eBook Gratuit

APPRENEZ three.js

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#three.js

Table des matières

À propos

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

Chapitre 1: Démarrer avec three.js

Remarques

Le but du projet est de créer une bibliothèque 3D légère avec un très faible niveau de complexité, en d'autres termes, pour les nuls. La bibliothèque fournit des rendus de type canvas, svg, CSS3D et WebGL.

Versions

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

Examples

Installation ou configuration

- Vous pouvez installer three.js via npm:

```
npm install three
```

- Vous pouvez l'ajouter à partir d'un CDN à votre HTML:

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

- Vous pouvez utiliser l' éditeur three.js pour l'essayer et télécharger le projet comme exemple ou comme point de départ.

Boilerplate simple: contrôles de cube et d'orbite tournants avec

amortissement

C'est le fichier HTML de base qui peut être utilisé comme point de départ lors du démarrage d'un projet. Ce passe-partout utilise des contrôles d'orbite avec amortissement (caméra pouvant se déplacer autour d'un objet avec effet de décélération) et crée un cube en rotation.

```
<!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>
```

Bonjour le monde!

L'exemple est tiré du site Web de threejs .

Vous souhaiterez peut-être télécharger three.js et modifier la source du script ci-dessous.

Il existe de nombreux exemples plus avancés sous ce lien.

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 scène de base avec un cube statique 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;
```

Pour voir quelque chose, nous avons besoin d'une boucle Render ():

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

Lire Démarrer avec three.js en ligne: https://riptutorial.com/fr/three-js/topic/2102/demarrer-avec-three-js

Chapitre 2: Boucles de rendu pour l'animation: mise à jour dynamique des objets

Introduction

Ce document décrit des méthodes courantes pour ajouter une animation directement dans vos scènes Three.js. Bien qu'il existe des bibliothèques et des frameworks pouvant ajouter du mouvement dynamique à votre scène (tweens, physique, etc.), il est utile de comprendre comment vous pouvez le faire simplement avec quelques lignes de code.

Remarques

Le concept de base de l'animation consiste à mettre à jour les propriétés d'un objet (rotation et traduction, généralement) en petites quantités sur une période donnée. Par exemple, si vous traduisez un objet en augmentant la position X de 0,1 tous les dixièmes de seconde, il sera 1 unité plus loin sur l'axe des X en 1 seconde, mais le spectateur le percevra comme s'étant doucement déplacé vers cette position. temps au lieu de sauter directement à la nouvelle position.

Pour nous aider, nous créons une *boucle de rendu* dans le script.

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

Dans l'exemple ci-dessus du cube en rotation, nous utilisons cette idée - petites mises à jour incrémentielles - pour modifier la rotation du cube à chaque fois qu'une nouvelle image est demandée. En incrémentant les propriétés `rotation.x` et `rotation.y` de l'objet `cube` sur chaque image, le cube semble tourner sur ces deux axes.

Autre exemple, il n'est pas rare de séparer votre mise à jour nécessaire dans d'autres fonctions, où vous pouvez effectuer des calculs et des vérifications supplémentaires tout en gardant la boucle de rendu dégagée. Par exemple, la boucle de rendu ci-dessous appelle quatre fonctions de mise à jour différentes, chacune destinée à mettre à jour un objet distinct (ou un tableau d'objets, dans le cas de `updatePoints()`) dans la scène.

```
//render loop
function render() {
    requestAnimationFrame( render );
    updateGrid();
    updateCube();
    updateCamera();
    updatePoints(pList);
```

```
    renderer.render( scene, camera);
}
render();
```

Vous pouvez remarquer dans les exemples en ligne que les commandes de la caméra font également partie de la boucle de rendu.

```
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);
};
```

C'est parce que le script pour contrôler la caméra fait la même chose; le mettre à jour au fil du temps. Les modifications peuvent être provoquées par une entrée utilisateur telle qu'une position de la souris ou quelque chose de programmé comme le suivi d'un chemin. Dans les deux cas cependant, nous ne faisons qu'animer la caméra.

Examples

Cube Tournant

```
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();
```

Lire Boucles de rendu pour l'animation: mise à jour dynamique des objets en ligne: https://riptutorial.com/fr/three-js/topic/8271/boucles-de-rendu-pour-l-animation--mise-a-jour-dynamique-des-objets

Chapitre 3: Commandes de caméra dans Three.js

Introduction

Ce document explique comment ajouter facilement des contrôles de caméra existants à votre scène, ainsi que des conseils sur la création de contrôles personnalisés. Notez que les scripts de contrôle prédéfinis se trouvent dans le dossier `/examples/js/controls` de la bibliothèque.

Examples

Commandes Orbit

Une caméra orbitale est une caméra qui permet à l'utilisateur de tourner autour d'un point central tout en gardant un axe particulier verrouillé. Ceci est extrêmement populaire car il empêche la scène d'être "inclinée" hors axe. Cette version verrouille l'axe Y (vertical) et permet aux utilisateurs de mettre en orbite, de zoomer et de faire un panoramique avec les boutons gauche, central et droit de la souris (ou des événements tactiles spécifiques).

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);
}
```

Le script OrbitControls dispose de plusieurs paramètres pouvant être modifiés. Le code est bien documenté, alors regardez dans OrbitControls.js pour voir ceux-ci. Par exemple, voici un extrait de code montrant quelques-uns de ceux qui ont été modifiés sur un nouvel objet OrbitControls.

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

Custom Camera Control - Glissement basé sur la souris

Voici un exemple de contrôleur de caméra personnalisé. Ceci lit la position de la souris dans la fenêtre du client, puis fait glisser la caméra comme si elle suivait la souris sur la fenêtre.

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);
}
```

Comme vous pouvez le voir, nous ne faisons ici que mettre à jour la position de la caméra pendant la phase d' `animate` du rendu, comme nous pourrions le faire pour n'importe quel objet de la scène. Dans ce cas, nous repositionnons simplement la caméra à un point décalé de ses coordonnées X et Y d'origine. Cela pourrait être aussi facilement les coordonnées X et Z, ou un point le long d'un chemin, ou quelque chose de complètement différent, même sans rapport avec la position de la souris.

Lire Commandes de caméra dans Three.js en ligne: https://riptutorial.com/fr/three-js/topic/8270/commandes-de-camera-dans-three-js

Chapitre 4: Cueillette d'objets

Examples

Préparation d'objets / Raycasting

Raycasting signifie lancer un rayon depuis la position de la souris sur l'écran vers la scène. Voici comment troisjs détermine l'objet sur lequel vous souhaitez cliquer si vous l'avez implémenté. Threejs obtient ces informations en utilisant un octree , mais en production, vous ne voudrez peut-être pas calculer le résultat à chaque image ou sur l'événement `mousemove` , mais plutôt sur l'événement `click` pour une application plus accessible avec des exigences peu élevées.

```
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)
        */
    }

}
```

MISE EN GARDE! Vous risquez de perdre votre temps à regarder l'écran vide si vous ne lisez pas la partie suivante.

Si vous souhaitez détecter l'assistant, définissez le deuxième paramètre de `raycaster.intersectObjects( scene.children );` à vrai.

Cela signifie `raycaster.intersectObjects( scene.children , true);`

Le code raycast ne détectera que l'assistant.

Si vous souhaitez qu'il détecte les objets normaux ainsi que les aides à la lumière, vous devez copier à nouveau la fonction Raycast ci-dessus. Voir cette question .

Le code raycast complet est

```
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)
        */
    }

}
```

Cueillette d'objets / GPU

La sélection d'objets à l'aide de Raycasting peut être une tâche lourde pour votre CPU en fonction de votre configuration (par exemple, si vous n'avez pas de configuration similaire à l'octree) et du nombre d'objets dans la scène.

Si vous n'avez pas besoin des coordonnées du monde sous le curseur de la souris, mais uniquement pour identifier l'objet sous celui-ci, vous pouvez utiliser la sélection GPU.

Brève explication, GPU peut être un outil puissant pour le calcul mais vous devez savoir comment récupérer les résultats. L'idée est que si vous restituez les objets avec une couleur qui représente leur identifiant, vous pouvez lire la couleur du pixel sous le curseur et trouver l'identifiant de l'objet qui est sélectionné. Rappelez-vous que RVB est juste une valeur hexadécimale donc il existe une conversion entre id (entier) et couleur (hex).

1. Créer une nouvelle scène et une nouvelle cible de rendu pour votre objet

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

2. Créer un nouveau matériau Matériau pour la sélection d'objets;

```
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. Ajoutez vos géométries maillage / ligne à un nouvel attribut qui représente leur identifiant en RVB, créez l'objet pickingObject en utilisant la même géométrie et ajoutez-le à la scène de sélection, puis ajoutez le maillage réel à un dictionnaire d'objets 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. Enfin, sur votre souris, cliquez sur le gestionnaire

```
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
}
```

Lire Cueillette d'objets en ligne: https://riptutorial.com/fr/three-js/topic/4848/cueillette-d-objets

Chapitre 5: Géométries

Remarques

Les exemples fonctionnent à partir de trois.js R79 (révision 79).

Examples

THREE.BoxGeometry

THREE.BoxGeometry construit des boîtes telles que cuboids et cubes.

Cubes

Les cubes créés avec THREE.BoxGeometry utiliseraient la même longueur pour tous les côtés.

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();
```

Notez la fonction "render". Cela rend le cube 60 fois par seconde.

Code complet (avec 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>
```

Cuboïdes

La ligne `var geometry = new THREE.BoxGeometry( 1, 1, 1 );` nous donne un cube. Pour créer un cuboïde, changez simplement les paramètres - ils définissent respectivement la longueur, la hauteur et la profondeur du cube.

Exemple:

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

Plus (prouver que le cube est en trois dimensions)

Le cube peut sembler être juste un carré. Pour prouver que c'est sans doute en trois dimensions, ajoutez les lignes de code suivantes à la fonction "render":

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

Et regardez comme le cube joyeux tourne rond… et rond… et rond…

Coloré

Pas pour les âmes sensibles...

La couleur uniforme du cube entier est ... verte. Ennuyeuse. Pour que chaque visage ait une couleur différente, nous devons creuser les faces de la géométrie.

```
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);
```

REMARQUE: La méthode de coloration des visages n'est pas la meilleure méthode, mais elle fonctionne bien (assez).

Remarques

Où est la `canvas` dans le corps du document HTML?

Il n'est pas nécessaire d'ajouter un `canvas` au corps manuellement. Les trois lignes suivantes

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

créer le rendu, son `canvas` et ajouter le `canvas` au DOM.

THREE.CylinderGeometry

THREE.CylinderGeometry construit des cylindres.

Cylindre

Dans l'exemple précédent, le code pour créer la boîte pourrait être remplacé par le code ci-dessous.

```
//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 );
```

Pour construire à partir de zéro, voici le 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();
```

Plus (prouver que le cylindre est en trois dimensions)

Le cylindre peut sembler être juste ... bidimensionnel. Pour prouver que c'est sans doute en trois dimensions, ajoutez les lignes de code suivantes à la fonction "render":

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

Et le cylindre brillant et joyeux tournait au hasard, au milieu d'un fond noir et sombre ...

Lire Géométries en ligne: https://riptutorial.com/fr/three-js/topic/5762/geometries

Chapitre 6: Meshes

Introduction

Un maillage Three.js est une classe de base qui hérite d' Object3d et permet d'instancier des objets polygonaux en combinant une géométrie avec un matériau . `Mesh` est également la classe de base pour les classes `MorphAnimMesh` et `SkinnedMesh` plus avancées.

Syntaxe

- nouveau TROIS.Mesh (géométrie, matériau);

Remarques

La géométrie et le matériau sont tous deux facultatifs et par défaut, `BufferGeometry` `MeshBasicMaterial` respectivement `BufferGeometry` et `MeshBasicMaterial` s'ils ne sont pas fournis dans le constructeur.

Examples

Rendu un maillage de cube avec une géométrie de boîte et un matériau de base

```
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();
```

Lire Meshes en ligne: https://riptutorial.com/fr/three-js/topic/8838/meshes

Chapitre 7: Textures et Matériaux

Introduction

Une belle introduction à la matière et aux textures.

Textures diffuses, bump, spéculaires et transparentes.

Paramètres

Paramètre	Détails
Couleur	Valeur numérique de la composante RVB de la couleur.
intensité	Valeur numérique de l'intensité / intensité de la lumière.
fov	Champ de vision vertical de la caméra
aspect	Rapport d'aspect du caméscope
près	Caméra frustum près de l'avion.
loin	Avion lointain de l'appareil photo.
rayon	rayon de la sphère. La valeur par défaut est 50
widthSegments	nombre de segments horizontaux. La valeur minimale est 3 et la valeur par défaut est 8.
heightSegments	nombre de segments verticaux. La valeur minimale est 2 et la valeur par défaut est 6.
phiStart	spécifier l'angle de départ horizontal. La valeur par défaut est 0.
longueur de phi	spécifiez la taille de l'angle de balayage horizontal. La valeur par défaut est Math.PI * 2.
thetaStart	spécifier l'angle de départ vertical. La valeur par défaut est 0.
thetaLength	spécifiez la taille de l'angle de balayage vertical. La valeur par défaut est Math.PI.

Remarques

Lien de démonstration

Examples

Créer une Terre Modèle

Les textures de cet exemple sont disponibles sur: http://planetpixelemporium.com/planets.html

Installation ou configuration

Vous pouvez installer trois via npm

```
npm install three
```

Ou ajoutez-le en tant que script à votre page 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>
```

Créer la scène

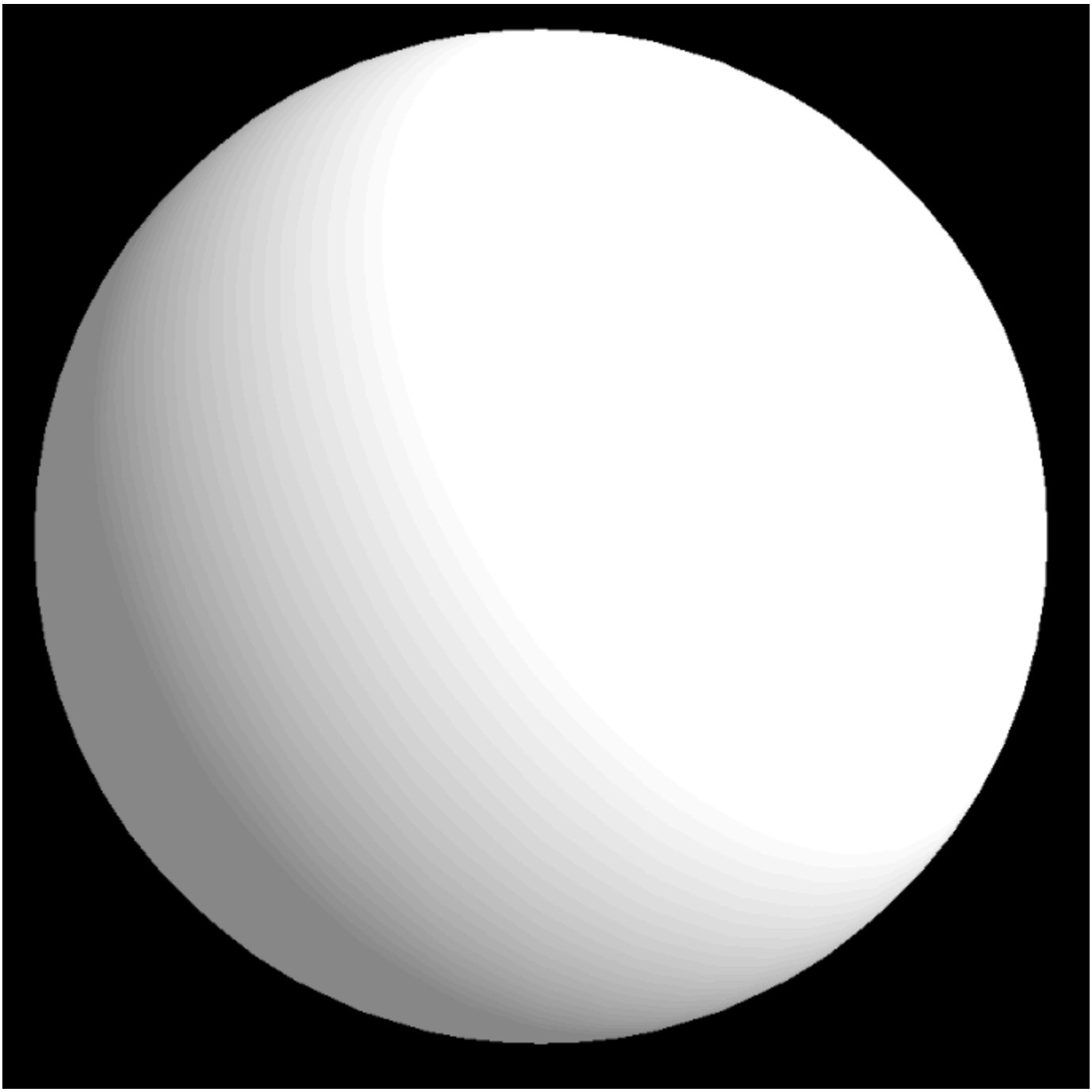
Pour pouvoir afficher quelque chose avec three.js, nous avons besoin de trois choses: une scène, une caméra et un rendu. Nous allons rendre la scène avec la caméra.

```
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 );
```

Créer la sphère

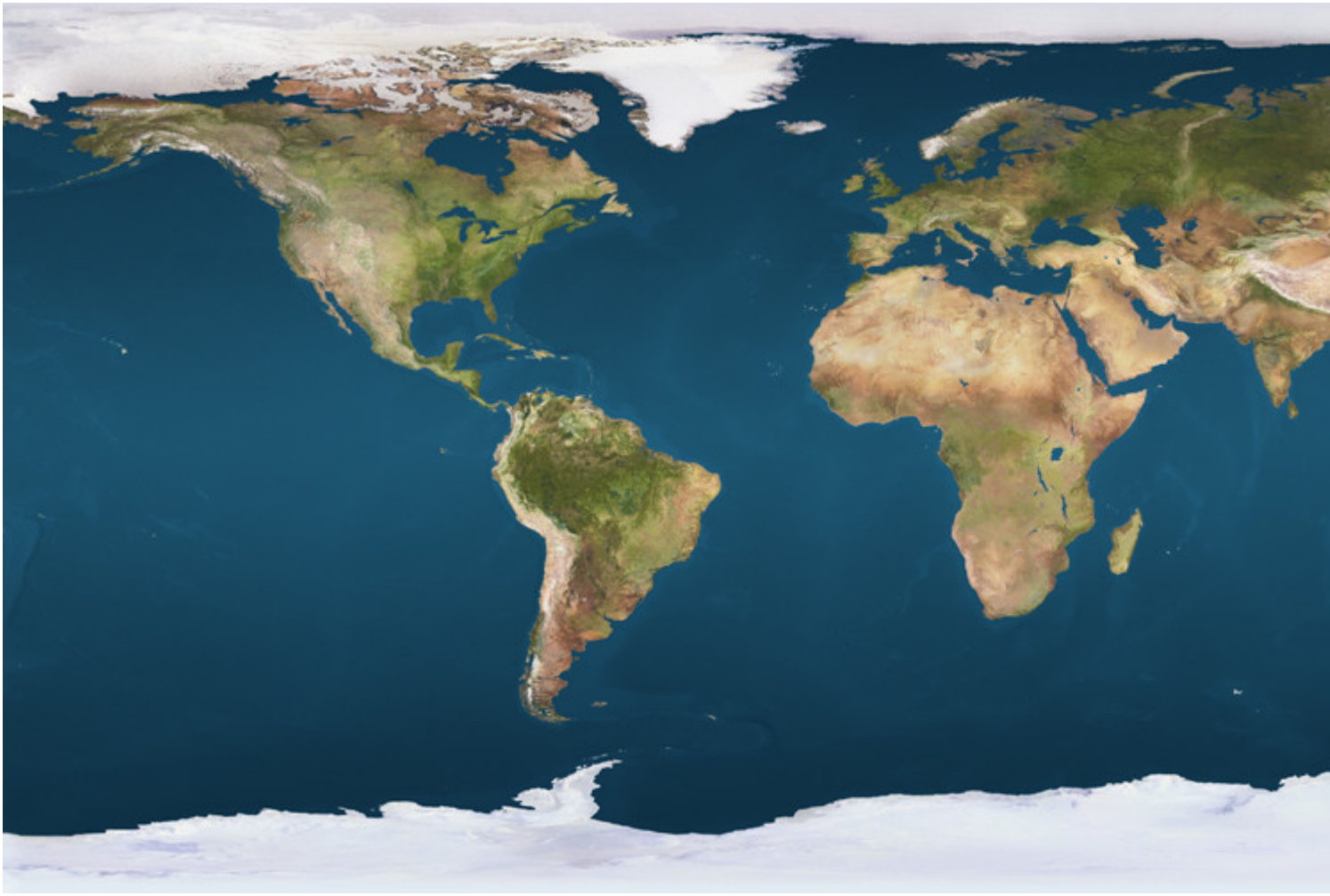
- Créer une géométrie pour la sphère
- Créer un matériau phong
- Créer un objet 3D
- Ajoutez-le à la scène



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

Ajouter une texture diffuse

La texture diffuse définit la couleur principale de la surface. Lorsque nous l'appliquons à une sphère, nous obtenons l'image suivante.



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

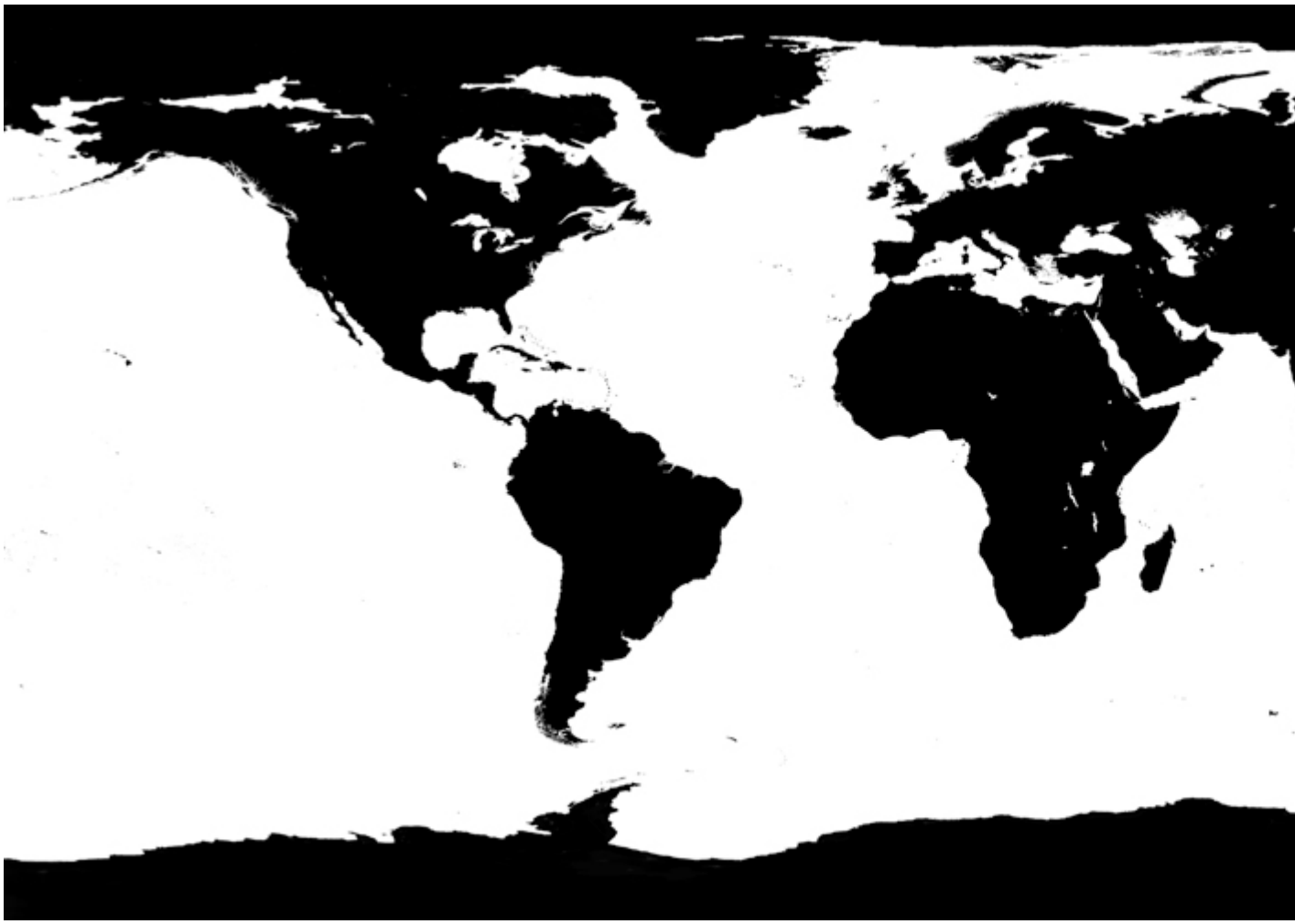
Ajout d'une texture de carte de relief

- Chacun de ses pixels agit comme une hauteur sur la surface.
- Les montagnes apparaissent plus clairement grâce à leur ombre.
- Il est possible de changer le degré d'effet de la carte sur l'éclairage avec le paramètre bumpScale.
- Aucun sommet supplémentaire n'est créé ou nécessaire pour utiliser une carte de relief (contrairement à une carte de déplacement)

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

Ajouter une texture spéculaire

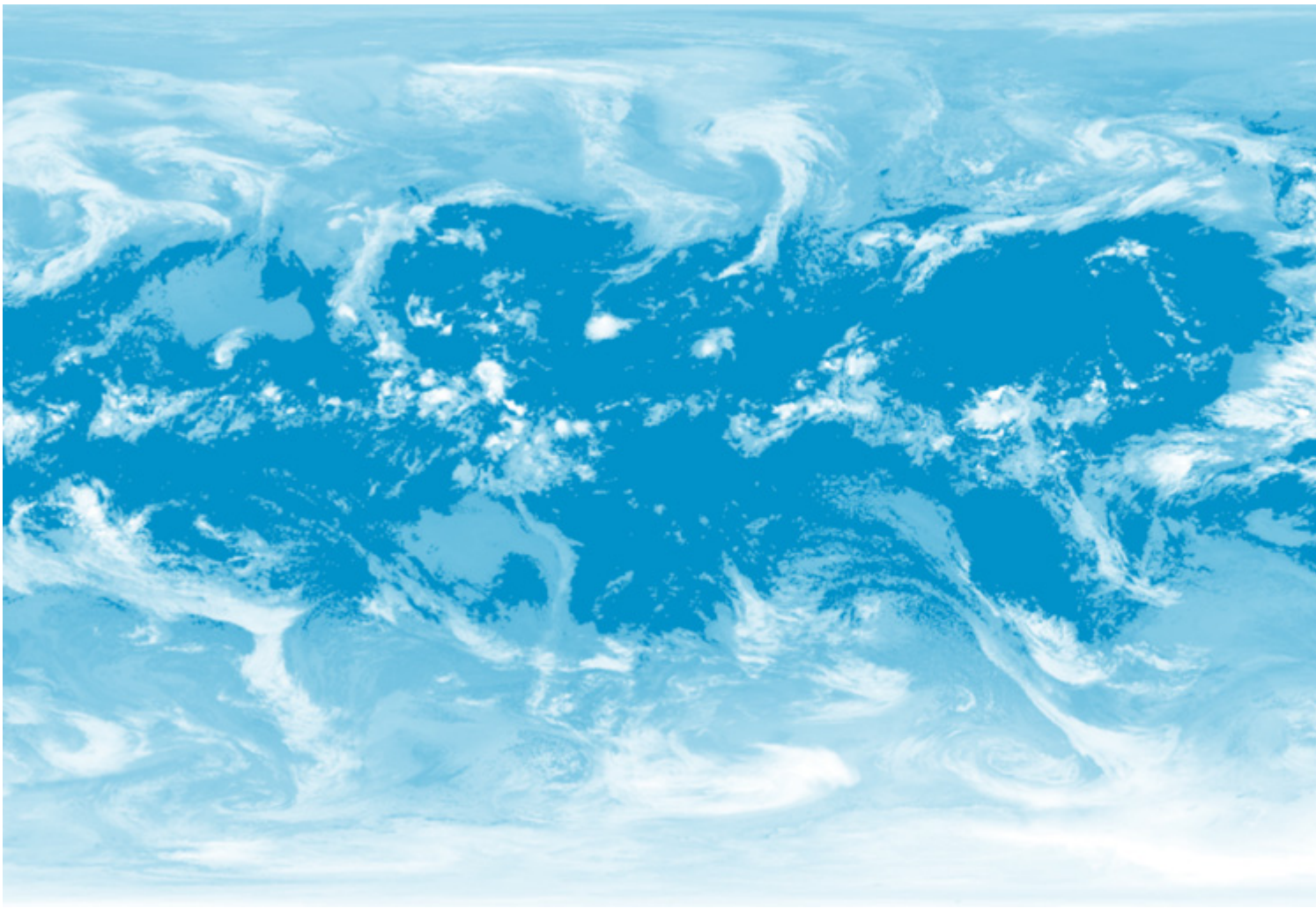
- Change la «brillance» d'un objet avec une texture.
- Chaque pixel détermine l'intensité de la spécularité.
- Dans ce cas, seule la mer est spéculaire car l’eau reflète la lumière plus que la terre.
- Vous pouvez contrôler la couleur spéculaire avec le paramètre spéculaire.

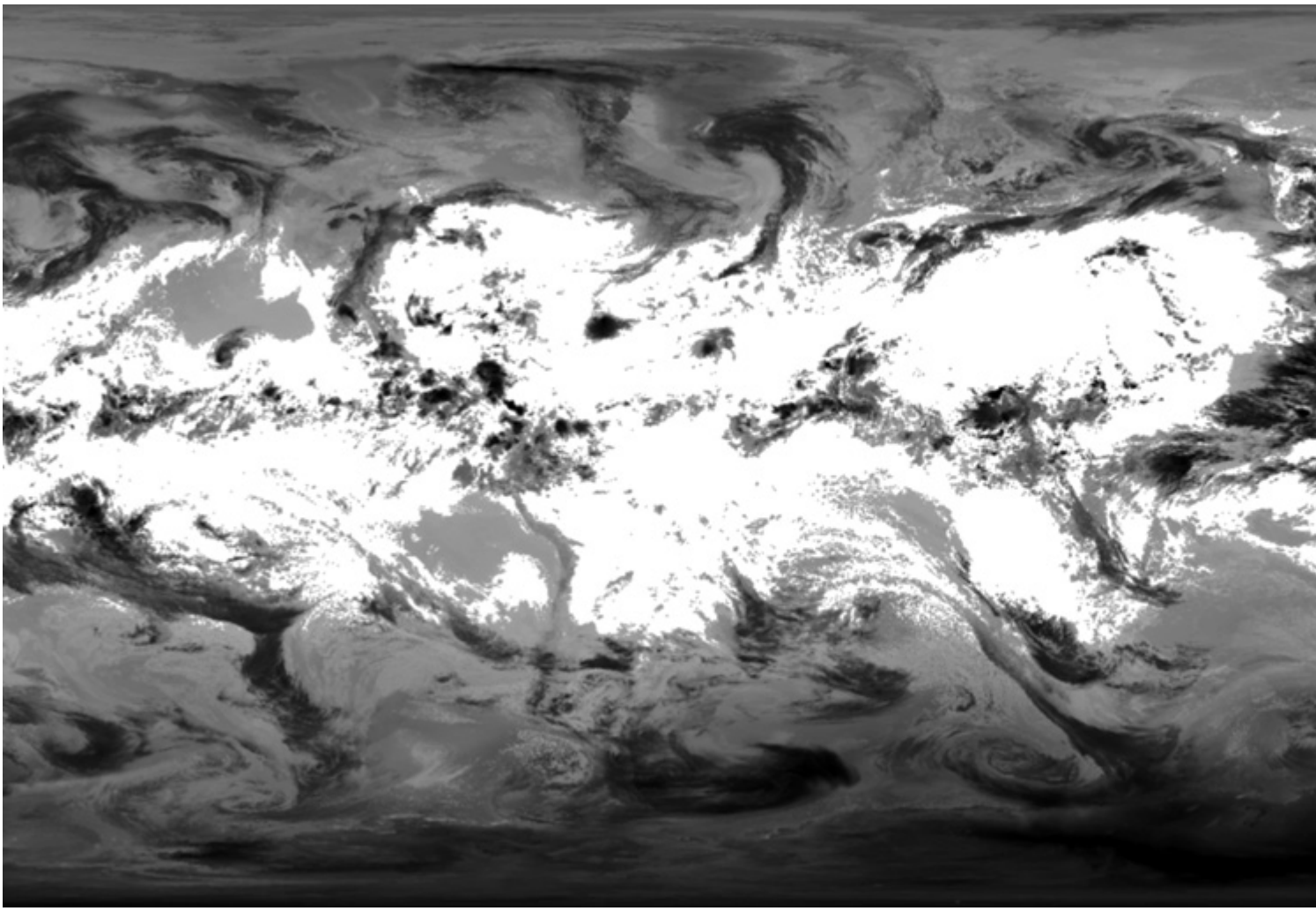


```
material.specularMap = THREE.ImageUtils.loadTexture('images/earthspec1k.jpg')
material.specular  = new THREE.Color('grey')
```

Ajouter une couche Cloud

- Nous créons `canvasCloud` avec un canevas et l'utilisons comme texture.
- Nous faisons cela parce que jpg ne gère pas un canal alpha. (Cependant, une image PNG fait)
- Nous devons créer le code pour créer la texture en fonction des images suivantes.





```
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)
```

- Nous attachons le `cloudMesh` à `earthMesh` afin qu'ils se déplacent ensemble.
- Nous désactivons `depthWrite` et définissons `transparent: true` pour dire three.js le cloudmesh est transparent.
- Nous définissons les côtés à `THREE.DoubleSide` . `THREE.DoubleSide` afin que les deux côtés soient visibles.
    - Cela évite de créer des artefacts sur le bord de la terre.
- Enfin, nous définissons l' `opacity: 0.8` pour rendre les nuages plus translucides

Ajout d'un mouvement de rotation

Dans votre boucle de rendu, vous augmentez simplement la rotation

Pour finir, nous allons animer la couche nuage afin de la rendre plus réaliste.

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

Lire Textures et Matériaux en ligne: https://riptutorial.com/fr/three-js/topic/9333/textures-et-materiaux

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec three.js	Atrahasis, Blogueira, Community, Gero3, guardabrazo, Hasan, Hectate, Joel Martinez, juagicre, Learn How To Be Transparent, Xander Luciano, Zeromatiker, zya
2	Boucles de rendu pour l'animation: mise à jour dynamique des objets	Hectate
3	Commandes de caméra dans Three.js	Hectate
4	Cueillette d'objets	Gero3, Kris Roofe, Learn How To Be Transparent, winseybash
5	Géométries	streppel, Theo
6	Meshes	Paul Graffam
7	Textures et Matériaux	Geethu Jose, Xander Luciano