

COMP4021
Internet Computing

HTML Canvas

Gibson Lam

HTML Canvas

- The HTML `<canvas>` element is an area inside an HTML page that you can use to draw things
- It handles bitmap content only, i.e. you can see its 'pixels' if you zoom in to the area
- You create visual content inside a canvas area using JavaScript code
- It is especially good for making complex visualizations and games in web pages

Canvas or SVG

- Basic reason – you choose canvas if you want bitmap content, whereas you choose SVG if you want vector graphics
- Other reasons to choose canvas over SVG:
 - You want to create highly dynamic content
 - You handle a lot of moving things
 - You want to create 2D and/or 3D content

The Canvas Element

- The `<canvas>` element is a rectangular area that can be created like this:

```
<canvas width="600" height="400">  
</canvas>
```

- The above HTML creates a 600 by 400 empty rectangular area in the web page
- You add content inside the area using JavaScript later

Using JavaScript

- As an example, you can create your canvas content in the jQuery ready event

...

```
<canvas width="600" height="400"></canvas>
```

...

```
<script>
```

```
$(document).ready(function() {  
    let cv = $("canvas").get(0);
```

...creating the canvas content...

```
});
```

```
</script>
```

...



*Get the canvas
element*

Getting the Context

- You need to get a 'drawing context' before starting to draw things in canvas
- There are two types of contexts:
 - Getting a 2D context for 2D content
`cv.getContext("2d");`
 - Getting a WebGL context for 3D content
`cv.getContext("webgl");`
- In this presentation, we will only look at how to use it to create 2D content

A Simple 2D Example

- Here is an example that gives you a black rectangle inside a canvas area:

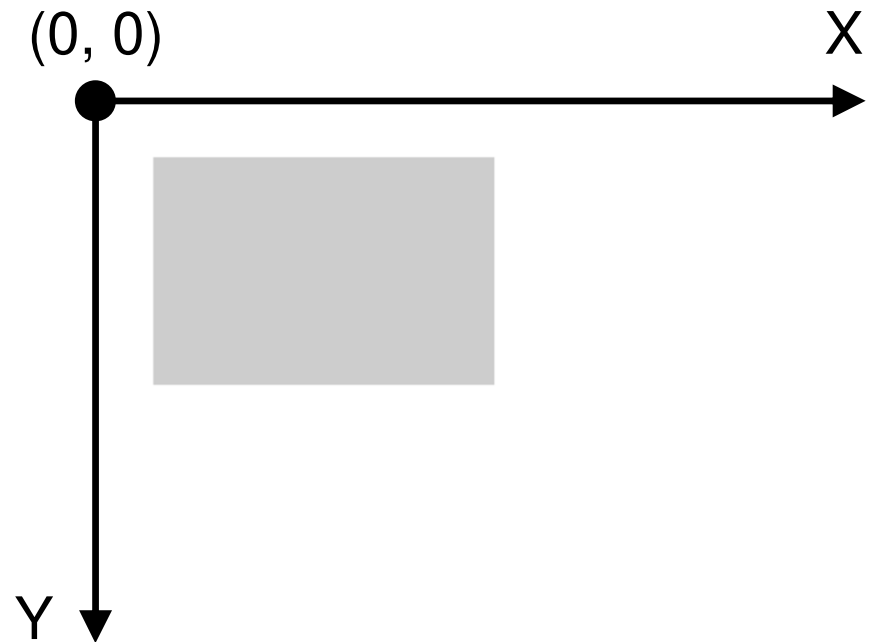


```
...  
<canvas width="600" height="400"></canvas>  
...  
<script>  
$(document).ready(function() {  
    let cv = $("canvas").get(0);  
    let context = cv.getContext("2d");  
    context.fillRect(50, 50, 300, 200);  
});  
</script>  
...
```

*Draws a rectangle
inside the canvas*

The Coordinate System

- The coordinate system inside canvas is similar to other systems, i.e. SVG:
 - The top left hand corner is the origin and the y value increases downwards

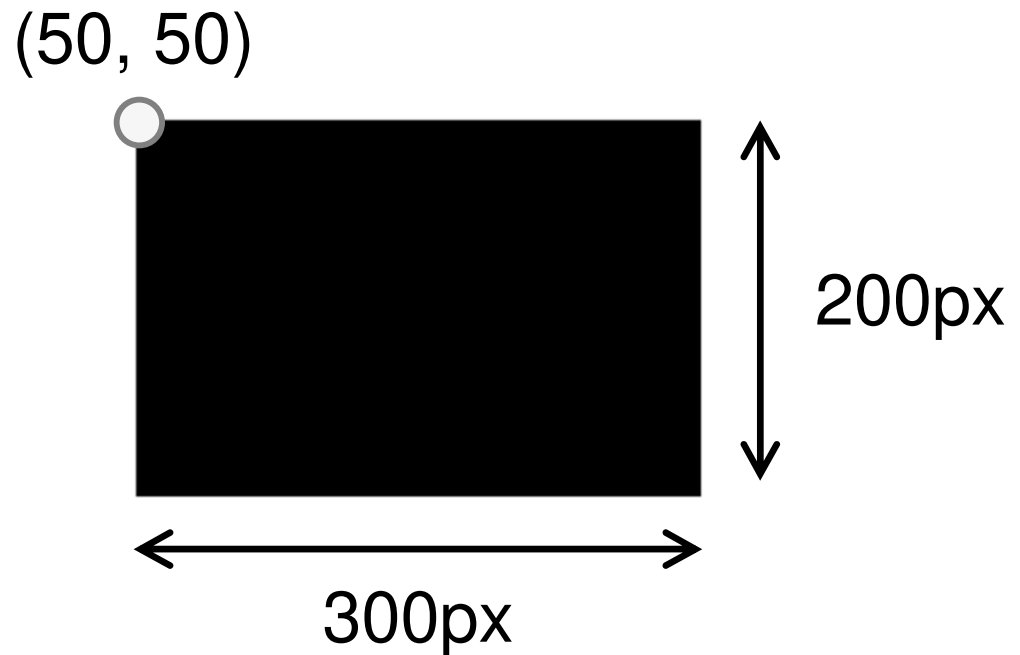


The Example Rectangle

- Using the coordinate system, this line of code:

```
context.fillRect(50, 50, 300, 200);
```

gives you this
rectangle in the
canvas area



Drawing Rectangles

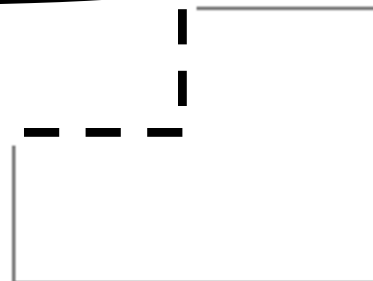
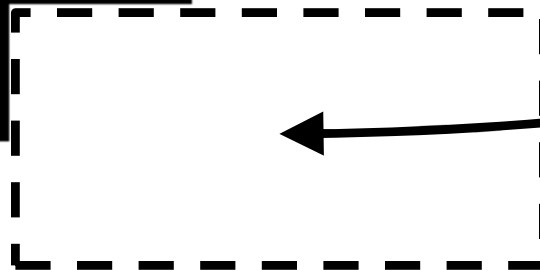
- As you can see in the previous example, `fillRect()` gives you a solid rectangle (a rectangle filled with black in the example)
- If you want to, you can also draw a hollow rectangle using `strokeRect()`
- Or, instead of drawing, you can clear a rectangular area using `clearRect()`

An Example With Two Rectangles

- For example, you can draw two rectangles and then clear part of those rectangles:

```
context.fillRect(50, 50, 200, 150);  
context.strokeRect(350, 200, 200, 150);
```

```
context.clearRect(150, 125, 300, 150);
```



*This area has
been cleared*

Fill And Stroke Styles 1/2

- Rather than the boring black colour, you can adjust the fill colour and outline of the rectangles
- For example, you can draw a rectangle filled with red like this:

```
context.fillStyle = "red";  
context.fillRect(50, 50, 200, 150);
```

Fill And Stroke Styles 2/2

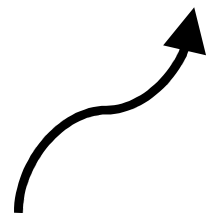
- You can also change the outline colour and width (line width) of the rectangle, like this:



*Rectangle
filled with red*



```
context.strokeStyle = "red";  
context.lineWidth = 3;  
context.strokeRect(350, 200, 200, 150);
```



Drawing Other Things

- You use `<path ... />` in SVG to draw shapes freely
- You use paths in canvas to draw simple shapes too, e.g.:
 - Drawing lines
 - Drawing circles
- You can also draw text and images in canvas easily

Using Paths

- Canvas paths are similar to SVG paths where you use some commands to move and draw around a 'path'
- You use `beginPath()` in canvas to begin a new path
- Once you finish a path, you can draw the path by filling the path, i.e. `fill()`, or draw the outline of the path, i.e. `stroke()`

Drawing Lines and Circles

- To draw a line, you move to a certain location and then draw a line to another location, e.g.:

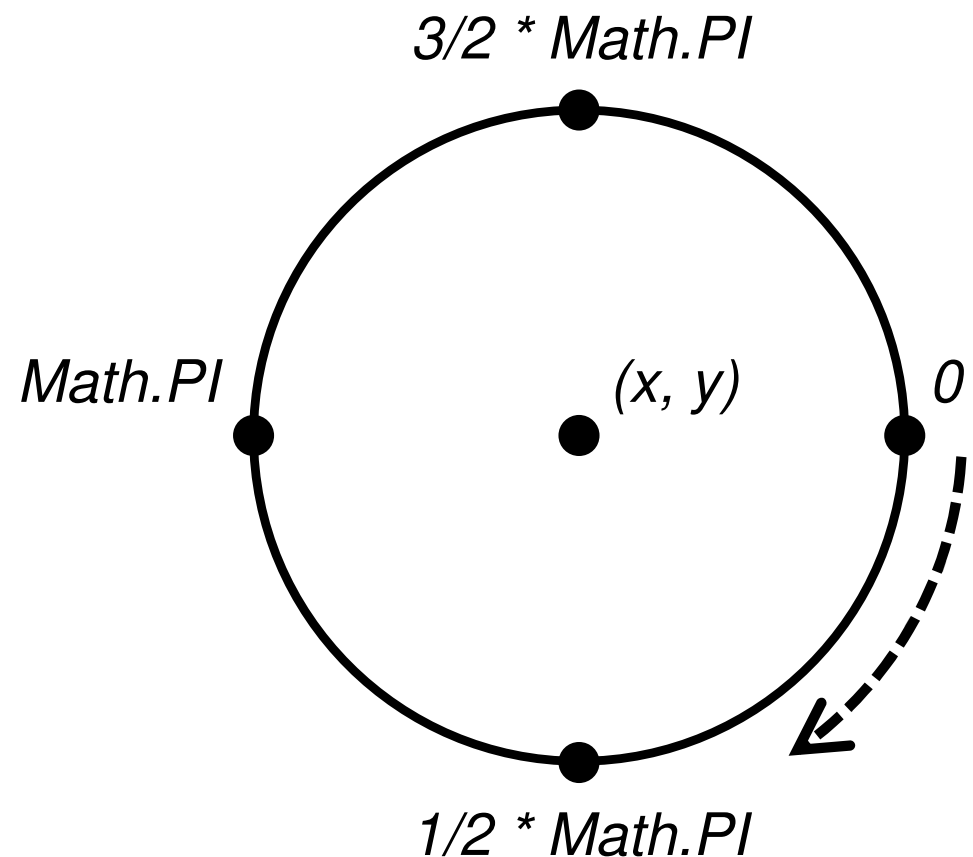
```
// a horizontal line  
context.moveTo(50, 50);  
context.lineTo(550, 50);
```

- You draw a circle using the `arc()` function that has several parameters:

```
arc(x, y, radius, start angle, end angle);
```

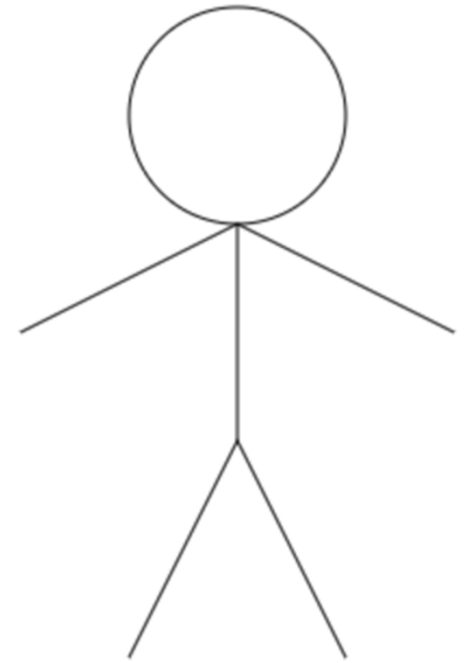

Drawing Circles

- An angle of 0 is on the right hand side of a circle
- The angle then rotates around in clockwise direction
- You can draw an arc from any two angle values



Drawing a Path

- Here is an example which draws a circle and a few lines in a single path:



```
context.beginPath();
```

```
context.arc(300, 100, 50, 0, 2 * Math.PI);
```

```
context.moveTo(300, 150); context.lineTo(300, 250);
```

```
context.moveTo(300, 150); context.lineTo(200, 200);
```

```
context.moveTo(300, 150); context.lineTo(400, 200);
```

```
context.moveTo(300, 250); context.lineTo(250, 350);
```

```
context.moveTo(300, 250); context.lineTo(350, 350);
```

```
context.stroke();
```

Draw the path using outlines

Using isPointInPath()

- `isPointInPath()` is a useful function
- It allows you to check whether a point is inside a path or not
- This is very useful for games:
 - For example, you can determine whether a player successfully clicks on some enemy objects inside a game

A Clicking Example

- In the following example, code has been used to draw three quarters of a red circle and a blue rectangle inside a canvas area:

```
/* Draw a circle */  
context.beginPath();  
context.arc(150, 100, 50, 0,  
           1.5 * Math.PI);  
context.lineTo(150, 100);  
context.fillStyle = "red";  
context.fill();
```

```
/* Draw a square */  
context.beginPath();  
context.rect(350, 200, 150, 100);  
context.fillStyle = "blue";  
context.fill();
```



*Create a
'path version'
of a rectangle*

Inside the Click Event

- After clicking on the canvas area, the event handler creates the paths again without actually drawing the shapes
- These newly created paths are tested against the mouse position
- First, the mouse position relative to the canvas element is returned using some jQuery code:

```
let offset = $("canvas").eq(0).offset();  
let x = event.pageX - offset.left;  
let y = event.pageY - offset.top;
```

Testing the Red Circle

- Then, each path is created without running `fill()` or `stroke()`
- For example, the following code creates the circle again but the path is used for checking against the mouse position:

```
context.beginPath();
context.arc(150, 100, 50, 0, 1.5 * Math.PI);
context.lineTo(150, 100);
if (context.isPointInPath(x, y)) {
    alert("You clicked on the red circle!");
}
```

Drawing Text

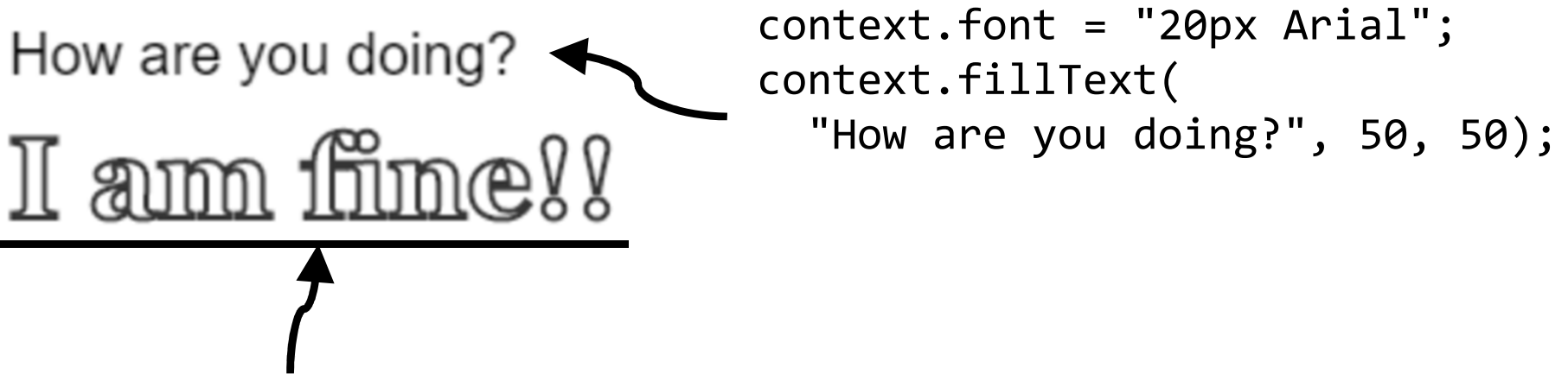
- You draw text by first defining the font and then drawing the text, like this:

```
context.font = "20px Arial";  
context.fillText("How are you doing?",  
                50, 50);
```

- You can create interesting text effect with only outline using `strokeText()`
- You can also use `measureText()` to get the width of some text before drawing

Drawing Some Example Text

- Here is the display of some text:



How are you doing?

I am fine!!

```
context.font = "20px Arial";  
context.fillText(  
    "How are you doing?", 50, 50);
```

- The outline text is drawn by this code:

```
context.strokeStyle = "red";  
context.lineWidth = 2;  
context.font = "bold 40px Georgia";  
context.strokeText("I am fine!!", 50, 100);
```


Drawing Images

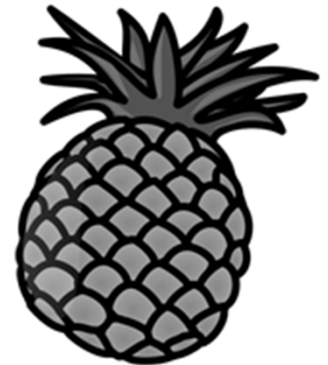
- You can draw images anywhere in canvas
- To do that, first you need to load an image, which can be:
 - An image object, e.g. from an `` tag
 - Another canvas area
 - A video element

Drawing Using an Image Element

- Let's assume that an `` tag has been put inside the web page:

```

```



- You can then draw the image at (50, 50) in canvas using this code:



```
context.drawImage(
  $("#pineapple").get(0), 50, 50);
```

*You get the image from
the `` element*

*The location to
put the image*

Loading and Drawing Images

- If you want to, you can load an image inside your JavaScript, like this:

```
let image = new Image();  
image.src = "pineapple.png";
```

- The image created this way stays in the JavaScript memory without getting inside the DOM, i.e. not in the web page body

Problem in the Code

- You may find that this code sometimes does not draw the image correctly :

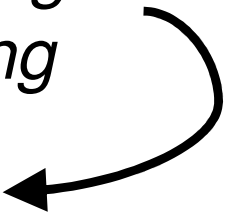
```
let image = new Image();  
image.src = "pineapple.png";  
context.drawImage(image, 50, 50);
```

- This is because the code may take some time to load the image after setting the image src attribute

Using the Load Event

- To make sure the code runs correctly, you need to use the load event of the image
- Here is an example that runs the drawing code only after the image is loaded:

```
let image = new Image();  
image.onload = function() {  
    context.drawImage(image, 50, 50);  
};  
image.src = "pineapple.png";
```

*Draw the image
after loading* 

Using the 3D Context

- The 2D context allows you to make nice 2D content such as 2D games
- Using the 3D context, you can then make 3D games as well inside a canvas
- It uses WebGL, which is an API based on the OpenGL 3D API
- An example is shown on the next slide but we will not go into the details in this course

The Example 3D Cube

- You can rotate, move or zoom the cube by a combination of the mouse, the shift key and the ctrl key

