Programação Visual e Autoria Web

HTML5 Canvas

Universidade Federal do Rio Grande do Norte

Introdução

- O elemento canvas faz parte do HTML5 e permite a renderização dinâmica de formas 2D e imagens de bitmap.
- O Canvas foi introduzido inicialmente pela Apple para uso em seu no Mac OS X WebKit em 2004.
- Em 2006 foi padronizado pelo Grupo de Trabalho de Tecnologia de Aplicação de Hipertexto da Web (WHATWG) nas novas especificações dos HTML.

Canvas

- O Canvas consiste em uma região desenhável definida no código HTML com atributos de altura e largura.
- Um código JavaScript pode acessar a área através de um conjunto completo de funções de desenho permitindo a geração dinâmica de gráficos.
- Usos comuns:
  - Criação de gráficos, animações, jogos e composição de imagens.

Canvas

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <meta http-equiv="X-UA-Compatible" content="ie=edge" />
        <title>Aula 13</title>
    </head>
    <body>
        <canvas id="canvas" width="200" height="200">
            Texto exibido se o browser não der suporte ao HTML5.
        </canvas>
        <script src="script.js"></script>
    </body>
</html>
```

Canvas

```
let canvas = document.getElementById("canvas");
let context = canvas.getContext("2d");
context.fillStyle = "red";
context.fillRect(0, 0, 200, 200);
context.fillStyle = “yellow";
context.fillRect(5, 5, 190, 190);
```

Canvas

- O elemento <canvas> se parece muito com o elemento <img> com a diferença de não possuir os atributos src e alt.
- O <canvas> possui apenas os atributos width e height (ambos opcionais)
  - Se não forem especificados, o canvas será iniciado com 300 pixels de largura por 150 pixels de altura.
  - Pode ser redimensionado por CSS (evitar) ou pelo DOM.

Canvas

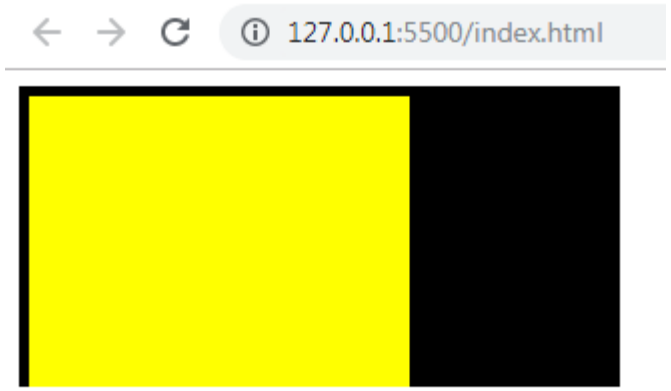
- O atributo id é comumente preenchido em um canvas, pois essa é a maneira mais fácil de obter sua referência no no script.
- O elemento <canvas> pode ser estilizado como qualquer imagem (margem, borda, fundo, etc). Contudo, essas regras não afetarão o desenho no canvas.
- Quando nenhuma regra de estilo for aplicada, o canvas iniciará totalmente transparente.

Canvas

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <meta http-equiv="X-UA-Compatible" content="ie=edge" />
        <title>Aula 13</title>
    </head>
    <body>
        <canvas id="canvas" style="background-color: black;">
            Texto exibido se o browser não der suporte ao HTML5.
        </canvas>
        <script src="script.js"></script>
    </body>
</html>
```

Canvas

```
let canvas = document.getElementById("canvas");
let context = canvas.getContext("2d");
context.fillStyle = "yellow";
context.fillRect(5, 5, 190, 190);
```



Canvas

- <canvas> cria uma superfície de desenho de tamanho com um ou mais contextos de renderização.
- Os contextos de renderização são usados para criar e manipular o conteúdo mostrado.
  - Foco no contexto de renderização 2D
  - Outros como WebGL usa um contexto 3D ("experimental-WebGL") baseado em OpenGL ES.

Canvas

- Incialmente o canvas possui a cor do background escolhido.
- Para que seja possível manipular o conteúdo o script precisa acessar o contexto de renderização e desenhar sobre ele.
- O elemento <canvas> tem um método chamado getContext(), usado para obter o contexto de renderização e suas funções de desenho.
  - getContext() recebe o tipo de contexto como parâmetro.
  - Para gráficos 2D utilizamos o contexto "2d".

```
let canvas = document.getElementById("canvas");
let context = canvas.getContext("2d");
```

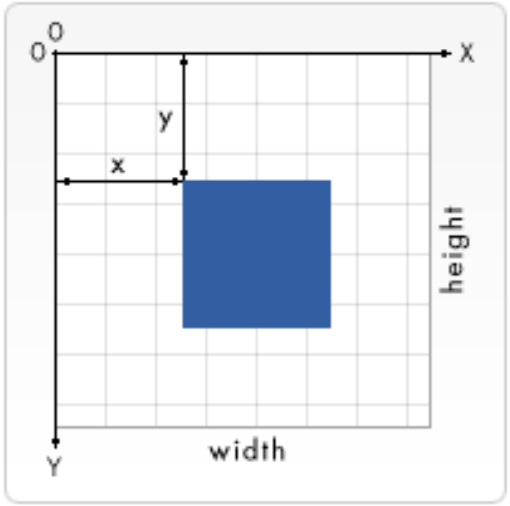
Propriedades básicas

fillStyle

Cor ou estilo para usar em formas internas.

strokeStyle

Cor ou estilo a ser usado nas linhas no contorno das formas



Desenhando caminhos

- Para criar uma camada usando caminhos (regiões ou paths) é necessário:
  - Criar a região de desenho.
  - Usar comandos de desenho para desenhar nesta região.
  - Fechar o caminho (path).
- O primeiro passo para criar um caminho é chamar o beginPath().
  - Internamente, caminhos são armazenados como uma lista de sub-caminhos (linhas, arcos, etc.) que juntos formam uma forma (shape).
  - Sempre que esse método é chamado, a lista é redefinida e é possível desenhar novas formas.

Desenhando caminhos

beginPath()

Cria um novo path. Uma vez criado, futuros comandos de desenho são direcionados do path atual para a construção de um novo path no canvas.

closePath()

Finaliza o path para fututos comandos de desenho, fazendo com que voltem a ser direcionados ao contexto.

stroke()

Desenha uma borda na camada.

fill()

Desenha uma forma sólida através de preenchimento.

moveTo(x,y)

Move a caneta (pen) para as coordenadas especificadas por x e y.

Nosso canvas básico paras exemplos

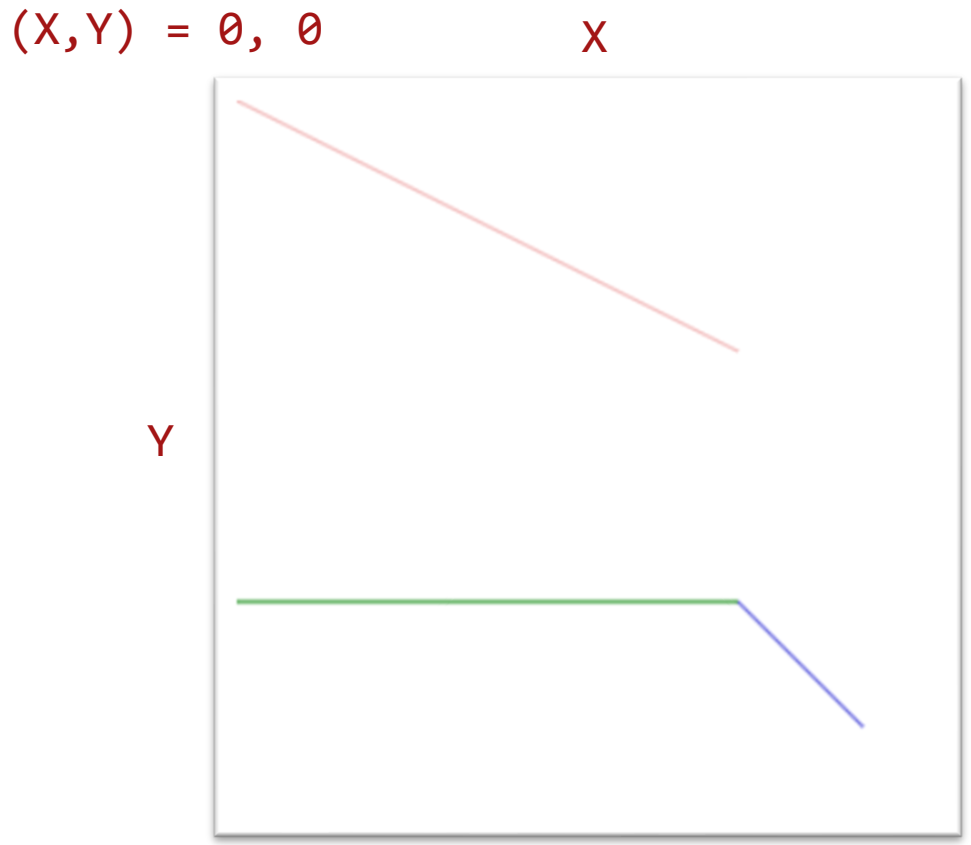
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <meta http-equiv="X-UA-Compatible" content="ie=edge" />
        <title>Aula 13</title>
    </head>
    <body>
        <canvas id="canvas" height="400" width="400">
            Texto exibido se o browser não der suporte ao HTML5.
        </canvas>
        <script src="script.js"></script>
    </body>
</html>
```

Desenhar caminho/linha

```
ctx.beginPath();
ctx.strokeStyle = "rgba(200, 0, 0, 0.5)";
ctx.moveTo(0, 0);
ctx.lineTo(200, 100);
ctx.closePath();
ctx.stroke();

ctx.beginPath();
ctx.strokeStyle = "rgb(0, 0, 200)";
ctx.moveTo(250, 250);
ctx.lineTo(200, 200);
ctx.closePath();
ctx.stroke();

ctx.beginPath();
ctx.strokeStyle = "green";
ctx.moveTo(200, 200);
ctx.lineTo(0, 200);
ctx.closePath();
ctx.stroke();
```



Desenhar triângulo

```
let canvas = document.getElementById("canvas");
let ctx = canvas.getContext("2d");

ctx.beginPath();
ctx.moveTo(20, 20);
ctx.lineTo(20, 100);
ctx.lineTo(70, 100);
ctx.closePath();
ctx.stroke();
```

Desenhar caminhos

- Quando a função fill() é chamada todas as formas abertas são fechadas automaticamente.
  - Não será necessário chamar closePath().
  - Isso não acontece quando você chamar stroke().

Desenhando retângulos

- O <canvas> suporta apenas a forma primitiva **retângulo**.
- Todas as outras formas são criadas a partir da combinação de um ou mais caminhos (paths), lista de pontos conectados por uma linha.

fillRect(x, y, width, height)

Desenha um retângulo preenchido.

strokeRect(x, y, width, height)

Desenha a borda do retângulo.

clearRect(x, y, width, height)

Limpa um retângulo específico, tornando-o totalmente transparente.

Desenhando retângulos

```
let canvas = document.getElementById("canvas");
let ctx = canvas.getContext("2d");

ctx.fillStyle = "blue";
ctx.fillRect(25, 25, 100, 100);
ctx.clearRect(45, 45, 60, 60);
ctx.strokeStyle = "red";
ctx.strokeRect(50, 50, 50, 50);
```

Desenhar Arco

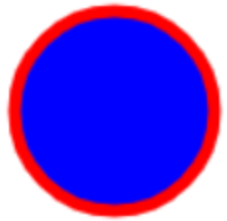
```
let canvas = document.getElementById("canvas");
let ctx = canvas.getContext("2d");

ctx.strokeStyle = "#ff0000";
ctx.beginPath();
//arc (x,y,r,anguloinicial,angulofinal, sentido)
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
ctx.lineWidth = "5";
ctx.stroke();
```

Desenhar Arco

```
let canvas = document.getElementById("canvas");
let ctx = canvas.getContext("2d");

ctx.strokeStyle = "#ff0000";
ctx.fillStyle = "#0000ff";
ctx.beginPath();
//arc (x,y,r,anguloinicial,angulofinal, sentido)
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
ctx.lineWidth = "5";
ctx.fill();
ctx.stroke();
```



Desenho de uma figura feliz

```
let canvas = document.getElementById("canvas");
let ctx = canvas.getContext("2d");

ctx.beginPath();
ctx.arc(75, 75, 50, 0, Math.PI * 2); // Círculo exterior
ctx.moveTo(110, 75);
ctx.arc(75, 75, 35, 0, Math.PI); // Boca
ctx.moveTo(65, 65);
ctx.arc(60, 65, 5, 0, Math.PI * 2); // Olho esquerdo
ctx.moveTo(95, 65);
ctx.arc(90, 65, 5, 0, Math.PI * 2); // Olho direito
ctx.stroke();
```

Desenho de uma figura triste

```
let canvas = document.getElementById("canvas");
let ctx = canvas.getContext("2d");

ctx.beginPath();
ctx.arc(75, 75, 50, 0, Math.PI * 2); // Círculo exterior
ctx.moveTo(110, 120);
ctx.arc(75, 120, 35, 0, Math.PI, true); // Boca, sentido anti-horário
ctx.moveTo(65, 65);
ctx.arc(60, 65, 5, 0, Math.PI * 2); // Olho esquerdo
ctx.moveTo(95, 65);
ctx.arc(90, 65, 5, 0, Math.PI * 2); // Olho direito
ctx.stroke();
```

Bibliografia Básica

- DUCKETT, J. **Javascript e Jquery**: desenvolvimento de interfaces web interativas. Rio de Janeiro: Alta Bokks, 2016
- FLANAGAN, D. **JavaScript**: O Guia Definitivo. Bookman, 2012. ROGERS, Y.; SHARP, H.; PREECE, J. **Design de Interação: além da interação homem-computador**. Bookman, 2013. MEYER
- Web Content Accessibility Guidelines (WCAG) 2.1. Disponível em: https://www.w3.org/TR/WCAG21/
- https://htmlreference.io/forms/
- https://developer.mozilla.org/pt-BR/docs/Web/Guide/HTML/Canvas_tutorial/Drawing_shapes
- https://www.w3schools.com/graphics/canvas_coordinates.asp

Bibliografia Complementar

- AMARAL, L. G. **CSS Cascading Style Sheets**: guia de consulta rápida. 2. ed. São Paulo: Novatec, c2006.
- KAWANO, W. **Crie aplicativos Web com HTML, CSS, JavaScript, PHP, PostgreSQL, Bootstrap, AngularJS e Laravel**. Rio de Janeiro: Ciência Moderna, 2016.
- PUREWAL, S. **Aprendendo a Desenvolver Aplicações Web**. Novatec, 2014.
- TONSIG, S. L. **Aplicações na nuvem**: como construir com HTML5, javaScript, CSS, PHP e MYSQL. Rio de Janeiro: Ciência moderna, 2012.
- USABILIDADE.com. Disponível em <http://www.usabilidade.com/>
- TASK-Centered User Interface Design: A Practical Introduction. Disponível em <http://hcibib.org/tcuid/>