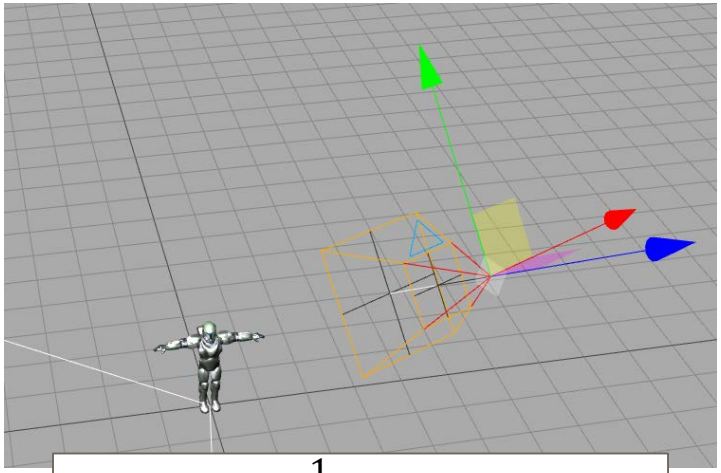
Sección de Informática Gráfica VALENCIA | Computer Graphics Group
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

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
  - *.projectionMatrix*: Matriz de proyección
  - *.lookAt*(Vector3 *poi*): orienta la cámara con el vector Look apuntando a *poi*



$$\vec{d}^T \mathbf{p}' = \vec{d}^T \frac{1}{p''_w} \mathbf{p}'' = \vec{d}^T \mathbf{PVM}\, \mathbf{p}$$

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 );
```

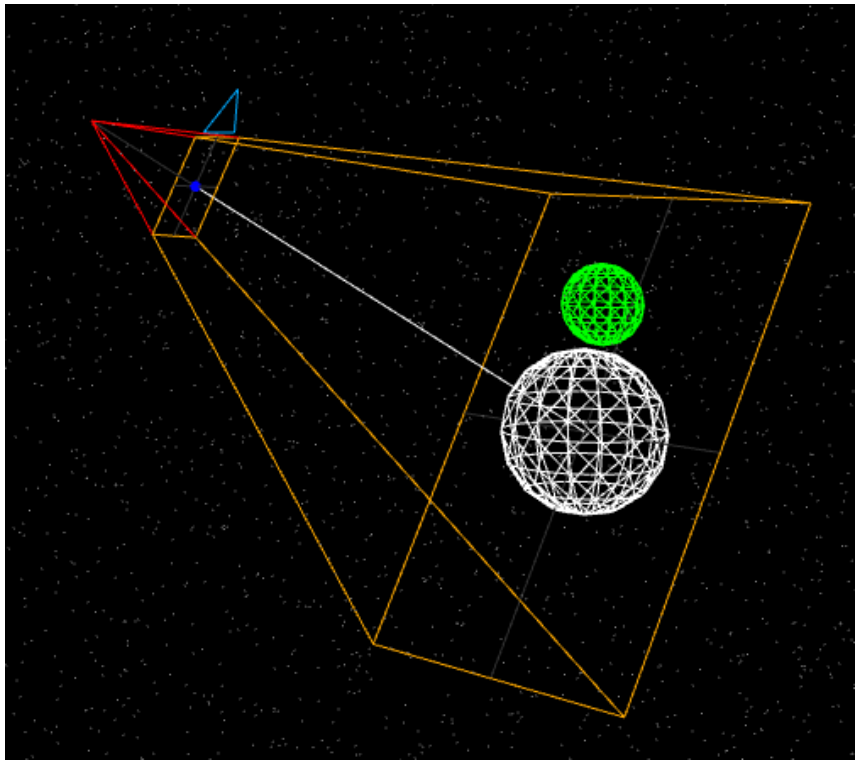
```
camera = new THREE.OrthographicCamera( xleft, xright, ybottom, ytop, znear, zfar );
camera.position.set( eye.x, eye.y, eye.z );
camera.lookAt( center );
```

```
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



Relación de aspecto

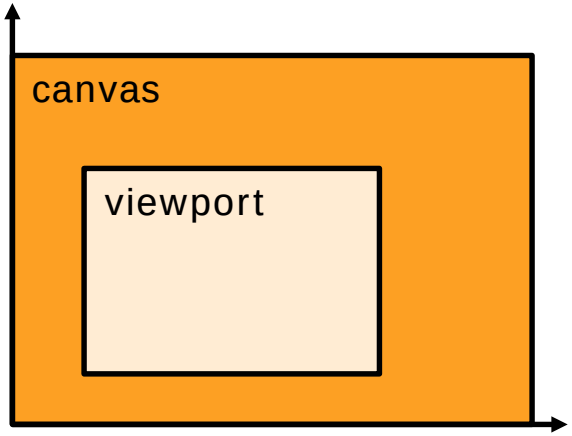
- Para que no haya distorsión deben mantenerse las relaciones de aspecto entre el área de dibujo y el plano de proyección de la cámara
- El área de dibujo posible es el *canvas* del *renderer*

```
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

```
67  function render()
68  {
69      requestAnimationFrame( render );
70      update();
71      cameraControls.update();
72      renderer.render( scene, camera );
73  }
```

```
 7  function init()
 8  {
 9      renderer = new THREE.WebGLRenderer();
10      renderer.setSize( canvas_width, canvas_height );
11      renderer.setClearColor( new THREE.Color(0xFFFFFF), 1.0 );
12      document.body.appendChild( renderer.domElement );
13
14      scene = new THREE.Scene();
15
16      var aspectRatio = canvas_width / canvas_height;
17      camera = new THREE.PerspectiveCamera( 50, aspectRatio , 0.1, 100 );
18      camera.position.set( 2, 1, 5 );
19
20      cameraControls = new THREE.OrbitAndPanControls( camera, renderer.domElement );
21      cameraControls.target.set( 0, 0, 0 );
22  }
```

http://personales.upv.es/rvivo/webgl/3jsParaguas.html

Movimiento de cámara

propiedades de OrbitControls

```
// Set to false to disable this control
this.enabled = true;
// "target" sets the location of focus, where the control orbits around
// and where it pans with respect to.
this.target = new THREE.Vector3();
// center is old, deprecated; use "target" instead
this.center = this.target;
// This option actually enables dollying in and out; left as "zoom" for
// backwards compatibility
this.noZoom = false;
this.zoomSpeed = 1.0;
// Limits to how far you can dolly in and out
this.minDistance = 0;
this.maxDistance = Infinity;
// Set to true to disable this control
this.noRotate = false;
this.rotateSpeed = 1.0;
// Set to true to disable this control
this.noPan = false;
this.keyPanSpeed = 7.0; // pixels moved per arrow key push
// Set to true to automatically rotate around the target
this.autoRotate = false;
this.autoRotateSpeed = 2.0; // 30 seconds per round when fps is 60
// 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
// Set to true to disable use of the keys
this.noKeys = false;
// The four arrow keys
this.keys = { LEFT: 37, UP: 38, RIGHT: 39, BOTTOM: 40 };
```

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)

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);
```