CS 5410

Intro to HTML5 Canvas Rendering

References

- W3 Schools
  - Overview: https://www.w3schools.com/html/html5_canvas.asp
  - API: https://www.w3schools.com/tags/ref_canvas.asp
- Others
  - Dive into HTML5: http://diveintohtml5.info/canvas.html
  - Mozilla: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D
  - Mozilla: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial

What is it?

- An HTML5 element that is a container for graphics rendering
  - 2D – native Canvas API
  - 3D – OpenGL ES
- Features
  - Lines
  - Shapes
  - Paths (including curves)
  - Text/Fonts
  - Images & Pixel Manipulation
  - Transformations (rotation, scaling)

Defining the Canvas Element

- `<canvas id = "id-canvas" width = "500" height = "500"></canvas>`
  - Size in browser is 500 x 500 pixels
  - Coordinate system is 500 x 500
- HTML width/height is the rendering coordinate system
- CSS width/height is the visible size of the canvas element

Canvas Element Examples

- ```
  <canvas id = "id-canvas" width = "500" height = "500"
      style = "width: 750px; height 750px;">
  </canvas>
  ```
  - Size in browser is 750 x 750 pixels
  - Coordinate system is 500 x 500
- ```
  <canvas id = "id-canvas"  width = "500" height = "500"
      style = "width: 75%; height 75%;">
  </canvas>
  ```
  - Size in browser is 75% of width and 75% of height
  - Coordinate system is 500 x 500

Coordinate System

- Upper Left: (0, 0)
- Lower Right: (width – 1, height – 1)
- Increasing Y moves down
- Increasing X moves right

- Notes
  - This is not typically what you are used to in a Cartesian coordinate system where lower left is (0, 0)
  - In context of browser, however, it makes sense to put (0, 0) in the upper left

JavaScript – HTML Connection

- Obtaining the Canvas Object
  - `let canvas = document.getElementById('id-canvas');`
  - `let context = canvas.getContext('2d');`
- `canvas` is a reference the HTML element
- `context` is a reference to the canvas API; this is what you want/need
  - `CanvasRenderingContext2D`

Clearing The Canvas

- Once something is drawn to the canvas, it persists until cleared.
- Clearing the whole canvas
  - `context.clearRect(0, 0, canvas.width, canvas.height);`
- Adding it to the prototype
  - ```
    CanvasRenderingContext2D.prototype.clear = function() {
        this.clearRect(0, 0, canvas.width, canvas.height);
    }
    ```
  - Use it as: `context.clear();`
    ```

Drawing a Rectangle

1. Set rendering style(s)
   - Color
   - Outline width
   - Shadow
   - Line join style
   - others...
2. (optional) Fill
3. (optional) Draw outline

Drawing a Rectangle - Code

```
context.strokeStyle = 'rgba(0, 0, 255, 1)';
context.lineWidth = 2;
context.strokeRect(
    canvas.width / 4, canvas.height / 4,
    canvas.width / 2, canvas.height / 2);
```

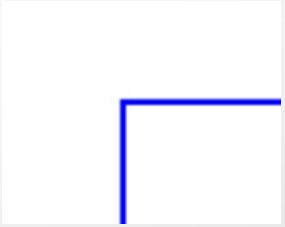
```
context.strokeStyle = 'rgba(0, 0, 255, 1)';
context.lineWidth = 2;
context.shadowColor = 'rgba(0, 255, 0, 1)';
context.shadowBlur = 10;
context.strokeRect(
    canvas.width / 4, canvas.height / 4,
    canvas.width / 2, canvas.height / 2);
```

...and now for something completely different...

Aliasing

```
context.lineWidth = 1;
context.strokeRect(
    canvas.width / 4, canvas.height / 4,
    canvas.width / 2, canvas.height / 2);
```

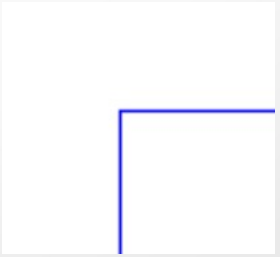
```
context.lineWidth = 2;
context.strokeRect(
    canvas.width / 4, canvas.height / 4,
    canvas.width / 2, canvas.height / 2);
```



Aliasing

```
context.lineWidth = 1;
context.strokeRect(
    canvas.width / 4, canvas.height / 4,
    canvas.width / 2, canvas.height / 2);
```

```
context.lineWidth = 1;
context.strokeRect(
    canvas.width / 4 + 0.5, canvas.height / 4 + 0.5,
    canvas.width / 2, canvas.height / 2);
```



...back to our regularly scheduled program...

Drawing a Polygon

1. Set rendering style(s)
   - Color
   - Outline width
   - Shadow
   - Line join style
   - others...
2. moveTo starting position
3. lineTo remaining vertices
4. (optional) stroke
5. (optional) fill

Drawing a Polygon – Code

```
context.beginPath();

context.moveTo(canvas.width / 2, canvas.height / 4);
context.lineTo(
    canvas.width / 2 + canvas.width / 4,
    canvas.height / 2 + canvas.height / 4);
context.lineTo(
    canvas.width / 2 - canvas.width / 4,
    canvas.height / 2 + canvas.height / 4);

context.closePath();

context.fillStyle = 'rgba(0, 0, 255, 1)';
context.fill();

context.lineWidth = 1;
context.strokeStyle = 'rgba(255, 0, 0, 1)';
context.stroke();
```

Rotating Shapes (or anything)

- For each object, maintain its center
- In general terms, three operations to rotate
  1. Translation based on object (to be rotated) center
  2. Rotation
  3. Negative translation based on object center
- Important Note: It is the canvas itself that is being translated and rotated!
  - This is completely different from APIs like OpenGL, Vulkan, DirectX

Rotation

```
context.translate(shape.center.x, shape.center.y);
context.rotate(rotation);
context.translate(-shape.center.x, -(shape.center.y));
```

...render shape here...

```
context.translate(shape.center.x, shape.center.y);
context.rotate(-rotation);
context.translate(-shape.center.x, -(shape.center.y));
```

Rotation – Alternative (and faster)

```
context.save();

context.translate(shape.center.x, shape.center.y);
context.rotate(rotation);
context.translate(-shape.center.x, -(shape.center.y));
```

...render shape here...

```
context.restore();
```