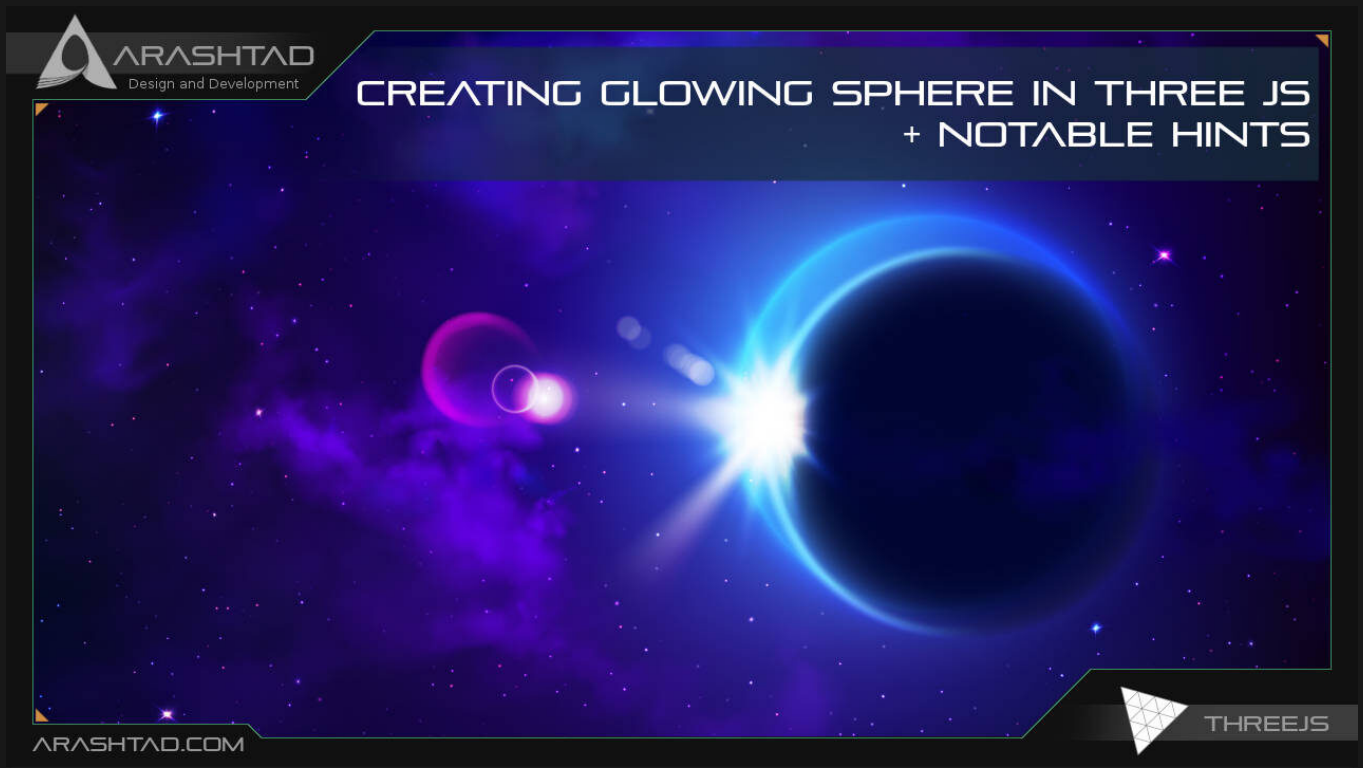
Creating Glowing Sphere in Three JS + Notable Hints

No comments



One of the very useful projects that can be used for larger projects is creating a glowing sphere using Three js. The use cases of this object are very wide. It can be used for simulating the sun, the moon, the planets, or the planet earth itself. A glowing sphere usually has a kind of halo based on the kind of object it is. For instance, the planet earth needs a kind of a halo for simulating its atmosphere or the moon needs a halo for simulating the white light around it. When we want to simulate the sun in open space, we need a glowing sphere as well.

Creating a Glowing Sphere from Scratch

In this article, we are going to get familiar with the steps we can take to create a glowing sphere in Three js. Of course, we are going to simulate the halo of the glowing sphere with the aid of shaders. If you are a beginner, don't worry because we are going to teach you how to do this task from scratch.

As we mentioned earlier, a glowing sphere is composed of the sphere itself with a halo which we are going to design

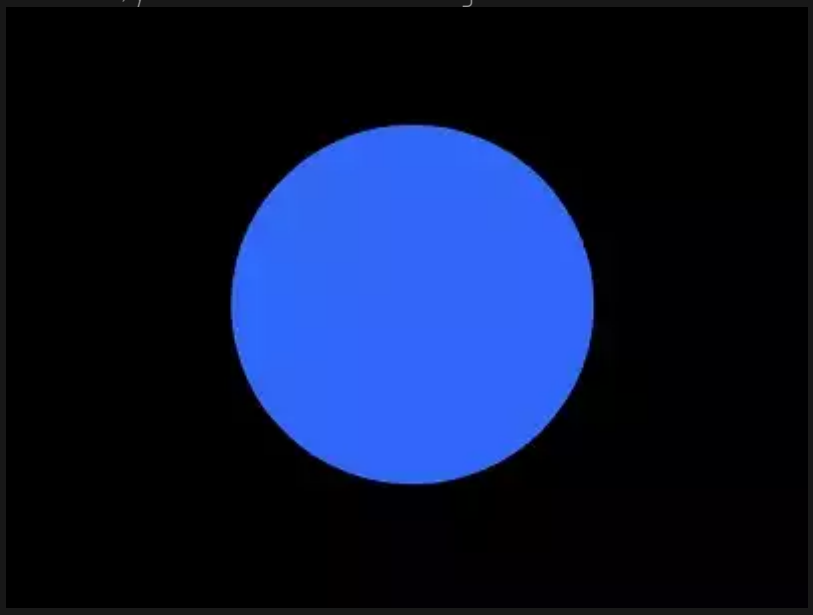
using a shader. There is also another important element that we should consider when designing a glowing sphere and that is considering a light source for it. The position of the light source should be the same as the position of the object. As a result, the object will work as a light source as well.

Basics of Creating Sphere in Three JS

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 files 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 glowingSphere` `cd glowingSphere` Then, inside of the glowing sphere 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 `glowingSphere` 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 glowingSphere` `npm install` Afterward, 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`



Creating A Glowing Sphere

Now, if we want to create a halo to make it glow, we need to create a shader. So, we modify the code a bit. To do so, add the followings to the `main.js`:

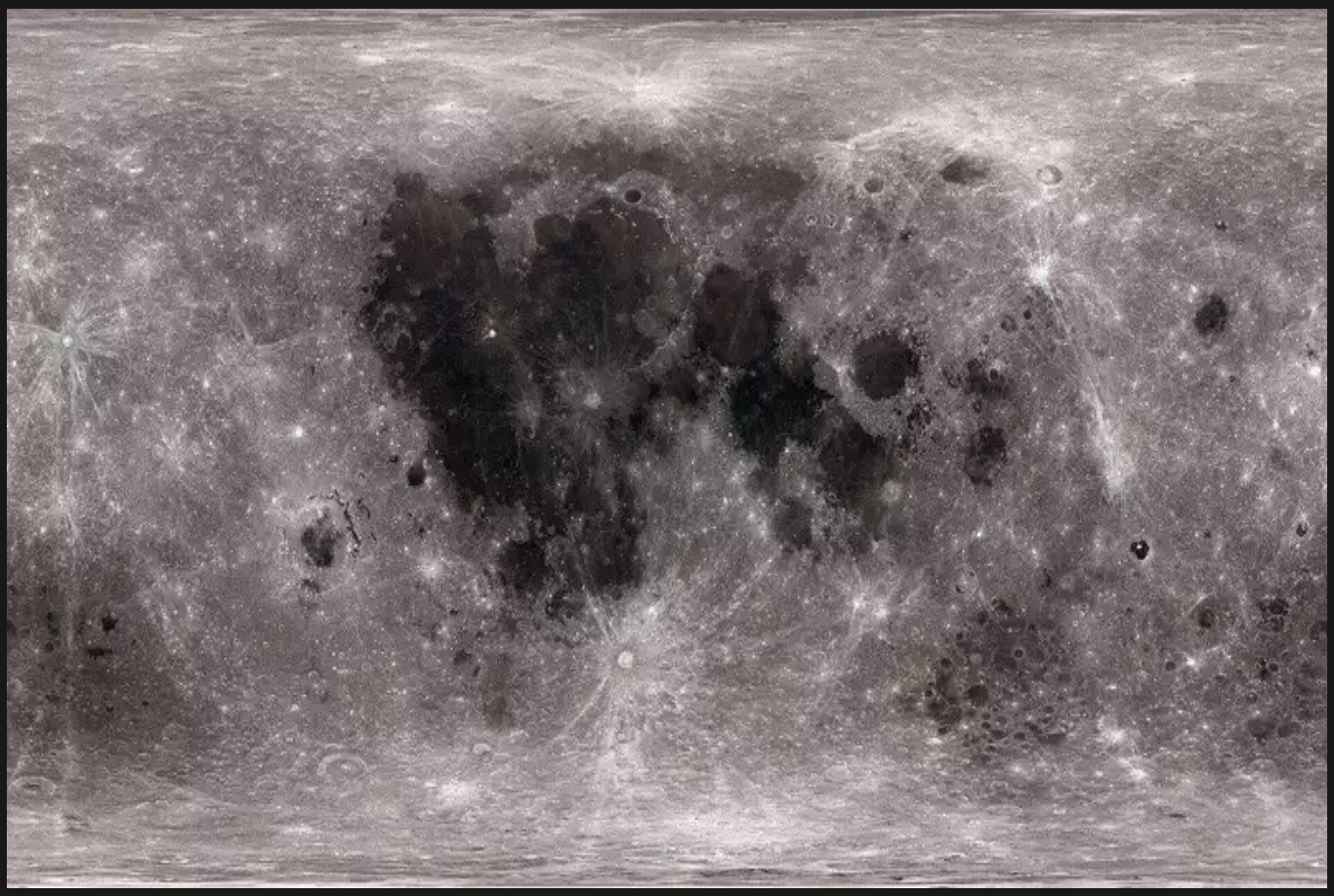
```
const haloVertexShader = /*glsl*/`
varying vec3 vertexNormal;
void main() {
     vertexNormal = normal;
     gl_Position = projectionMatrix * modelViewMatrix * vec4
(position, 1.0);
}
`;
const haloFragmentShader = /*glsl*/`
varying vec3 vertexNormal;
void main() {
float intensity = pow(0.9 - dot(vertexNormal, vec3(0, 0, 1.0)), 2.0);
gl_FragColor = vec4(0.8, 1.0, 0.6, 0.2) * intensity;
}
`;
const halo = new THREE.Mesh(
     new THREE.SphereGeometry(5, 50, 50),
     new THREE.ShaderMaterial({
          vertexShader:haloVertexShader,
          fragmentShader:haloFragmentShader,
```

```
            blending: THREE.AdditiveBlending,
            side: THREE.BackSide
        })
)

scene.add(sphere);
halo.scale.set(1.2, 1.2, 1.2);
scene.add(halo);
```

Now, we save the code and see a halo around the sphere as a result:

What we did here was to create a halo sphere using shaders. Shaders are usually written in other files called `vertex.glsl` and `fragment.glsl`. But here for the ease of code execution, we have written it in the `main.js` file with ` ` ` ` marks. Notice that you need to install the Comment tagged templates in VSCode to be able to support other languages in the main.js. This way writing shader is much easier and faster. In the haloFragmentShader, we have written an algorithm to create this kind of glowing effect and notice that we created another sphere in addition to the main sphere in the boilerplate to simulate a halo so that we can have the glowing sphere using the 2 of them in the same place.Adding the Texture of the Moon:In order to give meaning to our design, we can add the texture of the moon. To do so, we need the UV map of its surface. You can google UV map of the moon and the photo you need should be something like this:



Now, we need to create a folder and paste the uv map of the moon and change the material from:

```
const material = new THREE.MeshBasicMaterial({
     color:0x3268F8
})
```

To:

```
const material = new THREE.MeshBasicMaterial({
     map : new THREE.TextureLoader().load('./img/moon.jpg')
})
```

What we actually did was to change the color property to the map property. Finally, you will be to see the rotating moon like this:

The final thing that we need to do is to create a light point. We have covered a full tutorial on the point lights in our previous articles. We can use either the spotlight or the directional light to simulate the light in space. However, since we do not have any other objects here, we can neglect this part and do not add any light sources in the position of the moon. But if we want to have more objects and planets in space or simulate the moonlight in the night sky, we would rather use a spotlight or directional light.

Notice that if the planets are going to be simulated near each other, the spotlight might be a better choice. But if you want to simulate only the moon in the sky, the directional light may be a better option to provide the scene of a glowing sphere in the middle of the night, because we usually use the directional lights for the sources that are far away and have parallel beams. According to the above explanations, we use the directional light and consider the white color for it and then add it to the scene:

```
const directionalLight = new THREE.DirectionalLight( 0xffffff, 0.5 );
scene.add( directionalLight );
```

Notice that you should set the position of the point light according to the sphere object (They should have the same position).

Final Word

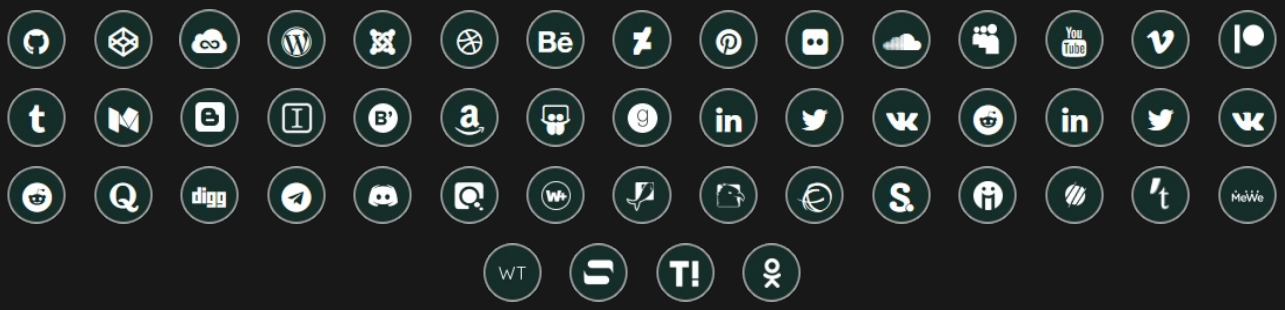
In this tutorial, we have provided the guidelines for creating a sphere from scratch using the Vite plugin and the base

code or boilerplate for every Three js project. Afterward, we added the fragment and vertex shaders written in GLSL to our script to simulate the glowing effect on our glowing sphere in Three js. The halo or the glowing effect comes from another sphere that has been scaled relative to the main sphere. Then, to give meaning to the glowing sphere we have created, we decided to create a moon with a UV map of it, which was downloaded from Google images. Finally, we considered adding a light source in case additional objects might be added to the scene in the future.

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