jQuery: E-Book und Video auf DVD

€ 6,99

www.ctspecial.de

c't kompakt

Programmieren

DVD

Listings: Win-8-App im Eigenbau

E-Book: jQuery – Das JavaScript-Framework für interaktives Design

Videotutorial
Moderne Webseiten gestalten mit jQuery UI

Visual Studio Express
- for Web 2012
- for Windows 8 2012
- for Windows Desktop 2012
- Team Foundation Server 2012

Webseiten aufpeppen
Upload-Manager · Animationen
C++ im Browser · JavaScript parallel

Grafikkarte ausreizen
Schnelle Grafik mit WebGL
Spaß mit Shadern

Coole Sprachen
Haskell · Smalltalk · D

Kinect am PC
Winken statt klicken
Bewegte Autostereogramme

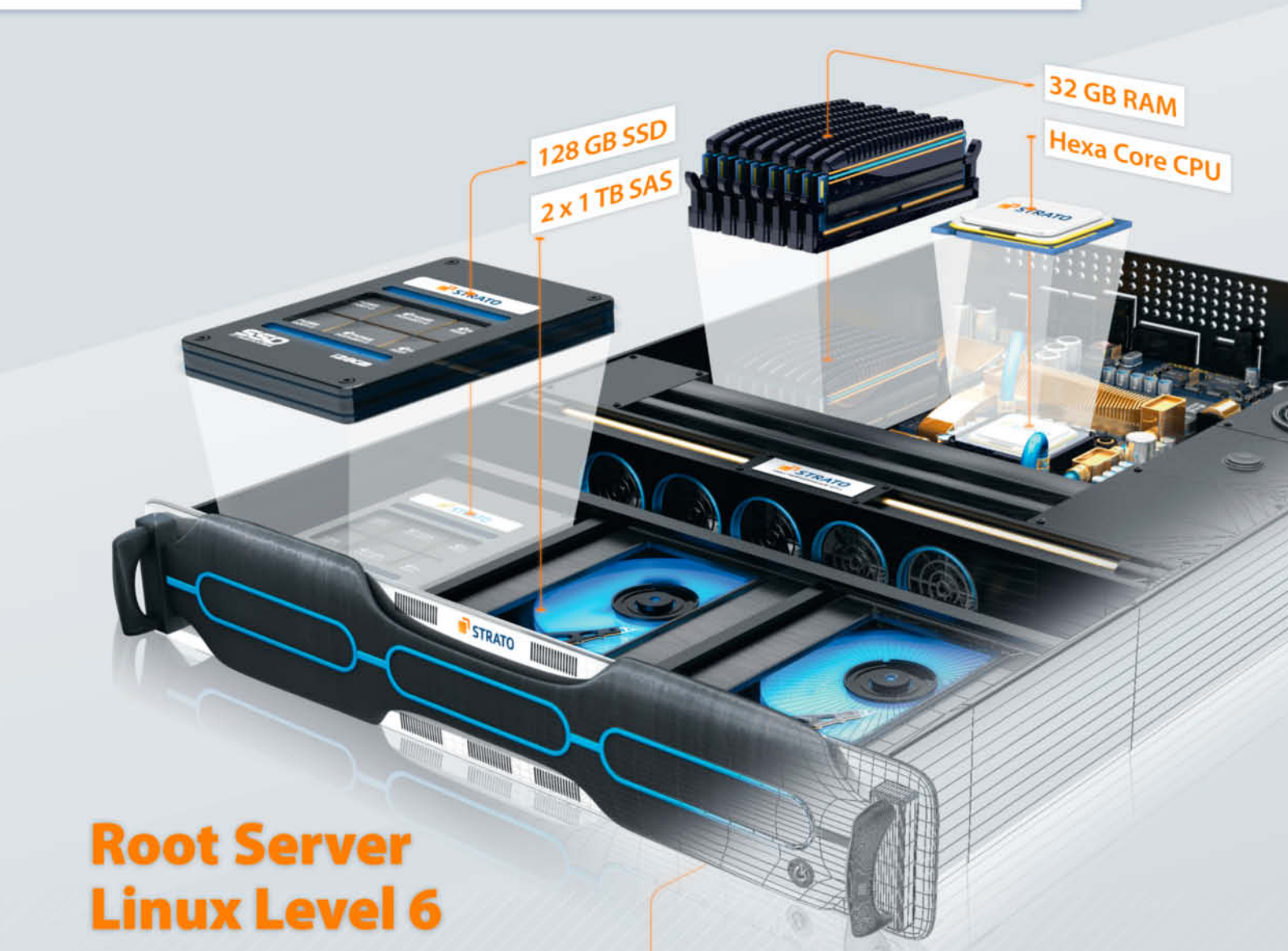
3D-Programmierung
Animationen mit CSS3
SketchUp skripten

Für iPhone, Android und Windows 8

Apps selber bauen

STRATO PRO

PreisRAMbazamba



Root Server Linux Level 6

✓ keine Mindestvertragslaufzeit ✓ keine Einrichtungsgebühr

99,00 €/Mon.*

CPU	Intel Xeon E5-1650 (Hexa Core)
Leistung	6 x 3,2 GHz
RAM	32 GB
HDD	1 x 128 GB SSD und 2 x 1 TB SAS
Traffic	Unlimited

HÖCHSTLEISTUNG ZUM KLEINEN PREIS

Der neue Root Server Level 6 punktet mit einem perfekten Zusammenspiel der Hardware. In jeder Situation steht Höchstleistung zur Verfügung.

Dank SSD-Drive starten Anwendungen und Prozesse blitzschnell. Ladezeiten waren gestern! Große Datenmengen finden auf den SAS-Festplatten (RAID 1) Platz. Für satte Beschleunigung sorgt der Intel Hexa-Core-Prozessor, dem ein großes und schnelles RAM zur Seite steht. Minimal ist hier nur eines – der kleine Preis!

* Aktion bis 30.11.2012. Traffic-Unlimited: Keine zusätzlichen Kosten durch Traffic (bei Traffic-Verbrauch über 1.000 GB/ Monat und danach je weitere 300 GB erfolgt eine Umstellung der Anbindung auf max. 10 MBit/s. Erneute Freischaltung der vollen Bandbreite jeweils kostenlos über den Kundenservicebereich). Preis inkl. MwSt.

Info: **030 – 300 146 111 | strato-pro.de**

Liebe Leserin, lieber Leser,

beinahe täglich wird irgendwo eine neue Programmiersprache erfunden oder eine neue Bibliothek veröffentlicht – kein Mensch kann sie alle beherrschen. Zu bequem sollte man es sich aber als Entwickler in seiner Nische auch nicht machen, sonst läuft man Gefahr, dass das eigene Wissen schon bald nicht mehr gefragt ist. Gerade Programmierern tut es also gut, gelegentlich über den Tellerrand dessen hinauszuschauen, mit dem sie täglich zu tun haben. Und für den, der das Programmieren als Hobby betreibt, gibt es ohnehin nichts Spannenderes, als neue Sprachen oder Techniken zu entdecken.

Mit diesem Heft wollen wir Ihnen einige Anregungen geben, in welche Richtungen Sie Ihren Forscherdrang richten können. Wie wäre es beispielsweise mit neuen Eingabemethoden? Statt Ihre Programme mit der Tastatur oder einer Maus zu bedienen, könnte ihnen der Anwender seine Befehle doch einfach zuwinken – der von der Xbox 360 stammende Eingabesensor Kinect machts möglich.

Wenn Sie nicht gerade zu den eingefleischten Zockern gehören, liegt die Rechenleistung Ihrer Grafikkarte die meiste Zeit brach. Dutzende, wenn nicht Hunderte von Rechenkernen warten darauf, zum Beispiel Fraktale oder andere grafische Spielereien zu berechnen. Mit WebGL funktioniert das sogar im Browser.

Apropos Browser: Gut gemachte, informative und möglichst gar interaktive Animationen sind auf Webseiten immer Hingucker – moderne Browser bringen alles Nötige mit, um sie zum Leben zu erwecken. Die Möglichkeiten reichen von reinem HTML5 und CSS3 über JavaScript zum Zeichnen auf dem Canvas-Element bis hin zum Native Client SDK, mit dem sich Googles Browser Chrome per C++erweitern lässt.

Gerade erst ist Windows 8 erschienen und die ersten Tablets mit Microsofts neuem Betriebssystem sind im Handel. Das Angebot an Apps für die neue Touch-Oberfläche ist aber noch recht überschaubar. Gute Chancen also, eine Marktlücke zu erschließen. Alles, was Sie dazu brauchen, finden Sie auf der Heft-DVD. Auch zu den anderen Themen gibt es dort eine umfangreiche Sammlung an Werkzeugen, damit Sie sofort mit dem Ausprobieren loslegen können.

Viel Spaß dabei!

Hajo Schulz

Apps selber bauen

Webseiten aufpeppen

Kreativ im Web

Immer mehr Web-Anwendungen übernehmen Aufgaben, für die früher Desktop-Programme zuständig waren. Mit der Verbreitung wachsen auch die Ansprüche an grafische Gestaltung und Interaktivität.

Mobile Apps

Anwendungen für Android und iOS entstehen mit wenig Aufwand aus mobilen Websites – mit den richtigen Werkzeugen sogar plattformübergreifend. Auch für Windows 8 gibt's alle Entwickler-Tools kostenlos.

Know-how

Programmieren ist eine kreative Tätigkeit. Spaß macht sie vor allem dann, wenn nicht unbedingt ein vermarktbares Produkt dabei herauskommen muss – zum Beispiel Pixelspielereien auf der Grafikkarte oder ein Programm, das ein dreidimensionales Abbild des Betrachters in Echtzeit berechnet. Mit den richtigen Werkzeugen arbeitet man dabei auch im Team reibungslos zusammen.

3D Programmierung

Coole Sprachen

Exotische Sprachen

Wussten Sie schon, dass es Programmiersprachen gibt, in denen sich die Inhalte von Variablen niemals ändern können? Oder bei denen selbst der Compiler ein Objekt ist, mit dem man zur Laufzeit hantieren kann? Oder in der man trotz automatischer Speicherverwaltung Inline-Assembler-Befehle benutzen kann? Lassen Sie sich überraschen!

Auf der Heft-DVD

Tools und Anwendungen, Entwicklungsumgebungen, Quellcodes sowie PDF-Tutorials und ein Video ergänzen die Artikel zum Heft. Highlights sind ein 494 Seiten starkes PDF über das JavaScript-Framework jQuery sowie ein Video-Training, das zeigt, wie man moderne Webseiten mit jQuery UI gestaltet.

Zum Heft

Download der DVD

Die Heft-DVD steht als Image zum Download unter **www.ct.de/cs1207004** bereit.

DVD-Inhalt

Video-Tutorial

Moderne Webseiten gestalten mit jQuery UI

E-Book (PDF)

jQuery: Das JavaScript-Framework für interaktives Design

Entwicklungsumgebungen und -Tools

Windows

ADT Plugin for Eclipse 20.0.3
Android SDK 20.0.3
Eclipse SDK Classic 4.2
Git 1.7.11
jEdit 4.5.2
Kinect SDK 1.5
Native Client SDK
Notepad++ 6.1.8
PSPad editor 4.5.6
Python 2.7.3
Qt Creator 2.5.2
Squeak 4.3
Visual Studio Express for Web 2012
Visual Studio Express for Windows 8 2012
Visual Studio Express for Windows Desktop 2012
Visual Studio Team Foundation Server Express 2012

Mac OS

ADT Plugin for Eclipse 20.0.3
Android SDK 20.0.3
Eclipse SDK Classic 4.2
Git 1.7.11.3
jEdit 4.5.2
Native Client SDK
Python 2.7.3
Qt Creator 2.5.2
Squeak 4.3

Linux

ADT Plugin for Eclipse 20.0.3
Android SDK 20.0.3
Eclipse SDK Classic 4.2
jEdit 4.5.2
Native Client SDK
Python 2.7.3
Qt Creator 2.5.2
Squeak 4.3

Tools

Windows

7-Zip 9.20
Audacity 2.0.2
Audacity Portable 2.0.2
Gimp 2.8.2
Gimp Portable 2.8.2
XAMPP 1.8.0

Mac OS

Audacity 2.0.2
Gimp 2.8.2
MacPorts 2.1.2
MacPorts 2.1.2 für Lion
MacPorts 2.1.2 für Mountain Lion
MacPorts 2.1.2 für Snow Leopard
XAMPP 1.7.3

Linux

XAMPP 0.9.4

Listings und Skripte

Windows

Ac'tien
Bilderbuch
jqmOrakel
jQuery Mobile
Metronom mit HTML5-Canvas
Pendeluhr
PhoneGap
Ruby-Skripte für SketchUp

Mac OS

Bilderbuch
jqmOrakel
jQuery Mobile
Metronom mit HTML5-Canvas
Pendeluhr
PhoneGap
Ruby-Skripte für SketchUp

Linux

Bilderbuch
jqmOrakel
jQuery Mobile
Metronom mit HTML5-Canvas
Pendeluhr
PhoneGap

Bibliotheken

Windows

Apache Ant 1.8.4
jQuery 1.8.2
jQuery Mobile 1.1.1
PhoneGap 2.0.0
Qt Libraries 4.8.3 (VS 2010)

Mac OS

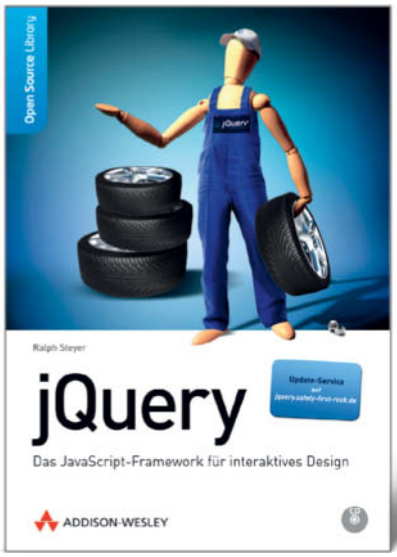
Apache Ant 1.8.4
jQuery 1.8.2
jQuery Mobile 1.1.1
Qt Libraries 4.8.3

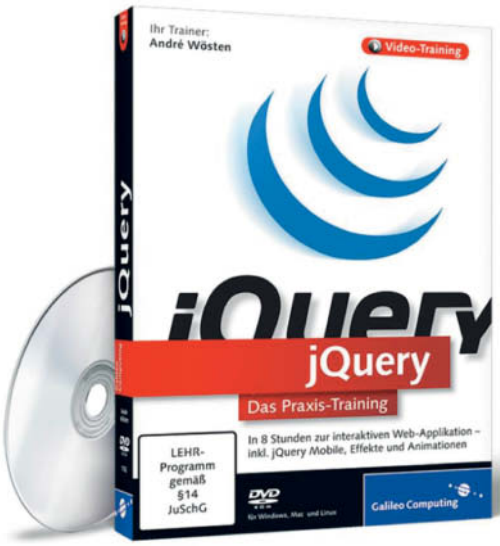
Linux

Apache Ant 1.8.4
jQuery 1.8.2
jQuery Mobile 1.1.1
Qt Libraries 4.8.3 (X11)

DVD-Highlights

Die Heft-DVD ergänzt viele Artikel mit insgesamt rund 4,6 Gigabyte Zusatzmaterial – unter anderem bietet sie komplette Entwicklungsumgebungen sowie Tools und Anwendungen, aber auch PDF- und Video-Tutorials wie etwa ein 494 Seiten starkes PDF über das JavaScript-Framework jQuery sowie ein Video-Training, das zeigt, wie man moderne Webseiten mit jQuery UI gestaltet. Ebenfalls auf DVD: das Listing einer Windows-8-App zur Anzeige von Börsenkursen aus Yahoo Financial.





E-Book (PDF)

jQuery ist ein freies JavaScript-Framework, das 2006 veröffentlicht wurde und das mittlerweile die OpenSource-Community weiterentwickelt. Es bietet unter anderem ein erweitertes Event-System sowie zahlreiche Effekte und Animationen und lässt sich durch eine ganze Reihe freier Plug-ins erweitern. Ein weiterer großer Vorteil von jQuery: Viele bekannte Web-Plattformen auch großer Industrieanbieter unterstützen es mittlerweile – beispielsweise Microsoft in der Entwicklungsumgebung Visual Studio in Verbindung mit dem APS.NET MVC Framework (einige Visual-Studio-Express-Editionen sind ebenfalls auf der Heft-DVD enthalten).

Auf DVD gespeichert ist das 494 Seiten starke Buch „jQuery – Das JavaScript-Framework für interaktives Design“. In dem bei Addison-Wesley erschienenen Buch (ISBN: 978-3-8273-3072-7, Preis: 34,80 Euro) zeigt der Autor Ralph Steyer, wie man jQuery für eigene Web-Applikationen nutzt – angefangen von einer einzelnen Website, die „nur“ um einige wenige Effekte aufgewertet wird, bis hin zu komplexen „Rich Internet Applications“ (RIAs). Dabei vermittelt das Buch in den ersten vier Kapiteln zunächst Grundlagenwissen, anschließend erläutert es, wie jQuery grundsätzlich arbeitet.

Dann erst geht es „ans Eingemachte“: Kapitel 5 bis 10 kümmern sich um die Möglichkeiten zur Auswahl von Objekten im Kontext einer Website, um das Formatieren mit Style Sheets, um die Ereignisbehandlung unter jQuery und um das Thema Ajax.

Last, not least stellen die Kapitel 11 bis 13 – im Buch sind das mehr als 100 Seiten – den jQuery-Aufsatz jQuery UI sowie Plug-ins und jQuery Mobile vor. (keh)

Video-Tutorial

Wie gestaltet man moderne Webseiten mit jQuery UI? Die Antwort gibt das gleichnamige Video-Training auf der DVD zum Heft. Dabei stellt Autor André Wösten zunächst das jQuery UI-Framework kurz vor, dann wendet er sich aber gleich der Praxis zu und zeigt in Kapitel 1.3, wie man Formulare gestalterisch aufwertet und Buttons beziehungsweise Icons einsetzt und mit einem Verhalten ausstattet.

Weitere Punkte erläutert Wösten anhand eines Beispielprojekts – einer rudimentären Oberfläche einer Kontaktbörse. Um sie Step by Step aufzubauen, legt er in Kapitel 1.4 zunächst das Layout als Foto-Mosaik fest und gestaltet im darauffolgenden Abschnitt ein Benutzerprofil-Fenster mithilfe eines jQuery-Dialogfensters. Dabei werden Formulare eingesetzt, die mit einer Akkordeon-Animation ein- und wieder ausgeklappt werden.

Kapitel 1.6 erläutert, wie man via jQuery UI eine Punktebewertung implementiert – und zwar mithilfe eines Schiebereglers, über den man einer Person eine bestimmte Zahl zuweist. Dabei wird der Slider in jQuery programmiert und mit der Werteskala verknüpft. Der vom Benutzer gewählte Wert wird dann dynamisch auf die Webseite geschrieben. Den Abschluss des Miniprojektes bildet dann Kapitel 1.7, das sich um Fortschrittsbalken kümmert. Das rund fünfminütige Video zeigt, wie man die jQuery-Progressbar ausliest, die Werte erhöht und erneut ausgibt.

Insgesamt dauert das jQuery UI-Videotraining rund eineinhalb Stunden. Dabei handelt es sich um ein Einzelkapitel des 8-stündigen Galileo-Videotrainings „jQuery – Das Praxis-Training“ (ISBN 978-3-8362-1762-0, Preis: 39,90 Euro). (keh)

Kreativ im Web

Gut gemachte Webseiten, möglichst mit interaktiven Animationen, sind Eyecatcher. Spezielle Plug-ins muss der Web-Designer dafür nicht mehr einsetzen: Mit HTML5 und CSS3 lassen sich schon ohne Programmierung erstaunliche Effekte erzielen. Hochperformantes JavaScript steht zu Gebote, um das Canvas-Element mit Inhalten zu füllen oder direkt die Grafikkarte anzusteuern. Googles Browser Chrome lässt sich sogar mit C++ erweitern.


F8
F9
Back
Ü
Return
Ö
Ä
Shift
Start
Alt
Ctrl

Oliver Lau

Zeichnen auf dem HTML5-Canvas-Element

Das mit HTML5 eingeführte Canvas-Element ist zum Zeichnen von 2D-Grafiken mittels JavaScript gedacht. Das Beispiel eines Metronoms mit verstellbarer Taktzeit, dessen Pendel man mit der Maus anstupsen und festhalten kann, führt in die Technik ein.

Canvas – so sagt man auf Englisch zu der Leinwand, auf der Maler ihre Kunstwerke in Ölfarben verewigen. Alle modernen Web-Browser kennen ein gleichnamiges HTML5-Element: Man kann darauf geometrische Formen wie Linien, Kurven, Rechtecke oder Kreise zeichnen, aber auch Pixelgrafiken aus PNG-, GIF- oder JPG-Dateien. Im Unterschied zu SVG beschreibt man das Aussehen der Leinwand nicht mit HTML-Tags, sondern bemalt sie mithilfe von JavaScript-Funktionen.

Damit lassen sich interaktive, dynamische Grafiken in den Browser bringen. Das Folgende reißt ein paar Möglichkeiten am Beispiel eines Metronoms an. Den Code in Gestalt von vier Dateien finden Sie auf der Heft-DVD. metronom.html enthält das HTML-Gerüst der Seite und etwas JavaScript für die Steuerelemente, metronom.js den JavaScript-Code, helper.js einige Hilfsfunktionen (etwa `$()` als Abkürzung für `document.getElementById()` oder `attachEvent()` zum Verknüpfen von HTML-Elementen mit Event-Handlern) und metronom.css die Absatz- und Zeichenformate. Die Implementierung vermeidet aus didaktischen Gründen den Einsatz von JavaScript-Bibliotheken wie jQuery und verwendet direkt die Canvas-Funktionen.

Das zentrale Element im HTML-Code ist die Leinwand, die zu Gunsten leichterer Auffindbarkeit via `$()` mit der Kennung „metronom“ versehen wurde:

```
<!DOCTYPE HTML>
<html>
<body onload="onload_handler()"
 onresize="Metronome.onchange()">
<canvas id="metronom"></canvas>
</body>
</html>
```

Diese HTML-Seite beginnt wie alle wohlgeformten HTML5-Dokumente mit der Angabe des Dokumenttyps „HTML“. Die Attribute im `<body>`-Tag sorgen dafür, dass die Funktion `onload_handler()` aufgerufen wird, wenn das Dokument fertig geladen wurde, und `Metronome.onchange()`, wenn der Benutzer die Größe des Fensters geändert hat. Letzteres ist wichtig, weil die Größe des Metronoms daran angepasst werden soll.

In `onload_handler()` wird die Initialisierungsroutine des Metronoms aufgerufen:

```
Metronome.init({
 startStopHandler: startStopCallback,
 countHandler: countCallback,
 freqChangedHandler: freqChangedCallback
});
```

Die drei Parameter sind Callbacks, die der Metronom-Code aufruft, wenn das Metronom gestartet/gestoppt wurde, das Pendel durch die Senkrechte schwingt oder das Pendelgewicht verschoben wurde (womit sich die Schwingfrequenz verändert). Das entkoppelt das GUI-Drumherum (etwa die Schaltflächen für die Sound-Auswahl und die Anzeige der verstrichenen Zeit) von den Interna des Metronoms im zentralen JavaScript-Objekt `Metronome`.

Zum Beispiel empfängt die Funktion `freqChangedCallback()` vom `Metronome`-Objekt eine neu eingestellte Frequenz und stellt sie in den dafür vorgesehenen HTML-Elementen in der rechten oberen Fensterecke dar:

```
function freqChangedCallback(f) {
 $('freq').innerHTML = f + ' min<sup>&minus;1</sup>';
 $('T').innerHTML = 'T = ' + (60/f).toFixed(2) + ' s';
}
```

Die Funktion `Metronome.init()` verknüpft das Canvas-Element mit den Handlern für Mausklicks und -bewegungen:

```
canvas = $('metronom');
attachEvent(canvas, 'mousedown', onmousedown);
attachEvent(canvas, 'mouseup', onmouseup);
attachEvent(canvas, 'mousemove', onmousemove);
```

Damit wird beispielsweise die Funktion `onmousedown()` aufgerufen, wenn der Anwender eine Maustaste auf dem Canvas-Element drückt.

Vorbereiten

Außerdem ermittelt `Metronome.init()` die Schnittstelle zu den Funktionen zum Zeichnen von 2D-Objekten auf einem Canvas-Element durch den Aufruf der Methode `getContext()`:

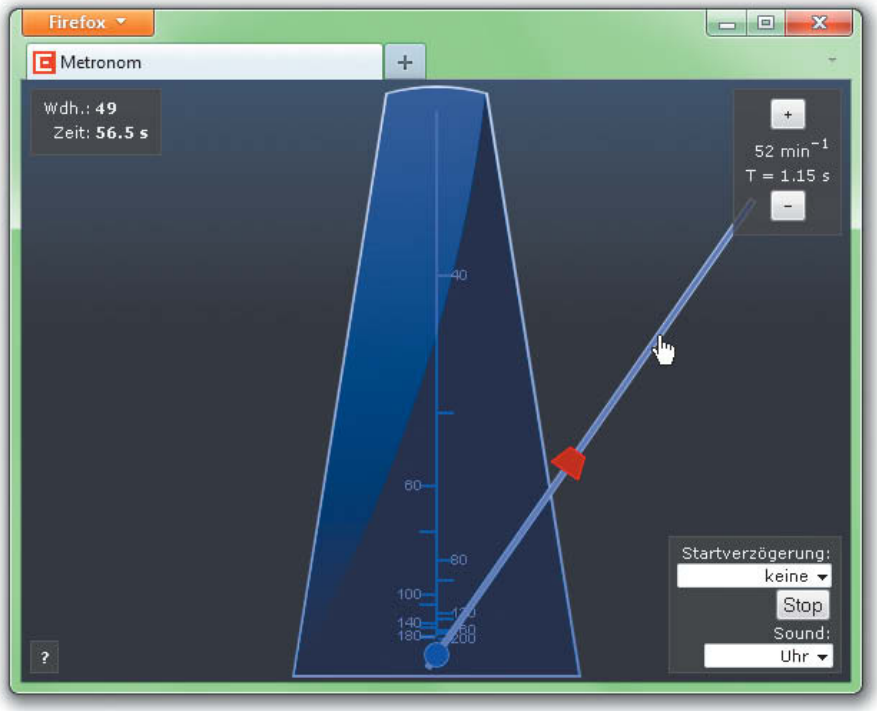
```
ctx = canvas.getContext('2d');
```

Anschließend wird die Funktion `Metronome.onchange()` aufgerufen, die die Fenstermaße in lokalen Variablen speichert und die Canvas-Größe sowie die Ausmaße der zu zeichnenden Elemente daran anpasst:

```
function onchange() {
 width = window.innerWidth;
 height = window.innerHeight;
 canvas.width = width;
 canvas.height = height;
 axis.x = Math.round(width/2);
 axis.y = height-axis.r-11;
 pendulum.length = Math.round(height-axis.r/2-30);
 T = 60/pendulum.frequency;
 weight.dist = T*T/(PI2*PI2)*G;
 updateCanvas();
}
```

Die Felder `x`, `y` und `r` der Variable `axis` bestimmen die Position der Pendelachse und ihren Durchmesser. In `updateCanvas()` werden die Bestandteile des Metronoms in Abhängigkeit vom Achsenmittelpunkt auf das Canvas gezeichnet. Der Browser stellt das Canvas immer dann neu dar, wenn die Ausführung des JavaScript-Codes beendet ist. Aus anderen Sprachen bekannte Befehle wie `update()` oder `invalidate()`, die ein Neuzeichnen erzwingen, gibt es nicht.

Wer die Theorie über Pendelschwingungen nicht mehr in petto hat, kann sie mit der Wikipedia auffrischen [1]. Der Beispielcode implementiert ein reibungsfreies mathematisches Pendel, bei dem sich der Abstand des Gewichts zur Drehachse `weight.dist` direkt aus der Schwingungsdauer `T` ergibt. Die Variable `PI2` repräsentiert das Doppelte der Kreiszahl Pi, `G` die auf das Pendel wirkende Schwerkraft, die für das Beispiel nicht auf die Erdschwerkraft, sondern aus Gründen besserer Darstellbarkeit des Gewichts willkürlich auf 3 (m/s^2) festgelegt wurde.



Die Maus fängt das Pendel des Metronoms ein, setzt es in Bewegung und verschiebt das Gewicht. Mit dem Scrollrad kann man die Skalierung ändern.

Zeichnen

Wie im richtigen Leben die Leinwand wird auch das Canvas von hinten nach vorne bemalt. Der Hintergrund soll in einem vertikalen Farbverlauf von Dunkelblau zu einem dunkleren Blau erscheinen. Dazu definiert man einen Gradienten:

```
var bkgGrad = ctx.createLinearGradient(0, 0, 0, height);
bkgGrad.addColorStop(0, '#373E5C');
bkgGrad.addColorStop(0.4, '#1A1D2B');
```

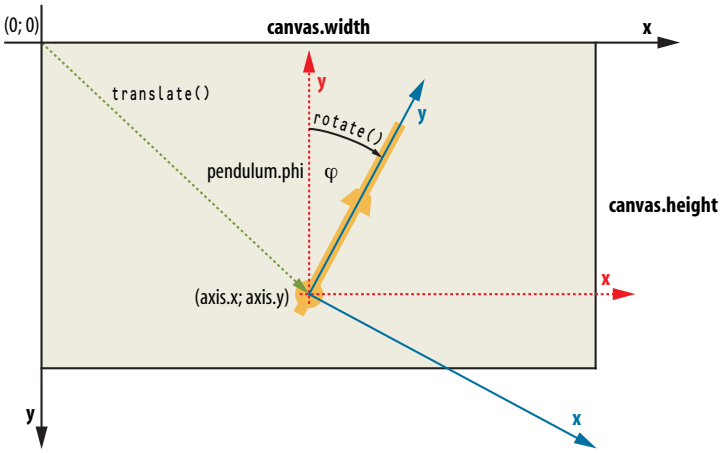
Die Parameter für `ctx.createLinearGradient()` geben die Start- und Endkoordinaten des Verlaufs an. Mit `addColorStop()` bestimmt man die Farbe an einer bestimmten Stelle des Gradienten: Der erste Parameter gibt die Position als Gleitkommazahl an (0 bedeutet am Startpunkt, 1 am Endpunkt, 0.5 zum Beispiel auf halber Strecke dazwischen). Der zweite Parameter ist die Farbe im von HTML/CSS bekannten Format.

Nachdem der Füllstil durch Zuweisen an das Attribut `ctx.fillStyle` auf den Gradienten gesetzt wurde, kann man damit die gesamte Hintergrundfläche füllen:

```
ctx.fillStyle = bkgGrad;
ctx.fillRect(0, 0, width, height);
```

Danach sichert der Aufruf der Methode

```
ctx.save();
```



Der aktuelle Winkel im Bogenmaß steht in der Variablen pendulum.phi. Wichtig: Die y-Koordinaten eines Canvas-Elements wachsen im Unterschied zu mathematischen Koordinatensystemen von oben nach unten.

den Zustand des Canvas-Elements. Damit landen etwa Informationen über Transformationen (Verschieben, Rotieren, …) und die aktuellen Füll- und Stiftstile auf einem Stack. Der Befehl `ctx.restore()` am Ende der Funktion stellt den zuletzt abgespeicherten Zustand wieder her.

Die Koordinaten der Zeichenoperationen sollen sich der Einfachheit halber auf die Drehachse des Pendels beziehen. Dazu verschiebt man einfach den Koordinatensystemursprung an die Stelle auf der Leinwand, an der die Drehachse des Pendels sitzen soll:

```
ctx.translate(axis.x, axis.y);
```

Nun gehts an die Konstruktion des Gehäuses. Bevor man einen neuen Pfad (Vieleck) definiert, muss man die Funktion `ctx.beginPath()` aufrufen. `ctx.moveTo()` hebt den virtuellen Stift an und setzt ihn auf den Startpunkt der nächsten Zeichenoperation. `ctx.lineTo()` zeichnet von diesem Punkt an eine Linie zu den angegebenen Koordinaten.

```
ctx.beginPath();
ctx.fillStyle = 'rgb(8, 24, 80)';
ctx.strokeStyle = '#D0D0F4';
ctx.lineWidth = 2;
ctx.moveTo(-100, axis.r+7);
ctx.lineTo(100, axis.r+7);
ctx.lineTo(35, -pendulum.length-5);
```

Der geschwungene obere Teil des Gehäuses und die Bögen des Glanzeffekts entstehen durch eine quadratische Bézierkurve: Das erste Zahlenpaar von `ctx.quadraticCurveTo()` gibt die Koordinaten des Kontrollpunktes an, das zweite den Zielpunkt:

```
ctx.quadraticCurveTo(0, -pendulum.length-15,
  -35, -pendulum.length-5);
ctx.closePath();
ctx.fill();
ctx.stroke();
```

Kubische Bézierkurven mit zwei Kontrollpunkten zeichnet man übrigens mit `ctx.bezierCurveTo()`.

`ctx.closePath()` schließt die Form mit einer geraden Linie zum Startpunkt ab. `ctx.fill()` füllt sie, `ctx.stroke` umrandet sie in der im Attribut `ctx.lineWidth` festgelegten Strichstärke und der Farbe beziehungsweise dem Farbverlauf in `ctx.strokeStyle`.

Außer mit Vollfarben und Gradienten kann man Flächen auch mit Mustern aus Grafiken pflastern – das Metronom macht aber keinen Gebrauch davon:

```
var img = new Image();
img.src = 'hintergrund.png';
// Muster aus img in x- und y-Richtung wiederholen
ctx.fillStyle = ctx.createPattern(img, 'repeat-xy');
```

Das Zeichnen der Skala für die Frequenzen führt in den Befehl `ctx.fillText()` zur Darstellung von Strings ein:

```
ctx.beginPath();
ctx.lineWidth = 1;
ctx.fillStyle = '#6666ff';
ctx.textBaseline = 'middle';
var tickHeight = 0, f = 100,
scale = -G*pendulum.length/(PI2*PI2);
while (tickHeight < pendulum.length && f > 0) {
 var t = 60/f;
 tickHeight = Math.round(t*t*scale);
 ctx.moveTo(-12, tickHeight);
 ctx.lineTo(-1, tickHeight);
 ctx.textAlign = 'right';
 ctx.fillText(2*f, -10, tickHeight);
 f -= 10;
}
ctx.closePath();
ctx.stroke();
```

Das Listing zeigt die gekürzte Fassung, die die Marken nicht wie im Screenshot wechselseitig, sondern nur auf einer Seite zeichnet. Im ersten Parameter erwartet `ctx.fillText()` den Text, in den beiden folgenden die Koordinaten. Steht das Attribut `ctx.textBaseline` wie im Beispiel auf „middle“, wird die Grundlinie des Textes auf die vertikale Mitte der darzustellenden Zeichenfolge versetzt. Der Wert „alphabetic“ behält sie an gewohnter Stelle, „top“ schiebt sie an die Textober-, „bottom“ an die Textunterkante. Das Attribut `ctx.textAlign` gibt im Unterschied zu `ctx.textBaseline` den horizontalen Punkt der Textausrichtung an. Er kann links („left“), rechts („right“) oder in der Mitte („center“) sitzen.

Transformieren

Während das Gehäuse auf Dauer unverändert bleibt, verändert sich die Lage des Pendels mit der Zeit. Trotzdem wird es so gezeichnet, als ob es wie die statischen Komponenten immer senkrecht steht. Der Trick beim Rotieren um die Drehachse besteht darin, nicht die Zeichenkoordinaten mit trigonometrischen Funktionen zu verändern, sondern das Koordinatensystem zu drehen. Das geschieht mit

```
ctx.rotate(pendulum.phi);
```

Er erwartet als einzigen Parameter den Drehwinkel, allerdings nicht in Grad (360° bilden einen Vollkreis), sondern im Bogenmaß (2π entsprechen 360°). Außer verschieben und rotieren kann man ein Canvas auch skalieren (`scale()`) und über eine Transformationsmatrix verzerren (`transform()`).

Zum Abschluss soll ein Kreis die Drehachse darstellen. Zum Zeichnen von Kreisabschnitten verwendet man den Befehl `ctx.arc()`, der in den ersten beiden Parametern den Mittelpunkt erwartet, im dritten den Radius sowie im vierten und fünften den Start- und Endwinkel des Kreisbogens:

```
ctx.beginPath();
ctx.fillStyle = '#339';
ctx.lineWidth = 2;
ctx.strokeStyle = '#66f';
ctx.arc(0, 0, axis.r, 0, PI2, false);
ctx.closePath();
ctx.stroke();
ctx.fill();
```

Der letzte Parameter bestimmt, in welcher Richtung die Winkelangaben zu lesen sind, bei `true` im Uhrzeigersinn.

Dynamik

Wirft man das Metronom an, wird die Funktion `Metronome.start()` ausgeführt. Sie initialisiert per `setInterval()` einen Timer, der in Abständen von `TimerInterval` Millisekunden die Funktion `tick()` aufruft. Voreingestellt ist ein Intervall von 40 ms, was einer Rate von 25 Bildern pro Sekunde entspricht und damit auf halbwegs modernen Rechnern für eine einigermaßen flüssige Darstellung sorgt:

```
function tick() {
 if (mouse.isDown && isPendulumTouched(mouse.pos)) {
  Metronome.hold();
  pendulum.touching = true;
  pendulum.dragging = true;
 }
 else {
  // Schwingdauer errechnen
  t = 1e-3 * (new Date().getTime() - t0);
  // Pendelauslenkung aus t bestimmen
  pendulum.phi = PhiMax *
   -Math.cos(Math.sqrt(G/weight.dist)*t);
  // Senkrechte überschritten?
  if (pendulum.phi.sign() != pendulum.lastPhi.sign())
   count();
  pendulum.lastPhi = pendulum.phi;
 }
 updateCanvas();
}
```

Ob die Maus auf dem Pendel steht, ermittelt die Funktion `isPendulumTouched()`, die zunächst mit `getPhiByPos()` aus den Mauskoordinaten den Winkel der Maus zur Senkrechten durch die Drehachse ermittelt und diesen Wert dann mit der Stellung des Pendels vergleicht [1, 2]:

```
function getPhiByPos(pos) {
 var x = pos.x-axis.x;
 var y = pos.y-axis.y;
 var z = Math.sqrt(x*x+y*y);
 return Math.asin(x/z);
}
function isPendulumTouched(pos) {
 var mPhi = getPhiByPos(pos);
 return Math.abs(mPhi-pendulum.phi) < Epsilon;
}
```

Außer in `tick()` wird `pendulum.touching` auch in der Metronom-Funktion `onmousemove()` gesetzt, die immer dann ausgeführt wird, wenn sich die Mausposition auf dem Canvas verändert:

```
function onmousemove(event) {
 var pos = getCursorPosition(event);
 pendulum.touching = isPendulumTouched(pos);
 if (pendulum.dragging) {
  pendulum.phi = getPhiByPos(pos);
  if (Math.abs(pendulum.phi) > PhiMax)
    pendulum.phi = PhiMax * pendulum.phi.sign();
 }
 // ...
 updateCanvas();
}
```

Ob die Maus das Pendel berührt, ist wichtig für das optische Feedback, sprich: die Farbe des Pendels, die dadurch etwas heller wird. Ist gleichzeitig die Maustaste gedrückt (siehe `onmousedown`), bedeutet das, dass der Anwender das Pendel „anfasst“; `pendulum. dragging` wird dann `true`. Bewegt man die Maus, bewegt sich das Pendel entsprechend bis zum maximalen Ausschlagwinkel `PhiMax` mit.

Beim Loslassen der Maustaste prüft `onmouseup()`, ob das Pendel festgehalten wurde. Falls es dann nah genug an der senkrechten Stellung ist, rastet es ein, sonst schwingt es los:

```
function onmouseup(event) {
 var pos = getCursorPosition(event);
 var mPhi = getPhiByPos(pos);
 if (pendulum.dragging) {
  if (Math.abs(mPhi) < PhiMax/8)
   snapBack();
  else {
   phi0 = mPhi;
   if (Math.abs(phi0) > PhiMax)
    phi0 = PhiMax * phi0.sign();
   Metronome.start();
  }
 }
 pendulum.dragging = false;
 mouse.isDown = false;
}
```

Das sollte für einen Überblick über die Funktionsweise des Metronoms genügen. Mit den Details machen Sie sich am besten durch einen Blick in den vollständigen (dokumentierten) Code vertraut.

Übrigens

Zurzeit nutzt diese Beispielimplementierung noch eine Flash-Applikation zum Abspielen von Sounds. Zeitgemäßer wäre es allerdings, das mit HTML5 eingeführte Audio-Element dafür zu verwenden. Natürlich ließe sich auch noch an der Optik feilen oder das Pendel könnte sanft zurückgleiten, wenn man es in der Nähe der Senkrechten loslässt oder, oder … (ola) c't

Literatur

[1] Mathematisches Pendel: http://de.wikipedia.org/wiki/Mathematisches_Pendel

[2] Trigonometrie: http://de.wikipedia.org/wiki/Trigonometrie

[3] Satz des Pythagoras: http://de.wikipedia.org/wiki/Satz_des_Pythagoras

www.ct.de/cs1207010

Oliver Lau

Verteiltes Rechnen mit JavaScript

Mit Web Workern lassen sich langwierige Rechenaufgaben erledigen, ohne die Bedienoberfläche einer Web-Applikation einzufrieren. Wie mit Threads kann man damit aufwendige Berechnungen zwecks schnellerer Ausführung parallelisieren.

Unser Ex-Kollege Jörn Loviscach, der mittlerweile als Mathe-Professor tätig ist, hat 1995 einen hübschen Algorithmus ausgetüftelt, der durch Farbraumtransformationen Grafiken berechnet, die an moderne Kunst à la Kandinsky, Macke oder Malewitsch erinnern. Die meisten damit generierten Bilder haben kubistische Züge, und deshalb hat er das Programm Qbist genannt [1]. Der Kasten auf Seite 18 beschreibt den Algorithmus.

Leider geht die seinerzeit für Windows 95/NT und Mac OS entstandene Software auch auf aktuellen Rechnern recht gemächlich zu Werke. Um einiges besser macht es die Portierung als Plug-in für die Open-Source-Bildbearbeitungssoftware Gimp. Mir stellte sich angesichts der in moderne Browser integrierten JavaScript-Compiler die Frage, wie schnell die Bilder wohl in einer Web-Anwendung auf dem Bildschirm erscheinen.

Das ursprüngliche C/C++-Programm ließ sich leicht nach JavaScript übertragen und läuft im

Browser erfreulicherweise um eine Größenordnung schneller. Doch missfiel mir, dass in meinem Multi-Core-PC immer nur ein Kern ausgelastet war, weil JavaScript-Code naturgemäß immer nur in einem Thread ausgeführt wird. Außerdem war das Web-Interface während der Berechnung der Bildvariationen blockiert, nicht einmal die vorgesehenen Loader-Icons rotierten, da der Browser im Hauptthread mit JavaScript-Code beschäftigt war und nicht gleichzeitig auch noch animierte GIFs oder PNGs darstellen konnte. Ein kleiner Trick löste immerhin das letztere Problem:

```
setTimeout(function(){ DrawOnCanvas(data); }, 0);
```

Das verfrachtet die an `setTimeout()` übergebene Funktion in den Hintergrund. Der Timeout von 0 sorgt dafür, dass die Funktion ohne Verzögerung ausgeführt wird. Im Vordergrund kann der Browser damit weiter auf Benutzeraktivitäten reagieren.

Zwei weitere Überlegungen führten zur Lösung beider Probleme:

– Die acht Variationen lassen sich unabhängig voneinander berechnen. Wenn deren Berechnung parallel im Hintergrund stattfinden könnte, würde das die zur Verfügung stehenden CPU-Kerne besser auslasten, im Hauptthread könnte sich der Browser um die Benutzerinteraktion kümmern.
– Bei der Erstellung des großformatigen Bildes kann man die Tatsache ausnutzen, dass sich jedes Pixel unabhängig von anderen berechnen lässt. Daher könnte eine Aufteilung des Gesamtbildes in n Streifen auf einem Rechner mit n Kernen die Ausführungszeit auf etwa 1/n verkürzen, sofern jeder Streifen in einem separaten Thread berechnet wird.

Genau hier kommen die Web Worker ins Spiel. Die Spezifikation beschreibt sie als Programmierschnittstelle, die es Web-Anwendungen erlaubt, JavaScript-Code im Hintergrund parallel zum Hauptthread laufen zu lassen [2, 3]. Der Hauptthread als Aufrufer kommuniziert mit den Workern über Nachrichten (lies: serialisierte Objekte). Concurrency-Probleme wie Race Conditions oder Deadlocks sind damit unmöglich, auch deshalb, weil die Worker keinen Zugriff auf nicht Thread-sichere Funktionen oder das DOM haben.

Den vollständigen Code finden Sie über den Link am Ende des Artikels. Zum Ausprobieren genügt es, die Datei qbist.html im Browser zu öffnen.

Die `Worker`-Programmierschnittstelle ist im Objekt `window.Worker` beheimatet. Ob ein Browser sie unterstützt, ermittelt man durch die Abfrage, ob das Objekt existiert. Im Beispielprogramm geschieht das durch folgenden (gekürzten) Codeschnipsel, wiederzufinden in der Datei helper.js:

```
var Feature = {
 WebWorker: !!window.Worker
};
```

Stehen Web Worker zur Verfügung, enthält die Variable `Feature.WebWorker` nach der Ausführung dieser Zeilen den Wert `true`. Das ist der Fall in aktuellen Ausgaben von Chrome, Firefox, Opera und Safari. Bei Chrome läuft jeder Web Worker in einem eigenen Prozess, bei Firefox und Safari in einem eigenen Thread. Durch Verteilen einer gut parallelisierbaren Aufgabe auf Web Worker erreicht man deshalb bei diesen Browsern eine Geschwindigkeitssteigerung, die sich ungefähr proportional zur Anzahl verfügbarer Prozessorkerne verhält. Bei Opera bleibt dieser angenehme Skalierungseffekt leider aus, weil die Web Worker allesamt im Hauptthread laufen.

In Workern verfügbare Funktionen

Interface	Funktion/Objekt
WindowTimers	setTimeout(), clearTimeout(), setInterval(), clearInterval()
WindowBase64	btoa(), atob()
WorkerUtils	importScripts(), navigator
WorkerGlobalScope	self, location, close()
DedicatedWorkerGlobalScope	postMessage()
SharedWorkerGlobalScope	postMessage(), name, applicationCache

Arbeitsteilung

Ein Web Worker wird wie folgt ins Leben gerufen:

```
mModule[i] = new Worker('worker.js');
```

Das lädt den in der Datei worker.js enthaltenen JavaScript-Code in den Worker und startet ihn sofort. Qbist speichert die Referenz auf den Worker in einem Array. Der Index `i` kann Werte zwischen 0 und `NUM_VARIATIONS` (im Beispiel gleich 8) annehmen und entspricht damit der Nummer der Variation, die aus dem ursprünglichen Bild links oben mit der Nummer 0 entstehen soll.

Die Referenz wird im weiteren Verlauf benötigt, um mit dem Worker kommunizieren zu können. Wie oben bereits angerissen, geschieht das über Nachrichten, die Hauptthread und Worker austauschen. Die `Worker`-Methode `postMessage()` überträgt die Daten an den Worker, gewöhnlich in Gestalt eines Objekts:

```
data = { /* ... */ };
mModule[i].postMessage(data);
```

Das löst im Worker ein `MessageEvent` aus, auf das er in einem dafür eingerichteten Handler reagiert (siehe auch das Listing auf der vorangegangenden Seite):

```
 1 importScripts('colorspace.js', 'drawoncanvas.js');
 2
 3 self.DEBUG = function(msg) {
 4     self.postMessage({ message: 'debug', info: msg });
 5 }
 6
 7 self.addEventListener('message', function(e) {
 8   var d = e.data;
 9   switch (d.command) {
10   case 'paint':
11     self.DrawOnCanvas(d);
12     break;
13   case 'close':
14     self.close();
15     break;
16   default:
17     self.DEBUG('Unbekannter Befehl: ' + d.command);
18     break;
19   };
20 }, false);
```

Im Kern besteht ein Web Worker aus einem Event-Handler, der vom Hauptthread per postMessage() ausgelöste Ereignisse verarbeitet. Die Kommunikation in umgekehrter Richtung verläuft analog dazu.

```
self.addEventListener('message',
  function(/* MessageEvent */ e) {
    var d = e.data;
    // ...
  });
```

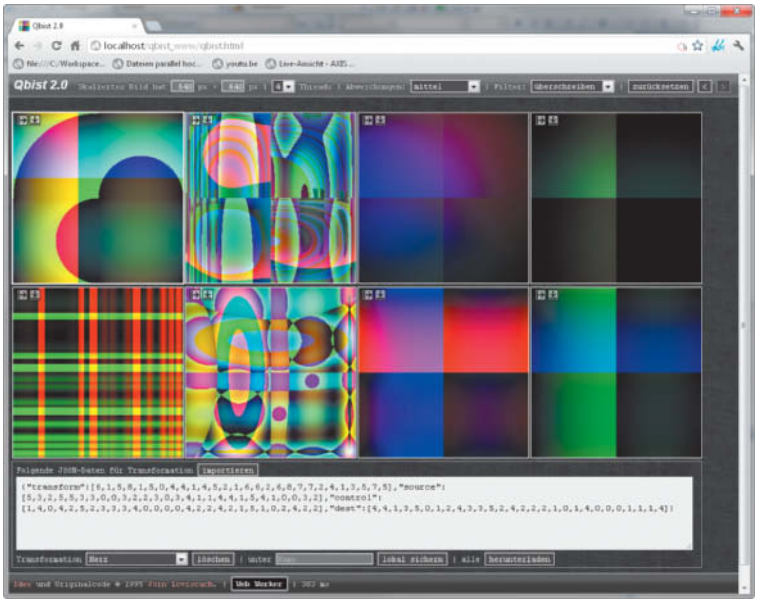
Im Feld `data` des `MessageEvent` befinden sich die per `postMessage()` übermittelten Daten.

Der umgekehrte Weg funktioniert freilich auch, denn irgendwie müssen die Arbeitsresultate eines Workers wieder zurück in den Hauptthread gelangen. Das Objekt `self` des Workers stellt dafür wie im Hauptthread die Funktion `postMessage()` bereit (siehe `self.DEBUG()`). Den Umweg über den Hauptthread zur Ausgabe von Debug-Meldungen muss man gehen, weil ein Worker keinen Zugriff auf `console.log()` hat. Welche Funktionen über `self` zugänglich sind, listet die Tabelle auf Seite 15 auf.

Wie man in der `switch`-Anweisung ab Zeile 9 sieht, ruft er beim Kommando „paint" die Funktion `DrawOnCanvas()` auf, die aus den übergebenen Parametern ein Bild berechnet, das anschließend in einem <`canvas`>-Element erscheinen soll. Wenn aber nun ein Worker keine DOM-Elemente modifizieren kann, wie soll er dann darauf zeichnen? Die Antwort: nicht direkt, sondern über den Umweg über ein `ImageData`-Objekt, das folgender Code aus dem `canvas`-Element extrahiert und als Bestandteil der Nachricht an den Worker sendet:

```
data.imageData = mCanvas[i].getContext('2d')
  .getImageData(0, 0, width, height);
mModule[i].postMessage(data);
```

Die Parameter `width` und `height` bestimmen die Größe des Ausschnitts (im Beispielprogramm 256 × 256 Pixel), die ersten beiden Parameter die Position. Der Ursprung (0, 0) liegt in der linken oberen Ecke, es werden also sämtliche Pixel ausgeschnitten. Das `ImageData`-Objekt lässt sich nun in `DrawOnCanvas()` modifizieren. Im Feld `data` stellt es ein `ByteArray` zur Verfügung, das in jeweils vier aufeinanderfolgenden Elementen die RGBA-Werte der Pixel enthält sowie in `width` und `height` die Breite und Höhe des Bildes:



Qbist 2.0 berechnet aus dem Bild oben links Variationen, indem es zufällig einzelne Werte in den Arrays mit den Transformationsvorschriften ändert.

Wenn man ein Bild wie das von der verunstalteten Angelina Jolie auf eine der Variationen zieht, wird dieses anstelle eines Farbverlaufs als Quelle für die Farbraumtransformationen verwendet.

Bild via Facebook

```
function DrawOnCanvas(p) {
  var imgd = p.imageData;
  var pixel_bits = imgd.data;
  var pp = 0;
  // ...
  for (y = 0; y < imgd.height; ++y) {
    // ...
    for (x = 0; x < imgd.width; ++x) {
      var r = pixel_bits[pp]/256;
      var g = pixel_bits[pp+1]/256;
      var b = pixel_bits[pp+2]/256;
      // Farbraumtransformation ...
      pixel_bits[pp++] = r;
      pixel_bits[pp++] = g;
      pixel_bits[pp++] = b;
      pixel_bits[pp++] = 255;
    }
  }
  self.postMessage({ message: 'painted',
    imageData: imgd,
    variation: p.variation
  });
};
```

Die folgende Zeile aus dem Hauptthread sorgt dafür, dass bei jedem Eintreffen einer per postMessage() ausgesendeten Nachricht die Funktion `handleMessageFromModule()` (siehe unten) aufgerufen wird:

```
mModule[i].addEventListener('message',
  handleMessageFromModule, false);
```

Der Event-Handler `handleMessageFromModule()` im Hauptthread unterscheidet die Nachrichten wie der Worker nach ihrem „Typ":

```
function handleMessageFromModule(e) {
  var d = e.data;
  switch (d.message) {
  case 'painted':
     mCanvas[d.variation].getContext('2d')
       .putImageData(d.imageData, 0, 0);
     break;
  case 'debug':
```

```
    console.log('[DEBUG] ' + d.info);
    break;
 default:
   console.log(d);
   break;
 }
}
```

Der Inhalt von Debug-Nachrichten landet somit in der Debug-Konsole des Browsers, ebenso Nachrichten unbekannten Typs. Was aber am wichtigsten ist: Per `putImageData()` werden die Bilddaten in das dafür vorgesehene `<canvas>`-Element zurückgeschrieben. Damit schließt sich der Kreis.

Fortschritt beobachten

Bis hierhin ging es darum, identische Aufgaben an Worker zu delegieren. Beim Berechnen der hochskalierten Version eines Bildes geht das aber nicht mehr, denn dafür muss eine Aufgabe in Teilaufgaben aufgeteilt und diese müssen delegiert werden. Das geschieht in `DrawTheBigThingOnACanvas()`.

Die Funktion zerlegt das `canvas`-Element in so viele horizontal angeordnete Streifen, wie Worker zur Verfügung stehen. Idealerweise ist die Höhe des `<canvas>`-Elements durch diese Anzahl (`threadCount`) teilbar. Jedem Worker teilt die Funktion im Feld `YO` mit, ab welcher Y-Koordinate er zuständig ist:

```
bigcanvas = document.createElement('canvas');
bigcanvas.width = w;
bigcanvas.height = h;
data = {
  tile: tile,
  YO: YO,
  imageData = bigcanvas.getContext('2d')
    .createImageData(w, h / threadCount),
 // ...
};
worker[tile] = new Worker('worker.js');
worker[tile].postMessage(data);
```

Die Nummer des Ausschnitts im Feld `tile` überträgt der Worker zusammen mit dem Ergebnis zurück an den Hauptthread (Nachricht „painted"), damit dieser abschließend jenem Worker zwecks Ressourcenschonung über die Nachricht „close" mitteilen kann, dass er sich beenden soll. Der berechnete Bildausschnitt wird an der richtigen Position des Ziel-`<canvas>` eingeklebt:

```
worker[tile].addEventListener('message', function(e) {
  var d = e.data;
  switch (d.message) {
  case 'painted':
    bigcanvas.getContext('2d')
      .putImageData(d.imageData, 0, d.YO);
    worker[d.tile].postMessage({ command: 'close' });
```