Extracted from:

3D Game Programming for Kids, Second Edition

Create Interactive Worlds with JavaScript

This PDF file contains pages extracted from *3D Game Programming for Kids, Second Edition*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

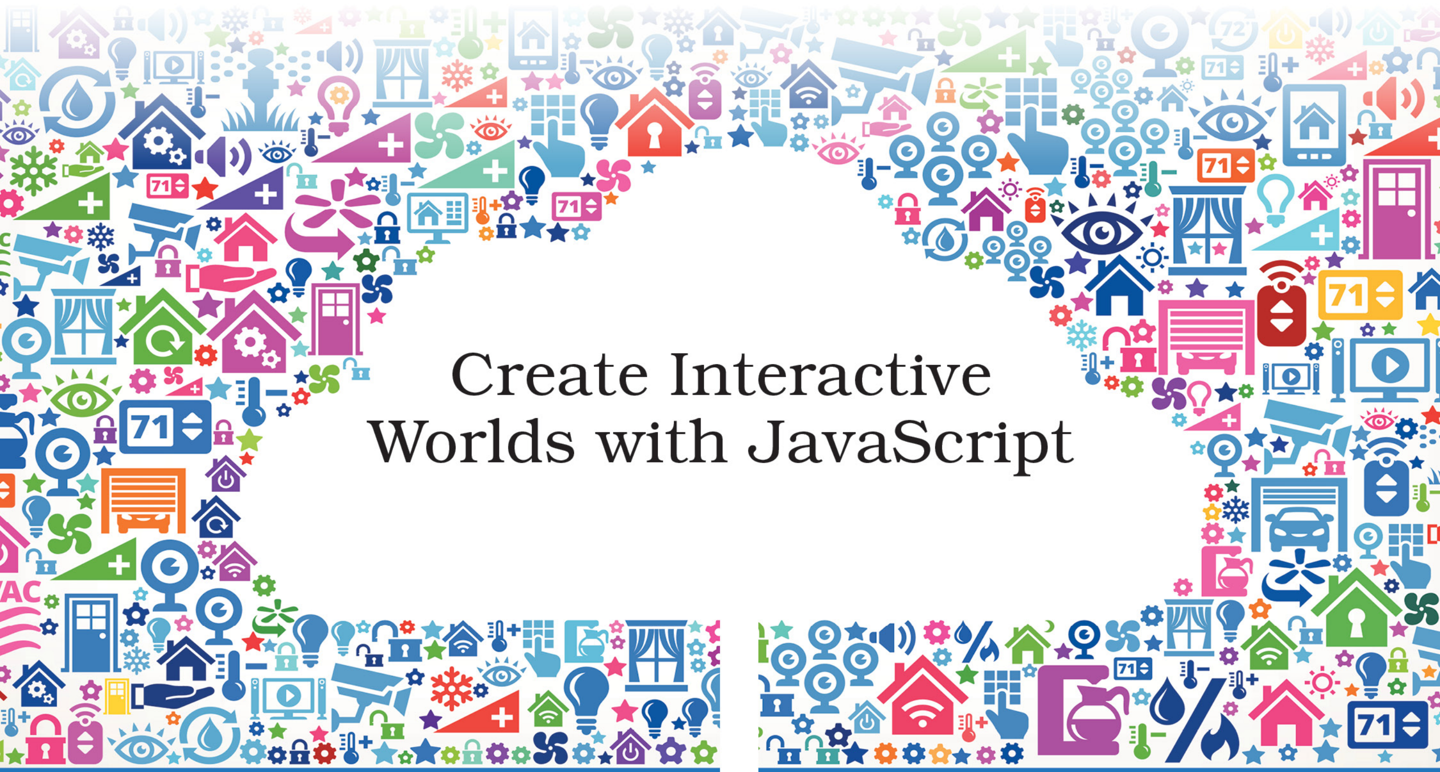
Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

The Pragmatic Bookshelf
Raleigh, North Carolina

The Pragmatic Programmers

3D Game Programming for Kids

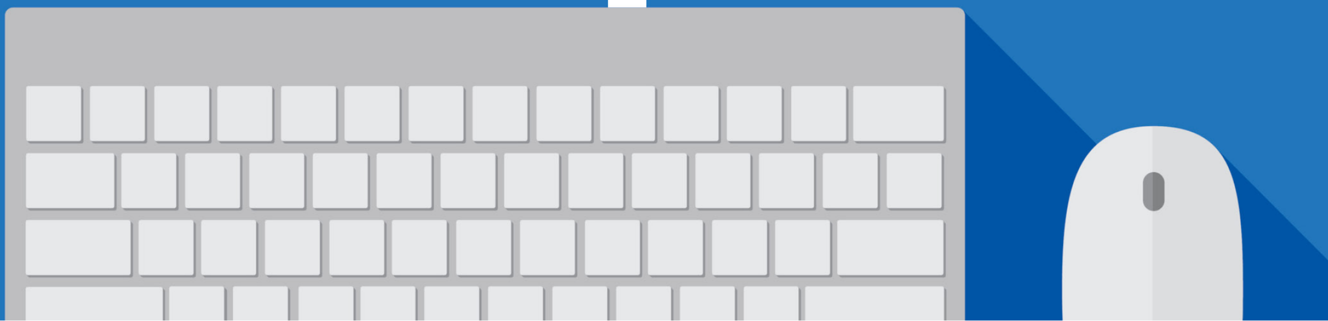
Second Edition



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

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

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

Creating Spheres

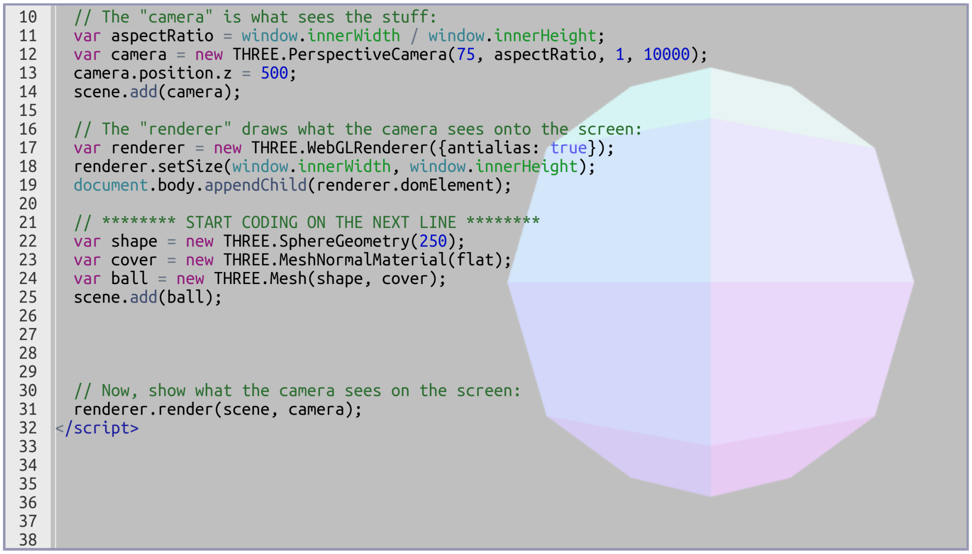
Balls are called *spheres* in geometry and in 3D programming. There are two ways to control spheres in JavaScript.

Size: SphereGeometry(100)

The first way that we can control a sphere is to describe how big it is. When we said new THREE.SphereGeometry(100), we created a ball whose radius was 100. What happens when you change the radius to 250?

```
var shape = new THREE.SphereGeometry(250);
var cover = new THREE.MeshNormalMaterial(flat);
var ball = new THREE.Mesh(shape, cover);
scene.add(ball);
```

This should make it much bigger:

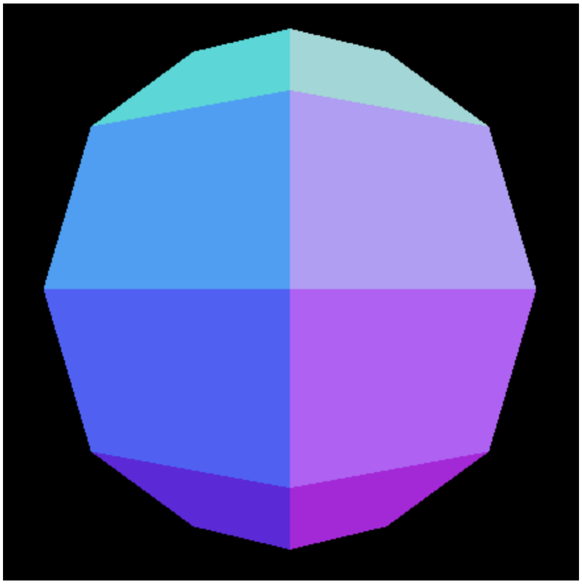


```
  // The "camera" is what sees the stuff:
  var aspectRatio = window.innerWidth / window.innerHeight;
  var camera = new THREE.PerspectiveCamera(75, aspectRatio, 1, 10000);
  camera.position.z = 500;
  scene.add(camera);

  // The "renderer" draws what the camera sees onto the screen:
  var renderer = new THREE.WebGLRenderer({antialias: true});
  renderer.setSize(window.innerWidth, window.innerHeight);
  document.body.appendChild(renderer.domElement);

  // ******** START CODING ON THE NEXT LINE ********
  var shape = new THREE.SphereGeometry(250);
  var cover = new THREE.MeshNormalMaterial(flat);
  var ball = new THREE.Mesh(shape, cover);
  scene.add(ball);




  // Now, show what the camera sees on the screen:
  renderer.render(scene, camera);
</script>
```

What happens if you change the 250 to 10? As you probably guessed, it gets much smaller. So that's one way we can control a sphere. What is the other way?

Not Chunky: SphereGeometry(100, 20, 15)

If you click the Hide Code button in 3DE, you may notice that our sphere isn't *really* a smooth ball:



You Can Easily Hide or Show the Code

If you click the `Hide Code` button in the upper-right corner of the 3DE window, you'll see just the game area and the objects in the game. This is how you'll play games in later chapters. To get your code back, click the `Show Code` button within the 3DE Code Editor.

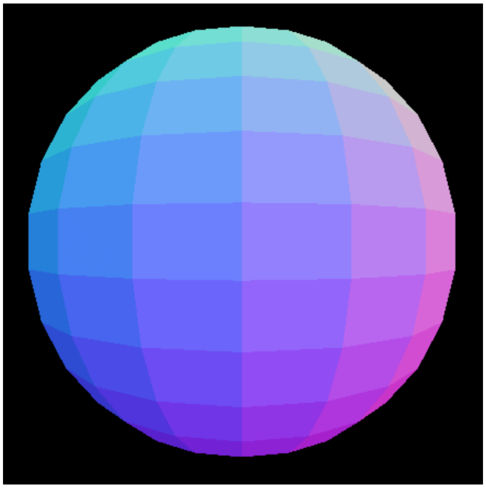
Computers can't really make a ball. Instead they fake it by joining a bunch of squares (and sometimes triangles) to make something that looks like a ball. Normally, we'll get enough *chunks*—the squares or triangles making up the surface—so that it's close enough.

Sometimes we want it to look a little smoother. To make it smoother, add some extra numbers to the `SphereGeometry()` line:

```
➤ var shape = new THREE.SphereGeometry(100, 20, 15);
  var cover = new THREE.MeshNormalMaterial(flat);
  var ball = new THREE.Mesh(shape, cover);
  scene.add(ball);
```

The first number is the size, the second number is the number of chunks around the sphere, and the third number is the number of chunks up and down the sphere.

This should make a sphere that is much smoother:



The number of chunks we get without telling SphereGeometry to use more may not seem great, but don't change it unless you must. The more chunks that are in a shape, the harder the computer has to work to draw it. It's usually easier for a computer to make things look smooth by choosing a different cover for the shape.

Let's Play!

Play around with the numbers a bit more. You're already learning quite a bit here, and playing with the numbers is a great way to keep learning!

Just don't make these numbers too high. Anything much beyond 1000 can lock the browser! Don't worry too much if the browser freezes or stops responding. You can always fix it with the steps described in *Recovering When 3DE Is Broken*, on page ?.

When you're done playing, move the ball out of the way by setting its position:

```
var shape = new THREE.SphereGeometry(100);
var cover = new THREE.MeshNormalMaterial(flat);
var ball = new THREE.Mesh(shape, cover);
scene.add(ball);
ball.position.set(-250,250,-250);
```

The three numbers move the ball to the left, up, and back. This frees up space to play with our next shape!