Drawing images

INTRODUCTION

Load images in the background, wait for them to be loaded before drawing!

Working with images is rather simple, except that we need the images to be fully loaded into memory before drawing them. Loading images is an *asynchronous* process we need to take care of. Working with multiple images might also be difficult for beginners. Later on in this course, we will present a multiple image loader.

It is also possible to draw images from a video stream, images corresponding to another canvas content, or images that are defined by `<img>` HTML elements in the page. We will see that as well in the following parts of this chapter.

But let's start with a basic example!

EXAMPLE 1: DRAWING AN IMAGE

Try this example online: http://jsbin.com/wifeka/1/edit

Source code:

```
<!DOCTYPE HTML>
<html>
<head>
  <script>
    window.onload = function() {
      // Necessity to run this code only after the web page has been loaded.
      var canvas = document.getElementById("myCanvas");
```

```
        var context = canvas.getContext("2d");
9.
        var imageObj = new Image();
        // Callback function called by the imageObj.src = .... line
        //located after this function
        imageObj.onload = function() {
          // Draw the image only when we have the guarantee
          // that it has been loaded
            context.drawImage(imageObj, 0, 0);
        };

        // Calls the imageObj.onload function asynchronously
        imageObj.src =
         "http://www.w3.org/html/logo/downloads/HTML5_Logo_512.png";
23.   };
    </script>
  </head>
  <body>
    <canvas id="myCanvas" width="512" height="512"></canvas>
  </body>
  </html>
```

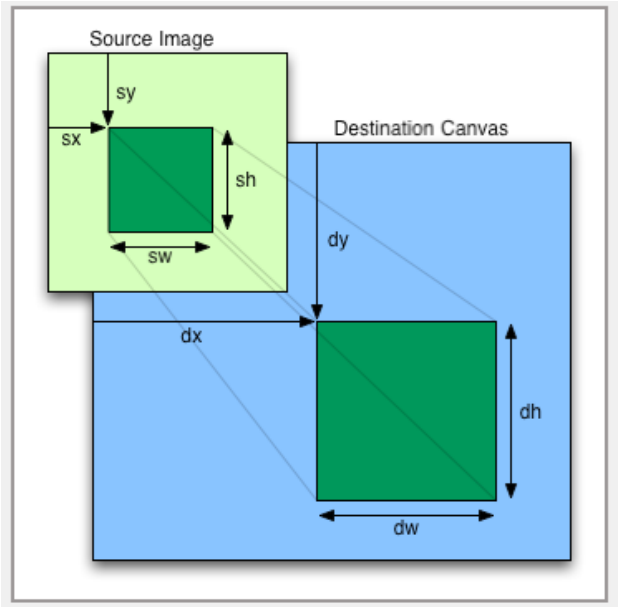
Several things need to be explained here:

1. We have to **create a JavaScript Image object** (line 10),
2. When we set the `src` attribute of this object with the URL of the image file, then**an asynchronous request is sent in the background by the browser**. Loading a big image may take some time, so the rest of the JavaScript code continues running. This is why we call it "asynchronous".
3. When the image file has been loaded, **the browser calls the `onload` callback associated with the image** (line 14).
4. **We draw the image only from inside this callback**, otherwise we have no guarantee that the image has been loaded and can be usable. The actual drawing here is done line 17.

There are numerous variants of the `drawImage(...)` context method at line 17:

- `drawImage(img, x, y)`: draws the image at position x, y, keeping the original image size.
- `drawImage(img, x, y, sizeX, sizeY)`: same as before except that the image drawn is resized.
- `drawImage(img, sx, sy, sw, sh, dx, dy, dw, dh)`: for drawing sub-images, (sx, sy, sw, sh) define the source rectangle, while dx, dy, dw, sh define the target rectangle. If these rectangles don't have the same size, the source sub-image is resized.

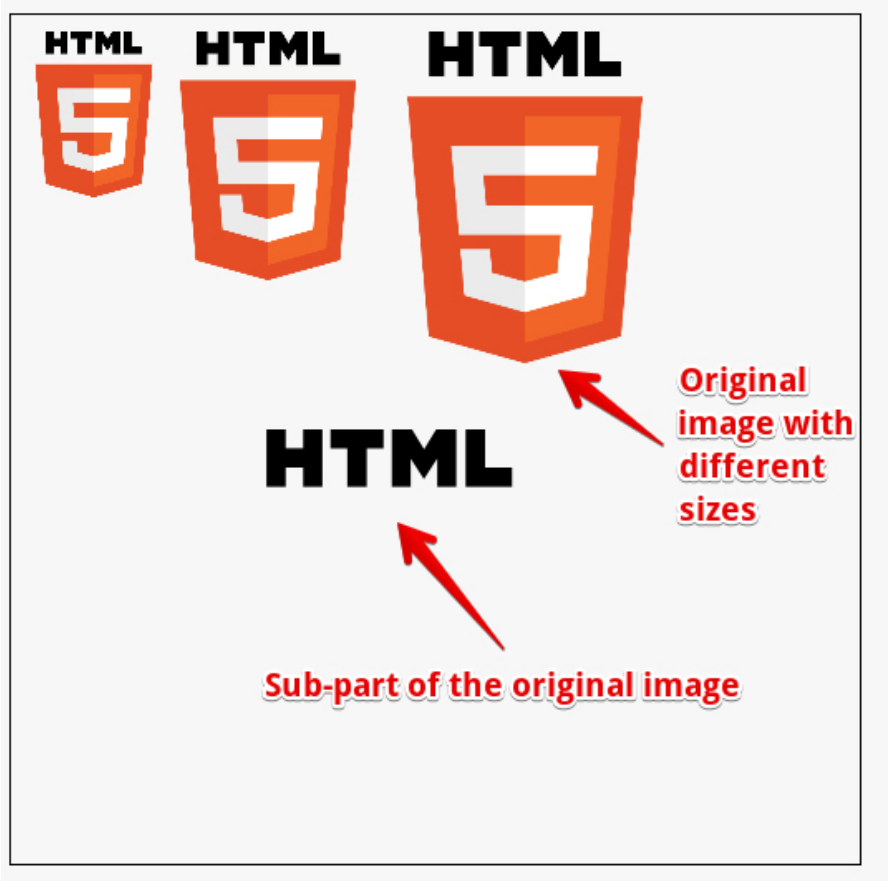
See picture below :



Reference: www.whatwg.org

EXAMPLE 2: SHOW THE DIFFERENT VARIANTS OF `DRAWIMAGE(...)`

Online example: http://jsbin.com/pusafe/2/edit



Source code extract:

```
 var imageObj = new Image();

imageObj.onload = function() {
  // Try commenting/uncommenting the following lines to see the
  // effect of the different drawImage variants
  // Original, big image
  // context.drawImage(imageObj, 0, 10);
  // Original image drawn with size = 100x100 pixels
  context.drawImage(imageObj, 0, 10, 100, 100);
  // with size = 150x150
  context.drawImage(imageObj, 80, 10, 150, 150);
  // with size = 200x200
  context.drawImage(imageObj, 210, 10, 200, 200);

 // draw the sub image at 0, 0, width = 512, height = 100
 // at position 100, 250, with a width of 256 and a height of 50
 context.drawImage(imageObj, 0, 0, 512, 100, 100, 250, 256, 50);
};
imageObj.src ="http://www.w3.org/html/logo/downloads/HTML5_Logo_512.png";
};
```

EXAMPLE 3: DRAW AN IMAGE DEFINED IN THE PAGE BY AN `<IMG SRC="...">` ELEMENT

Sometimes, you may want to draw an image that is already declared in the HTML document as an `<img src="...">` element. Remember that when you add an `<img>` in the document, the browser starts downloading it in background.

WRONG => indeed, you could try drawing it using some code like this:

```
<script>
 window.onload = function() {
   var canvas = document.getElementById("myCanvas");
   var ctx = canvas.getContext("2d");
   var logo = document.querySelector("#logo");
   ctx.drawImage(logo, 0, 0, 100, 100);
};
</script>
<body>
 <canvas id="myCanvas" width="512" height="512"></canvas>
 <p>Original image as an <img> element:
 </p>
 <img id="logo"
   src="http://www.w3.org/html/logo/downloads/HTML5_Logo_512.png">
```

```
</body>
```

Although you will find many examples on the Web that do it this way, they will only work most of the time with small images, or with images that are in the browser's cache. Remember that **you cannot draw an image that has not been fully loaded!**

If you try to draw an image that is not loaded or partially loaded, you will have unexpected results!

> ***Best practice:*** *only draw an image that is fully loaded, use the* `onload` *callback or test the* `complete` *property of the image object!*

GOOD => the right way to do that is shown in this example: http://jsbin.com/faqedu/1/edit

```
<!DOCTYPE HTML>
<html>
 <head>
 <script>
  var canvas, context;
  window.onload = function() {
    canvas = document.getElementById("myCanvas");
    context = canvas.getContext("2d");

    draw();
  };
  function draw() {
    var imageObj = document.querySelector("#logo");
    if(imageObj.complete) {
      console.log("image is already loaded, we draw it!");
      context.drawImage(imageObj, 0, 10, 100, 100);
      ...
    } else {
       console.log("image not loaded, trying again in 100ms");
       setTimeOut(draw, 100);
    }
 }
 </script>
 </head>
 <body>
 <canvas id="myCanvas" width="512" height="512"></canvas>
 </p>
 <img id="logo"src="http://www.w3.org/html/logo/downloads/HTML5_Logo_512.png">
 </body>
</html>
```

Line 17 tests if the image is completely loaded in memory. If this is not the case, the`draw()` function is called again 100ms later (line 31-32), using `setTimeout(function, delay);`

With large image files, this will not break nor produce unexpected results: http://jsbin.com/mayoma/3/edit

Results with a very large image (5160x3270 pixels):

Original image as an <img> element:

KNOWLEDGE CHECK 3.3.3 (NOT GRADED)

In this page, we both mentioned an "onload callback" and the "complete" property of images... why?

- For checking that an image has been loaded entirely before we try to draw it in a canvas.
- It's just a recommendation, checking if an image is loaded is generally not necessary before trying to draw it, as it will appear anyway line by line as image data arrive in the browser.