3IN017 – TECHNOLOGIES DU WEB

WebGL et Three.js

8 avril 2025

Gilles Chagnon

Plan

Introduction

Dessiner et animer pour le Web

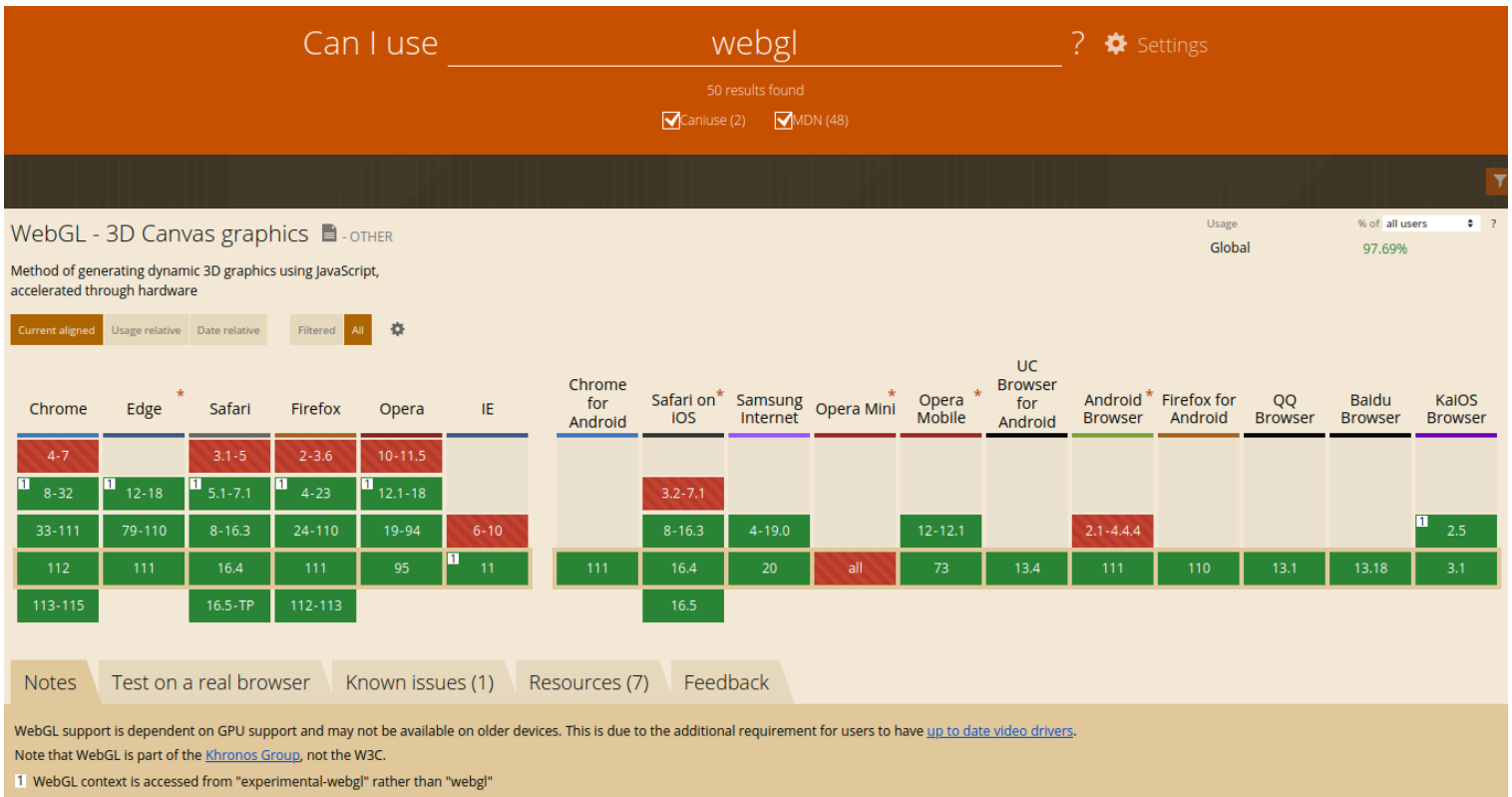
Par le passé : Adobe Flash.

Maintenant, animations CSS mais limitées :

- pas de concept d'objet à animer (on déplace des formes)
- pas de réelle 3D
- pas d'utilisation de l'accélération matérielle du GPU

WebGL

- Une API de 3D dynamique pour HTML5 (élément `canvas`)
- repose sur le pilote OpenGL pris en charge par les OS
- Fonctionnement : élément graphique WebGL dans un document Web ⟶ JS ⟶ API WebGL ⟶ pilote OpenGL de l'OS qui fait les calculs et affiche
- Maintenant bien supportée :



`https://caniuse.com/?search=webgl`

Comment ?

SCIENCES SORBONNE UNIVERSITÉ

On peut utiliser WebGL :

- nativement
- avec des bibliothèques dédiées : BabylonJS (`https://www.babylonjs.com/`), Three.js (`https://threejs.org/`), avec React Three Fiber (`https://github.com/pmndrs/react-three-fiber`) ou React Unity, très utilisé pour les jeux (`https://react-unity-webgl.dev/`)

Nous allons nous focaliser sur Three.js.

Three.js : les étapes

Construction progressive

1. le code HTML
2. la scène
3. les caméras
4. le moteur de rendu
5. les objets
6. les sources de lumière

Le code HTML avec canvas

canvas est un élément HTML5 vide, mais dans lequel on va pouvoir « dessiner » avec JavaScript :

```
<canvas id="canvas1"></canvas>
```

Utiliser Three.js

Par exemple avec Vite :

1. `npm create vite@latest`, puis Vanilla et JavaScript
2. `npm install three`

Puis dans un fichier `demo.js` :

```
import * as THREE from "three";

function main() {
        const canvas = document.getElementById('canvas1');

// Code de la scène
}
main();
```

Dans le document HTML :

```
<div id="app">
        <canvas id="canvas1">
                <p>Votre navigateur ne prend pas en charge les canevas.</p>
        </canvas>

</div>
<script type="module" src="demo.js"></script>
```

La scène

Comme pour une pièce de théâtre, c'est dans la scène qu'on va disposer objets, sources de lumière et caméras.

C'est facile, il suffit de l'ajouter :

```
import * as THREE from "three";

const scene = new THREE.Scene();
```

Le moteur de rendu

C'est ce qui va permettre de spécifier comment la scène va être restituée. Par exemple...

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

(`canvas` est le nom de la variable associée à l'élément `canvas` de la page *via* `const canvas = document.getElementById('canvas1');`). L'antialias évite les problèmes de crénelage.

La caméra

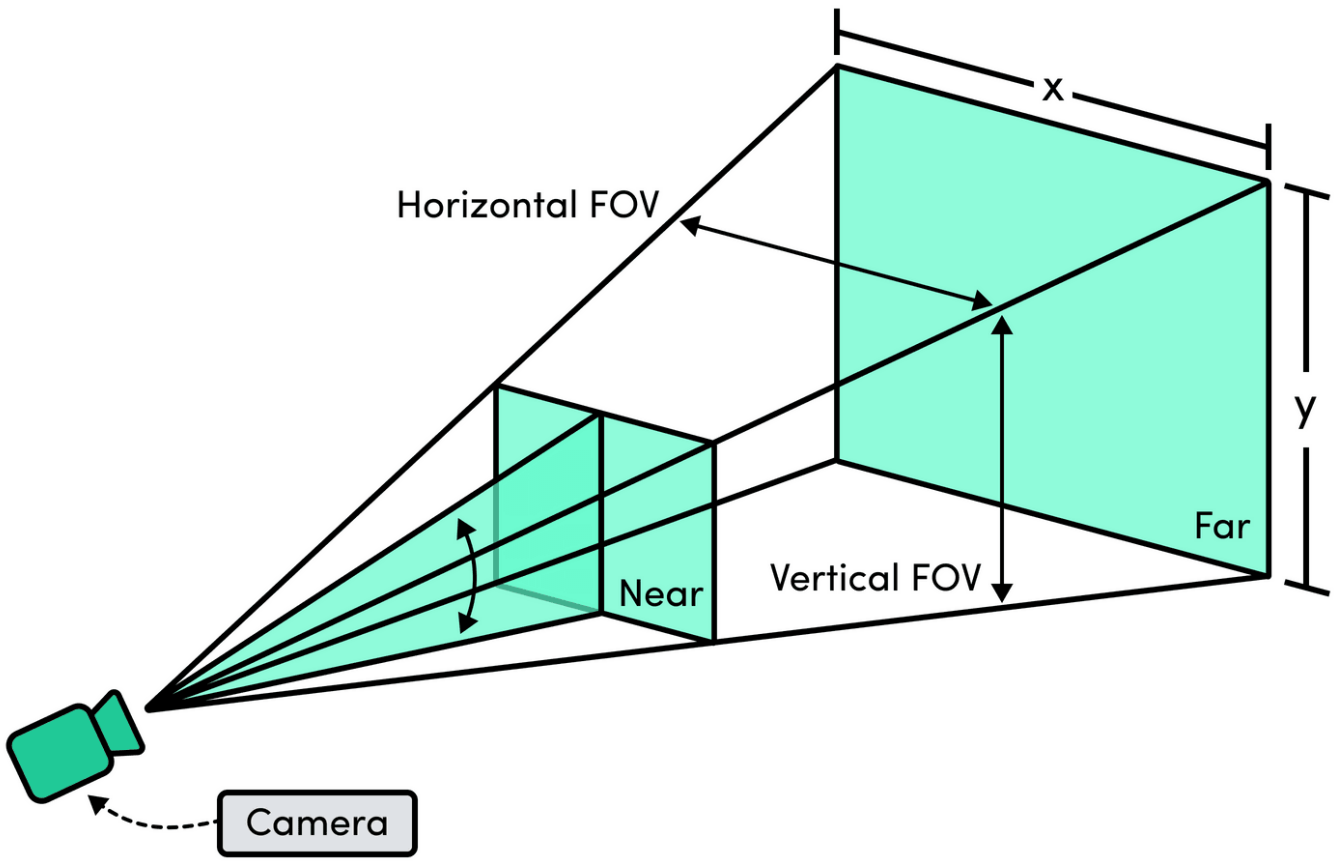
Il y a plusieurs types de caméra avec Three.js. La plus courante est `PerspectiveCamera` :

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

. . . avec 4 paramètres :

- fov : pour *field of view*, le champ de vue vertical en degrés (souvent aux alentours de 50°)
- aspect : pour *aspect ratio*. On le fixe d'habitude au rapport largeur du canvas/hauteur du canvas
- near : distance minimale à partir de la caméra pour commencer à afficher des objets
- far : distance maximale à partir de la caméra au-delà de laquelle aucun objet n'est visible.

Schéma global



Les objets : principe

Ils sont caractérisés par

- leur forme (géométrie)
- leur texture (*material*) qui détermine comment l'objet apparaît dans la lumière
- leur *mesh* qui combine géométrie et texture

Par exemple...

```
const boxWidth = 1;
const boxHeight = 1;
const boxDepth = 1;
const geometry = new THREE.BoxGeometry(boxWidth, boxHeight, boxDepth);

const material = new THREE.MeshPhongMaterial({color: 0x44aa88});

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

Les objets : paramètres

Géométries

BoxGeometry, CapsuleGeometry, CircleGeometry, ConeGeometry, CylinderGeometry, DodecahedronGeometry, EdgesGeometry, ExtrudeGeometry, IcosahedronGeometry, LatheGeometry, OctahedronGeometry, PlaneGeometry, PolyhedronGeometry, RingGeometry, ShapeGeometry, SphereGeometry, TetrahedronGeometry, TorusGeometry, TorusKnotGeometry, TubeGeometry, WireframeGeometry

Les textures

LineBasicMaterial, LineDashedMaterial, Material, MeshBasicMaterial, MeshDepthMaterial, MeshDistanceMaterial, MeshLambertMaterial, MeshMatcapMaterial, MeshNormalMaterial, MeshPhongMaterial, MeshPhysicalMaterial, MeshStandardMaterial, MeshToonMaterial, PointsMaterial, RawShaderMaterial, ShaderMaterial, ShadowMaterial, SpriteMaterial

Il faut regarder la documentation ! Chaque géométrie et chaque texture ont leurs propres paramètres.

On ne voit rien

Enfin, il faut ajouter l'objet à la scène :

```
scene.add(cube);
```

On ne voit toujours rien

Il faut encore ajouter au moins une source de lumière, par exemple...

```
const color = 0xFFFFFF;
const intensity = 1;
const light = new THREE.DirectionalLight(color, intensity);
scene.add(light);
```

Mais rien de rien...

Nous avons créé la scène, un objet, une caméra, une source de lumière... mais en fait par défaut tous ces éléments sont placés au point de coordonnées (0,0,0) et donc on ne voit toujours rien. Il faut préciser la position de ces éléments :

```
camera.position.z=2;
light.position.set(-1, 2, 4);
cube.position.x=-1;
```

Animations

On utilise la fonction standard `requestAnimationFrame` qui prend un *callback* en paramètre (en fait une méthode de l'objet `window`) :

```
function BougePetitCubeBouge(time) {
time *= 0.001;  // convert time to seconds

cube.rotation.x = time;
cube.rotation.y = time;

renderer.render(scene, camera);

requestAnimationFrame(BougePetitCubeBouge);
}
requestAnimationFrame(BougePetitCubeBouge);
```