

Visualización: Vistas y Luces

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática
Curso 2021-2022

Contenidos

1 Vistas

2 Luces

Objetivos

- Saber crear y modificar vistas de la escena
- Saber crear fuentes de luz

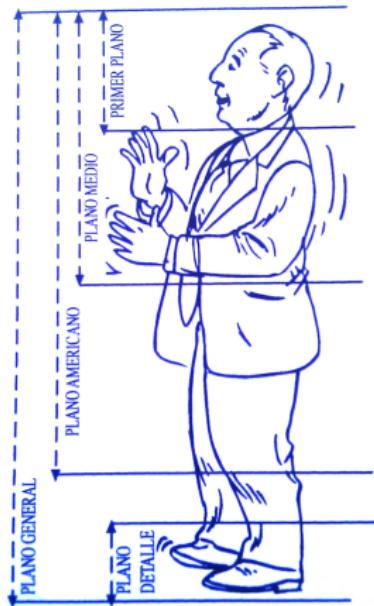
Vistas

La cámara

- La cámara ayuda a atraer la atención del espectador
- Puede ser un personaje más
- Se puede animar, cambiar su posición, orientación, etc.
- La animación de la cámara debe realizarse con precaución

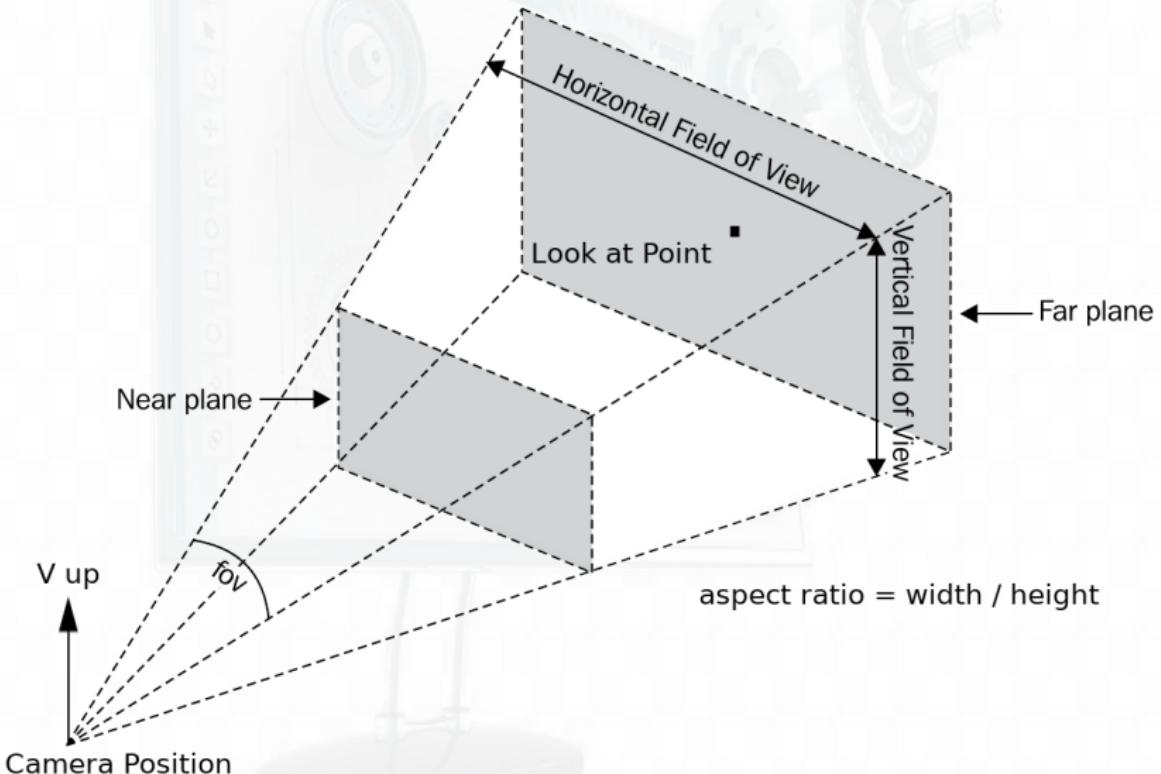


Planos de encuadre

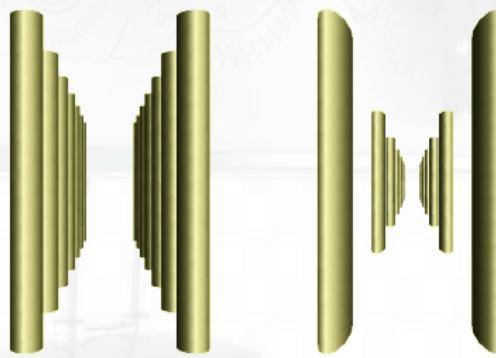
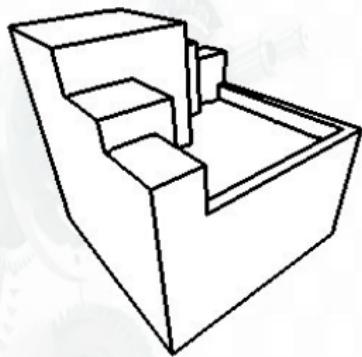
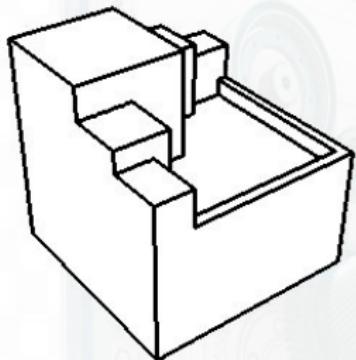


Parámetros de una cámara

Vista cónica



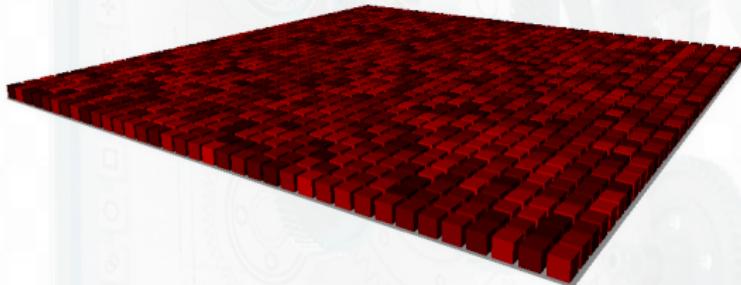
Parámetros de una cámara



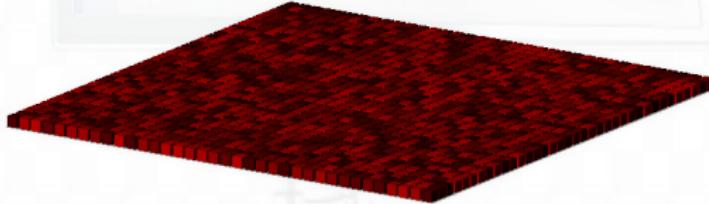
Vistas

Three.js

- Se pueden definir vistas
 - ▶ En perspectiva, con `THREE.PerspectiveCamera`



- ▶ Ortográficas, con `THREE.OrthographicCamera`



Vistas

Creación

- Se crean con:

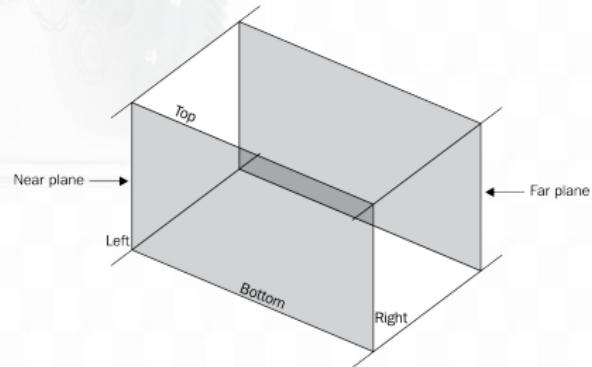
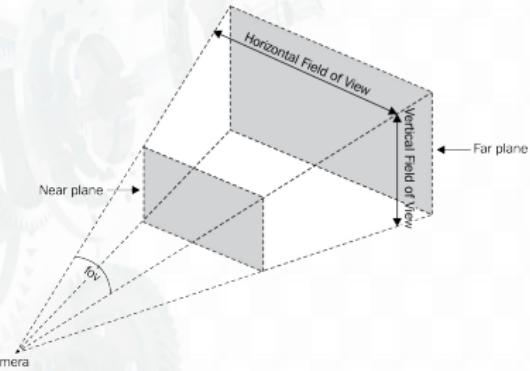
- ▶ `THREE.PerspectiveCamera`
(`fov`, `aspect`, `near`, `far`)
- ▶ `THREE.OrthographicCamera`
(`left`, `right`, `top`,
`bottom`, `near`, `far`)

Hacer cambios a posteriori requiere
`camara.updateProjectionMatrix();`

- Atributos

- ▶ `position`
- ▶ `lookAt`
- ▶ `up`

Three.js



Actualización de vistas

- Las vistas pueden requerir ser actualizadas cuando:
 - ▶ El usuario modifica el tamaño de la ventana de la aplicación
 - ▶ Se cambia la orientación de la cámara
 - ★ Por ejemplo, si se quiere captar a un personaje que se mueve estando la cámara fija en la escena



- ▶ Se cambia algún parámetro que define una vista
 - ★ Habría que actualizar la matriz de proyección

Actualización de vistas

Ejemplos

Actualización de Vistas: Cambio de tamaño de la ventana

```
// En el main
window.addEventListener ("resize", () => scene.onWindowResize());
```

// El método onWindowResize sería

```
onWindowResize () {
    var camara = this.camara;
    var nuevaRatio = window.innerWidth / window.innerHeight;
    if (camara.isOrthographicCamera) {
        var altoVista = camara.top - camara.bottom;
        var centroAncho = (camara.left + camara.right)/2;
        camara.left = centroAncho - altoVista*nuevaRatio/2;
        camara.right = centroAncho + altoVista*nuevaRatio/2;
    } else { // con las vistas en perspectiva es más fácil
        camara.aspect = nuevaRatio;
    }
    // no olvidarse de actualizar la matriz de proyección
    camara.updateProjectionMatrix();
    // también hay que actualizar el renderer
    this.renderer.setSize (window.innerWidth , window.innerHeight);
}
```

Actualización de vistas

Ejemplos

Actualización de Vistas: Cambio de orientación

```
// cámara siempre quiere captar a personaje  
// si se mueve el personaje  
  
// this.nuevoTarget es un Object3D que se tiene para esto  
  
// se obtiene una copia de la posición del personaje  
// está en coordenadas locales  
this.nuevoTarget.set (personaje.position.x, personaje.position.y, personaje.position.z);  
  
// se transforma a coordenadas del mundo  
personaje.getWorldPosition (this.nuevoTarget);  
  
// se actualiza el lookAt de la cámara  
camara.lookAt (this.nuevoTarget);  
  
// Ojo:  
Si el personaje no se ha movido pero la cámara sí,  
solo habría que volver a ejecutar lookAt
```

Cámara subjetiva

- La cámara se *cuelga* de la figura con la que es solidaria
- La orientación se da de manera relativa
 - ▶ En THREE requiere un pequeño cálculo porque el método lookAt requiere una posición absoluta en coordenadas del mundo

Ejemplo: Cámara subjetiva

```
var camara = new THREE.PerspectiveCamera(...);  
casco.add(camara);  
// Posición de la cámara con relación al casco  
camara.position.set(0,15,0);  
// Posición del punto de mira con relación a la cámara  
var target = new THREE.Vector3(10,-5,0);  
// Se transforma el target a coordenadas del mundo teniendo en cuenta la matriz de  
// transformación global de la cámara  
camara.getWorldPosition(target);  
// Ya se puede usar el target (en coordenadas del mundo) para ajustar el lookAt de la cámara  
camara.lookAt(target);
```

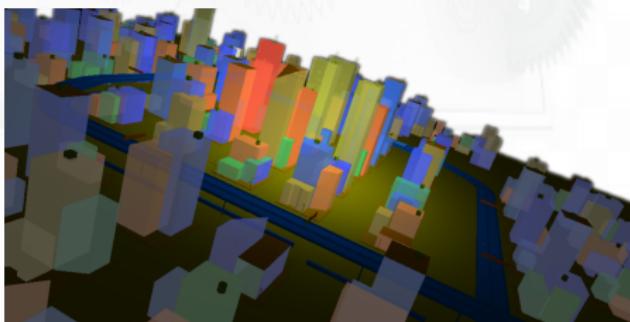


Controlador de movimientos de cámara

Simulando controles de vuelo

- Requiere la biblioteca `FlyControls.js`
- `import { FlyControls } from '../libs/FlyControls.js'`

Control	Movimiento
Botón izquierdo y central	Avanzar
Botón derecho del ratón	Retroceder
Movimiento del ratón	Giros hacia el puntero
Tecla Q	Inclinarse a la izquierda
Tecla E	Inclinarse a la derecha

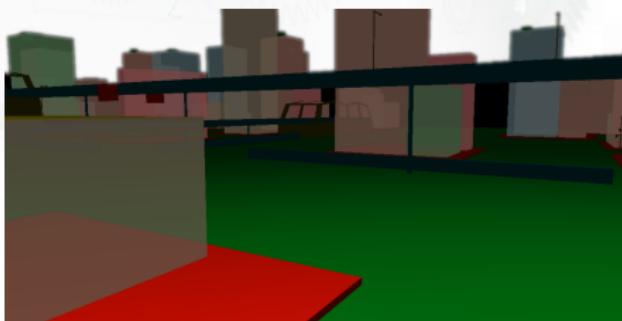


Controlador de movimientos de cámara

Primera persona

- Requiere la biblioteca `FirstPersonControls.js`
- `import { FirstPersonControls } from '../libs/FirstPersonControls.js'`

Control	Movimiento
Movimiento del ratón	Mirar alrededor
Cursores	Moverse en esa dirección
Tecla R	Subir
Tecla F	Bajar
Tecla Q	Detener el movimiento

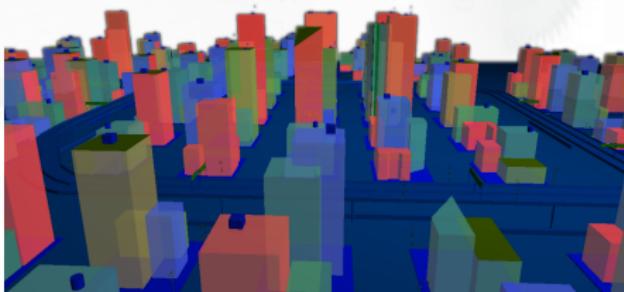


Controlador de movimientos de cámara

Órbita

- Requiere la biblioteca `TrackballControls.js`
- `import { TrackballControls } from '../libs/TrackballControls.js'`

Control	Movimiento
Arrastre con botón izquierdo	Giros respecto al centro de la escena
Arrastre con botón derecho	Reencuadre
Rueda del ratón	Zoom



Necesidad de usar de un objeto THREE.Clock

- Los controladores FlyControls y FirstPersonControls necesitan de un objeto THREE.Clock

Controladores de cámara: Uso del objeto THREE.Clock

```
// En el constructor  
this.clock = new THREE.Clock();  
  
// En el update  
var delta = this.clock.getDelta();  
this.controlCamera.update(delta);
```

Ejemplo

Ejemplo: Creación y control de una cámara (vuelo)

```
// No olvidarse el import
import { FlyControls } from '../libs/FlyControls.js'

// Al crear la cámara en el constructor
this.clock = new THREE.Clock();
this.camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight,
    0.1, 1000);
this.camera.position.set(100,100,100);
this.camera.lookAt(0, 0, 0);

// Se crea el control de vuelo
this.flyControls = new FlyControls(this.camera, this.renderer.domElement);
this.flyControls.movementSpeed = 25;
this.flyControls.rollSpeed = Math.PI / 24;
this.flyControls.autoForward = false;

// En el método update
var delta = this.clock.getDelta();
this.flyControls.update(delta);
```

Varias vistas en varios viewports

- Se pueden visualizar varios viewports mostrando una cámara distinta en cada viewport



Vistas: Varias cámaras en varios viewports

```
// Se define un método para agrupar las instrucciones necesarias
// Los parámetros left, top, width, height toman valores entre 0 y 1
renderViewport (escena, camara, left, top, width, height) {
    var l = left * window.innerWidth;      var t = top * window.innerHeight;
    var w = width * window.innerWidth;     var h = height * window.innerHeight;
    this.renderer.setViewport (l,t,w,h);    this.renderer.setScissor (l,t,w,h);
    this.renderer.setScissorTest (true);
    camara.aspect = w/h; // se considera que es en perspectiva
    camara.updateProjectionMatrix ();
    this.renderer.render (escena, camara);
}

// En el método update de la clase principal se llama al método anterior
// para todas los viewports que se deseen actualizar
this.renderViewport (escena, camara1, 0, 0, 1, 1);
this.renderViewport (escena, camara2, 0.5, 0.5, 0.5, 0.5);
```

Luces

- Imprescindibles para que se “vean las cosas”
- También usadas para crear “ambiente”
- Una fuente de iluminación artificial está compuesta de:
 - ▶ **Lámpara:** Fuente (bombilla) que emite la energía luminosa, caracterizada por el color e intensidad de la luz
 - ▶ **Luminaria:**
Aparato que aloja a la lámpara y que distribuye el flujo luminoso.

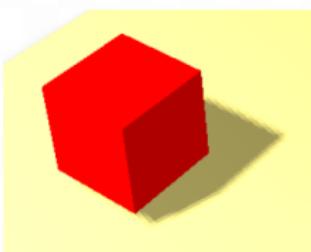
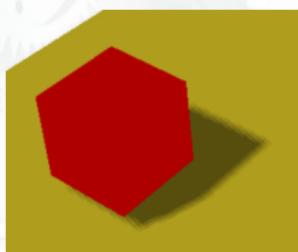
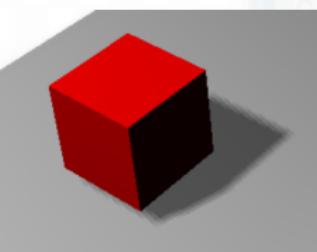


Fuentes de iluminación en

Three.js

Luz ambiental: AmbientLight

- La luz ambiental ilumina toda la escena por igual
- Se usa para simular la luz indirecta
- Permite añadir fácilmente una dominante de color
- Constructor: `AmbientLight (color, intensity)`



Ejemplo: Definición de la luz ambiental

```
var ambientLight = new THREE.AmbientLight (0x0c0c0c, 0.5);  
scene.add (ambientLight);
```

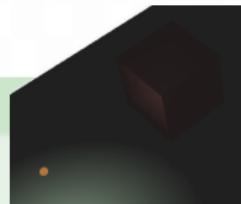
Luz puntual

PointLight

- Desde una determinada posición, ilumina en todas direcciones
- Constructor:
`PointLight (color, intensity, distance, decay)`
 - ▶ `intensity`, por defecto es 1
 - ▶ `distance`, la intensidad es cero a partir de esa distancia
si `distance == 0`, valor por defecto, no se toma en cuenta
 - ▶ `decay`, la caída: 1 (por defecto), lineal; 2, cuadrática
- Atributos:
 - ▶ `position`, sitúa la luz en dicha posición
 - ▶ `visible`, un booleano que enciende/apaga la luz

Ejemplo: Definición de la luz puntual

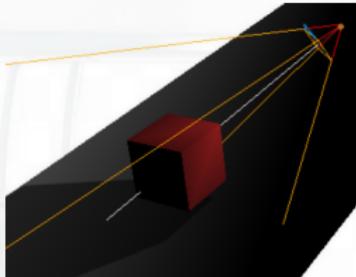
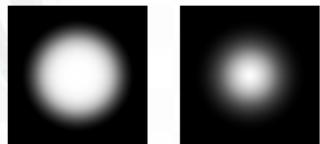
```
var pointLight = new THREE.PointLight (0xfcfcfc, 1, 50);  
pointLight.position.set (2, 3, 5);
```



Luz focal

SpotLight

- Desde una posición, ilumina en una dirección y ángulo
- Constructor:
`SpotLight (color, intensity, distance,
angle, penumbra, decay)`
 - ▶ angle, desde su dirección, por defecto es $\pi/3$ (máximo $\pi/2$)
 - ▶ penumbra, cómo decrece la intensidad desde el centro.
Toma valores entre 0 (por defecto) y 1.
- Atributos:
 - ▶ target, el objeto a donde apunta la luz (debe estar en la escena)



SpotLight

Ejemplo: Definición de la luz puntual

```
// Se crea con un color y una intensidad
var spotLight = new THREE.SpotLight (0xfcfcfc, 0.7);

// Se ajusta su posición y el punto a donde se orienta
spotLight.position.set (2, 30, 5);

// Se podría "apuntar" a un Object3D existente
// (o cualquiera de sus clases derivadas)
// O como en este caso, se crea un objeto sin geometría
// para darle la posición a la que se desea "apuntar"
var target = new THREE.Object3D ();
target.position.set (2, 0, 5);
spotLight.target = target;

// El target debe estar directa o indirectamente en la escena
scene.add (target);

// Por último se añade la luz a la escena
scene.add (spotLight);
```

Luz direccional

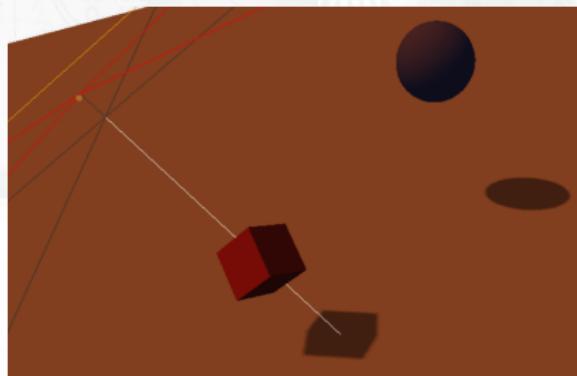
DirectionalLight

- Emite rayos paralelos entre sí según una dirección

- Constructor:

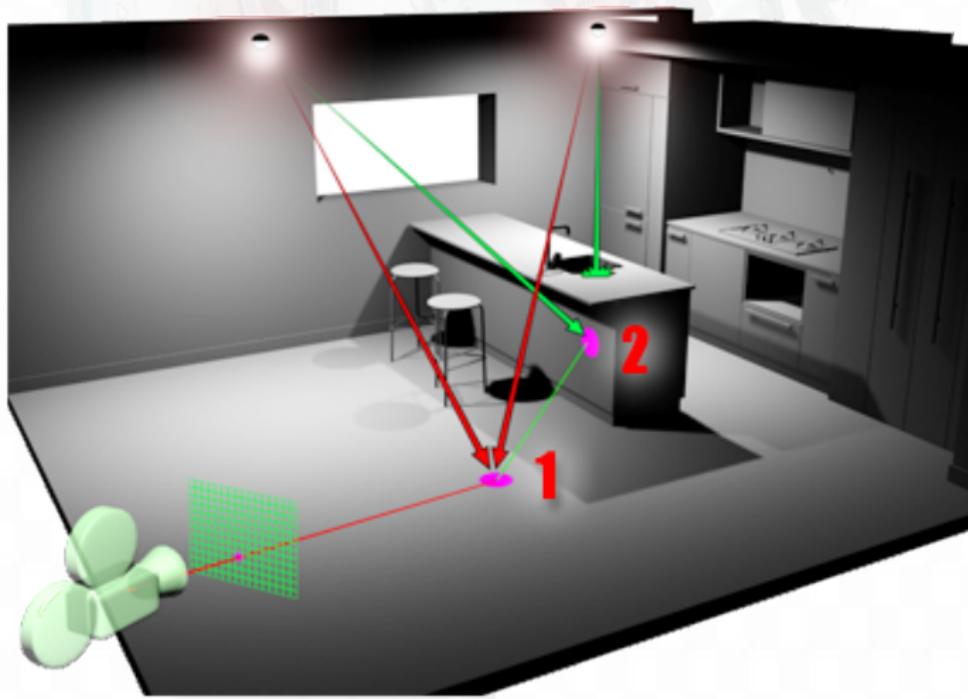
`DirectionalLight (color, intensity)`

- ▶ La dirección se establece con los atributos `position` y `target`
- ▶ Si no hay `target` los rayos se dirigen al $(0, 0, 0)$



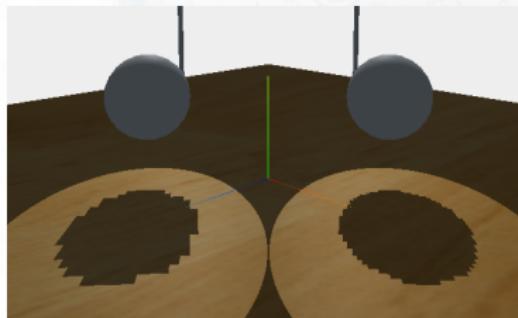
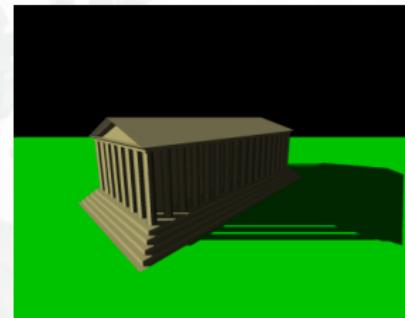
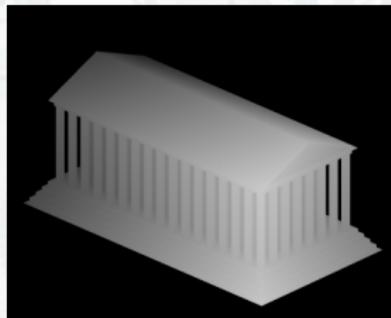
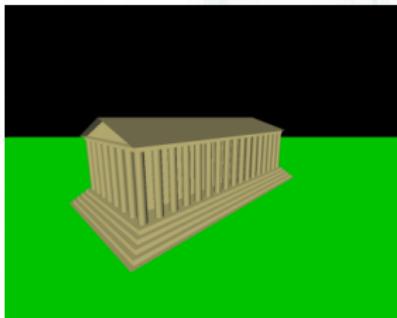
Sombras arrojadas

- Cálculo mediante Ray-Tracing



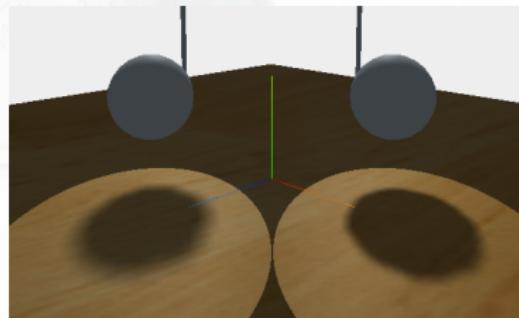
Sombras arrojadas

- Cálculo mediante Shadow Mapping



Mapa de: 32x32

64x64



Bordes suavizados

Sombras arrojadas en

Three.js

- Implementa Shadow Mapping
- Elementos a configurar:
 - ▶ El renderer para que las calcule
 - ▶ La luz para que las proyecte
 - ▶ Los objetos que las proyectan
 - ▶ Los objetos que las reciben

Ejemplo: Configuración del Renderer para proyectar sombras

Se habilita el cálculo

```
renderer.shadowMap.enabled = true;
```

Se indica método de cálculo

```
renderer.shadowMap.type = THREE.BasicShadowMap; // sombras pixeladas
```

Se puede usar

- THREE.PCFShadowMap (valor por defecto)
- THREE.PCFSOFTShadowMap (más calidad)

Sombras arrojadas en

Three.js

Ejemplo: Configuración de las **Luces** para proyectar sombras

```
// Para cualquier tipo de luz
```

```
light.castShadow = true;  
light.shadow.mapSize.width = 512;  
light.shadow.mapSize.height = 512;  
light.shadow.camera.near = 0.5;  
light.shadow.camera.far = 500;
```

esta luz arroja sombras
la resolución del mapa
valores por defecto
distancia mínima y máxima
con sombra

```
// Si la luz es direccional, además
```

```
light.shadow.camera.left = -5;  
light.shadow.camera.bottom = -5;  
light.shadow.camera.right = 5;  
light.shadow.camera.top = 5;
```

valores por defecto
fuera de estos límites
NO hay sombras
puede que haya que ampliarlos

Sombras arrojadas en

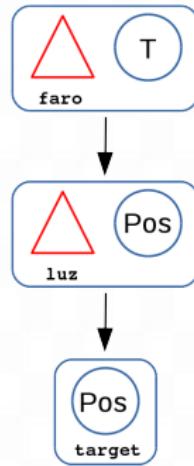
Three.js

Ejemplo: Configuración de las **Figuras** para proyectar sombras

```
// Para modificar un nodo concreto  
  
figura.castShadow = true;      esta figura arroja sombras en otros  
figura.receiveShadow = true;   esta figura recibe sombras de otros  
  
// Para modificar un nodo y todos sus hijos  
  
figura.traverseVisible ((unNodo) => {  
    unNodo.castShadow = true;  
    unNodo.receiveShadow = true;  
});
```

Luces subjetivas

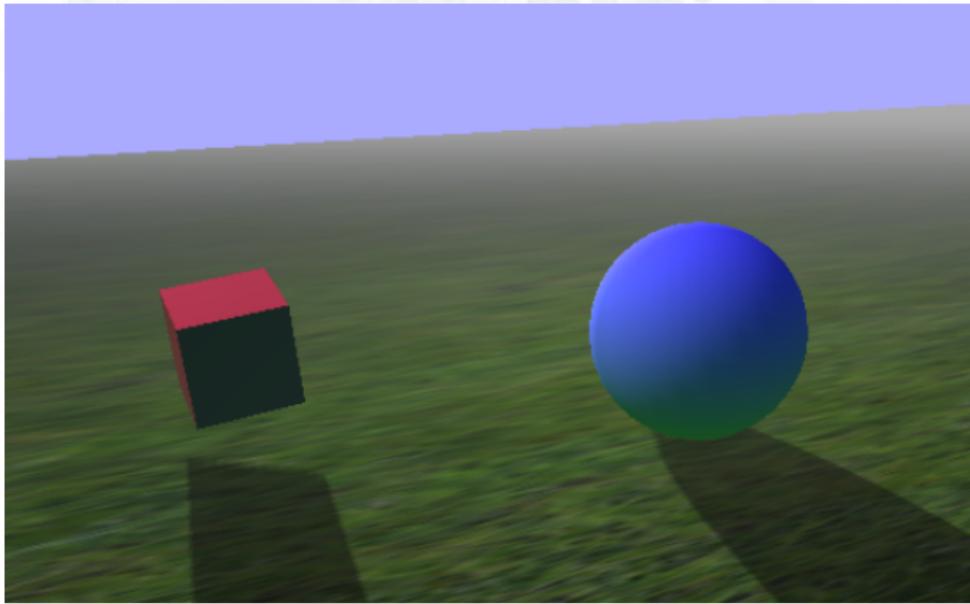
- Con respecto a la fuente de luz
 - ▶ Debe *colgar* del elemento con el que es solidaria
 - ▶ Su posición se indica relativamente a la posición de este elemento
- Con respecto al target, si lo hay, se considera lo mismo pero *colgándolo* de la propia fuente de luz



Luz natural en exteriores

HemisphereLight

- Se trata de una efecto que incorpora THREE
- Simula la luz difusa del cielo y la que pueda reflejar el suelo



HemisphereLight

- Constructor:

```
HemisphereLight ( sKyColor,groundColor,intensity)
```

- Además hay que poner la luz direccional que represente al sol

Ejemplo: Uso de luz del cielo y reflejo del suelo

```
// Suelo: Un plano con textura que recibe sombras
var loader = new THREE.TextureLoader();
var textureGrass = loader.load ("grass.jpg");
var planeMat = new THREE.MeshLambertMaterial ({map: textureGrass});

var plane = new THREE.Mesh (planeGeometry, planeMat);
plane.receiveShadow = true;
// Sol: Una luz direccional
var sun = new THREE.DirectionalLight (0xffffff);

// Luz del cielo y reflejos del suelo
var skyGroundLight = new THREE.HemisphereLight (0xaaaaaf, 0xaaffaa);
```

Dibujado del disco del sol

Lensflare

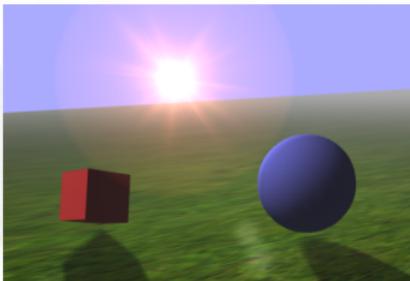
- Se puede simular el disco del sol (u otras luces) mediante texturas
- O añadir el efecto de reflejo de esa luz en la lente

- Imports:

```
import { Lensflare, LensflareElement }  
from '../libs/Lensflare.js'
```

- Uso:

- ▶ Se instancia la clase Lensflare
- ▶ Se le añaden los LensflareElement que sean necesarios, para el disco de la luz y los reflejos que se quieran mostrar
- ▶ Se cuelga el Lensflare de la luz que corresponda



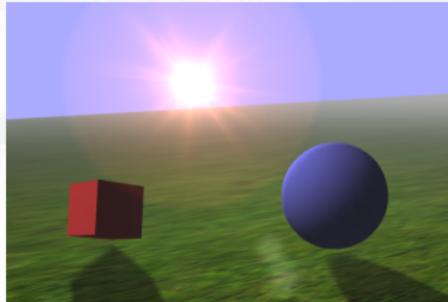
Dibujado del disco del sol

LensflareElement

- Constructor:

`LensflareElement (texture, size, distance, color)`

- ▶ `texture`, la textura a usar
- ▶ `size`, el tamaño en píxeles, -1 para tomar el tamaño de la imagen
- ▶ `distance`, la distancia entre la fuente de luz (0) y la cámara (1)
- ▶ `color`



LensFlare

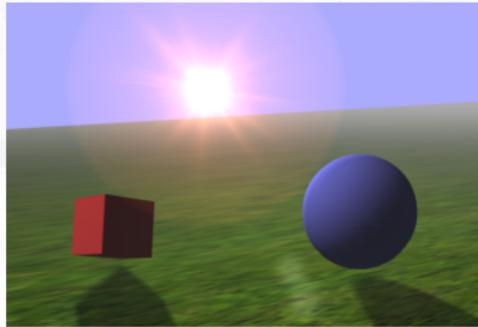
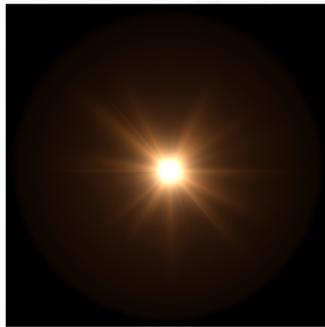
Ejemplo: Disco del sol y reflejo en la lente

```
var textureSun = loader.load ("lensflareSun.png");
var textureFlare = loader.load ("lensflare.png");

var flareColor = new THREE.Color (0xffaaee);
var lensflare = new Lensflare();

lensflare.addElement (new LensflareElement (textureSun, 350, 0.0, flareColor));
lensflare.addElement (new LensflareElement (textureFlare, 60, 0.6));
lensflare.addElement (new LensflareElement (textureFlare, 70, 0.7));

directionalLight.add (lensflare);
```



Visualización: Vistas y Luces

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática
Curso 2021-2022