EECS 368
Programming Language Paradigms

David O. Johnson

Fall 2022

Reminders

- Assignment 3 due: 11:59 PM, Monday, October 3
- Assignment 4 due: 11:59 PM, Monday, October 17

Any Questions?

In-Class Problem Solution

- 11-(9-19) In-Class Problem Solution.pptx

Any Questions?

Chapter 17 - Drawing on Canvas

- Displaying Graphics
- SVG
- The canvas element
- Lines and surfaces
- Paths
- Curves
- Drawing a pie chart
- Text
- Images
- Transformation
- Storing and clearing transformations
- Choosing a graphics interface

Displaying Graphics (DOM)

- Browsers give us several ways to display graphics.
- The simplest way is to use styles to position and color regular DOM elements.
- This can get you quite far.
- By adding partially transparent background images to the nodes, we can make them look exactly the way we want.
- It is even possible to rotate or skew nodes with the transform style.

- But we'd be using the DOM for something that it wasn't originally designed for.
- Some tasks, such as drawing a line between arbitrary points, are extremely awkward to do with regular HTML elements.

Displaying Graphics (SVG)

There are two alternatives.

- The first is DOM-based but utilizes Scalable Vector Graphics (SVG), rather than HTML.
- Think of SVG as a document markup dialect that focuses on shapes rather than text.
- You can embed an SVG document directly in an HTML document or include it with an <img> tag.

Displaying Graphics (Canvas)

- The second alternative is called a canvas.
- A canvas is a single DOM element that encapsulates a picture.
- It provides a programming interface for drawing shapes onto the space taken up by the node.

SVG vs. Canvas

SVG:

- The original description of the shapes is preserved so that they can be moved or resized at any time.

Canvas:

- Converts the shapes to pixels (colored dots on a raster) as soon as they are drawn and does not remember what these pixels represent.
- The only way to move a shape on a canvas is to clear the canvas (or the part of the canvas around the shape) and redraw it with the shape in a new position.

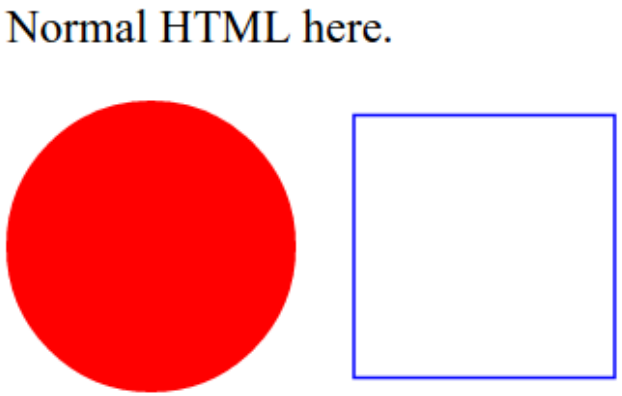
- Both Canvas and SVG could be considered Domain Specific Languages (DSL), recall:
  - A Domain-Specific Language (DSL) is a computer language specialized to a particular application domain.
  - This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains.

Any Questions?

Scalable Vector Graphics (SVG)

This is an HTML document with a simple SVG picture in it:

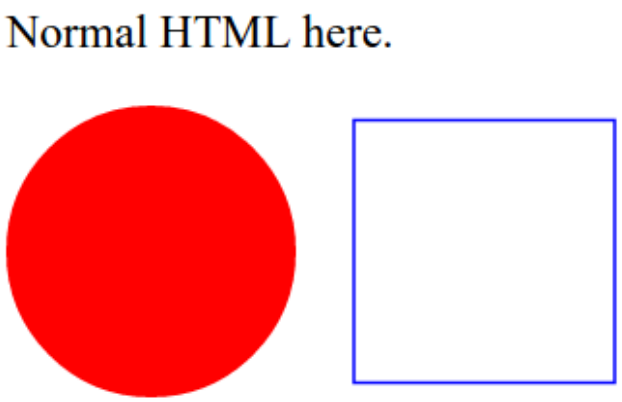
```
<p>Normal HTML here.</p>
<svg xmlns="http://www.w3.org/2000/svg">
  //Draw a red circle
  <circle r="50" cx="50" cy="50" fill="red"/>
  //Draw a blue box
  <rect x="120" y="5" width="90" height="90"
    stroke="blue" fill="none"/>
</svg>
```



- The xmlns attribute identified by a URL, specifies the dialect that we are currently speaking.
- The <circle> and <rect> tags, which do not exist in HTML, do have a meaning in SVG—they draw shapes using the style and position specified by their attributes.

Scalable Vector Graphics (SVG)

```
<p>Normal HTML here.</p>
<svg xmlns="http://www.w3.org/2000/svg">
//Draw a red circle
<circle r="50" cx="50" cy="50" fill="red"/>
//Draw a blue box
<rect x="120" y="5" width="90" height="90"
  stroke="blue" fill="none"/>
</svg>
```



- These tags create DOM elements, just like HTML tags, that scripts can interact with.
- For example, this changes the <circle> element to be colored cyan instead:

```
let circle = document.querySelector("circle");
circle.setAttribute("fill", "cyan");
```

Any Questions?

The Canvas Element

- Canvas graphics can be drawn onto a <canvas> element.
- You can give such an element width and height attributes to determine its size in pixels.
- A new canvas is empty, meaning it is entirely transparent and thus shows up as empty space in the document.
- The <canvas> tag is intended to allow different styles of drawing.
- You can have more than one <canvas> element on the same page.
- Each canvas will show up in the DOM.
- Each canvas maintains its own state.
- If you give each canvas an id attribute, you can access them just like any other element.

Canvas Context

- To get access to an actual drawing interface, we first need to create a context, an object whose methods provide the drawing interface.
- There are currently two widely supported drawing styles:
  - "2d" for two-dimensional graphics
  - "webgl" for three-dimensional graphics through the OpenGL interface.
- We won't discuss WebGL in this course, but if you are interested in three-dimensional graphics, look into WebGL.
- It provides a direct interface to graphics hardware and allows you to render even complicated scenes efficiently, using JavaScript.

Creating a Context for Canvas

- You create a context with the getContext method on the <canvas> DOM element:

```
<p>Before canvas.</p>
<canvas width="120" height="60"></canvas>
<p>After canvas.</p>
<script>
let canvas = document.querySelector("canvas");
let context = canvas.getContext("2d");
context.fillStyle = "red";
context.fillRect(10, 10, 100, 50);
</script>
```

Drawing a Rectangle

```
<p>Before canvas.</p>
<canvas width="120" height="60"></canvas>
<p>After canvas.</p>
<script>
let canvas = document.querySelector("canvas");
let context = canvas.getContext("2d");
context.fillStyle = "red";
context.fillRect(10, 10, 100, 50);
</script>
```

After creating the context object, the example draws a red rectangle 100 pixels wide and 50 pixels high, with its top-left corner at coordinates (10,10).

Just like in HTML (and SVG), the coordinate system that the canvas uses puts (0,0) at the top-left corner, and the positive y-axis goes down from there.

So (10,10) is 10 pixels below and to the right of the top-left corner.

Drawing a Rectangle

```
<p>Before canvas.</p>
<canvas width="120" height="60"></canvas>
<p>After canvas.</p>
<script>
let canvas = document.querySelector("canvas");
let context = canvas.getContext("2d");
context.fillStyle = "red";
context.fillRect(10, 10, 100, 50);
</script>
```

Before canvas.

After canvas.

- The fillRect method draws a filled rectangle.
- It takes first the x- and y-coordinates of the rectangle's top-left corner, then its width, and then its height.

Drawing a Rectangle

```
<p>Before canvas.</p>
<canvas width="120" height="60"></canvas>
<p>After canvas.</p>
<script>
let canvas = document.querySelector("canvas");
let context = canvas.getContext("2d");
context.fillStyle = "red";
context.fillRect(10, 10, 100, 50);
</script>
```

- The color of the fill is not determined by an argument to the method (as you might reasonably expect) but rather by properties of the context object.
- The fillStyle property controls the way shapes are filled.
- It can be set to a string that specifies a color, using the color notation used by CSS.

Cascading Style Sheets (CSS)

- The styling system for HTML is called CSS, for Cascading Style Sheets.
- A style sheet is a set of rules for how to style elements in a document.
- It can be given inside a <style> tag.

```
<style>
strong {
font-style: italic;
color: gray;
}
</style>
```

```
<p>Now <strong>strong text</strong> is italic and gray.</p>
```

- The *cascading* in the name refers to the fact that multiple such rules are combined to produce the final style for an element.
- In the example, the default styling for tags, which gives them font-weight: bold, is overlaid by the rule in the <style> tag, which adds font-style and color.

Cascading Style Sheets (CSS)

CSS style sheets are specified in the header of the HTML document.

```
<!doctype html>
<html>
 <head>
  <meta charset="utf-8">
  <title>My home page</title>
  <style type="text/css">
 </head>
 <body>
  Rest of the webpage.
 </body>
</html>
```

“<style type="text/css">” tells the browser where the CSS file is

- We won’t be using style sheets all that much in this course.
- Understanding them is helpful when programming in the browser, but they are complicated enough to warrant a separate book.

Any Questions?

Lines and Surfaces

- In the canvas interface,
- a shape can be *filled* (fillRect), meaning its area is given a certain color or pattern,
- or it can be *stroked* (strokeRect), which means a line is drawn along its edge.
- The same terminology is used by SVG.

- This code draws two blue squares, using a thicker line for the second one.

- When no width or height attribute is specified, as in the example, a canvas element gets a default width of 300 pixels and height of 150 pixels.

```
<canvas></canvas>
<script>
let cx = document.querySelector("canvas").getContext("2d");
cx.strokeStyle = "blue";
cx.strokeRect(5, 5, 50, 50); //draw left square
cx.lineWidth = 5; //set width of right square
cx.strokeRect(135, 5, 50, 50); //right square
</script>
```

Lines and Surfaces

- The strokeStyle property works similar to the fillStyle property but determines the color used for a stroked line.
- The width of that line is determined by the lineWidth property, which may contain any positive number.

```
<canvas></canvas>
<script>
let cx = document.querySelector("canvas").getContext("2d");
cx.strokeStyle = "blue"; //color used for stroked line
cx.strokeRect(5, 5, 50, 50);
cx.lineWidth = 5; //width used for stroked line
cx.strokeRect(135, 5, 50, 50);
</script>
```

Any Questions?

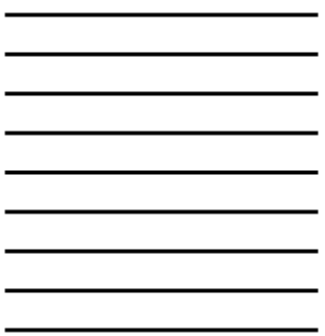
Paths

- A path is a sequence of lines.
- The 2D canvas interface takes a peculiar approach to describing such a path.
- It is done entirely through side effects.
- Paths are not values that can be stored and passed around.
- Instead, if you want to do something with a path, you make a sequence of method calls to describe its shape.

Paths

- This example creates a path with a number of horizontal line segments and then strokes it using the stroke method.
- Each segment created with lineTo starts at the path's *current* position.
- That position is usually the end of the last segment, unless moveTo was called.
- In that case, the next segment would start at the position passed to moveTo.

```
<canvas></canvas>
<script>
let cx = document.querySelector("canvas").getContext("2d");
cx.beginPath(); //start path
for (let y = 10; y < 100; y += 10) { //draw 9 lines 10 pixels apart
cx.moveTo(10, y); //move to here
cx.lineTo(90, y); //draw a line to here
}
cx.stroke(); //stroke path
</script>
```



Filled Paths

- When filling a path (using the fill method), each shape is filled separately.
- A path can contain multiple shapes.
- Each moveTo motion starts a new one.
- But the path needs to be closed (meaning its start and end are in the same position) before it can be filled.
- If the path is not already closed, a line is added from its end to its start, and the shape enclosed by the completed path is filled.

Filled Paths

- This example draws a filled triangle.
- Note that only two of the triangle's sides are explicitly drawn.
- The third, from the bottom-right corner back to the top, is implied and wouldn't be there when you stroke the path.

```
<canvas></canvas>
<script>
let cx = document.querySelector("canvas").getContext("2d");
cx.beginPath();
cx.moveTo(50, 10);
cx.lineTo(10, 70);
cx.lineTo(90, 70);
cx.fill();
</script>
```

- You could also use the closePath method to explicitly close a path by adding an actual line segment back to the path's start.
- This segment *is* drawn when stroking the path.

Any Questions?

In-Class Problem

Write an HTML document that draws the trapezoid (a rectangle that is wider on one side) below in Canvas.