Chris Strom
Übersetzung von Kathrin Lichtenberg

Kids programmieren

3D-Spiele mit JavaScript

Spannende 3D-Welten mit JavaScript
Mit vielen coolen Spielideen

o'reillys basics

O'REILLY®

Keine Programmier-kenntnisse nötig

KIDS PROGRAMMIEREN 3D-SPIELE MIT JAVASCRIPT

Chris Strom

Deutsche Übersetzung von Kathrin Lichtenberg

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
E-Mail: kommentar@oreilly.de

1. Auflage 2014

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten
sind im Internet über *http://dnb.de* abrufbar.

Lektorat: Imke Hirschmann, Köln
Übersetzung: Kathrin Lichtenberg, Ilmenau
Korrektorat: Sibylle Feldmann, Düsseldorf
Umschlaggestaltung: Michael Oreal, Köln
Produktion: Karin Driesen, Köln
Satz: III-satz, Husby; www.drei-satz.de
Belichtung, Druck und buchbinderische Verarbeitung: Himmer AG, Augsburg

ISBN: 978-3-95561-496-6

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

INHALT

Danksagung

Ohne meine wunderbare Frau Robin bin ich nichts. Sie erträgt nicht nur, dass ich tagelang verschwinde, um zu schreiben. Sie hilft mir auch auf unzählige andere Arten. Sie hat die allerersten Fassungen dieses Buches gelesen und korrigiert. Sie hilft bei der Durchführung der Kid Hackathons, die zur Entwicklung dieses Buches beitrugen (okay, sie führt sie durch). Und ja – sie ist eine fantastische Ehefrau und Mutter.

Ein riesiges Dankeschön geht auch an meinen Sohn Luke, der das wichtigste Versuchskaninchen für die ersten Versionen dieses Buches war. Sein sachliches Feedback hat dazu beigetragen, dass es viel besser wurde. Danke auch an meine Tochter Elora, die mit ihren Einsichten nicht hinter dem Berg gehalten hat.

Und natürlich geht ein großer Dank auch an meine technischen Gutachter. Es ist nicht leicht, ein Buch aus Sicht eines Kindes zu bewerten, doch meine Gutachter waren dieser Aufgabe mehr als gewachsen. Danke, Alec M., Hana B., Dave S., Thad K., Maik Schmidt, Silvia Domenech und Mark Musante.

Ein besonderer Dank geht an Sophie H., die das Spiel inspiriert hat, aus dem schließlich *Projekt: Obstjagd* wurde.

Dieses Buch wäre ohne die großartige Arbeit von Ricardo Cabello Miguel, von allen liebevoll »Mr.doob« genannt, nicht entstanden. Ricardo ist der wichtigste Programmierer von Three.js, der 3D-JavaScript-Bibliothek, die wir in diesem Buch benutzen. Er schrieb außerdem die ursprüngliche Implementierung des ICE Code Editors, die wir hier verwenden. Ohne seine unglaublichen Fähigkeiten wäre dieses Buch nicht zu dem geworden, was es ist. Danke auch an Chandler Prall für seine Arbeit an der Physijs-Physik-Engine, die wir hier ausgiebig benutzen. Chandler hat auch die vielen, vielen Fragen beantwortet, die ich beim Lernen hatte.

Einleitung

Herzlich willkommen in der Welt der Programmierung!

Ich will nicht lügen – es ist manchmal eine frustrierende Welt (ich muss wenigstens einmal in der Woche weinen). Aber der ganze Frust lohnt sich. Du kannst diese Welt nach deinem Belieben gestalten. Du kannst deine Welt mit anderen teilen. Du kannst Dinge bauen, die wirklich etwas bewegen.

Dieses Buch, das du so eifrig zu lesen begonnen hast, ist eine großartige Möglichkeit, um mit dem Programmieren zu beginnen. Es ist vollgestopft mit klaren und verständlichen Erklärungen. Und das Beste ist, dass wir einige wirklich coole Spiele herstellen werden. Das wird richtig klasse!

Wie ich zu programmieren gelernt habe

Als Kind habe ich die Programme für Computerspiele aus Büchern kopiert. Das ist schon lange her. Ich kaufte Bücher, in denen nichts weiter als die Programme zu lesen waren, und tippte sie in den Computer ein.

Als ich damit begann, hatte ich keine Ahnung, was ich eigentlich tat. Schließlich fing ich an, bestimmte Dinge zu erkennen, die immer wieder vorkamen, und ich verstand sie auch schon ein bisschen.

Ich begann damit, Dinge zu ändern – zuerst nur Kleinigkeiten –, um zu sehen, was passierte. Danach kamen größere Änderungen. Schließlich wurde ich recht gut darin. Und irgendwann konnte ich meine eigenen Programme schreiben. Ich hoffe, dass dieses Buch auch dir dies ermöglicht, allerdings mit einem großen Unterschied: Ich erkläre, was hier passiert, sodass du nicht so viel raten musst.

Was du für dieses Buch brauchst

Nicht alle Webbrowser können die coolen 3D-Spielobjekte erzeugen, die wir in diesem Buch bauen werden. Um das meiste aus diesem Buch herauszuholen, solltest du auf deinem Computer den Webbrowser Google Chrome (*https://www.google.com/chrome/*) installieren. Andere Webbrowser funktionieren auch, allerdings greifen einige der Übungen in diesem Buch auf Eigenschaften und Funktionen zurück, die es nur in Google Chrome gibt. Ein Browser, der definitiv nicht für diese Übungen zu gebrauchen ist, ist der Microsoft Internet Explorer.

Für die meisten Beispiele in diesem Buch ist ein beliebiger Computer, auf dem Google Chrome installiert ist, ausreichend. Spätere Übungen, die interessante Lichter, Schatten und 3D-Materialien benutzen, erfordern einen Computer, der WebGL unterstützt. Du kannst die diesbezüglichen Fähigkeiten Deines Computers testen, indem du die Get-WebGL-Site (*http://get.webgl.org/*) besuchst. Mach dir aber keine allzu großen Sorgen um WebGL; du kannst immer noch eine Menge programmieren, auch wenn dein Computer nicht mit aufwendigeren 3D-Grafiken zurechtkommt.

Was ist JavaScript?

Es gibt viele, viele Programmiersprachen. Manche Programmierer führen gern lange Streitgespräche darüber, welche die *beste* ist, in Wahrheit aber bieten alle Sprachen einzigartige und nützliche Dinge.

Wir benutzen in diesem Buch die Programmiersprache JavaScript. Wir programmieren in JavaScript, weil es die Sprache des Webs ist. Es ist die einzige Programmiersprache, die alle Webbrowser ohne zusätzliche Software verstehen. Wenn du in JavaScript program-

mieren gelernt hast, kannst du nicht nur solche Spiele herstellen, die du in diesem Buch kennenlernen wirst, sondern kannst auch alle möglichen Websites programmieren.

Wir werden allerdings keine Experten in JavaScript.

Wir werden hier gerade so viel JavaScript behandeln, dass du in der Lage bist, die Spiele in diesem Buch zu programmieren. Das ist schon eine ganze Menge an JavaScript – ausreichend, um damit ohne größere Schwierigkeiten weiterlernen zu können.

Wie du dieses Buch lesen solltest

Du findest in diesem Buch zwei Arten von Kapiteln: Projektkapitel und Lernkapitel. Die Projektkapitel starten mit »Projekt«, wie etwa Kapitel 1, *Projekt: Einfache Formen herstellen* auf Seite 1. Alle anderen sind Lernkapitel.

Falls du das Programmieren genauso lernen möchtest wie ich, dann lies die Projektkapitel und führe alle Übungen durch. Du stellst coole Spielfiguren her sowie Welten, in denen du spielen kannst. Du erzeugst Weltraumsimulationen. Du bastelst lila Monster. Du produzierst alle möglichen tollen Sachen.

Solltest du dich aber fragen, *warum* die Spiele so geschrieben wurden, wie sie es sind, dann lies die Lernkapitel. Wir werden nicht *alles* über die Programmierung behandeln, aber es sollte ausreichend Stoff sein, um zu verstehen, warum wir Dinge so gemacht haben, wie sie sind. Das sind die Kapitel, die ich als Kind gern gehabt hätte.

Hinweise zur deutschen Fassung dieses Buches

Du beschäftigst dich bestimmt schon eine Weile mit Computern und dem Internet und weißt deshalb, dass du dabei an der englischen Sprache nicht vorbeikommst. Das ist beim Programmieren nicht anders. So gut wie alle Programmiersprachen basieren auf der englischen Sprache. Das gilt auch für JavaScript, das du in diesem Buch kennenlernst. Die Programmierplattform, die du benutzen wirst, um die hier gezeigten Beispiele auszuprobieren, verwendet englische Befehle. In den bereitgestellten Vorlagen für den Code, den sogenannten Templates, findest du englischsprachige Kom-

mentare, Variablen und Funktionen, und auch die geladenen JavaScript-Bibliotheken sind auf Englisch.

Zur besseren Orientierung haben wir im vorderen Teil des Buches den englischen Kommentaren ihre deutschen Übersetzungen beigefügt, später haben wir dann darauf verzichtet, schließlich kennst du dich dann schon aus und es ist nicht mehr nötig. Wenn du irgendwann anfängst, eigene Programme zu entwickeln, kannst du dir eigene Namen für deine Variablen und Funktionen ausdenken. Ob diese auf Deutsch oder auf Englisch sind oder auf einem ganz geheimen Code beruhen, den du dir selbst ausgedacht hast, ist ganz dir überlassen (bzw. hängt davon ab, was du mit deinen Mitstreitern vereinbart hast). Am besten legst du dir beim Durcharbeiten dieses Buches ein Wörterbuch bereit, um Wörter, deren Bedeutung dir unklar ist, nachzuschlagen.

Du musst außerdem beachten, dass die Schreibweise von Dezimalzahlen anders ist als im deutschsprachigen Raum üblich: Anstelle eines Kommas zum Abtrennen wird ein Punkt benutzt. Denke daran, das ebenfalls zu tun, da du dir ansonsten Fehler einhandelst, deren Ursache möglicherweise schwer zu finden ist.

Legen wir los!

Jetzt reicht es mit der Einführung – stürzen wir uns in die Programmierung!

PROJEKT: EINFACHE FORMEN HERSTELLEN

1

WENN DU DIESES KAPITEL GELESEN HAST, DANN

- weißt du, was ein Codeeditor ist und wie du ihn zum Programmieren benutzt
- weißt du, wie man verschiedene 3D-Formen herstellt
- kannst du einfaches JavaScript programmieren
- verstehst du, wie man 3D-Formen dazu bringt, sich zu bewegen

Wir haben später in diesem Buch noch genügend Zeit für Erklärungen. Fangen wir jetzt erst mal zu programmieren an!

Programmieren mit dem ICE Code Editor

Wir benutzen in diesem Buch den ICE Code Editor zum Programmieren. Der ICE Code Editor läuft direkt in einem Browser. Wir können unseren Programmcode eintippen und sehen sofort die Ergebnisse.

Öffne zuerst den ICE Code Editor unter *http://gamingJS.com/ice* mit dem Webbrowser Chrome von Google. Das sollte dann ungefähr so aussehen:

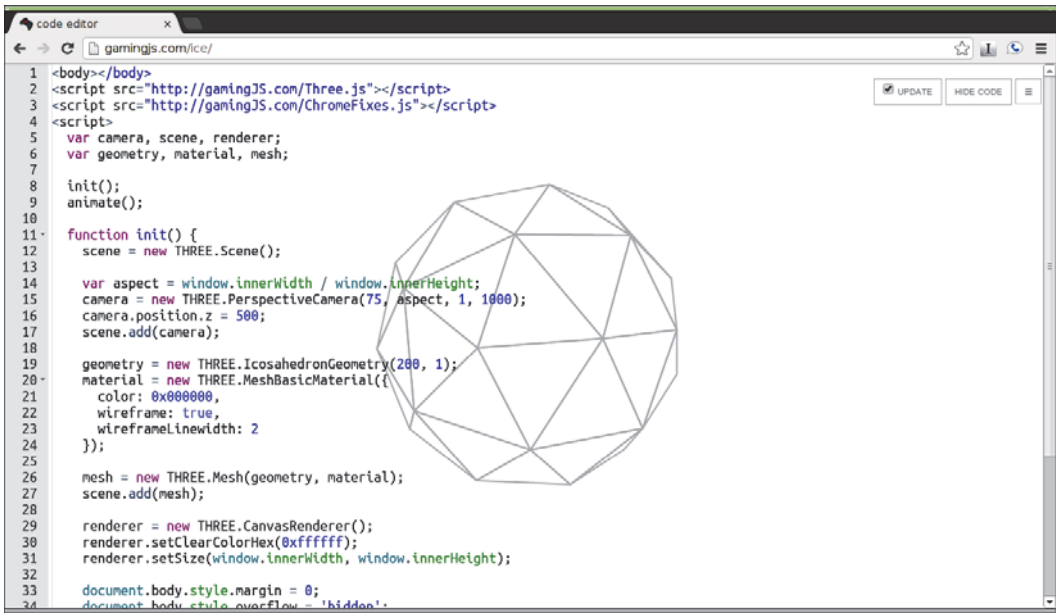


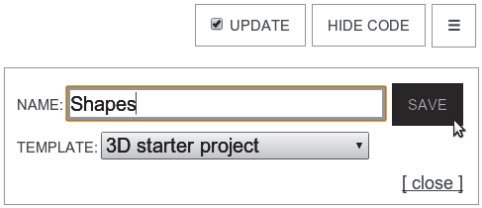
Abbildung 1-1 ▲
Der ICE Code Editor in Google Chrome

Das sich drehende Ding mit den vielen Seitenflächen ist ein Beispiel für die Sachen, die wir in diesem Buch machen wollen. In diesem Kapitel legen wir ein neues Projekt namens `Formen` an.

Um im ICE Code Editor ein neues Projekt anzulegen, klicken wir auf den Menü-Button in der oberen rechten Ecke des Bildschirms (das ist das Kästchen mit den drei waagerechten Strichen) und wählen `New` aus dem Menü.

Abbildung 1-2 ▶
Das geöffnete Menü

Tippe den Namen des Projekts, `Formen`, in das Textfeld ein und klicke dann auf `Save`. Das Template (ein Template ist eine Vorlage) lässt du einfach auf `3D starter project` stehen.

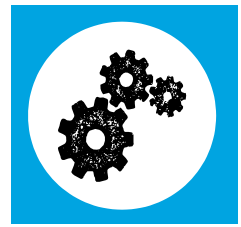


◀ Abbildung 1-3
Wir legen ein neues Projekt an.

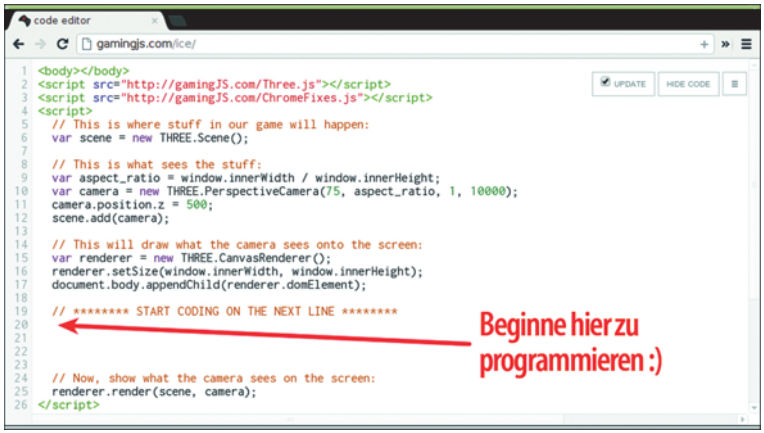
Denk dran, dass die Projekte in diesem Buch nicht funktionieren, wenn du den ICE Code Editor im Internet Explorer benutzt. Einige der Übungen klappen zwar mit Mozilla Firefox, am besten wäre es aber, wenn du für alle unsere Projekte bei einem einzigen Browser (Google Chrome) bleibst.

Mit dem ICE Code Editor programmieren

Wir benutzen in diesem Buch den ICE Code Editor. Du musst nur beim ersten Aufruf von *http://gamingJS.com/ice/* Zugang zum WWW haben. Nach dem ersten Besuch ist der ICE in deinem Browser gespeichert, sodass du auch dann damit arbeiten kannst, wenn du nicht mit dem Internet verbunden bist.



Wenn der ICE ein neues 3D-Projekt öffnet, gibt es in der Datei schon eine Menge Code. Wir schauen uns diesen Code später genauer an. Im Moment wollen wir jedoch unser Programmierabenteuer auf Zeile 20 beginnen. Suche nach der Zeile, auf der `START CODING ON THE NEXT LINE` steht.



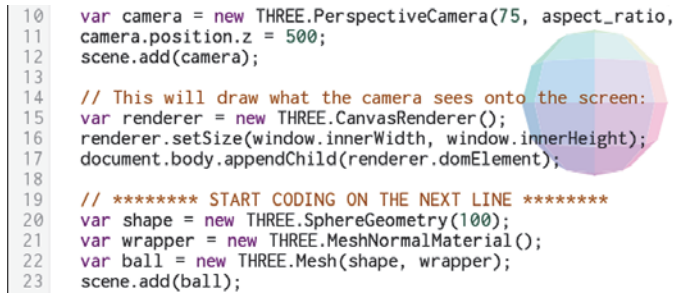
◀ Abbildung 1-4
Hier legst Du los.

Tippe in Zeile 20 Folgendes ein:

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

Sobald du damit fertig bist, solltest du etwas Cooles sehen:

Abbildung 1-5 ▶
Hier hat sich schon etwas getan.



Der Ball, den wir eingetippt – der Ball, den wir *programmiert* – haben, ist im ICE aufgetaucht. Herzlichen Glückwunsch! Du hast gerade dein erstes JavaScript-Programm geschrieben!

Mach dir erst einmal keine Sorgen um die Struktur des Codes. Du wirst dich in Kapitel 7, *Die Grundlagen von JavaScript näher untersucht,* damit vertraut machen. Im Moment wollen wir die 3D-Programmierung betrachten, die wir gerade durchgeführt haben.

3D-Dinge bestehen aus zwei Teilen: der Form und etwas, das diese Form bedeckt. Die Kombination aus beidem, der Form und ihrer Umhüllung, trägt in der 3D-Programmierung einen besonderen Namen: *Mesh* (Gitter oder auch Gewebe).

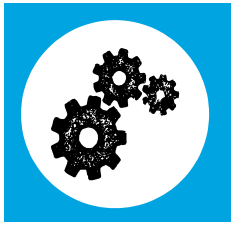
Mesh ist ein schickes Wort für ein 3D-Ding. Meshes brauchen Formen (manchmal als *Geometrie* bezeichnet) und etwas, um sie zu umhüllen (sogenanntes *Material*). Wir schauen uns in diesem Kapitel verschiedene Formen an. Zu unterschiedlichen Umhüllungen für unsere Formen kommen wir erst in Kapitel 12, *Mit Licht und Material arbeiten*.

Sobald wir ein Mesh haben, fügen wir es der *Szene* hinzu. Die Szene ist die Stelle in der 3D-Programmierung, an der gezaubert wird. Es ist die Welt, in der alles passiert. In diesem Fall ist es der Ort, an dem unser Ball herumlungert und auf Freunde wartet. Fügen wir

der Szene einige weitere Formen hinzu, damit der Ball nicht so allein ist.

Deine Arbeit wird automatisch gesichert

Deine Arbeit wird automatisch gesichert, du musst das also nicht selbst machen. Falls du deinen Code trotzdem selbst speichern möchtest, klickst du im ICE auf den Menü-Button mit den drei Linien und wählst den Save-Befehl. Ganz einfach!



Formen mit JavaScript herstellen

Bisher haben wir nur eine Art von Form gesehen: eine Kugel. Formen können einfach sein: Würfel, Pyramiden, Kegel und Kugeln. Formen können aber auch komplexer sein, wie Gesichter oder Autos. In diesem Buch bleiben wir bei einfachen Formen. Wenn wir so etwas wie Bäume bauen, kombinieren wir einfache Formen, wie Kugeln und Zylinder miteinander.

Kugeln herstellen

Bälle werden in der Geometrie und der 3D-Programmierung als Kugeln (oder mathematisch korrekt als Sphären) bezeichnet. Es gibt zwei Möglichkeiten, die Form einer Kugel in JavaScript zu kontrollieren.

Größe: SphereGeometry(100)

Zunächst einmal können wir eine Kugel kontrollieren, indem wir beschreiben, wie groß sie ist. Wir schufen einen Ball, dessen Radius `100` war, als wir `new THREE.SphereGeometry(100)` sagten. Was passiert, wenn du den Radius auf `250` änderst?

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

❶ An dieser Stelle müsstest du die Größe der Kugel ändern.

Sie sollte jetzt viel größer werden:

Abbildung 1-6 ▶
Die Kugel ist gewachsen.

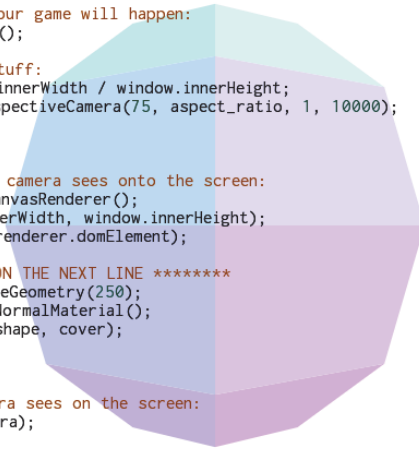
```
<script>
  // This is where stuff in our game will happen:
  var scene = new THREE.Scene();

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

  // This will draw what the camera sees onto the screen:
  var renderer = new THREE.CanvasRenderer();
  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();
  var ball = new THREE.Mesh(shape, cover);
  scene.add(ball);


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

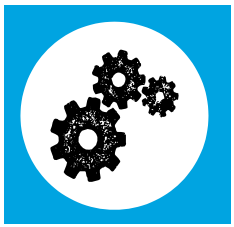


Was passiert, wenn du die 250 zu 10 änderst? Wie du sicher erraten hast, wird sie viel kleiner. Das ist also eine Möglichkeit, um die Form einer Kugel zu kontrollieren. Welche andere Möglichkeit hast du?

Nicht klobig: SphereGeometry(100, 20, 15)

Wenn du auf den Hide Code-Button im ICE klickst, bemerkst du sicher, dass unsere Kugel eigentlich kein *wirklich* glatter Ball ist:

Abbildung 1-7 ▶
Eigentlich keine Kugel, sondern ein kugelförmiges Gebilde aus Flächen



Du kannst den Code ganz leicht ein- und ausblenden

Wenn du in der oberen rechten Ecke des ICE-Fensters auf den weißen Hide Code-Button klickst, siehst du nur den Spielbereich und die Objekte im Spiel. So wirst du in späteren Kapiteln auch die Spiele bedienen. Um den Code wieder hervorzuzaubern, klickst du auf den weißen Show Code-Button im ICE Code Editor.

Computer sind nicht in der Lage, tatsächlich einen Ball herzustellen. Stattdessen tun sie nur so, indem sie einen Haufen Quadrate

(und manchmal auch Dreiecke) so zusammensetzen, dass das Ganze dann aussieht wie ein Ball. Normalerweise erhalten wir die richtige Anzahl an *Segmenten*, sodass es ähnlich genug wirkt.

Manchmal aber möchten wir, dass der Ball ein bisschen glatter wirkt. Dazu fügen wir auf der `SphereGeometry()`-Zeile zusätzliche Werte hinzu:

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

❶ Die erste Zahl ist die Größe, die zweite Zahl ist die Anzahl der Segmente um die Kugel herum, und die dritte Zahl ist die Anzahl der Segmente nach oben und unten.

Dies sollte eine Kugel ergeben, die viel glatter ist:

◀ Abbildung 1-8
So sieht es schon viel besser aus.

Spiel doch noch ein bisschen mit diesen Zahlen herum. Du lernst gerade eine ganze Menge, und durch ein wenig Experimentieren wirst du noch mehr erfahren!

Ändere die Segmentierung nur, wenn Du unbedingt musst

Die Anzahl der Segmente, die wir erhalten, ohne dass wir `SphereGeometry` anweisen, mehr zu benutzen, ist vielleicht nicht so toll, aber du solltest sie nur ändern, wenn du unbedingt musst. Je mehr Segmente in einer Form sind, desto schwerer muss der Computer arbeiten, um sie zu zeichnen. Wie du später sehen wirst, ist es meist einfacher für den Computer, etwas glatt wirken zu lassen, wenn du eine andere Hülle für die Form wählst.

Wenn du weiter machen möchtest, schiebe den Ball aus dem Weg, indem du seine Position festlegst:

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

❶ ball.position.set(-250,250,-250);
```

❶ Die drei Zahlen verschieben den Ball nach links, nach oben und nach hinten. Kümmere dich im Augenblick nicht darum, was diese Zahlen genau machen – wir reden über Positionen, wenn wir in Kapitel 3, *Projekt: Einen Avatar herstellen*, damit beginnen, Spielecharaktere zu bauen.

Mit der Würfelform Kisten herstellen

Als Nächstes erzeugen wir einen Würfel, also im Prinzip eine Kiste. Es gibt drei Möglichkeiten, die Form eines Würfels zu ändern: Du kannst ihn in der Breite, in der Höhe und in der Tiefe ändern.

Größe: CubeGeometry(300, 100, 20)

Um eine Kiste herzustellen, schreiben wir unter das, was wir für unseren Ball benutzt haben, weiteren JavaScript-Code. Tippe Folgendes ein:

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

Wenn du alles richtig gemacht hast, solltest du ... richtig, ein Quadrat sehen:

Abbildung 1-9 ▶
Ganz offensichtlich ein Quadrat

Hm, das ist langweilig. Warum sehen wir ein Quadrat statt einer Kiste? Die Antwort lautet: ... weil unsere *Kamera*, also unsere Perspektive, direkt auf eine Seite der Kiste gerichtet ist. Falls wir mehr von der Kiste sehen wollen, müssen wir entweder die Kamera verschieben oder die Kiste drehen. Drehen wir also die Kiste:

```
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 Diese drei Zahlen drehen die Kiste entgegen dem Uhrzeigersinn nach unten und nach rechts.

In diesem Fall drehen wir `0.5` nach unten und `0.5` nach rechts:

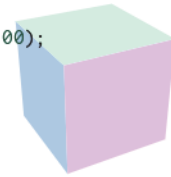
◀ Abbildung 1-10
Die verdeckten Flächen des Würfels werden sichtbar.

Probiere es selbst aus!

An das Drehen von Dingen muss man sich erst gewöhnen, deshalb solltest du ein bisschen mit den Werten herumspielen. Probiere kleinere und größere Zahlen aus. Eine volle Drehung erreichst du mit `6.3` (wir reden später über diese Zahl). Versuche einmal, zwei der Zahlen auf 0 und eine andere auf `0.1`, dann auf `0.25` und schließlich auf `0.5` zu setzen. Wenn du die Zahlen schnell genug änderst, dann ist das fast so, als würde der Würfel herumwirbeln!

Mit einer Drehung von `(0.5, 0.5, 0)` müsste der Würfel so weit herumgedreht worden sein, dass man tatsächlich einen Würfel erkennen kann:

```
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);
```



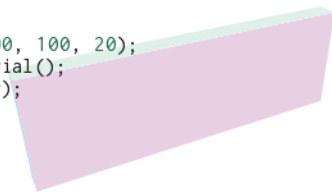
◀ Abbildung 1-11
So sieht der Würfel im Code Editor aus.

Die Seiten der Kiste müssen nicht alle gleich groß sein. Unsere Kiste ist momentan 100 breit (von links nach rechts), 100 hoch (von oben nach unten) und 100 tief (von vorn nach hinten). Ändern wir sie so, dass sie 300 breit, 100 hoch und nur 20 tief ist:

```
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);
```

Das sollte jetzt etwa so aussehen:

```
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);
```



◀ Abbildung 1-12
Aus dem Würfel ist jetzt eine langgezogene Kiste geworden.

Probiere noch einige andere Werte aus, um ein Gefühl dafür zu bekommen.

Ob du es glaubst oder nicht, du weißt jetzt schon eine ganze Menge über JavaScript und die 3D-Programmierung. Natürlich gibt es noch viel mehr zu lernen, aber du kannst jetzt bereits Bälle und Kisten herstellen. Du kannst sie schon verschieben und herumdrehen. Und dazu musstest du nur zehn Zeilen JavaScript-Code schreiben – wirklich toll! Schieben wir unsere Kiste aus dem Weg, damit wir mit noch mehr Formen spielen können:

```
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);
```

Zylinder für alle möglichen Formen

Ein Zylinder, manchmal auch Röhre genannt, ist eine überraschend nützliche Form in der 3D-Programmierung. Denk einmal darüber nach: Zylinder können als Baumstämme, Blechdosen oder auch als Räder benutzt werden ... Wusstest du, dass man mit Zylindern Kegel, Tannenbäume und sogar Pyramiden herstellen kann? Und so geht's!

Größe: CylinderGeometry(20, 20, 100)

Tippe die folgenden Zeilen unter dem Kistencode ein, um einen Zylinder herzustellen:

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

Wenn du das ein bisschen drehst (du erinnerst dich aus dem letzten Abschnitt noch daran, oder?), siehst du vermutlich so etwas Ähnliches (siehe Abbildung 1-13).

Mach dir keine Sorgen, solltest du nicht mehr wissen, wie man die Röhre dreht. Tippe einfach nach der Zeile mit `scene.add(tube)` dies hier ein:

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

Beim Herstellen eines Zylinders beschreiben die ersten beiden Zahlen, wie groß der Zylinder oben und unten ist. Die letzte Zahl gibt die Höhe des Zylinders an. Unser Zylinder ist also oben und unten jeweils 20 groß. Außerdem ist er 100 hoch.

Abbildung 1-13
Siehe da, ein Zylinder!

Was passiert, wenn du die ersten beiden Werte auf 100 und die letzte Zahl auf 20 änderst? Was passiert, wenn du die Spitze auf 1, das Unterteil auf 100 und die Höhe auf 100 stellst?

Probiere es selbst aus

Spiele mit diesen Zahlen ein bisschen herum und erlebe selbst, was du herstellen kannst!

Was hast du herausgefunden?

Ein flacher Zylinder ist eine Scheibe:

```
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);
```

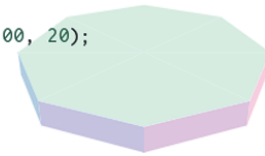


Abbildung 1-14
Die Welt ist eine Scheibe. ;-)

Und ein Zylinder, der entweder eine Spitze oder ein Unterteil mit der Größe 1 hat, ist ein Kegel:

```
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);
```

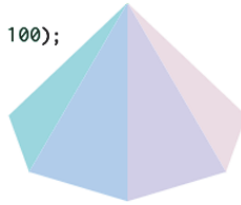


Abbildung 1-15
Ein Zirkuszelt?

Es sollte klar sein, dass du mit Zylindern viel erreichen kannst, allerdings haben wir bisher gar nicht alles gesehen. Ein Trick ist noch übrig.

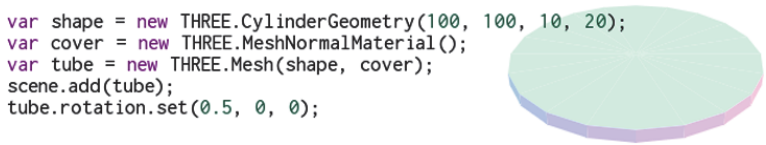
Pyramiden: CylinderGeometry(1, 100, 100, 4)

Hast du gemerkt, dass Zylinder klobig aussehen? Es sollte dich deshalb nicht überraschen, dass man die Grobheit von Zylindern kontrollieren kann. Wenn du zum Beispiel die Anzahl der Segmente bei der Scheibe auf 20 setzt, so wie hier:

```
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);
```

dann müsstest du so etwas sehen:

Abbildung 1-16 ▶
Ein ganz flacher Zylinder mit einem stark geglätteten Rand



Genau wie bei den Kugeln solltest du nur dann viele Segmente benutzen, wenn du das wirklich, wirklich brauchst.

Kannst du dir vorstellen, wie man das in eine Pyramide verwandeln könnte? Du hast schon alle Hinweise, die du brauchst.

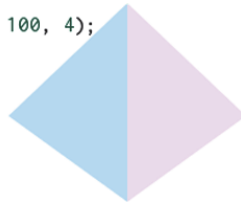
Probiere es selbst aus!

Spiele mit unterschiedlichen Zahlen herum und schaue dir an, was passiert!

Hast du es herausbekommen? Keine Panik, falls nicht. Was wir machen, ist wirklich ziemlich raffiniert.

Du musst nämlich die Anzahl der Segmente, aus denen du einen Kegel herstellst, *verringern*. Wenn du die Spitze auf 1, das Unterteil auf 100, die Höhe auf 100 und die Anzahl der Segmente auf 4 setzt, bekommst du das:

```
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);
```

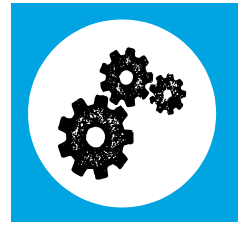


◀ Abbildung 1-17
Eine Pyramide

Es kommt dir vielleicht wie Schummelei vor, auf diese Weise eine Pyramide herzustellen, aber das bringt uns zu einem sehr wichtigen Tipp für alle Programmierungen:

Schummle, wann immer das möglich ist

Im wirklichen Leben solltest du nicht schummeln, aber bei der Programmierung – vor allem bei der 3D-Programmierung – solltest du immer nach einfacheren Methoden suchen, um etwas zu tun. Selbst wenn es eine *übliche* Methode für etwas gibt, könnte es einen *besseren* Weg geben.



Du warst bis hierher ganz klasse. Schiebe die Röhre genau wie den Würfel und die Kugel aus der Mitte heraus:

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

Wir kommen jetzt zu den letzten beiden Formen in diesem Kapitel.

Mit Ebenen flache Oberflächen bauen

Eine Ebene ist eine flache Oberfläche. Ebenen sind besonders für den Boden nützlich, man kann mit ihnen aber auch gut Türen und Ecken in unseren Spielen kennzeichnen.

PlaneGeometry(100, 100)

Da Ebenen einfach nur flache Quadrate sind, sind sie viel einfacher als die anderen Objekte, die wir bisher gesehen haben. Tippe Folgendes ein:

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

Vergiss nicht die Drehung auf der letzten Zeile. Ebenen sind so dünn, dass du sie vielleicht gar nicht siehst, wenn du direkt auf sie blickst.

Die Zahlen beim Herstellen einer Ebene bezeichnen Breite und Tiefe. Eine Ebene, die 300 breit und 100 tief ist, könnte so aussehen:

Abbildung 1-18 ▶
Eine Ebene

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


Mehr musst du über Ebenen eigentlich gar nicht wissen. Schieb unsere Ebene aus dem Weg:

```
var shape = new THREE.PlaneGeometry(300, 100);
var cover = new THREE.MeshNormalMaterial();
var ground = new THREE.Mesh(shape, cover);
scene.add(ground);
ground.position.set(-250, -250, -250);
```

Kommen wir nun zur allerbesten Form auf der ganzen weiten Welt.

Mit einem Ring einen Donut zeichnen (leider nicht essbar)

Im Kauderwelsch der 3D-Programmierung wird ein Donut auch als *Ring* (oder Torus) bezeichnet. Für den einfachsten Ring müssen wir zwei Werte zuweisen: einen für den Abstand von der Mitte bis zur Außenkante und einen anderen für die Dicke der Röhre.

TorusGeometry(100, 25)

Tippe Folgendes in den ICE ein:

```
var shape = new THREE.TorusGeometry(100, 25);
var cover = new THREE.MeshNormalMaterial();
var donut = new THREE.Mesh(shape, cover);
scene.add(donut);
```

Du solltest einen ziemlich unförmigen Donut sehen, so wie der in Abbildung 1-19.

Inzwischen weißt du vermutlich, wie du den Donut ein bisschen glatter hinbekommen kannst: