COMS3006A: Computer Graphics and Visualisation

Lab 0: Javascript Canvas

1 HTML

This lab is to help you get acquainted with the HTML canvas and how to draw on it. We will be using this canvas for all of the labs in 2D graphics before we move to 3D graphics with OpenGL later on in C.

The canvas provides an API for custom rendering using JavaScript. This canvas is then embedded inside the Document Object Model (DOM) of an HTML web page.

A typical HTML page consists of various tags structured in a hierarchical way. Tags are indicated by angle brackets and tags must have an opening (`<MyTag>`) and closing (`</MyTag>`) element. The opening tag contains the type of the tag and the closing element contains a forward slash followed by the tag type again.

The HTML language is a declarative language, which is different to what you have seen so far. HTML code simply describes the content of a web page, you have no control over the flow of execution as you would in an imperative programming language. In HTML, we describe the content of the page that the browser will display. We then use other technologies, like Cascading Style Sheets (CSS) to describe the look/style of the page. Using a programming language like JavaScript (JS), we can perform logic, fetch data asynchronously, and update both the content (HTML) and style (CSS) of the page. Websites typically use all of these technologies together to display dynamic, interactive and visually appealing pages.

As this is not a web development course, we will not focus on the specifics of HTML/CSS/JavaScript but will use JavaScript to paint/draw on an HTML Canvas element.

In the example in Listing 1 on the following page, we see an HTML page with "Page Title" as the title that will be shown in the browser window. "My Heading" will be displayed in the default font for an `h1` heading (`<h1></h1>`) and there is a paragraph of text (`<p></p>`).

We also have a `canvas` element which we can draw on later. We set the `id`, `width`, `height`, and `style` properties of the element inside the opening tag.[1] Inside the tag, we also place text that should be shown if the browser does not support the HTML5 canvas.

Using the example in Listing 1 on the next page, create `lab0.html` in a text editor. Save the file and then open it in a browser.

2 Making Things Dynamic

As it stands, our webpage is a static page and will always display the same thing. Usually, websites can make their pages dynamic in two different ways – server-side or client-side programming languages. Server-side languages (e.g. PHP, Python, ASPX, C#, JavaScript[2] etc.) run on the webserver and are used to generate the HTML/CSS content that gets sent to the web browser. Client-side languages (e.g.

[1] Usually the style would be set in a CSS file.

[2] Note that JavaScript can be used as a server-side programming language through frameworks such as Node.js.

```
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>My Heading</h1>
    <p>This is a very basic webpage.</p>
    <canvas id="myCanvas" width="600" height="600"
            style="border:1px solid #d3d3d3;">
        Your browser does not support the HTML5 canvas tag.
    </canvas>
  </body>
</html>
```

Listing 1: A basic HTML page.

JavaScript, VBScript, WebASM) are sent from the server to the browser in text/bytecode and then run locally in the browser which acts as an interpreter.

JavaScript is the most popular language used in the browser and the Chrome V8 JavaScript engine is a particularly fast JavaScript runtime bundled in many browsers.

In Listing 2 on the following page we have updated the page to include two `script` sections. One after the body and one in the head. The script sections will be parsed and run as they are encountered by the browser. It is important to note that the rest of the webpage might not have been loaded yet, as the browser loads resources at different times. In the code, line 22 displays an alert that is triggered as the browser encounters it. On line 5, we rather declare a function called `init`. On line 12, we then update the `body` tag, telling it to call our `init` function as soon as the page is loaded and ready.

Update your lab0.html file and refresh the page in the browser.

3 Drawing

In this section, we will do some very basic drawings. At this stage, we will manage the process manually, but later on, we'll see that geometric transformations and hierarchical modelling make this process significantly easier.

```
<html>
  <head>
    <title>Page Title</title>
    <script>
    function init(){
            // This will run when the page as fully loaded and the
            //   body's onload event is triggered.
            alert("Init Function");
    }
    </script>
  </head>
  <body onload="init()">
    <h1>My Heading</h1>
    <p>This is a very basic webpage.</p>
    <canvas id="myCanvas" width="600" height="600"
            style="border:1px solid #d3d3d3;">
      Your browser does not support the HTML5 canvas tag.
    </canvas>
  </body>
  <script>
  // This will run as the browser's interpreter reaches the line.
  alert("Hello World!");
  </script>
</html>
```

Listing 2: A basic HTML page with JavaScript.

3.1 2D Context

The HTML5 canvas provides a context object that we can use to draw to draw on the canvas. At this stage, we will be using the 2D context, but later on, in the course, we will do 3D rendering with WebGL. In line 1 we use JavaScript to get a reference to the `myCanvas` object that was defined in our HTML. In line 2 we get the 2D context from the canvas, which gives us an API to stroke/fill points, lines, paths, arcs, polygons etc.

```
1 var c = document.getElementById("myCanvas");
2 var g = c.getContext("2d");
```

We will put this in the script tag before the init function.

Now that we have a reference to the graphics context we can use its member functions to render to the canvas.

We distinguish between stroking and filling.

- Stroking - we run a virtual pen along a path.
- Filling - we paint the inside of a path to colour in a polygon.

We can set the colour for these operations using any valid CSS colour string (e.g. “blue”, “rgb(0,0,255)”, “#0000FF”) using the following lines:

```
1 // Sets the strokeStyle to black
2 g.strokeStyle = "rgb(0,0,0)";
3 // Sets the fillStyle to red
4 g.fillStyle = "#ff0000";
```

To create a path we have a few functions:

- `g.beginPath()` - must be called to create a new path at the beginning of each new path.
- `g.moveTo(x,y)` - move the path cursor to (x, y).
- `g.lineTo(x,y)` - make a line from the previous point on the path to (x,y).
- `g.closePath()` - make a line from the previous point on the path to the first point in the path.

Once you have created a path you can then call:

- `g.stroke()` - use the strokeStyle to draw a line over the path.
- `g.fill()` - use the fillStyle to colour the inside of the path.

We can use the functions above to update our code as shown in Listing 3 on the next page and draw a line on the canvas.

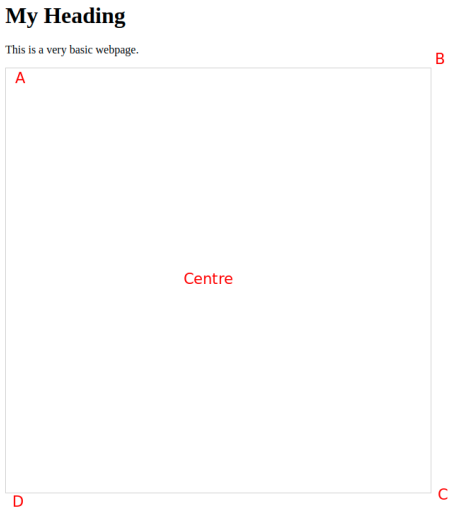
```
<html>
  <head>
    <title>Page Title</title>
    <script>
      var canvas;
      var graphics;

      function line(x1, y1, x2, y2){
        graphics.beginPath();
        graphics.moveTo(x1, y1);
        graphics.lineTo(x2, y2);
        graphics.stroke();
      }

      function init(){
        canvas = document.getElementById("myCanvas");
        graphics = canvas.getContext("2d");

        line(0,0,300,300);
      }
    </script>
  </head>
  <body onload="init()">
    <h1>My Heading</h1>
    <p>This is a very basic webpage.</p>
    <canvas id="myCanvas" width="600" height="600"
            style="border:1px solid #d3d3d3;">
      Your browser does not support the HTML5 canvas tag.
    </canvas>
  </body>
</html>
```

Listing 3: Some basic drawing.

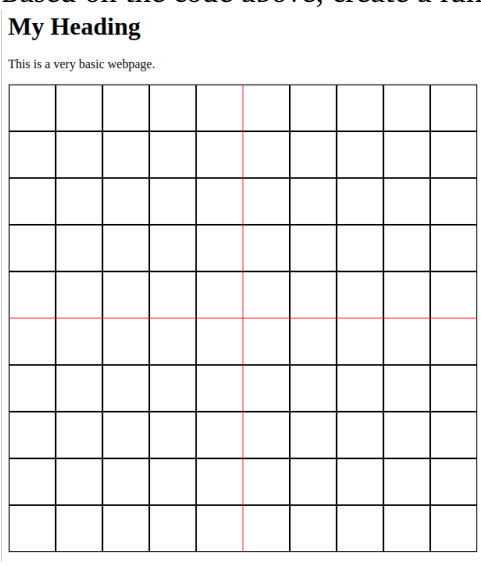
3.2 Tell me the time.

Adapt the code in Listing 3 to complete the tasks in this section. We are going to build a clock face.

1. What is the width and height of the canvas?
2. Based on the line drawn by the code, where is the origin of the canvas and what are the coordinates of the areas marked below?



3. Based on the code above, create a function that draws axes as shown below.[3]



4. Knowing that you can parameterise a circle with radius, r, and centre, c, as follows

$$p(\theta) = \begin{bmatrix} x(\theta) \\ y(\theta) \end{bmatrix} = r \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix},$$

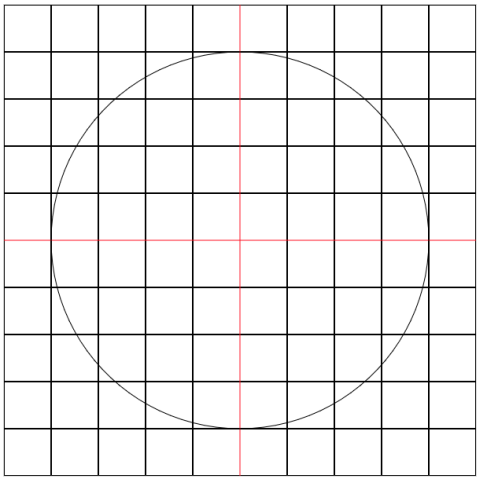
create a function that draws a circle of radius 240 centered in the middle of the canvas. Draw the circle by generating points by sweeping over different values of $\theta \in [0, 2\pi]$ and creating a path from $p(\theta_i)$ to $p(\theta_{i+1})$. Close and then stroke the path. See how the circle becomes smoother

[3] Do not hardcode the lines. For-loops are your friend.

as you increase the number of points.

My Heading

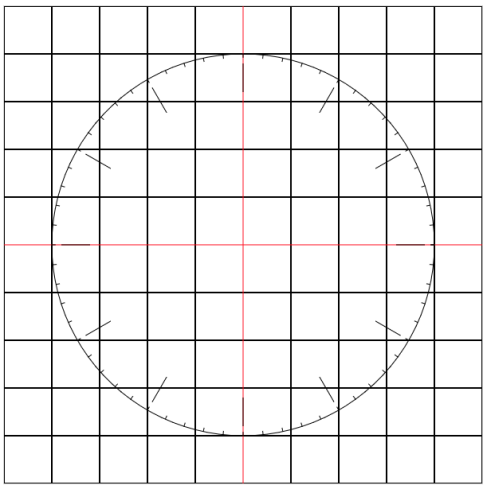
This is a very basic webpage.



5. We need to lines to represent the 'tics' for the hours and minutes of our clock. Create a function that adds these lines in. Hint: You want to draw a line between points with a different r but the same θ.

My Heading

This is a very basic webpage.



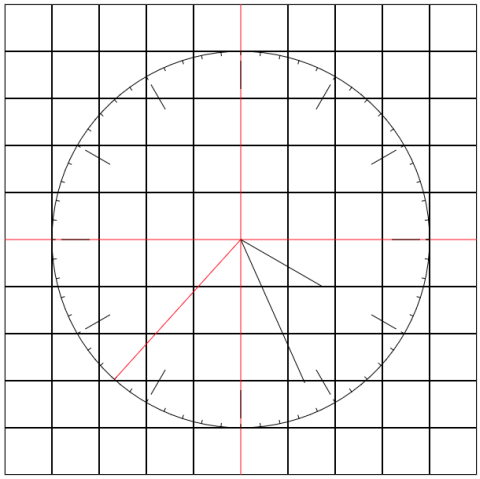
6. Using a JavaScript date object, change the text on screen to show the current datetime.

```
let d = new Date();
document.getElementById("some\_id").innerHTML = d;
```

7. Create the hour, minute and second hands. You need to create a function that converts the current time into a θ value for each hand before drawing each of the three lines.

My Heading

Mon Mar 07 2022 16:26:37 GMT+0200 (South Africa Standard Time)

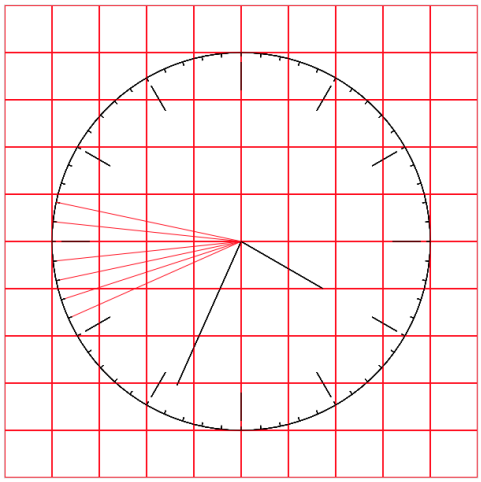


8. Use the `setInterval` function to call your init function every second – `setInterval(init, 1000)`. This function should redraw your clock with the current time. Do you notice anything strange?

    - If you set did not set the colour back to black after drawing the second-hand you'll notice that this causes a strange side effect.
    - Because we are just drawing every second, the old content of the canvas is still visible. We need to clear the canvas each time we draw so that we have a fresh start for each render of the clock.

My Heading

Mon Mar 07 2022 16:34:47 GMT+0200 (South Africa Standard Time)

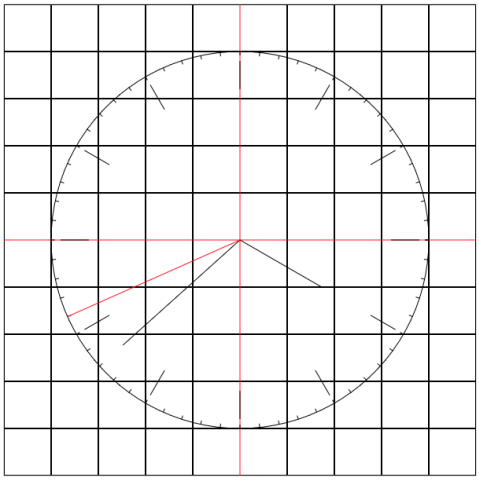


9. Correct the issues above. Make sure that you reset the colour after drawing the second-hand in red. Before drawing the clock each time, you should wipe the canvas by calling:

```
1 graphics.clearRect(0, 0, canvas.width, canvas.height);
```

My Heading

Mon Mar 07 2022 16:38:41 GMT+0200 (South Africa Standard Time)



10. Finally, see if you can adjust your code so that the hands all move smoothly around the clock (i.e. the hour hand can be at 3.3) and the second hand is updated every 10ms. Also, remove the axes/guidelines that we added earlier to help us design the clock.

My Heading

Mon Mar 07 2022 16:43:21 GMT+0200 (South Africa Standard Time)