FREE eBook

LEARNING three.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#three.js

Table of Contents

About

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

Chapter 1: Getting started with three.js

Remarks

The aim of the project is to create a lightweight 3D library with a very low level of complexity — in other words, for dummies. The library provides canvas, svg, CSS3D and WebGL renderers.

Versions

Version	Changelog	Release Date
R85	Link	2017-04-25
R84	Link	2017-01-19
R83	Link	2016-12-15
R82	Link	2016-12-15
R81	Link	2016-09-16
R80	Link	2016-08-23
R79	Link	2016-07-14
R78	Link	2016-06-20

Examples

Installation or Setup

- You can install three.js via npm:

```
npm install three
```

- You can add it from a CDN to your HTML:

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

- You can use the three.js editor to give it a try and download the project as an example or starting point.

Simple Boilerplate : spinning cube and orbit controls with damping

This is the basic HTML file that can be used as a boilerplate when starting a project. This

boilerplate uses orbit controls with damping (camera that can move around an object with deceleration effect) and creates a spinning cube.

```
<!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>
```

Hello world!

The example is taken from threejs website.

You may want to download three.js and change the script source below.

There are many more advanced examples under this link.

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>
```

The basic scene with a static cube in 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;
```

To actually see anything, we need a Render() loop:

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

Read Getting started with three.js online: https://riptutorial.com/three-js/topic/2102/getting-started-with-three-js

Chapter 2: Camera Controls in Three.js

Introduction

This document outlines how you can easily add some existing Camera Controls to your scene, as well as provide guidance on creating custom controls. Note, the pre-made control scripts can be found in the `/examples/js/controls` folder of the library.

Examples

Orbit Controls

An Orbit Camera is one that allows the user to rotate around a central point, but while keeping a particular axis locked. This is extremely popular because it prevents the scene from getting "tilted" off-axis. This version locks the Y (vertical) axis, and allows users to Orbit, Zoom, and Pan with the left, middle, and right mouse buttons (or specific touch events).

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);
}
```

The OrbitControls script has a several settings that can be modified. The code is well documented, so look in OrbitControls.js to see those. As an example, here is a code snippet showing a few of those being modified on a new OrbitControls object.

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

Custom Camera Control - Mouse-based Sliding

Here's an example of a custom camera controller. This reads the position of the mouse within the client window, and then slides the camera around as if it were following the mouse on the window.

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);
}
```

As you can see, here we are merely updating the Camera position during the rendering's `animate` phase, like we could for any object in the scene. In this case, we are simply re-positioning the camera at a point offset from it's original X and Y coordinates. This could just as easily be the X and Z coordinates, or a point along a path, or something completely different not even related to the mouse's position at all.

Read Camera Controls in Three.js online: https://riptutorial.com/three-js/topic/8270/camera-controls-in-three-js

Chapter 3: Geometries

Remarks

Examples work as of three.js R79 (revision 79).

Examples

THREE.BoxGeometry

THREE.BoxGeometry builds boxes such as cuboids and cubes.

Cubes

Cubes created using THREE.BoxGeometry would use the same length for all sides.

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();
```

Notice the 'render' function. This renders the cube 60 times a second.

Full Code (with 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>
```

Cuboids

The line `var geometry = new THREE.BoxGeometry( 1, 1, 1 );` gives us a cube. To make a cuboid, just change the parameters - they define the length, height and depth of the cube respectively.

Example:

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

More (proving the cube is three-dimensional)

The cube may seem to be just a square. To prove that it is, without doubt, three-dimensional, add the following lines of code to the 'render' function:

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

And watch as the merry cube spins round... and round... and round...

Colourful

Not for the faint-hearted...

The uniform colour for the entire cube is... green. Boring. To make each face a different colour, we've to dig to the geometry's faces.

```
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);
```

NOTE: The method of colouring the faces is not the best method, but it works well (enough).

Notes

Where's the `canvas` in the HTML document's body?

There is no need to add a `canvas` to the body manually. The following three lines

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

create the renderer, its `canvas` and add the `canvas` to the DOM.

THREE.CylinderGeometry

THREE.CylinderGeometry build cylinders.

Cylinder

Continuing from the previous example, the code to create the box could be replaced with the below.

```
//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 );
```

To build from scratch, here's the code.

```
//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();
```

More (proving the cylinder is three-

dimensional)

The cylinder may seem to be just... two-dimensional. To prove that it is, without doubt, three-dimensional, add the following lines of code to the 'render' function:

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

And the happy bright cylinder would spin randomly, amidst a dark, black background...

Read Geometries online: https://riptutorial.com/three-js/topic/5762/geometries

Chapter 4: Meshes

Introduction

A Three.js Mesh is a base class that inherits from Object3d and is used to instantiate polygonal objects by combining a Geometry with a Material. `Mesh` is also the base class for the more advanced `MorphAnimMesh` and `SkinnedMesh` classes.

Syntax

- new THREE.Mesh(geometry, material);

Remarks

Both the geometry and material are optional and will default to `BufferGeometry` and `MeshBasicMaterial` respectively if they are not provided in the constructor.

Examples

Render a cube mesh with a box geometry and a basic material

```
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();
```

Read Meshes online: https://riptutorial.com/three-js/topic/8838/meshes

Chapter 5: Object Picking

Examples

Object picking / Raycasting

Raycasting means throwing a ray from the mouse position on the screen to the scene, this is how threejs determines what object you want to click on if you have implemented it. Threejs gets that information using an octree, but still in production you may not want to compute the result at each frame or on the `mousemove` event, but rather on the `click` event for a more accessible app with low requirements.

```
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)
        */
    }

}
```

> CAUTION! You may lose your time gazing at the blank screen if you don't read the next part.

If you want to detect the light helper, set the second parameter of `raycaster.intersectObjects( scene.children );` to true.

It means `raycaster.intersectObjects( scene.children , true);`

The raycast code will only detect the light helper.

If you want it to detect normal objects as well as light helper, you need to copy the above raycast function again. See this question.

The full raycast code is

```
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)
        */
    }

}
```

Object Picking / GPU

Object picking using Raycasting might be a heavy task for your CPU depending on your setup (for example if you don't have an octree like setup) and number of objects in the scene.

If you don't need the world coordinates under the mouse cursor but only to identify the object under it you can use GPU picking.

Short explanation, GPU can be a powerful tool for computation but you need to know how to get the results back. The idea is, if you render the objects with a color that represents their id, you can read the color of the pixel under the cursor and findout the id of the object that is picked. Remember RGB is just a hex value so there is a conversion exists between id (integer) and color (hex).

1. Create a new scene and a new rendering target for your object

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

2. Create a new shader Material for object picking;

```
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. Add your mesh/line geometries a new attribute that represents their id in RGB, create the pickingObject using the same geometry and add it to the picking scene, and add the actual mesh to a id->object dictionary

```
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. Finally, on your mouse click handler

```
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
}
```

Read Object Picking online: https://riptutorial.com/three-js/topic/4848/object-picking

Chapter 6: Render Loops for Animation: Dynamically updating objects

Introduction

This document describes some common ways to add animation directly into your Three.js scenes. While there are libraries and frameworks that can add dynamic movement to your scene (tweens, physics, etc), it is helpful to understand how you can do this yourself simply with a few lines of code.

Remarks

The core concept of animation is updating an object's properties (rotation and translation, usually) in small amounts over a period of time. For example, if you translate an object by increasing the X position by 0.1 every tenth of a second, it will be 1 unit further on the X axis in 1 second, but the viewer will perceive it as having smoothly moved to that position over that time instead of jumping directly to the new position.

To assist us, we create a *render loop* in the script.

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

In the spinning cube example above, we use this idea - small incremental updates - to change the rotation of the cube every time a new frame of animation is requested. By incrementing the `rotation.x` and `rotation.y` properties of the `cube` object on every frame, the cube appears to spin on those two axes.

As another example, it's not uncommon to separate your needed update into other functions, where you can do additional calculations and checks while keeping the render loop uncluttered. For example, the render loop below calls four different update functions, each one intended to update a separate object (or an array of objects, in the case of `updatePoints()`) in the scene.

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

You may notice in examples online that the camera controls are also part of the render loop.

```
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);
};
```

This is because the script for controlling the camera is doing the same thing; updating it over time. The changes might be caused by user input such as a mouse position, or something programmatic like following a path. In either case though, we are just animating the camera as well.

Examples

Spinning Cube

```
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();
```

Read Render Loops for Animation: Dynamically updating objects online: https://riptutorial.com/three-js/topic/8271/render-loops-for-animation--dynamically-updating-objects

Chapter 7: Textures and Materials

Introduction

A nice introduction to material and textures.

Diffuse, Bump, Specular, and Transparent Textures.

Parameters

Parameter	Details
color	Numeric value of the RGB component of the color.
intensity	Numeric value of the light's strength/intensity.
fov	Camera frustum vertical field of view.
aspect	Camera frustum aspect ratio.
near	Camera frustum near plane.
far	Camera frustum far plane.
radius	sphere radius. Default is 50.
widthSegments	number of horizontal segments. Minimum value is 3, and the default is 8.
heightSegments	number of vertical segments. Minimum value is 2, and the default is 6.
phiStart	specify horizontal starting angle. Default is 0.
phiLength	specify horizontal sweep angle size. Default is Math.PI * 2.
thetaStart	specify vertical starting angle. Default is 0.
thetaLength	specify vertical sweep angle size. Default is Math.PI.

Remarks

Demo Link

Examples

Creating a Model Earth

Textures for this example are available at: http://planetpixelemporium.com/planets.html

Installation or Setup

You can install three via npm

```
npm install three
```

Or add it as a script to your HTML page

```
<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>
```

Creating the scene

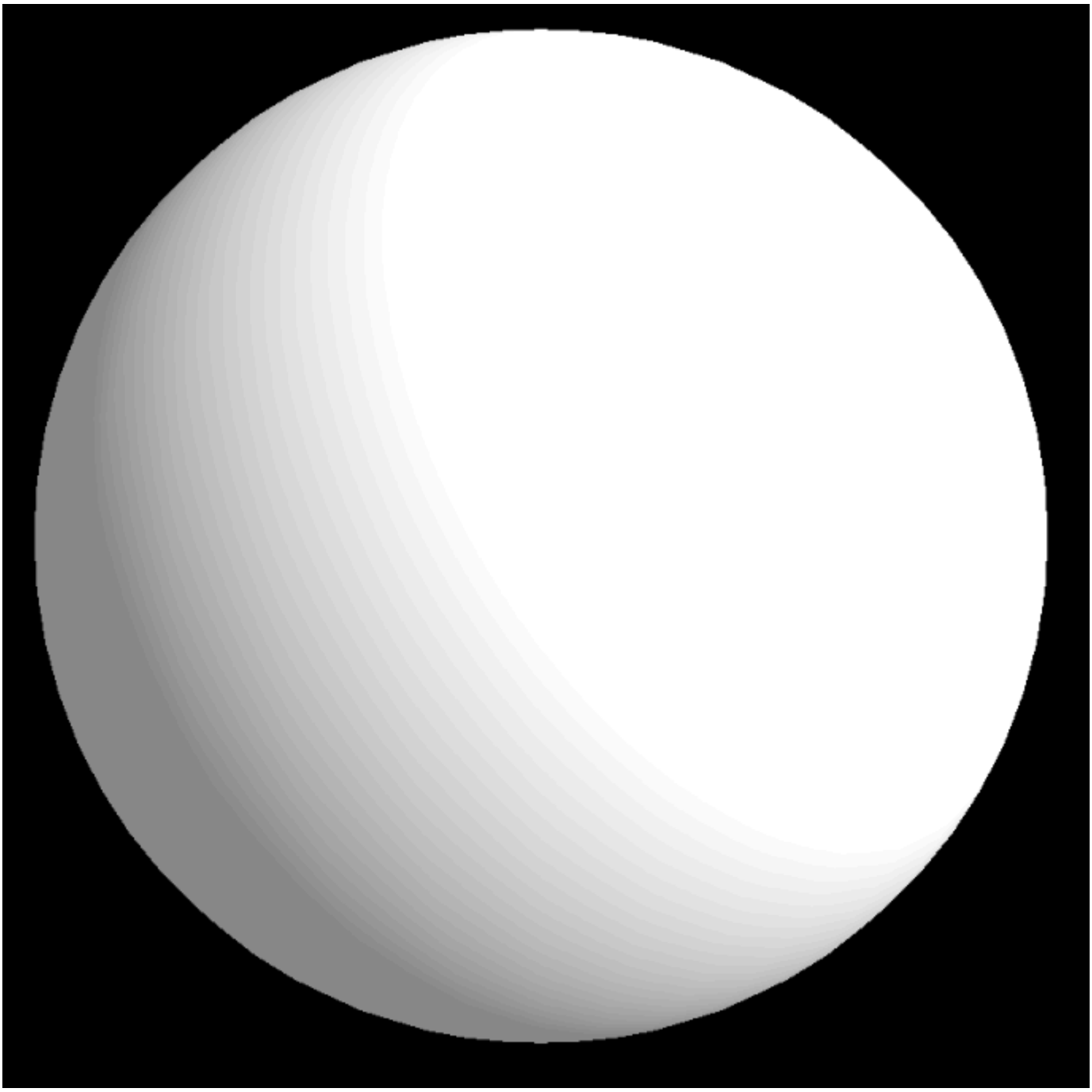
To actually be able to display anything with three.js, we need three things: A scene, a camera, and a renderer. We will render the scene with the camera.

```
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 );
```

Creating the Sphere

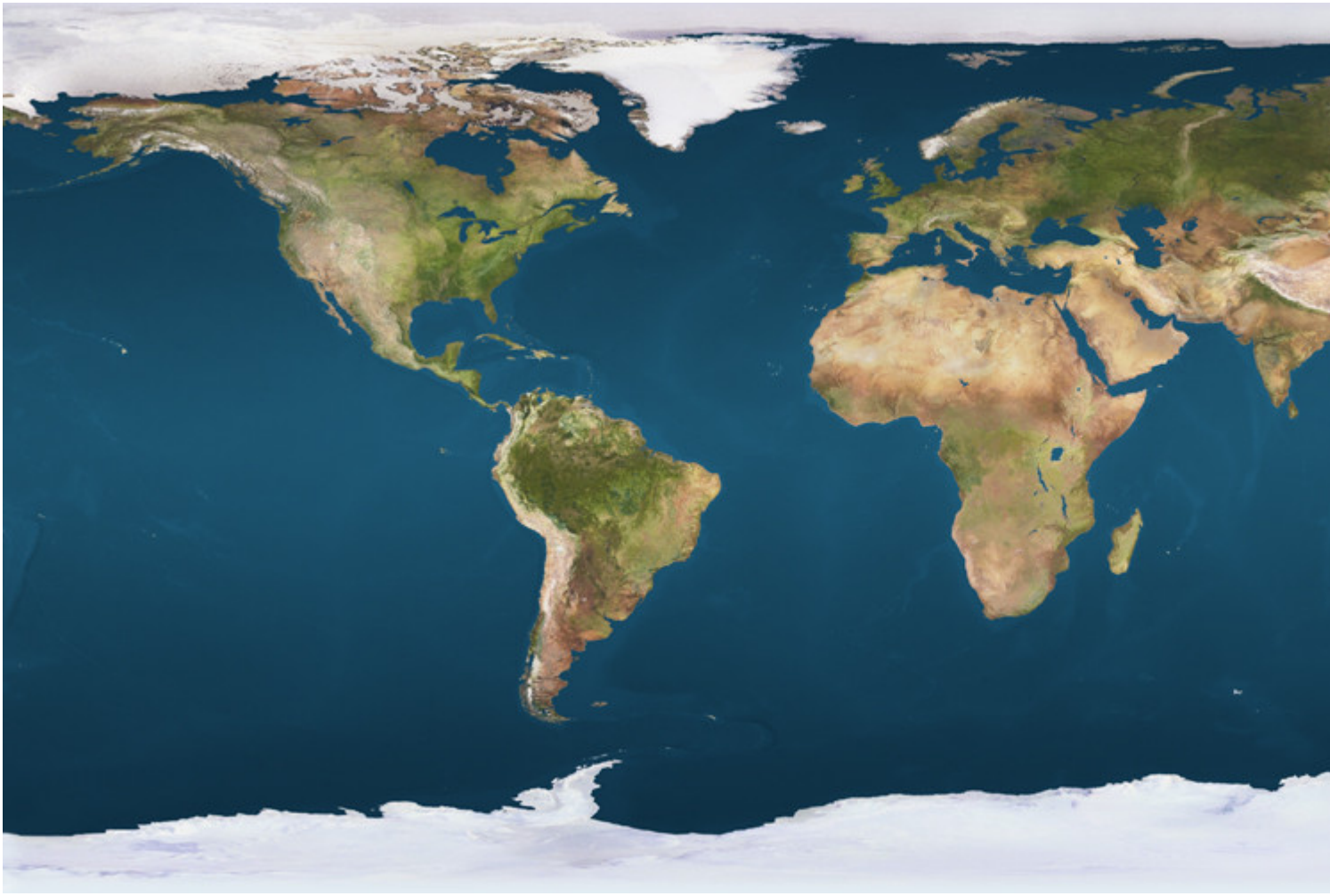
- Create geometry for the sphere
- Create a phong material
- Create a 3D Object
- Add it to the scene



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

Add a Diffuse Texture

The diffuse texture set the main color of the surface. When we apply it to a sphere, we get the following image.



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

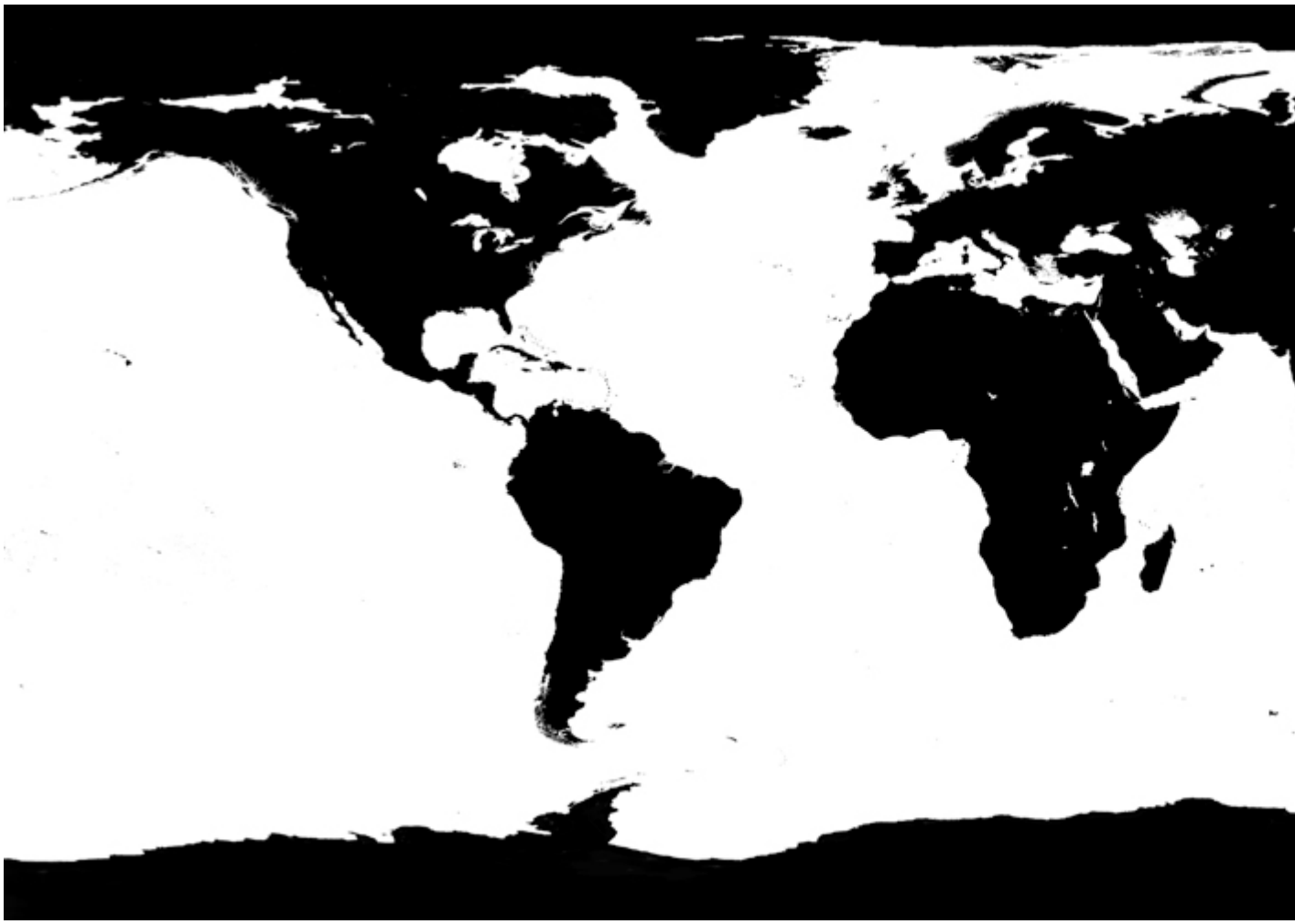
Adding a Bump Map Texture

- Each of its pixels acts as a height on the surface.
- The mountains appear more clearly thanks to their shadow.
- It is possible to change how much the map affects lighting with bumpScale parameter.
- No extra vertices are created or needed to use a bump map (unlike a displacement map)

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

Adding a Specular Texture

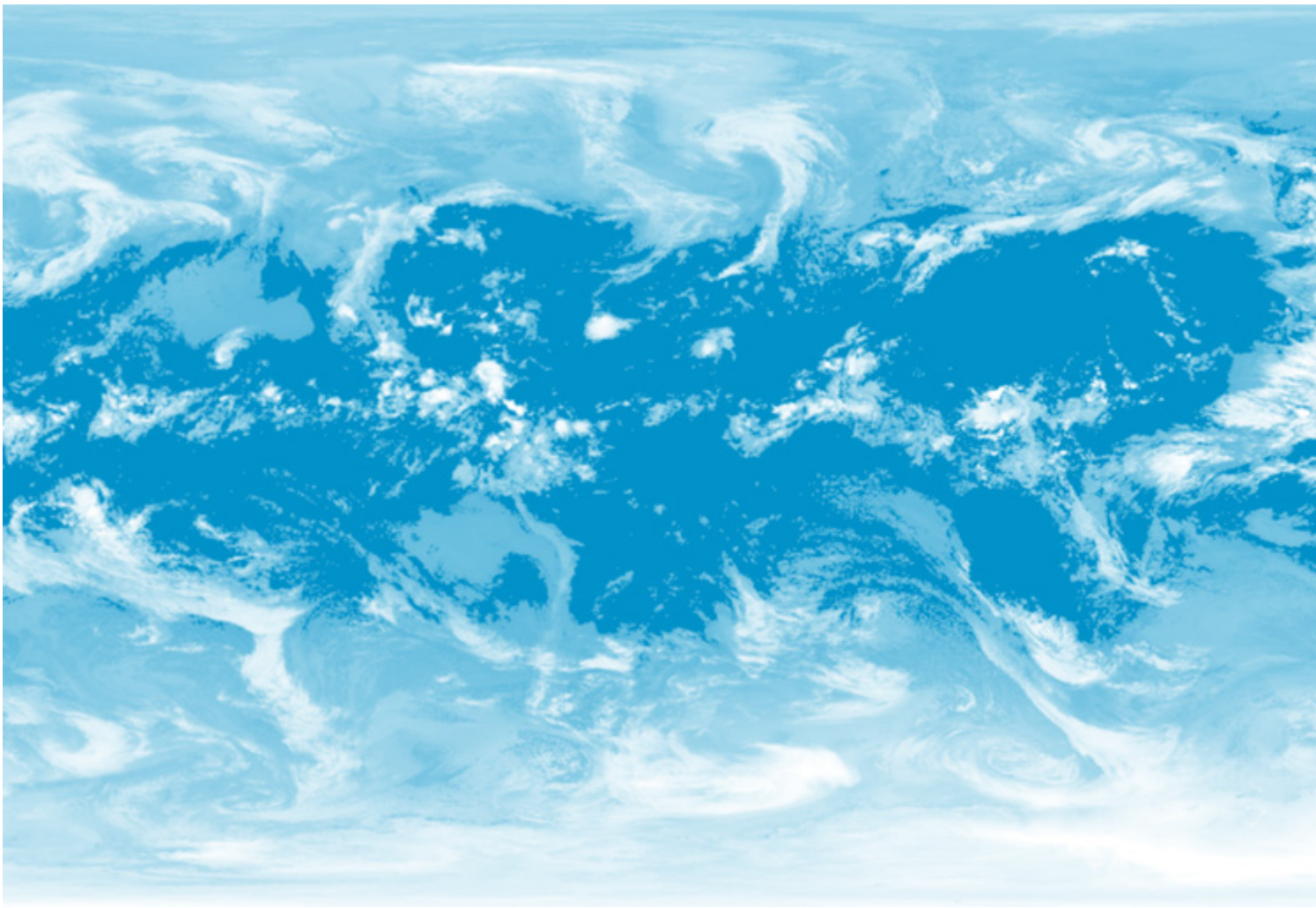
- Changes the 'shininess' of an object with a texture.
- Each pixel determines the intensity of specularity.
- In this case, only the sea is specular because water reflects light more than earth.
- You can control the specular color with the specular parameter.

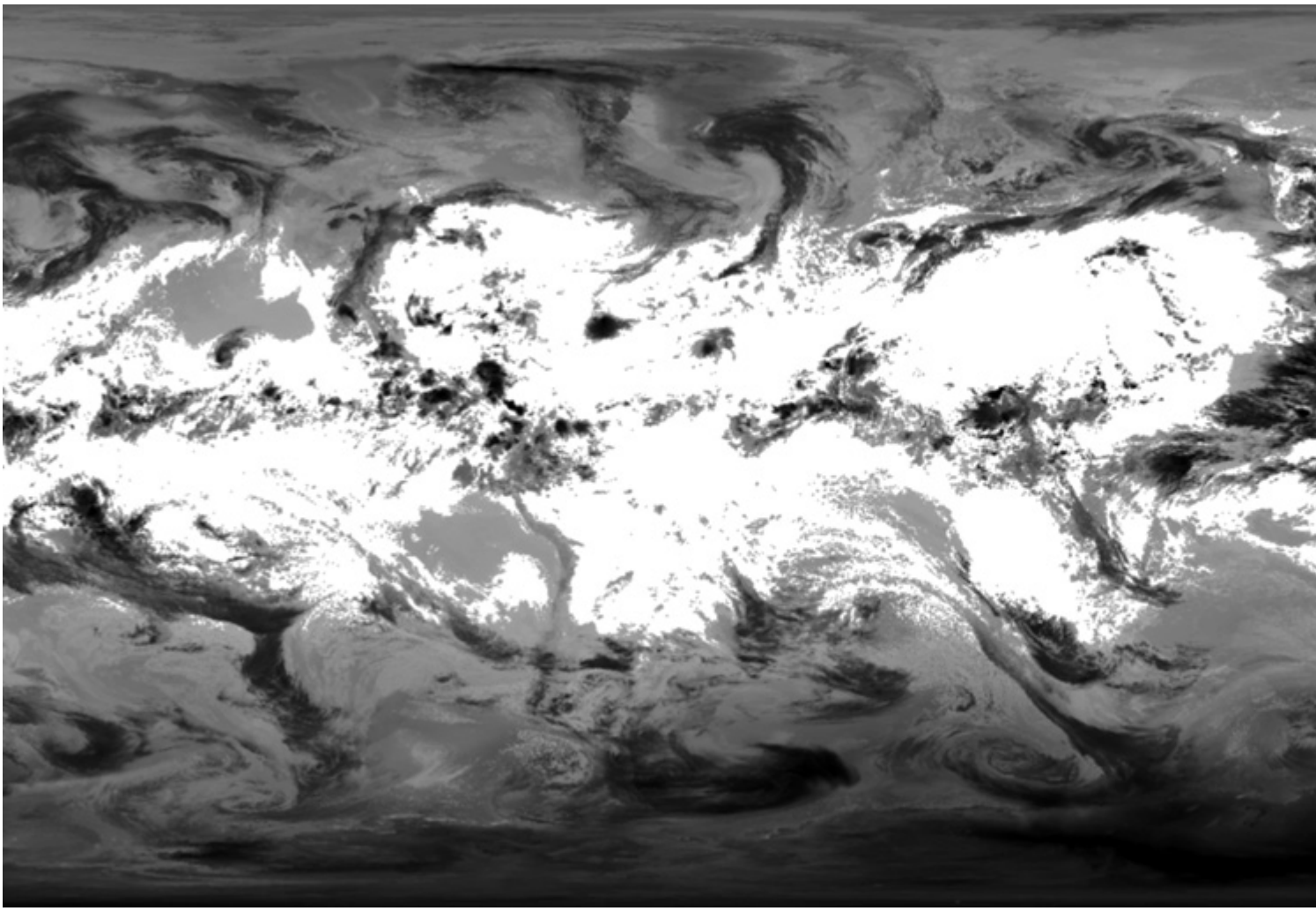


```
material.specularMap = THREE.ImageUtils.loadTexture('images/earthspec1k.jpg')
material.specular  = new THREE.Color('grey')
```

Adding a Cloud Layer

- We create `canvasCloud` with a canvas, and use it as a texture.
- We do this because jpg doesn't handle an alpha channel. (However, a PNG image does)
- We need to make the code to build the texture based on the following images.





```
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)
```

- We attach the `cloudMesh` to `earthMesh` so they will move together.
- We disable `depthWrite` and set `transparent: true` to tell three.js the cloudmesh is transparent.
- We set sides to `THREE.DoubleSide` so both sides will be visible.
    - This avoids creating artifacts on the edge of the earth.
- Finally, we set `opacity: 0.8` to make the clouds more translucent

Adding Rotational Movement

In your render loop, you simply increase the rotation

As a final touch, we will animate the cloud layer in order to make it look more realistic.

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

Read Textures and Materials online: https://riptutorial.com/three-js/topic/9333/textures-and-materials

Credits

S. No	Chapters	Contributors
1	Getting started with three.js	Atrahasis, Blogueira, Community, Gero3, guardabrazo, Hasan, Hectate, Joel Martinez, juagicre, Learn How To Be Transparent, Xander Luciano, Zeromatiker, zya
2	Camera Controls in Three.js	Hectate
3	Geometries	streppel, Theo
4	Meshes	Paul Graffam
5	Object Picking	Gero3, Kris Roofe, Learn How To Be Transparent, winseybash
6	Render Loops for Animation: Dynamically updating objects	Hectate
7	Textures and Materials	Geethu Jose, Xander Luciano