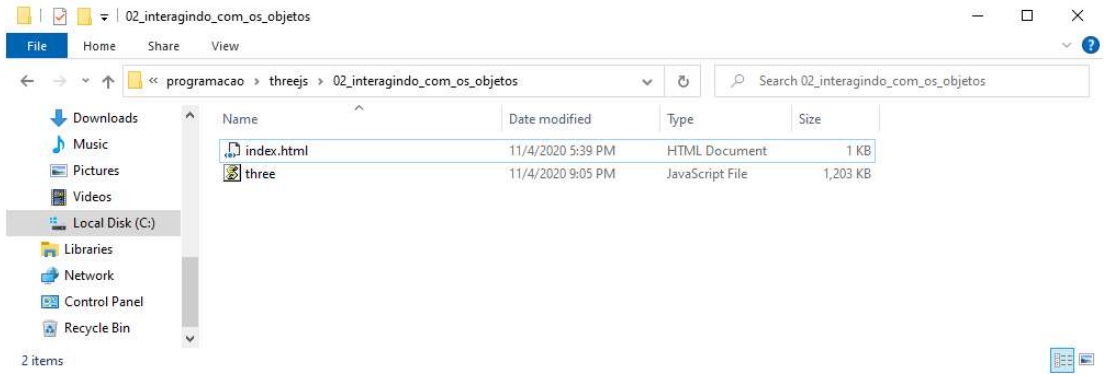
3)Interagindo com os objetos

Uma coisa importante é a capacidade de interagir com os objetos 3d na cena. A principal forma de interação são os mecanismos de toque/clique. Neste capítulo aprenderemos como a partir da posição do clique detectar se um objeto foi clicado e reagir a esse clique.

3.1)Preparando o ambiente

- Criaremos uma nova pasta, 02_interagindo_com_os_objetos e colocaremos nela uma index.html com apenas uma div vazia e o arquivo da biblioteca da three.js:



- Reconstruiremos nessa nova index.html o necessário para uma cena da three.js funcionar:
- Criaremos uma div dentro do body para conter a cena:

```
<body>
        <div>Hello world</div>
        <div id="target_div" style="height:500px"></div>
    </body>
```

- Criaremos o bloco de script no final da body para importar o arquivo da three.js e para definirmos nossos proprios códigos:

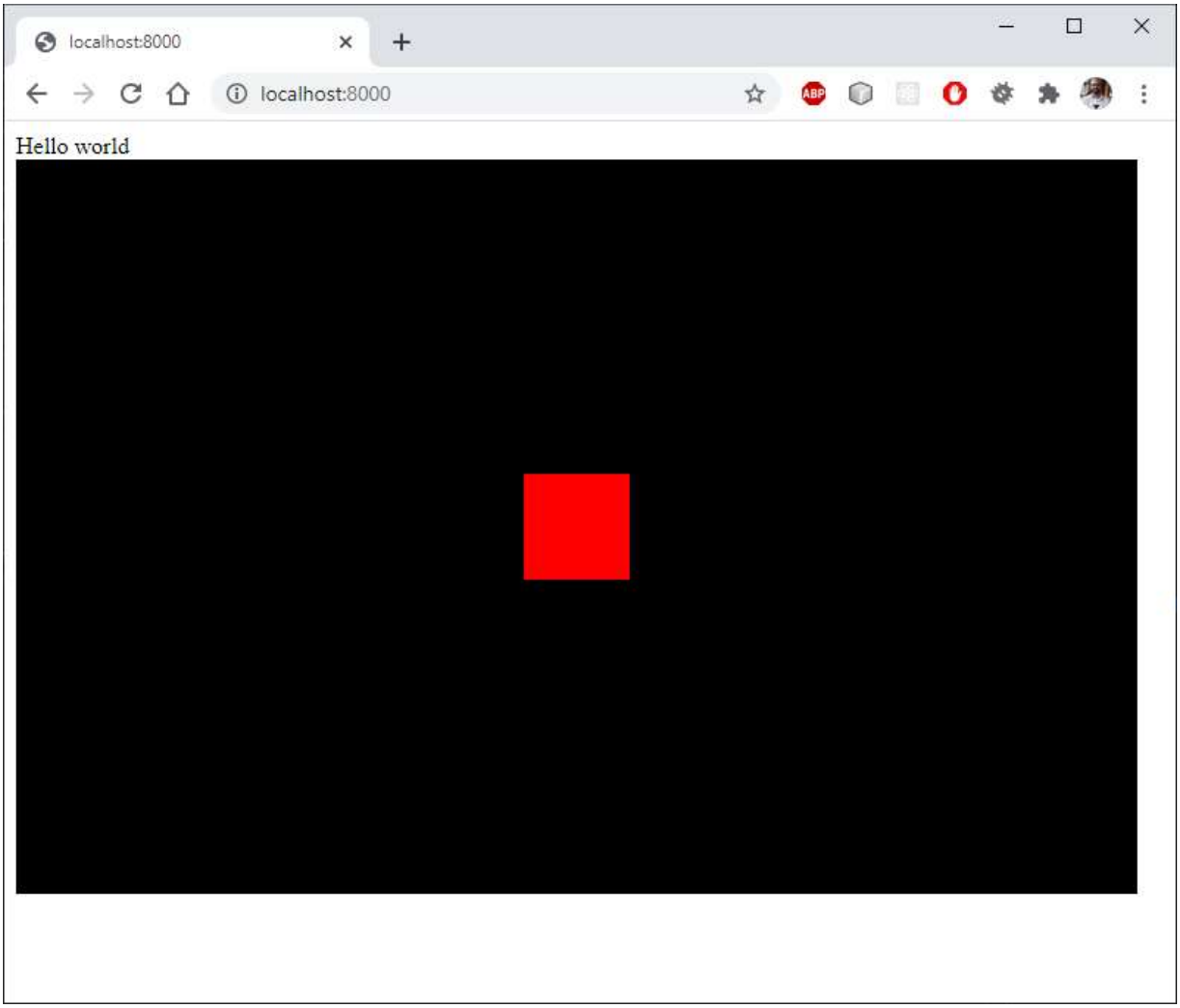
```
<script src="three.js"></script>
<script>
</script>
```

- Criaremos o renderer, a câmera, a cena e o cubo:

```
        <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);
            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();
    </script>
```

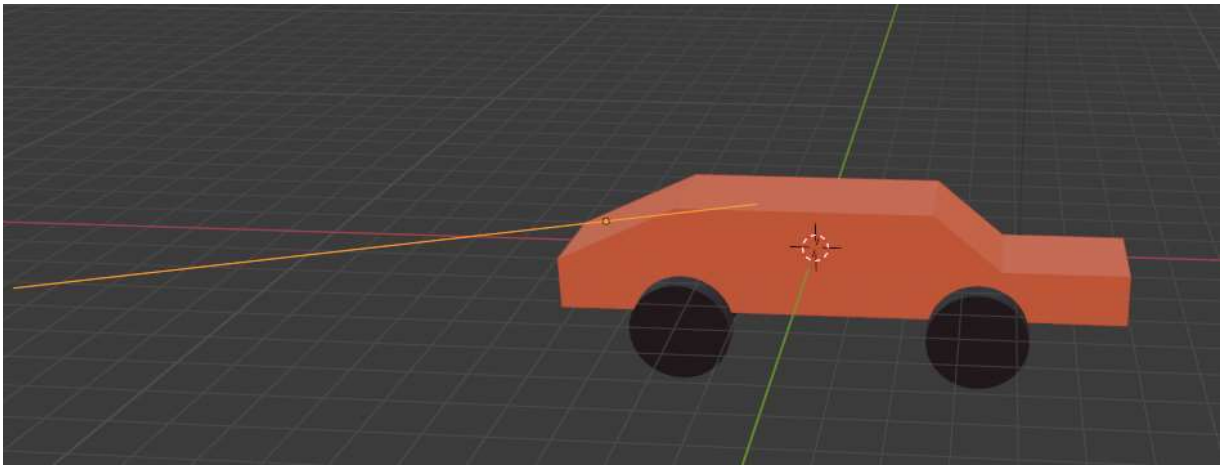
- Com isso chegamos ao ponto em que paramos na seção anterior:



- Agora começam as novidades: O objetivo é determinar em qual objeto o usuário clicou. O nome desse processo é picking.

3.2) Picking

- Existem dois tipos de picking. Um baseado na CPU e mais simples de implementar mas que não lida com casos especiais como texturas com buracos transparentes e um baseado em GPU que é mais difícil de implementar mas lida com casos especiais. Daremos foco ao primeiro tipo.
- O algoritmo consiste em obter o ponto no sistema de coordenadas da tela onde o usuário clicou, passar esse ponto para o sistema de coordenadas 3d do mundo e então projetar um raio desse ponto, ao longo da direção em que a câmera está olhando até encontrar objetos. É o processo chamado raycasting.
- Na ilustração abaixo a linha reta laranja é um exemplo de um raio que percorre a cena e intercepta objetos. O raio do picking é gerado a partir da posição da câmera. Sua equação paramétrica é da forma R(t) = P0 + t * V, onde P0 é a posição na tela convertida para o sistema 3d, pertencendo ao plano da câmera, t é o parâmetro e V é o vetor de direção da câmera. Quando a Threejs nos retornar a lista de objetos interceptados pelo raio os retornará ordenados por distância.



- Para implementarmos o raycast em nosso programa precisamos saber onde o usuário clicou, passar essa posição na tela para a Threejs, ordenar o raycast e pegar a lista de objetos que o raio que sai da tela para dentro do mundo 3d interceptou, sendo que essa lista pode ter 0 ou vários.
- A primeira parte é criarmos um objeto Raycaster:

```
camera.position.z = 5;
///NOVO - O Raycaster
const raycaster = new THREE.Raycaster();
```

- Criado o raycaster, criamos um evento para tratar do click na div onde a cena 3d está. Nós já temos a div em nosso código. Então basta criarmos uma função para tratar o evento click. Quando o usuário clicar na div a função definida abaixo será invocada. No momento ela está vazia.

```

///NOVO - Evento de click
```

```
targetDiv.addEventListener('click',(event)=>{
});
```

- Agora devemos passar onde o usuário clicou. O raycaster espera que a posição esteja normalizada. A matemática para normalizar os pontos do mouse é complicada então eu criarei uma função separada, chamada de mousePositionAsNormalizedCoordinates, que recebe o evento de click e o renderer e retorna o Vector2 com a posição normalizada (um Vector2 é um tipo de dado da Threejs que é o par ordenado xy).:

```
function mousePositionAsNormalizedCoordinates(event, renderer){
     const canvas = renderer.domElement;
     const rect = canvas.getBoundingClientRect();
     const pos = {
          x: (event.clientX - rect.left) * canvas.width  / rect.width,
          y: (event.clientY - rect.top ) * canvas.height / rect.height,
     };
     const pickPosition = new THREE.Vector2(0,0);
     pickPosition.x = (event.x / canvas.width ) *  2 - 1;
     pickPosition.y = (event.y / canvas.height) * -2 + 1;  // note we flip Y
     return pickPosition;
}
```

- Definida essa função que encapsulará o processo de normalização do ponto do mouse o event listener de click pode obter a lista de objetos que o mouse intercepta. A lista de objetos interceptados pelo raio (interceptedObjects) contém objetos com informações da interceptação como em qual face do objeto houve a interceptação, a distância onde ela ocorreu, a coordenada de textura de onde ela ocorreu, o ponto no espaço e por fim o objeto em si.

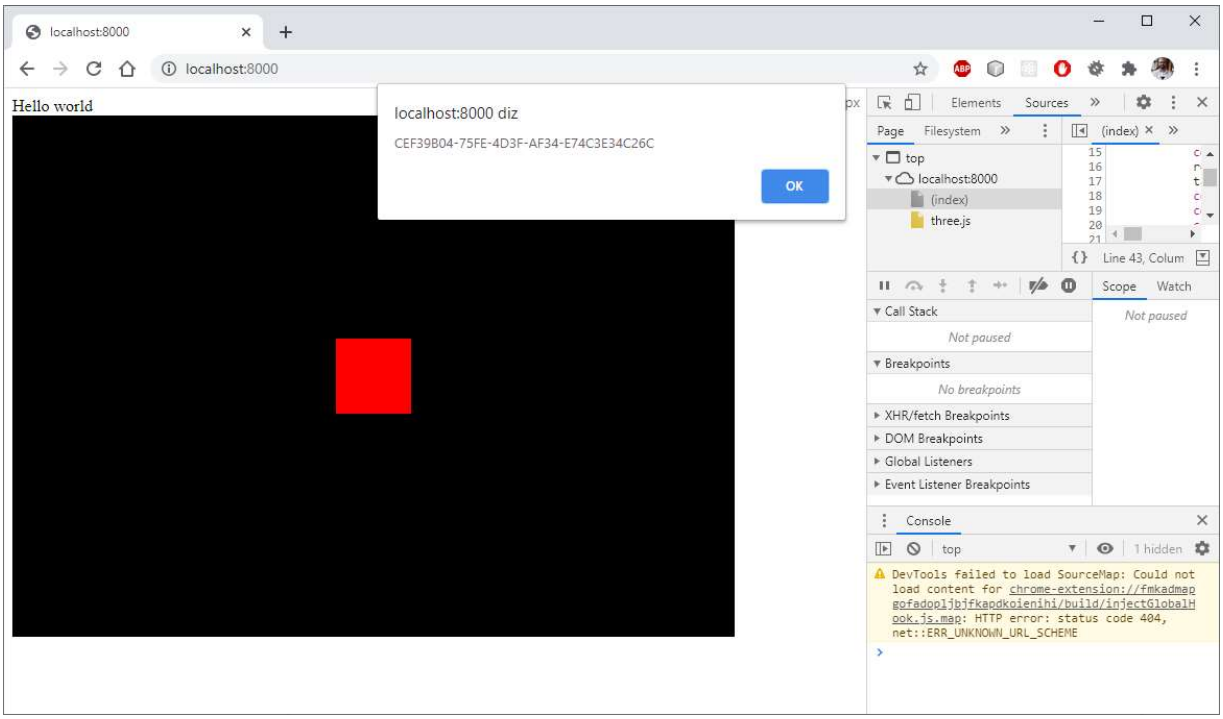
```
targetDiv.addEventListener('click',(event)=>{
  const pickPosition = mousePositionAsNormalizedCoordinates(event, renderer)
  raycaster.setFromCamera(pickPosition, camera);
  const interceptedObjects = raycaster.intersectObjects(scene.children,
                  true);
  if(interceptedObjects.length > 0 ){
     alert(interceptedObjects[0].object.uuid);
  }});
```

- O que os resultados de uma interceptação carregam como informação (visto no debugger do chrome):

```
interceptedObjects[0]
    {distance: 4.530862634823196, point: Vector3, object: Mesh, uv: Vector2, face:
    Face3, …}
           distance: 4.530862634823196
           face: Face3 {a: 7, b: 2, c: 0, normal: Vector3, vertexNormals: Array(3), …}
           faceIndex: 9
```

```
object: Mesh {uuid: "0054D3E4-F283-428F-A53E-76395A6D631B", name: "amarelo",
type: "Mesh", parent: Scene, children: Array(0), …}
point: Vector3 {x: 1.1000380044951055, y: 0.005490542146381561, z: -1.5,
isVector3: true}
uv: Vector2 {x: 0.6000380044951055, y: 0.5054905421463816, isVector2: true}
__proto__: Object
```

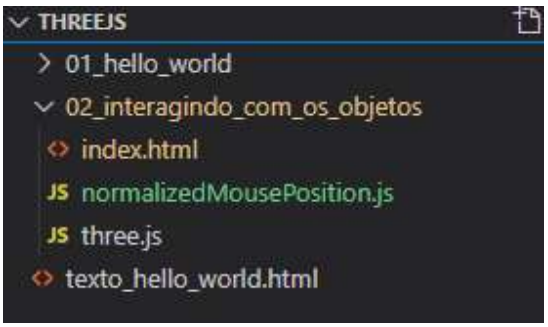
- O resultado é que quando o usuário clicar no objeto um alert com o id do objeto será exibido. Quando clicar fora do objeto, não. Isso acontece porque o raio de interceptação do raycast não interceptou objeto algum até chegar em seu limite. Um objeto é considerado interceptado se sua “bounding box” (a menor caixa virtual possivel capaz de conter todos os vértices do objeto) é interceptada pelo raio e, caso a bounding box seja, a Threejs verifica se realmente a superficie do objeto intercepta o raio. Com isso ela poupa processamento.



3.3) Melhorando o código:

- No momento todo o código está no bloco <script> da página e todas as partes dependem umas das outras, com um grande acoplamento. A medida que o código crescer tal arranjo o torna complexo e difícil de modificar. A forma de evitar tal problema é a separação em partes menores modularizadas, com poucas dependências.
- Nesse momento a parte que pode ser separada é a que calcula a posição normalizada do mouse. Ela é uma função que depende da posição do mouse no evento de click e do renderer para fornecer a canvas3d onde o click está acontecendo (lembre-se que dentro da div há uma canvas3d criada pela Threejs.).

- Criaremos um arquivo chamado normalizedMousePosition.js na mesma pasta onde estamos trabalhando:



- Recortamos a função mousePositionAsNormalizedCoordinates para esse arquivo e o incluiremos nos scripts da index.html, abaixo do script da three.js. O resultado é que o código da index estará mais limpo e isso facilitará modificações no futuro.
- Com isso nós somos capazes de determinar quando um objeto na cena é clicado. A próxima etapa é navegar na cena 3d.
- Index.html no final:

```
<head>
</head>
<body>
    <div>Hello world</div>
    <div id="target_div" style="height:500px"></div>
    <script src="three.js"></script>
    <script src="normalizedMousePosition.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);
        const geometry = new THREE.BoxGeometry();
        const material = new THREE.MeshBasicMaterial({color:0xff0000});
        const cube = new THREE.Mesh(geometry, material);
        cube.name = "MyCube";
        scene.add(cube);
        camera.position.z = 5;
        ///NOVO - O Raycaster
        const raycaster = new THREE.Raycaster();
```

```
            ///NOVO - Evento de click
            targetDiv.addEventListener('click',(event)=>{
                const pickPosition = mousePositionAsNormalizedCoordinates(event,
                    renderer)
                raycaster.setFromCamera(pickPosition, camera);
                const interceptedObjects =
raycaster.intersectObjects(scene.children,
                    true);
                if(interceptedObjects.length > 0 ){
                    alert(interceptedObjects[0].object.uuid);
                }
            });
            function animate(){
                requestAnimationFrame(animate);
                renderer.render(scene, camera);
            }
            animate();
        </script>
    </body>
</html>
```

normalizedMousePosition.js no final:

```
function mousePositionAsNormalizedCoordinates(event, renderer){
    const canvas = renderer.domElement;
    const rect = canvas.getBoundingClientRect();
    const pos = {
        x: (event.clientX - rect.left) * canvas.width  / rect.width,
        y: (event.clientY - rect.top ) * canvas.height / rect.height,
    };
    const pickPosition = new THREE.Vector2(0,0);
    pickPosition.x = (event.x / canvas.width ) *  2 - 1;
    pickPosition.y = (event.y / canvas.height) * -2 + 1;  // note we flip Y
    return pickPosition;
}
```

- Faça experimentos com o que foi aprendido até agora. Dependendo do seu domínio de javascript, atualize um span com a quantidade de vezes que o objeto foi clicado. Outro experimento pode ser mudar o material do objeto clicado, mudando a cor por exemplo. Pode também adicionar vários objetos na cena (lembre de mudar a position deles para eles não ficarem no mesmo lugar) e ver o que acontece quando clica.