11 Canvas

Neste capítulo você vai ver uma das maneiras de utilizar o JavaScript para fazer desenhos no seu documento. Você vai ver que com uns poucos métodos para desenhar linhas, arcos e curvas é possível fazer praticamente tudo que você imagina. Vai entender como utilizar transformações de coordenadas para modelar com facilidade o sistema físico de interesse. Finalmente, vai ver como produzir animações, integrando temporizadores aos seus desenhos.

O canvas é o elemento onde podemos fazer desenhos. É colocado em um documento HTML com os marcadores `<canvas> ... </canvas>` que, além dos atributos gerais dos elementos HTML (`id`, `style`), tem dois atributos específicos: `width` e `height`, que definem, respectivamente, a largura e a altura do canvas, em pixels.

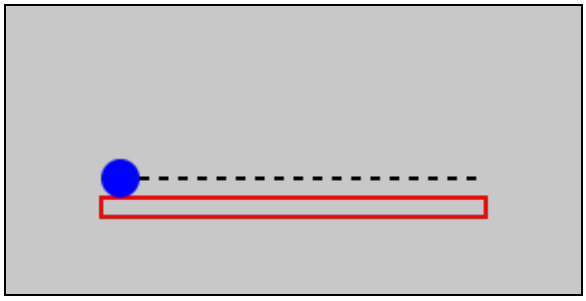
Uma vez no documento, a sua referência pode ser obtida com a instrução `document.getElementById()`. A partir dessa referência têm-se acesso a suas propriedades, em particular à sua largura (`.width`), altura (`.height`) e o seu contexto gráfico (`.getContext('2d')`). Vários métodos do contexto gráfico permitem o desenho de linhas, retângulos, círculos etc., com diferentes propriedades, tais como cor, espessura de linha, preenchimento, opacidade etc. O exemplo a seguir ilustra o esquema básico de utilização de um canvas para desenhar os principais elementos básicos.

Listagem:

```
<div style="text-align:center">
<canvas id="cnv" width="300" height="150"
        style="border: 1px solid black;
               background-color:rgb(200,200,200)">
</canvas>
</div>

<script>
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
```

```
// retângulo vermelho delineado
ctx.beginPath();
ctx.rect(50,100,200,10);
ctx.strokeStyle = "red";
ctx.lineWidth = 2;
ctx.stroke();

// linha preta horizontal
ctx.beginPath();
for (var i=0;i<19;i++) {
   ctx.moveTo(60 + 10*i,90);
   ctx.lineTo((60 + 10*i + 5),90);
}
ctx.lineWidth = 2;
ctx.strokeStyle = "black";
ctx.stroke();

// bola azul preenchida
ctx.beginPath();
ctx.arc(60,90,10,0,2*Math.PI,true);
ctx.fillStyle = "blue";
ctx.fill();

</script>
```

Resultado:



Em primeiro lugar, observe como é definido o canvas no documento HTML, com um `id` e os atributos de largura (`width`) e altura (`height`), além de especificações de estilo (no caso, relacionadas à borda e a um fundo cinza claro). O `canvas` está definido dentro de uma `div` com uma declaração de estilo que centraliza o seu conteúdo.

O script inicia buscando o elemento `canvas` desejado e a seguir atribui à variável `ctx` o seu contexto gráfico. A seguir, o script desenha três estruturas: o retângulo vermelho delineado, a linha preta horizontal tracejada e o círculo azul preenchido.

O retângulo vermelho delineado é desenhado com o método `rect(x,y,w,h)`, que recebe como parâmetros as coordenadas `x,y` do canto superior esquerdo do retângulo e a sua largura `w` e altura `h`. Para que seja desenhado sem preenchimento, é preciso executar o método `stroke()` ("pincelada").

A linha tracejada é desenhada como uma sucessão de pequenas linhas separadas por espaços em branco. O método `moveTo(x,y)` "levanta" o "pincel" e leva-o até o ponto `x,y` sem deixar marcas; o método `lineTo(x,y)` traça uma linha reta do ponto em que o pincel estiver até as coordenadas `x,y` que recebeu. No caso, as coordenadas `x` do início de cada

traço são 60, 70, 80 etc. (`60 + 10*i`), as coordenadas do fim dos traços são 65, 75, 85 etc. (`60 + 10*i + 5`) e a coordenada `y`, fixa, é 90. A cor da pincelada é controlada pela propriedade `strokeStyle` (estilo da pincelada) e sua espessura pela propriedade `lineWidth` (largura da linha). No exemplo, o laço `for` desenha uma linha com comprimento de 5 pixels a cada 10 pixels.

O círculo azul é desenhado com o método `arc(x,y,r,ain,afn,ahor)`, onde `x,y` são as coordenadas do centro do arco, `r` o seu raio, `ain` e `afn` os ângulos inicial e final em radianos. O último parâmetro, `ahor`, é uma expressão booleana (`true` ou `false`, 0 ou 1) que, se verdadeira, indica que o arco deve ser desenhando no sentido anti-horário e, se falsa, no sentido horário. A cor de preenchimento é atribuída à propriedade `fillStyle` e o desenho, com preenchimento, é feito com uma chamada ao método `fill()`.

Sua vez... (11-1)

1. Modifique o exemplo anterior de modo que (a) faça o retângulo ser desenhado preenchido e o círculo somente com contorno e (b) desenhe um arco de um quarto de círculo (0 a π/2), especificando `true` ou `false` para o último parâmetro.

2. Cada desenho começa com uma chamada ao método `beginPath()` do contexto gráfico, que indica o início de um conjunto de instruções relacionadas a uma determinada trajetória de desenho no documento. Comente algumas ou todas as linhas com `beginPath()` e veja o que acontece. Como você interpreta os resultados? O que conseguiu descobrir na internet a respeito disso?

Você encontrará facilmente na Internet tutoriais bastante abrangentes sobre o uso do canvas que podem lhe ensinar a fazer sofisticadas obras de arte animadas. Aqui vamos nos ater ao básico para visualizar informações e modelos elementares de sistemas físicos.

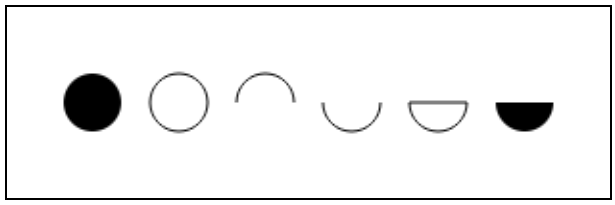
O exemplo a seguir foi pensado para deixar mais claras as diferenças entre o `stroke()` e o `fill()`, entre o uso de `true` ou `false` para a direção do arco e entre o uso ou não do método `closePath()` para fechar ou não o desenho (o método `closePath()` também pode ser utilizado para fechar um polígono construído a partir de linhas).

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="315" height="100"
        style="border-width: 1px; border-style:solid">
</canvas>
</p>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

// cores de linha e de preenchimento
ctx.strokeStyle = "black";
```

```
ctx.fillStyle = "black";
// círculo preenchido
ctx.beginPath();
ctx.arc(45,50,15,0,2*Math.PI,true);
ctx.fill();
// círculo delineado
ctx.beginPath();
ctx.arc(90,50,15,0,2*Math.PI,true);
ctx.stroke();
// semi-círculo para cima
ctx.beginPath();
ctx.arc(135,50,15,0,Math.PI,true);
ctx.stroke();
// semi-círculo para baixo
ctx.beginPath();
ctx.arc(180,50,15,0,Math.PI,false);
ctx.stroke();
// semi-círculo fechado
ctx.beginPath();
ctx.arc(225,50,15,0,Math.PI,false);
ctx.closePath();
ctx.stroke();
// semi-círculo preenchido
ctx.beginPath();
ctx.arc(270,50,15,0,Math.PI,false);
ctx.fill();

</script>
```

Resultado:



Sua vez.. (11-2)

Modifique o exemplo anterior de modo a atribuir uma cor diferente a cada figura desenhada. Depois de ver o resultado comente alguns ou todos os `beginPath()` e analise os resultados.

É possível fazer desenhos sobre uma imagem previamente carregada (uma foto, por exemplo). O exemplo a seguir ilustra como fazer isto, carregando uma imagem com o fundo quadriculado e desenhando algumas curvas de Bezier sobre ela.

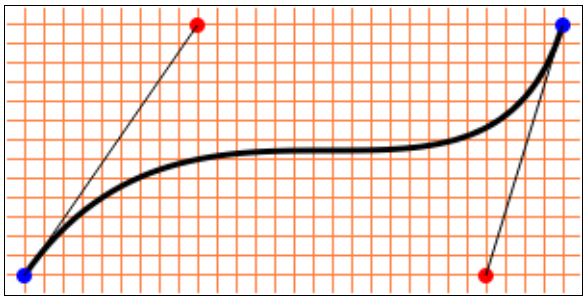
Listagem:

```
<div style="text-align:center">
<canvas id="cnv" width="300" height="150"
```

```
        style="border:1px solid black">
</canvas>
</div>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

var img = new Image();
img.src = "js-canvas-quadriculado.png";
img.onload = function() { desenhaTudo(); }

function desenhaTudo() {

    // imagem de fundo
    ctx.drawImage(img,0,0);

    var x1  = 10, y1  = 140, x2 = 290, y2 = 10;
    var xct1 = 100, yct1 = 10, xct2 = 250, yct2 = 140;

    // curva de Bézier
    ctx.beginPath();
    ctx.moveTo(x1,y1);
    ctx.bezierCurveTo(xct1, yct1, xct2, yct2, x2, y2)
    ctx.lineWidth = 3;
    ctx.strokeStyle = "black";
    ctx.stroke();

    // pto inicial
    ctx.beginPath();
    ctx.arc(x1,y1,4,0,2*Math.PI,true);
    ctx.fillStyle = "blue";
    ctx.fill();

    // 1o. pto de controle
    ctx.beginPath();
    ctx.arc(xct1,yct1,4,0,2*Math.PI,true);
    ctx.fillStyle = "red";
    ctx.fill();

    // linha ligando pto inicial a 1o. pto de controle
    ctx.beginPath();
    ctx.moveTo(x1,y1);
    ctx.lineTo(xct1,yct1);
    ctx.lineWidth = 1;
    ctx.strokeStyle = "black";
    ctx.stroke();

    // pto final
    ctx.beginPath();
    ctx.arc(x2,y2,4,0,2*Math.PI,true);
    ctx.fillStyle = "blue";
    ctx.fill();

    // 2o. pto de controle
    ctx.beginPath();
    ctx.arc(xct2,yct2,4,0,2*Math.PI,true);
    ctx.fillStyle = "red";
    ctx.fill();

    // linha ligando pto final a 2o. pto de controle
    ctx.beginPath();
    ctx.moveTo(x2,y2);
    ctx.lineTo(xct2,yct2);
    ctx.lineWidth = 1;
    ctx.strokeStyle = "black";
```

```
    ctx.stroke();

}
</script>
```

Resultado:



Depois da definição do canvas fora do script e da captura do contexto gráfico no início do script, é criado um objeto `Image()`, que tem várias propriedades e métodos. Em seguida, a propriedade `src` recebe o nome do arquivo com a imagem a ser carregada, que, como especificado, tem que estar na mesma pasta que o arquivo HTML.

A instrução seguinte, `img.onload = function() { desenhaTudo() }`, especifica que a função `desenhaTudo()` somente deve ser executada *depois* que o arquivo de imagem for carregado. Isso é necessário porque o tempo de acesso à imagem, armazenada em geral no disco rígido ou em algum outro local da internet, é muito maior do que o tempo de execução das instruções, o que pode levar a alguns resultados bizarros.

Sua vez.. (11-3)

Modifique o exemplo anterior de modo a substituir a instrução `img.onload = desenhaTudo()` por `desenhaTudo()`, isto é, uma chamada direta à função. Veja o que acontece quando você recarrega a página "do zero" (clicando sobre o arquivo na pasta ou utilizando `CTRL+F5`) ou simplesmente atualizando a página (`F5`).

Os resultados podem ser diferentes em diferentes navegadores (Chrome, Firefox, IE, Safari etc.) e diferentes sistemas operacionais (Androide, Linux, Windows etc.).

A instrução `ctx.drawImage(img,0,0)` coloca a imagem carregada no canvas, fazendo com que seu canto superior esquerdo seja colocado no canto superior esquerdo do canvas, que tem coordenadas (0,0). O método `drawImage()` pode receber dois parâmetros adicionais, que aplicam um fator de escala às direções horizontal e vertical da imagem.

Sua vez.. (11-4)

Modifique o exemplo anterior de modo a substituir a instrução `ctx.drawImage(img,0,0)` por (a) `ctx.drawImage(img,0,0,150,75)` e (b) `ctx.drawImage(img,75,37,150,75)` e observe os resultados.

Além de mostrar como carregar uma imagem e desenhar sobre ela, o exemplo mostra um uso do método `bezierCurveTo(xct1,yct1,xct2,yct2,x,y)`, que desenha curva do ponto onde o pincel encontra-se até o ponto `x,y` utilizando dois pontos de controle com coordenadas `xct1,yct1` e `xct2,yct2`. A figura sobre o fundo quadriculado foi construída para ilustrar o significado dos pontos de controle: a curva é desenhada com o método `bezierCurveTo()`, utilizando os pontos de controle indicados pelos círculos vermelhos.

Sua vez.. (11-5)

Modifique o exemplo anterior de modo a explorar o que acontece quando as coordenadas dos pontos de controle são modificadas. O que acontece quando os pontos de controle são os mesmos? Quando os valores de *x* ou *y* são os mesmos?

Transformações de coordenadas

Frequentemente o sistema de coordenadas próprio do canvas (com origem 0,0 no canto superior esquerdo e unidades contadas em pixels) é muito inconveniente para a modelagem de sistemas físicos e outras aplicações. Para contornar esse problema, o canvas dispõe de alguns métodos de transformação de coordenadas: `translate()`, `rotate()`, `scale()`, `transform()` e `setTransform()`, que serão explorados em detalhes a seguir.

`translate(x,y)`

O método `translate(x,y)` move a origem do canvas para as coordenadas `x,y`. Deste modo, para um canvas 300 × 300, a instrução `translate(150,150)` move a origem, anteriormente no canto superior esquerdo, para o centro do canvas, fazendo com que passe a mostrar elementos com coordenadas entre −150 e 150.

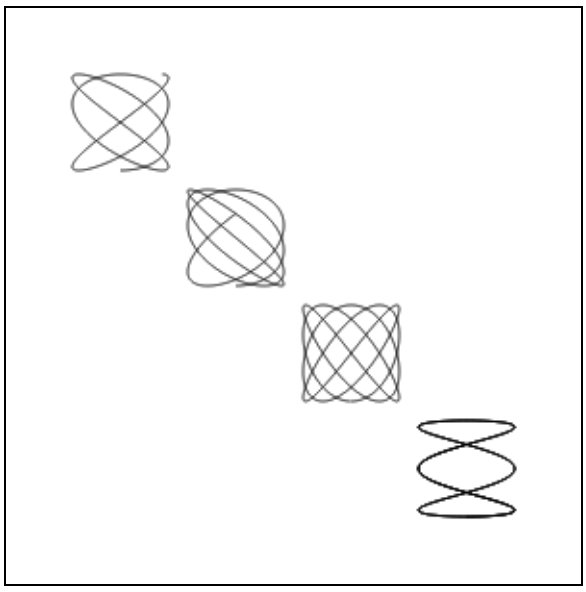
No exemplo a seguir, é definida uma função que gera uma figura de Lissajous (uma combinação de senos e cossenos nas direções *x* e *y*) centrada em uma origem (0,0). No laço `for` do script são desenhadas quatro figuras transladadas de 60 pixels nas direções *x* e *y*, em relação à posição prévia.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="300"
        style="border:1px solid black">
</canvas>
</p>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
```

```
for (var i=0;i<4;i++) {
   ctx.translate(60,60);
   var Tx = 100*parseInt(1+Math.random()*7);
   var Ty = 100*parseInt(1+Math.random()*9);
   lissajous(ctx,Tx,Ty);
}

function lissajous(lctx,Tx,Ty) {
   var x = 0;
   var y = 25;
   lctx.beginPath();
   lctx.lineWidth = 1;
   lctx.strokeStyle = "black";
   lctx.moveTo(x,y);
   for (var t=0;t<2000;t++) {
      x = 25 * Math.sin(2*Math.PI*(t/Tx));
      y = 25 * Math.cos(2*Math.PI*(t/Ty));
      lctx.lineTo(x,y);
   }
   lctx.stroke();
}

</script>
```

Resultado:



Quando o canvas é inicializado, o pincel está na posição (0,0), que é o canto superior esquerdo do canvas. Na primeira interação do laço for, a origem do canvas é movida para a posição (60,60) e a figura de Lissajous é desenhada neste referencial. Na segunda interação, um deslocamento adicional de (60,60) é dado à origem e uma nova figura desenhada em torno desse ponto. O resultado final é que as figuras são desenhadas em torno das coordenadas (60,60), (120,120), (180,180) e (240,240).

Note que, neste exemplo, uma variável global `ctx` é utilizada para receber a referência do contexto gráfico obtido do canvas. Esta variável é passada como parâmetro para a função `lissajous()` que a referencia internamente como a variável `lctx`. Esta estratégia pode ser

interessante para lidar com documentos com vários canvas, em que a confusão de nomes pode prejudicar a visualização.

Sua vez.. (11-6)

(a) Modifique o exemplo anterior de modo utilize dois laços `for` para desenhar 16 figuras de Lissajous sobre o canvas. (b) Remova a aleatoriedade dos períodos de modo que o seu "mapa" mostre como ficam as figuras com períodos indexados pela posição (isto é, 1,1 a 1,4 para a primeira linha, 2,1 a 2,4 para a segunda, 3,1 a 3,4 para a terceira e 4,1 a 4,4 para a quarta).

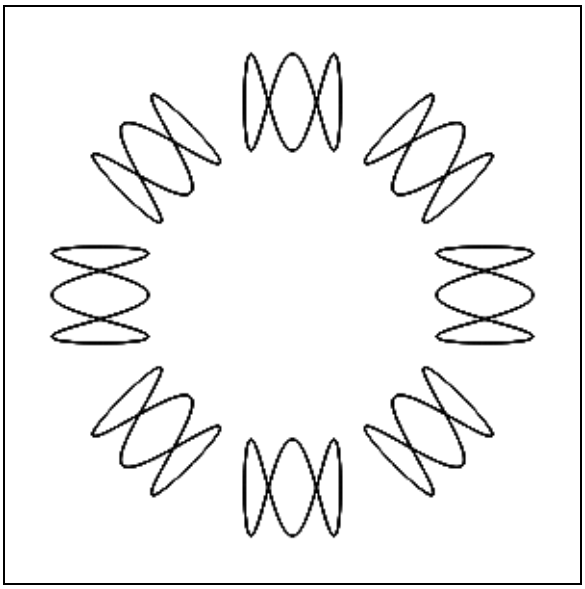
`rotate(ang)`

O sistema de coordenadas do canvas pode ser rodado de um ângulo `ang` (em radianos) utilizando-se o método `rotate(`*`ang`*`)`. O exemplo a seguir primeiro translada a origem do canvas para o centro e depois chama a função `lissajous()` definida no exemplo anterior dentro de um laço `for`, onde uma translação e uma rotação são utilizadas para posicionar cada nova figura (para que esse exemplo funcione, a função `lissajous()` deve ser incluída no script).

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="300"
        style="border:1px solid black">
</canvas>
</p>

<script>

var ctx = document.getElementById("cnv").getContext("2d");

ctx.translate(150,150);
for (var i=0;i<8;i++) {
   ctx.save();
   ctx.translate(100,0);
   lissajous(ctx,100,300);
   ctx.restore();
   ctx.rotate(Math.PI/4);
}

</script>
```

Resultado:



O exemplo apresenta dois métodos extremamente úteis: `save()` (salva) e `restore()` (restaura). Como seus nomes indicam, esses métodos salvam e restauram a configuração do canvas no momento em que são chamados. Tudo o que for feito após o um `save()` (transformações de coordenadas, mudanças de espessuras de linhas, especificações de cores etc.) será "esquecido" após um `restore()`.

Quando o laço inicia, o pincel está em (0,0), no centro do canvas, e a configuração é salva. A seguir, é feita uma translação de 100 pixels para a direita, desenhada a figura e restaurada a configuração anterior, isto é, o pincel é recolocado em (0,0). A última instrução do laço gira o sistema de coordenadas em 45 graus, o que faz com que, na segunda interação, os 100 pixels da translação não sejam mais para a direita, mas a sudeste, e assim por diante.

> Neste exemplo em particular, seria mais econômico utilizar um `translate(-100,0)` após desenhar a figura, ao invés de chamar os métodos `save()` e `restore()`. Entretanto, a estratégia utilizada é mais genérica, o que pode ser mais conveniente quando uma série de tranformações são feitas e têm que ser revertidas.

`scale(sx,sy)`

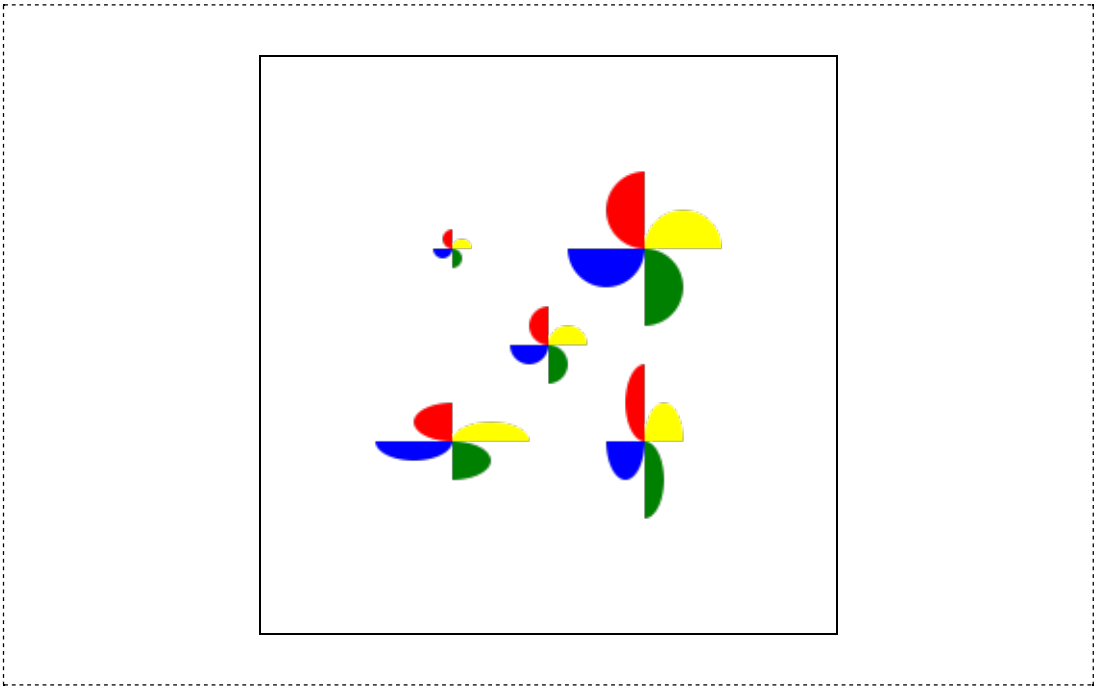
O exemplo a seguir ilustra o uso de mais uma transformação, a transformação de escala, bem como outro exemplo de uso dos métodos `save()` e `restore()`. No exemplo, a função `desenhaCatavento()` é chamada em cinco configurações diferentes do canvas. Na primeira vez, é desenhado o catavento no centro do canvas, em escala 1:1; na segunda, é desenhado o catavento no canto superior esquerdo, com metade do tamanho; na terceira, no canto superior direito, com o dobro do tamanho; e, nas últimas duas com cada uma das direções em escala 2:1.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="300"
```

```
        style="border:1px solid black">
</canvas>
</p>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

ctx.save();
ctx.scale(1,1);
ctx.translate(150,150);
desenhaCatavento(ctx);
ctx.restore();

ctx.save();
ctx.scale(0.5,0.5);
ctx.translate(200,200);
desenhaCatavento(ctx);
ctx.restore();

ctx.save();
ctx.scale(2,2);
ctx.translate(100,50);
desenhaCatavento(ctx);
ctx.restore();

ctx.save();
ctx.scale(2,1);
ctx.translate(50,200);
desenhaCatavento(ctx);
ctx.restore();

ctx.save();
ctx.scale(1,2);
ctx.translate(200,100);
desenhaCatavento(ctx);
ctx.restore();

function desenhaCatavento(lctx) {
   lctx.beginPath();
   lctx.arc(+10,0,10,0,Math.PI,true);
   lctx.fillStyle = "yellow";
   lctx.fill();
   lctx.beginPath();
   lctx.arc(-10,0,10,0,Math.PI,false);
   lctx.fillStyle = "blue";
   lctx.fill();
   lctx.beginPath();
   lctx.arc(0,+10,10,Math.PI/2,Math.PI*3/2,true);
   lctx.fillStyle = "green";
   lctx.fill();
   lctx.beginPath();
   lctx.arc(0,-10,10,Math.PI/2,Math.PI*3/2,false);
   lctx.fillStyle = "red";
   lctx.fill();
}
</script>
```

Resultado:



Note que os argumentos para o método `translate()` que vêm após a chamada ao método `scale()` devem ser adaptados para a nova escala. Uma maneira de amarrar uma coisa com outra é definir fatores de escala `xfat` e `yfat`, que seriam utilizados tanto na chamada a `scale()` quanto na chamada a `translate()`:

```
ctx.scale(xfat,yfat);
ctx.translate(posx/xfat,posy/yfat);
```

Sua vez.. (11-7)

Modifique o exemplo anterior de modo que os desenhos sejam feitos dentro de um único laço `for` com os argumentos da escala e da translação armazenados em objetos `Array()`.

`tranform()` e `setTranform()`

Esses métodos são uma síntese das outras transformações. Quando o canvas é inicializado, é atribuída a ele uma *matriz de transformação* **T** que, multiplicada pelo vetor de coordenadas **X**, leva um novo vetor de coordenadas **X'**. Em linguagem matricial, essa operação seria:

$$\mathbf{X'} = \mathbf{T} \cdot \mathbf{X}$$

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} m_{11} & m_{21} & dx \\ m_{12} & m_{22} & dy \\ 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Essa equação matricial corresponde às equações:

$$x' = m_{11}\, x + m_{21}\, y + dx$$
$$y' = m_{12}\, x + m_{22}\, y + dy$$
$$1 = 1$$

Se (m_{11}, m_{12}, m_{21}, m_{22}) = (cosθ, sinθ, −sinθ, cosθ), essa operação será uma rotação de um ângulo θ, seguida de uma translação (dx, dy).

O método `transform(m11, m12, m21, m22, dx, dy)` multiplica a matriz de transformação corrente, que inicialmente é uma matriz identidade, pela matriz dada pelos coeficientes (m11, m12, m21, m22, dx, dy). O método `setTransform(m11, m12, m21, m22, dx, dy)` restaura a matriz de transformação inicial e a multiplica pela matriz composta pelos argumentos passados ao método.

> Note que o método `transform()` multiplica sempre a matriz *corrente* pela matriz fornecida. Se, por exemplo, tratar-se de uma matriz que leva a uma rotação de 1 grau, 10 aplicações sucessivas levarão a uma rotação final de 10 graus, o que não aconteceria com a aplicação do método `setTransform(m11, m12, m21, m22, dx, dy)` que levaria sempre ao mesmo resultado.

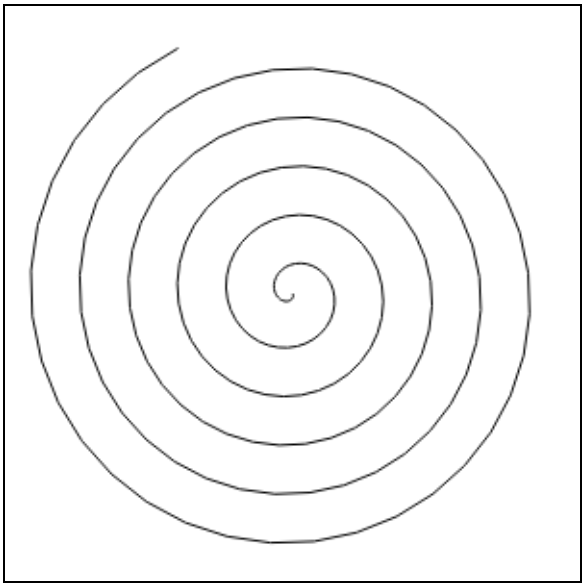
O exemplo a seguir mostra como produzir uma espiral desenhando apenas linhas retas e aplicando uma matriz de transformação sucessivas vezes.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="300"
        style="border:1px solid black">
</canvas>
</p>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

ctx.scale(0.1,0.1);
ctx.translate(1500,1500);
ctx.lineWidth = 10;

var xprev = 0;
var yprev = 0;
var xcurr, ycurr;

var sin = Math.sin(Math.PI/18);
var cos = Math.cos(Math.PI/18);

for (var i=0;i<200;i++) {
   xcurr = 5*i;
   ycurr = 5*i;
   ctx.beginPath();
   ctx.moveTo(xprev,yprev);
   ctx.transform(cos,sin,-sin,cos,0,0);
   ctx.lineTo(xcurr,ycurr);
   ctx.stroke();
   xprev = xcurr;
```

```
    yprev = ycurr;
}
</script>
```

Resultado:



O script inicialmente ajusta a escala, ampliando o canvas de um fator 10. A seguir, muda a origem para o centro e acerta o valor da espessura do traço para 10 unidades. Isto é feito porque agora uma unidade corresponde a um décimo de um pixel e, se a espessura não fosse aumentada, mal se veria a figura.

A seguir, como os valores dos senos e cossenos serão utilizados muitas vezes, o script os deixa pré-calculados nas variáveis `sin` e `cos`, para aumentar a eficiência do programa. Antes do laço, são declaradas e inicializadas as variáveis que armazenarão as coordenadas iniciais (`xprev`,`yprev`) e finais (`xcurr`,`ycurr`) utilizadas no desenho das retas. A cada interação do laço `for`, o pincel é movido para o ponto inicial, a rotação é realizada e a linha desenhada. Desenhada a linha, às coordenadas iniciais do ponto inicial da próxima linha são atribuídas as coordenadas do ponto final da linha recém-desenhada.

Sua vez... (11-8)

Modifique o exemplo anterior de modo a colorir aleatoriamente as retas desenhadas (veja como fazer isso no exemplo sobre as curvas de Bezier) e verificar a variação de seus tamanhos. Aproveite também para ver como fica a figura quando é comentada a linha que faz a transformação de coordenadas.

Sistema do canvas × sistema do modelo

Talvez o uso mais importante das transformações do canvas seja a sua adaptação para um sistema de coordenadas do sistema que queremos modelar. Vamos supor, por exemplo,

um oscilador harmônico amortecido cuja amplitude em função do tempo seja dada por:

$$x(t) = A_0 \cos(\omega t) \exp(-bt)$$

Queremos fazer um gráfico da posição em função do tempo para uma amplitude inicial A_0 = 1 cm, uma frequência angular $\omega = 2\pi$ rad/s (1 rotação por segundo), um coeficiente de amortecimento $b = 0.5\ \mathrm{s}^{-1}$ para um intervalo de tempo entre 0 e 10 segundos. A direção x de nosso canvas terá que representar valores entre $x_{\mathrm{mín}} = 0$ e $x_{\mathrm{máx}} = 10$ e a direção y entre $y_{\mathrm{mín}} = -1$ e $y_{\mathrm{máx}} = 1$, sendo o −1 no extremo inferior do canvas e o 1 no extremo superior. Vamos supor um canvas com largura $c_{\mathrm{x}} = 300$ pixels e altura $c_{\mathrm{y}} = 150$ pixels.

Estes dois sistemas estão representados nas figuras a seguir. Note que a origem do eixo y no sistema de coordenadas do canvas está na borda *superior* do canvas e que os valores da coordenada y crescem para *baixo*, diferentemente do usualmente encontrado em matemática e física.

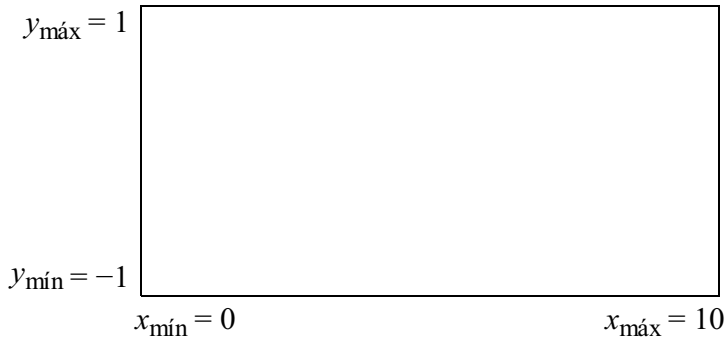


Figura 11.1 Sistema de coordenadas do modelo.

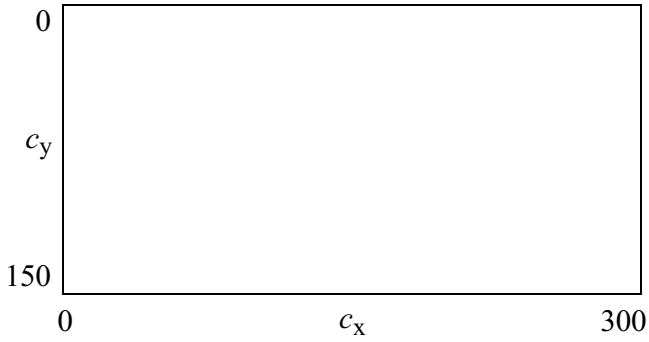


Figura 11.2 Sistema de coordenadas do canvas.

Chamando as coordenadas do modelo de x_{m} e y_{m} e as coordenadas do canvas de x_{c} e y_{c}, as transformações que levam de um sistema a outro são dadas por:

$$x_{\mathrm{c}} = A_{\mathrm{x}} + B_{\mathrm{x}} \cdot x_{\mathrm{m}}$$
$$y_{\mathrm{c}} = A_{\mathrm{y}} + B_{\mathrm{y}} \cdot y_{\mathrm{m}}$$

Colocando nestas equações os valores conhecidos das coordenadas nos dois sistemas, obtemos para a coordenada x:

$$0 = A_{\mathrm{x}} + B_{\mathrm{x}} \cdot x_{\mathrm{mín}}$$

$$c_x = A_x + B_x \cdot x_{máx}$$

e para a coordenada *y*:

$$0 = A_y + B_y \cdot y_{máx}$$
$$c_y = A_y + B_y \cdot y_{mín}$$

Resolvendo para *A* e *B* obtemos, para a coordenada *x*:

$$B_x = c_x / (x_{máx} - x_{mín})$$
$$A_x = - B_x \cdot x_{mín} = - [c_x / (x_{máx} - x_{mín})] \cdot x_{mín}$$

e para a coordenada *y*:

$$B_y = c_y / (y_{mín} - y_{máx})$$
$$A_y = - B_y \cdot y_{máx} = - [c_y / (y_{mín} - y_{máx})] \cdot y_{máx}$$

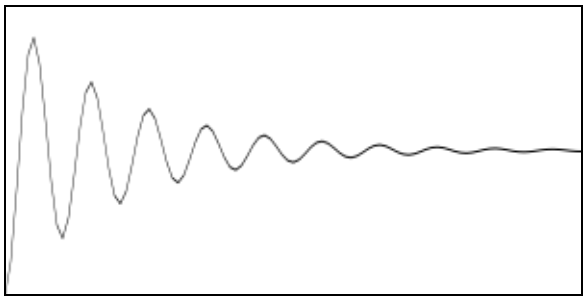
O exemplo a seguir implementa utiliza um conjunto de instruções para estabelecer os limites do canvas e do modelo e a instrução `setTranform()` para definir um sistema de coordenadas apropriado para o sistema físico em questão.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="150"
        style="border:1px solid black">
</canvas>
</p>

<script>
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
var cw  = cnv.width;
var ch  = cnv.height;
var xmin = 0;
var xmax = 10;
var ymin = -1;
var ymax = +1;
var m11 = cw/(xmax-xmin);
var m12 = 0;
var m21 = 0;
var m22 = - ch/(ymin-ymax);
var dx  = - cw*xmin/(xmax-xmin);
var dy  = - ch*ymax/(ymin-ymax);
ctx.setTransform(m11,m12,m21,m22,dx,dy);
ctx.lineWidth = 1/Math.max(m11,m22);

function OHSA(Ao,w,b,t) {
   return (Ao * Math.cos(w*t) * Math.exp(-b*t));
}

var t = 0;
var A = OHSA(1,2*Math.PI,0.5,0);
var dt = 0.1;
ctx.beginPath();
ctx.moveTo(t,A);
for (var i=0;i<100;i++) {
   t = t + dt;
   A = OHSA(1,2*Math.PI,0.5,t);
   ctx.lineTo(t,A);
```

```
}
ctx.stroke();

</script>
```

Resultado:



No início do script são declaradas todas as variáveis necessárias para a realização da transformação de coordenadas e são calculados os coeficientes da matriz de transformação segundo o esquema descrito anteriormente. O método `setTransform()` é chamado com os parâmetros calculados, e a partir desse ponto o novo sistema de coordenadas está vigente. Como a espessura da linha fica afetada pela transformação de coordenadas, esta também deve ser ajustada utilizando o maior fator de escala dentre os dois calculados.

Note que o script também utiliza, por conveniência, a função `OHSA(Ao,w,b,t)`, que recebe como argumentos os valores da amplitude inicial, a frequência angular, o coeficiente de amortecimento e um instante de tempo, e retorna o valor da amplitude nesse instante.

Sua vez... (11-9)

Modifique o exemplo anterior de modo que o fator de amortecimento seja 10 vezes menor que o especificado, e que a escala na direção horizontal seja modificada de modo a fazer com que o gráfico resultante também esteja próximo a zero no último terço do quadro.

Animações

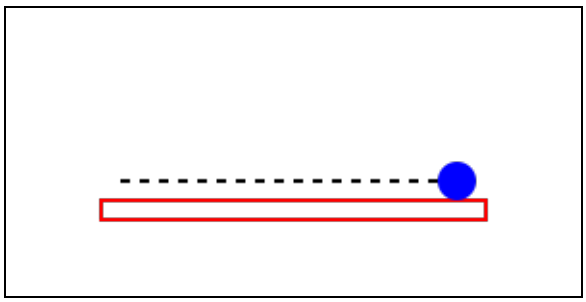
Animações no canvas são feitas de maneira semelhante às animações discutidas no capítulo sobre tempo, utilizando o método `setInterval()`. O exemplo a seguir retoma o exemplo da bola sobre a mesa do início deste capítulo, fazendo com que vá de um lado a outro.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="150"
```

```
        style="border:1px solid black">
</canvas>
</p>

<script>
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
var xbola = 60;
var timerBola = setInterval("passo()",100);

function passo() {

   ctx.clearRect(0,0,300,150);

   // retângulo vermelho delineado
   ctx.beginPath();
   ctx.rect(50,100,200,10);
   ctx.strokeStyle = "red";
   ctx.lineWidth = 2;
   ctx.stroke();

   // linha preta horizontal
   ctx.beginPath();
   for (var i=0;i<19;i++) {
      ctx.moveTo(60 + 10*i,90);
      ctx.lineTo((60 + 10*i + 5),90);
   }
   ctx.lineWidth = 2;
   ctx.strokeStyle = "black";
   ctx.stroke();

   // bola azul preenchida
   ctx.beginPath();
   xbola = xbola + 5;
   ctx.arc(xbola,90,10,0,2*Math.PI,true);
   ctx.fillStyle = "blue";
   ctx.fill();

   if (xbola>230) clearInterval(timerBola);

}
</script>
```

Resultado:



Compare o exemplo acima com o do início do capítulo e analise atentamente as diferenças. Em primeiro lugar, o procedimento de desenho de *tudo* o que desejamos que apareça no canvas foi colocado dentro de uma função `passo()`. Esta função é passada como argumento para o método `setInterval()` no começo do script e é ela que vai repetir-se (a

cada 100 milissegundos) até que a coordenada *x* da bola ultrapasse o valor estipulado no `if` da última linha de código da função, quando é desligada.

A animação propriamente dita é feita adicionando 5 unidades à coordenada *x* da bola a cada disparo do temporizador. Note que, no início da função, o canvas é totalmente limpo com o método `clearRect(x,y,w,h)`, onde `x,y` são as coordenadas iniciais do retângulo a ser apagado e `w,h` sua largura e altura.

Os desenhos no canvas, apesar de feitos um a um, resultam apenas em pixels coloridos na tela, e não em estruturas independentemente gerenciáveis. Assim, quando muitos elementos têm que ser apagados e redesenhados no canvas durante uma animação, ela pode ficar mais lenta do que o desejável. Tecnologias vetoriais, como o SVG (*scalable vector graphics*), também compatíveis com o HTML, são uma opção, mas não serão abordadas neste texto.

Sua vez... (11-10)

Modifique o exemplo anterior de modo que (a) o movimento horizontal seja substituído por um lançamento oblíquo a 45 graus com uma velocidade inicial tal que a bola atinja a outra extremidade da mesa e (b) ao invés de parar, o movimento fique sendo repetido indefinidamente (*dica:* ao invés de desligar o temporizador quando a condição de fim for atingida, pense em reinicializar as variáveis).

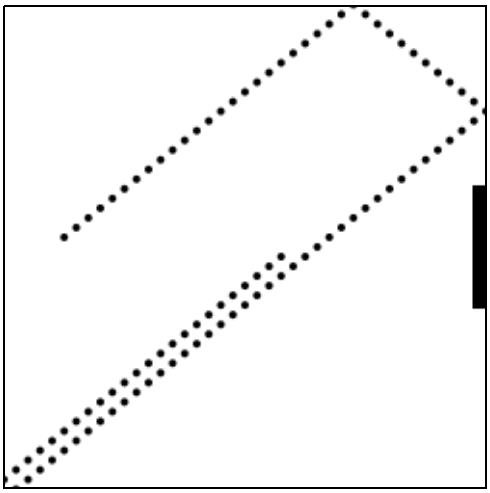
O exemplo a seguir modela uma bola movendo-se dentro de uma caixa bidimensional. A bola sai sempre do mesmo ponto, mas com uma velocidade (vetorial) aleatoriamente escolhida. No modelo, não há gravidade ou dissipação de energia nas colisões, e a bola sempre anda em linha reta e ficaria quicando indefinidamente caso sua trajetória não fosse interrompida ao atingir o "gol".

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="250" height="250"
        style="border:1px solid black">
</canvas>
</p>

<script>
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
var cW = cnv.width, cH = cnv.height;
var xmin = -1, xmax = +1;
var ymin = -1, ymax = +1;
ctx.setTransform(cW/(xmax-xmin),0,0,-cH/(ymax-ymin),cW/2,cH/2);
var rpix = (xmax-xmin)/cW;
ctx.lineWidth = rpix;

var x = -0.8;
var y = 0;
var vx = 1;
var vy = -1 + 2 * Math.random();
var dt = 0.05;
var tictac = setInterval("passoBola()",50);
```

```
function passoBola() {
    ctx.clearRect(-1,-1,2,2);
    if ((x>0.95*xmax)&&(y<0.25*ymax)&&(y>0.25*ymin))
     clearInterval(tictac);
    if ((x>xmax)||(x<xmin))
        vx = -vx;
    if ((y>ymax)||(y<ymin))
        vy = -vy;
    x = x + vx * dt;
    y = y + vy * dt;
    ctx.beginPath();
    ctx.rect(0.95*xmax,0.25*ymin,0.05,0.5);
    ctx.fill();
    ctx.beginPath();
    ctx.arc(x,y,2*rpix,0,2*Math.PI,0);
    ctx.fill();
}
</script>
```

Resultado:



No primeiro bloco do script são definidas várias variáveis que auxiliam na construção da transformação de coordenadas. As variáveis `xmin`, `xmax`, `ymin` e `ymax` contêm os limites do espaço considerado no modelo (o "mundo"); as variáveis `cW` e `cH` contém a largura e a altura do canvas. Como especificada, a instrução `setTransform` coloca o eixo *x* (horizontal) e o eixo *y* (vertical) no meio do canvas, com a origem no centro dele. Os extremos nas duas direções passam a ser −1 e +1. O sinal de menos no quarto parâmetro faz com que o eixo *y*, que originalmente aponta para baixo, passe a apontar para cima.

A instrução `var rpix = (xmax-xmin)/cW` define uma variável de conveniência, que contém o tamanho aproximado de 1 pixel no novo sistema de coordenadas. Esse tamanho pode ser utilizado como referência para definir a espessura mínima de uma linha e para o raio mínimo de um círculo (arco) a ser desenhado no canvas.

A função `passoBola()` limpa o canvas, verifica se a bola bateu no "gol" (desligando o temporizador, caso isso ocorra) ou nas paredes (invertendo o sinal da respectiva componente da velocidade, caso isso ocorra). Finalmente, atualiza as coordenadas e desenha a bola.

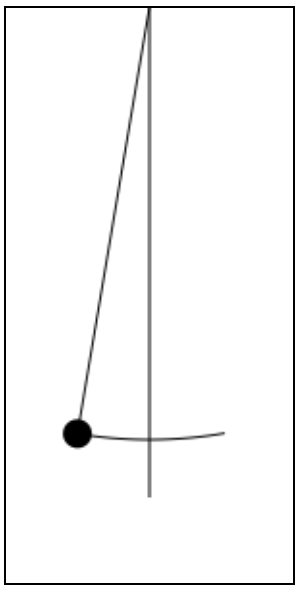
Note particularmente os usos dos operadores lógicos `&&` (E) e `||` (OU) para a construção da condição de fim e de reversão das velocidades.

Sua vez.. (11-11)

Modifique o exemplo anterior de modo a (a) fazer com que a bola saia do centro do canvas e (b) incluir um segundo "gol" na lateral esquerda do canvas (oposto ao existente).

Exercícios

1. A figura abaixo ilustra um pêndulo simples. Reproduza-a utilizando os métodos de desenho em um canvas.



Algumas dicas:

a. Escolha uma escala proporcional. A figura foi gerada em um canvas 150 × 300 com as coordenadas na direção *x* variando de −0.5 a 0.5 e na direção *y* variando de 0 a 2.

b. O desenho tem quatro elementos: uma esfera, uma linha ligando o ponto de sustentação à esfera, uma linha vertical e um arco que, no exemplo, tem 10 graus à esquerda e 10 graus à direita da linha vertical.

c. No sistema de coordenadas do pêndulo, a linha vertical faz um ângulo de 270 graus com a origem dos ângulos.

d. Codifique cada um deles separadamente, isto é, com um `beginPath()` encerrado

por um `stroke()` ou um `fill()`.

2. Anime o pêndulo do exercício anterior. Algumas dicas:

   a. O ângulo que o pêndulo faz com a vertical em função do tempo pode ser dado por $\theta(t) = \theta_0 \cos(2\pi f t)$, onde θ_0 é o ângulo no instante $t = 0$ e f é a frequência.

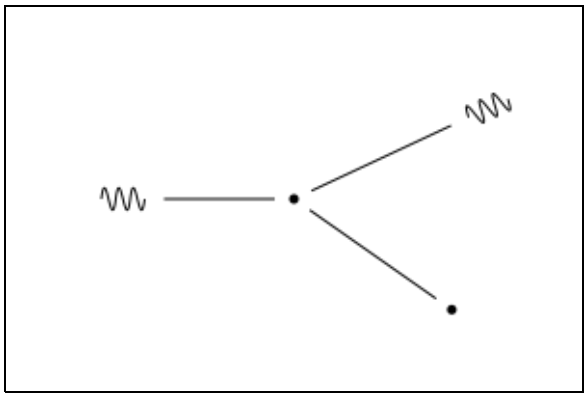
   b. A frequência f depende do comprimento L do pêndulo e da aceleração da gravidade g: $f = (g/L)^{1/2}$.

   c. As coordenadas x e y são dadas por: $x = L \operatorname{sen}\theta$ e $y = H - L \cos\theta$, onde H é a altura do suporte onde o pêndulo está preso.

   d. Para uma animação realista, utilize o objeto `Date()` para obter o tempo inicial e o tempo de cada atualização, utilizando a diferença entre eles como valor de t. Não se esqueça que a diferença é dada em milissegundos (volte ao exemplo do cronômetro no capítulo sobre tempo).

3. Modifique o script do exercício anterior para incluir um amortecimento (volte ao problema do oscilador harmônico amortecido discutido no texto) e faça gráficos das coordenadas x e y do pêndulo em função do tempo.

4. Uma das formas da interação da radiação com a matéria é o chamado espalhamento Compton. Nele, uma partícula de luz, o fóton, colide com uma partícula carregada, por exemplo um elétron. Após a colisão, um novo fóton com uma energia menor sai com um ângulo e o elétron com outro ângulo em relação à direção de incidência. A diferença entre a energia do fóton incidente e do fóton espalhado, bem como os ângulos com que eles saem após a colisão podem ser calculados utilizando as leis da conservação da energia e do momento linear. A figura abaixo ilustra o processo:



   Escreva um programa para desenhar a figura. Algumas dicas:

   a. Foi definido um canvas 300 × 200 com uma escala de -15 a 15 na direção x e -10 a 10 na direção y.

   b. O fóton e o elétron antes da colisão estão em (-10,0) e (0,0), respectivamente.

   c. Os fótons podem ser desenhados concatenando curvas Bézier ou funções harmônicas (seno ou cosseno).

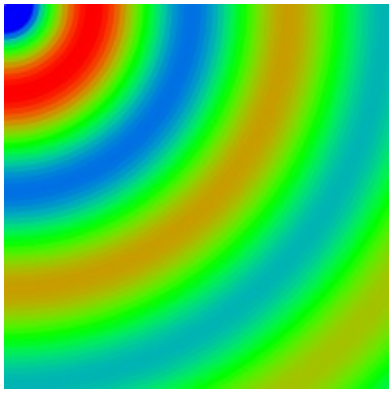
   d. Para desenhar o fóton espalhado, o canvas foi rodado de 25 graus. No sistema rodado, as coordenadas do fóton são (10,0).

e. Para desenhar o elétron espalhado, o canvas foi rodado de -35 graus. No sistema rodado, as coordenadas do elétron espalhado são (10,0).

f. Utilize os métodos `save()` e `restore()` para preservar e recuperar o sistema de coordenadas original do canvas.

5. A figura ao lado ilustra um tipo de *passeio aleatório* (andar do bêbado), em que um objeto dá um passo de tamanho arbitrário, mas sempre igual, em uma direção aleatoriamente escolhida (um ângulo entre 0 e 2π sorteado ao acaso, por exemplo). Faça um programa que modele o passeio aleatório da figura. *Dica:* o canvas ao lado tem 100 pixels × 100 pixels, o início do passeio se dá em seu centro e cada passo tem 2 pixels.

6. Refaça o programa que modela uma onda bidimensional animada, feito utilizando uma tabela no capítulo anterior, mas agora utilizando o canvas.



7. Faça um relógio analógico que mostre o tempo real. Utilize os métodos do objeto `Date()` para obter horas, minutos e segundos do relógio do sistema e transforme seus valores em ângulos para os ponteiros.

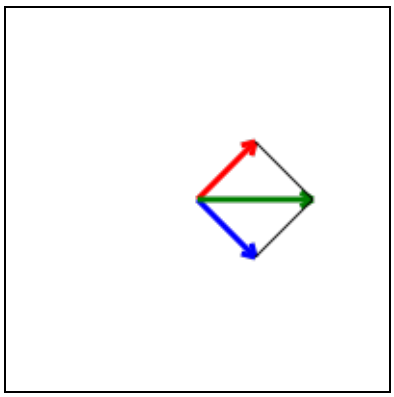
8. Volte ao capítulo sobre interação e utilize caixas de texto para construir um documento que contenha entradas para as componentes de dois vetores e saídas para as componentes do vetor soma resultante. Além disto, o script deve mostrar em um canvas os vetores fornecidos e o vetor resultante.

$\mathbf{v}_1$ = 3 **i** + 3 **j**

$\mathbf{v}_2$ = 3 **i** + -3 **j**

SOMA

$\mathbf{v}_3$ = 6 **i** + 0 **j**



Bibliografia comentada

1. BOS, B., CELIK, T., HICKSON, I. (Eds.) **Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, W3C Recommendation 07 June 2011.** Cambridge: W3C Consortium, 2011. Disponível em: http://www.w3.org/TR/CSS21/>. Acesso em: 29/04/2015.

   Este é o documento em que o consórcio internacional que gerencia a web propõe essencialmente como devem funcionar os navegadores no que se refere à interpretação das declarações de estilo. Não é um documento destinado a estudantes de programação, mas principalmente a desenvolvedores de navegadores e similares, mas vale a pena uma visita.

2. HICKSON, I. (Ed.) **HTML5: A vocabulary and associated APIs for HTML and XHTML. W3C Proposed Recommendation 16 September 2014.**. Cambridge: W3C (World Wide Web Consortium). Disponível em . Acesso em 29/10/2015.

   Este é o enorme documento em que o consórcio internacional que gerencia a web propõe essencialmente como devem funcionar os navegadores no que se refere à interpretação de HTML. Não é um documento destinado a estudantes de programação, mas principalmente a desenvolvedores de navegadores e similares, mas vale a pena uma visita.

3. LAURSEN, O., SCHNUR, D. **FLOT: an attractive JavaScript plotting for jQuery, v. 0.8.3.** Alborg: IOLA, 2015. Disponível em . Acesso em: 29/10/2015.

   É de onde você pode baixar o FLOT, *plug-in* para facilitar a geração de gráficos bonitos com funções ou dados. Vale a pena olhar em detalhes todos os exemplos para ter uma dimensão de suas potencialidades. Se você achar que precisa de mais, experimente buscar por HIGHCHARTS, por exemplo. Se realmente quiser se profissionalizar em visualização de dados, tente buscar pelo D3.

4. POWELL, T. A. **HTML & CSS: the complete reference (Fifth Edition).** New York: McGraw Hill, 2010.

   Um livro que talvez você queira ter quando não tem uma conexão com a internet. Aborda HTML e CSS em detalhes, mas não tem nada sobre JavaScript ou programação.

5. PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., FLANNERY, B. P. **Numerical Recipes (Third Edition)**. Cambridge: Cambridge University Press, 2007. Disponível em . Acesso em: 29/04/2015.

   Provavelmente a referência mais conhecida e citada sobre métodos numéricos. Tem de tudo, e códigos para tudo em várias linguagens. Ainda não há versão para JavaScript, mas como há para C e C++, a adaptação é fácil e direta.

6. REFSNES Data. **w3schools Tutorials**. 2015. Disponível em: . Acesso em 22/10/2015.

   Esse é "o" site de referência para HTML, CSS, JavaScript e várias outras tecnologias. E você nem precisa ter entre os seus favoritos: qualquer busca que incluir "JavaScript" como uma das palavras

chave vai trazer um link para esse site no topo da lista.

7. SCHERER, C. **Métodos Computacionais da Física.** São Paulo: Livraria da Física, 2005.

   Um tradicional livro de física computacional, em português, mais voltado para aqueles interessados em seguir algum tipo de pesquisa em física utilizando métodos numéricos.

8. WIRFS-BROCK, A. (Ed.) **ECMA-262 ECMAScript Language Specification, 6 Edition, June 2015.** Geneva: ECMA International, 2015. . Acesso em: 06/10/2015.

   Este é o documento que contém a especificação do que é comumente chamado de JavaScript, cujo nome oficial é ECMA-262. É um documento longo, não escrito para ser lido por um estudante de programação, mas para os desenvolvedores de navegadores. No entanto, um passeio pelo documento pode dar uma boa dimensão do que é uma linguagem de programação e de como são pensadas as suas características.