Installation de Three.Js

Création d’un dossier contenant le projet

Ajout d’un index.html contenant ce code :

  <script  type="importmap">{

    "imports": {

        "three": "https://threejs.org/build/three.module.js"

    }

}</script>

<script type="module" src="/script.js"></script>

Création d’un fichier script dans ce cas il s’agit de script.js /!\ Il faut préciser le type=’’module’’

Dans mon fichier script j’importe Three :

import \* as THREE from 'three';

On va maintenant installer Three dans notre projet via le Terminal.

Au préalable il faut avoir Node pour utiliser NPM.

Dans le dossier du projet lancer les commandes suivantes :

npm install --save three

npm install --save-dev vite

Lancer le serveur de production avec la commande :

npx vite

Maintenant vous pourrez accéder à votre projet en mode live server en général sur le <http://localhost:5173>

Création de l’espace de travail

Dans le HTML on va venir ajouter une balise canvas avec un ID

                <canvas id="c"></canvas>

Dans notre script.js on va crée une fonction ou sera contenu tout notre développement en Tree, ça sera notre scène principale pour un canvas

function main() {

    const canvas = document.querySelector( '#c' );

    const renderer = new THREE.WebGLRenderer( { antialias: true, canvas } );

Dans notre function main on va déclarer deux constante

La première pour sélectionner notre canvas grâce à son id

La seconde va servir appeler l’api de WebGL qui sert au rendu 3D

   //Je gère la distance de point de vue

    const fov = 75;

    const aspect = 2; // the canvas default

    const near = 0.1;

    const far = 5;

    const camera = new THREE.PerspectiveCamera( fov, aspect, near, far );

    camera.position.z = 4;

    camera.position.y = 0;

fov (Field of View): C'est l'étendue du monde visible à partir de la caméra, en degrés. Un fov de 75 indique un champ de vision relativement large.

aspect (Aspect Ratio): C'est le rapport entre la largeur et la hauteur de la zone de visualisation. Un aspect de 2 indique que la zone de visualisation est deux fois plus large que haute.

near (Near Clipping Plane): C'est la distance minimale à laquelle la caméra peut voir. Les objets plus proches que cette distance ne seront pas affichés.

far (Far Clipping Plane): C'est la distance maximale à laquelle la caméra peut voir. Les objets au-delà de cette distance ne seront pas affichés.

camera.position.z: C'est la position de la caméra sur l'axe Z (profondeur). Une valeur de 4 la place à 4 unités de l'origine.

camera.position.y: C'est la position de la caméra sur l'axe Y (hauteur). Une valeur de 0 signifie qu'elle est au niveau de l'origine verticale.

WebGL est largement supporté dans les navigateurs modernes, mais certains anciens navigateurs ou certains appareils avec du matériel graphique limité peuvent ne pas le supporter, ou il peut être désactivé pour diverses raisons.

Voici un exemple simple de comment on pourrait vérifier la compatibilité WebGL :

if (WEBGL.isWebGLAvailable()) {

        // WebGL est disponible et fonctionnel

    } else {

        // WebGL n'est pas disponible ou fonctionnel

        var warning = WEBGL.getWebGLErrorMessage();

        document.getElementById('container').appendChild(warning);

    }

Désormais on a une scène on va donc vouloir créer un objet 3D sur cette scène :

Imaginons que l’on souhaite avoir un cube 3D

On va déclarer nos constante qui vont gérer le cube :

    const boxWidth = 1;

    const boxHeight = 1;

    const boxDepth = 1;

    const geometry = new THREE.BoxGeometry( boxWidth, boxHeight, boxDepth );

Dans la j’ai définie la forme de mon bloc.

Maintenant je vais le faire apparaitre dans ma scène :

    function makeInstance(geometry, color, x) {

        // Crée un nouveau cube ici

        const material = new THREE.MeshBasicMaterial( { color: 0xffffff } );

        const cube = new THREE.Mesh(geometry, material);

        scene.add(cube);

        cube.position.x = x;

        return cube;

const cubes = [

makeInstance(geometry, 0x44aa88, 0),

makeInstance(geometry, 0x8844aa, -2),

makeInstance(geometry, 0xaa8844, 2),

];

    }

La fonction va donc prendre en compte mes paramtre de forme, la couleur qui est définis ainsi que la position du cube.

On pourra maintenant ajouter de l’éclairage a se cube, une rotation au click and drop, la possibilité de déplacer le cube avec la souris dans le canva etc…

En l’etat mon cube va apparaitre pixélisé je vais donc devoir lui rajouter deux fonctions qui vont adapter le cube a la taille de l’écran (pour le responsive) et va venir dépixéliser le cube en fonction de cette taille d’écran :

function resizeRendererToDisplaySize(renderer) {

        const canvas = renderer.domElement;

        const width = canvas.clientWidth;

        const height = canvas.clientHeight;

        const needResize = canvas.width !== width || canvas.height !== height;

        if (needResize) {

          renderer.setSize(width, height, false);

        }

        return needResize;

      }

    function render() {

        // Redimensionne le rendu à la taille de l'affichage si nécessaire

        if (resizeRendererToDisplaySize(renderer)) {

            const canvas = renderer.domElement;

            camera.aspect = canvas.clientWidth / canvas.clientHeight;

            camera.updateProjectionMatrix();

        };

        renderer.render(scene, camera);

        requestAnimationFrame(render);

    }

    requestAnimationFrame( render );

}

main();

Création de ligne :

On va repartir avec le même script de base mais dans cet exemple je regle la caméra d’une autre façons :

import \* as THREE from 'three';

function main() {

    const canvas = document.querySelector('#c');

    const renderer = new THREE.WebGLRenderer({ antialias: true, canvas });

    renderer.setClearColor(0x000000, 0); // Couleur noire, mais totalement transparente

    const camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight, 1, 500);

    camera.position.set(0, 0, 100);

    camera.lookAt(0, 0, 0);

    const scene = new THREE.Scene();

    const material = new THREE.LineBasicMaterial({ color: 0x0000ff });

Je d’indique la couleur de mes lignes :

const points = [];

    points.push(new THREE.Vector3(20, 0, 0));

    points.push(new THREE.Vector3(0, 10, 0));

    points.push(new THREE.Vector3(10, 0, 0));

    points.push(new THREE.Vector3(50, 0, 0));

    points.push(new THREE.Vector3(0, 50, 0));

Ensuite je vais créer un tableau contenant mes lignes avec leur position en paramètre (x, y, z)

Puis je demande a ce que mes lignes soient créée :

    const geometry = new THREE.BufferGeometry().setFromPoints(points);

    const line = new THREE.Line(geometry, material);

Je clôture mon script avec le code vu précédemment pour le resize et la dépixélisassions :

function resizeRendererToDisplaySize(renderer) {

        const canvas = renderer.domElement;

        const width = canvas.clientWidth;

        const height = canvas.clientHeight;

        const needResize = canvas.width !== width || canvas.height !== height;

        if (needResize) {

            renderer.setSize(width, height, false);

        }

        return needResize;

    }

    function render() {

        // Redimensionne le rendu à la taille de l'affichage si nécessaire

        if (resizeRendererToDisplaySize(renderer)) {

            const canvas = renderer.domElement;

            camera.aspect = canvas.clientWidth / canvas.clientHeight;

            camera.updateProjectionMatrix();

        };

        renderer.render(scene, camera);

        requestAnimationFrame(render);

        scene.add(line);

        renderer.render(scene, camera)

    }

    requestAnimationFrame(render);

    render();

}

main();

function resizeRendererToDisplaySize(renderer) {

        const canvas = renderer.domElement;

        const width = canvas.clientWidth;

        const height = canvas.clientHeight;

        const needResize = canvas.width !== width || canvas.height !== height;

        if (needResize) {

            renderer.setSize(width, height, false);

        }

        return needResize;

    }

    function render() {

        // Redimensionne le rendu à la taille de l'affichage si nécessaire

        if (resizeRendererToDisplaySize(renderer)) {

            const canvas = renderer.domElement;

            camera.aspect = canvas.clientWidth / canvas.clientHeight;

            camera.updateProjectionMatrix();

        };

        renderer.render(scene, camera);

        requestAnimationFrame(render);

        scene.add(line);

        renderer.render(scene, camera)

    }

    requestAnimationFrame(render);

    render();

}

main();