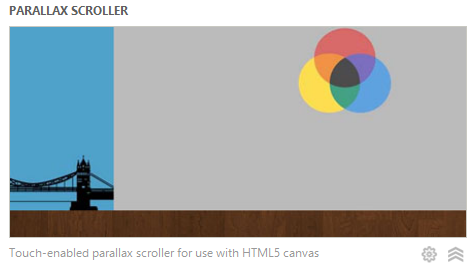
|  |
| --- |
| HTML5 Elements Cookbook |
| H5E Experiment: Parallax Scrolling |
| DRAFT  Published 15 July, 2011 |
| |  | | --- | |  | | H5E Scout Team  Windows Web Partners  Microsoft Corporation  Microsoft Confidential | |

# Summary

Parallax scrolling is a visual technique that suggests depth of an image through motion-based illusion. When the foreground image is scrolled, the image in the background moves at a slower rate, creating the illusion of depth, with the foreground and background images appearing to be on separate planes. Parallax scrolling enhances aspirational experience by simulating a real-world behavior.



## Scope

This document describes an experiment conducted by the H5E scout team using Clarity Consulting. Our objective is to test the limits of HTML5 solving real-world partner questions. This document assumes an existing knowledge of JavaScript and jQuery. This document does not supersede any requirements or instructions provided by the IE team.

## Keywords

HTML5, Canvas, JavaScript, Sprites, IE9, Parallax Scrolling, scroll techniques, scroll speed

## Contact us

To contact us for questions or support, please email Chewy Chong ([ChewyC](mailto:ChewyC?subject=HTML5%20Cookbooks)). Feedback is welcome.

Contents

Summary 2

Scope 2

Keywords 2

Contact us 2

Parallax Scroll Overview 4

Walkthrough 4

How do I build this? 6

Method 1: Canvas 7

Method 2: DOM + JavaScript 10

Conclusions and Recommendations 11

Resources 11

Relevant Web sites and specifications 11

Microsoft Resources 11

Appendix A: About H5E 12

What is an HTML5 Elements Cookbook? 12

Contact us 12

Copyright and trademark information 12

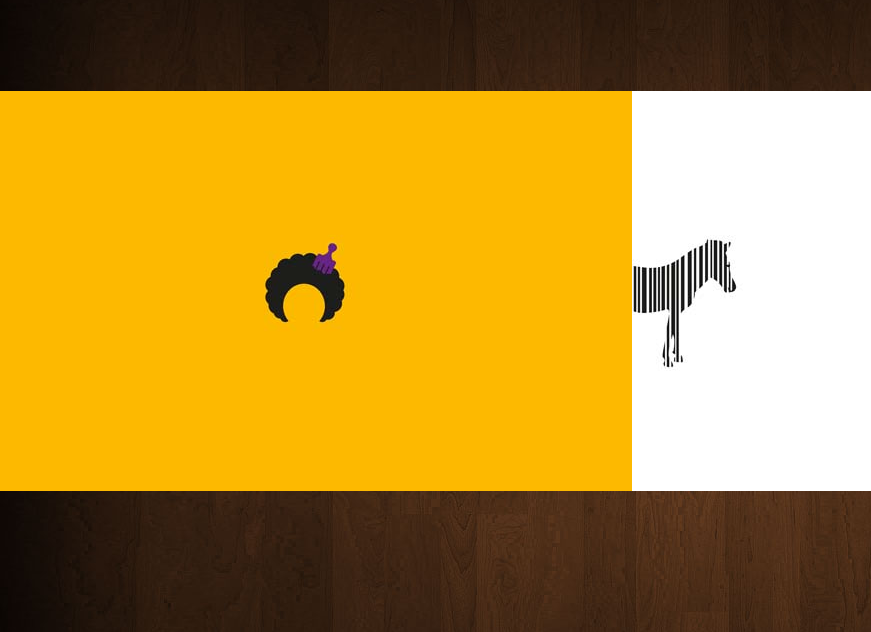
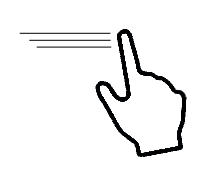
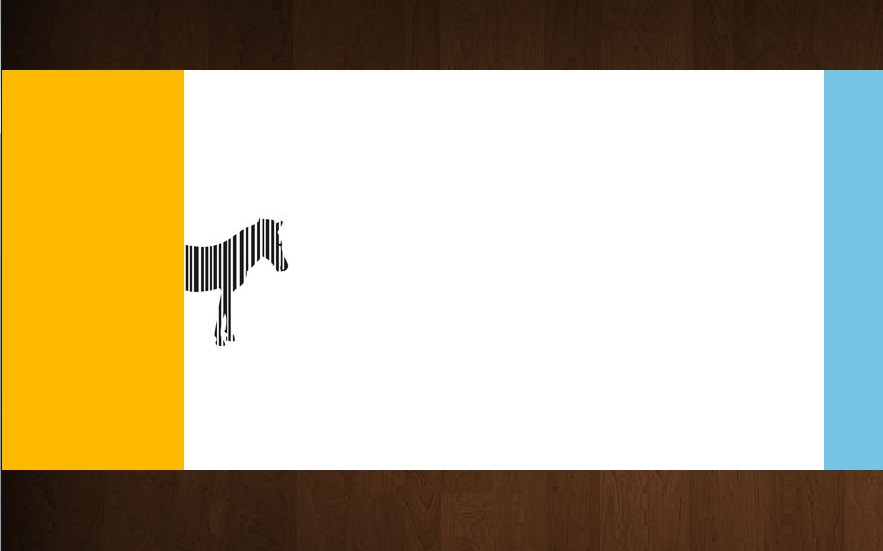
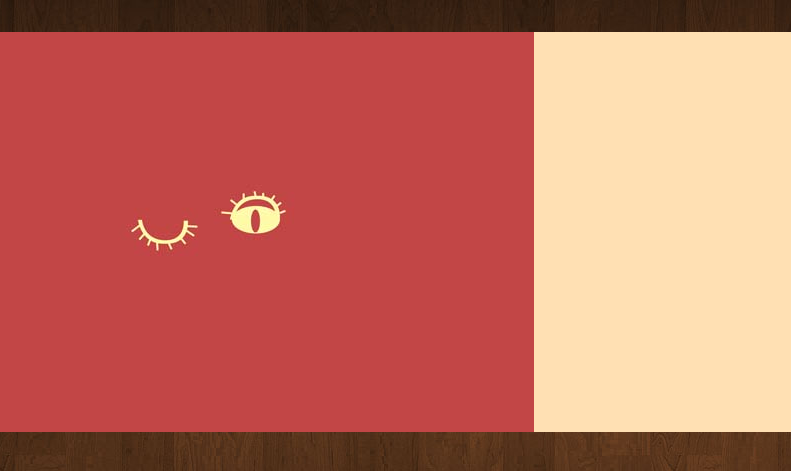
Document Revision History 13

# Parallax Scroll Overview

Parallax scrolling is a technique that is used to provide the sensation of visual depth in applications. When the user scrolls, the foreground and background images appear to be on separate planes that scroll at different speeds.

## Walkthrough

This demo includes the following behavior:

1. The user views the initial screen, showing a foreground and background image.  
   
2. The user scrolls to the right by dragging one of the images to the right.  
   
3. The user drops the image, moving the foreground and background at different rates.  
   

# How do I build this with HTML5?

There are multiple approaches to creating this experience using HTML5-related technologies. In this case, this HTML5 Experiment focuses on the following:

* [Method 1: Canvas](#_[SAMPLE_CONTENT]Method_1:)
* [Method 2: DOM + JavaScript](#_Method_2:_DOM)

***NOTE****: Many of the HTML5 Experiments are still under development. Our initial target is to build prototypes that work on current HTML5-supported browsers and tablet devices, including the iPad. The experiments do not aim for full cross-browser support at this stage, but we will likely build in graceful degradation in future updates.*

To ensure that users have a similar cross-browser experience, the following table describes the compatibility of the solutions in this document:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HTML5 Logo**HTML5 Feature** | IE6.0 | IE7.0 | IE8.0 | IE9.0 | IEPP | Chrome11 | | Chrome12 | Safari4.x | Safari5.x | Firefox3.6 | Firefox4.x | Firefox5.x | Opera11 |
| Canvas |  |  |  |  |  |  | |  |  |  |  |  |  |  |
|  | | | | | | | Full Support | | | | | | |  |
|  | | | | | | | Supported with Shim | | | | | | |  |
|  | | | | | | | No current support | | | | | | |  |

***NOTE****: In general, shims are not incorporated into the HTML5 Experiments at this stage. If a shim or polyfill is required for cross-browser support, see* [*http://browserexperiments.com*](http://browserexperiments.com) *for details on shim implementation.*

## Primary files in this solution

### Source Location

<https://github.com/molant/BrowserExperiments/tree/master/cookbook/3_ParallaxScroller>

### Sample Location

<http://employees.claritycon.com/eklimczak/html5Cookbook/3_ParallaxScroller/>

## Method 1: Canvas

Working with the HTML Canvas can be a difficult task. To reduce the amount of infrastructure necessary to use Canvas, this sample uses EaseJS, a JavaScript library that provides a retained graphics mode for Canvas. EaselJS includes a a full hierarchical display list, a core interaction model, and helper classes to make working with Canvas much easier. You can learn more about EaseJS from the EaselJS website: <http://easeljs.com/>

### Moving the elements

The key to working with Canvas is to remember that your code is responsible for managing everything within the Canvas element. This means that you have to take care of hit testing, animate the elements, etc. This provides a lot of flexibility but when developing a large project it can become really difficult. Most developers choose a library like EaselJS to build prototypes quickly, and where appropriate, for production.

***Note****: For this document, working with Canvas directly (without using an existing library for this purpose) is beyond the scope of this demo. Our goal was to address this aspirational experience using common developer practice, which includes leveraging existing libraries such as EaselJS.*

Canvas can be thought of as similar to a traditional animation surface that does not inherently manage any objects that appear on the screen. Each frame repaints the entire Canvas surface at the interval that you choose. This means that each object or item within the Canvas is repainted each time a new frame is displayed. In practical terms, this means your code must manage the state of anything that appears within the Canvas element, and is repainted in it’s current state each time the frame is painted to the screen.

To work with EaselJS, the first step is to create a **Stage** object where we can add our elements. In our case it is a background and an image container:

this.stage = new Stage(this.canvas, false);

var \_bgImg = new Bitmap(mybackground);

this.stage.addChild(\_bgImg);

var \_imageContainer = new Container();

\_imageContainer.width = this.stage.width;

\_imageContainer.height = this.stage.height;

this.stage.addChild(\_imageContainer);

**Stage**, **Bitmap**, and **Container** are objects provided by EaselJS. These elements allow you to manipulate elements at a higher level, reducing the need to manage every pixel within the canvas surface.

Now that we have our initial frame, the next step is moving the elements. In this experience, the user will drag and drop the images following a horizontal axis. Depending on the device, you will have to subscribe to touch events (on an iPad, for example) or to mouse events (desktop browser).

if (!useTouch) {

$(this.canvas).mousedown(this.OnMouseDown);

$(this.canvas).mouseup(this.OnMouseUp);

$(this.canvas).mousemove(this.OnMouseMove);

} else {

this.canvas.addEventListener('touchstart', this.OnTouchDown, false);

this.canvas.addEventListener('touchmove', this.OnTouchMove, false);

this.canvas.addEventListener('touchend', this.OnTouchEnd, false);

}

The core code in each of those functions is going to be the same for touch and mouse. The only change is the event type and how to access the event information.

For the move event, we need to calculate the delta from the previous position, and then the new x position (we are only working in the x axis). Because we are not working on a relation 1:1 with the mouse/touch movement, we slow the movement down a bit. Your code should also manage the edges of the element to keep the images within bounds.

this.OnMouseMove = function (mouseEvent) {

var newX = mouseEvent.pageX;

var deltaX = newX - \_prevMouseX;

\_prevMouseX = newX;

//Elastic Edge

if ((\_targetX \* -1) > 0 && (\_targetX \* -1) < \_viewPortMax - Application.canvas.width) {

\_targetX += deltaX;

} else {

\_targetX += deltaX / 4;

}

};

After this is complete, the only remaining task is to update the position of the elements. As part of managing Canvas, you redraw the frame with the objects in their current state. Each frame that is rendered to the screen can be thought of as a snapshot of the current state of everything that appears within the Canvas element. To assist with this, there is a standard under development that requests the animation frame rate and then updates, but because it is a draft it uses a vendor prefix (and some older browsers do not support it). To ensure the correct behavior, we do the following:

function animate() {

requestAnimFrame(animate);

draw();

}

window.requestAnimFrame = (function () {

return window.requestAnimationFrame || window.webkitRequestAnimationFrame || window.mozRequestAnimationFrame || window.oRequestAnimationFrame || window.msRequestAnimationFrame ||

function ( /\* function \*/ callback, /\* DOMElement \*/ element) {

window.setTimeout(callback, 1000 / 60);

};

This code will add the **requestAnimationFrame** to the window object (if needed), adapting it to the browser’s built-in features. In the worst case scenario (we end up using **setTimeout**) you have to keep in mind that depending on the browser this timeout might be higher (they are free to pad **setTimeout**() with an arbitrary amount of time) and the time the callback takes place is not factored in the interval (but if you did a **setInteval**() it would be worse because several intervals could be fired at the same time if something blocked one for longer time than expected…).

The only remaining thing is actually animate the background and the elements. As we said previously, we are not doing a 1:1 relation but rather doing some smooth movements that also use inertia.

this.draw = function () {

this.update();

this.stage.update();

};

this.update = function () {

var dX = \_targetX - \_prevViewportX;

\_prevViewportX = \_targetX;

var velocity = Math.abs(dX);

\_curVelX += (velocity - \_curVelX) \* .3;

\_direction = dX < 0 ? -1 : 1;

var ease = 0.12;

var speed = (\_targetX - \_viewPortX) \* ease;

\_viewPortX += speed;

\_imageContainer.x = Math.round(\_viewPortX);

\_bgImg.x = Math.round(\_viewPortX / 10);

};

The most important code here is the draw function and the latest two lines of update.

\_imageContainer.x = Math.round(\_viewPortX);

\_bgImg.x = Math.round(\_viewPortX / 10);

In here we are actually moving the two elements we have in our Canvas (the background and the image container) at different speeds to create the illusion of depth. The background movement is divided by 10 to create this. If we wanted to be further this number will be higher, and smaller if we wanted it to be closer.

The draw function updates the position of the elements in the Canvas (by calling **update**) and then forces the Canvas to be redraw calling **this.stage.update()**.

## Method 2: DOM + JavaScript

You could achieve a similar experience using DOM elements and JavaScript. Probably the best way to achieve this will be using CSS 2D transformations instead of margins to prevent any issue with the position of the items.

The technique used will be similar:

1. You have to know when the user has clicked
2. Calculate the different in the x axis
3. Update your elements accordingly
4. Stop when the user releases the mouse

You will use a background div.

The pros of this approach include:

* You don’t have to rely on third party libraries
* It should easily work across multiple browsers, even the old ones
* The relative newness of the Canvas element means that most developers are likely more familiar with the DOM, which may translate into shorter development time.

The cons include:

* It is not hardware accelerated (if you are using IE9)
* You will probably have to deal with rendering differences across your supported browsers
* If you are building a game you should probably use the Canvas option

# Conclusions and Recommendations

* Canvas integration is more or less standardized across all the browsers. If you are not dealing with edge cases the result should look the same in all of them.
* Not all the devices have the same performance when working with Canvas. The iPad, for example, may provide a less than desirable experience because of hardware limitations.
* It is a good idea to separate logic from frame animation. In this case we update some value (touch/mouse offsets, etc.) at one time (when the user interacts) and we update what is drawn in the Canvas at another time (at a fixed rate of 60 frames per second).
* Easljs is a great library to work with Canvas. It allows us to work with each object directly and update their position. The key is to remember to force the Canvas to update when changing the elements position.
* Creating a full application with Canvas can be very difficult. Also think about SEO and how this type of application could impact you. There are various methods to approach SEO when working with the Canvas elements that will be discussed in a future experiment.
* Development with Canvas can be time consuming, particularly if you are starting from scratch. Start creating your own libraries or use existing libraries such as EaselJS to reduce development time.

# Resources

## Relevant Web sites and specifications

|  |  |
| --- | --- |
| EaselJS | <http://easeljs.com/> |
|  |  |

## Microsoft Resources

|  |  |
| --- | --- |
| H5E Primary Contact | Chewy Chong ([ChewyC](mailto:%20chewyc?subject=HTML5%20Experiments%20Cookbooks)) |
| H5E Development Contact | Anton Molleda Quintana ([v-anmoll](mailto:v-anmoll?subject=HTML5%20Experiments%20Cookbooks)) |

# Appendix A: About H5E

## What is an HTML5 Elements Cookbook?

Each HTML5 Elements Cookbook reflects a case study of an aspirational experience that is provided by a native or component-based application. The HTML5 Experiments that are conducted by the H5E team use HTML5 and related technologies to replicate these experiences. Our primary objective is to learn from these experiments to determine if an HTML5 alternative to component-based or native implementations is both possible, and practical. Each Cookbook provides a description of the element and technical details of the HTML5 replication of that feature. We also include recommendations on whether it makes sense to pursue this approach.

## Contact us

If you need assistance with technical solutions or have a best practice to share, please contact us by sending email to Chewy Chong ([ChewyC](mailto:chewyc?subject=H5E%20Cookbooks%20and%20Documentation)).

## Copyright and trademark information

This document is Microsoft Confidential.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

 2011 Microsoft Corporation. All rights reserved.

# Document Revision History

|  |  |  |
| --- | --- | --- |
| **Reviser** | **Date** | **Revisions** |
| **v-anmoll** | 14 July 2011 | Initial draft |
| **v-jgeige** | 15 July 2011 | Minor edit |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |