Introducción a Three.js

Visualización 3D en el navegador

Grupo Universitario de Informática

Instalación

- Creamos la carpeta donde queramos el proyecto.
- Descargamos `npm (Node Package Manager)`
- Abrimos una terminal (`bash` o `powershell`)
- Ejecutamos `npm create vite@latest` y despues elegimos el nombre de proyecto que queramos, y muy importante seleccionar `JavaScript vanilla` y no otras configuraciones.
- Es posible que al ejecutar este comando nos pidan instalar `Vite`, pero `npm` lo hace por nosotros con tan solo indicar que sí queremos descargarlo.

Instalación II

- Una vez realizado todo lo anterior, nos movemos a la carpeta que ha creado `Vite`. (`cd <nombre_carpeta>`)
- Ahora instalamos `Three.js` con el siguiente comando: `npm install three`
- Una vez instalado `Three.js` ya podemos ponernos con el código.
- Si queréis comprobar que tenéis esta libreria descargada podéis ejecutar `npm list three`.
- Para hacer funcionar nuestro servidor web local, tan solo hace falta ejecutar `npm run dev`

Componentes esenciales

```
import * as THREE from "three";
// Escena
const scene = new THREE.Scene();

// Camara
const camera = new THREE.PerspectiveCamera(
  75,                                    // fov
  window.innerWidth / window.innerHeight, // relacion de aspecto
  0.1,                                   // near
  1000                                   // far
);

// Renderer
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

- Escena: Contenedor 3D
- Cámara: Perspectiva/Visión
- Renderer: Motor gráfico

Objetos 3D

```
// Geometria + Material
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;
```

- Se pueden usar todo tipo de geometrías
- Los materiales definen cómo se ve un objeto
- Mesh (Malla) = Geometría + Material

Iluminación Global

- Es el tipo de iluminación más básico.
- Afecta en cualquier espacio y sobre cualquier superficie de nuestro modelo.
- No genera sombras.

```
const ambientLight = new THREE.AmbientLight(0xffffff, 0.5);
//Color blanco, intensidad 0.5
scene.add(ambientLight);
```

Iluminación Direccional

- Proyecta lineas paralelas desde una determinada dirección.
- El sol emite este tipo de luz, por ejemplo.
- Sí proyecta sombras. (aunque hay que activarlas)

```
const directionalLight = new THREE.DirectionalLight(0xffffff,1);
// Color blanco, intensidad 1
directionalLight.position.set(5, 10, 5);
// Posición en la escena
scene.add(directionalLight);
```

Iluminación puntual

- Proyecta rayos de luz en todas las direcciones.
- Una farola redonda, o una vela son ejemplos de este tipo de luz.
- También proyecta sombras.

```
const pointLight = new THREE.PointLight(0xffffff, 1, 50);
//Color, intensidad, distancia de atenuación
pointLight.position.set(2, 5, 2);
scene.add(pointLight);
```

Iluminación Focal

- Proyecta rayos de luz dentro del volumen de un cono.
- Una linterna es un buen ejemplo de este tipo de iluminación.
- Es algo más compleja de crear, hay que saber qué características determinan este cono de luz.
- Por supuesto, también pueden generar sombras.
- A diferencia de otras luces, este tipo de luz sí tiene orientación. Es importante hacia donde mire. Esto se modifica alterando sus atributos.

```
const spotLight = new THREE.SpotLight(
    0xffffff, 1, 100, Math.PI / 6, 0.1, 2
);
// Color, intensidad, distancia, ángulo, penumbra, atenuación
spotLight.position.set(5, 10, 5);
scene.add(spotLight);
```

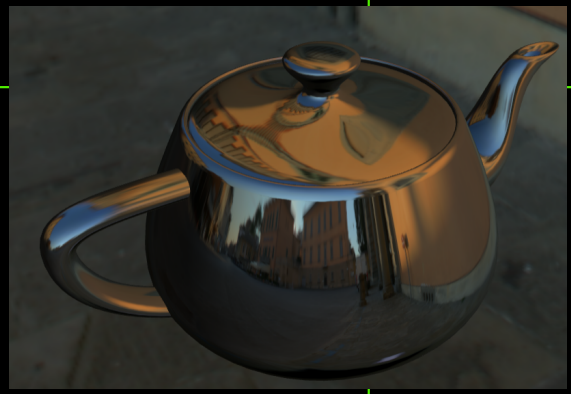
¿Qué materiales debo usar?

- Hay muchos tipos de materiales disponibles.
- Nos interesan tres en particular:
  - `MeshBasicMaterial`: para materiales simples y no complejos.
  - `MeshPhongMaterial`: Cuando queremos usar algún reflejo o que la luz "modifique" el color del modelo (aparentemente).
  - `MeshPhysicalMaterial`: Para objetos más complicados, donde es conveniente tener mucha interacción con la luz y realismo.

Ejemplos

→ Material básico

→ Material de phong



→ Material físico

Loop de animación

```
function animate() {
  requestAnimationFrame(animate);

  cube.rotation.x += 0.01;
  cube.rotation.y += 0.01;

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

- `requestAnimationFrame` sincroniza con el refresco.
- Normalmente esta función recibe la propia función desde donde se llama, como en el ejemplo.
- Se pueden modificar propiedades en cada actualización.

Eventos

```
// Aumentar o disminuir el FOV, viene en event.deltaY
window.addEventListener("wheel", (event) => {
    const zoomSpeed = 0.05;
    camera.fov += event.deltaY * zoomSpeed;
    camera.fov = Math.max(20, Math.min(100, camera.fov));
    camera.updateProjectionMatrix();
});
```

- Podemos añadir cualquier tipo de interacción y así modificar la cámara, luces, modelo, etc...
- Hacerlo a mano es un poco rollo, así que hay cosas ya hechas que nos ayudan

OrbitControls

```
import { OrbitControls } from "three/examples/jsm/controls/
    OrbitControls.js";

const controls = new OrbitControls(camera, renderer.domElement);
controls.enableDamping = true;
controls.dampingFactor = 0.05;
controls.zoomSpeed = 0.5;
controls.panSpeed = 1;

function animate(){
    // todo lo que habia antes ....
    controls.update();
}
```

- `OrbitControls` nos permite manejar de forma muy sencilla los controles básicos de rotación sobre el modelo.
- Permite manejar rotación y zoom con tan solo crear la clase, dar valores a sus atributos y ejecutar su método `update()`.

FlyControls

- Estos controles sirven para rotar la cámara con el teclado
- Similar a `OrbitControls`.

```
import { FlyControls } from 'three/examples/jsm/controls/
    FlyControls.js';

const flyControls = new FlyControls(camera, renderer.domElement)
    ;

function animate(){
    // añadimos despues de todo lo anterior...
    flyControls.update(0.05);
    renderer.render(scene, camera);
}
```

Grupos de mallas

```
const grupo = new THREE.Group();
grupo.add(malla1);
grupo.add(malla2);
grupo.position.set(5, 5, 0);
// malla1 y malla2 cambian su posicion (relativa al grupo)
```

- Los grupos nos permiten agrupar varias mallas en un solo objeto.
- Si modificamos su posición, rotación o animamos este objeto, todos los componentes del grupo se verán afectados.

Carga de modelos externos

- Podemos cargar también modelos externos de muchos formatos
- El más completo y el que voy a mostrar es el formato `.gltf`. Este formato se puede generar desde Blender, Maya, etc...

```
import { GLTFLoader } from 'three/examples/jsm/loaders/
    GLTFLoader.js';

const loader = new GLTFLoader();

// Carga el modelo
loader.load(
    'ruta/al/archivo.gltf',
    (gltf) => {
        scene.add(gltf.scene); // Agregar el modelo a la escena
        console.log("Completado!");
    },
    (xhr) => {
        console.log(`Carga: ${(xhr.loaded / xhr.total) * 100}%
            completado`); // barra de carga
    },
    (error) => {
        console.error('Error al cargar el modelo:', error);
    }
);
```

Exportación de Modelos

- Existen formas de exportar modelos a archivos externos.
- El que tendrá este taller es el formato `STL`.
- Es el formato que se usa para impresión 3D.

Exportación de Modelos

```
function exportToSTL() {
    const exporter = new STLExporter();
    const stlData = exporter.parse(scene); // Convierte la
        escena a STL
    const blob = new Blob([stlData], { type: 'application/octet-
        stream' });

    // Crear enlace de descarga
    const link = document.createElement('a');
    link.href = URL.createObjectURL(blob);
    link.download = 'modelo.stl';
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
}

// Agregar botón para exportar
const button = document.createElement('button');
button.innerText = 'Exportar a STL';
button.style.position = 'absolute';
button.style.top = '10px';
button.style.left = '10px';
button.addEventListener('click', exportToSTL);
document.body.appendChild(button);
```