Höhere Technische Bundeslehranstalt
und Bundesfachschule
im Hermann Fuchs Bundesschulzentrum

HTML5 – Canvas2d

Skriptum

Christian Hanl

15. Februar 2013

1 HTML5 – Canvas

1.1 Was ist ein Canvas?

Das `<canvas>`-Element stellt eine Zeichenfläche („Leinwand“) zur Verfügung, auf die mittels JavaScript dynamische Bitmap-Grafiken gezeichnet werden können.

Es handelt sich also im Prinzip um ein Pixel für Pixel programmierbares `<img>`-Element. [1, S. 353ff.]

1.2 Wofür kann man Canvas verwenden?

Mit dem `<canvas>`-Element lassen sich dynamische Diagramme, Animationen, Interfaces und Bilder erstellen – alles direkt im Browser.

1.3 Das Canvas-Element

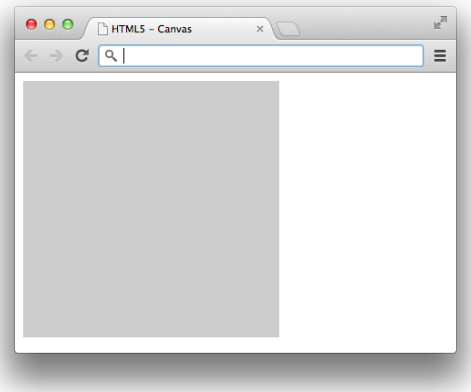
1.3.1 Canvas-Tag

Das `<canvas>`-Element kann ähnlich wie das `<img>`-Element verwendet werden. Mit den Attributen `width` und `height` kann die Größe angepasst werden. Diese Attribute sind optional und können auch über ein StyleSheet gesetzt werden.

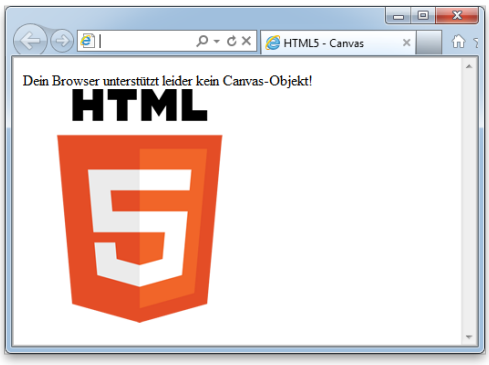
Zwischen dem öffnenden und dem schließenden `<canvas>`-Tag kann der Inhalt angegeben werden, der dargestellt wird, wenn ein Browser das `<canvas>`-Element nicht interpretieren kann.

```
1 <canvas id="my2dCanvas" width="256" height="256">
2     Dein Browser unterstuetzt Canvas leider nicht!<br>
3     <img src="img/HTML5_Logo_256.png" alt="HTML5-Logo" width="256px"/>
4 </canvas>
```

Ein `<canvas>`-Element ist für sich alleine leer und muss per JavaScript gefüllt werden. Um einfach auf das Objekt zugreifen zu können, wurde die id `my2dCanvas` vergeben.



(a) Bei Browsern, die das `canvas`-Element darstellen können, wird der alternative Inhalt nicht angezeigt.



(b) Browser, die das `canvas`-Element nicht darstellen können, zeigen den alternativen Inhalt an.

Abbildung 1: Ein leeres `<canvas>`-Element mit dem grauer Hintergrundfarbe.

1.3.2 Rendering Context

Um auf ein `<canvas>`-Objekt zeichnen zu können, stellt das Objekt eine Schnittstelle zur Verfügung, genannt *Rendering Context*. Dieser *Rendering Context* muss immer angegeben werden, wenn auf die Canvas zugegriffen werden soll.

```
// Referenz zum Canvas und Schnittstelle herstellen
var lCanvas = document.getElementById("my2dCanvas");
var lContext = lCanvas.getContext("2d");
```

1.3.3 Einfache Formen zeichnen

Formen wie Rechtecke oder Kreise zu zeichnen ist relativ einfach. Ein farbiges Rechteck und ein farbiger Kreis wie in Abbildung 2 zu sehen, lassen sich zum Beispiel so erstellen:

```
function drawCanvas ()
{
    //  Referenz zum Canvas und Schnittstelle herstellt
    var lCanvas = document.getElementById("my2dCanvas");
    var lContext = lCanvas.getContext("2d");

    // Ein Rechteck zeichnen
    lContext.fillStyle = "red"; // Zeichenfarbe ist jetzt rot
    lContext.fillRect ( 25, 25, 100, 50 ); // x, y, Breite,  Hoehe

    // Einen Kreis zeichnen
    lContext.beginPath ();  // Ein neuer Pfad beginnt
    lContext.fillStyle = "green";   // Zeichenfarbe ist jetzt gruen
    lContext.arc ( 180, 50, 25, 0, 2 * Math.PI, true); // x, y, Radius,
       Startwinkel, Endwinkel, Uhrzeigersinn
    lContext.fill();        // Pfad ausfuellen
    lContext.closePath ();  // Pfad schliessen
}
```

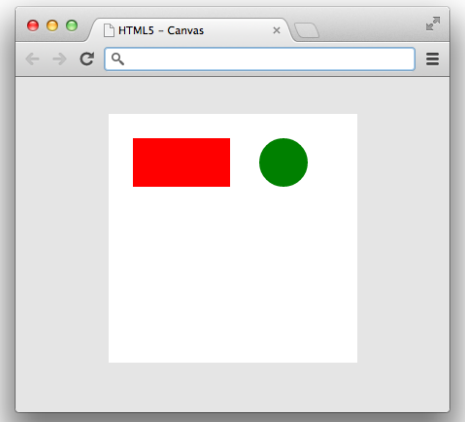
1.3.4 Einbinden von Bitmap-Grafiken

Um Bitmap-Grafiken einzubinden muss ein `Image`-Objekt angelegt werden. Mit dem Befehl `drawImage` wird die Grafik dann in die Canvas gezeichnet. Die beiden Parameter Breite und Höhe sind optional. Werden sie nicht angegeben, so wird die Grafik mit ihren tatsächlichen Abmessungen eingebunden:

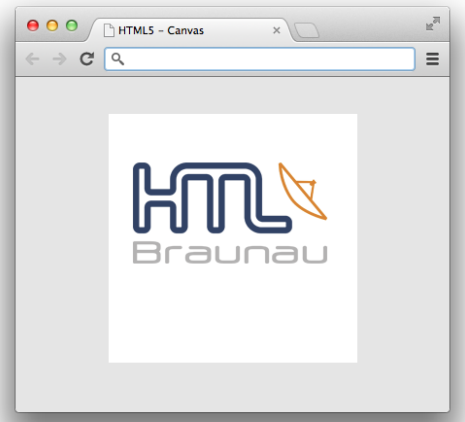
```
// Image-Objekt anlegen
lImage = new Image();

// Bildquelle angeben
lImage.src = "./logo.png"

// Bitmap-Grafik zeichnen
lContext.drawImage  ( lImage, 50, 50, 200, 100 );
                // Bildobjekt, x, y, Breite, Hoehe
```



(a) Grafische Primitive



(b) Image-Objekt

Abbildung 2: Einfache Objekte, die per JavaScript in die Canvas gezeichnet wurde.

1.3.5 Farbangaben

Für alle Farbangaben Strings verwendet werden, die CSS3-Farbangaben [2] entsprechen. Alles was in einem Stylesheet stehen würde, funktioniert auch beim `canvas`-Element.

```
// Angabe mit Schluesselwoerter
lContext.fillStyle = "red";

// RGB mit numerischen Farbwerten
lContext.fillStyle = "#00ff00";              // #rrggbb
lContext.fillStyle = "rgb(255,0,0)";         // Integer
lContext.fillStyle = "rgb(100%,0,0)";        // Prozent
lContext.fillStyle = "rgba(0,0,255,0.5)";    // Mit Transparenz
lContext.fillStyle = "rgba(0,0,100%,0.5)";   // Mit Transparenz

// Farbton, Luminanz und Saettigung (hue, saturation, lightness)
hsl(0, 100%, 50%) }          // rot
hsl(120, 100%, 50%) }        // gruen
hsl(120, 100%, 25%) }        // dunkelgruen
hsl(120, 100%, 75%) }        // hellgruen
```

1.3.6 Löschen des Canvas

Um Teilbereiche oder die gesamte Zeichenfläche zu löschen kann der Befehl `clearRect()` verwendet werden:

```
// Loeschen eines Teilbereichs des Canvas-Objekts
lContext.clearRect( 10, 10, 50, 50 );   // x, y, Breite,  Hoehe

// Loeschen der gesamten Zeicheflaeche
lContext.clearRect( 0, 0, lContext.canvas.width, lContext.canvas.height );
```

Einen kompletten Reset des gesamten Canvas erhält man, wenn man die Höhe oder die Breite des `<canvas>`-Objekts neu setzt. Dabei muss sich die Größe nicht ändern:

```
// Kompletter Reset des Canvas-Elements
lContext.canvas.height = lContext.canvas.height;
```

1.4 Das Koordinatensystem und die Transformationsmatrix

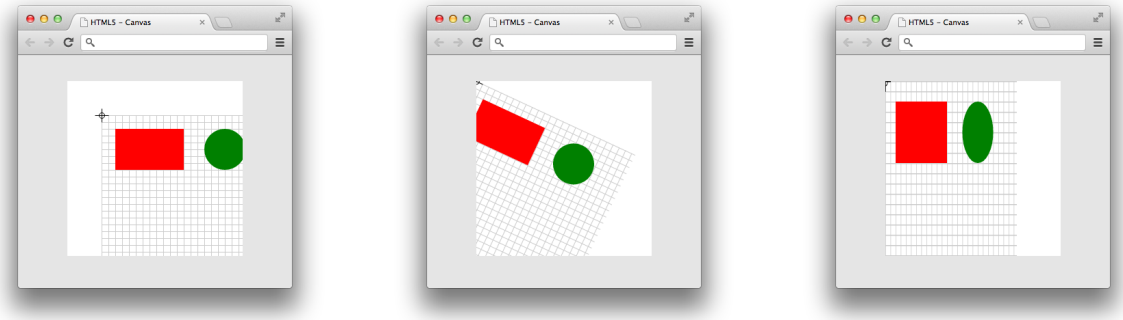
Das Koordinatensystem für das `<canvas>`-Objekt kann man sich als Gitternetz vorstellen. Dieses Gitternetz liegt unsichtbar über dem Bitmap und wird durch die *Transformationsmatrix* abgebildet.

Zu Beginn entspricht die *Transformationsmatrix* der *Einheitsmatrix* und weist folgende Eigenschaften auf:

- Der Nullpunkte `(x=0, y=0)` liegt in der linken oberen Ecke des `<canvas>`-Objekts.
- Die Breite und die Höhe des Koordinatensystems stimmen mit der Breite und der Höhe des `<canvas>`-Objekts überein.

Die *Transformationsmatrix* und somit das Gitternetz können durch **Skalieren**, **Translieren** und **Rotieren** verändert werden. Werden danach Objekte gezeichnet, so werden diese Transformationen auf die Objekte angewendet.

1.4.1 Translation, Rotation, Skalierung



(a) Translation (b) Rotation (c) Skalierung

Abbildung 3: Tranformationen verändern das Gitternetz, mit dem die Zeichenfläche beschrieben wird.

Translation
Eine Translation verschiebt den Ursprung der Zeichenfläche.

```
// Verschieben des Nullpunkts des Gitternetzes um 50 x 50 Einheiten
lContext.translate ( 50, 50 );
```

Rotation
Eine Rotation dreht die Zeichenfläche.

```
// Rotieren des Gitternetzes um 25 Grad
lContext.rotate ( 25 * Math.PI / 180 );
```

Skalierung
Eine Skalierung verzerrt die Zeichenfläche.

```
// Verzerren des Gitternetzes: Faktor 0.75 fuer X, 1.5 fuer Y
lContext.scale ( .75, 1.5 );
```

1.4.2 Zusammenspiel mehrerer Transformationen

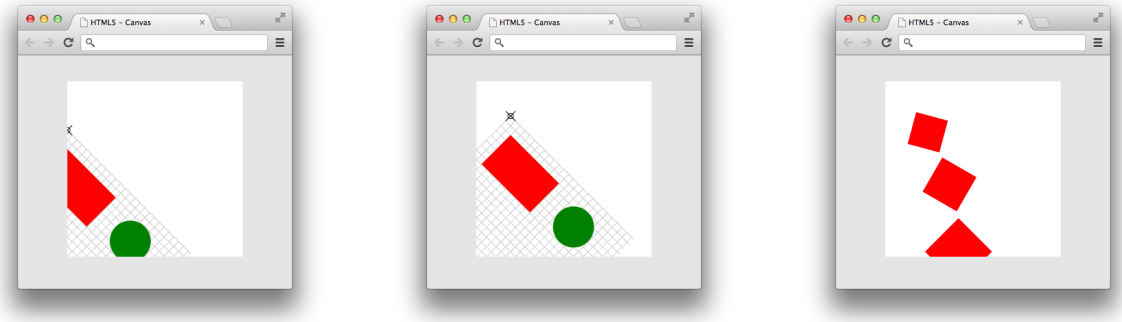
Das Koordinatensystem kann mehrfach transformiert werden. Soll zum Beispiel ein Objekt gedreht und verschoben werden, so müssen die Transformationen `rotate` und `translate` angewendet werden. Diese Transformationen sind dann für alle Objekte gültig, die danach gezeichnet werden.

Soll danach das Koordinatensystem wieder zurückgesetzt werden, so müssen die angewandten Transformationen wieder rückgängig gemacht werden. Das passiert, indem die Transformationen in umgekehrter Reihenfolge angewendet werden:

```
1 lContext.rotate ( 25 * Math.PI / 180 );      // Drehen des Gitternetzes
2 lContext.translate ( 50, 50 );               // Verschieben des Gitternetzes
3 lContext.fillRect ( 25, 25, 100, 50 );       // Rechteck schraeg zeichnen
4 lContext.translate ( -50, -50 );             // Zurueck schieben
5 lContext.rotate ( -25 * Math.PI / 180 );     // Zurueck drehen
```

Diese Vorgehensweise kann recht schnell sehr umständlich werden. Eine andere Möglichkeit ist es die Transformationsmatrix zu speichern und wieder zurückzusetzen:

```
1 lContext.save (); // Speicher der Transformationsmatrix
2
3 lContext.rotate ( 25 * Math.PI / 180 );      // Rotieren des Gitternetzes
4 lContext.translate ( 50, 50 );               // Translieren des Gitternetzes
5 lContext.fillRect ( 25, 25, 100, 50 );       // Rechteck schraeg zeichnen
6
7 lContext.restore (); // Wiederherstellen der Transformationsmatrix
8
9 lContext.fillRect ( 25, 25, 100, 50 ); // Rechteck wieder gerade zeichnen
```



(a) Rotation um 45 Grad mit anschließender Translation um 50/50 Einheiten.

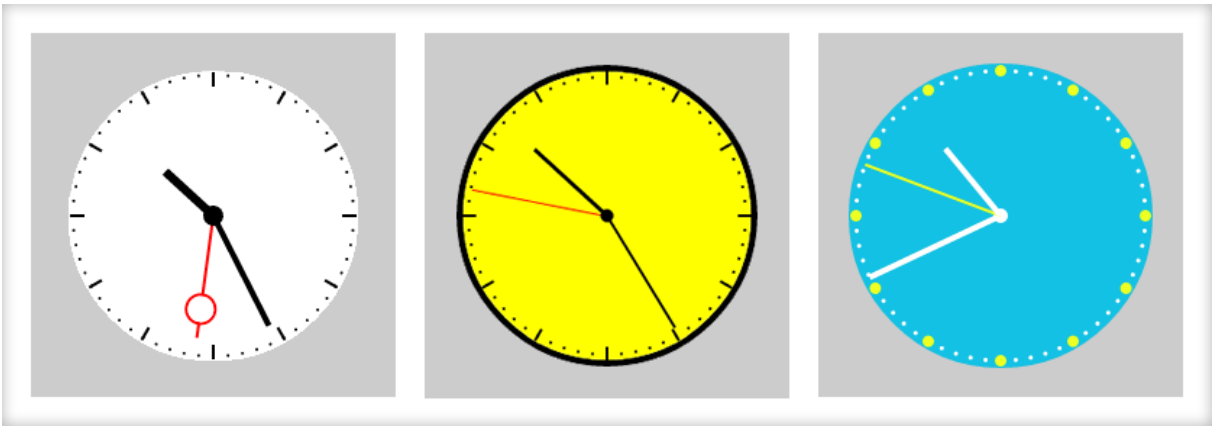
(b) Translation um 50/50 Einheiten mit anschließender Rotation um 45 Grad.

(c) Mehrfache Translation und Rotation mit anschließendem Zeichnen eines Quadrats.

Abbildung 4: Die Reihenfolge der Transformationen wirkt sich auf die Transformationsmatrix aus.

1.5 Übungsbeispiel - Analoge Uhr

Aufgabe:
Es soll eine analoge Uhr erstellt werden, die die aktuelle Zeit ausgibt:



Anleitung:
Zuerst muss mit `window.setInterval(...)` eine Callback-Funktion definiert werden. Diese Funktion soll laufend aufgerufen werden und die Uhr aktualisieren:

- Zeichenfläche löschen
- Ziffernblatt zeichnen
- Auslesen der Systemzeit mit dem Objekt
  ```
  var lDate = new Date();
  var lMin = lDate.getMinutes();
  ...
  ```
- Zeiger zeichnen

Optionale Erweiterungen:

- Lege einen Button im HTML an, mit dem man zwischen Sommerzeit und Winterzeit umschalten kann.
- Es soll eine analoge Stopp-Uhr umgesetzt werden. Erstelle dazu Buttons im HTML für *Start*, *Rundenzeit*, *Stopp* und *Reset*. Auf den Buttons liegen `onclick()`-Events, die JavaScript-Funktionen aufrufen, die dann auf die Canvas zugreifen.

Literatur

[1] P. Kröner, *HTML5*. Open Source Press, 2011.

[2] W3C, "Css color module level 3." Online, 2011. http://www.w3.org/TR/css3-color/, Letzer Besuch: 11.02.2012.