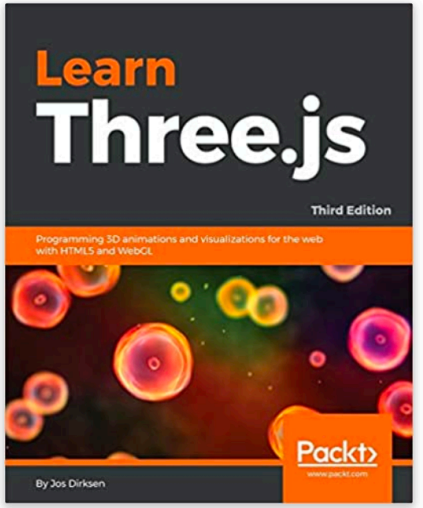
COMPUTER GRAPHICS



* Adapted from CISC 3326 lecture by Michael Mandel

INTRODUCTION TO CANVAS DRAWING

Based on CS 307 lecture 2a

Topics for today

- Covered last time
  - The HTML5 <canvas> element
  - Drawing rectangles
  - Drawing paths
- Drawing arcs in paths
- Setting properties of path segments
- Transformations and saving and restoring state

Topics for today

- Exercises
  - draw a tree
  - draw a house
  - draw a village

Homework

- Read Canvas tutorials:
  - MDN Canvas Tutorial
  - HTML Canvas Graphics

The HTML5 <canvas> element

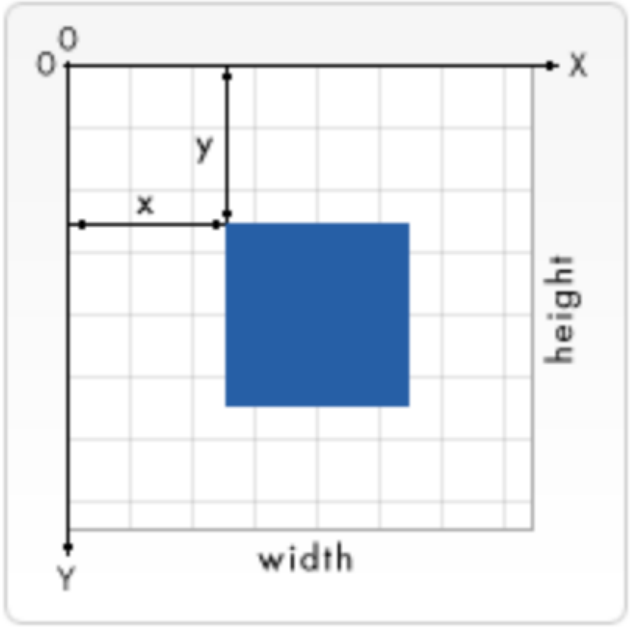
- This section is based on the MDN Canvas Tutorial
- Provides a canvas on which JavaScript can draw
  - <canvas id=”canvas" width=”300" height=”300"></canvas>
- We will use the 2D drawing context for now
  - Also used by WebGL and Three.js for 3D
    - behind the scenes

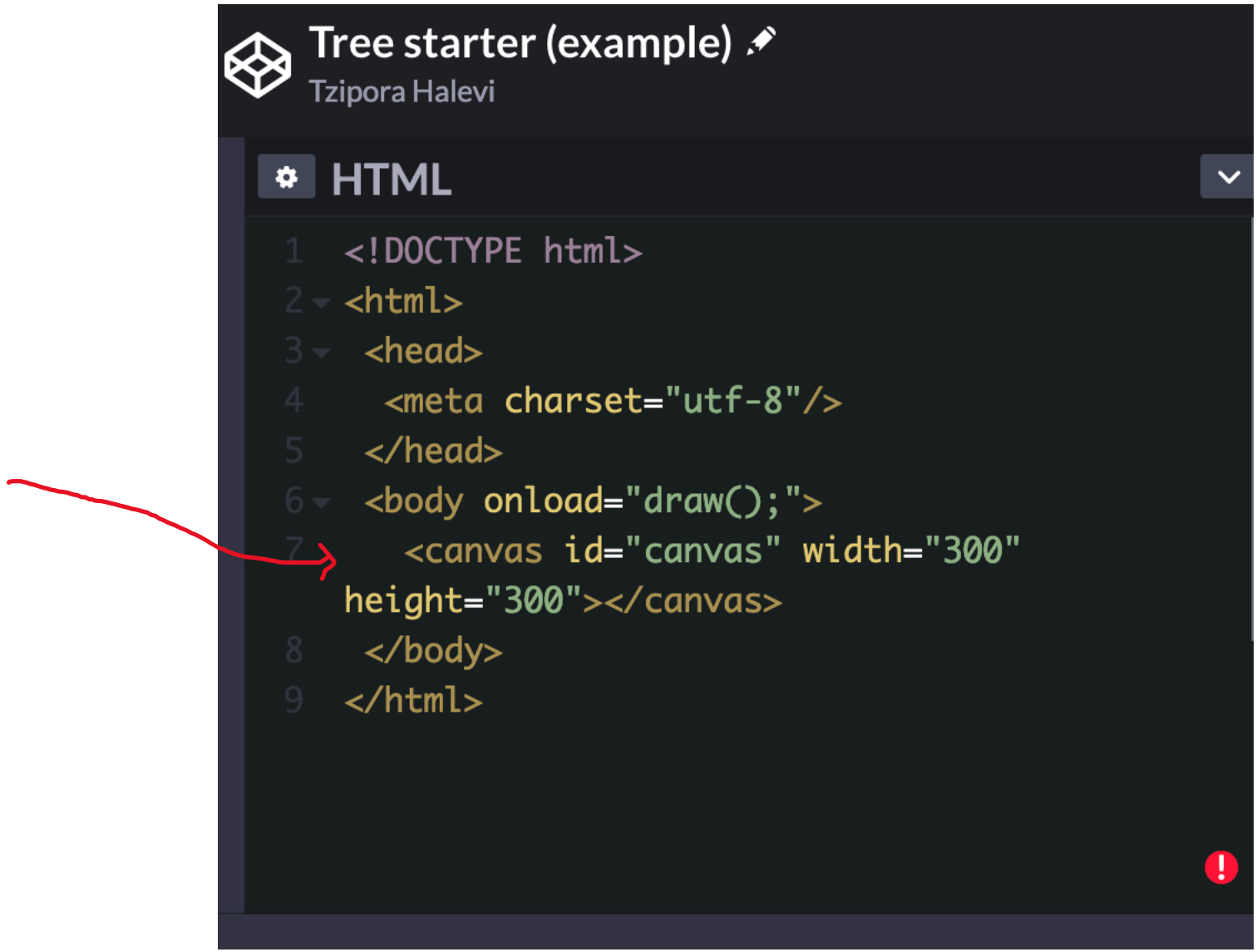
A simple example (codepen)

- <!DOCTYPE html>
- <html>
  - <head>
    - <meta charset="utf-8"/>
      - <script type="application/javascript">
      - function draw() {
        - var canvas = document.getElementById('canvas’);
        - if (canvas.getContext) {
          - var ctx = canvas.getContext('2d’);
          - ctx.fillStyle = 'rgb(200, 0, 0)’;
          - ctx.fillRect(10, 10, 50, 50);
          - ctx.fillStyle = 'rgba(0, 0, 200, 0.5)’;
          - ctx.fillRect(30, 30, 50, 50);
        - }
      - } </script> </head> <body onload="draw();"> <canvas id="canvas" width="150" height="150"></canvas> </body>
- </html>

Canvas grid

- All elements are placed relative to this origin
  - => Position of the top left corner of the blue square = x pixels from the left and y pixels from the top
    - at coordinate (x, y)




Tree starter (example)
Tzipora Halevi
HTML
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
</head>
<body onload="draw();">
<canvas id="canvas" width="300"
height="300"></canvas>
</body>
</html>

The HTML5 <canvas> element

- To access the 2D context from JavaScript:
  - var canvas = document.getElementById(‘canvas’);
  - var ctx = canvas.getContext('2d');


JS
var canvas =
document.getElementById('canvas');
if (canvas.getContext) {
var ctx = canvas.getContext('2d');
// flip the coordinate frame
ctx.save();
flipY(ctx);
drawTreeAt(ctx,200,50,90,20,30);
ctx.restore();
}
}

Example - canvas

- Tree Starter

Drawing rectangles

- Canvas only supports two primitive shapes: rectangles and paths
- To draw rectangles:
  - fillRect(x, y, width, height): Draws a filled rectangle
  - strokeRect(x, y, width, height): Draws a rectangular outline
  - clearRect(x, y, width, height): Clears the specified rectangular area, making it fully transparent
- (x,y) in this case specifies the top left corner

Example: strokeRect

- Example: strokeRect
- strokeRect example

Drawing paths

- A path is a list of points, connected by segments of lines
- Segments can be different shapes, curved, straight, different colors

Drawing paths

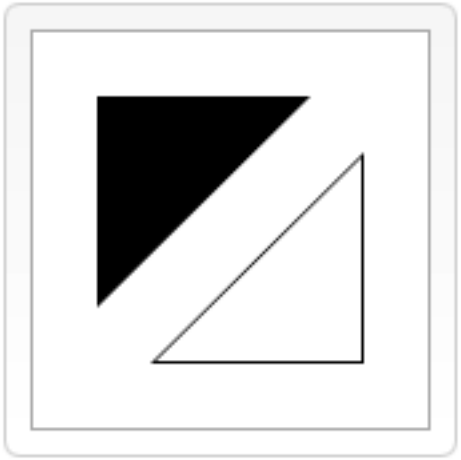
- To make shapes using paths
  - Create the path using beginPath()
  - Use moveto(x,y) to go to the start point
  - Add segments to the path
    - Use lineto(x,y) to continue to each point
  - Call closePath();
    - closePath is optional when using fill
  - Draw it using either fill() or stroke()
    - fill() = solid shape
    - stroke() = just outline

Drawing Paths

- Draw a triangle:
  - Triangle - outline only
  - Triangle Canvas Example

Exercise: Drawing two triangles

- Start from the Two-triangle starter
- Try to draw this picture using the above path functions



Answer: Drawing two triangles

```
Function draw() {
var canvas = document.getElementById('canvas’);
if (canvas.getContext) {
    var ctx = canvas.getContext('2d'); // Filled triangle
    ctx.beginPath();
    ctx.moveTo(25, 25);
    ctx.lineTo(105, 25);
    ctx.lineTo(25, 105);
    ctx.fill(); // Stroked triangle
    ctx.beginPath();
    ctx.moveTo(125, 125);
    ctx.lineTo(125, 45);
    ctx.lineTo(45, 125);
    ctx.closePath();
    ctx.stroke();
    }
}
```

Drawing arcs in paths

- There are two path functions to add arcs (portions of circles)
  - arc(x, y, radius, startAngle, endAngle, anticlockwise) draws an arc
    - centered at (x, y)
    - with radius r
    - starting at startAngle (in radians with 0 to the right)
    - ending at endAngle
    - going in the given direction indicated by anticlockwise (defaulting to clockwise)

Drawing arcs in paths

- arcTo(x1, y1, x2, y2, radius) draws an arc
  - starting at current point
  - going to (x1, y1) and then (x2, y2)
  - for a circle with radius radius
- Angles in radians can be computed with
- $Radians = (\frac{\pi}{180}) * degrees$
-

Example: **complete the smiley face**

- Start from this codepen
- Complete the smiley face to match this picture
  - don’t worry about the orange lines:

Questions?

Coloring shapes

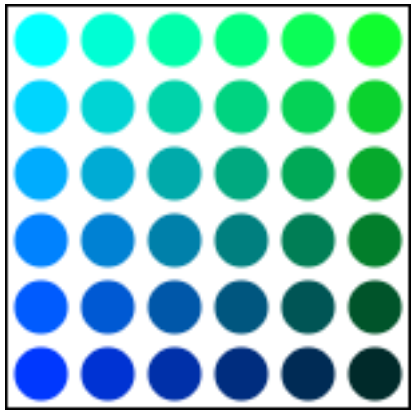
- Setting the context's fillStyle property affects all future shapes
  - until the property is set again
  - Same for strokeStyle
- Can use any CSS color specification
  - ctx.fillStyle = 'orange';
  - ctx.fillStyle = '#FFA500';
  - ctx.fillStyle = 'rgb(255, 165, 0)';
  - ctx.fillStyle = 'rgba(255, 165, 0, 1)';

Coloring shapes - RGBA

- ctx.fillStyle = 'rgba(255, 165, 0, 1)’;
- Fourth parameter is alphe
  - Defines the opacity as a number between 0.0 (fully transparent) and 1.0 (fully opaque)

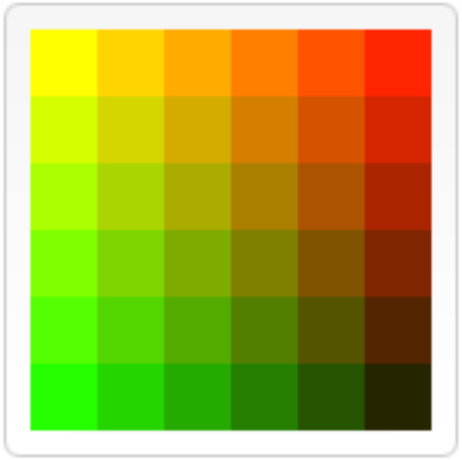
Example: coloring circles

- Function draw() {
- var ctx = document.getElementById('canvas').getContext('2d’);
- for (var i = 0; i < 6; i++) {
  - for (var j = 0; j < 6; j++) {
    - ctx.fillStyle = 'rgb(0, ' + Math.floor(255 - 42.5 * i) + ', ‘ + Math.floor(255 - 42.5 * j) + ')’;
    - ctx.beginPath();
    - ctx.arc(12.5 + j * 25, 12.5 + i * 25, 10, 0, Math.PI * 2, true);
    - ctx.fill();
    - }
  - }
- }



Example: rectangle grid

- Start from this codepen
- Try to match this picture



Questions?

Answer:

```
function draw() {
var ctx = document.getElementById('canvas').getContext('2d’);
for (var i = 0; i < 6; i++) {
  for (var j = 0; j < 6; j++) {
    ctx.fillStyle = 'rgb(' + Math.floor(255 - 42.5 * i) + ', ' +
    Math.floor(255 - 42.5 * j) + ‘,’+ 0+’)’;
    ctx.fillRect(j * 25, i * 25, 25, 25);
    }
  }
}
```

Example: rectangle grid

- Filled rectangle

- More info can be found here

Saving and restoring state

- There are two canvas methods that can make your life much easier
  - save() save the entire state of the canvas to a stack
  - restore() restore the state from the top of the stack
- The state that is saved is ‘transforms’
  - strokeStyle, fillStyle, etc.
- If you use them in pairs, your code will be much more modular

Example: nested squares

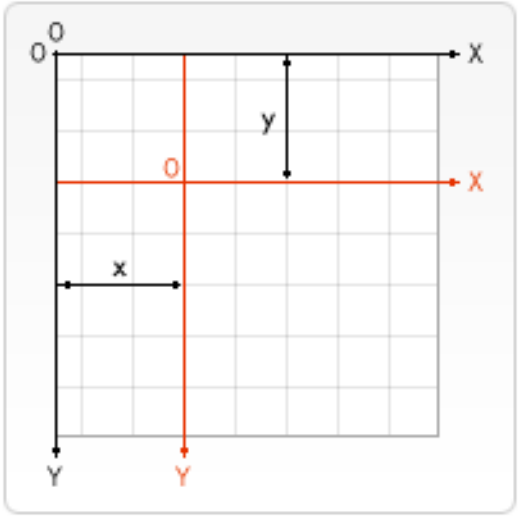
- Function draw() {
  - var ctx = document.getElementById('canvas').getContext('2d');
    ctx.fillRect(0, 0, 150, 150);
  - ctx.**save**();
  - ctx.fillStyle = '#09F';
    ctx.fillRect(15, 15, 120, 120);
  - ctx.**save**();
    ctx.fillStyle = '#FFF';
    ctx.globalAlpha = 0.5;
  - ctx.fillRect(30, 30, 90, 90);
  - ctx.**restore**();
    ctx.fillRect(45, 45, 60, 60);
  - ctx.**restore**();
    ctx.fillRect(60, 60, 30, 30);
- }

Example: nested squares

- Example

Transformations: Translating

- Canvas transformations move the canvas and origin around
- Translate moves the canvas and its origin to a different point
  - translate(x,y)



- save() the canvas state before callin
  - restore() after

Example: translation

```
function draw() {
  var ctx = document.getElementById('canvas').getContext('2d');
  for (var i = 0; i < 3; i++) {
    for (var j = 0; j < 3; j++) {
      ctx.save();
      ctx.fillStyle = 'rgb(' + (51 * i) + ', ' + (255 - 51 * i) + ', 255)';
      ctx.translate(10 + j * 50, 10 + i * 50);
      ctx.fillRect(0, 0, 25, 25);
      ctx.restore();
    }
  }
}
```

Transformations: Translating

- Translation

Other transformations

- Other transformations that exist are rotation and scaling:
  - rotate(angle)
  - scale(xScale, yScale)

Questions?

Summary

- Drawing in a 2D coordinate system isn't so bad.
- Achieve effects using methods on a *context* object.
- Code using functions to achieve higher-level effects
- Parameterize the functions
  - e.g. a function to draw a tree with a given width and height.

Summary

- Make the functions be *generic* , e.g. a house with its origin at the lower left.
- Use transformations to translate the generic objects.

Questions?