EBook Gratis

APRENDIZAJE three.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#three.js

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: three-js

It is an unofficial and free three.js ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official three.js.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con three.js

Observaciones

El objetivo del proyecto es crear una biblioteca 3D liviana con un nivel de complejidad muy bajo, en otras palabras, para los maniquíes. La biblioteca proporciona representaciones de lienzo, svg, CSS3D y WebGL.

Versiones

Versión	Registro de cambios	Fecha de lanzamiento
R85	Enlazar	2017-04-25
R84	Enlazar	2017-01-19
R83	Enlazar	2016-12-15
R82	Enlazar	2016-12-15
R81	Enlazar	2016-09-16
R80	Enlazar	2016-08-23
R79	Enlazar	2016-07-14
R78	Enlazar	2016-06-20

Examples

Instalación o configuración

- Puede instalar three.js a través de npm:

```
npm install three
```

- Puedes agregarlo desde un CDN a tu HTML:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r83/three.js"></script>
```

- Puede usar el editor three.js para intentarlo y descargar el proyecto como ejemplo o punto de partida.

Placa de calderas simple: cubo giratorio y controles de órbita con

amortiguación

Este es el archivo HTML básico que se puede usar como repetitivo al iniciar un proyecto. Esta placa de calderas utiliza controles de órbita con amortiguación (cámara que puede moverse alrededor de un objeto con efecto de desaceleración) y crea un cubo giratorio.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Three.js Boilerplate</title>

        <!--This is important to get a correct canvas size on mobile-->
        <meta name='viewport' content='width=device-width, user-scalable=no'/>

        <style>
            body{
                margin:0;
                overflow:hidden;
            }

            /*
              Next 2 paragraphs are a good practice.
              In IE/Edge you have to provide the cursor images.
            */
            canvas{
                cursor:grab;
                cursor:-webkit-grab;
                cursor:-moz-grab;
            }
            canvas:active{
                cursor:grabbing;
                cursor:-webkit-grabbing;
                cursor:-moz-grabbing;
            }
        </style>
    </head>
    <body>

        <script src='three.js/build/three.js'></script>
        <script src='three.js/examples/js/controls/OrbitControls.js'></script>

        <script>
            var scene, renderer, camera, controls, cube;

            init();

            function init () {
                renderer = new THREE.WebGLRenderer();

                //this is to get the correct pixel detail on portable devices
                renderer.setPixelRatio( window.devicePixelRatio );

                //and this sets the canvas' size.
                renderer.setSize( window.innerWidth, window.innerHeight );
                document.body.appendChild( renderer.domElement );

                scene = new THREE.Scene();

                camera = new THREE.PerspectiveCamera(
```

```
                    70,                                         //FOV
                    window.innerWidth / window.innerHeight,     //aspect
                    1,                                          //near clipping plane
                    100                                         //far clipping plane
                );
                camera.position.set( 1, 3, 5 );

                controls = new THREE.OrbitControls( camera, renderer.domElement );
                controls.rotateSpeed = .07;
                controls.enableDamping = true;
                controls.dampingFactor = .05;

                window.addEventListener( 'resize', function () {
                    camera.aspect = window.innerWidth / window.innerHeight;
                    camera.updateProjectionMatrix();
                    renderer.setSize( window.innerWidth, window.innerHeight );
                }, false );

                cube = new THREE.Mesh(
                    new THREE.BoxGeometry( 1, 1, 1 ),
                    new THREE.MeshBasicMaterial()
                   );
                scene.add( cube );

                animate();
            }

            function animate () {
                requestAnimationFrame( animate );
                controls.update();
                renderer.render( scene, camera );

                cube.rotation.x += 0.01;
            }
        </script>
    </body>
</html>
```

¡Hola Mundo!

El ejemplo está tomado del sitio web de threejs .

Es posible que desee descargar three.js y cambiar la fuente del script a continuación.

Hay muchos más ejemplos avanzados en este enlace.

HTML:

```
<html>
<head>
    <meta charset=utf-8>
    <title>My first Three.js app</title>
    <style>
        body { margin: 0; }
        canvas { width: 100%; height: 100% }
    </style>
</head>
<body>
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r83/three.js"></script>
    <script>
        // Our JavaScript will go here.
    </script>
</body>
```

La escena básica con un cubo estático en JavaScript:

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1, 1000
);

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;
```

Para ver realmente algo, necesitamos un bucle Render ():

```
function render() {
    requestAnimationFrame( render );
    renderer.render( scene, camera );
}
render();
```

Lea Empezando con three.js en línea: https://riptutorial.com/es/three-js/topic/2102/empezando-con-three-js

Capítulo 2: Controles de cámara en Three.js

Introducción

Este documento describe cómo puede agregar fácilmente algunos controles de cámara existentes a su escena, así como proporcionar orientación sobre cómo crear controles personalizados. Tenga en cuenta que los scripts de control prefabricados se pueden encontrar en la carpeta `/examples/js/controls` de la biblioteca.

Examples

Controles de órbita

Una cámara de órbita es aquella que permite al usuario girar alrededor de un punto central, pero manteniendo un eje particular bloqueado. Esto es extremadamente popular porque evita que la escena se "incline" fuera del eje. Esta versión bloquea el eje Y (vertical) y permite a los usuarios Orbitar, Zoom y Panorámica con los botones izquierdo, central y derecho del mouse (o eventos táctiles específicos).

index.html

```
<html>
    <head>
        <title>Three.js Orbit Controller Example</title>
        <script src="/javascripts/three.js"></script>
        <script src="/javascripts/OrbitControls.js"></script>
    </head>
    <body>
        <script src="javascripts/scene.js"></script>
    </body>
</html>
```

scene.js

```
var scene, renderer, camera;
var cube;
var controls;

init();
animate();

function init()
{
    renderer = new THREE.WebGLRenderer( {antialias:true} );
    var width = window.innerWidth;
    var height = window.innerHeight;
```

```
    renderer.setSize (width, height);
    document.body.appendChild (renderer.domElement);

    scene = new THREE.Scene();

    var cubeGeometry = new THREE.BoxGeometry (10,10,10);
    var cubeMaterial = new THREE.MeshBasicMaterial ({color: 0x1ec876});
    cube = new THREE.Mesh (cubeGeometry, cubeMaterial);

    cube.position.set (0, 0, 0);
    scene.add (cube);

    camera = new THREE.PerspectiveCamera (45, width/height, 1, 10000);
    camera.position.y = 160;
    camera.position.z = 400;
    camera.lookAt (new THREE.Vector3(0,0,0));

    controls = new THREE.OrbitControls (camera, renderer.domElement);

    var gridXZ = new THREE.GridHelper(100, 10);
    gridXZ.setColors( new THREE.Color(0xff0000), new THREE.Color(0xffffff) );
    scene.add(gridXZ);

}

function animate()
{
    controls.update();
    requestAnimationFrame ( animate );
    renderer.render (scene, camera);
}
```

El script OrbitControls tiene varias configuraciones que pueden modificarse. El código está bien documentado, así que mire en OrbitControls.js para verlos. Como ejemplo, aquí hay un fragmento de código que muestra algunos de los que se están modificando en un nuevo objeto OrbitControls.

```
controls = new THREE.OrbitControls( camera, renderer.domElement );
controls.enableDamping = true;
controls.dampingFactor = 0.25;
controls.enableZoom = true;
controls.autoRotate = true;
```

Control de cámara personalizado - Deslizamiento basado en ratón

Aquí hay un ejemplo de un controlador de cámara personalizado. Esto lee la posición del mouse dentro de la ventana del cliente y luego desliza la cámara como si estuviera siguiendo al mouse en la ventana.

index.html

```
<html>
    <head>
        <title>Three.js Custom Mouse Camera Control Example</title>
```

```
        <script src="/javascripts/three.js"></script>
    </head>
    <body>
        <script src="javascripts/scene.js"></script>
    </body>
</html>
```

scene.js

```
var scene, renderer, camera;
var cube;
var cameraCenter = new THREE.Vector3();
var cameraHorzLimit = 50;
var cameraVertLimit = 50;
var mouse = new THREE.Vector2();

init();
animate();

function init()
{
    renderer = new THREE.WebGLRenderer( {antialias:true} );
    var width = window.innerWidth;
    var height = window.innerHeight;
    renderer.setSize (width, height);
    document.body.appendChild (renderer.domElement);

    scene = new THREE.Scene();

    var cubeGeometry = new THREE.BoxGeometry (10,10,10);
    var cubeMaterial = new THREE.MeshBasicMaterial ({color: 0x1ec876});
    cube = new THREE.Mesh (cubeGeometry, cubeMaterial);

    cube.position.set (0, 0, 0);
    scene.add (cube);

    camera = new THREE.PerspectiveCamera (45, width/height, 1, 10000);
    camera.position.y = 160;
    camera.position.z = 400;
    camera.lookAt (new THREE.Vector3(0,0,0));
    cameraCenter.x = camera.position.x;
    cameraCenter.y = camera.position.y;

    //set up mouse stuff
    document.addEventListener('mousemove', onDocumentMouseMove, false);
    window.addEventListener('resize', onWindowResize, false);

    var gridXZ = new THREE.GridHelper(100, 10);
    gridXZ.setColors( new THREE.Color(0xff0000), new THREE.Color(0xffffff) );
    scene.add(gridXZ);
}

function onWindowResize ()
{
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize (window.innerWidth, window.innerHeight);
}
```

```
function animate()
{
    updateCamera();
    requestAnimationFrame ( animate );
    renderer.render (scene, camera);
}

function updateCamera() {
    //offset the camera x/y based on the mouse's position in the window
    camera.position.x = cameraCenter.x + (cameraHorzLimit * mouse.x);
    camera.position.y = cameraCenter.y + (cameraVertLimit * mouse.y);
}

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

function onWindowResize() {
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize(window.innerWidth, window.innerHeight);
}
```

Como puede ver, aquí solo estamos actualizando la posición de la cámara durante la fase `animate` la representación, como podríamos hacer con cualquier objeto en la escena. En este caso, simplemente estamos reubicando la cámara en un punto de desplazamiento con respecto a sus coordenadas X e Y originales. Esto podría ser igual de fácil que las coordenadas X y Z, o un punto a lo largo de una ruta, o algo completamente diferente ni siquiera relacionado con la posición del mouse.

Lea Controles de cámara en Three.js en línea: https://riptutorial.com/es/three-js/topic/8270/controles-de-camara-en-three-js

Capítulo 3: Geometrías

Observaciones

Los ejemplos funcionan a partir de three.js R79 (revisión 79).

Examples

TRES.

THREE.BoxGeometry construye cajas como cuboides y cubos.

Cubitos

Los cubos creados con THREE.BoxGeometry usarían la misma longitud para todos los lados.

JavaScript

```
//Creates scene and camera

var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1,
1000 );

//Creates renderer and adds it to the DOM

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

//The Box!

//BoxGeometry (makes a geometry)
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
//Material to apply to the cube (green)
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
//Applies material to BoxGeometry
var cube = new THREE.Mesh( geometry, material );
//Adds cube to the scene
scene.add( cube );

//Sets camera's distance away from cube (using this explanation only for simplicity's sake -
in reality this actually sets the 'depth' of the camera's position)

camera.position.z = 5;

//Rendering

function render() {
  requestAnimationFrame( render );
  renderer.render( scene, camera );
```

```
}
render();
```

Observe la función 'render'. Esto hace que el cubo 60 veces por segundo.

Código completo (con HTML)

```
<!DOCTYPE html>
<html>

  <head>
    <title>THREE.BoxGeometry</title>
    <script src="http://threejs.org/build/three.js"></script>
  </head>

  <body>

    <script>
      //Above JavaScript goes here
    </script>

  </body>

</html>
```

Cuboides

La línea `var geometry = new THREE.BoxGeometry( 1, 1, 1 );` nos da un cubo. Para hacer un cuboide, simplemente cambie los parámetros, ellos definen la longitud, la altura y la profundidad del cubo respectivamente.

Ejemplo:

```
...
//Longer cuboid
var geometry = new THREE.BoxGeometry( 2, 1, 1 );
...
```

Más (demostrando que el cubo es tridimensional)

El cubo puede parecer solo un cuadrado. Para probar que es, sin lugar a dudas, tridimensional, agregue las siguientes líneas de código a la función 'render':

```
...
cube.rotation.x += 0.05;
cube.rotation.y += 0.05;
...
```

Y mira como el cubo feliz gira alrededor ... y redondo ... y redondo ...

Vistoso

No para el débil de corazón...

El color uniforme para todo el cubo es ... verde. Aburrido. Para hacer que cada cara tenga un color diferente, tenemos que excavar en las caras de la geometría.

```
var geometry = new THREE.BoxGeometry(3, 3, 3, 1, 1, 1);

/*Right of spawn face*/
geometry.faces[0].color = new THREE.Color(0xd9d9d9);
geometry.faces[1].color = new THREE.Color(0xd9d9d9);

/*Left of spawn face*/
geometry.faces[2].color = new THREE.Color(0x2196f3);
geometry.faces[3].color = new THREE.Color(0x2196f3);

/*Above spawn face*/
geometry.faces[4].color = new THREE.Color(0xffffff);
geometry.faces[5].color = new THREE.Color(0xffffff);

/*Below spawn face*/
geometry.faces[6].color = new THREE.Color(1, 0, 0);
geometry.faces[7].color = new THREE.Color(1, 0, 0);

/*Spawn face*/
geometry.faces[8].color = new THREE.Color(0, 1, 0);
geometry.faces[9].color = new THREE.Color(0, 1, 0);

/*Opposite spawn face*/
geometry.faces[10].color = new THREE.Color(0, 0, 1);
geometry.faces[11].color = new THREE.Color(0, 0, 1);

var material = new THREE.MeshBasicMaterial( {color: 0xffffff, vertexColors: THREE.FaceColors}
);
var cube = new THREE.Mesh(geometry, material);
```

NOTA: El método de colorear las caras no es el mejor método, pero funciona bien (suficiente).

Notas

¿Dónde está el `canvas` en el cuerpo del documento HTML?

No es necesario añadir un `canvas` al cuerpo manualmente. Las siguientes tres lineas

```
var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

cree el renderizador, su `canvas` y agregue el `canvas` al DOM.

TRES.Geometría de cilindros

TRES.CilindrosGeometría construyen cilindros.

Cilindro

Continuando con el ejemplo anterior, el código para crear el cuadro podría reemplazarse con el siguiente.

```
//Makes a new cylinder with
// - a circle of radius 5 on top (1st parameter)
// - a circle of radius 5 on the bottom (2nd parameter)
// - a height of 20 (3rd parameter)
// - 32 segments around its circumference (4th parameter)
var geometry = new THREE.CylinderGeometry( 5, 5, 20, 32 );
//Yellow
var material = new THREE.MeshBasicMaterial( {color: 0xffff00} );
var cylinder = new THREE.Mesh( geometry, material );
scene.add( cylinder );
```

Para construir desde cero, aquí está el código.

```
//Creates scene and camera

var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1,
1000 );

//Creates renderer and adds it to the DOM

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

//The Cylinder!

var geometry = new THREE.CylinderGeometry( 5, 5, 20, 32 );
//Yellow
var material = new THREE.MeshBasicMaterial( {color: 0xffff00} );
var cylinder = new THREE.Mesh( geometry, material );
scene.add( cylinder );

//Sets camera's distance away from cube (using this explanation only for simplicity's sake -
in reality this actually sets the 'depth' of the camera's position)

camera.position.z = 30;

//Rendering

function render() {
  requestAnimationFrame( render );
  renderer.render( scene, camera );
}
render();
```

Más (demostrando que el cilindro es tridimensional)

El cilindro puede parecer justo ... bidimensional. Para probar que es, sin lugar a dudas, tridimensional, agregue las siguientes líneas de código a la función 'render':

```
...
cylinder.rotation.x += 0.05;
cylinder.rotation.z += 0.05;
...
```

Y el cilindro brillante y feliz giraría al azar, en medio de un fondo negro oscuro ...

Lea Geometrías en línea: https://riptutorial.com/es/three-js/topic/5762/geometrias

Capítulo 4: Mallas

Introducción

A Three.js Mesh es una clase base que se hereda de Object3d y se utiliza para instanciar objetos poligonales combinando una geometría con un material . `Mesh` también es la clase base para las clases más avanzadas de `MorphAnimMesh` y `SkinnedMesh` .

Sintaxis

- nuevo THREE.Mesh (geometría, material);

Observaciones

Tanto la geometría como el material son opcionales y se establecerán de forma predeterminada en `BufferGeometry` y `MeshBasicMaterial` respectivamente, si no se proporcionan en el constructor.

Examples

Renderiza una malla de cubos con una geometría de caja y un material básico.

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 50);
camera.position.z = 25;

var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

var geometry = new THREE.BoxGeometry(1, 1, 1);
var material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
var cubeMesh = new THREE.Mesh(geometry, material);
scene.add(cubeMesh);

var render = function () {
    requestAnimationFrame(render);

    renderer.render(scene, camera);
};

render();
```

Lea Mallas en línea: https://riptutorial.com/es/three-js/topic/8838/mallas

Capítulo 5: Recogida de objetos

Examples

Recolección de objetos / Raycasting

Raycasting significa lanzar un rayo desde la posición del mouse en la pantalla a la escena, así es como threejs determina en qué objeto desea hacer clic si lo ha implementado. Threejs obtiene esa información usando un octree , pero aún en producción, es posible que no desee calcular el resultado en cada fotograma o en el evento `mousemove` , sino en el evento `click` para una aplicación más accesible con bajos requisitos.

```
var raycaster, mouse = { x : 0, y : 0 };

init();

function init () {

    //Usual setup code here.

    raycaster = new THREE.Raycaster();
    renderer.domElement.addEventListener( 'click', raycast, false );

    //Next setup code there.

}

function raycast ( e ) {

    //1. sets the mouse position with a coordinate system where the center
    //   of the screen is the origin
    mouse.x = ( e.clientX / window.innerWidth ) * 2 - 1;
    mouse.y = - ( e.clientY / window.innerHeight ) * 2 + 1;

    //2. set the picking ray from the camera position and mouse coordinates
    raycaster.setFromCamera( mouse, camera );

    //3. compute intersections
    var intersects = raycaster.intersectObjects( scene.children );

    for ( var i = 0; i < intersects.length; i++ ) {
        console.log( intersects[ i ] );
        /*
            An intersection has the following properties :
                - object : intersected object (THREE.Mesh)
                - distance : distance from camera to intersection (number)
                - face : intersected face (THREE.Face3)
                - faceIndex : intersected face index (number)
                - point : intersection point (THREE.Vector3)
                - uv : intersection point in the object's UV coordinates (THREE.Vector2)
        */
    }

}
```

> ¡PRECAUCIÓN! Puede perder su tiempo mirando la pantalla en blanco si no lee la siguiente parte.

Si desea detectar el ayudante de luz, configure el segundo parámetro de `raycaster.intersectObjects( scene.children );` a la verdad

Significa `raycaster.intersectObjects( scene.children , true);`

El código de raycast solo detectará al ayudante de luz.

Si desea que detecte objetos normales, así como un ayudante de luz, debe volver a copiar la función raycast anterior. Vea esta pregunta .

El código completo de raycast es

```
function raycast ( e ) {
// Step 1: Detect light helper
    //1. sets the mouse position with a coordinate system where the center
    //   of the screen is the origin
    mouse.x = ( e.clientX / window.innerWidth ) * 2 - 1;
    mouse.y = - ( e.clientY / window.innerHeight ) * 2 + 1;

    //2. set the picking ray from the camera position and mouse coordinates
    raycaster.setFromCamera( mouse, camera );

    //3. compute intersections (note the 2nd parameter)
    var intersects = raycaster.intersectObjects( scene.children, true );

    for ( var i = 0; i < intersects.length; i++ ) {
        console.log( intersects[ i ] );
        /*
            An intersection has the following properties :
                - object : intersected object (THREE.Mesh)
                - distance : distance from camera to intersection (number)
                - face : intersected face (THREE.Face3)
                - faceIndex : intersected face index (number)
                - point : intersection point (THREE.Vector3)
                - uv : intersection point in the object's UV coordinates (THREE.Vector2)
        */
    }
// Step 2: Detect normal objects
    //1. sets the mouse position with a coordinate system where the center
    //   of the screen is the origin
    mouse.x = ( e.clientX / window.innerWidth ) * 2 - 1;
    mouse.y = - ( e.clientY / window.innerHeight ) * 2 + 1;

    //2. set the picking ray from the camera position and mouse coordinates
    raycaster.setFromCamera( mouse, camera );

    //3. compute intersections (no 2nd parameter true anymore)
    var intersects = raycaster.intersectObjects( scene.children );

    for ( var i = 0; i < intersects.length; i++ ) {
        console.log( intersects[ i ] );
        /*
            An intersection has the following properties :
                - object : intersected object (THREE.Mesh)
                - distance : distance from camera to intersection (number)
```

```
                - face : intersected face (THREE.Face3)
                - faceIndex : intersected face index (number)
                - point : intersection point (THREE.Vector3)
                - uv : intersection point in the object's UV coordinates (THREE.Vector2)
        */
    }

}
```

Recolección de objetos / GPU

La selección de objetos utilizando Raycasting puede ser una tarea pesada para su CPU dependiendo de su configuración (por ejemplo, si no tiene una configuración de tipo octárbol) y la cantidad de objetos en la escena.

Si no necesita las coordenadas mundiales debajo del cursor del mouse, pero solo para identificar el objeto debajo de él, puede usar la selección de GPU.

Breve explicación, la GPU puede ser una herramienta poderosa para el cálculo, pero necesita saber cómo recuperar los resultados. La idea es que, si representa los objetos con un color que representa su identificación, puede leer el color del píxel debajo del cursor y eliminar la identificación del objeto seleccionado. Recuerde que RGB es solo un valor hexadecimal, por lo que existe una conversión entre id (entero) y color (hex).

1. Crea una nueva escena y un nuevo objetivo de representación para tu objeto

```
var pickingScene = new THREE.Scene();
var pickingTexture = new THREE.WebGLRenderTarget(renderer.domElement.clientWidth,
renderer.domElement.clientHeight);
    pickingTexture.texture.minFilter = THREE.LinearFilter;
```

2. Crear un nuevo material de sombreado para recoger objetos;

```
var vs3D = `
attribute vec3 idcolor;
varying vec3 vidcolor;
void main(){
vidcolor = idcolor;
gl_Position = projectionMatrix * modelViewMatrix * vec4( position, 1.0);
}`;

var fs3D = `
varying vec3 vidcolor;
void main(void) {
gl_FragColor = vec4(vidcolor,1.0);
}`;

var pickingMaterial = new THREE.ShaderMaterial(
    {
        vertexShader: vs3D,
        fragmentShader: fs3D,
        transparent: false,
        side: THREE.DoubleSide
    });
```

3. Agregue a su geometría de malla / línea un nuevo atributo que represente su ID en RGB, cree el pickingObject usando la misma geometría y agréguelo a la escena de picking, y agregue la malla real a un diccionario de objetos id>

```
var selectionObjects = [];

for(var i=0; i<myMeshes.length; i++){
    var mesh = myMeshes[i];
    var positions = mesh.geometry.attributes["position"].array;
    var idColor = new Float32Array(positions.length);

    var color = new THREE.Color();
    color.setHex(mesh.id);

    for (var j=0; j< positions.length; j+=3){
        idColor[j] = color.r;
        idColor[j+1] = color.g;
        idColor[j+2] = color.b;
    }

    mesh.geometry.addAttribute('idcolor', new THREE.BufferAttribute(idColor, 3));

    var pickingObject = new THREE.Mesh(mesh.geometry, pickingMaterial);

    pickingScene.add(pickingObject);
    selectionObjects[mesh.id] = mesh;
}
```

4. Por último, en su ratón haga clic en el controlador.

```
renderer.render(pickingScene, camera, pickingTexture);
var pixelBuffer = new Uint8Array(4);
renderer.readRenderTargetPixels(pickingTexture, event.pageX, pickingTexture.height -
event.pageY, 1, 1, pixelBuffer);
var id = (pixelBuffer[0] << 16) | (pixelBuffer[1] << 8) | (pixelBuffer[2]);

if (id>0){
    //this is the id of the picked object
}else{
    //it's 0. clicked on an empty space
}
```

Lea Recogida de objetos en línea: https://riptutorial.com/es/three-js/topic/4848/recogida-de-objetos

Capítulo 6: Render Loops for Animation: Actualización dinámica de objetos

Introducción

Este documento describe algunas formas comunes de agregar animación directamente en sus escenas de Three.js. Si bien hay bibliotecas y marcos que pueden agregar movimiento dinámico a su escena (interpolaciones, física, etc.), es útil comprender cómo puede hacerlo usted mismo simplemente con unas pocas líneas de código.

Observaciones

El concepto central de animación es actualizar las propiedades de un objeto (rotación y traducción, generalmente) en pequeñas cantidades durante un período de tiempo. Por ejemplo, si traduces un objeto aumentando la posición X en 0.1 cada décima de segundo, será 1 unidad más en el eje X en 1 segundo, pero el espectador lo percibirá como si se hubiera movido suavemente a esa posición sobre esa posición Tiempo en lugar de saltar directamente a la nueva posición.

Para ayudarnos, creamos un *bucle de render* en el script.

```
var render = function () {
    requestAnimationFrame( render );
    //update some properties here
    renderer.render(scene, camera);
}
```

En el ejemplo de cubo giratorio anterior, usamos esta idea (pequeñas actualizaciones incrementales) para cambiar la rotación del cubo cada vez que se solicita un nuevo cuadro de animación. Al incrementar las propiedades `rotation.x` y `rotation.y` del objeto de `cube` en cada fotograma, el cubo parece girar en esos dos ejes.

Como otro ejemplo, no es raro que separe su actualización necesaria en otras funciones, donde puede hacer cálculos y verificaciones adicionales mientras mantiene el bucle de procesamiento ordenado. Por ejemplo, el siguiente bucle de procesamiento llama a cuatro funciones de actualización diferentes, cada una destinada a actualizar un objeto separado (o una matriz de objetos, en el caso de `updatePoints()`) en la escena.

```
//render loop
function render() {
    requestAnimationFrame( render );
    updateGrid();
    updateCube();
    updateCamera();
    updatePoints(pList);
    renderer.render( scene, camera);
}
render();
```

Puede observar en los ejemplos en línea que los controles de la cámara también forman parte del bucle de procesamiento.

```
controls = new THREE.OrbitControls( camera, renderer.domElement );
controls.enableDamping = true;
controls.dampingFactor = 0.25;
controls.enableZoom = true;
controls.autoRotate = true;

var render = function () {
    requestAnimationFrame( render );
    controls.update();
    renderer.render(scene, camera);
};
```

Esto se debe a que el script para controlar la cámara está haciendo lo mismo; Actualizándolo a lo largo del tiempo. Los cambios pueden ser causados por la entrada del usuario, como la posición del mouse, o algo programático, como seguir una ruta. En cualquier caso, sin embargo, también estamos animando la cámara.

Examples

Cubo de hilado

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1, 1000
);

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;

//Create an render loop to allow animation
var render = function () {
    requestAnimationFrame( render );

    cube.rotation.x += 0.1;
    cube.rotation.y += 0.1;

    renderer.render(scene, camera);
};

render();
```

Lea Render Loops for Animation: Actualización dinámica de objetos en línea: https://riptutorial.com/es/three-js/topic/8271/render-loops-for-animation--actualizacion-dinamica-de-objetos

Capítulo 7: Texturas y materiales

Introducción

Una buena introducción al material y las texturas.

Texturas difusas, rugosas, especulares y transparentes.

Parámetros

Parámetro	Detalles
color	Valor numérico del componente RGB del color.
intensidad	Valor numérico de la intensidad / intensidad de la luz.
fov	Cámara vertical de campo de visión vertical.
aspecto	Relación de aspecto cámara frustum.
cerca	Cámara frustum cerca de avión.
lejos	Cámara frustum plano lejano.
radio	radio de la esfera. El valor predeterminado es 50.
anchoSegmentos	Número de segmentos horizontales. El valor mínimo es 3, y el valor predeterminado es 8.
AlturaSegmentos	Número de segmentos verticales. El valor mínimo es 2, y el valor predeterminado es 6.
phiStart	Especifique el ángulo de inicio horizontal. El valor predeterminado es 0.
phiLength	Especifique el tamaño del ángulo de barrido horizontal. El valor predeterminado es Math.PI * 2.
thetaStart	especificar el ángulo de inicio vertical. El valor predeterminado es 0.
thetaLength	Especifique el tamaño del ángulo de barrido vertical. El valor predeterminado es Math.PI.

Observaciones

Enlace de demostración

Examples

Creando una Tierra Modelo

Las texturas para este ejemplo están disponibles en: http://planetpixelemporium.com/planets.html

Instalación o configuración

Puedes instalar tres via npm

```
npm install three
```

O agregarlo como un script a su página HTML

```
<script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r85/three.min.js" />
```

HTML:

```
<html>
<head>
    <meta charset=utf-8>
    <title>Earth Model</title>
    <style>
        body { margin: 0; }
        canvas { width: 100%; height: 100% }
    </style>
</head>
<body>
   <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r83/three.js" />
    <script>
        // Our Javascript will go here.
    </script>
</body>
</html>
```

Creando la escena

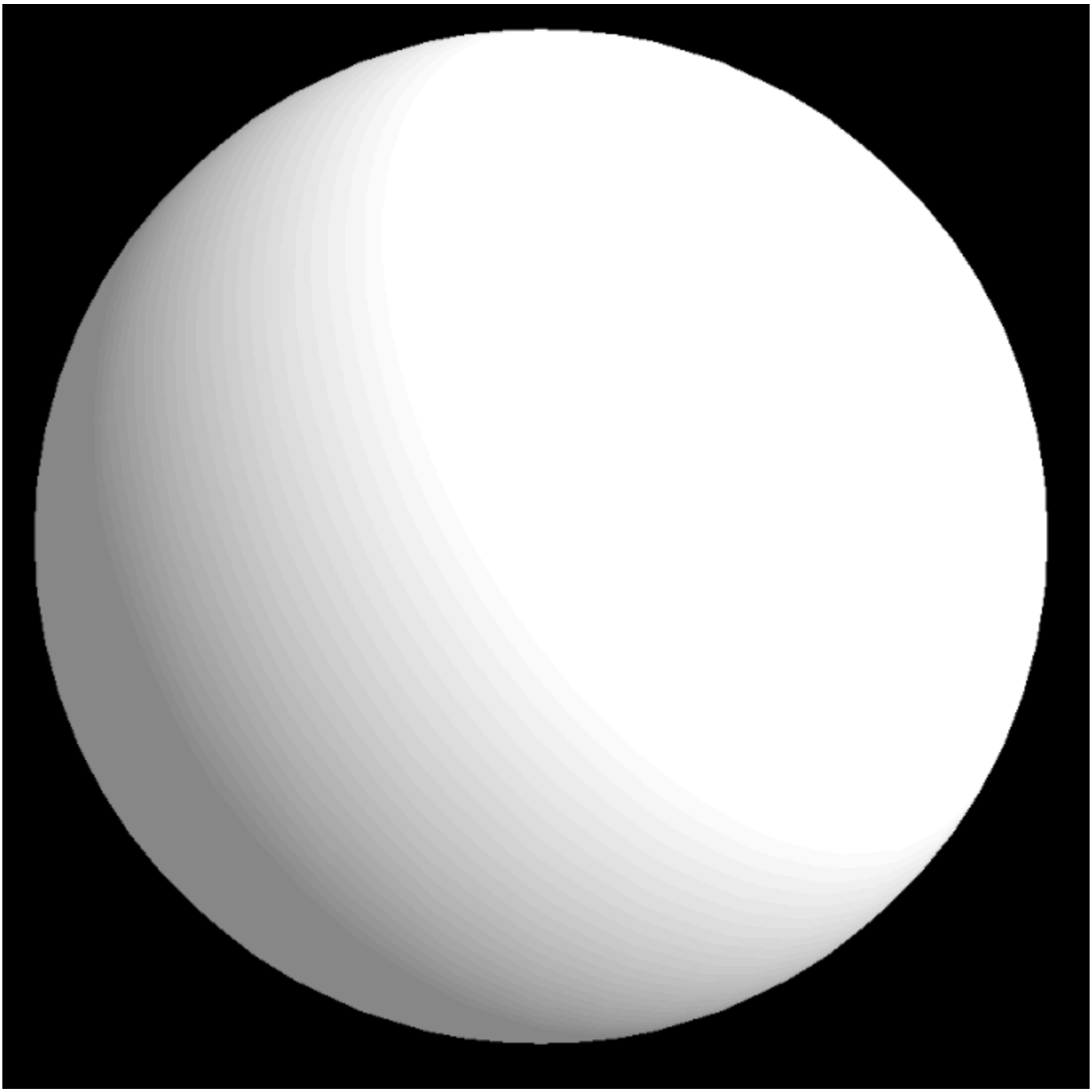
Para poder mostrar cualquier cosa con three.js, necesitamos tres cosas: una escena, una cámara y un renderizador. Vamos a hacer la escena con la cámara.

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1,
1000 );

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

Creando la esfera

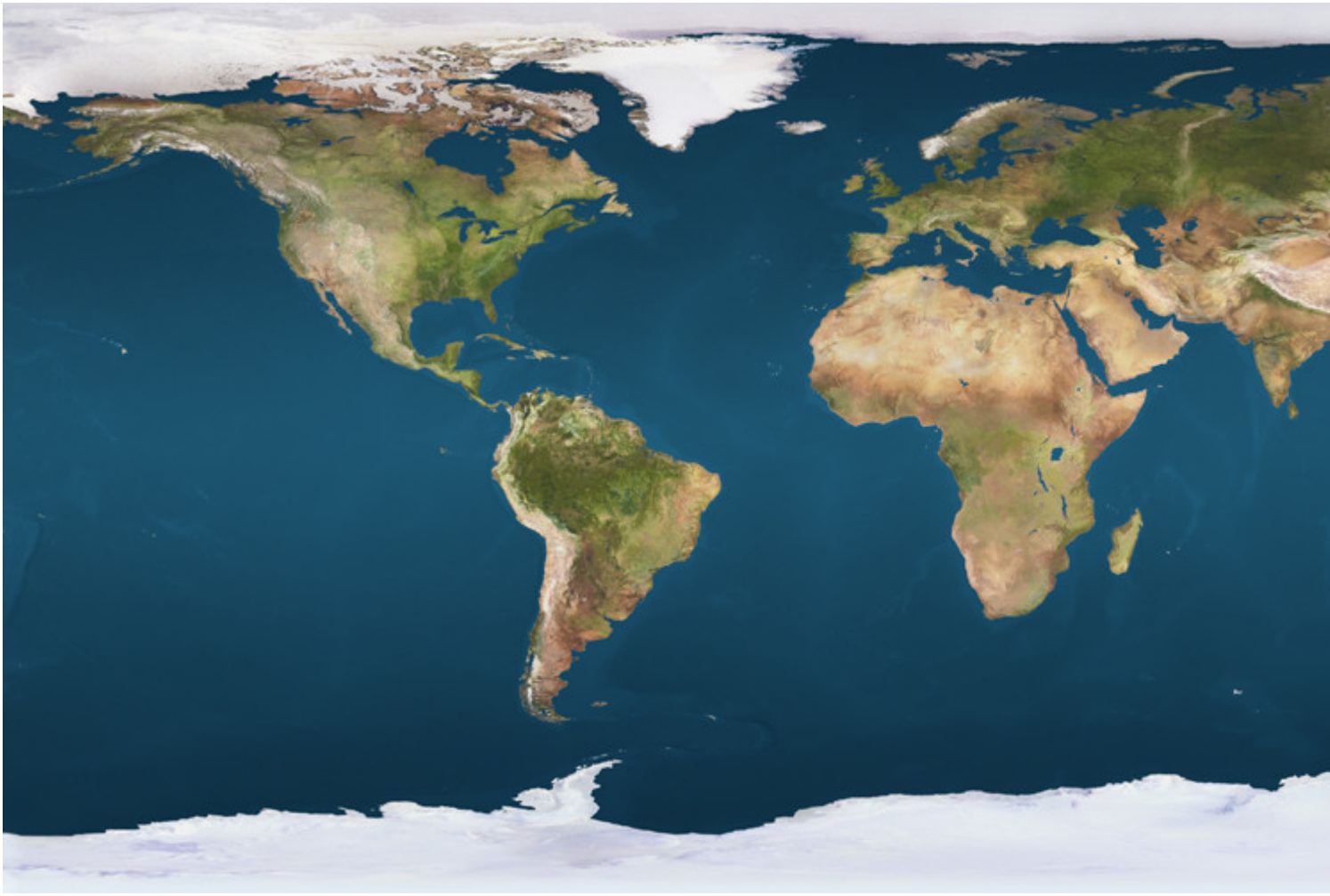
- Crear geometría para la esfera.
- Crear un material phong.
- Crea un objeto 3D
- Añádelo a la escena.



```
var geometry = new THREE.SphereGeometry(1, 32, 32);
var material = new THREE.MeshPhongMaterial();
var earthmesh = new THREE.Mesh(geometry, material);
```

Añadir una textura difusa

La textura difusa establece el color principal de la superficie. Cuando lo aplicamos a una esfera, obtenemos la siguiente imagen.



```
material.map    = THREE.ImageUtils.loadTexture('images/earthmap1k.jpg');
```

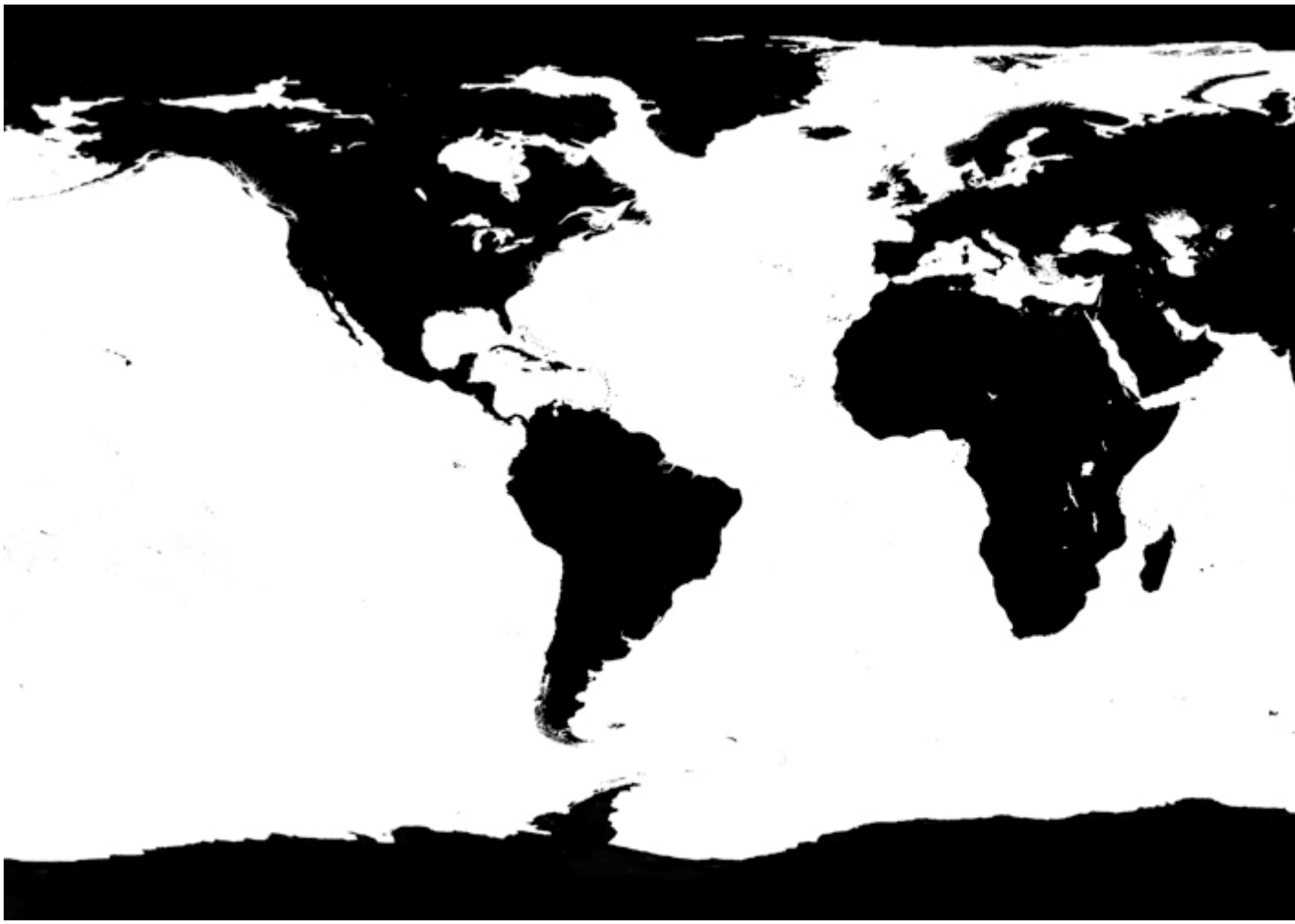
Agregar una textura de mapa de relieve

- Cada uno de sus píxeles actúa como una altura en la superficie.
- Las montañas aparecen más claramente gracias a su sombra.
- Es posible cambiar cuánto afecta el mapa a la iluminación con el parámetro bumpScale.
- No se crean ni se necesitan vértices adicionales para usar un mapa de relieve (a diferencia de un mapa de desplazamiento)

```
material.bumpMap   = THREE.ImageUtils.loadTexture('images/earthbump1k.jpg');
material.bumpScale = 0.05;
```

Añadiendo una textura especular

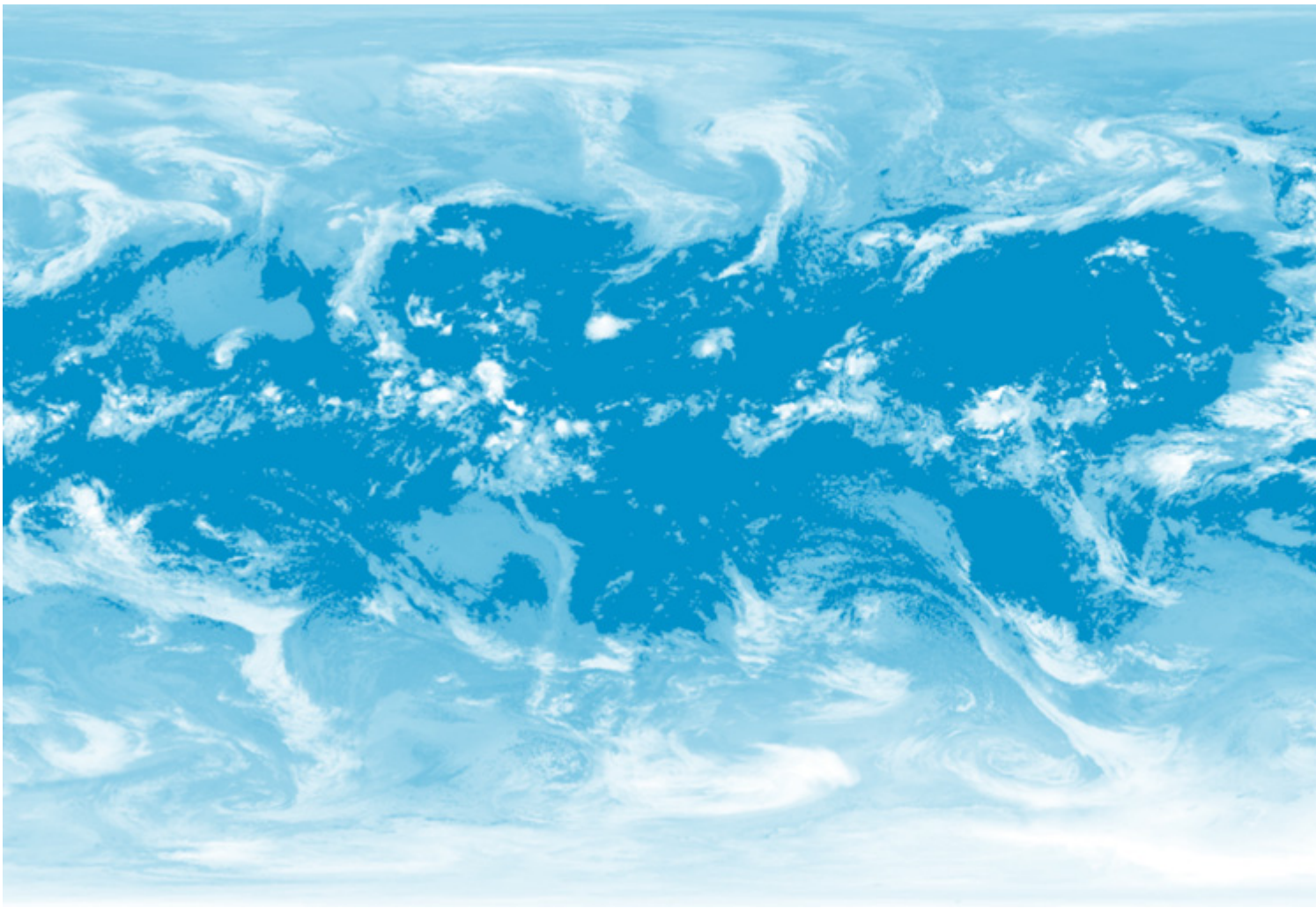
- Cambia el 'brillo' de un objeto con una textura.
- Cada píxel determina la intensidad de la especularidad.
- En este caso, solo el mar es especular porque el agua refleja la luz más que la tierra.
- Puedes controlar el color especular con el parámetro especular.

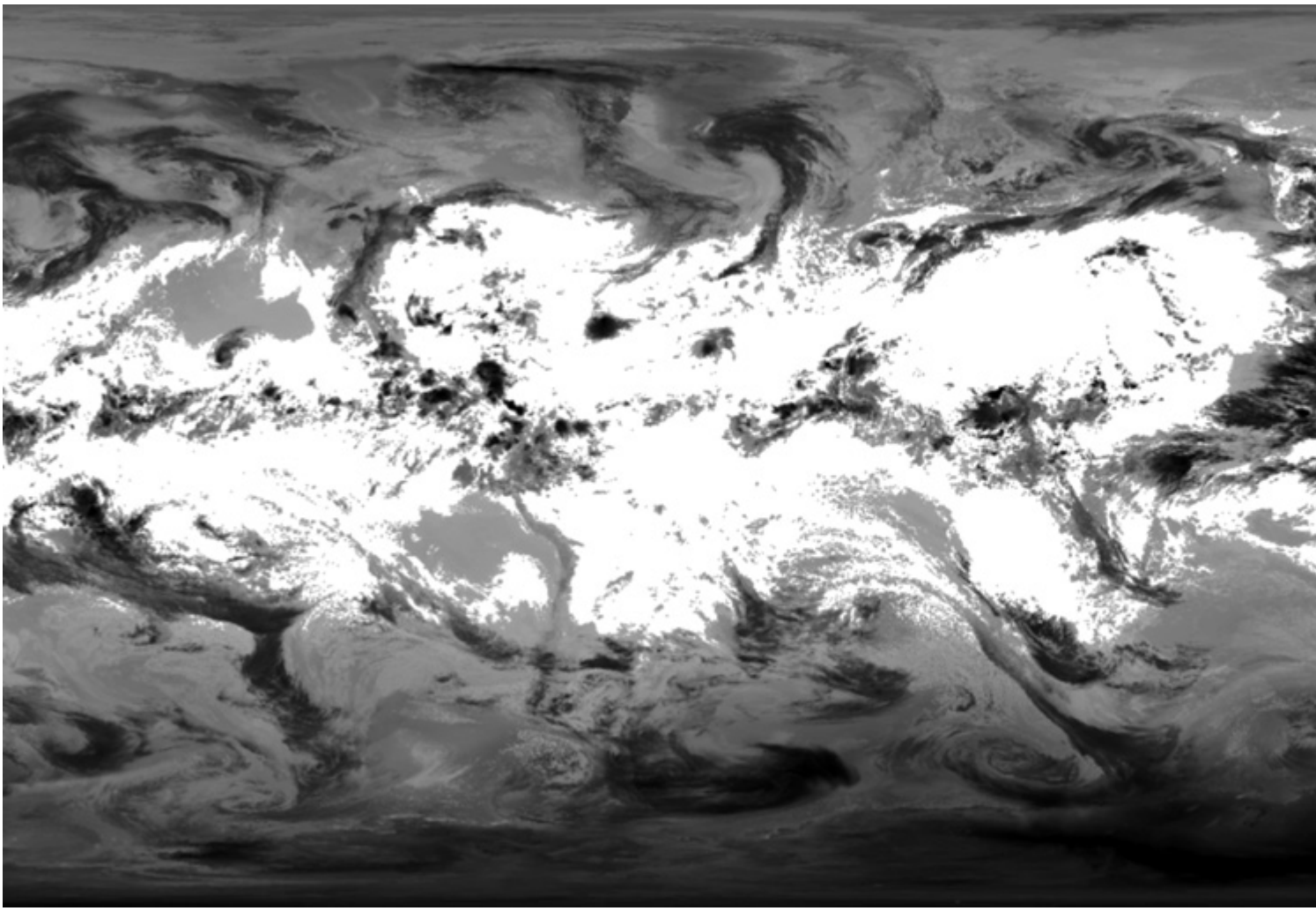


```
material.specularMap = THREE.ImageUtils.loadTexture('images/earthspec1k.jpg')
material.specular  = new THREE.Color('grey')
```

Añadiendo una capa de nubes

- Creamos `canvasCloud` con un lienzo y lo usamos como textura.
- Hacemos esto porque jpg no maneja un canal alfa. (Sin embargo, una imagen PNG hace)
- Necesitamos hacer el código para construir la textura basada en las siguientes imágenes.





```
var geometry   = new THREE.SphereGeometry(0.51, 32, 32)
var material  = new THREE.MeshPhongMaterial({
  map         : new THREE.Texture(canvasCloud),
  side        : THREE.DoubleSide,
  opacity     : 0.8,
  transparent : true,
  depthWrite  : false,
});
var cloudMesh = new THREE.Mesh(geometry, material)
earthMesh.add(cloudMesh)
```

- Adjuntamos `cloudMesh` a `earthMesh` para que se muevan juntos.
- Deshabilitamos `depthWrite` y configuramos `transparent: true` para decirle a three.js que cloudmesh es transparente.
- Establecemos los lados en `THREE.DoubleSide` para que ambos lados sean visibles.
    - Esto evita la creación de artefactos en el borde de la tierra.
- Finalmente, configuramos la `opacity: 0.8` para hacer que las nubes sean más translúcidas.

Añadiendo Movimiento Rotacional

En su bucle de render, simplemente aumenta la rotación

Como toque final, animaremos la capa de la nube para que se vea más realista.

```
updateFcts.push(function(delta, now) {
    cloudMesh.rotation.y += 1 / 8 * delta;
    earthMesh.rotation.y += 1 / 16 * delta;
  })
```

Lea Texturas y materiales en línea: https://riptutorial.com/es/three-js/topic/9333/texturas-y-materiales

Creditos

S. No	Capítulos	Contributors
1	Empezando con three.js	Atrahasis, Blogueira, Community, Gero3, guardabrazo, Hasan, Hectate, Joel Martinez, juagicre, Learn How To Be Transparent, Xander Luciano, Zeromatiker, zya
2	Controles de cámara en Three.js	Hectate
3	Geometrías	streppel, Theo
4	Mallas	Paul Graffam
5	Recogida de objetos	Gero3, Kris Roofe, Learn How To Be Transparent, winseybash
6	Render Loops for Animation: Actualización dinámica de objetos	Hectate
7	Texturas y materiales	Geethu Jose, Xander Luciano