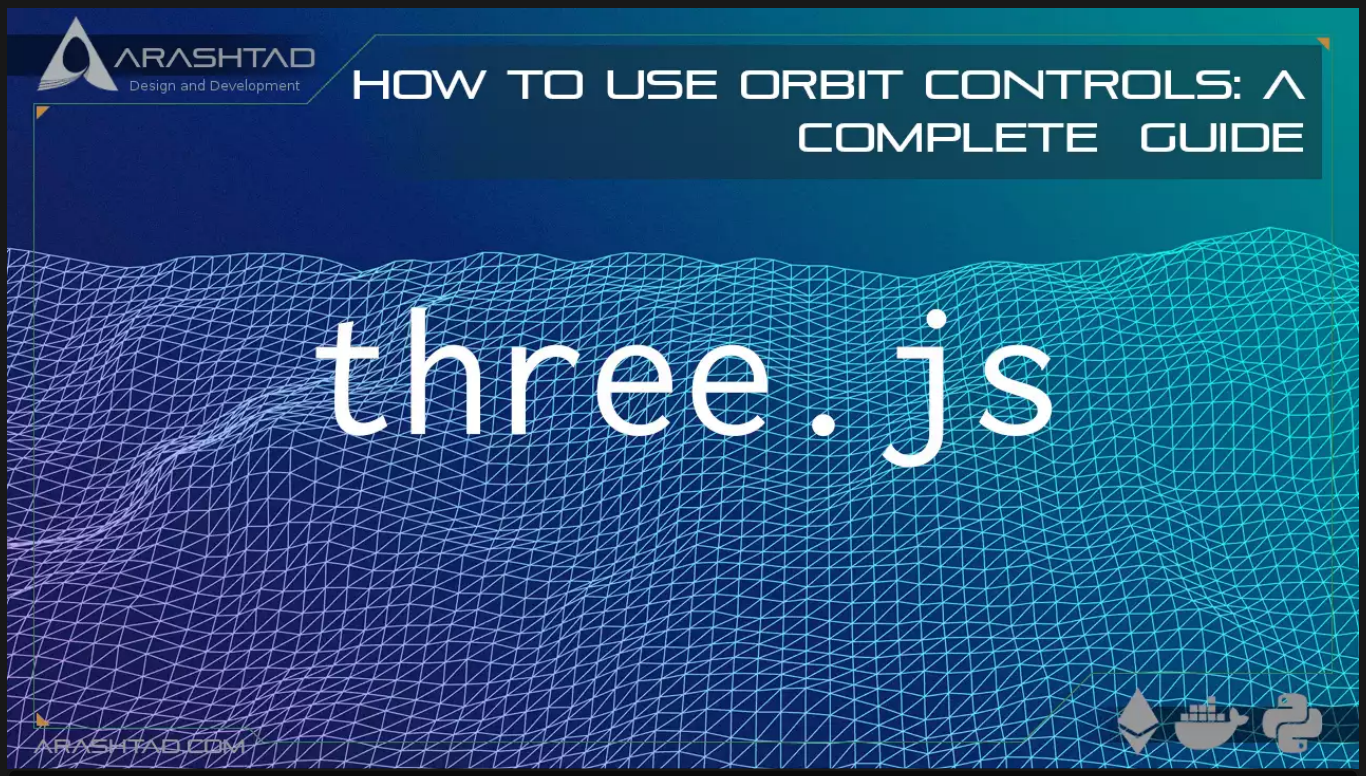
How to Use Orbit Controls: A Complete Guide

No comments



In Three JS, we have many different ways of rendering objects. One of them is orbit controls, When we use orbit controls, we want to have a 360-degree view of the object; meaning that we want to move around the object with complete freedom. And when we use orbit controls, we can use many other animations for the object and these two won't interfere with each other. Suppose you want to animate a rotating sphere, which simulates the globe (planet earth) and at the same time, you want to give the user the capability to move around the object in 360 degrees, Orbit control is the way to do it. In this tutorial, we will get started with the Vite plugin and then enter our boilerplate code. Afterward, we will modify the code and add the orbit controls to our rendering, so that the user can have the freedom to rotate around the object that is also rotating.

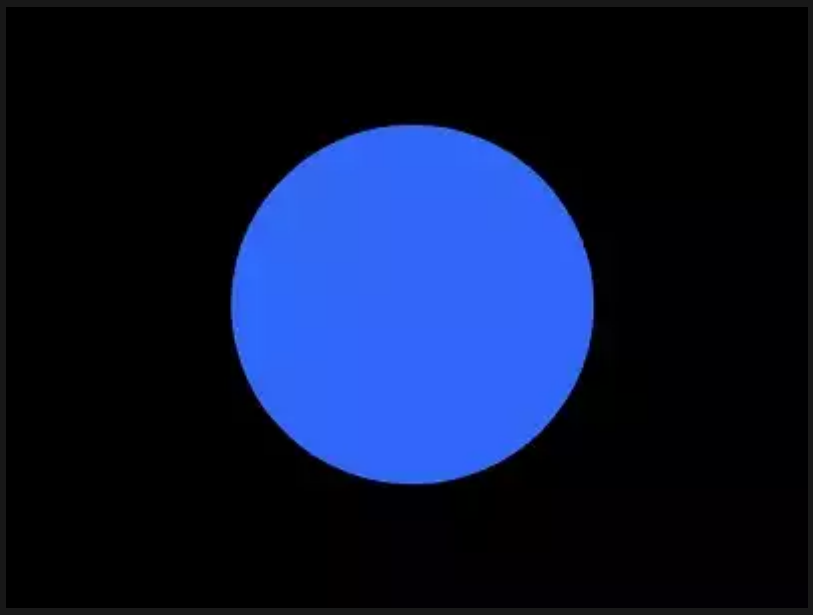
Getting started with the basics:

We are going to get started with the simple elements of a Three js scene including the camera, the renderer, the scene, the object and the light source (if necessary). Before we do that, we'd rather use the Vite plugin to be able to easily create all the folders and file that you need to run the Three.js code. First off, create a folder in the directory of your

projects by using the following commands: `mkdir Textures` `cd OrbitControls` Then, inside of the your project folder, create the necessary files and folders by simply running the Vite plugin command: `npm create vite@latest` Then enter the name of the project. You can write the name of your project as the name. And also the package (the name is arbitrary and you can choose anything that you want). Then select vanilla as the framework and variant. After that, enter the following commands in the terminal: `cd OrbitControls` `npm install` Afterwards, you can enter the javascript code that you want to write in the main.js file. So, we will enter the base or template code for running every project with an animating object, such as an sphere. Also do not forget to install Three.js package library every time create a project: `npm install three` Now, enter the following script in the main.js file:

```
import * as THREE from 'three';
import { Mesh } from 'three';
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(75
, innerWidth / innerHeight , 0.1, 1000);
const renderer = new THREE.WebGLRenderer({
     antialias : true
});
renderer.setSize(innerWidth, innerHeight);
document.body.appendChild(renderer.domElement);
//creating a sphere
const geometry = new THREE.SphereGeometry(5, 50, 50);
const material = new THREE.MeshBasicMaterial({
     color:0x3268F8
});
const sphere = new THREE.Mesh(geometry,material);
scene.add(sphere);
camera.position.z = 15;
function animate(){
     requestAnimationFrame(animate);
     renderer.render(scene,camera);
     sphere.rotation.y += 0.003;
}
animate();
```

The above code can be used as a boilerplate for later projects. The output of this code will be blue sphere like the below photo. But to be able to show that, you should write the following command in the terminal: `npm run dev`



What is orbit control in 3 js:

In Three JS, we use orbit controls for showing a 360-degree view of the scene. The The 360-degree view is very popular among designers since many objects and scenes are represented this way, disregarding the fact that the object or the camera may need to be rotated or have a certain kind of animation. In the next part of this article, we will write the modify the code of the boilerplate in a way that we have orbit controls in addition to the rotation animation.

Three.js texture scripts:

The first thing we do is to import the libraries of Three js as we have done it in the boilerplate. Afterward, we will import the {OrbitControls} and Stats modules from their directory at node modules. Notice that if you are not working with the Vite plugin, the directory might be in a different address. The below script is the modified version of the boilerplate code. Notice that in order to clearly see if the orbit control is working, we have added a Polka dot texture like below. You can add any other texture of your choice to make sure you can turn around the object and see everything from all aspects. Now let's see what the modified code looks like:

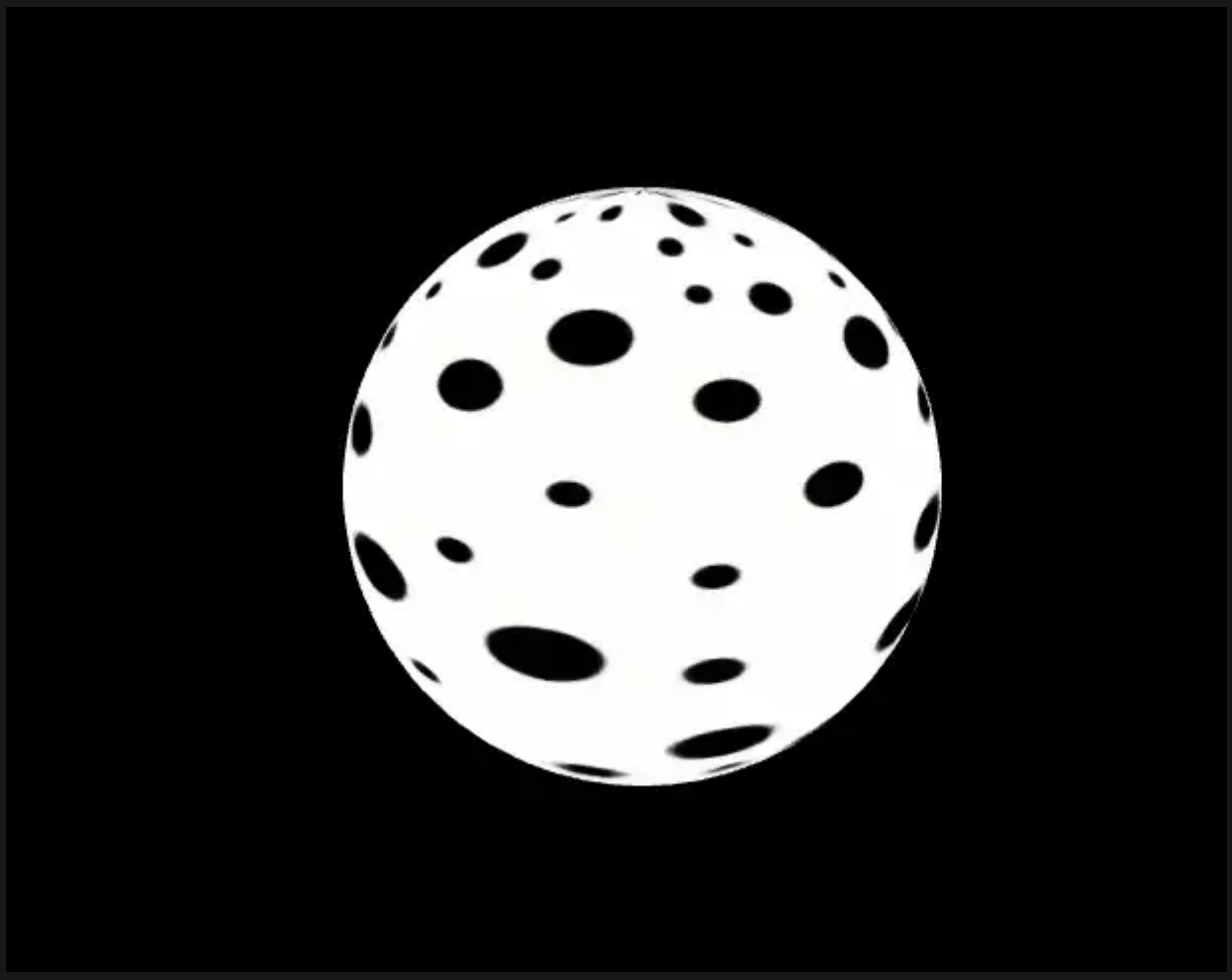
```
import * as THREE from '/node_modules/three/build/three.module.js';
import { Mesh } from 'three';
import {OrbitControls} from
"/node_modules/three/examples/jsm/controls/OrbitControls.js";
import Stats from
"/node_modules/three/examples/jsm/libs/stats.module.js";

const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(75
,innerWidth / innerHeight, 0.1, 1000);
const renderer = new THREE.WebGLRenderer({
     antialias : true
});
```

```
renderer.setSize(innerWidth, innerHeight);
document.body.appendChild(renderer.domElement);
const controls = new OrbitControls(camera, renderer.domElement);
//creating a sphere
const geometry = new THREE.SphereGeometry(5, 50, 50);
const material = new THREE.MeshBasicMaterial({
     map : new THREE.TextureLoader().load('./img/polkadot.jpg');
})
const sphere = new THREE.Mesh(geometry, material);
scene.add(sphere);
camera.position.z = 15;
window.addEventListener('resize', onWindowResize, false);
function onWindowResize(){
     camera.aspect = window.innerWidth / window.innerHeight;
     camera.updateProjectionMatrix();
     renderer.setSize(window.innerWidth, window.innerHeight);
     render();
}
const stats = Stats();
document.body.appendChild(stats.dom);
function animate() {
     requestAnimationFrame(animate);
     render();
     stats.update();
     sphere.rotation.y += 0.008;
}

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

Once we save the code, we will see a result like this:



You will also see that not only is the ball rotating, but you can also have a 360-degree view of the object as well. If you want to use texture, make sure you add it in the same directory and name as mentioned in the code. Otherwise, change the directory and the name.

```
map : new THREE.TextureLoader().load('./img/polkadot.jpg');
```

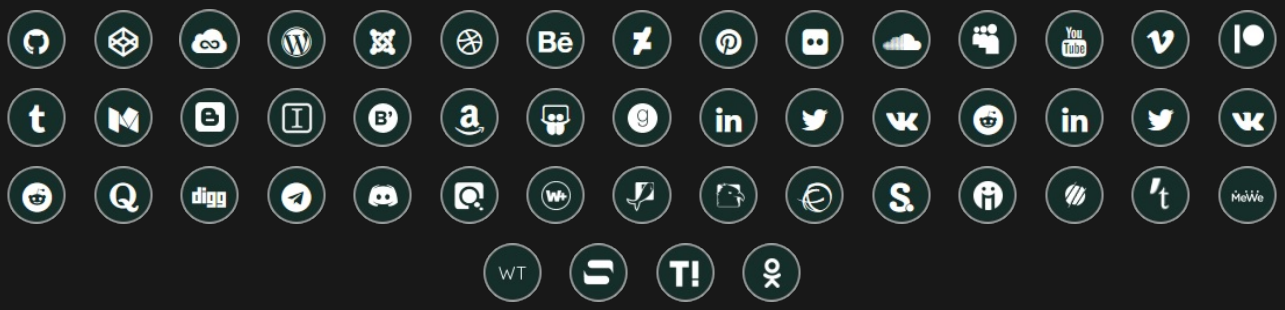
Conclusion

In this article, we have covered how we can use orbit controls to have 360-degree view of the object. We also saw that using this tool, we can still have our own animations such as the rotation of the object or the light source. Moreover, we learned about the code behind orbit controls, how to write it and what libraries you need to import in order to use the orbit controls tool. The example that we covered was creating a polka dot sphere which is rotating around itself and it is possible to turn around the object and look at it from all aspects. There are controls similar to orbit controls such as track bar control. But the main tool which enables you to have 360-degree freedom of view is the orbit controls.

Join Arashtad Community

Follow Arashtad on Social Media

We provide variety of content, products, services, tools, tutorials, etc. Each social profile according to its features and purpose can cover only one or few parts of our updates. We can not upload our videos on SoundCloud or provide our eBooks on Youtube. So, for not missing any high quality original content that we provide on various social networks, make sure you follow us on as many social networks as you're active in. You can find out Arashtad's profiles on different social media services.



Get Even Closer!

Did you know that only one universal Arashtad account makes you able to log into all Arashtad network at once? Creating an Arashtad account is free. Why not to try it? Also, we have regular updates on our newsletter and feed entries. Use all these benefitial free features to get more involved with the community and enjoy the many products, services, tools, tutorials, etc. that we provide frequently.

SIGN UP | NEWSLETTER | RSS FEED