

Extracted from:

3D Game Programming for Kids

Create Interactive Worlds with JavaScript

This PDF file contains pages extracted from *3D Game Programming for Kids*, 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.

Copyright © 2013 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

3D Game Programming for Kids

Create Interactive Worlds with JavaScript

Chris Strom

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

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 courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at <http://pragprog.com>.

The team that produced this book includes:

Fahmida Rashid (editor)
Potomac Indexing, LLC (indexer)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Copyright © 2013 The Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-937785-44-4
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—September, 2013

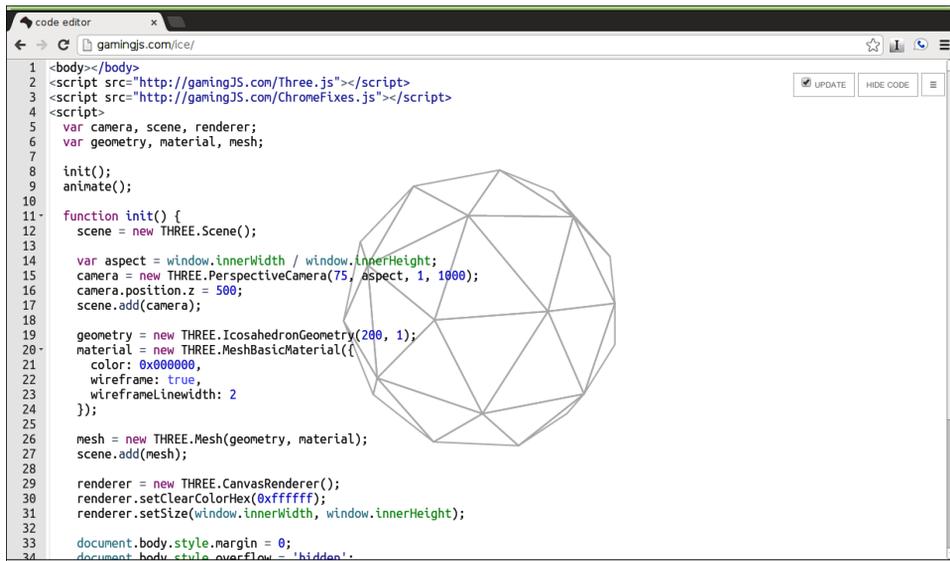
*For Greta, so that she knows she can do
anything.*

There will be plenty of time for detailed explanations later in this book. For now, let's get started with programming!

1.1 Programming with the ICE Code Editor

In this book, we'll use the ICE Code Editor to do our programming. The ICE Code Editor runs right inside a browser. It lets us type in our programming code and see the results immediately.

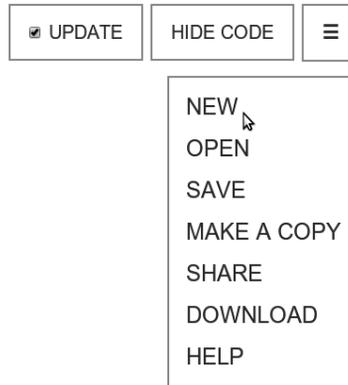
To get started, open the ICE Code Editor at <http://gamingJS.com/ice> using Google's Chrome web browser. It should look something like this:

A screenshot of a web browser displaying the ICE Code Editor. The browser's address bar shows 'gamingjs.com/ice/'. The editor's code area contains JavaScript code for creating a 3D scene with a wireframe icosahedron. The code includes imports for Three.js and ChromeFixes.js, and an 'init' function that sets up a scene, camera, and mesh. The mesh is a THREE.IcosahedronGeometry with 200 segments and a wireframe material. The scene is rendered in a perspective view. The browser's developer tools are not visible, but the 3D object is rendered in the center of the editor's canvas area.

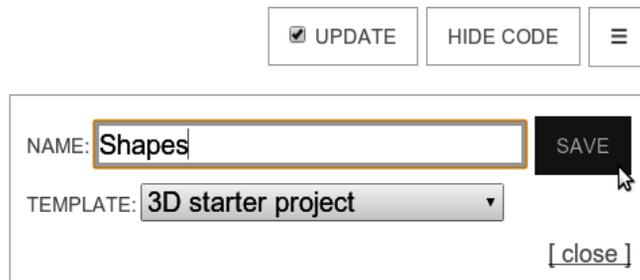
```
1 <body></body>
2 <script src="http://gamingJS.com/Three.js"></script>
3 <script src="http://gamingJS.com/ChromeFixes.js"></script>
4 <script>
5   var camera, scene, renderer;
6   var geometry, material, mesh;
7
8   init();
9   animate();
10
11 function init() {
12   scene = new THREE.Scene();
13
14   var aspect = window.innerWidth / window.innerHeight;
15   camera = new THREE.PerspectiveCamera(75, aspect, 1, 1900);
16   camera.position.z = 500;
17   scene.add(camera);
18
19   geometry = new THREE.IcosahedronGeometry(200, 1);
20   material = new THREE.MeshBasicMaterial({
21     color: 0x000000,
22     wireframe: true,
23     wireframeLinewidth: 2
24   });
25
26   mesh = new THREE.Mesh(geometry, material);
27   scene.add(mesh);
28
29   renderer = new THREE.CanvasRenderer();
30   renderer.setClearColorHex(0xffffffff);
31   renderer.setSize(window.innerWidth, window.innerHeight);
32
33   document.body.style.margin = 0;
34   document.body.style.overflow = 'hidden';
```

That spinning, multisided thing is a sample of some of the stuff we'll be working on in this book. In this chapter we'll create a new project named Shapes.

To create a new project in the ICE Code Editor, we click on the menu button (the button with three horizontal lines) in the upper-right corner of the screen and select New from the drop-down.



Type the name of the project, Shapes, in the text field and click the Save button. Leave the template set as 3D starter project.



Remember, none of the projects in this book will work if you're using the ICE Code Editor in Internet Explorer. Although some of the exercises will work with Mozilla Firefox, it's easiest to stick with a single browser (Google Chrome) for all our projects.

Coding with the ICE Code Editor



We'll be using the ICE Code Editor throughout this book. You only need web access the first time that you connect to <http://gamingJS.com/ice/>. After the first visit, ICE is stored in your browser so you can keep working even if you're not connected to the Internet.

When ICE opens a new 3D project, there is already a lot of code in the file. We'll look closely at that code later, but for now let's begin our programming adventure on line 20. Look for the line that says START CODING ON THE NEXT LINE.

```

1 <body></body>
2 <script src="http://gamingJS.com/Three.js"></script>
3 <script src="http://gamingJS.com/ChromeFixes.js"></script>
4 <script>
5 // This is where stuff in our game will happen:
6 var scene = new THREE.Scene();
7
8 // This is what sees the stuff:
9 var aspect_ratio = window.innerWidth / window.innerHeight;
10 var camera = new THREE.PerspectiveCamera(75, aspect_ratio, 1, 10000);
11 camera.position.z = 500;
12 scene.add(camera);
13
14 // This will draw what the camera sees onto the screen:
15 var renderer = new THREE.CanvasRenderer();
16 renderer.setSize(window.innerWidth, window.innerHeight);
17 document.body.appendChild(renderer.domElement);
18
19 // ***** START CODING ON THE NEXT LINE *****
20 ← Start programming here :)
21
22
23
24 // Now, show what the camera sees on the screen:
25 renderer.render(scene, camera);
26 </script>

```

On line 20, type the following:

```

var shape = new THREE.SphereGeometry(100);
var cover = new THREE.MeshNormalMaterial();
var ball = new THREE.Mesh(shape, cover);
scene.add(ball);

```

Once you finish typing that, you should see something cool:

```

10 var camera = new THREE.PerspectiveCamera(75, aspect_ratio,
11 camera.position.z = 500;
12 scene.add(camera);
13
14 // This will draw what the camera sees onto the screen:
15 var renderer = new THREE.CanvasRenderer();
16 renderer.setSize(window.innerWidth, window.innerHeight);
17 document.body.appendChild(renderer.domElement);
18
19 // ***** START CODING ON THE NEXT LINE *****
20 var shape = new THREE.SphereGeometry(100);
21 var wrapper = new THREE.MeshNormalMaterial();
22 var ball = new THREE.Mesh(shape, wrapper);
23 scene.add(ball);

```

The ball that we typed—the ball that we *programmed*—showed up in ICE. Congratulations! You just wrote your first JavaScript program!

Don't worry about the structure of the code just yet; you'll get familiar with it in [A Closer Look at JavaScript Fundamentals](#). For now, let's examine the 3D programming that we just did.

3D things are built from two parts: the shape and something that covers the shape. The combination of these two things, the shape and its cover, is given a special name in 3D programming: *mesh*.

Mesh is a fancy word for a 3D thing. Meshes need shapes (sometimes called *geometry*) and something to cover them (sometimes called *materials*). In this

chapter we'll look at different shapes. We won't deal with different covers for our shapes until [Working with Lights and Materials](#).

Once we have a mesh, we add it to the *scene*. The scene is where the magic happens in 3D programming. It is the world in which everything takes place. In this case, it's where our ball is hanging out, waiting for some friends. Let's add some other shapes to the scene so that the ball isn't lonely.

Your Work Is Saved Automatically



Your work is saved automatically, so you don't have to do it yourself. If you want to save the code yourself anyway, click the three-line menu button in ICE and select the Save option from the drop-down. That's it!

1.2 Making Shapes with JavaScript

So far we have seen only one kind of shape: a sphere. Shapes can be simple, like cubes, pyramids, cones, and spheres. Shapes can also be more complex, like faces or cars. In this book we'll stick with simple shapes. When we build things like trees, we'll combine simple shapes, such as spheres and cylinders, to make them.

Creating Spheres

Balls are always called spheres in geometry and in 3D programming. There are two ways to control the shape of a sphere in JavaScript.

Size: SphereGeometry(100)

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

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

- ❶ This points to where you should change the sphere's size.

This should make it much bigger:

```

4 <script>
5 // This is where stuff in our game will happen:
6 var scene = new THREE.Scene();
7
8 // This is what sees the stuff:
9 var aspect_ratio = window.innerWidth / window.innerHeight;
10 var camera = new THREE.PerspectiveCamera(75, aspect_ratio, 1, 10000);
11 camera.position.z = 500;
12 scene.add(camera);
13
14 // This will draw what the camera sees onto the screen:
15 var renderer = new THREE.CanvasRenderer();
16 renderer.setSize(window.innerWidth, window.innerHeight);
17 document.body.appendChild(renderer.domElement);
18
19 // ***** START CODING ON THE NEXT LINE *****
20 var shape = new THREE.SphereGeometry(250);
21 var cover = new THREE.MeshNormalMaterial();
22 var ball = new THREE.Mesh(shape, cover);
23 scene.add(ball);
24
25
26 // Now, show what the camera sees on the screen:
27 renderer.render(scene, camera);
28 </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's shape. What is the other way?

Not Chunky: SphereGeometry(100, 20, 15)

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



You Can Easily Hide or Show the Code



If you click the white Hide Code button in the upper-right corner of the ICE 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 white Show Code button within the ICE 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 the right number of *chunks* 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:

```

1 var shape = new THREE.SphereGeometry(100, 20, 15);
  var cover = new THREE.MeshNormalMaterial();

```

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

- 1 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:



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!

Don't Change the Chunkiness Unless You Have To



The number of chunks that 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. As you'll see in later chapters, it's usually easier for a computer to make things look smooth by choosing a different cover for the shape.

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

- 1 ball.position.set(-250,250,-250);

- 1 The three numbers move the ball to the left, up, and back. Don't worry much about what the numbers do right now—we'll talk about position when we start building game characters in [Chapter 3, Project: Making an Avatar, on page ?](#).

Making Boxes with the Cube Shape

Next we'll make a cube, which is another name for a box. There are three ways to change a cube's shape: the width, the height, and the depth.

Size: CubeGeometry(300, 100, 20)

To create a box, we'll write more JavaScript below everything that we used to create our ball. Type the following:

```
var shape = new THREE.CubeGeometry(100, 100, 100);
var cover = new THREE.MeshNormalMaterial();
var box = new THREE.Mesh(shape, cover);
scene.add(box);
```

If you have everything correct, you should see...a square:

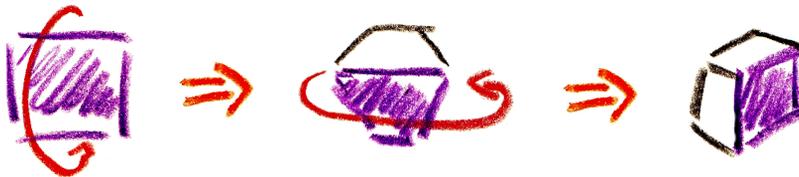


Well, that's boring. Why do we see a square instead of a box? The answer is that our *camera*, our perspective, is looking directly at one side of the box. If we want to see more of the box, we need to move the camera or turn the box. Let's turn the box by rotating it:

```
var shape = new THREE.CubeGeometry(100, 100, 100);
var cover = new THREE.MeshNormalMaterial();
var box = new THREE.Mesh(shape, cover);
scene.add(box);
❶ box.rotation.set(0.5, 0.5, 0);
```

- ❶ These three numbers turn the box down, counterclockwise, and left-right.

In this case, we rotate 0.5 down and 0.5 to the right:



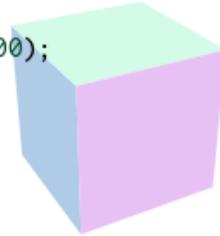
Try This Yourself



Rotating things takes a little getting used to, so play with the numbers. Try smaller and bigger numbers. A full rotation is 6.3 (we'll talk about that number later). Try setting two of the numbers to 0 and another to 0.1, then to 0.25, and finally to 0.5. If you can change the numbers fast enough, it's almost like the cube is spinning!

Setting the box rotation to (0.5, 0.5, 0) should rotate the cube so we can see that it really is a cube:

```
var shape = new THREE.CubeGeometry(100, 100, 100);
var cover = new THREE.MeshNormalMaterial();
var box = new THREE.Mesh(shape, cover);
scene.add(box);
box.rotation.set(0.5, 0.5, 0);
```

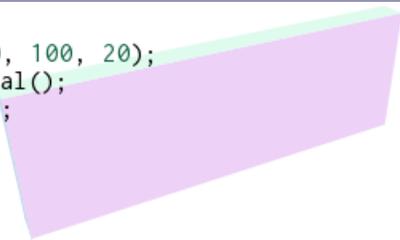


Each side of a cube doesn't have to be the same size. Our box so far is 100 wide (from left to right), 100 tall (up and down), and 100 deep (front to back). Let's change it so that it is 300 wide, 100 tall, and only 20 deep:

```
var shape = new THREE.CubeGeometry(300, 100, 20);
var cover = new THREE.MeshNormalMaterial();
var box = new THREE.Mesh(shape, cover);
scene.add(box);
box.rotation.set(0.5, 0.5, 0);
```

This should show something like this:

```
var shape = new THREE.CubeGeometry(300, 100, 20);
var cover = new THREE.MeshNormalMaterial();
var box = new THREE.Mesh(shape, cover);
scene.add(box);
box.rotation.set(0.5, 0.5, 0);
```



Play around with the numbers to get a good feel for what they can do.

Believe it or not, you already know a ton about JavaScript and 3D programming. There is still a lot to learn, of course, but you can already make balls and boxes. You can already move them and turn them. And you only had to write ten lines of JavaScript to do it all—nice!

Let's move our box out of the way so we can play with more shapes:

```
var shape = new THREE.CubeGeometry(300, 100, 20);
var cover = new THREE.MeshNormalMaterial();
var box = new THREE.Mesh(shape, cover);
scene.add(box);
box.rotation.set(0.5, 0.5, 0);
box.position.set(250, 250, -250);
```

Using Cylinders for All Kinds of Shapes

A cylinder, which is also sometimes called a tube, is a surprisingly useful shape in 3D programming. Think about it: cylinders can be used as tree trunks, tin cans, wheels.... Did you know that cylinders can be used to create cones, evergreen trees, and even pyramids? Let's see how!

Size: `CylinderGeometry(20, 20, 100)`

Below the box code, type in the following to create a cylinder:

```
var shape = new THREE.CylinderGeometry(20, 20, 100);
var cover = new THREE.MeshNormalMaterial();
var tube = new THREE.Mesh(shape, cover);
scene.add(tube);
```

If you rotate that a little (you remember how to do that from the last section, right?), then you might see something like this:



If you were not able to figure out how to rotate the tube, don't worry. Just add this line after the line with `scene.add(tube)`:

```
tube.rotation.set(0.5, 0, 0);
```

When making a cylinder, the first two numbers describe how big the top and bottom of the cylinder is. The last number is how tall the cylinder is. So our cylinder has a top and bottom that are 20 in size and 100 in height.

If you change the first two numbers to 100 and the last number to 20, what happens? What happens if you make the top 1, the bottom 100, and the height 100?

Try This Yourself



Play with those numbers and see what you can create!

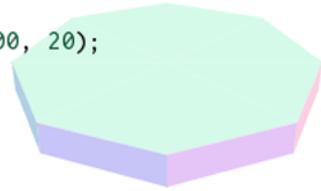
What did you find?

A flat cylinder is a disc:

```

var shape = new THREE.CylinderGeometry(100, 100, 20);
var cover = new THREE.MeshNormalMaterial();
var tube = new THREE.Mesh(shape, cover);
scene.add(tube);
tube.rotation.set(0.5, 0, 0);

```

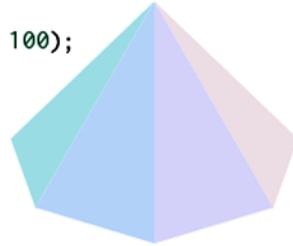


And a cylinder that has either the top or bottom with a size of 1 is a cone:

```

var shape = new THREE.CylinderGeometry(1, 100, 100);
var cover = new THREE.MeshNormalMaterial();
var tube = new THREE.Mesh(shape, cover);
scene.add(tube);
tube.rotation.set(0.5, 0, 0);

```



It should be clear that you can do a lot with cylinders, but we haven't seen everything yet. We have one trick left.

Pyramids: `CylinderGeometry(1, 100, 100, 4)`

Did you notice that cylinders look chunky? It should be no surprise then, that you can control the chunkiness of cylinders. If you set the number of chunks to 20, for instance, with the disc, like this:

```

var shape = new THREE.CylinderGeometry(100, 100, 10, 20);
var cover = new THREE.MeshNormalMaterial();
var tube = new THREE.Mesh(shape, cover);
scene.add(tube);
tube.rotation.set(0.5, 0, 0);

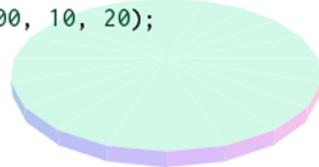
```

then you should see something like this:

```

var shape = new THREE.CylinderGeometry(100, 100, 10, 20);
var cover = new THREE.MeshNormalMaterial();
var tube = new THREE.Mesh(shape, cover);
scene.add(tube);
tube.rotation.set(0.5, 0, 0);

```



Just as with spheres, you should use lots of chunks like that only when you really, really need to.

Can you think how you might turn this into a pyramid? You have all of the clues that you need.

Try This Yourself

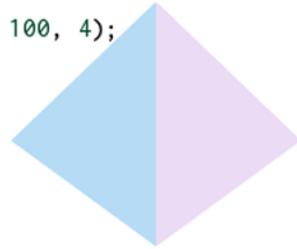


Play with different numbers and see what you can create!

Were you able to figure it out? Don't worry if you weren't. The way we'll do it is actually pretty sneaky.

The answer is that you need to *decrease* the number of chunks that you use to make a cone. If you set the top to 1, the bottom to 100, the height to 100, and the number of chunks to 4, then you'll get this:

```
var shape = new THREE.CylinderGeometry(1, 100, 100, 4);
var cover = new THREE.MeshNormalMaterial();
var tube = new THREE.Mesh(shape, cover);
scene.add(tube);
tube.rotation.set(0.5, 0, 0);
```



It might seem like a cheat to do something like this to create a pyramid, but this brings us to a very important tip with any programming:

Cheat Whenever Possible



You shouldn't cheat in real life, but in programming—especially in 3D programming—you should always look for easier ways of doing things. Even if there is a *usual* way to do something, there may be a *better* way to do it.

You're doing great so far. Move the tube out of the center like we did with the cube and the sphere:

```
tube.position.set(250, -250, -250);
```

Now let's move on to the last two shapes of this chapter.