



The
University
Of
Sheffield.

COM1008: Web and Internet Technology

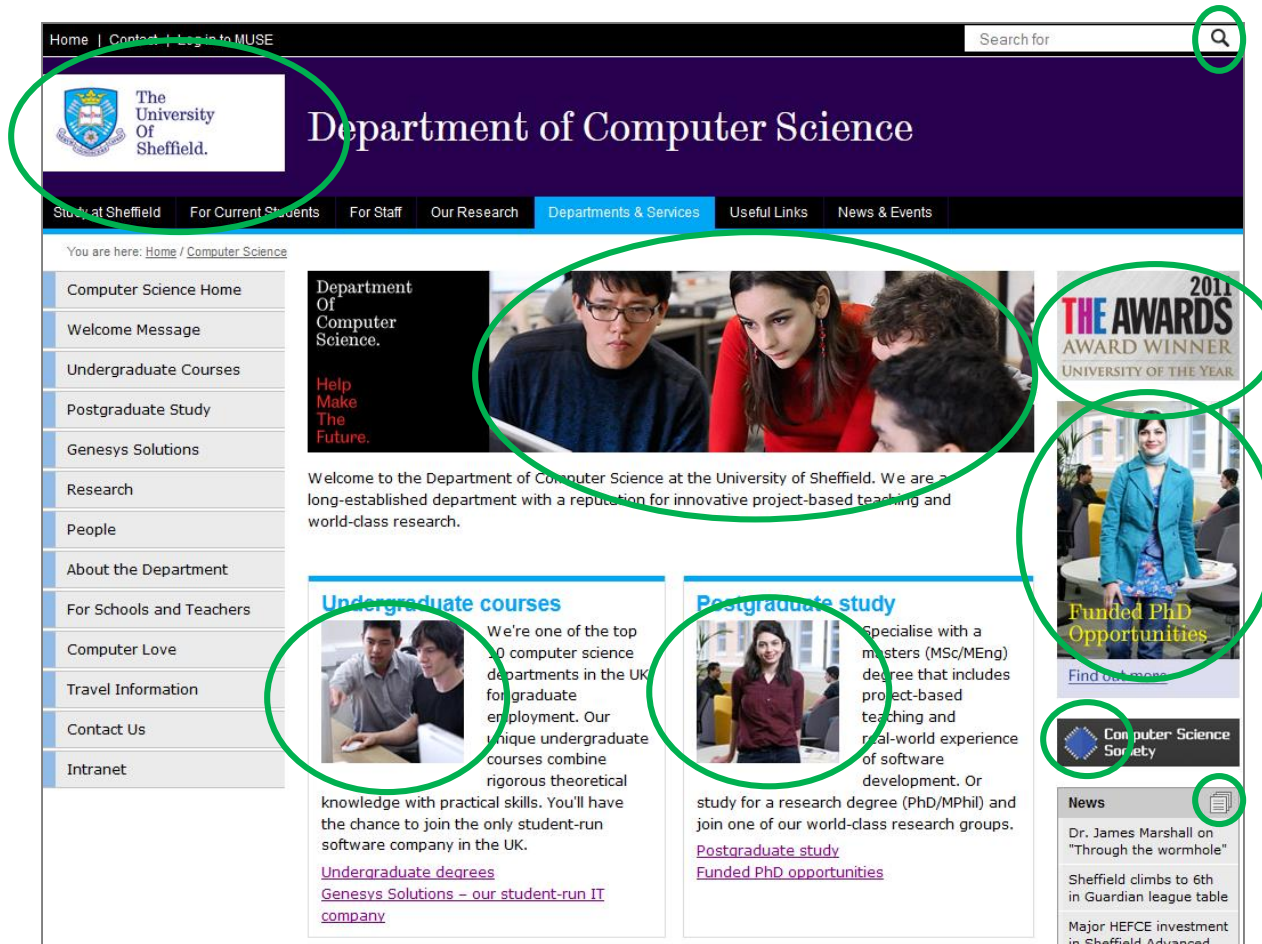
Lecture 14: Graphics on the Web: Part 1



Dr. Steve Maddock
s.maddock@sheffield.ac.uk

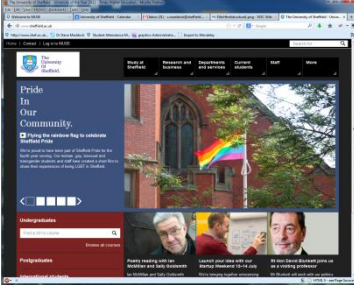


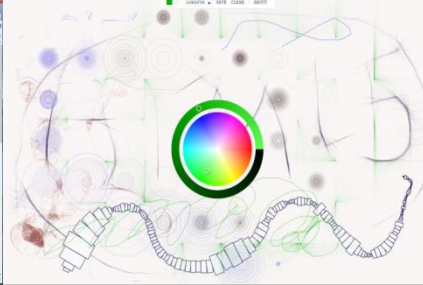
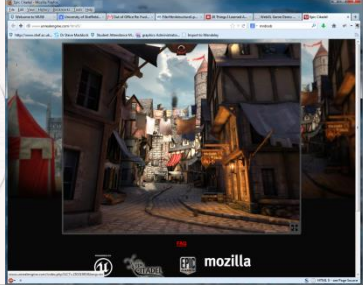
1. Introduction

- Most Web pages feature some form of graphical image



1. Introduction

- Lots of different kinds of graphics can be displayed on a web page

				
www.shef.ac.uk	YouTube	Google charts	Harmony, mrdoob.com	Epic Citadel
images	video	data visualisation	graphical interaction	3D games

- Cover aspects of these over the next few lectures
 - Vector and raster graphics (today and next week)
 - Main focus: The canvas (today)**
 - Animation (today and next week)
 - Interaction on the canvas (next week)

2. Vector graphics and raster graphics

- In general, two kind of 2D computer graphics approaches:
- Raster graphics
 - Drawing/painting commands produce sets of coloured pixels on the screen
 - Screen image is made of a rectangular grid of pixels – bitmap
- Vector graphics
 - Image is a list of geometric shapes with attributes

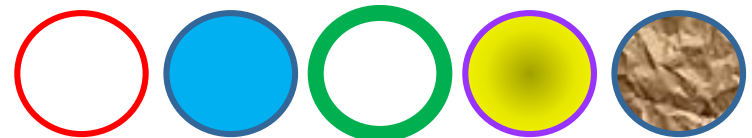


List of shapes:

Circle(centre, radius,
border_colour, fill_colour,
fill_image)

Circle(...

...



2.1 Raster graphics

- Screen image is made of a rectangular grid of pixels
 - A bitmap (or pixmap) is an array of numbers
 - Each number represents the *colour* of a pixel (24-bits)
- Data stored in a block of memory called the frame buffer
- Screen redrawn many times per second
- (Data stored in file in possibly compressed format. e.g. .img)

D98407	EDAF00	FFDB14	FFDD00	FFCE00	FABB00
AD111D	C2361C	C66017	E5C100	FFE000	FFE32C
AC191D	B3101D	AC111C	BB4C1A	CF9601	F1CF00
A51B1C	AE131D	AB201C	A0191C	99141B	A92E1C
A65619	A23E1A	AF421B	AD331C	A51D1C	A9191C
D5A900	B76017	AC4C1A	AC5219	B04E1A	AD401B

stored values

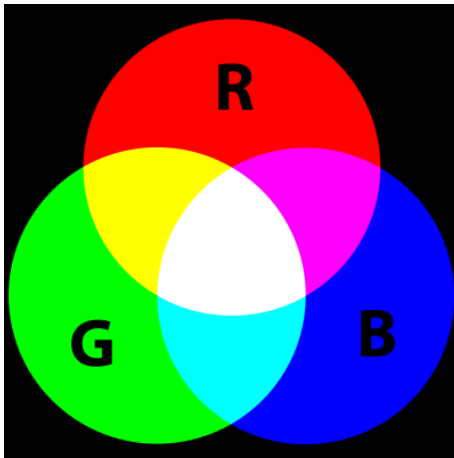


displayed pixels

Chapman, N and J. Chapman, Web Design: A complete introduction, John Wiley & Sons, 2006.

2.1 Raster graphics

- Pixel colour is a mix of three additive primaries: red, green and blue (8 bits for each: 0..255, 0..255, 0..255)
 - RGB colour model



<http://en.wikipedia.org/wiki/File:AdditiveColor.svg>



2.2 Vector graphics

- Image is a list of geometric shapes
 - A mathematical description of a picture
- Each shape can have attributes
 - e.g. centre coordinates
- Screen image created by drawing the list
 - List is retained
- When drawn on a display the floating point vector data is eventually converted to integer grid of pixels

List of shapes:

```
Circle(centre, radius,  
        border_colour, fill_colour,  
        fill_image)
```

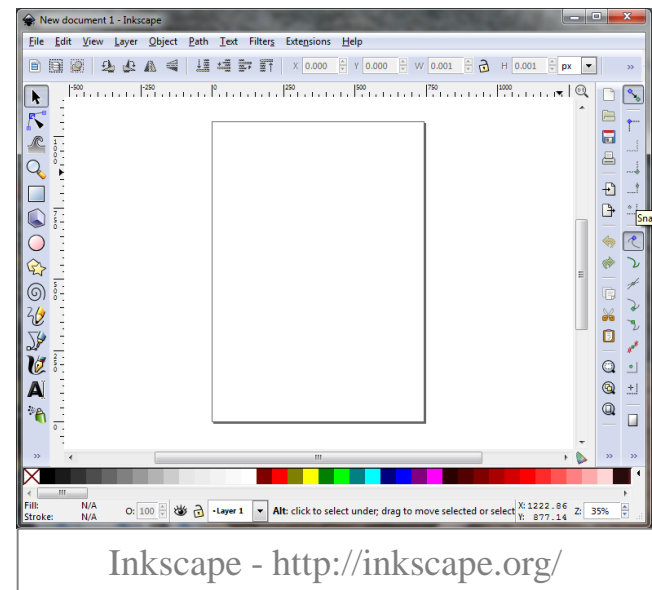
```
Circle(...  
...
```



<http://en.wikipedia.org/wiki/File:VectorBitmapExample.png>

2.3 Examples

- Raster graphics – Paint package
 - Assign colours to pixels using paintbrush
 - Only colour of pixels is saved
 - Can only subsequently edit pixels
 - Typically supports layers of bitmaps
- Vector graphics – Drawing program
 - Add geometric shapes chosen from menu
 - Saved as list of shapes and attributes
 - Can edit individual shapes
- Some programs offer a mixture of both

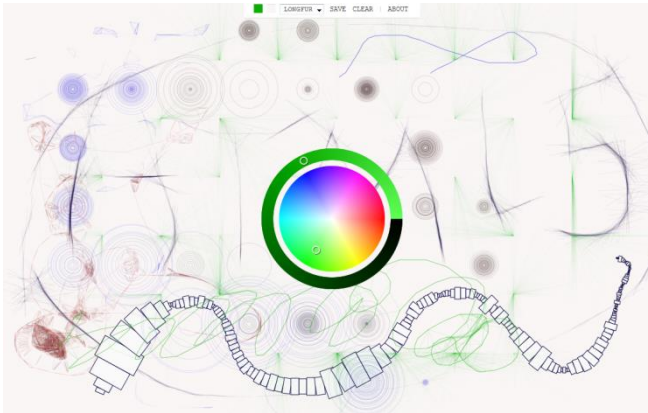


2.4 Summary

- Raster graphics
 - HTML5 and the canvas element
 - The program uses drawing/painting commands to set particular pixels on the screen
 - The set of pixels representing the screen image is the representation
- Vector graphics
 - SVG is W3C's recommended markup language for vector graphics
 - (Much of vector graphics on the Web is done using SWF (Flash) files)
 - The list of drawing/painting commands is the representation
 - More on SVG in a later lecture...

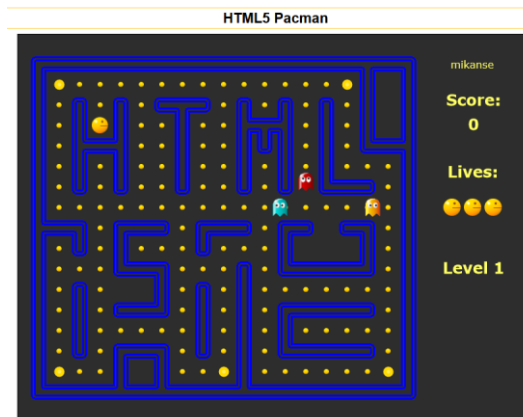
3. The canvas

- The canvas is an HTML5 element on which graphics can be displayed
 - E.g. Interactive graphical software such as paint tools or games
 - Recent Web browser versions also support 3D graphics using WebGL



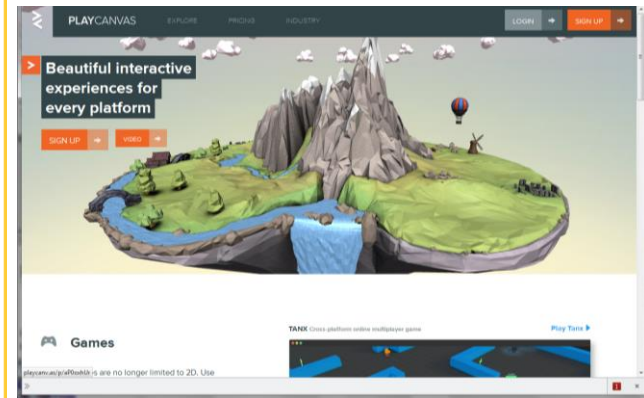
Harmony -

<http://mrdoob.com/projects/harmony>



Games

<http://www.mikanse.com/PacMan/pacman.html>

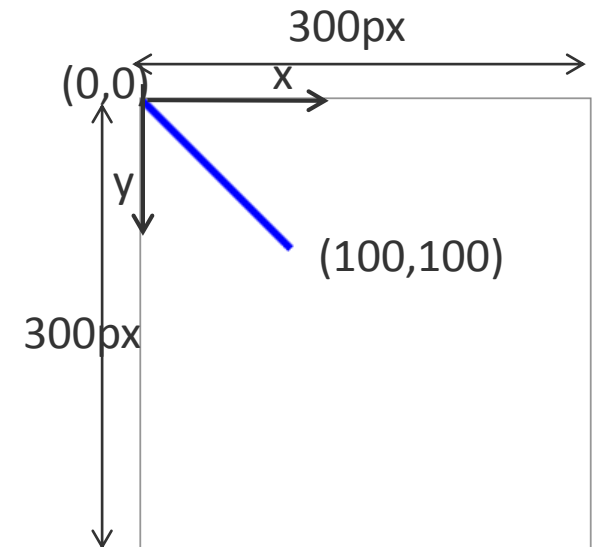


<https://playcanvas.com/>

4. Drawing a line - [Example 1](#)

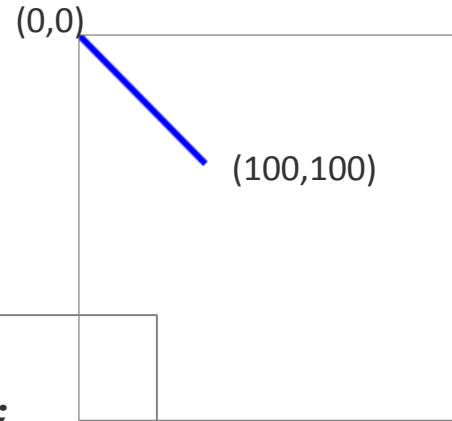
- Used in a similar way to the img element

```
<body>
  <canvas id="example"
    width="300" height="300"></canvas>
</body>
```



```
function render() {
  var canvas = document.getElementById("example");
  var context = canvas.getContext("2d");
  context.strokeStyle = "rgb(0,0,255)";
  context.lineWidth = "5";
  context.beginPath();
    context.moveTo(0,0);
    context.lineTo(100,100);
  context.closePath();
  context.stroke();
}
```

4. Example 1



```
function render() {  
  var canvas = document.getElementById("example");  
  var context = canvas.getContext("2d");  
  context.strokeStyle = "rgb(0,0,255)";  
  context.lineWidth = "5";  
  context.beginPath();  
    context.moveTo(0,0);  
    context.lineTo(100,100);  
  context.closePath();  
  context.stroke();  
}
```

- First get the canvas element
- Then get the rendering context – used to manipulate the content of the drawing surface

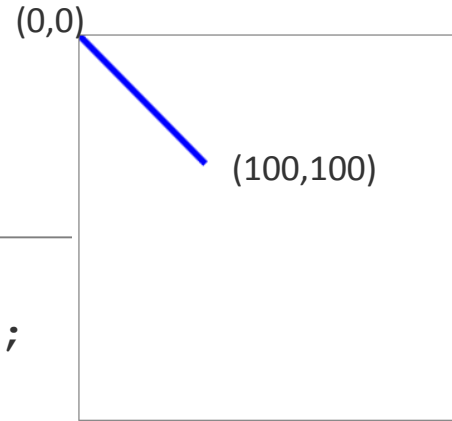
4. Example 1

```
function render() {  
  var canvas = document.getElementById("example");  
  if (canvas.getContext) {  
    var context = canvas.getContext("2d");  
    context.strokeStyle = "rgb(0,0,255)";  
    context.lineWidth = "5";  
    context.beginPath();  
      context.moveTo(0,0);  
      context.lineTo(100,100);  
    context.closePath();  
    context.stroke();  
  }  
  else {  
    // fallback  
  }  
}
```

- Should check the context exists

4. Example 1

```
function render() {  
  var canvas = document.getElementById("example");  
  var context = canvas.getContext("2d");  
  context.strokeStyle = "rgb(0,0,255)";  
  context.lineWidth = "5";  
  context.beginPath();  
    context.moveTo(0,0);  
    context.lineTo(100,100);  
  context.closePath();  
  context.stroke();  
}
```



- Set the attributes, e.g. **colour**, **width** of the pen
- Line is not drawn until the method **'stroke'** is called

5. jQuery version

- Instead of using DOM functions, and dealing with the extra JavaScript that may be required for cross-browser support, jQuery can be used
 - Open source JavaScript library, www.jquery.com

```
<!-- incorporate jQuery -->  
<script src =  
    "http://code.jquery.com/jquery-2.1.3.min.js"></script>  
<!-- more stuff -->
```

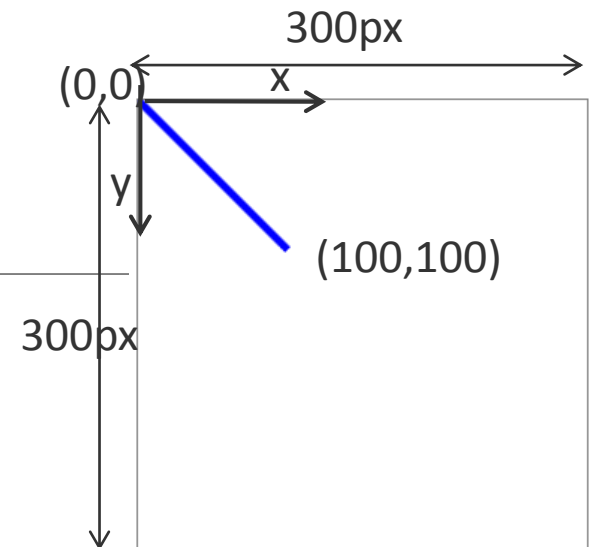
- Can also download a version for inclusion locally

```
<script src = "../js/jquery-2.1.3.min.js"></script>
```


5.1 Example 1

- Returns jQuery object
- Returns DOM element

```
function render() {  
    var context = $('#canvas_example')[0].getContext("2d");  
    context.strokeStyle = "rgb(0,0,255)";  
    context.lineWidth = "5";  
    context.beginPath();  
        context.moveTo(0,0);  
        context.lineTo(100,100);  
    context.closePath();  
    context.stroke();  
}
```



6. Example 2 – Lots of lines

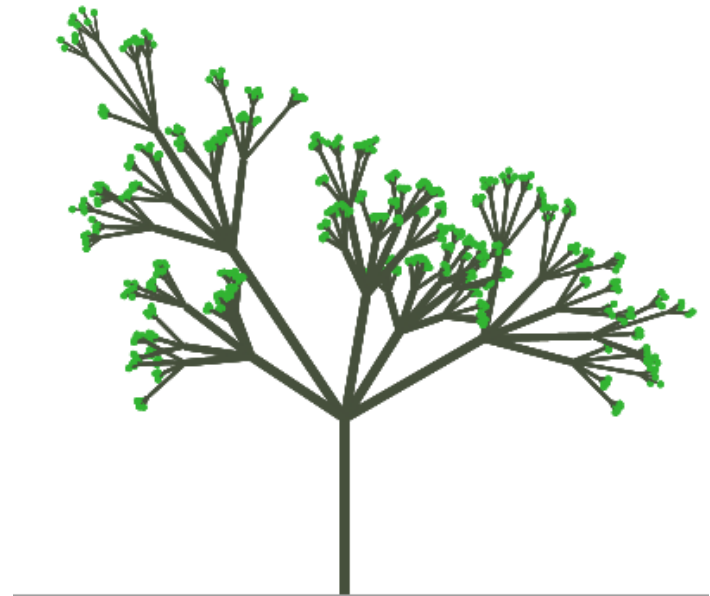
```
function render() {  
  var canvas = document.getElementById("example1");  
  var context = canvas.getContext("2d");  
  <!-- now draw -->  
  context.strokeStyle = "rgb(0,0,255)";  
  context.lineWidth = "1";  
  for (var i=0; i<100; i++) {  
    var x1 = Math.random()*300;  
    var y1 = Math.random()*300;  
    var x2 = Math.random()*300;  
    var y2 = Math.random()*300;  
    context.beginPath();  
    context.moveTo(x1, y1);  
    context.lineTo(x2, y2);  
    context.closePath();  
    context.stroke();  
  }  
}
```



7. Trees

Pseudocode (recursion):

```
draw tree (depth) {  
  if depth = 0 draw leaf  
  else {  
    draw branch  
    for number of branches {  
      draw tree(depth-1)  
    }  
  }  
}
```





8. Paths

- `beginPath()`, `closePath()`, `stroke()`, `fill()`
- Path can contain `moveTo`, `lineTo`, `arc`, `quadraticCurveTo`, `bezierCurveTo`, `rect`

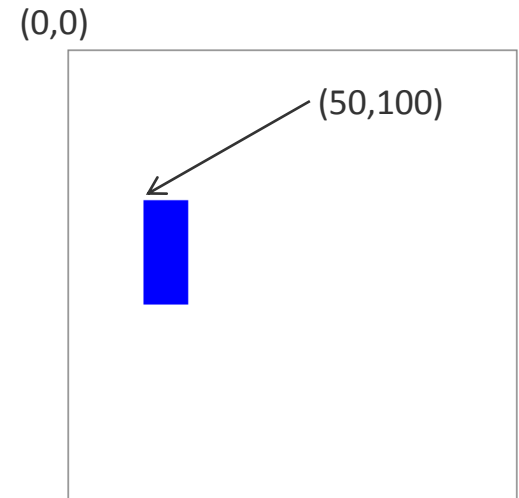
`arc(x, y, radius, startAngle, endAngle, anticlockwise)`

```
function render() {  
  var ctx = document.getElementById('canvas').getContext('2d');  
  ctx.beginPath();  
    ctx.arc(75, 75, 50, 0, Math.PI*2, true); // Outer circle  
    ctx.moveTo(110, 75);  
    ctx.arc(75, 75, 35, 0, Math.PI, false); // Mouth (clockwise)  
    ctx.moveTo(65, 65);  
    ctx.arc(60, 65, 5, 0, Math.PI*2, true); // Left eye  
    ctx.moveTo(95, 65);  
    ctx.arc(90, 65, 5, 0, Math.PI*2, true); // Right eye  
  ctx.closePath();  
  ctx.stroke();  
}
```

Note: If line drawing: First step in a path is `moveTo`; Last point is automatically joined to first point on call of `closePath`

9. Example 4 – Shapes and paths

- A rectangle
 - [rect](#)(x, y, width, height)
 - x, y is the top left corner
- (There is also an alternative way to draw rectangles using [strokeRect\(\)](#) or [fillRect\(\)](#).)



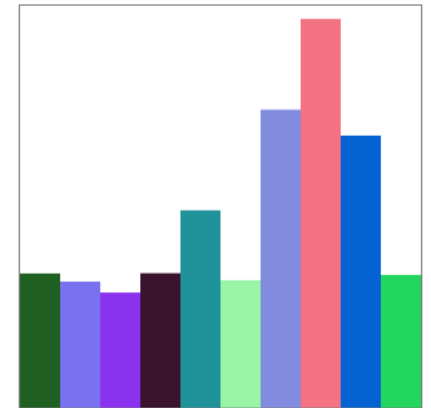
```
function render() {  
  var context = $('#canvas_example')[0].getContext("2d");  
  context.fillStyle = "rgb(0,0,255)";  
  context.beginPath();  
    context.rect(50,100,30,70);  
  context.closePath();  
  context.fill();  
}
```

9.1 Drawing multiple rectangles

```
const WIDTH, HEIGHT; var context;

function drawBar(x, y, width, height, c) {
    context.fillStyle = c;
    context.fillRect(x, HEIGHT-y-height, width, height);
}

function render() {
    var canvas = document.getElementById("canvas_example");
    context = canvas.getContext("2d");
    WIDTH = canvas.width;
    HEIGHT = canvas.height;
    var numbars = 10;
    var width = WIDTH/numbars;
    for (var i=0; i<numbars; ++i) {
        var height = Math.random()*HEIGHT;
        var r = Math.floor(Math.random()*256);
        var g = Math.floor(Math.random()*256);
        var b = Math.floor(Math.random()*256);
        var colour = "rgb("+r+", "+g+", "+b+")";
        drawBar(width*i, 0, width, height, colour);
    }
}
```



10. Images

```
function init() {  
    var canvas = document.getElementById("example");  
    var context = canvas.getContext("2d");  
    var img = new Image();  
    img.onload = function() {  
        context.drawImage(img, 50, 50);  
    }  
    img.src = "image1.jpg";  
}
```

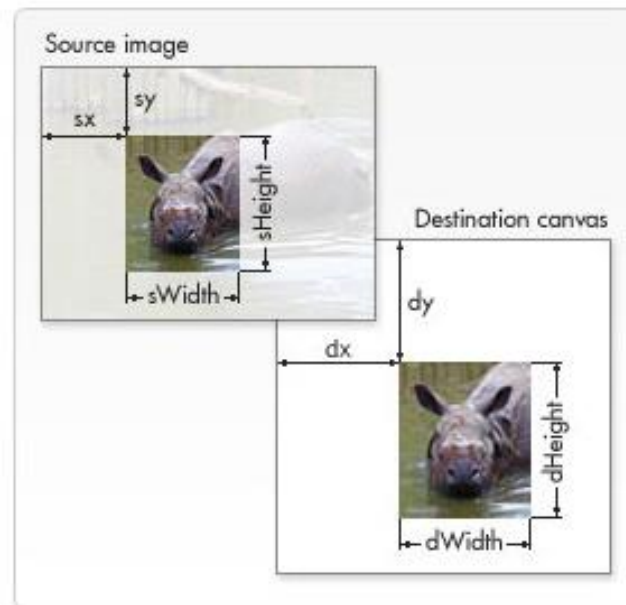
- Create an Image object
- Need to ensure image is loaded before drawing for first time

- onload is given before src
- Need to know what to call when src statement has finished loading the file into the cache



10.1 Image display variants

- `drawImage(image, x, y)`
- `drawImage(image, x, y, width, height)` – scales the image
- `drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)`
– slice part of image before drawing



https://developer.mozilla.org/en/Canvas_tutorial:Using_images

10.2 Multiple images

- Preloading multiple images is a common operation
- Alternative approaches are available
 - see <http://www.html5canvastutorials.com/tutorials/html5-canvas-image-loader/>
- Example: use a counter when loading two images; then draw the images



10.2 Multiple images

- The general structure of the JavaScript program is as follows:



```
// some variables, including the context and image variables  
  
// render function  
  
// function to check all images are loaded before calling render  
  
// function to load the images  
  
// function to initialise everything  
  
// main program body
```

10.2 Multiple images



```
// some variables, including the context and image variables
var canvas = null;
var context = null;
const WIDTH, HEIGHT;

// render function
// function to check all images are loaded before calling render
// function to load the images

// function to initialise everything
function init() {
    canvas = document.getElementById("example");
    context = canvas.getContext("2d");
    WIDTH = canvas.width;
    HEIGHT = canvas.height;  loadResources();
}

// main program body
init();
```

10.2 Multiple images

```
// some variables, including the context and image variables
var img1 = new Image();
var img2 = new Image();
var imageCount=0, NUM_IMAGES=2;

// render function

// function to check all images are loaded before calling render
function startInteraction() {
    imageCount++;
    if (imageCount == NUM_IMAGES)
        render();
}

// function to load the images
function loadResources() {
    img1.onload = startInteraction;
    img1.src = "image1.jpg";
    img2.onload = startInteraction;
    img2.src = "image2.jpg";
}
```



10.2 Multiple images

```
// some variables, including the context and image variables

// render function
function render() {
    context.drawImage(img1, WIDTH*0.1, HEIGHT*0.1);
    context.drawImage(img2, WIDTH*0.5, HEIGHT*0.5);
}

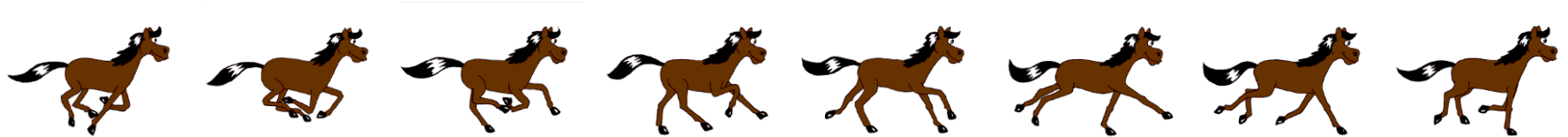
// function to check all images are loaded before calling render
function startInteraction() {
    imageCount++;
    if (imageCount == NUM_IMAGES)
        render();
}

// function to load the images
// function to initialise everything
// main program body
```



11. Animation

- General idea is to repeat the following:
 - **Draw** a scene, i.e. a frame of animation – a collection of things such as lines, shapes and images, etc
 - **Update** something in the scene, e.g. change a colour, move a shape, alter an image, etc
- Need to repeat this process fast enough to fool the eye into thinking there is continuous movement



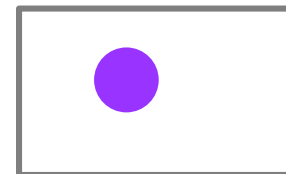
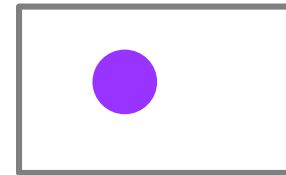
Animated horse, made by rotoscoping 19th century photos by Eadweard Muybridge. Artistic license has been used to achieve the cartoony look. Animation by Jan-Eric Nyström, Helsinki, Finland. <http://en.wikipedia.org/wiki/File:Animhorse.gif>

11.1 Basic steps

- **Clear the canvas** (or part of the canvas)
 - use `clearRect`
- *If necessary*, save the current canvas state
 - E.g. attributes such as current stroke colour, current transformation, etc.
 - May need to make some changes for the next frame of animation, and revert back to old settings after this frame
- **Draw this frame**
 - including any new settings for attributes
- *If necessary*, restore canvas state



→ Save
'state'



← Restore
'state'

11.2 Controlling the animation

- We need a way to keep calling the drawing routines
- We could keep calling them in a loop, but we need some way to control the amount of time taken for each frame
- Instead use a 'callback function':
 - `var intervalID = setInterval(render, 500)`
 - This executes the render function every 500 milliseconds
 - `clearInterval(intervalID)` cancels the repeated action
- Other:
 - Specify that something happens at some point in the future:
 - `setTimeout(render, 500)`
 - Only calls the method once after 500 milliseconds have elapsed

11.2 Controlling the animation

- More recent way to deal with this is to use `requestAnimationFrame()`:

```
function nextFrame() {  
    requestID = requestAnimationFrame(nextFrame);  
    render();  
}  
  
nextFrame();
```

- This automatically attempts to run the animation at 60 frames per second in the Web browser.
- It produces smoother animation
- The animation pauses when the browser tab is not the current display tab (unlike `setInterval`)

11.2 Controlling the animation

- Can be used in conjunction with `setTimeout` to control the speed of the animation

```
function nextFrame() {  
    setTimeout(function() {  
        requestID = requestAnimationFrame(nextFrame);  
        render();  
    }, 1000);    // render one frame every second  
}  
  
nextFrame();
```

11.3 Example: a bouncing ball

```
// some variables, including the context and image variables
const WIDTH, HEIGHT;
var canvas = null, context = null, intervalId;

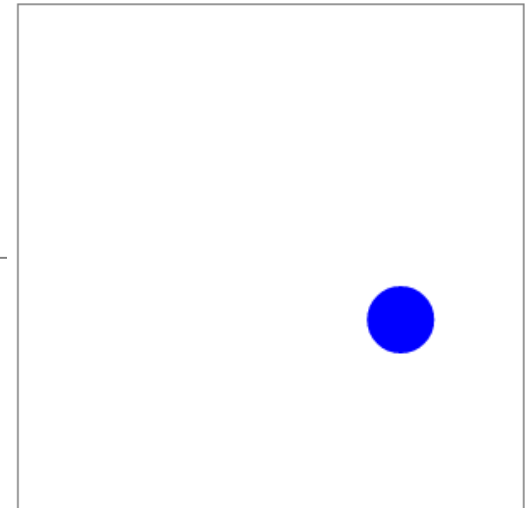
function nextFrame() {
    requestID = requestAnimationFrame(nextFrame);
    render();
}

function init() {
    var canvas = document.getElementById("canvas_example");
    context = canvas.getContext("2d");
    WIDTH = canvas.width;
    HEIGHT = canvas.height;
    initBall();

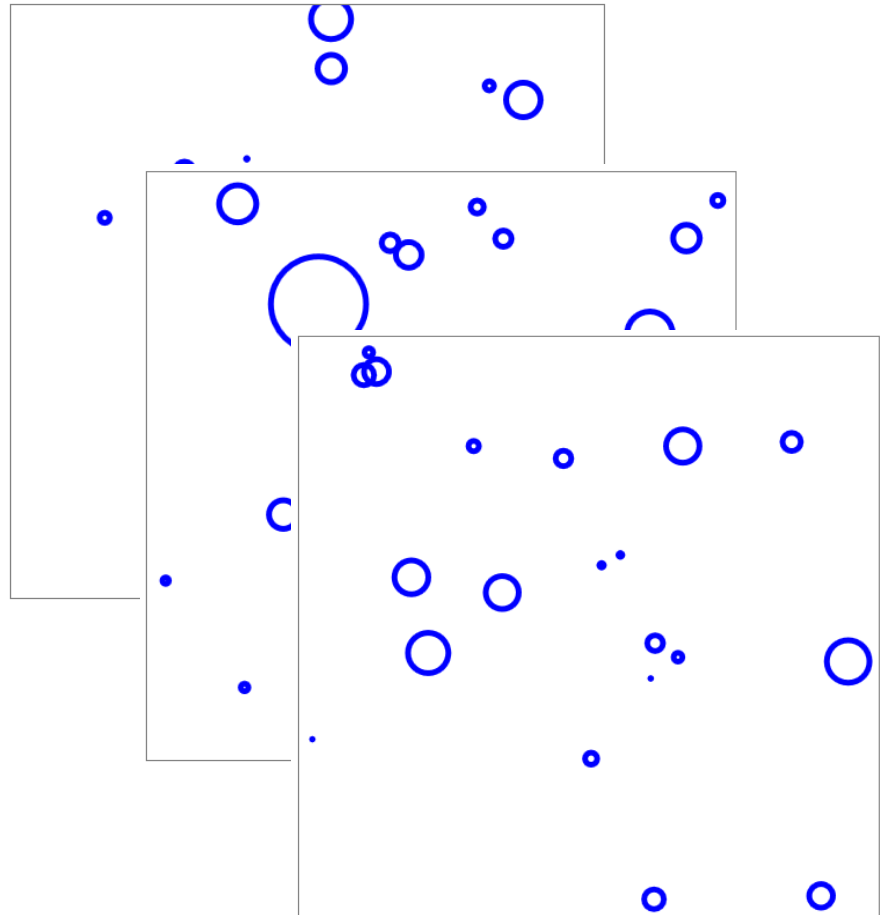
    <!-- start animation -->
    nextFrame();
}

// main program body
init();
```

```
function render() {  
  <!-- clear -->  
  clear()  
  
  <!-- draw -->  
  circle(x, y, radius, "rgb(0,0,255)");  
  
  <!-- update -->  
  if (x + dx > WIDTH || x + dx < 0) {  
    dx = -dx;  
  }  
  if (y + dy > HEIGHT || y + dy < 0) {  
    dy = -dy;  
  }  
  x += dx;  
  y += dy;  
}
```



11.4 Examples: Loads of balls



12. Summary

- We make a distinction between raster graphics and vector graphics
- Raster graphics
 - Screen image is made of a rectangular grid of pixels – bitmap
 - HTML5 canvas element
- Vector graphics – see next week
 - Image is a list of geometric shapes
 - SVG is W3C's recommended markup language for vector graphics
- Animation
 - Repeat { draw, update } fast enough so the eye perceives continuous motion
- *Next:* interaction on the canvas...

Further work

- Some recommended canvas work:
 - <http://hakim.se/projects>
 - mrdoob.com
- Video
 - <http://html5doctor.com/video-canvas-magic/>
- 3D Computer Graphics
 - https://developer.mozilla.org/en-US/docs/Web/WebGL/Getting_started_with_WebGL