoxGeometry();
        const material = new THREE.MeshBasicMaterial({color:0x00ff00});
        const cube = new THREE.Mesh(geometry, material);
        scene.add(cube);
        camera.position.z = 5;
        function animate(){
```

```
            requestAnimationFrame(animate);
            renderer.render(scene, camera);
        }
        animate();
```

- O cubo é uma Mesh. Uma mesh é um conjunto de geometria (os vértices, coordenadas de textura e normais que definem um objeto) e material (como essa geometria será tingida pelas cores). Primeiro criamos a geometria de uma caixa (a BoxGeometry). Depois criamos um material básico que recebe uma cor (o MeshBasicMaterial).
- Com a geometria e o material criamos uma mesh, o cubo.
- Uma vez criado o cubo o adicionamos à cena (scene.add).
- Mudamos a posição Z da câmera, para ela sair de (0,0,0) para (0,0,5).
- Definimos uma função animate. Essa função tem o ciclo de animação da cena, com a cada execução dela pedindo um novo quadro de animação para o navegador e usando o renderer criado acima para renderizar a cena usando a câmera).
- Por fim mandamos executar a animate pela primeira vez. O resultado é um cubo verde na tela:



- Com isso concluímos o nosso primeiro programa com a three.js. Não é muito mas é o fundamento de todos os programas 3d na web. Todos criarão uma cena, colocarão meshes com geometrias e materiais diversos nessa cena e a renderização continuamente usando o renderer em um loop.
- Se esse mesmo programa fosse escrito sem a ThreeJS teríamos que criar a canvas3d manualmente, escrever o vertex shader com a projeção dos vértices usando a matriz de transformação do objeto (rotação, translação, escala), a matriz de transformação da câmera e a matriz de projeção (a chamada matrix MVP), escrever o fragment shader para tingir cada vértice do objeto com uma cor. Depois teríamos que definir manualmente uma lista com os vértices, e empurrar essa lista de vértices como atributos e as matrizes como valores uniformes. O programa de menos de 20 linhas rapidamente atingiria 100 linhas, com grandes chances de erros e com pouca versatilidade devido ao baixo nível de abstração.
- No próximo capítulo será visto como detectar cliques em objetos e reagir a eles.