Extracted from:

3D Game Programming for Kids, Second Edition

Create Interactive Worlds with JavaScript

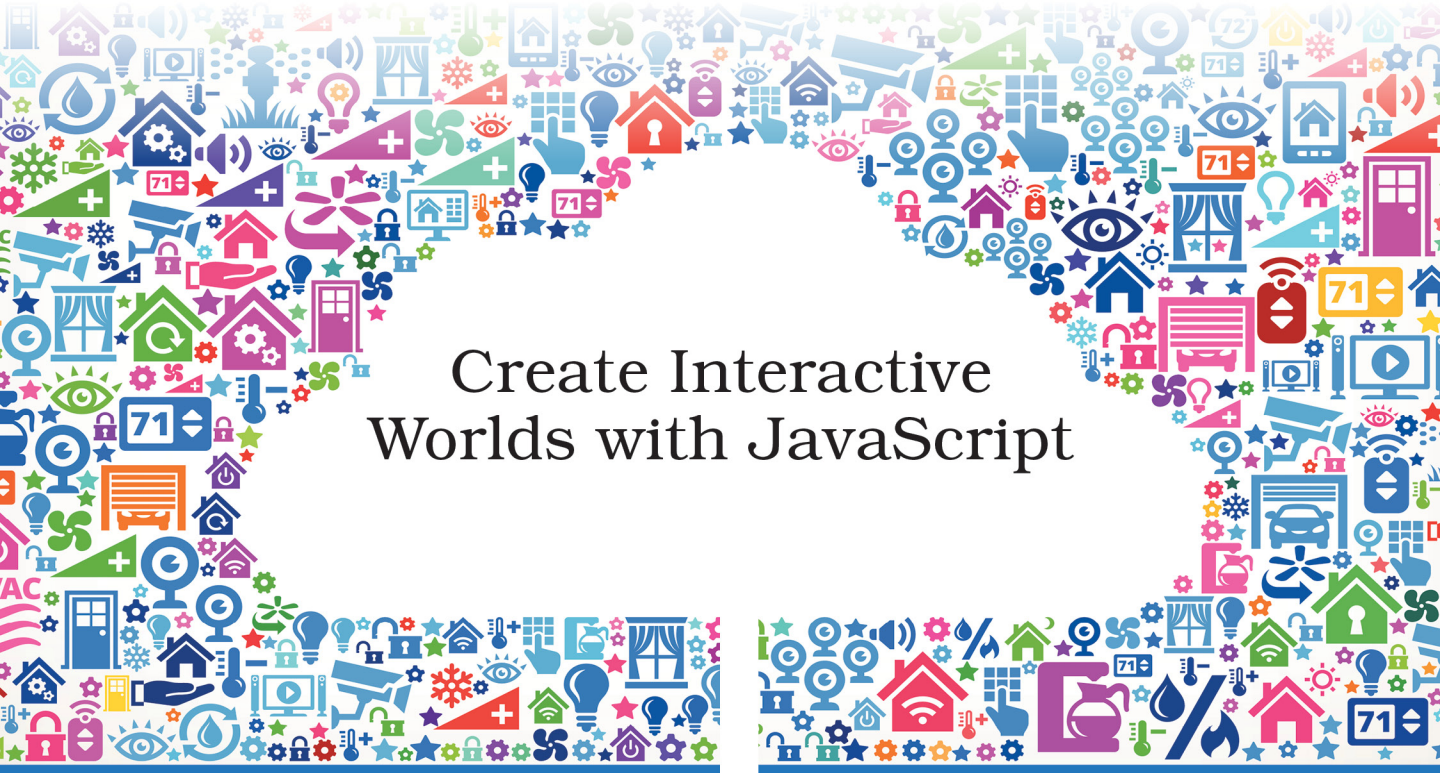
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

3D Game Programming for Kids

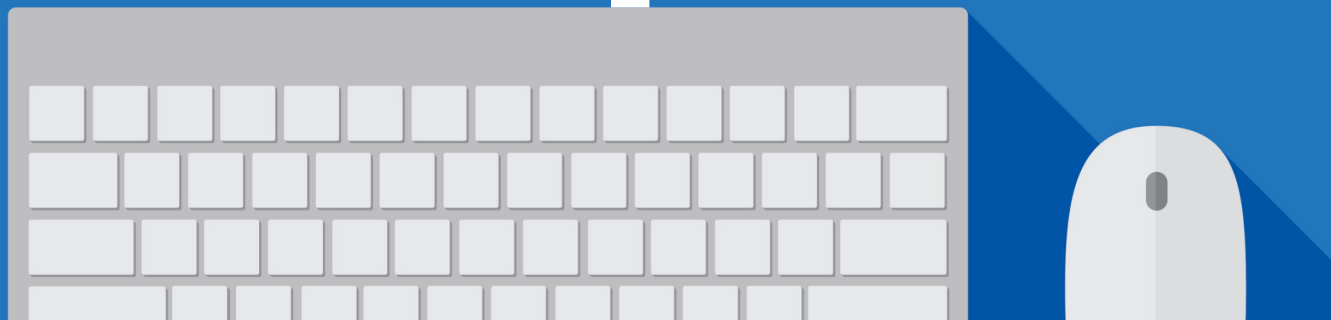
Second Edition



Create Interactive Worlds with JavaScript

Chris Strom

edited by Adaobi Obi Tulton



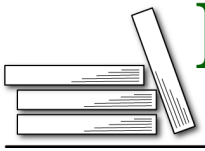
3D Game Programming for Kids, Second Edition

Create Interactive Worlds with JavaScript

Chris Strom

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Brian MacDonald
Supervising Editor: Jacquelyn Carter
Development Editor: Adaobi Obi Tulton
Copy Editor: Paula Robertson
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

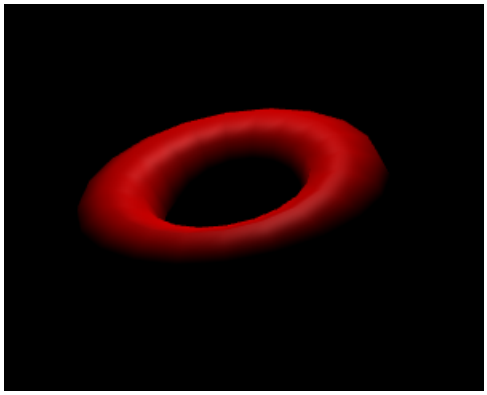
ISBN-13: 978-1-68050-270-1
Book version: P1.0—July 2018

Point Light

To increase the realism, let's add a "point light," which is kind of like a light bulb. Add the code for our point light below the donut code.

```
var point = new THREE.PointLight('white', 0.8);
point.position.set(0, 300, -100);
scene.add(point);
```

We're positioning the light above and behind the donut. The point light is bright, but not too bright at 0.8. The result should be a pretty cool-looking donut:



One weird thing about programming with 3D lights is that the light is not coming from an object in the scene. We positioned the source of the light at X-Y-Z coordinates of (0, 300, -100). The light comes from here, but there's no bulb to be seen.

It often helps to see a bulb, so let's add a glowing white bulb to the point light. The light is already there, so this is just a marker—a phony bulb. Add the phony bulb below the real point light code.

```
var shape = new THREE.SphereGeometry(10);
var cover = new THREE.MeshPhongMaterial({emissive: 'white'});
var phonyLight = new THREE.Mesh(shape, cover);
point.add(phonyLight);
```

Just because it is fake, doesn't mean that we have to make it look fake. We give it an emissive color of bright white so that it emits white—kind of like a real light bulb.

Shininess

The donut is pretty cool looking already. Next, let's play with the shininess of the material wrapping the donut. In 3D programming, the shininess color is called the *specular color*. The specular color combines with the light shining

on it. If a bright light shines on a dark specular color, it produces very little shine. This is how the material looks now. If we make a specular color brighter, then a bright light combines to produce shininess.

Set the specular color of our donut's material just after we create the cover.

```
var shape = new THREE.TorusGeometry(50, 20, 8, 20);
var cover = new THREE.MeshPhongMaterial({color: 'red'});
➤ cover.specular.setRGB(0.9, 0.9, 0.9);
var donut = new THREE.Mesh(shape, cover);
donut.position.set(0, 150, 0);
scene.add(donut);
```

We are using red-green-blue colors like we did to build random colors back in Chapter 5, *Functions: Use and Use Again*, on page ?. But here, we're not making colors, just variations of gray. When all of the RGB values are the same, you get gray. When they're near 0, an almost black gray is produced. When they are all close to 1, you get a very light, almost white gray.

And when that specular color is light, we see more of the shine as shown in the figure on page 9.

We've now met all the different kinds of color that work together to add realism to 3D objects. Let's have a look at something even cooler: shadows.

Shadows

Drawing shadows is a lot of work for computers, so don't go crazy with them. Since they are so much work, they're disabled at first. We have to carefully work through each object that needs a shadow—and turn on shadows for the scene and light as well.

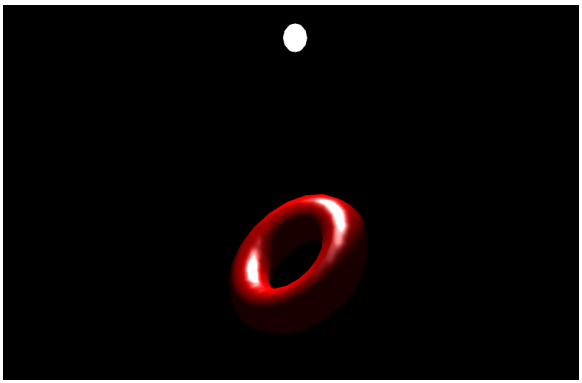
The first thing we need is some ground to see a shadow—we won't see a shadow unless the shadow falls on something. Add the code for the ground below the code for the donut, and above the light code.

```
var shape = new THREE.PlaneGeometry(1000, 1000, 10, 10);
var cover = new THREE.MeshPhongMaterial();
var ground = new THREE.Mesh(shape, cover);
ground.rotation.x = -Math.PI/2;
scene.add(ground);
```

That creates a plane, rotates it flat, and adds it to the scene—without any shadows.

To enable shadows, we follow these four steps:

1. Enable shadows in the renderer.



2. Enable shadows in a light.
3. Enable shadows in the object that casts a shadow.
4. Enable shadows in the object on which the shadow falls.

3D code will not even bother about shadows unless they are enabled in the renderer. To do so, set the `shadowMap.enabled` property in the renderer, which is above the `START CODING` line.

```
var renderer = new THREE.WebGLRenderer({antialias: true});
renderer.shadowMap.enabled = true;
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

Unless we say otherwise, lights do not cast shadows. So let's enable shadows from our point light.

```
var point = new THREE.PointLight('white', 0.8);
point.position.set(0, 300, -100);
point.castShadow = true;
scene.add(point);
```

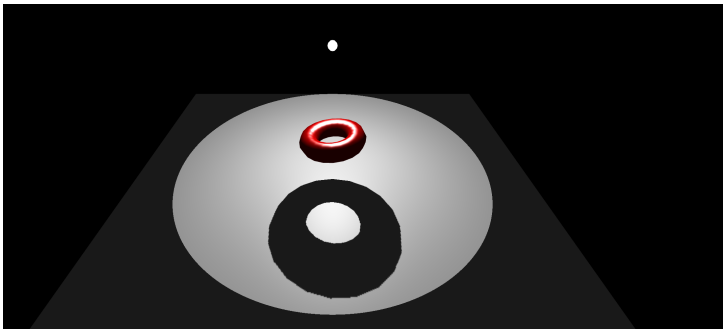
Next, we want our donut to cast a shadow, so enable the `castShadow` property on the donut.

```
var shape = new THREE.TorusGeometry(50, 20, 8, 20);
var cover = new THREE.MeshPhongMaterial({color: 'red'});
cover.specular.setRGB(0.9, 0.9, 0.9);
var donut = new THREE.Mesh(shape, cover);
donut.position.set(0, 150, 0);
donut.castShadow = true;
scene.add(donut);
```

Last, we tell the ground that it *receives* a shadow.

```
var shape = new THREE.PlaneGeometry(1000, 1000, 10, 10);
var cover = new THREE.MeshPhongMaterial();
var ground = new THREE.Mesh(shape, cover);
ground.rotation.x = -Math.PI/2;
ground.receiveShadow = true;
scene.add(ground);
```

With that, we should have a shadow for our spinning donut. If you don't see a shadow, check the JavaScript console *and* make sure that you've followed each of the four steps required for shadows.



Four steps might seem like a lot, but shadows require a lot of work by the computer. If every light makes shadows and every object casts a shadow and every object can have a shadow fall on it…well, then the computer is going to use all of its power drawing shadows and have nothing left for the user to actually play games.

Let's Play!

Lights and materials have a lot of properties that interact with each other. The best way to understand them is to play! Change the color of the point light to purple. What happens if the light is blue, but the donut is red? Change the amount of specular RGB in the donut—first keeping all three numbers the same and then making the green and blue values (the second and third values) 0.