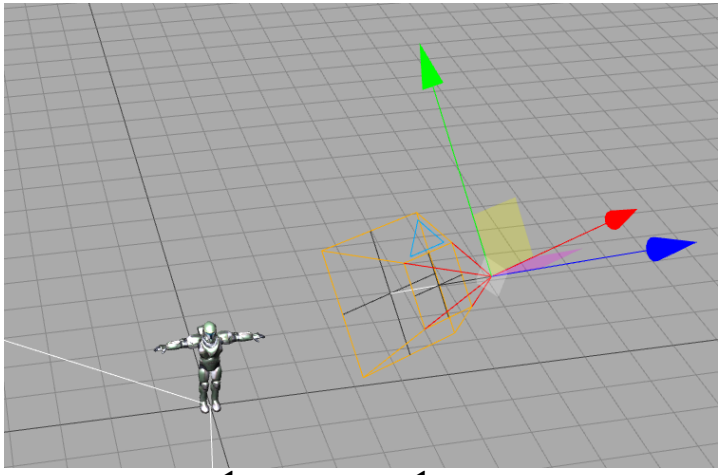
Sección de Informática Gráfica VALENCIA | Computer Graphics Group
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cámaras en Three.js

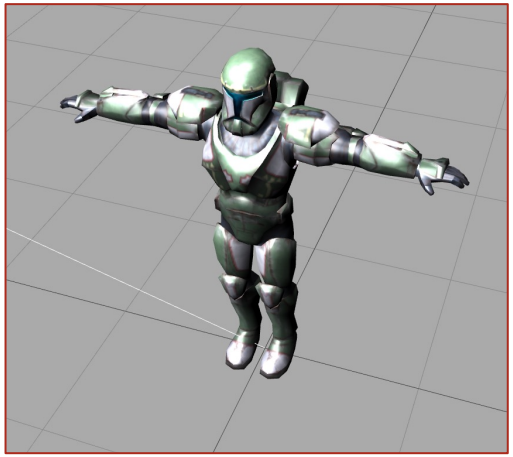
Gráficos 3D en la web

La clase Camera en Three.js

- Object3D
  - *.matrixWorld*: matriz de situación de la cámara en la escena
    - .position
    - .rotation
  - *.matrixAutoUpdate*: calcula la matrixWorld en cada frame
- Camera
  - *.matrixWorldInverse*: inversa de la matrixWorld. Es la matriz de la vista ***V***
  - *.projectionMatrix*: Matriz de proyección $P_{orto}M_{po}$
  - *.lookAt(poi)*: orienta la cámara con el vector *Look* apuntando al punto de interés ***poi***



$$\vec{d}^T p' = \vec{d}^T \frac{1}{p''_w} p'' = \vec{d}^T \frac{1}{p''_w} DP_{orto} M_{po} VMp$$



Cámaras en THREE

- Cámara ortográfica
- Cámara perspectiva
- Matriz de la vista
  - position
  - up
  - lookAt()
- Matriz de proyección
  - límites de frustum
    - Ortográfica: left,right,bottom,top,near,far
    - Perspectiva: fov, aspectRatio, near, far
- Matriz del dispositivo
  - límites del viewport
    - left, bottom, width, height

```
camera = new THREE.PerspectiveCamera( fov, canvasWidth/ canvasHeight, near, far );
camera.position.set( eye.x, eye.y, eye.z );
camera.lookAt( center );
```

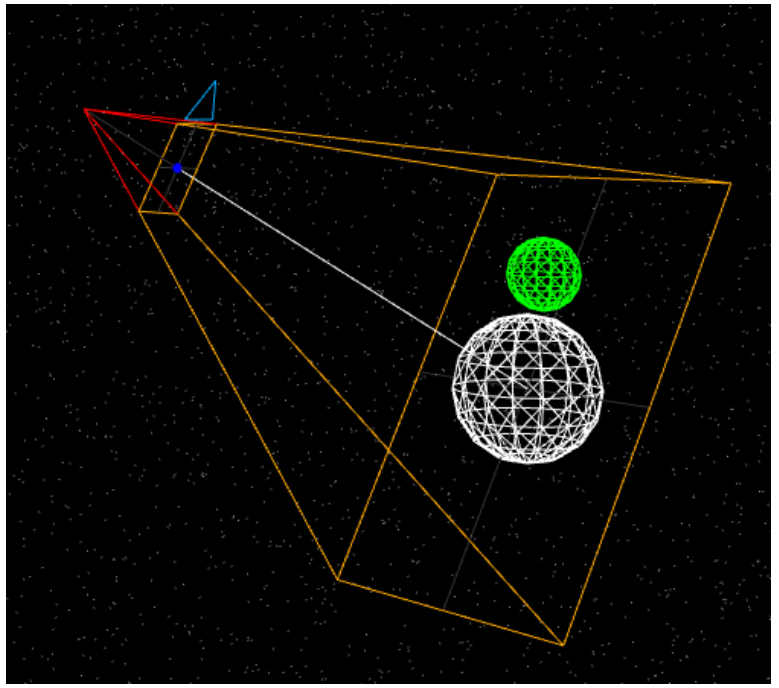
```
camera = new THREE.OrthographicCamera( xleft, xright, ybottom, ytop, znear, zfar );
camera.position.set( eye.x, eye.y, eye.z );
camera.lookAt( center );
```

```
camera.fov = newfov;
camera.updateProjectionMatrix();
renderer.render(scene, camera);
```

```
renderer.setViewport( 0, 0, canvasWidth, canvasHeight );
renderer.render( scene, camera );
```

Ayudantes para el frustum

- CameraHelper(camera)
  - Es un objeto que dibuja el frustum, la línea principal de visión y la vertical subjetiva
  - Como todos los objetos puede ser visible o no



```
const camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
window.innerHeight, 0.1, 1000 );
const helper = new THREE.CameraHelper( camera );
scene.add( helper );
```

Relación de aspecto

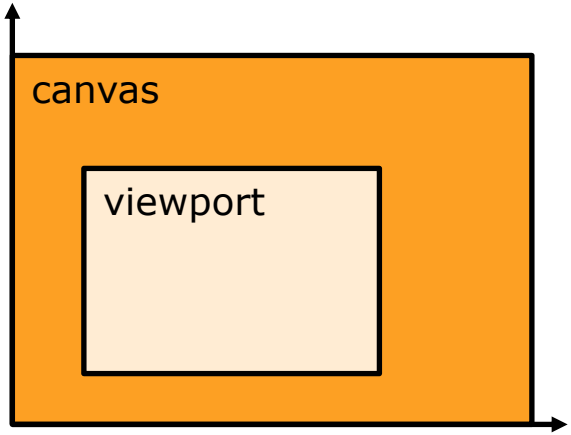
- Para que no haya distorsión deben mantenerse las relaciones de aspecto entre el área de dibujo y el plano de proyección de la cámara
- El área de dibujo posible es el *canvas* del *renderer*

```
renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
```

- Dentro del *canvas* se fija el marco

```
renderer.setViewport( xorigen, yorigen, ancho, alto );
```

  - El motor dibuja dentro del marco
  - El marco debe conservar la relación de aspecto de la cámara
  - Por defecto el marco es la totalidad del *canvas*



Relación de aspecto

- Al redimensionar el usuario el tamaño del documento, se puede optar:
  - Mantener el *canvas* fijo
  - Redimensionar el *canvas* al nuevo tamaño del documento
    - Actualizamos el canvas/viewport
    - Actualizamos la relación de aspecto de la cámara
    - Forzamos el recalculo de la matriz de la proyección

```
// Atencion al evento de resize del documento
window.addEventListener('resize', updateAspectRatio );
```

```
function updateAspectRatio()
{
    // Renueva la relacion de aspecto por cambio del documento
    renderer.setSize(window.innerWidth, window.innerHeight);
    camera.aspect = window.innerWidth/window.innerHeight;
    // Hay que actualizar la projection
    camera.updateProjectionMatrix();
}
```

Movimiento de la cámara

- Utilidad OrbitControls.js
  - Manejo de cámara con el ratón: orbitar, zoom, panning
  - Mirar propiedades en el código de OrbitControls: p.e. limitar el zoom, liberar teclado, ...

```
import * as THREE from "../lib/three.module.js"
import {OrbitControls} from "../lib/OrbitControls.module.js"
//...
// Controlador de camara
let cameraControls;
//...
const ar = window.innerWidth / window.innerHeight;
camera = new THREE.PerspectiveCamera( 40, ar, 0.1, 100 );
camera.position.set( 2,2,10 );
cameraControls = new OrbitControls( camera,renderer.domElement );
cameraControls.target.set( 0,2,0 );
camera.lookAt( 0,2,0, );
//...
```

Movimiento de cámara

Algunas propiedades de OrbitControls

```
// How far you can dolly in and out ( PerspectiveCamera only )
this.minDistance = 0;
this.maxDistance = Infinity;

// How far you can zoom in and out ( OrthographicCamera only )
this.minZoom = 0;
this.maxZoom = Infinity;

// How far you can orbit vertically, upper and lower limits.
// Range is 0 to Math.PI radians.
this.minPolarAngle = 0; // radians
this.maxPolarAngle = Math.PI; // radians

// How far you can orbit horizontally, upper and lower limits.
// If set, the interval [ min, max ] must be a sub-interval of [ - 2 PI, 2 PI ], with ( max
this.minAzimuthAngle = - Infinity; // radians
this.maxAzimuthAngle = Infinity; // radians

// Set to true to enable damping (inertia)
// If damping is enabled, you must call controls.update() in your animation loop
this.enableDamping = false;
this.dampingFactor = 0.05;

// This option actually enables dollying in and out; left as "zoom" for backwards compatibi
// Set to false to disable zooming
this.enableZoom = true;
this.zoomSpeed = 1.0;

// Set to false to disable rotating
this.enableRotate = true;
this.rotateSpeed = 1.0;
```

Niebla

- Es posible añadir niebla a la escena dependiendo de la distancia a la cámara
  - Lineal

Constructor

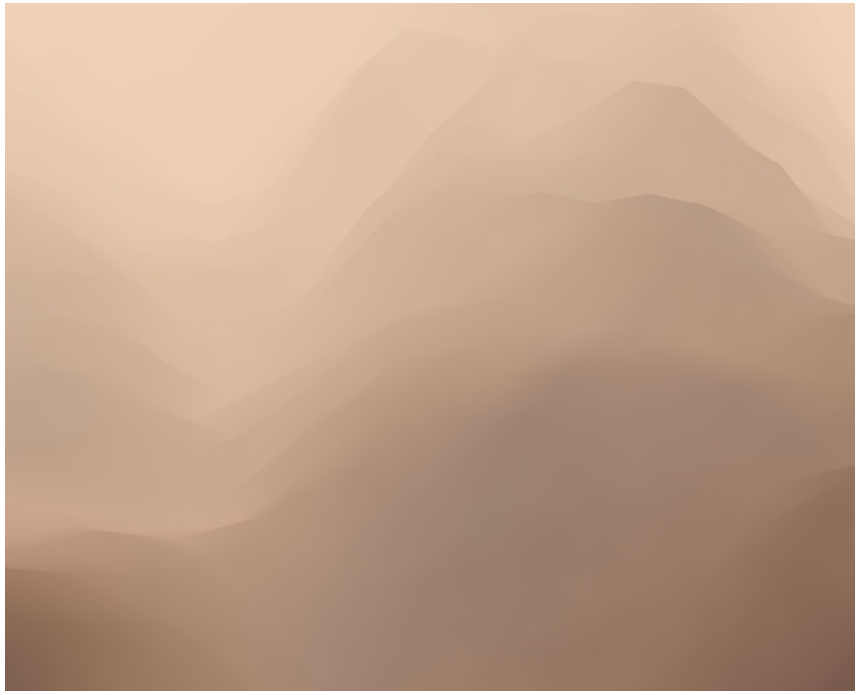
```
Fog( hex, near, far )
```

  - Exponencial

Constructor

```
FogExp2( hex, density)
```

color de la niebla



```
scene.fog = new THREE.Fog( 0xefd1b5, 1, 1000 );
```

Selección (picking)

- Descubrir qué objeto se encuentra más cerca de la cámara en una visual
- Trazado del rayo
  - rayo
    - origen
    - dirección
  - objetos
    - método de intersección
  - intersección
    - punto, normal, cara, objeto, ...
- Clase *RayCaster(origen,dirección)*
  - .setFromCamera(cursor, cámara)
  - .intersectObjects(objetos, *deep*)
- Usar *.isXXX* para selección de objetos

normalizado

```
var raycaster = new THREE.Raycaster();
var mouse = new THREE.Vector2();

function onMouseMove( event ) {

        // calculate mouse position in normalized device coordinates
        // (-1 to +1) for both components

        mouse.x = ( event.clientX / window.innerWidth ) * 2 - 1;
        mouse.y = - ( event.clientY / window.innerHeight ) * 2 + 1;

}

function render() {

        // update the picking ray with the camera and mouse position
        raycaster.setFromCamera( mouse, camera );

        // calculate objects intersecting the picking ray
        var intersects = raycaster.intersectObjects( scene.children );

        for ( var i = 0; i < intersects.length; i++ ) {

                intersects[ i ].object.material.color.set( 0xff0000 );

        }

        renderer.render( scene, camera );

}

window.addEventListener( 'mousemove', onMouseMove, false );

window.requestAnimationFrame(render);
```