Estudo de Caso: efeito Page Flip do 20thingsilearned.com

By Hakim El Hattab

Publicado: Janeiro 20th, 2011
Comments: 151

Introdução

Em 2010, o F-i.com (link em inglês) e a equipe do Google Chrome colaboraram para a criação de um aplicativo da web para educação baseado em HTML5 chamado 20 lições que aprendi sobre navegadores e a web (www.20thingsilearned.com/pt-BR). Uma das principais ideias do projeto é a de que ele seria mais bem apresentado no contexto de um livro (link em inglês). Como o conteúdo do livro é basicamente sobre tecnologias de web aberta, acreditamos que seria importante nos mantermos fiéis a esse princípio fazendo com que seu formato seja também um exemplo do que essas tecnologias nos permitem realizar hoje em dia.

Capa e página inicial de "20 lições que aprendi sobre navegadores e a web" (http://www.20thingsilearned.com/pt-BR)

Decidimos que a melhor forma de alcançar a experiência de um livro real seria simular as partes positivas da experiência de leitura analógica sem descartar os benefícios do

ambiente digital em áreas como a navegação. O tratamento gráfico e interativo do fluxo de leitura foi tratado com cuidado especial, especialmente no que diz respeito à virada (flip) de uma página para a outra.

Primeiros passos

Este tutorial serve de guia para o processo de criação de seu próprio efeito de virada de página utilizando o elemento canvas e muito JavaScript. Parte do código rudimentar, como declarações de variáveis e assinatura de escuta de eventos, foi deixada de fora dos snippets deste artigo. Por isso, lembre-se de consultar o exemplo funcional.

Antes de iniciarmos, recomendamos que você veja a demonstração para conhecer o que planejamos produzir.

Marcação

É sempre importante lembrar: o que é desenhado em canvas não pode ser indexado por mecanismos de pesquisa, selecionado por um visitante ou encontrado por pesquisas no navegador. Por isso, o conteúdo com o qual trabalharemos é colocado diretamente no elemento DOM e, em seguida, manipulado por JavaScript, caso disponível. A marcação necessária para isso é mínima:

```
<div id="book">
  <canvas id="pageflip-canvas"></canvas>
  <div id="pages">
    <section>
      <div> <!-- Any type of contents here --> </div>
    </section>
    <!-- More <section>'s here -->
  </div>
</div>
```

Temos um elemento do recipiente principal para o livro, que por sua vez contém as diferentes páginas de nosso livro e o elemento `canvas` no qual desenharemos as páginas que serão viradas. No elemento `section`, existe um `div` wrapper para o conteúdo. Ele é necessário para que possamos modificar a largura da página sem afetar o layout do conteúdo. A `div` tem largura fixa e a `section` é configurada para ocultar sua ultrapassagem, fazendo com que a largura da `section` aja como uma máscara horizontal para a `div`.

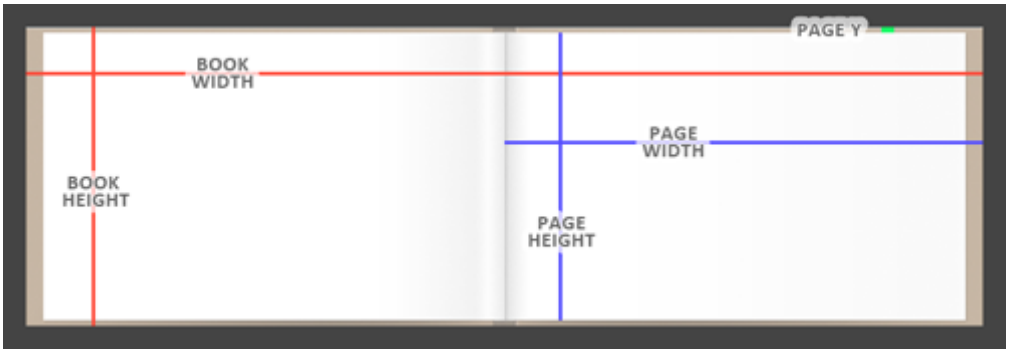
Uma imagem de plano de fundo com textura de papel e a sobrecapa marrom do livro é adicionada ao elemento livro.

Lógica

O código necessário para fazer com que a virada de página funcione não é muito complexo, mas é extenso, uma vez que envolve muitos gráficos gerados processualmente. Em primeiro lugar, apresentaremos a descrição dos valores constantes utilizados em todo o código.

```
var BOOK_WIDTH = 830;
var BOOK_HEIGHT = 260;
var PAGE_WIDTH = 400;
var PAGE_HEIGHT = 250;
var PAGE_Y = ( BOOK_HEIGHT - PAGE_HEIGHT ) / 2;
var CANVAS_PADDING = 60;
```

O `CANVAS_PADDING` é adicionado ao redor do canvas para que o papel possa ultrapassar os limites do livro quando uma página é virada. Algumas constantes definidas aqui também são definidas em CSS. Por isso, para alterar o tamanho do livro, também é necessário atualizar os valores naqueles scripts.

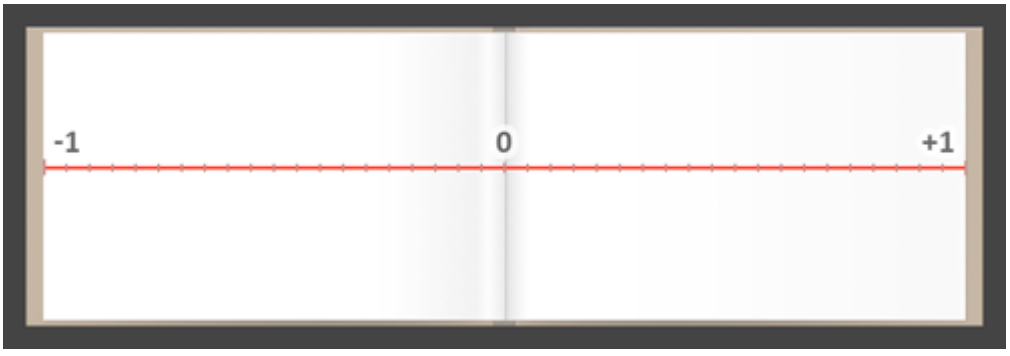


Os valores das constantes utilizados ao longo do código para acompanhar a interação e desenhar a virada de página.

Em seguida, precisamos definir um objeto de virada (flip) para cada página. Esses objetos serão atualizados constantemente à medida que interargimos com o livro para refletir o status atual da virada.

```
// Create a reference to the book container element
var book = document.getElementById( "book" );

// Grab a list of all section elements (pages) within the book
var pages = book.getElementsByTagName( "section" );

for( var i = 0, len = pages.length; i < len; i++ ) {
    pages[i].style.zIndex = len - i;

    flips.push( {
    progress: 1,
    target: 1,
    page: pages[i],
    dragging: false
  });
}
```

Primeiro, devemos organizar as páginas corretamente em camadas por meio da organização dos z-indexes dos elementos da seção de forma que a primeira página esteja disposta na parte superior e a última na parte inferior. As propriedades mais importantes dos objetos de virada (flip) são os valores `progress` e `target`. Estes são utilizados para determinar a extensão da dobra atual da página; -1 significa inteiramente para a esquerda, 0 é exatamente no centro e +1 indica a extremidade direita do livro.



Os valores "progress" e "target" da virada são utilizados para determinar onde a dobra da página deve ser desenhada em uma escala de −1 a +1.

Agora que temos um objeto de virada definido para cada página, é necessário capturar e utilizar as entradas dos usuários para atualizar o estado da virada.

```
function mouseMoveHandler( event ) {
  // Offset mouse position so that the top of the book spine is
0,0
  mouse.x = event.clientX - book.offsetLeft - ( BOOK_WIDTH / 2 );
  mouse.y = event.clientY - book.offsetTop;
}
```

```
function mouseDownHandler( event ) {
  // Make sure the mouse pointer is inside of the book
  if (Math.abs(mouse.x) < PAGE_WIDTH) {
    if (mouse.x < 0 && page - 1 >= 0) {
      // We are on the left side, drag the previous page
      flips[page - 1].dragging = true;
    }
    else if (mouse.x > 0 && page + 1 < flips.length) {
      // We are on the right side, drag the current page
      flips[page].dragging = true;
    }
  }

  // Prevents the text selection
  event.preventDefault();
}

function mouseUpHandler( event ) {
  for( var i = 0; i < flips.length; i++ ) {
    // If this flip was being dragged, animate to its destination
    if( flips[i].dragging ) {
      // Figure out which page we should navigate to
      if( mouse.x < 0 ) {
        flips[i].target = -1;
        page = Math.min( page + 1, flips.length );
      }
      else {
        flips[i].target = 1;
        page = Math.max( page - 1, 0 );
      }
    }

    flips[i].dragging = false;
  }
}
```

A função `mouseMoveHandler` atualiza o objeto `mouse` para que possamos trabalhar sempre a partir da localização mais recente do cursor.

Em `mouseDownHandler`, primeiro verificamos se o mouse foi pressionado na página direita ou na esquerda, a fim de saber para qual direção a virada deve ser iniciada. Também verificamos se existe outra página naquela direção, já que podemos estar na primeira ou na última página. Se uma opção válida de virada estiver disponível depois dessas verificações, configuramos a sinalização `dragging` do objeto de virada correspondente como `true`.

Ao alcançarmos o `mouseUpHandler`, passamos por todos os `flips` e verificamos se algum deles foi sinalizado como `dragging` e deve ser liberado. Quando uma virada é liberada, configuramos seu valor target para corresponder ao lado para o qual deve

virar, conforme a posição atual do mouse. O número da página também é atualizado para refletir essa navegação.

Renderização

Agora que a maior parte de nossa lógica está estruturada, apresentaremos detalhes sobre como renderizar o papel no elemento canvas. A maior parte do processo ocorre na função `render()`, que é solicitada 60 vezes por segundo para atualizar e desenhar o estado atual de todas as viradas ativas.

```
function render() {
  // Reset all pixels in the canvas
  context.clearRect( 0, 0, canvas.width, canvas.height );

  for( var i = 0, len = flips.length; i < len; i++ ) {
    var flip = flips[i];

    if( flip.dragging ) {
      flip.target = Math.max( Math.min( mouse.x / PAGE_WIDTH, 1 ),
-1 );
    }

    // Ease progress towards the target value
    flip.progress += ( flip.target - flip.progress ) * 0.2;

    // If the flip is being dragged or is somewhere in the middle
    // of the book, render it
    if( flip.dragging || Math.abs( flip.progress ) < 0.997 ) {
      drawFlip( flip );
    }

  }
}
```

Antes de começarmos a renderizar os `flips`, redefinimos o canvas com o método `clearRect(x,y,w,h)`. A limpeza total do canvas tem um alto custo de desempenho e seria muito mais eficiente limpar somente as regiões em que desenhamos. Para não fugir ao tópico deste tutorial, consideraremos a limpeza de todo o canvas.

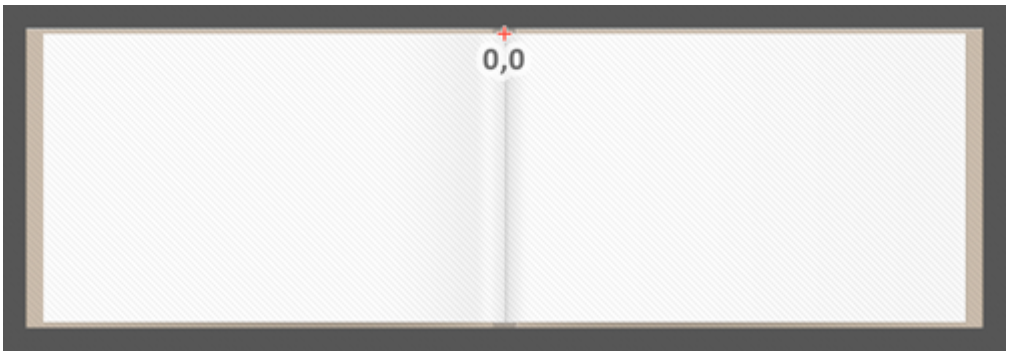
Se um flip estiver sendo arrastado, atualizaremos seu valor `target` para corresponder à posição do mouse em uma escala de -1 a +1 em vez de pixels reais. Também incrementamos o valor `progress` em uma fração da distância do `target`. Isso resulta em uma progressão suave e animada da virada, já que esta é atualizada a cada frame.

Já que estamos passando por todos os `flips` em cada frame, devemos nos certificar de que apenas os ativos sejam redesenhados. Se um flip não estiver muito próximo à extremidade do livro (cerca de 0,3% da `BOOK_WIDTH`), ou se for sinalizado como `dragging`, será considerado ativo.

Agora que toda a lógica está estruturada, devemos desenhar a representação gráfica de um flip, dependendo de seu estado atual. É hora de analisarmos a primeira parte da função `drawFlip(flip)`.

```
// Determines the strength of the fold/bend on a 0-1 range
var strength = 1 - Math.abs( flip.progress );

// Width of the folded paper
var foldWidth = ( PAGE_WIDTH * 0.5 ) * ( 1 - flip.progress );

// X position of the folded paper
var foldX = PAGE_WIDTH * flip.progress + foldWidth;

// How far outside of the book the paper is bent due to
perspective
var verticalOutdent = 20 * strength;

// The maximum widths of the three shadows used
var paperShadowWidth = (PAGE_WIDTH*0.5) * Math.max(Math.min(1 -
flip.progress, 0.5), 0);
var rightShadowWidth = (PAGE_WIDTH*0.5) *
Math.max(Math.min(strength, 0.5), 0);
var leftShadowWidth = (PAGE_WIDTH*0.5) *
Math.max(Math.min(strength, 0.5), 0);

// Mask the page by setting its width to match the foldX
flip.page.style.width = Math.max(foldX, 0) + "px";
```

Esta seção do código começa pelo cálculo de diversas variáveis visuais necessárias para desenharmos a dobra de forma realista. O valor `progress` do flip desenhado tem função importante aqui, já que é onde queremos que a dobra da página apareça. Para dar profundidade ao efeito de virada, devemos fazer com que a extensão do papel ultrapasse as extremidades superior e inferior do livro e que este efeito esteja em seu ponto mais alto quando uma virada estiver próxima à lombada do livro.



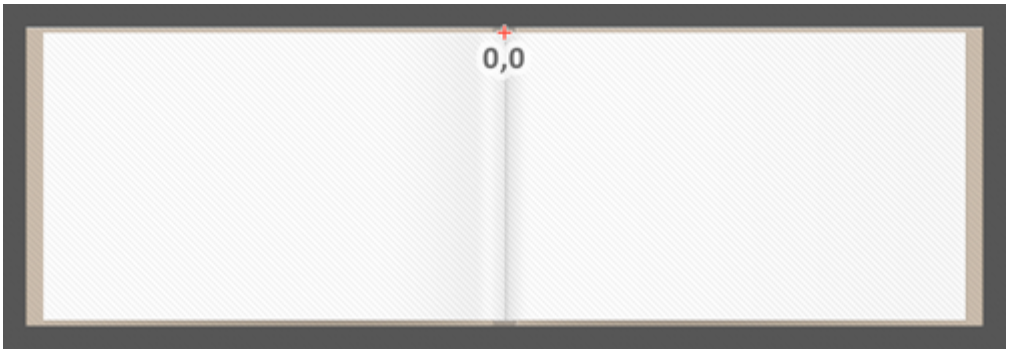
Esta é a aparência de uma dobra de página ao ser virada ou arrastada.

Agora que todos os valores estão preparados, resta desenhar o papel.

```
context.save();
context.translate( CANVAS_PADDING + ( BOOK_WIDTH / 2 ), PAGE_Y +
CANVAS_PADDING );

// Draw a sharp shadow on the left side of the page
context.strokeStyle = 'rgba(0,0,0,'+(0.05 * strength)+')';
context.lineWidth = 30 * strength;
context.beginPath();
context.moveTo(foldX - foldWidth, -verticalOutdent * 0.5);
context.lineTo(foldX - foldWidth, PAGE_HEIGHT + (verticalOutdent *
0.5));
context.stroke();

// Right side drop shadow
var rightShadowGradient = context.createLinearGradient(foldX, 0,
              foldX + rightShadowWidth, 0);
rightShadowGradient.addColorStop(0, 'rgba(0,0,0,'+
(strength*0.2)+')');
rightShadowGradient.addColorStop(0.8, 'rgba(0,0,0,0.0)');

context.fillStyle = rightShadowGradient;
context.beginPath();
context.moveTo(foldX, 0);
context.lineTo(foldX + rightShadowWidth, 0);
context.lineTo(foldX + rightShadowWidth, PAGE_HEIGHT);
context.lineTo(foldX, PAGE_HEIGHT);
context.fill();

// Left side drop shadow
var leftShadowGradient = context.createLinearGradient(
    foldX - foldWidth - leftShadowWidth, 0, foldX - foldWidth, 0);
leftShadowGradient.addColorStop(0, 'rgba(0,0,0,0.0)');
leftShadowGradient.addColorStop(1, 'rgba(0,0,0,'+
(strength*0.15)+')');

context.fillStyle = leftShadowGradient;
context.beginPath();
context.moveTo(foldX - foldWidth - leftShadowWidth, 0);
context.lineTo(foldX - foldWidth, 0);
context.lineTo(foldX - foldWidth, PAGE_HEIGHT);
context.lineTo(foldX - foldWidth - leftShadowWidth, PAGE_HEIGHT);
context.fill();

// Gradient applied to the folded paper (highlights & shadows)
var foldGradient = context.createLinearGradient(
    foldX - paperShadowWidth, 0, foldX, 0);
foldGradient.addColorStop(0.35, '#fafafa');
foldGradient.addColorStop(0.73, '#eeeeee');
foldGradient.addColorStop(0.9, '#fafafa');
foldGradient.addColorStop(1.0, '#e2e2e2');

context.fillStyle = foldGradient;
context.strokeStyle = 'rgba(0,0,0,0.06)';
context.lineWidth = 0.5;
```

```
// Draw the folded piece of paper
context.beginPath();
context.moveTo(foldX, 0);
context.lineTo(foldX, PAGE_HEIGHT);
context.quadraticCurveTo(foldX, PAGE_HEIGHT + (verticalOutdent *
2),
                         foldX - foldWidth, PAGE_HEIGHT +
verticalOutdent);
context.lineTo(foldX - foldWidth, -verticalOutdent);
context.quadraticCurveTo(foldX, -verticalOutdent * 2, foldX, 0);

context.fill();
context.stroke();

context.restore();
```

O método `translate(x,y)` da API canvas é utilizado para deslocar o sistema de coordenadas para que possamos desenhar a virada de página fazendo com que a parte superior da lombada represente a posição 0,0. Também devemos utilizar as funções `save()`, para salvar a matriz de transformação atual do canvas, e `restore()` para restaurar quando terminarmos o desenho.



É a partir deste ponto que desenhamos a virada de página. O ponto 0,0 original está na parte superior esquerda da imagem, mas ao modificar este parâmetro por meio de translate(x,y), simplificamos a lógica de desenho.

Preencheremos a forma do papel dobrado com `foldGradient` para dar a ele iluminação e sombras realistas. Também adicionaremos uma linha fina em torno do desenho do papel para que este não desapareça quando posicionado sobre planos de fundo claros.

Resta agora desenhar a forma do papel dobrado utilizando as propriedades definidas acima. Os lados esquerdo e direito do papel são desenhados como linhas retas e os lados superior e inferior são curvados para causar a impressão de curvatura do papel encadernado. A intensidade da dobra do papel é determinada pelo valor `verticalOutdent`.

Pronto! Agora você tem uma navegação de virada de página totalmente funcional.

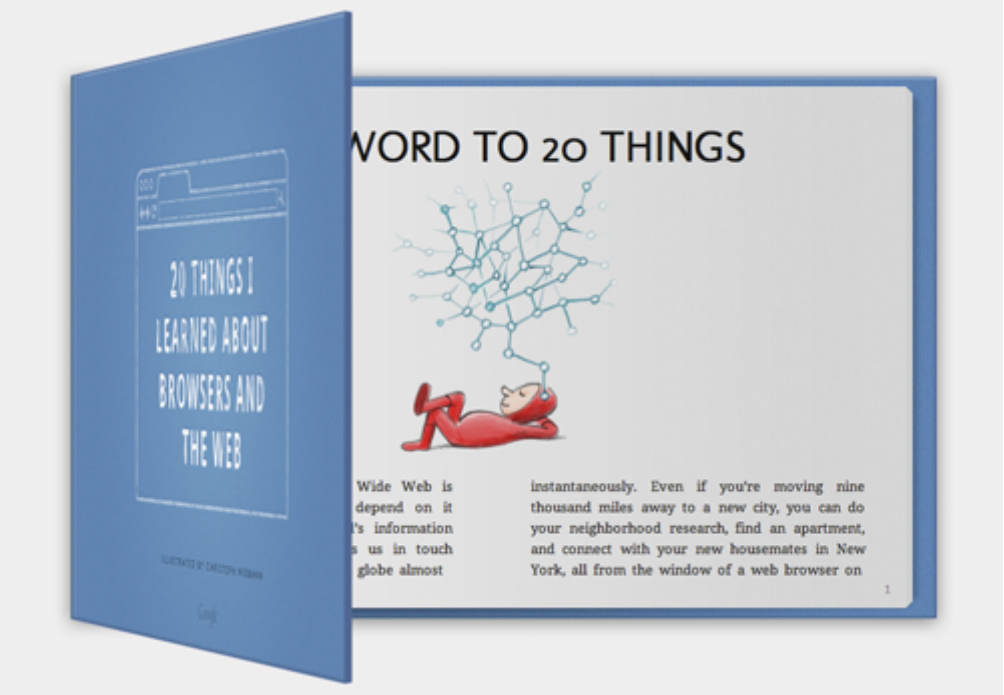
Demonstração de virada de página

O efeito de virada de página tem como objetivo causar a sensação interativa correta, por isso somente visualizar imagens não é o suficiente. Use os links abaixo para testar o resultado final.

Visualizar um exemplo em funcionamento

Fazer download do código-fonte (75k .zip)

Próximas etapas



A virada de página suave neste tutorial é ainda mais poderosa quando aliada a outros recursos relacionados a livros, como uma capa dura interativa.

Este é apenas um exemplo do que pode ser feito utilizando recursos HTML5 como o elemento canvas. Recomendo que você confira a experiência de livro mais refinada da qual esta técnica é apenas uma amostra: www.20thingsilearned.com/pt-BR. Lá, você verá como as viradas de página podem ser utilizadas em um aplicativo real e como podem ser poderosas se aliadas a outros recursos HTML5.

Referências

- Especificação da API Canvas