Lab # 01

Computer Graphics

Shoaib Ahmed Siddiqui
Reg. # 00468
BSCS-2B

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

CS361: Computer Graphics

Class: BSCS-2AB & BESE-3AB

Lab01: Introduction to HTML5 Canvas

Date: 7th September, 2015

Time: 2:00pm- 5:00pm

Instructor: Dr. Muhammad Muddassir Malik

Lab 1: Introduction to HTML 5 Canvas

Introduction

With the ongoing progress of HTML5 and its powerful specifications, web developers have gained the ability to do things that were impossible in the past. Drawing graphics and creating animations directly in the browser is now completely possible thanks to a technology called HTML 5 Canvas.

Objectives

After performing this lab students should be able to:

Know the basics of HTML 5 Canvas

Tools/Software Requirement

For testing HTML 5, CSS, JS
http://codepen.io/pen

Description

What is HTML5 Canvas?

The canvas element is an element defined in HTML code using width and height attributes. The real power of the canvas element, however, is accomplished by taking advantage of the HTML5 Canvas API. This API is used by writing JavaScript that can access the canvas area through a full set of drawing functions, thus allowing for dynamically generated graphics.

What's so Great About HTML5 Canvas?

Here are some reasons you might want to consider learning to use HTML5's canvas feature:

Interactivity: Canvas is fully interactive. It can respond to a user's actions by listening for keyboard, mouse, or touch events. So a developer is not restricted to only static graphics and images.

Animation: Every object drawn on the canvas can be animated. This allows for all levels of animations to be created, from simple bouncing balls to complex forward and inverse kinematics.

Flexibility. Developers can create just about anything using canvas. It can display lines, shapes, text, images, etc. — with or without animation. Plus, adding video and/or audio to a canvas application is also possible.

Browser/Platform Support. HTML5 Canvas is supported by all major browsers and can be accessed on a wide range of devices including desktops, tablets, and smart phones.

Popularity. Canvas popularity is rapidly and steadily growing so there is no shortage of resources available.

It's a web standard. Unlike Flash and Silverlight, Canvas is open technology that's part of HTML5. And although not all of its features are implemented in all browsers, developers can be certain canvas will be around indefinitely.

Develop once, run everywhere. HTML5 Canvas offers great portability. Once created, Unlike Flash and Silverlight, a canvas application can run almost anywhere — from the largest computers to the smallest mobile devices.

Free and accessible development tools. The basic tools for creating HTML5 canvas applications are just a browser and a good code editor. Plus, there are many great and free libraries and tools to help developers with advanced canvas development.
What Applications Can You Create with HTML5 Canvas?

OK, canvas is great. But what exactly can use it for? Let's see.

Giamng. The HTML5 Canvas' feature set is an ideal candidate for producing all sorts of 2D and 3D games.

Advertising. HTML5 Canvas is a great replacement for Flash-based banners and ads. It has all the needed features for attracting buyers' attention.

Data Representation. HTML5 can collect data from global data sources and use it to generate visually appealing and interactive graphs and charts with the canvas element.

Education and Training. HTML5 canvas can be used to produce effective and attractive learning experiences, combining text, images, videos, and audio.

Art and Decoration. With a little bit of imagination and canvas's wide variety of colors, gradients, patterns, transparency, shadows, and clipping features, all kinds of artistic and decorative graphics can be drawn.

Now that we know why we should learn canvas, let's look at the basics of HTML5 Canvas and how to start using it.

Canvas Rendering Contexts

Every HTML5 canvas element must have a *context*. The context defines what HTML5 Canvas API you'll be using. The 2d context is used for drawing 2D graphics and manipulating bitmap images. The 3d context is used for 3D graphics creation and manipulation. The latter is actually called WebGL and it's based on OpenGL ES.

Getting Started

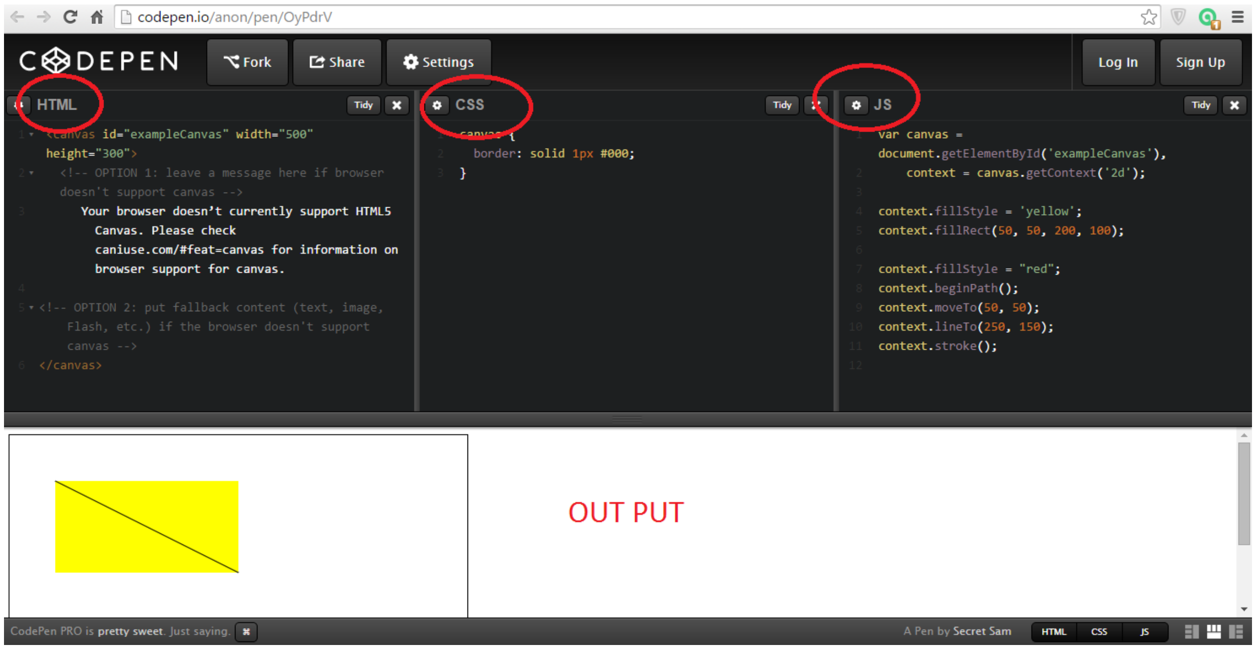
To get started with canvas, all you need is a code editor and a browser with HTML5 canvas support. I use http://codepen.io/anon/pen/OyPdrV , but you can use whatever you want.

Our HTML to start will look like this:

```
<canvas id="exampleCanvas" width="500" height="300">

  <!-- OPTION 1: leave a message here if browser doesn't support 
canvas -->
    <p>Your browser doesn't currently support HTML5 Canvas.
    Please check caniuse.com/#feat=canvas for information on
    browser support for canvas.</p>

  <!-- OPTION 2: put fallback content (text, image, Flash, etc.)
                 if the browser doesn't support canvas -->

</canvas>
```

And here is our JavaScript, which we can include at the bottom of our HTML page:

```
var canvas = document.getElementById('exampleCanvas'),
    context = canvas.getContext('2d');
```



HTML

```html
<canvas id="exampleCanvas" width="500"
height="300">
    <!-- OPTION 1: leave a message here if browser
    doesn't support canvas -->
        Your browser doesn't currently support HTML5
          Canvas. Please check
          caniuse.com/#feat=canvas for information on
          browser support for canvas.

<!-- OPTION 2: put fallback content (text, image,
     Flash, etc.) if the browser doesn't support
     canvas -->
</canvas>
```

CSS

```css
canvas {
    border: solid 1px #000;
}
```

JS

```javascript
var canvas =
document.getElementById('exampleCanvas'),
    context = canvas.getContext('2d');

context.fillStyle = 'yellow';
context.fillRect(50, 50, 200, 100);

context.fillStyle = "red";
context.beginPath();
context.moveTo(50, 50);
context.lineTo(250, 150);
context.stroke();
```

By default, the browser creates canvas elements with a width of 300 pixels and a height of 150 pixels, even if these aren't specified in the HTML. You can change the size by specifying the width andheight,

Notice also that we've given the canvas an id attribute that we'll use in our JavaScript. And finally, if you want, you can use CSS to add a border to make the canvas visible by means of a thin frame. This is not required, it's used as a visual aid for us to see the boundary of the canvas element.

Notice that between the opening and closing <canvas> tags, I've added content that will be displayed if the browser doesn't support canvas. This can be just about any type of content that an older browser supports.

The JavaScript starts with two lines. In the first line we create a variable that caches our canvas element via its ID. The second line creates a variable (context) that references the 2D context for the canvas using the getContext() function. We'll use the context variable to access all canvas drawing functions and properties.

With our canvas ready to go we can start experimenting with the Canvas API. But before that, let's clarify a few more aspects of the canvas feature.

HTML5 Canvas Element Size vs. Drawing Surface Size

Besides the canvas element's width and height attributes, you can also use CSS to set the size of a canvas element. However, sizing a canvas element with CSS is not the same as setting the element'swidth and height attributes. This is because a canvas actually has two sizes: the size of the element itself and the size of the element's drawing surface.

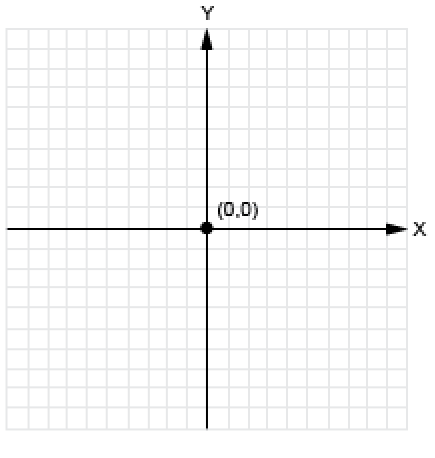
When you set the element's width and height attributes, you set both the element's size and the size of the element's drawing surface; however, when you use CSS to size a canvas element, you set only the element's size and not the drawing surface. When the canvas element's size does not match the size of its drawing surface, the browser scales the drawing surface to fit the element.

Because of this, it's a good idea to use the canvas element's width and height attributes to size the element, instead of using CSS.

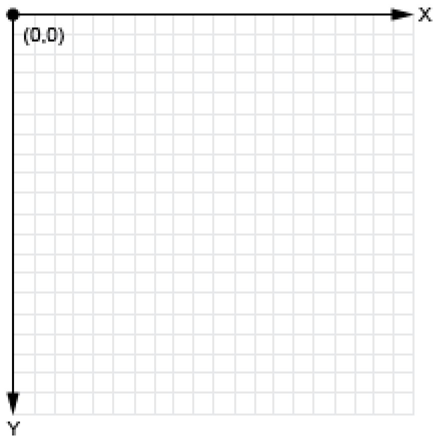
Understanding the Canvas Coordinate System

In a 2D space, positions are referenced using X and Y coordinates. The X axis extends horizontally, and the Y axis extends vertically. The center has a position x = 0 and y = 0, that can also be expressed as(0, 0). This method of positioning objects, used in mathematics, is known as the Cartesian coordinate system.

The Canvas coordinate system, however, places the origin at the upper-left corner of the canvas, with X coordinates increasing to the right and Y coordinates increasing toward the bottom of the canvas. So unlike a standard Cartesian coordinate space, the Canvas space doesn't have visible negative points. Using negative coordinates won't cause your application to fail, but objects positioned using negative coordinate points won't appear on the page.



1. Cartesian coordinate space



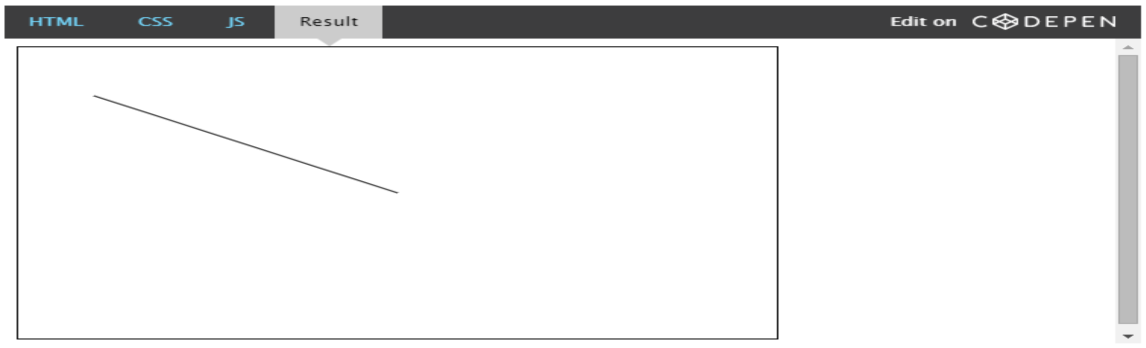
2. Canvas coordinate space

Drawing Lines

To draw a line, we use four canvas API methods. We start with the beginPath() method which instructs the browser to prepare to draw a new path. Next, we use the moveTo(x, y) method to set the line's starting point. Then lineTo(x, y) method sets the ending point and draws the line by connecting the two points. At this point the line will be drawn, but it's still invisible. To make it visible we use the stroke() method to display the line with the default black color.

```
context.beginPath();
context.moveTo(50, 50);
context.lineTo(250, 150);
context.stroke();
```

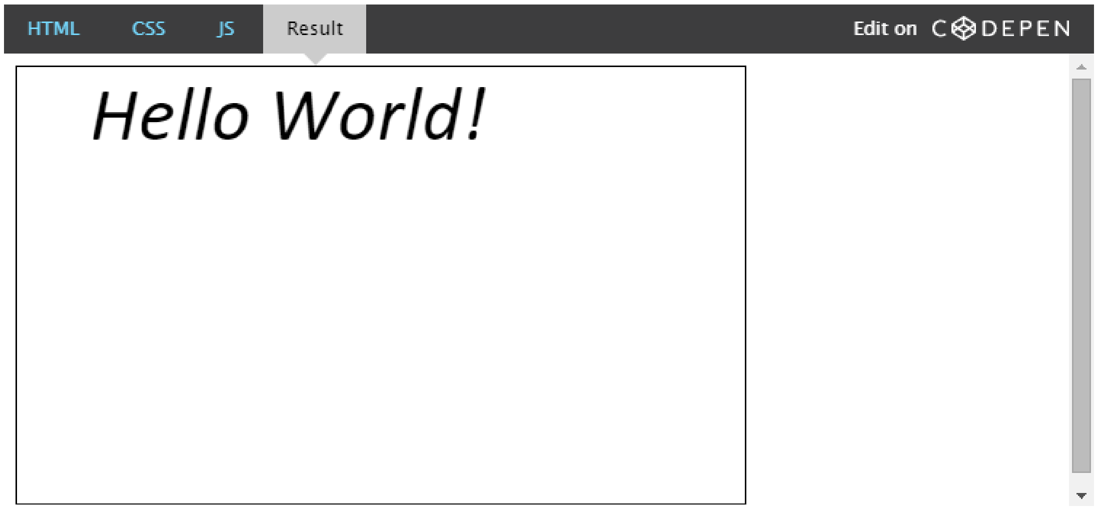
Example: http://codepen.io/anon/pen/OyPdrV



Draw Text

To draw text on the canvas, we can use fillText(string, x, y) along with the font property to set the font, size, and style of the text (similar to font shorthand in CSS).

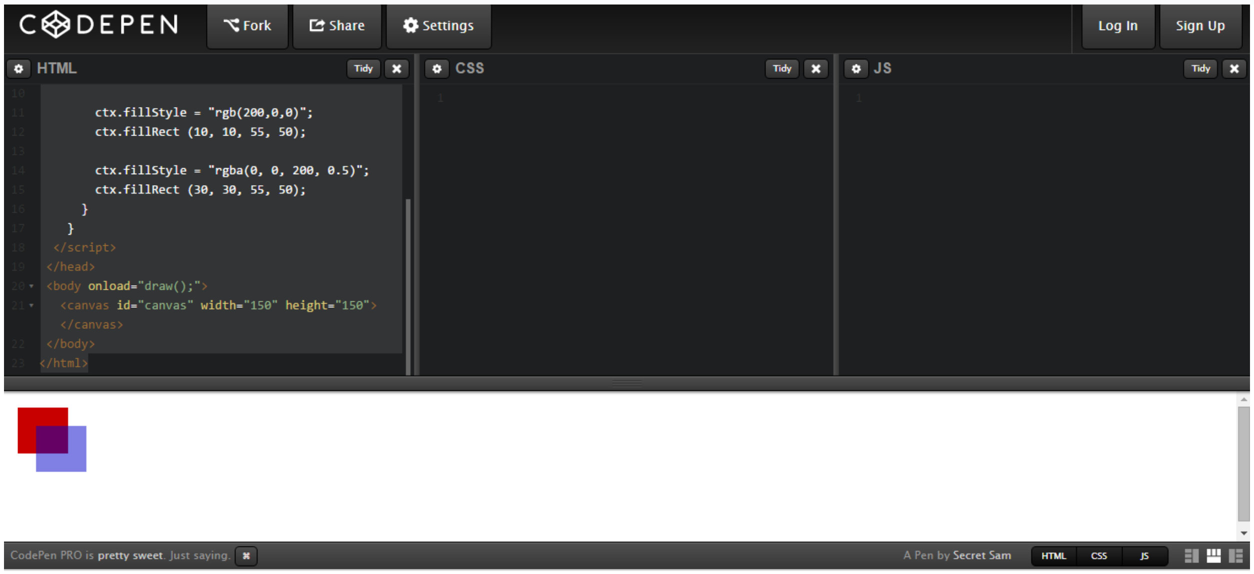
```
context.font = 'italic 40pt Calibri, sans-serif';
context.fillText('Hello World!', 50, 50);
```



Example

Take a look at a simple example that draws two intersecting rectangles, one of which has alpha transparency.

```
<!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, 55, 50);

        ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
        ctx.fillRect (30, 30, 55, 50);
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```



```
        ctx.fillStyle = "rgb(200,0,0)";
        ctx.fillRect (10, 10, 55, 50);

        ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
        ctx.fillRect (30, 30, 55, 50);
      }
    }
  </script>
</head>
<body onload="draw();">
  <canvas id="canvas" width="150" height="150">
  </canvas>
</body>
</html>
```

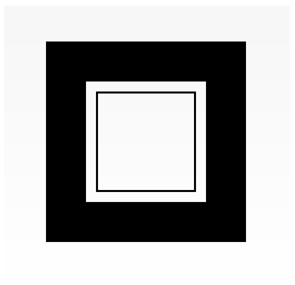
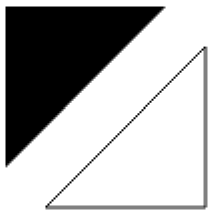
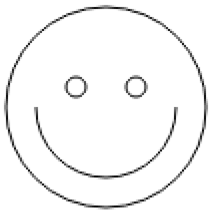
Drawing a triangle

For example, the code for drawing a triangle would look something like this:

```
function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext){
    var ctx = canvas.getContext('2d');

    var path=new Path2D();
    path.moveTo(75,50);
    path.lineTo(100,75);
    path.lineTo(100,25);
    ctx.fill(path);
  }
}
```

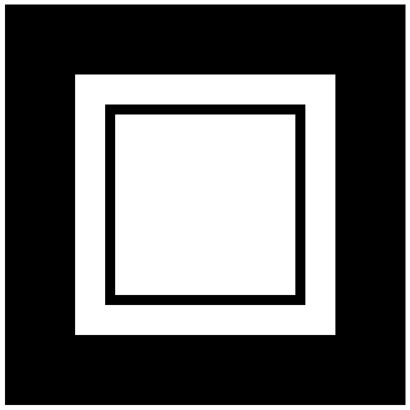
Lab Task

Draw the following shapes using canvas

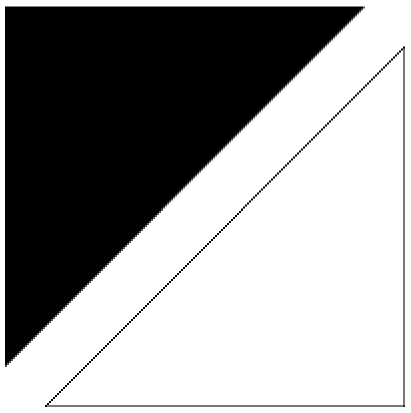
Task # 01:

```
<!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(0,0,0)";
        ctx.fillRect (0, 0, 200, 200);

        ctx.fillStyle = "rgb(255, 255, 255)";
        ctx.fillRect (35, 35, 130, 130);

        ctx.fillStyle = "rgb(0,0,0)";
        ctx.fillRect (50, 50, 100, 100);

        ctx.fillStyle = "rgb(255, 255, 255)";
        ctx.fillRect (55, 55, 90, 90);
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="200" height="200"></canvas>
 </body>
</html>
```



Task # 02:

```
<!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");

        //Top Triangle
        ctx.beginPath();
        ctx.moveTo(180, 0);
        ctx.lineTo(0, 180);
        ctx.lineTo(0, 0);
        ctx.closePath();

        //Fill it with Black color
        ctx.fillStyle = "#000000";
        ctx.fill();

        //Lower Triangle
        ctx.beginPath();
        ctx.moveTo(200, 20);
        ctx.lineTo(200, 200);
        ctx.lineTo(20, 200);
        ctx.closePath();

        ctx.stroke();
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="200" height="200"></canvas>
 </body>
</html>
```



Task # 03:

```
<!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");

        //Draw Face
        ctx.beginPath();
        ctx.arc(100,100,100,0,2*Math.PI);
        ctx.stroke();

        //Draw Eyes
        ctx.beginPath();
        ctx.arc(70,70,10,0,2*Math.PI);
        ctx.stroke();

        ctx.beginPath();
        ctx.arc(130,70,10,0,2*Math.PI);
        ctx.stroke();

        //Draw Mouth
        ctx.beginPath();
        ctx.arc(100,100,70,0,Math.PI);
        ctx.stroke();
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="200" height="200"></canvas>
 </body>
</html>
```