

Visualización: Vistas y Luces

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

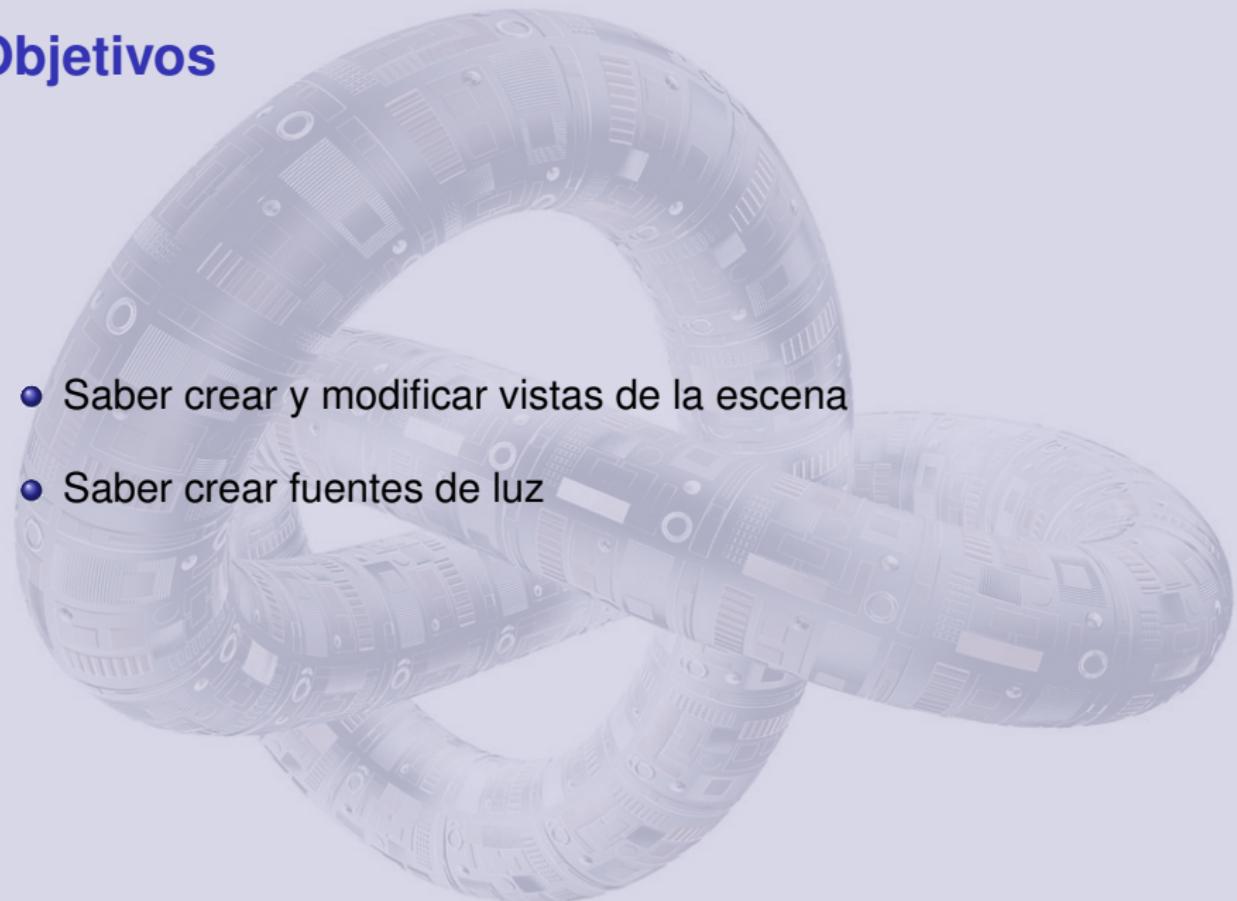
Grado en Ingeniería Informática
Curso 2023-2024

Contenidos

- 
- 1 Vistas
 - 2 Luces

Objetivos

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



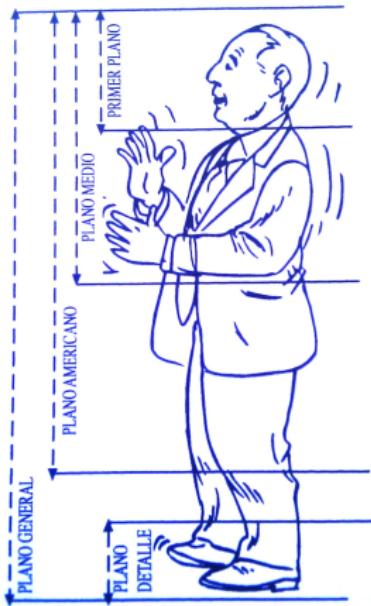
Vistas

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

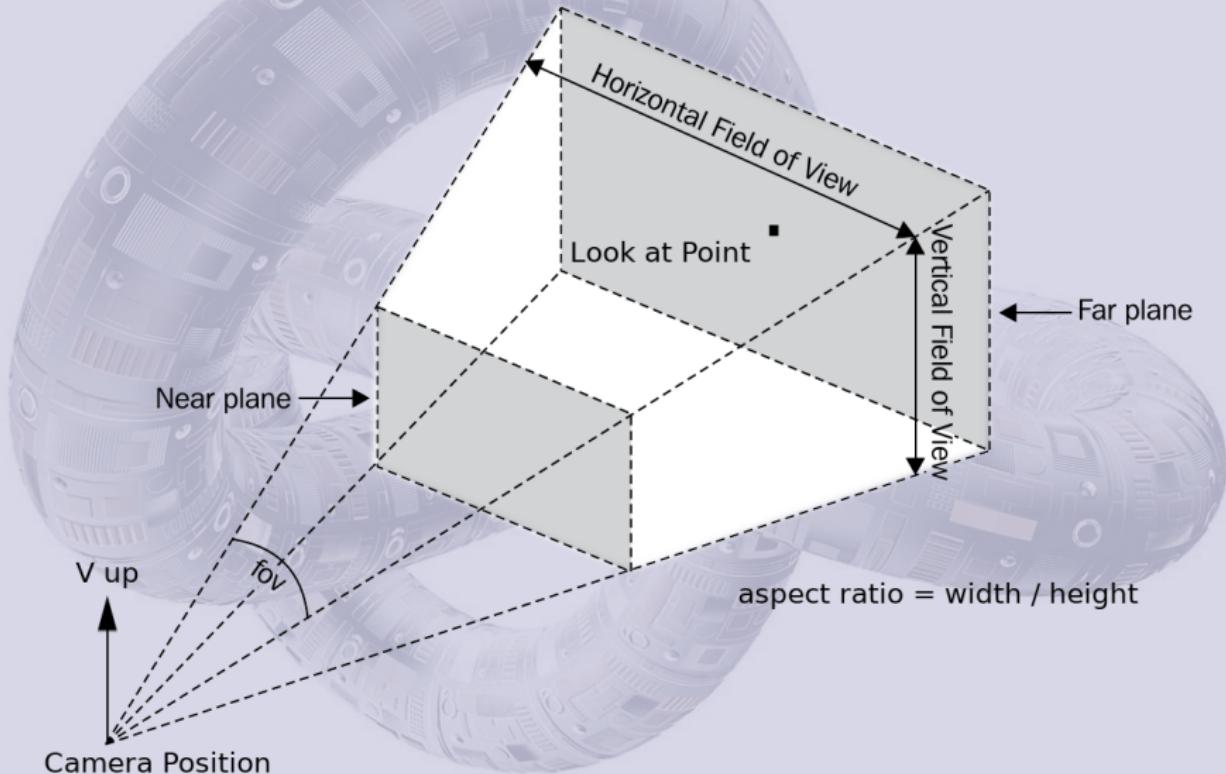


Planos de encuadre

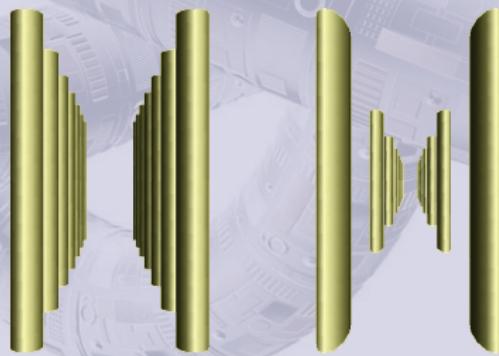
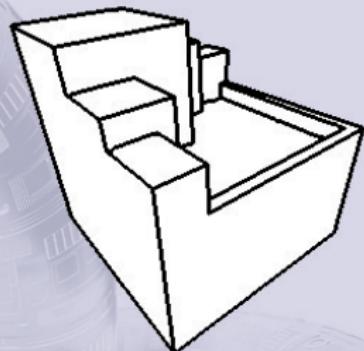
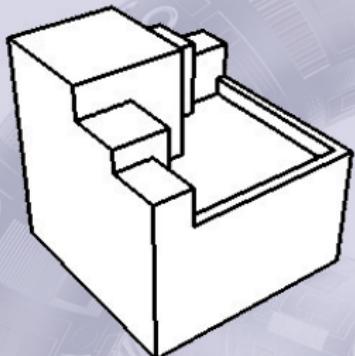


Parámetros de una cámara

Vista cónica



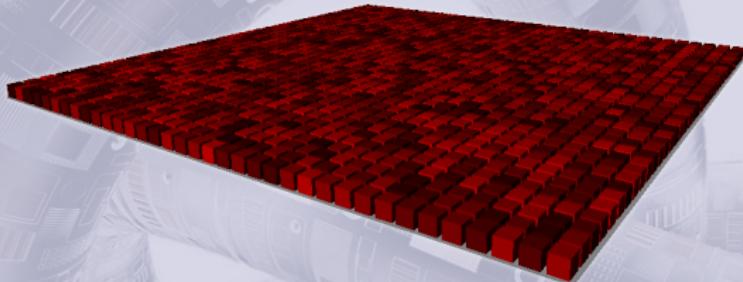
Parámetros de una cámara



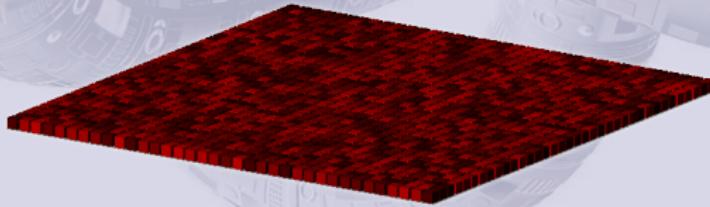
Vistas

Three.js

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



- ▶ Ortográficas, con `THREE.OrthographicCamera`



Vistas

Creación

- Se crean con:

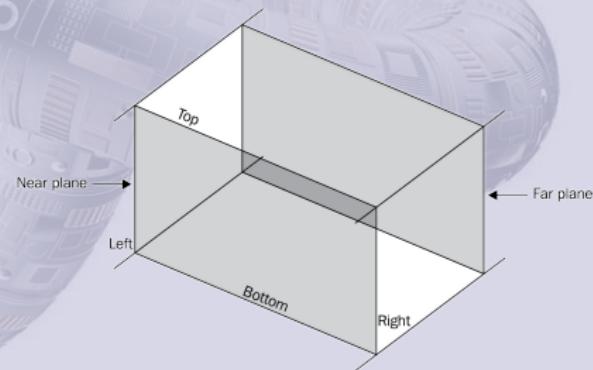
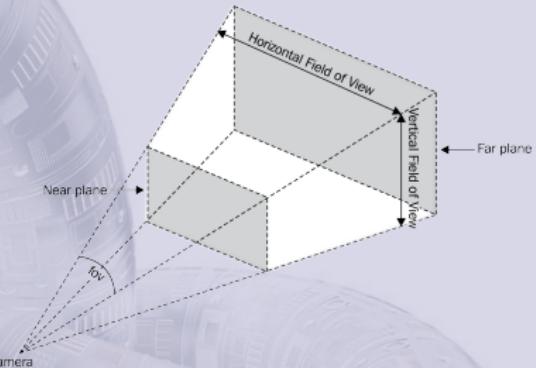
- ▶ `THREE.PerspectiveCamera`
(`fov`, `aspect`, `near`, `far`)
- ▶ `THREE.OrthographicCamera`
(`left`, `right`, `top`,
`bottom`, `near`, `far`)

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

```
// camara siempre quiere captar a personaje  
// si se mueve el personaje  
  
// this.nuevoTarget es un Vector3 que se tiene para esto  
  
// se obtiene la posición del personaje en 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 posición y 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 relativa de la cámara con relación al casco
camara.position.set(0,0.15,0);

// Posición relativa del punto de mira con relación a la cámara
var puntoDeMiraRelativo = new THREE.Vector3(0.10,-0.05,0);

// Se obtiene la posición de la cámara en coordenadas del mundo
var target = new THREE.Vector3();
camara.getWorldPosition(target);

// Se modifica con el punto de mira relativo
target.add(puntoDeMiraRelativo);

// Ahora se usa el target modificado para ajustar el lookAt de la cámara
camara.lookAt(target);
```

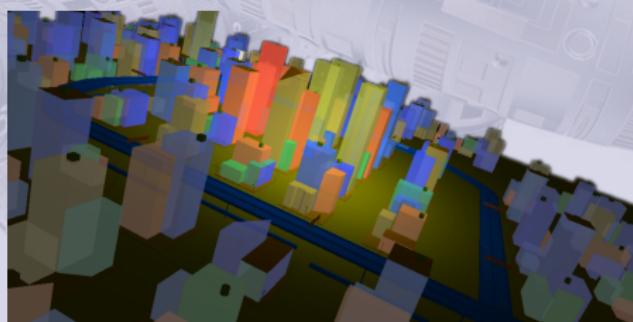


Controlador de movimientos de cámara

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



Necesidad de usar de un objeto THREE.Clock

- El controlador FlyControls necesita 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, 100);
this.camera.position.set(10, 10, 10);
this.camera.lookAt(0, 0, 0);

// Se crea el control de vuelo
this.flyControls = new FlyControls(this.camera, this.renderer.domElement);
this.flyControls.movementSpeed = 2.5;
this.flyControls.rollSpeed = Math.PI / 24;
this.flyControls.autoForward = false;

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

Controlador de movimientos de cámara

Órbita

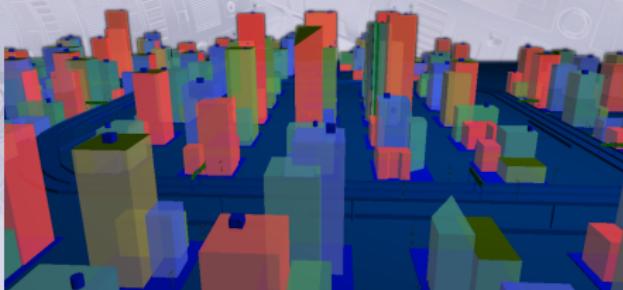
- Requiere la biblioteca `TrackballControls.js`
- `import { TrackballControls } from '../libs/TrackballControls.js'`

Control

Arrastre con botón izquierdo
Arrastre con botón derecho
Rueda del ratón

Movimiento

Giros respecto al centro de la escena
Reencuadre
Zoom



Controlador de movimientos de cámara

Primera persona con la clase `PointerLockControls`

- Se usa la clase **PointerLockControls**, que permite:
 - ▶ Capturar el ratón para “mirar” y dejarlo libre para usar el ratón para otras interacciones
 - ▶ Avanzar en la dirección en la que miramos, sin modificar la altura
 - ▶ Obtener la dirección en la que miramos para usarla en un raycast y detectar si vamos a colisionar al avanzar

PointerLockControls: Construcción

```
// Se hace el import correspondiente
import { PointerLockControls } from '../libs/PointerLockControls.js'

// Se crea la cámara de la manera habitual
camara = new THREE.PerspectiveCamera (45, window.innerWidth / window.innerHeight , 5, 200)
camara.position.set (0,1.75,0);
camara.lookAt (new THREE.Vector3 (0,1.75,-1));
escena.add (camara);

// Y después el controlador
controlador = new PointerLockControls (camara, renderer.domElement)
```

PointerLockControls

Captura y liberación del ratón

- Para usar el movimiento del ratón para “mirar” hay que capturarlo
- Y liberarlo para poder moverlo en otras interacciones sin que la cámara se vea afectada

PointerLockControls: Captura y liberación del ratón

```
// Vamos a usar la tecla Ctrl para mirar
// Hay que capturar los eventos del teclado KeyDown y KeyUp

// En onKeyDown (event) ...
case KeyCode.KEY_CONTROL :
    controlador.lock();
    break;

// En onKeyUp (event) ...
case KeyCode.KEY_CONTROL :
    controlador.unlock();
    break;
```

PointerLockControls

Avanzando sin tropezarse

- Hay que usar sus métodos `moveForward` y `getDirection` respectivamente

PointerLockControls: Avanzando sin tropezarse

```
// Vamos a avanzar con el cursor arriba
// Hay que capturar los eventos del teclado KeyDown y KeyUp

// En onKeyDown (event) ...
case KeyCode.KEY_UP : // tecla de cursor arriba
    variable_auxiliar_avanzar = true;
    break;

// En el método update()
if (variable_auxiliar_avanzar) {
    // Chocamos si avanzamos?
    donde_estoy.copy (camara.position);           // Reutilizamos un par de variables Vector3
    controlador.getDirection(a_donde_miro);
    // El test se hace con la dirección en horizontal
    a_donde_miro.y = 0;                          // ahora el vector apunta hacia donde avanzariamos
    a_donde_miro.normalize();                    // debe estar normalizado
    if (! testColision (donde_estoy, a_donde_miro)) // parámetros para 'setear' un raycast
        controlador.moveForward (cantidad_de_avance); // Sin peligro, avanzamos
}

```

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

- Se diferencian en cómo se distribuye el flujo luminoso

Luz ambiental: AmbientLight

- Se usa para simular la luz indirecta
- Ilumina toda la escena por igual
- Constructor: `AmbientLight (color, intensity)`
 - ▶ Se usa con una intensidad pequeña, entre 0 y 1
 - ▶ Normalmente con un color neutro (blanco) pero podría ser otro color para 'dar un tono' general a la escena



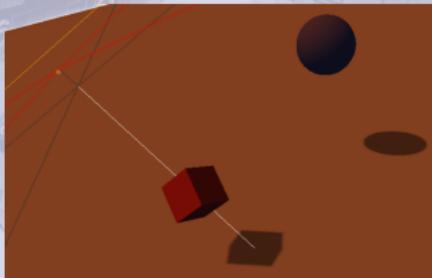
Ejemplo: Definición de la luz ambiental

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

Luz direccional

DirectionalLight

- Emite rayos paralelos entre sí según una dirección
- Se usa en exteriores para simular la luz del sol
- Constructor: `DirectionalLight (color, intensity)`
 - ▶ El color suele ser blanco (luz día) pero se pueden usar tonos cálidos para simular un atardecer
 - ▶ La intensidad suele ser un valor entre 0 y 1 (o ligeramente mayor para días especialmente luminosos)
 - ▶ La dirección se establece con el atributo `position`
Los rayos siguen la dirección del vector $\overrightarrow{position, (0, 0, 0)}$



Luz puntual

PointLight

- Desde una determinada posición, ilumina en todas direcciones
- Se usa para simular una luz artificial con ese flujo luminoso
 - ▶ En interiores: una bombilla colgando del cable, una vela, etc.
 - ▶ En exteriores: una farola con la luminaria esférica
- Tiene una caída cuadrática (físicamente correcta)
- Constructor: `PointLight (color)`
- Atributos:
 - ▶ `power`, la potencia en lúmenes de la luz
 - ▶ `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);
pointLight.power = 200;
pointLight.position.set (0, 3, 0); // bombilla a 3 m de altura
```



¿Cuánta potencia en lúmenes necesito?

- En internet encontramos recomendaciones según la estancia a iluminar y el uso que se hace de dicha estancia (en lux por m^2)
 - ▶ Multiplicando por los m^2 de la habitación obtengo la potencia en lúmenes a conseguir entre todas las luces de la estancia

Ejemplo: Iluminación de un dormitorio de $10\ m^2$ y una zona de estudio

- El dormitorio lo tengo iluminado con 4 focos integrados en el techo
 - ▶ Veo que recomiendan entre 100 y 200 lux por m^2
 - ▶ Para los $10\ m^2$ de la habitación necesitaré conseguir entre 1.000 y 2.000 lúmenes
 - ▶ Cada foco del techo me tiene que proporcionar entre 250 y 500 lúmenes
- En mi mesa de estudio tengo un flexo que me ilumina solo la mesa ($1\ m^2$ aprox.)
 - ▶ La recomendación es de 500 lux por m^2 , por tanto la bombilla del flexo procurará que sea de 500 lúmenes.

Recomendación de iluminación (lux por m²)

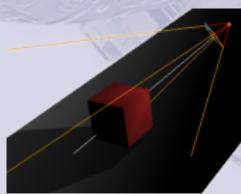
<https://blog.ledbox.es/niveles-recomendados-lux/>

- Sala de estar: Iluminación general 100 lux,
para ver la tele 50-70 lux, y para leer 500 lux
- Dormitorios: Iluminación general 100-200 lux,
y en la zona donde se lee 500 lux
- Cocina: Iluminación general 300 lux,
y en la zona de cortar y de preparado 500-600 lux
- Baño: Iluminación general 200 lux,
y en la zona del espejo 300-500 lux
- Escaleras: Iluminación general mínimo 100 lux
- Otros: 500-750 lux donde hagan trabajos manuales

Luz focal

SpotLight

- Desde una posición, ilumina en una dirección y ángulo
- Se usa para simular una luz artificial con ese flujo luminoso
 - ▶ En interiores: un flexo, una luminaria que emite un cono de luz, etc.
 - ▶ En exteriores: una farola con una luminaria cónica
- Tiene una caída cuadrática (físicamente correcta)
- Constructor: `SpotLight (color)`
- Atributos: (además de power, position, visible)
 - ▶ angle, desde su dirección, por defecto es $\pi/3$ (máximo $\pi/2$)
 - ▶ penumbra, entre 0 y 1, cómo es la transición entre zona iluminada y sombra
 - ▶ target, el objeto a donde apunta la luz (debe estar en la escena)



penumbra baja



penumbra alta

SpotLight

Ejemplo: Definición de una luz tipo foco con luz cónica

```
// Se crea con un color
var spotLight = new THREE.SpotLight (0xfcfcfc);
// Se ajustan el resto de sus parámetros principales
spotLight.power = 350; // 350 lúmenes
spotLight.angle = Math.PI/6; // pi/6 hacia cada lado = pi/3 en total
spotLight.penumbra = 1; // transición suave entre lo iluminado y la sombra
spotLight.position.set (0, 5, 0); // situada a 5 m de altura
spotLight.target = elCoche; // un Object3D que representa un coche
// Que no se nos olvide añadir la luz a la escena
scene.add (spotLight);
```



Luces rectangulares

- La luz no proviene de un punto sino de un área rectangular
 - ▶ Simula la luz que puede entrar por una ventana,
 - ▶ la que emite un plafón de techo tipo “oficina”,
 - ▶ o la que emite una pantalla de un estudio fotográfico



Luz rectangular

RectAreaLight

- A tener en cuenta:

- ▶ No emiten sombras
- ▶ Solo tienen efecto con los materiales físicos
- ▶ La luz se emite solamente hacia el lado de la Z negativa
- ▶ Hay que incluir e inicializar la clase `RectAreaLightUniformsLib`
- ▶ Hay que incluir y usar la clase `RectAreaLightHelper` para que se vea la luminaria y no solo su efecto

- Constructor:

`RectAreaLight (color, intensidad, anchoX, altoY)`

- Atributos:

- ▶ `power`, su potencia en lúmenes
- ▶ Se posicionan y orientan como cualquier otra figura, usando sus atributos `position` y `rotation`

RectAreaLight

Ejemplo

Ejemplo: Definición de una luz rectangular

```
// import OBLIGATORIO
import { RectAreaLightUniformsLib } from '../libs/RectAreaLightUniformsLib.js'

// import si queremos que se vea el rectángulo de la luminaria
import { RectAreaLightHelper } from '../libs/RectAreaLightHelper.js'

// OBLIGATORIO: Inicializar la clase RectAreaLightUniformsLib
RectAreaLightUniformsLib.init();

// Creamos, configuramos, orientamos y posicionamos la luz
var pantallaR = new THREE.RectAreaLight (0xff0000, 1, 4, 5);
pantallaR.power = 800;
pantallaR.rotation.y = -Math.PI/4;
pantallaR.position.set (-4, 2.5, 2);

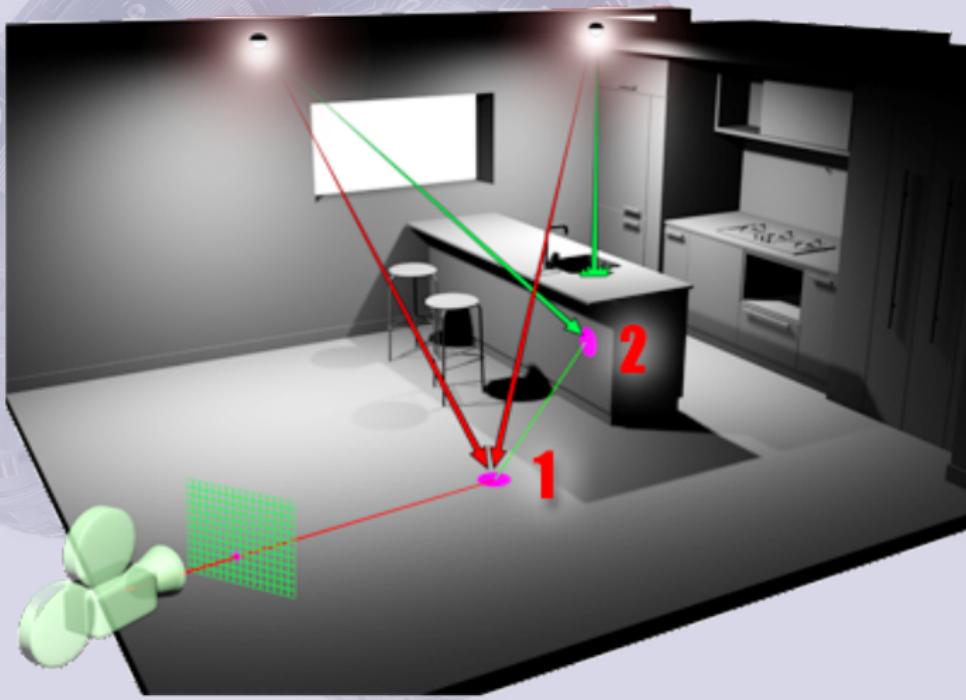
// La añadimos a la escena
escena.add (pantallaR);

// Hacemos que sea visible la luminaria, no solo los efectos de su luz
escena.add (new RectAreaLightHelper (pantallaR));
```



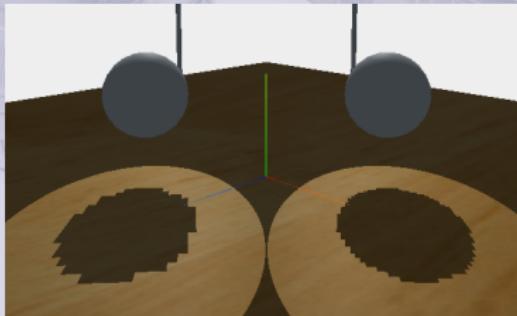
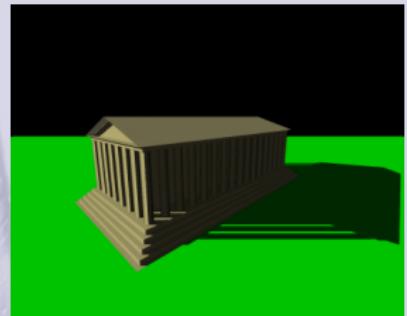
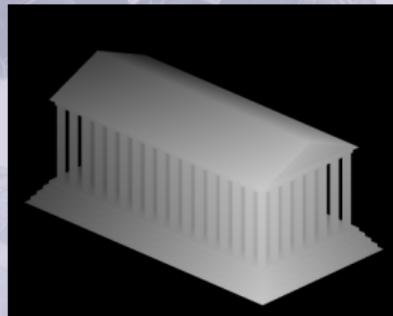
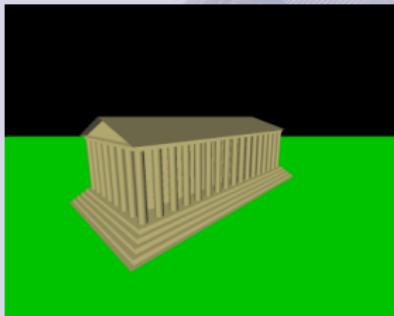
Sombras arrojadas

- Cálculo mediante Ray-Tracing



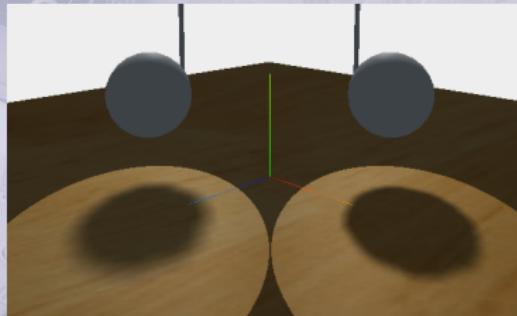
Sombras arrojadas

- 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

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

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

```
light.castShadow = true;           esta luz arroja sombras  
light.shadow.mapSize.width = 512;  la resolución del mapa  
light.shadow.mapSize.height = 512; valores por defecto  
light.shadow.camera.near = 0.5;    distancia mínima y máxima  
light.shadow.camera.far = 50;      con sombra
```

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

```
light.shadow.camera.left = -5;     valores por defecto  
light.shadow.camera.bottom = -5;   fuera de estos límites  
light.shadow.camera.right = 5;    NO hay sombras  
light.shadow.camera.top = 5;      puede que haya que ampliarlos  
son valores en unidades del mundo que, recordad, estamos considerando que son metros
```

Sombras arrojadas en

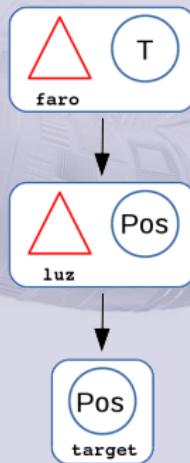
Three.js

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
- El target, que indica hacia donde 'apunta' la luz, se cuelga de la luz y se posiciona relativamente a esta



Visualización: Vistas y Luces

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática
Curso 2023-2024