

Texturas y THREE.js

<a.soroa@ehu.eus>

EHU

Texturas en THREE.js.

- El uso de texturas en THREE.js es muy sencillo.
- Dos pasos:

- ① Crear un objeto texture con una imagen

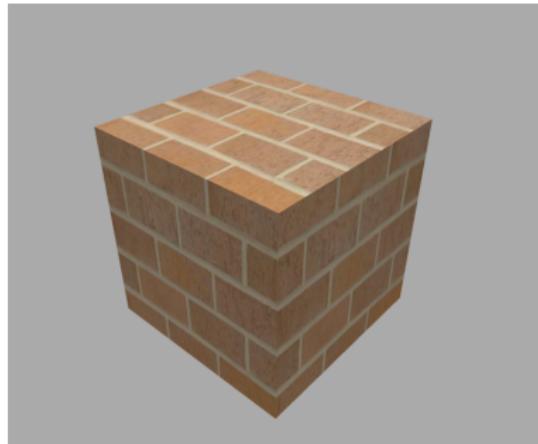
```
THREE.ImageUtils.loadTexture
```

- ② Asignar la textura a un material

campo map

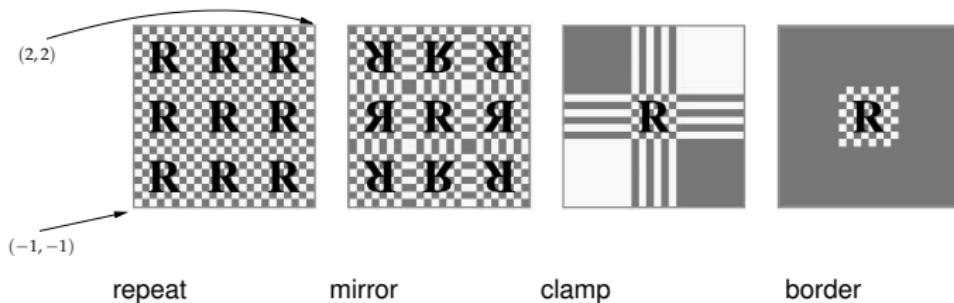
Texturas en THREE.js.

```
// cargar una textura y asignarlo a un material  
var texture = THREE.ImageUtils.loadTexture( "brick.jpg" );  
var material = new THREE.MeshPhongMaterial( { map: texture } );  
// crear un objeto con la textura  
var obj = new THREE.Mesh(new THREE.BoxGeometry(1,1,1), material);  
scene.add(obj);
```



Crear objeto textura

- Usar el método `THREE.ImageUtils.loadTexture` para cargar una imagen.
 - la resolución debe ser potencia de dos.
- Modificar parámetros de la textura:
 - mapping: cómo aplicar las texturas. El valor por defecto es usar las coordenadas (u, v).
 - wrapS: Qué hacer si la coordenada u (s en OpenGL) es mayor que 1.
 - wrapT: Qué hacer si la coordenada v (t en OpenGL) es mayor que 1.



Crear objeto textura

- Parámetros de la textura:

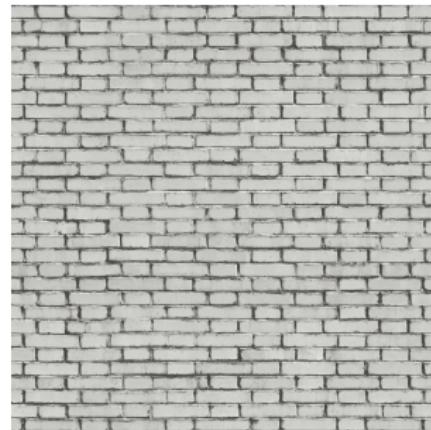
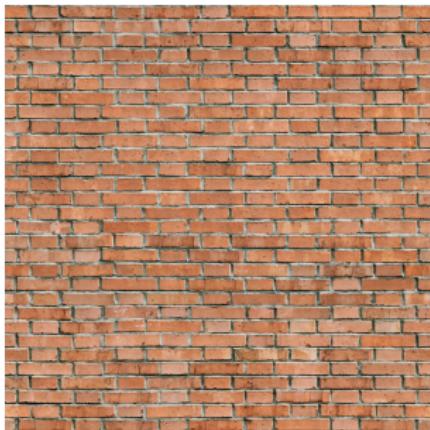
- `magFilter`: El filtro *magnification*. Los posibles valores son `THREE.NearestFilter`, `THREE.LinearFilter`
- `minFilter`: El filtro *minification*. Los posibles valores son
 - Sin *mipmapping*: `THREE.NearestFilter`, `THREE.LinearFilter`
 - Con *mipmapping*: `THREE.LinearMipMapLinearFilter`,
`THREE.NearestMipMapLinearFilter`,
`THREE.LinearMipMapNearestFilter`,
`THREE.NearestMipMapNearestFilter`
- `generateMipmaps`: Utilizar *mipmap* (y generarlos automáticamente). Por defecto es `true`.

Ejemplo

```
var texture = THREE.ImageUtils.loadTexture( "brick.jpg" );
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.RepeatWrapping;
texture.generateMipmaps = true;
texture.magFilter = THREE.LinearFilter;
texture.minFilter = THREE.LinearMipMapLinearFilter;
```

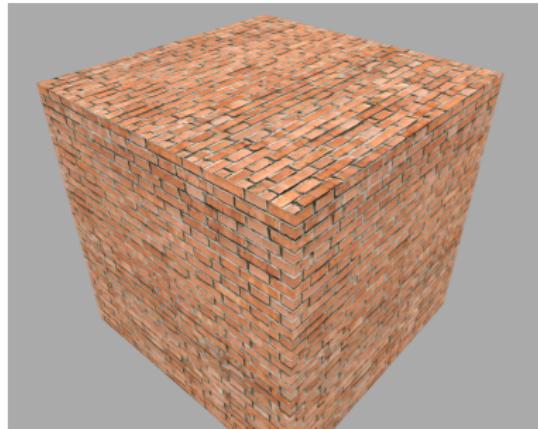
Bump mapping/ Normal mapping

- Rugosidad en las superficies.
- Se utilizan dos texturas:
 - textura de color (`map`)
 - textura de rugosidad (`normalMap`)



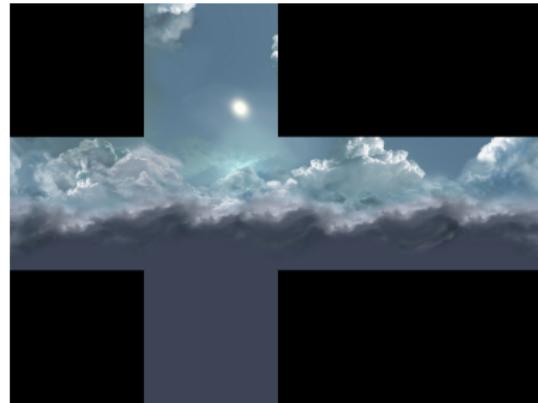
Bump mapping/ Normal mapping

```
var material = new THREE.MeshPhongMaterial( {  
    color: 0xdddddd,  
    specular: 0x222222,  
    shininess: 35,  
    map: THREE.ImageUtils.loadTexture( "images/brick_diffuse.jpg" ),  
    normalMap: THREE.ImageUtils.loadTexture( "images/brick_bump.jpg" ),  
    normalScale: new THREE.Vector2( 0.8, 0.8 )  
} );
```



Environment mapping

- Utilizar una textura para reflejar los objetos de alrededor:
 - siempre que no se muevan.
- Uso típico: reflejar el cielo en los objetos.
- Cube map:



Environment mapping

```
var skyBoxFnames = [
  "images/sky_xpos.png",
  "images/sky_xneg.png",
  "images/sky_ypos.png",
  "images/sky_yneg.png",
  "images/sky_zpos.png",
  "images/sky_zneg.png"
];
var cubemap = THREE.ImageUtils.loadTextureCube(skyBoxFnames);
var skyShader = THREE.ShaderLib[ "cube" ];
skyShader.uniforms[ "tCube" ].texture = cubemap;
var reflectionMaterial = new THREE.MeshPhongMaterial({
  color: 0xcccccc,
  envMap: cubemap
});
```

Environment mapping



Otros usos

- THREE.js también soporta:
 - Specular map
 - Occlusion map
 - Light map
- Además, podemos usar las texturas en los *shaders*

Utilizando shaders

- Recordad el proceso de asignación de texturas:
 - Cada vértice tiene una coordenada de textura.
 - en forma de variable `attribute`
 - En `THREE.js`: el atributo `uv` (de tipo `vec2`)
 - Los fragmentos reciben una coordenada de textura por interpolación.
 - La imagen se recibe por medio de una variable `uniform` de tipo `sampler2D` (en caso de texturas 2D)
 - El *fragment shader* utiliza la función

```
texture2D(sampler, coordinate)
```

para acceder al color RGBA.

Ejemplo mínimo

Vertex shader:

```
varying vec3 color; // color vertice
varying vec2 tCoord; // coord. de textura
void main() {
    color = calculate_ilum(normal, ...); // calculo de iluminacion
    tCoord = uv; // el atributo "uv" esta implicitamente definido en THREE.js
    ...
}
```

Fragment shader:

```
varying vec3 color; // color fragmento
varying vec2 tCoord; // textura de fragmento
uniform sampler2D uTexture; // la textura
void main() {
    vec4 texColor = texture2D(uTexture, tCoord);
    gl_FragColor = mix(texColor, vec4(color,1), 0.5); // mezcla color con ←
    textura
}
```

Ejemplo mínimo

Programa THREE.js

```
var texture = THREE.ImageUtils.loadTexture( "image.jpg" );
// cambiar parametros de la textura (wrap, filtros , etc)
var material = new THREE.ShaderMaterial( {
    uniforms: {
        uTexture: { type: "t", value: texture },
    },
    vertexShader: document.getElementById( "vertexShader" ).textContent,
    fragmentShader: document.getElementById( "fragmentShader" ).textContent
});
```