CSC 470 – Section 3

Topics in Computer Science: Advanced Browser Technologies

Mark F. Russo, Ph.D.

Spring 2016

Lecture 9

Eloquent JavaScript: Chapter 16

Canvas

- A single DOM element that encapsulates a drawing surface
- Provides a programming interface for <u>painting</u> graphics
- Originally developed by Apple
- Graphics are rendered by setting pixels – graphics elements are <u>not</u> preserved as distinct objects, unlike SVG
- Has very few drawing commands
  - line, path, arc, image, rectangle, text, …
- Any change to a graphic requires redrawing all affected pixels
- No objects, no individual graphic element event handling
- Can be much faster than SVG, depending upon application

Drawing Model

1. "Trace" out where the drawing should occur
   - *think of this as drawing an outline lightly with a pencil*
2. Set <u>stroke parameters</u> (color, width, …) and <u>stroke</u> the shape (optional)
   - *think of this as tracing over the pencil with a marker*
3. Set <u>fill parameters</u> (color, …) and <u>fill</u> the shape (optional)
   - *think of this as coloring in the outline with a marker*

- Rectangles and Text have shorthand methods

Canvas Element

- `<canvas>` is a standard DOM element
- Attributes
  - `width` - the width of the canvas DOM element
  - `height` - the height of the canvas DOM element
- May have other standard attributes, e.g. `id`
- Initially renders as a transparent element

```html
<canvas id="cvs" width="600" height="600"></canvas>
```

Canvas Rendering Contexts

Drawing is not accomplished with the `<canvas>` element…

- To draw, one must first get a **drawing context** object
- All drawing occurs using drawing context methods

Currently, two drawing contexts are available:

1. `CanvasRenderingContext2D`
   - Provides the 2D rendering context for the <canvas> element.

```
    var ctx = canvas.getContext("2d");
```

2. `WebGLRenderingContext`
   - Provides the OpenGL ES 2.0 rendering context for the <canvas> element.

```
    var ctx = canvas.getContext("webgl");
```

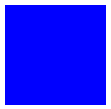
   - 3D and hardware accelerated
   - Experiments
     - http://www.martin-laxenaire.fr/experiments/8000ers/
     - http://madebyevan.com/webgl-water/
     - http://akirodic.com/p/jellyfish/

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D
https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext

Using A Rendering Context

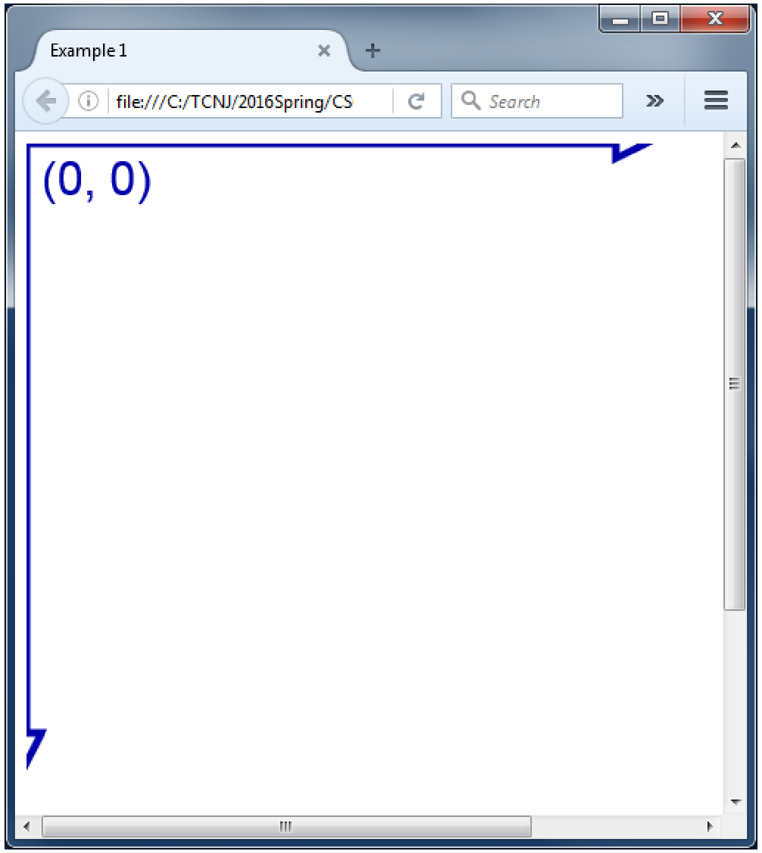
1. Get a reference to the `<canvas>` DOM element
2. Get an appropriate Rendering Context object
3. Issue drawing commands

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Example 1</title>
  </head>
  <body>
    <canvas id="cvs" width="600" height="600"></canvas>
    <script type="text/javascript">
      var cvs = document.getElementById("cvs");
      var ctx = cvs.getContext("2d");

      // Rectangle shorthand
      ctx.fillStyle = "#0000FF";
      ctx.fillRect(10, 10, 50, 50);
    </script>
  </body>
</html>
```



Coordinate System

`(0, 0)` is in upper left corner of `<canvas>`



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

ctx.beginPath();

ctx.moveTo(0, 0);
ctx.lineTo(0, 400);      // y-axis
ctx.lineTo(-10, 400);    // y-axis arrow
ctx.lineTo(0, 420);
ctx.lineTo(10, 400);
ctx.lineTo(0, 400);

ctx.moveTo(0, 0);
ctx.lineTo(400, 0);      // x-axis
ctx.lineTo(400, -10);    // x-axis arrow
ctx.lineTo(420, 0);
ctx.lineTo(400, 10);
ctx.lineTo(400, 0);

ctx.lineWidth = 5;
ctx.strokeStyle = "#0000AA";
ctx.stroke();

ctx.fillStyle = "#0000AA";
ctx.font = "24pt arial";
ctx.fillText("(0, 0)", 10, 35);
```

Drawing Paths

Nearly all shapes are drawn as paths

1. Begin the path
2. Move to the starting location
3. Add line and curve segments
4. Close the path (optional)
5. Set stroke style and line width (optional)
6. Stroke the path (optional)
7. Set fill style (optional)
8. Fill the path (optional)

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

ctx.beginPath();

ctx.moveTo(0, 0);
ctx.lineTo(0, 400);      // y-axis
ctx.lineTo(-10, 400);    // y-axis arrow
ctx.lineTo(0, 420);
ctx.lineTo(10, 400);
ctx.lineTo(0, 400);

ctx.moveTo(0, 0);
ctx.lineTo(400, 0);      // x-axis
ctx.lineTo(400, -10);    // x-axis arrow
ctx.lineTo(420, 0);
ctx.lineTo(400, 10);
ctx.lineTo(400, 0);

ctx.lineWidth = 5;
ctx.strokeStyle = "#0000AA";
ctx.stroke();

ctx.fillStyle = "#0000AA";
ctx.font = "24pt arial";
ctx.fillText("(0, 0)", 10, 35);
```

09/coords.html

Building Paths

`context.moveTo(x, y);`

- Move to (x, y) without tracing the path

`context.lineTo(x, y);`

- Add a straight line from the current location to (x, y)

`context.quadraticCurveTo(cx, cy, x, y);`

- Add a quadratic curve to a path from current location to (x, y)

`context.bezierCurveTo(cx1, cy1, cx2, cy2, x, y);`

- Add a bezier curve to a path from the current location to (x, y)

`context.arc(x, y, radius, startAngle, endAngle);`

- Add an arc to a path with center (x, y), radius, from startAngle to endAngle

`context.rect(x, y, width, height);`

- Add a rectangle to a path with upper left corner (x, y), width, height

`context.closePath();`

- Add a line segment to the path between the current location and the start location

Styling

Fill Styles

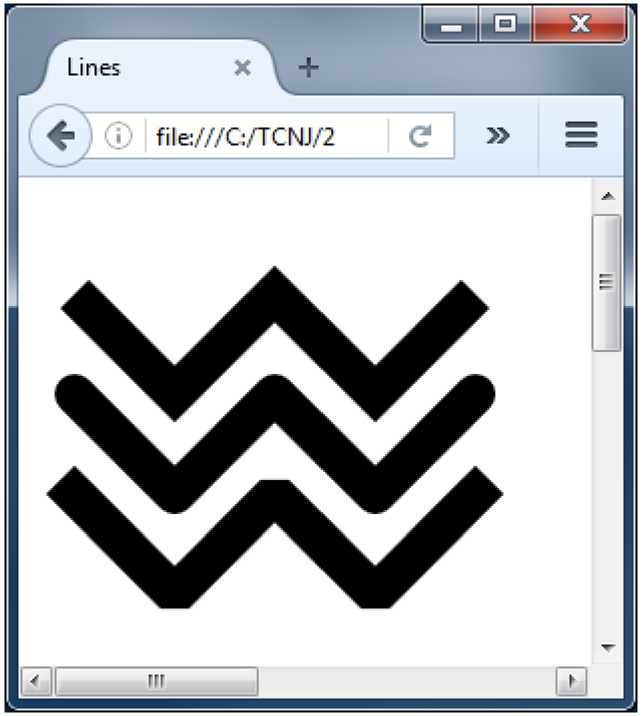
```
// Set fill color using any CSS color string
context.fillStyle = '#8ED6FF';
```

Stroke Styles

```
// Set line style using any CSS color string
context.strokeStyle = 'blue';

// Specify width in pixels
context.lineWidth = 5;

// Both line join and end cap styles may be specified
context.lineCap  = 'round'; // butt|round|square;
context.lineJoin = 'miter'; // miter|round|bevel;
```

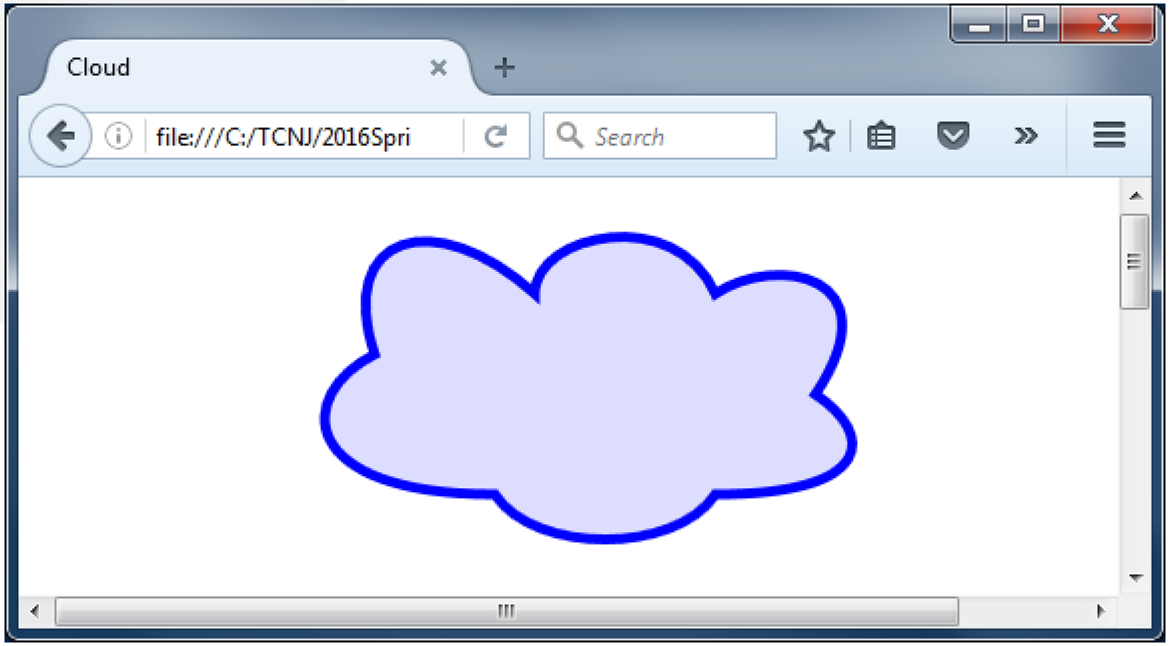
Path Examples

```
(function() {
  var zigzag = function(x, y) {
    ctx.beginPath();
    ctx.moveTo(x, y);
    ctx.lineTo(x+50, y+50);
    ctx.lineTo(x+100, y+0);
    ctx.lineTo(x+150, y+50);
    ctx.lineTo(x+200, y);
  };

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

  zigzag(20, 50);
  ctx.lineWidth = 20;
  ctx.lineCap   = 'butt';
  ctx.lineJoin  = 'miter';
  ctx.stroke();

  zigzag(20, 100);
  ctx.lineWidth = 20;
  ctx.lineCap   = 'round';
  ctx.lineJoin  = 'round';
  ctx.stroke();

  zigzag(20, 150);
  ctx.lineWidth = 20;
  ctx.lineCap   = 'square';
  ctx.lineJoin  = 'bevel';
  ctx.stroke();
})();
```



Path Examples

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

// Begin cloud
ctx.beginPath();
ctx.moveTo(170, 80);
ctx.bezierCurveTo(130, 100, 130, 150, 230, 150);
ctx.bezierCurveTo(250, 180, 320, 180, 340, 150);
ctx.bezierCurveTo(420, 150, 420, 120, 390, 100);
ctx.bezierCurveTo(430, 40, 370, 30, 340, 50);
ctx.bezierCurveTo(320, 5, 250, 20, 250, 50);
ctx.bezierCurveTo(200, 5, 150, 20, 170, 80);

// Complete cloud
ctx.closePath();
ctx.fillStyle = "#DDDDFF"
ctx.fill();
ctx.lineWidth = 5;
ctx.strokeStyle = 'blue';
ctx.stroke();
```



09/cloud.html

http://www.html5canvastutorials.com/advanced/html5-clear-canvas/

Rectangles and Circles

- Shorthand for drawing rectangles

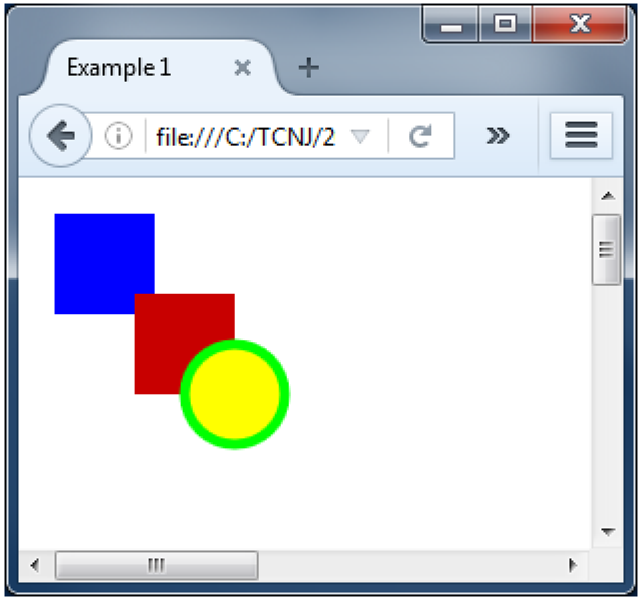
```
ctx.fillRect(x, y, width, height);
ctx.strokeRect(x, y, width, height);
```

- Draw circles using an arc path from angles 0 to 2π radians

```
ctx.beginPath();
ctx.arc(x, y, radius, 0, 2*Math.PI);
```

Path Examples

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

// Rectangle shorthand
ctx.fillStyle = "#0000FF";
ctx.fillRect(10, 10, 50, 50);

// Rectangle path
ctx.beginPath();
ctx.rect(50, 50, 50, 50);
ctx.fillStyle = "rgb(200,0,0)";
ctx.fill();

// Circle made with an arc path
ctx.beginPath();
ctx.arc(100, 100, 25, 0, 2*Math.PI);
ctx.fillStyle = "yellow";
ctx.fill();
ctx.strokeStyle = "#00FF00";
ctx.lineWidth = 5;
ctx.stroke();
```



What about an ellipse?

Clearing the Canvas

To clear the entire canvas or just a part, invoke the `clearRect()` method

```
// Clear the entire canvas using canvas element attributes
context.clearRect(0, 0, canvas.width, canvas.height);
```

Drawing Text

- Text drawing commands trace characters to a rendering context
- Text stroke and fill styles may be set like any other path

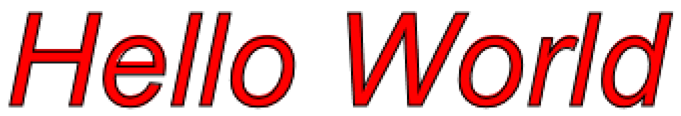
```
ctx.fillStyle = "red";
ctx.strokeStyle = "black";
ctx.lineWidth = 1;
```

- Text fonts may be set by assigning the context `font` attribute to a standard CSS font string

```
ctx.font = "48pt sans-serif";
```

- Once traced, text may be stroked and filled with special commands

```
ctx.fillText("Hello World", 100, 100);
ctx.strokeText("Hello World", 100, 100);
```


Hello World

09/ex1.html

Drawing Text

- Text may be aligned with respect to its specified x-value

```
ctx.textAlign = "left" || "right" || "center" || "start" || "end";
```

- Text may be drawn with respect to its baseline

```
ctx.textBaseline = "top" || "hanging" || "middle" || "alphabetic" ||
"ideographic" || "bottom";
```

- Rendered size of text may be measured. Returns a `TextMetrics` object, which specified width, bounding box, and other parameters

```
var metrix = ctx.measureText(text);
```

Transformations

- Shapes may be transformed with Canvas similar to the way they are transformed with SVG
- With no distinct shape objects, we have no way to save transformations with shapes
- Only the canvas coordinate system may be transformed
- Shape transformations must be applied before each draw and reset when complete, if appropriate
- Multiple transformations may be applied prior to drawing
- SVG's "rotation about a point" option is not available

Rendering Context Coordinate Transforms

- Translate

```
  context.translate( dx, dy );
```

- Scale

```
  context.scale( sx, sy );
```

- Rotate

```
  context.rotate( radians );
```

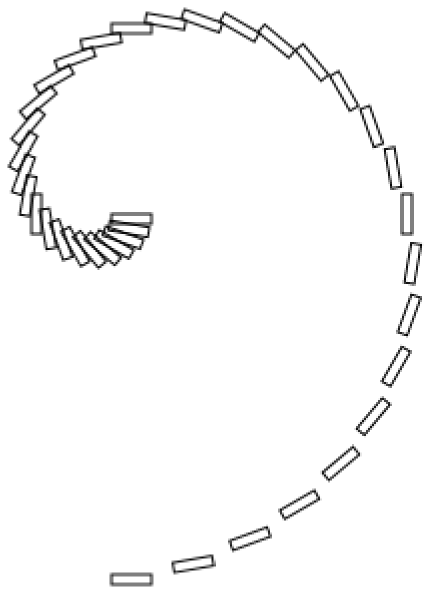
- Reset all applied transforms

```
  context.setTransform(1, 0, 0, 1, 0, 0);
```

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/translate
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/scale
https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/rotate

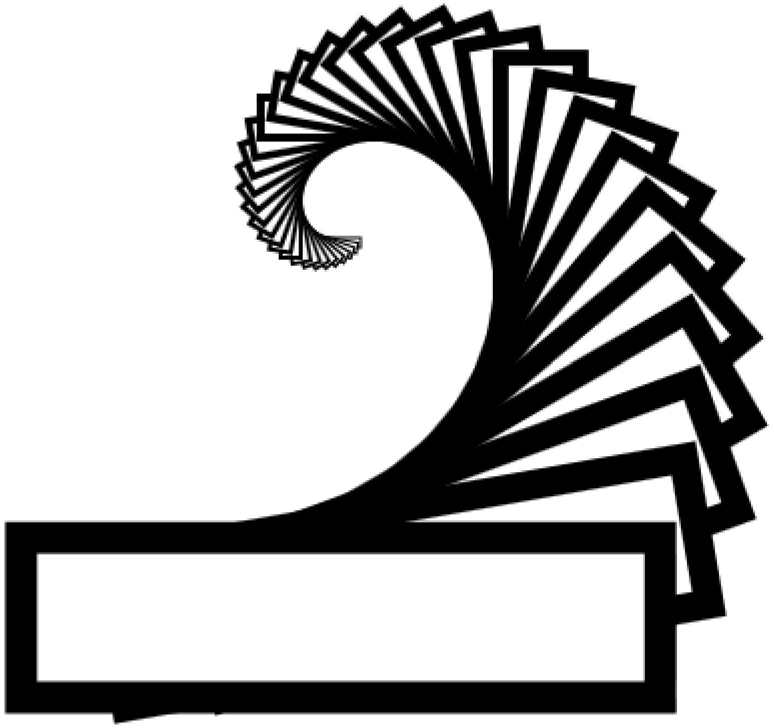
Transformations Example 1

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

// Initially translate origin
ctx.translate(300, 200);

// Angle and radius increment for each
iteration
var dangle = 2*Math.PI/36;
var dradius = 5;
ctx.fillStyle = "white";

// Draw rectangle after repeated rotations
for (var i=0; i<=36; i++) {
  ctx.beginPath();
  ctx.rect(-10, i*dradius, 20, 5);
  ctx.stroke();
  ctx.rotate(dangle);
}
```



Transformations Example 2

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

// Initially translate origin
ctx.translate(300, 200);

// Angle and radius increment for each
iteration
var dangle = 2*Math.PI/36;
var dradius = 5;
ctx.fillStyle = "white";

// Draw rectangle after repeated rotations
for (var i=0; i<=36; i++) {
  ctx.beginPath();
  ctx.rect(-10, 10, 20, 5);
  ctx.fill();
  ctx.stroke();
  ctx.rotate(dangle);
  ctx.scale(1.08, 1.08);
}
```



…with added scale factor

Drawing State

- Drawing state may be modified temporarily while drawing using an internal drawing state stack
- Process
  1. Push entire state to internal state stack (using `ctx.save()`)
  2. Modify state as necessary
  3. Draw using new state
  4. Pop/reapply saved state off stack (using `ctx.restore()`)
- Drawing state saved includes:
  - The current transformation matrix.
  - The current values of: strokeStyle, fillStyle, lineWidth, lineCap, lineJoin, …
  - …more
- Being a stack, `ctx.save()` may be called multiple times in a row to push state onto stack. Calling `ctx.restore()` resets state in a FIFO manner.
- Very useful when needing to temporarily modify transformation and quickly restore when done
- Particularly important due to there existing only the single global transformation

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/save

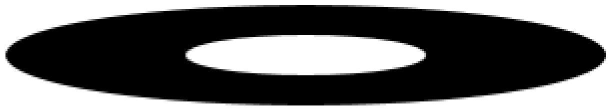
Drawing an Ellipse

- There is no built-in ellipse drawing method
- Fortunately, we can draw an ellipse by applying a scaling transformation in one dimension and then drawing a circle (an arc from 0 to 2π)
- We can benefit from `ctx.save()`/`ctx.restore()` when drawing an ellipse to avoid impacting other transformations currently in effect.

Drawing an Ellipse

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

// Draw ellipse as a scalled circle
var cx = 200;
var cy = 100;
var rx = 60;
var ry = 10;
ctx.translate(cx, cy);
ctx.scale(rx, ry);
ctx.beginPath();
ctx.arc(0, 0, 1, 0, 2*Math.PI, false);
ctx.lineWidth = 3;
ctx.fillStyle = "white";
ctx.stroke();
ctx.fill();
```

Outline is distorted!



Drawing an Ellipse

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

// Draw ellipse as a scalled circle
var cx = 200;
var cy = 100;
var rx = 60;
var ry = 10;
ctx.save();
ctx.translate(cx, cy);
ctx.scale(rx, ry);
ctx.beginPath();
ctx.arc(0, 0, 1, 0, 2*Math.PI, false);
ctx.restore();
ctx.lineWidth = 3;
ctx.fillStyle = "white";
ctx.stroke();
ctx.fill();
```

To fix outline,

1. save current drawing state
2. trace path in transformed coordinates
3. restore original state
4. render after restoring original drawing state

09/ellipse2.html

Handling Events

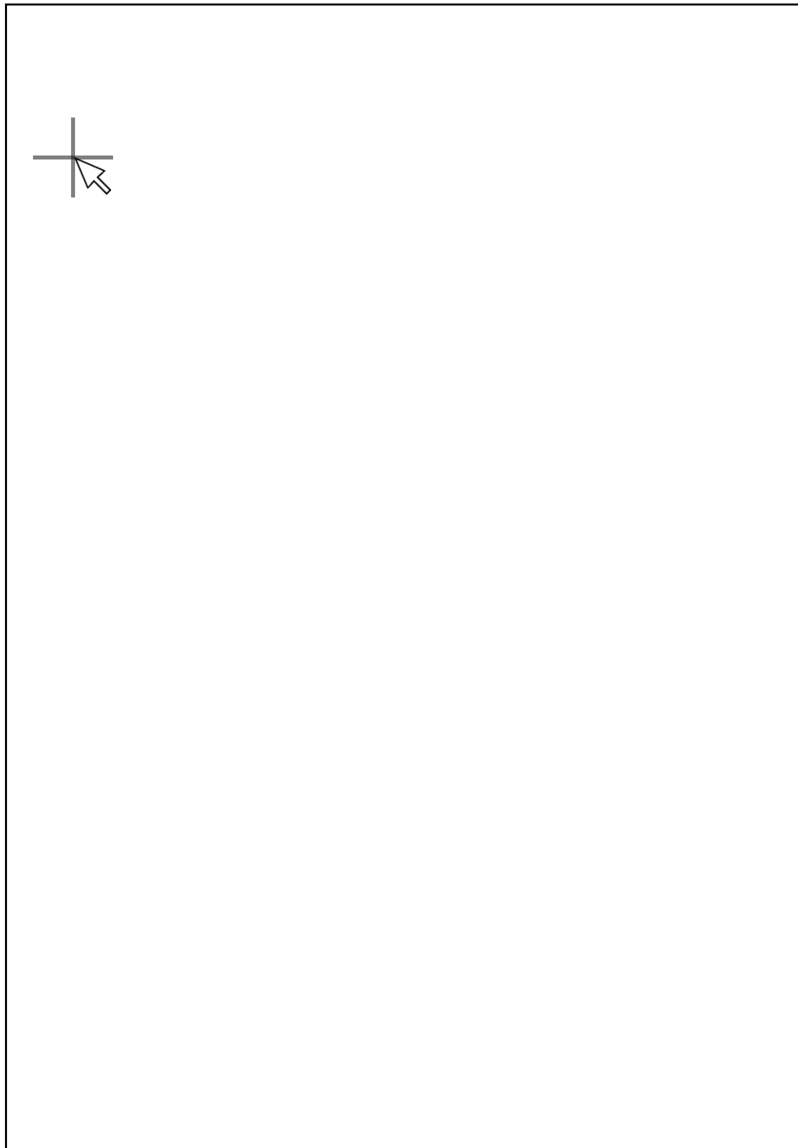
- A rendering context does not dispatch events
- Events dispatched by the `<canvas>` element may be used
- To get coordinates with respect to `<canvas>` element and not the entire page, use the Element method…

```
rectObject = canvas.getBoundingClientRect();
```

http://www.html5canvastutorials.com/advanced/html5-canvas-mouse-coordinates/
https://developer.mozilla.org/en-US/docs/Web/API/Element/getBoundingClientRect

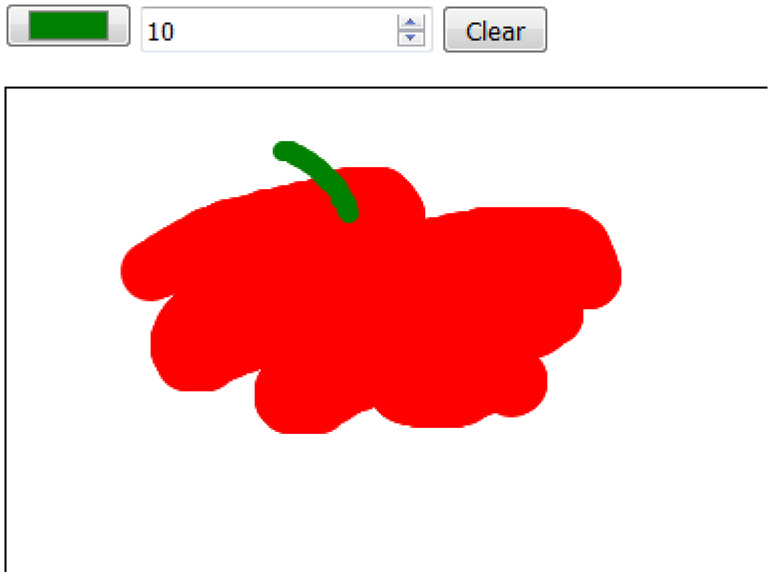
Handling Events

```
// Get mouse position as an object
function getMousePos(cvs, evt) {
  var rect = cvs.getBoundingClientRect();
  return {
    x: evt.clientX - rect.left,
    y: evt.clientY - rect.top
  };
}

var cvs = document.getElementById('cvs');
var ctx = cvs.getContext('2d');

// Draw an X at the mouse position
cvs.addEventListener('mousemove', function(evt)
{
  var pos = getMousePos(cvs, evt);
  ctx.clearRect(0, 0, cvs.width, cvs.height);
  ctx.beginPath();
  ctx.moveTo(pos.x-20, pos.y);
  ctx.lineTo(pos.x+20, pos.y);
  ctx.moveTo(pos.x, pos.y-20);
  ctx.lineTo(pos.x, pos.y+20);
  ctx.stroke();
  console.log(pos);
});
```



Drawing Program

```
// <snip>>

var cvs = document.getElementById('cvs');
var ctx = cvs.getContext('2d');

// Get mouse position as an object
function getMousePos(cvs, evt) {
  var rect = cvs.getBoundingClientRect();
  return {
    x: evt.clientX - rect.left,
    y: evt.clientY - rect.top
  };
}

// Track last mouse position
var lastPos = null;

// Handle events
cvs.addEventListener('mousedown', function(evt) {
  lastPos = getMousePos(cvs, evt);
});

cvs.addEventListener('mouseup', function(evt) {
  lastPos = null;
});

// Draw a line if last pos is not null
cvs.addEventListener('mousemove', function(evt) {
  if (lastPos === null) return;
  var pos = getMousePos(cvs, evt);
  ctx.beginPath();
  ctx.moveTo(lastPos.x, lastPos.y);
  ctx.lineTo(pos.x, pos.y);
  ctx.stroke();
  lastPos = pos;
});
```



Components
- Color input
- Spinner
- Button
- Canvas

```
<p>
  <input  id="clr" type="color" />
  <input  id="wid" type="number" min="1" value="1"/>
  <button id="cle">Clear</button>
</p>
<canvas id="cvs" width="600" height="600"></canvas>
```

09/scribble.html

Image Object

- Image - a built-in JavaScript object
- May correspond to an `<img src="…">` tag (not necessary)
- Image object constructor may take optional width and height parameters
- An Image may be created and loaded from script

```
// Create an Image object
var img = new Image();

// Load the image
img.src = "bug.png";
```

Image Object

- The Image Object has numerous properties and events
- Image load is performed asynchronously – it is often important to wait until after an image has been loaded to complete a script
- The load event is triggered when the image has completed loading

```
// Create an Image object
var img = new Image();

// Set up the load event handler
img.onload = function(e) {
  console.log("bug.png loaded");
};

// Setting the src property triggers load
// when the image load is complete
img.src = "bug.png";
```

Drawing Images

- Once loaded, an image may be drawn to a canvas
- 2D Drawing Context has three image draw methods

Draw the Image img at (x, y)

```
context.drawImage( img, x, y );
```

Draw the Image img at (x, y) with width and height

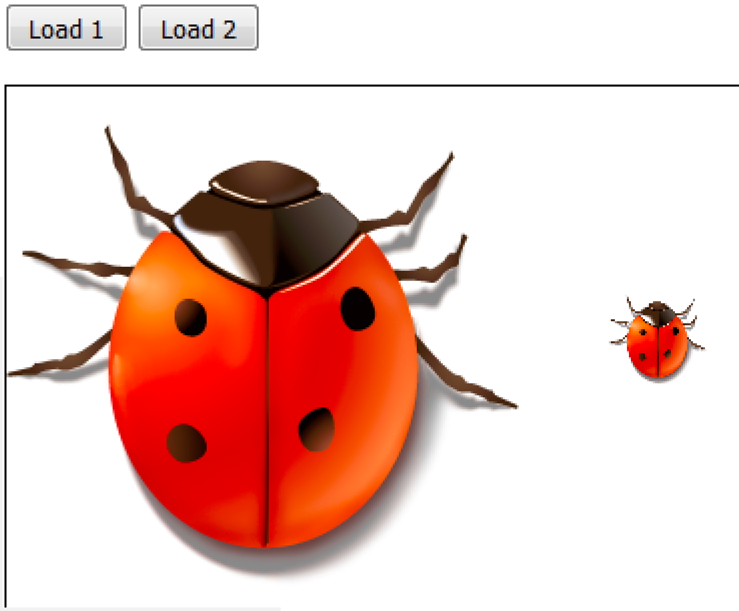
```
context.drawImage( img, x, y, width, height );
```

Transfer part of the Image img to the canvas

- *source at (sx, sy) with width sw, height sh*
- *destination at (dx, dy) with width dw, height dh*

```
context.drawImage( img, sx, sy, sw, sh, dx, dy, dw, dh );
```

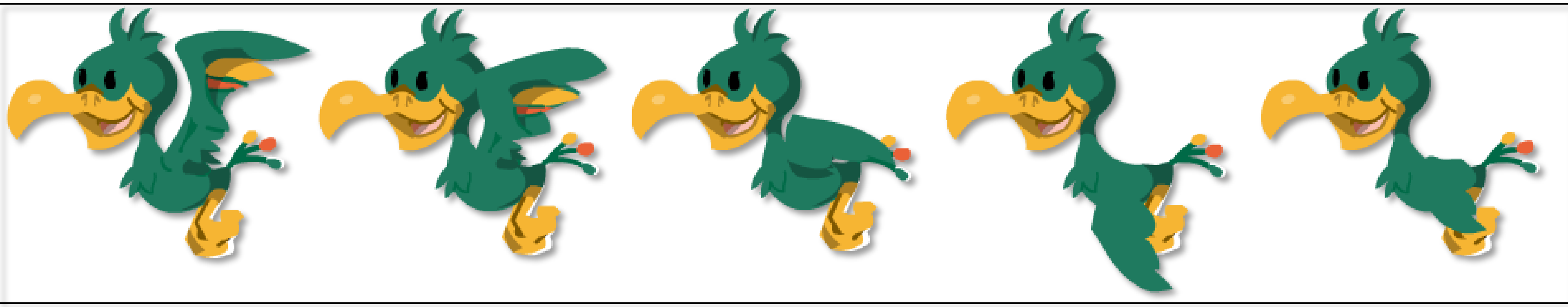
Drawing Images



```
// Create and load an image
var img = new Image();
img.onload = function(e) {
  console.log("bug.png loaded");
};
img.src = "bug.png";

// Get canvas and context
var cvs = document.getElementById("cvs");
var ctx = cvs.getContext("2d");

// Handle button clicks
document.getElementById("btn1").onclick = function(e) {
  ctx.drawImage( img, 0, 0 );
};

document.getElementById("btn2").onclick = function(e) {
  ctx.drawImage( img, 300, 100, 50, 50 );
};
```

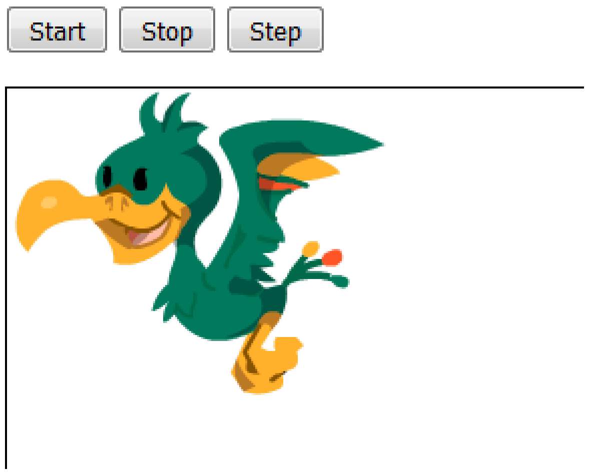
Sprites

- A sprite sheet is a single large image containing multiple frames from a short animation
- Use a timer and the canvas function to copy parts of an image to a canvas to render the animation



```
var cvs = document.getElementById('cvs');
var ctx = cvs.getContext('2d');

// Draw frame helper function
var drawFrame = function( num ) {
  var x = num*190;
  ctx.clearRect(0, 0, cvs.width, cvs.height);
  ctx.drawImage(img, x, 0, 190, 180, 0, 0, 190, 180);
};

// Current frame count
var frame = 0;

// Step forward and draw frame
var step = function(e) {
  frame = (frame+1) % 5;
  drawFrame(frame);
};

// Load sprite sheet image
// 190 x 180
var img = new Image();
img.onload = function() {
  // After loaded for first time draw first frame
  drawFrame(0);
};
img.src = "bird.png";

// Handle button events
var timerId = null;
document.getElementById("start").onclick = function(ev) {
  timerId = setInterval( step, 100 );
};
document.getElementById("stop").onclick = function(ev) {
  if (timerId) clearInterval(timerId);
};
document.getElementById("step").onclick = function(ev) {
  step();
};
```

Sprites



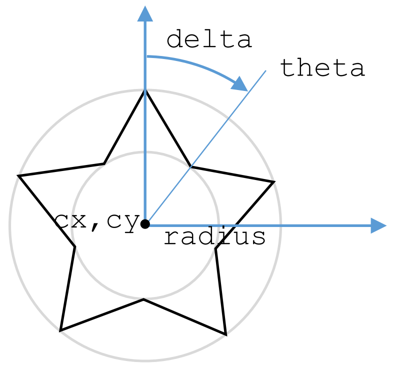
Examples

Examples canvas-based applications…

• CanvasXpress	(http://canvasxpress.org/)
• AcqMan	(http://doodlepad.org/apps/acqman/)
• LPG	(http://doodlepad.org/apps/lpg4/)

Star-drawing Algorithm

1. Input a center point `(cx, cy)`, number of points (`npoints`), and enclosing `radius`
2. Compute an angle increment (`delta`) as `360/(2*npoints)`
3. Initialize an angle variable (`theta`) as -90° and a radius variable (`r`) as `radius`
4. Begin a path and move to a starting point of `(r, theta)`
5. Loop for `i` starting at `0` and continuing while `i < 2*npoints`
6. For each iteration
   - increase `theta` by `delta`
   - if `i` is even, set `r` to `0.5*radius`. Otherwise, set `r` to `radius`.
   - compute `(x, y)` given `(r, theta)`. Make sure to convert `theta` to radians.
   - add a line to the path from the current location to `(x, y)`
7. After the loop, close the path
8. Fill and stroke the generated path


delta
theta
cx,cy
radius