Transitions from Image to Image
Animation, internal buffering, and calculation of pixel coordinates

By Jeanine Meyer  jeanine.meyer@purchase.edu  http://faculty.purchase.edu/jeanine.meyer

In this article I will explain how to use HTML5 and JavaScript to produce animated transitions from one image into another. The techniques include the use of an internal canvas element as a buffer, setInterval for animation, calculations for the canvas coordinates, and setting the cursor icon. I also include general remarks on creating programs involving iterations.

You probably are familiar with setting up short animations for transitioning between lecture charts or scenes in a video using special software. I will describe how you can produce similar transitions for changing from one image to another using the HTML5 canvas element, demonstrating the techniques with 3 types of transitions.

Figure 1 shows the opening screen for the transition I named "blinds" for its suggestion of window blinds.

Figure 2 shows the screen at a point during the blinds transition.

Figure 3 shows that transition action.

You can transition back to the original image by clicking the change button again. I have linked the three different transition HTML documents together. Clicking on the button labeled Try center outward transition takes the viewer to the transitionctr.html script. Figure 3 shows that transition action.
General techniques

The three transitions certainly are different, but they all make use of the same techniques. I will explain these techniques before plunging into the coding details that distinguish the transitions. Understanding the general techniques will help you create your own applications.

Creating a canvas element dynamically

JavaScript can be written to create elements dynamically. For the transition programs, I wrote code to create a canvas element in addition to the one created by HTML5 markup in the body of the document. My name for this canvas element is buffer: my code will use it to hold the target image. This canvas element is invisible because it is never made visible by being appended as a child of any visible element of the document. The code for the init function is shown in Listing 1.

```
function init() {
  ctx = document.getElementById("canvas").getContext("2d");
  ctx.drawImage(first,0,0,cols,rows);
  buffer = document.createElement("canvas");
  buffer.width=cols;
  buffer.height=rows;
  bufferctx = buffer.getContext("2d");
  bufferctx.drawImage(second,0,0,cols,rows);
}
```

Listing 1: The init function

The global variables first and second hold the two Image objects and cols and rows hold the number of columns and number of rows, respectively. You can check out all the code of the three HTML documents by taking the link indicated below in the Learn More section. So after the init function, invoked in the usual way by an attribute in the <body> tag, there are two canvas elements, one visible and one not visible, holding the two images.

Using drawImage with a canvas as source

The typical use of the drawImage method is to draw Image objects onto the canvas. In a previous article in <jsmag>: Making a collage with video, images and drawings on canvas, April 2012, I demonstrated the use of drawImage to render a frame from a video element. However, there is still another type of source for drawing: a canvas element. This is the basis for the transition programs. The parameters of the drawImage method provide a way to specify the horizontal and vertical location, width and height of the source and the horizontal and vertical location, width and height for the target. This allows parts, and only parts, of the canvas to be re-drawn at set intervals of time, which is what is required for the transition programs. Actually, HTML5 provides more flexibility than what is needed because the coordinates of the source do not have to be the same as the target.
Creating the animation

The transition animations are different but there are ways to think about what you as the programmer need to do for these types of programs. You need to address three questions:

- What do you need your code to do to start the animation?
- What do you need your code to do each iteration, specifically, each call of a function change referenced in a call of setInterval?
- What do you need your code to do to finish and/or determine that the transition is complete?

Starting was essentially the same for each of the animations. I wrote code to draw the appropriate image onto the buffer canvas. This will either be the image held in the variable first or the image in the variable second. I use a Boolean variable I named onfirst to make this determination. To be more precise, at the end of each transition, my code sets up the buffer for the next transition. A call of setInterval(change, dur) is what starts the animation.

The task for each iteration of the animation, each invocation of the change function, is the critical part defining the transitions. For the blinds transition, each iteration involved multiple invocations of the drawImage method, corresponding to rows of the image. Similarly, the random transition featured some large number (I used 500) of individual pixels changing each iteration. The center outward transition was distinctly different from the other two transitions. One and only one invocation of drawImage was performed by the change function.

Lastly, how does the code determine that the transition is complete? For this task, the blinds and the center out transition are similar. You will read in the next sections how a combination of for loops and if statements determine that the transition is complete. The random transition is different. I decided that it wasn’t a good idea to wait for the transition to be complete based on switching of random pixels because it could take too long and the checking itself would be computationally-intensive. Instead, I decided that I would set a fixed limit of 100 iterations and count down. When this number of iterations had been performed, my code draws the whole new image on the canvas. This has the effect of switching over any pixel that had not been changed to the target image. In all three transitions, the animation is halted using the clearInterval function.

Sometimes novice programmers, and perhaps even those of us with more experience, hesitate when contemplating a task because it seems to be so many operations. We need to remember that “the program/computer” is doing the work and, with looping and functions such as setInterval, it isn’t even that much code. Our job is to determine how to start, what is done each iteration, and how to end or determine if the task is complete. Another factor to keep in mind is that the starting and ending may be more difficult conceptually than the iteration step and we can tackle the three tasks in any order.

Using buttons

My programs make use of the HTML5 button element for both starting the transitions and for going to the next transition website. Listing 2 shows the complete body element for the transitionrandom.html document. The other two programs are similar.

```html
<body onload="init();">
<button onClick="startchange();">change </button><br />
<canvas id="canvas" width="500" height="500">
No canvas support
</canvas>
<br />
<button onClick="parent.location='transition.html'">Try blinds transition </button>
</body>
```

Listing 2: Body element of transitionrandom.html

Making the button perform as a hyperlink was accomplished by setting the onClick attribute to be the JavaScript assignment statement setting parent.location to be the desired relative address (refer to Listing 2). Notice the use of single quotes within the double quotes needed to surround the whole statement.

The startchange function is the same for all three programs, as shown in Listing 3.

```javascript
function startchange() {
  tid = setInterval(change,dur);
}
```

Listing 3: The startchange function

I also included a style directive to change the cursor to a pointer as well as provide a margin around the button. The style directive is shown in Listing 4.

```html
<style>
battery {cursor: pointer; margin:10px;}
</style>
```

Listing 4: Style directive for buttons

With this general background, now I will explain the coding for each of the three transitions.

Blinds

The critical aspect of the blinds transition is that each iteration itself involves a set of steps, that is, iterations.

The change function (shown in Listing 5) for the blinds transition changes the 0, 10, 20, and so on rows in the first iteration; then the 1, 11, 21, 31, etc., in the second iteration, and so on. Notice the use of the variable row in Listing 5. The for-loop does the drawing and the logic afterwards determines if the transition is over and if so, sets up for the next transition.
function change() {
  var i;
  for (i=row;i<rows;i+=10){
    ctx.drawImage(buffer,0,i,cols,1,0,i,cols,1);
  }
  row++;
  if (row>=10){
    clearInterval(tid);
    row = 0;
    if (onfirst) {
      onfirst = false;
      bufferctx.drawImage(first,0,0,cols,rows);
    } else {
      onfirst = true;
      bufferctx.drawImage(second,0,0,cols,rows);
    }
  }
}

Listing 5: The change function for the blinds transition

Center out

The change function (shown in Listing 6) for the center out transition does not contain a for-loop at the start because there is only one drawing operation per iteration. The variables midx and midy hold the values indicating the midpoints of the images and the variable x and y hold the changing values for the upper left corners of the changing portions. The aspect ratio of the images is 480 rows to 320 columns so this is why x and y are incremented by 2 and 3, respectively. (No shame in getting this reversed the first time you try it.)

function change() {
  x=x-2;
  y=y-3;
  if ((x>0) &&(y>0)) {
    w = 2*(midx-x);
    h = 2*(midy-y);
    ctx.drawImage(buffer,x,y,w,h,x,y,w,h);
  } else {
    ctx.drawImage(buffer,0,0,cols,rows);
    clearInterval(tid);
    x = midx;
    y = midy;
    if (onfirst) {
      onfirst = false;
      bufferctx.drawImage(first,0,0,cols,rows);
    } else {
      onfirst = true;
      bufferctx.drawImage(second,0,0,cols,rows);
    }
  }
}

Listing 6: The change function for the center out transition

Random spots

The random transition resembles the blinds transition in that each iteration performs a set of operations, but it is unlike blinds or center out in how to make the determination on completeness.

function change() {
  for(i=0;i<per;i++) {
    x = Math.floor(Math.random()*(cols-2));
    y = Math.floor(Math.random()*(rows-3));
    ctx.drawImage(buffer,x,y,2,3,x,y,2,3);
  }
  iterations--;
  if (iterations<0) {
    ctx.drawImage(buffer,0,0,cols,rows);
    clearInterval(tid);
    iterations = limit;
    if (onfirst) {
      onfirst = false;
      bufferctx.drawImage(first,0,0,cols,rows);
    } else {
      onfirst = true;
      bufferctx.drawImage(second,0,0,cols,rows);
    }
  }
}

Listing 7: The change function for the random transition

The pseudo-random operations can and should be random, so it could happen that the transitioning does not appear continuously progressive, that is, with more and more of the image switching over to the other image.

The teacher in me says that you should immediately design and program your own transition. Do learn from the specific examples and general remarks to increase your confidence about animation and looping. Enjoy.

Learn more

There are many sources, online and in-print and some sort of e-books, for learning HTML5 and JavaScript techniques. Here are links to two of my books, another book from friendsOfEd/Apress publishers and first of the three transition examples featured in this article.

- The Essential Guide to HTML5: Using Games to learn HTML5 and JavaScript, http://www.friendsofed.com/book.html?isbn=9781430233831. This is a text for beginners at programming as well as more experienced programmers who want to learn about HTML and JavaScript, including the new features of HTML5. The Bouncing Ball; Cannonball and Slingshot; and Rock, Paper, Scissors chapters illustrate distinct examples of animation.

- HTML5 and JavaScript Projects, http://www.apress.com/9781430240327. This book is more advanced than the first one. It includes chapters on the Google Maps API and on php and MySql. There are several examples involving video, including a chapter on origami that contains material using algebra and geometry.
• **HTML5 Animation with JavaScript**, http://www.apress.com/9781430236658. I was given a chance to review this book for ACM Computing Reviews and strongly recommend it for a variety of animation techniques.

• To see the transition scripts in action and to view the source code, go to http://faculty.purchase.edu/jeanine.meyer/html5/transition.html for the blinds transition, use the change button, and click on the button below the image to get to the center out transition; repeat for the random spots transition.

Jeanine Meyer lives just north of New York City and currently teaches at Purchase College/SUNY after many years at IBM, doing research on robotics and manufacturing and consulting on educational grants. She likes providing programming examples for her Mathematics/Computer Science and New Media students and really, really likes working with images and video clips of her granddaughter and other family members.