Immediate drawing mode vs path drawing mode

IMMEDIATE MODE = EXECUTING A CALL TO A DRAWING METHOD MEANS IMMEDIATELY DRAWING IN THE CANVAS

In the previous examples, we saw how to draw some rectangles using the `fillRect(x, y, width, height)` and `strokeRect(x, y, width, height)` methods of the context.

We also learned how to draw a text message using the `fillText(message, x, y)`method. We did not use it, but there is a sister method called `strokeText(message, x, y)` that draws a text in wireframe mode.

These methods, along with the `drawImage(...)` method we will see a bit further in this course, are *"immediate methods"*: as soon as they are executed, the results are displayed on screen, the drawings are performed, pixels on the canvas area change their colors, etc.

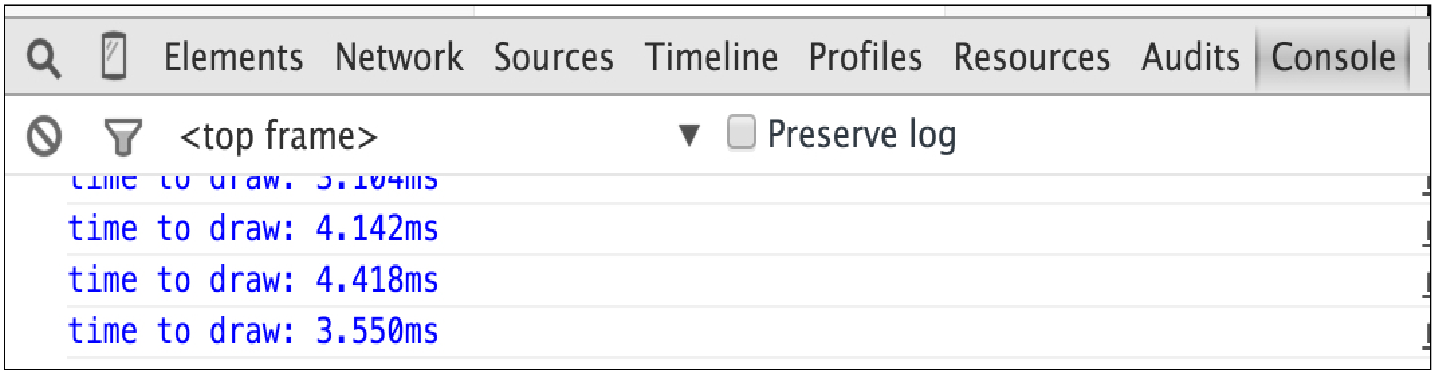
Here is a code extract that will draw 1000 random rectangles in a canvas, using immediate mode rectangle drawing calls.

Online example: http://jsbin.com/zanizu/3/edit

```
var canvas, ctx, w, h;
function init() {
   canvas = document.getElementById('myCanvas');
   ctx = canvas.getContext('2d');
   w = canvas.width;
   h = canvas.height;
   console.time("time to draw");
   for(var i=0; i < 1000; i++) {
      var x = Math.random() * w;
      var y = Math.random() * h;
      var width = Math.random() * w;
      var height = Math.random() * h;
      ctx.strokeRect(x, y, width, height);
   }
   console.timeEnd("time to draw");
}
```

Lines 12-18 draw 1000 rectangles of random sizes in immediate mode. We also measure the time using the usual `console.time(name_of_timer)` and`console.timeEnd(name_of_timer)` that will write in the browser console the time elapsed. Note that `console.time(...)` and `console.timeEnd(...)` display results only in the browser's console, not in the JS Bin console.

On a Mac Book Pro from 2015, the result is:



```
time to draw: 3.104ms
time to draw: 4.142ms
time to draw: 4.418ms
time to draw: 3.550ms
```

PATH MODE = FILL A BUFFER THEN EXECUTE ALL BUFFERED ORDERS AT ONCE TO ENABLE OPTIMIZATION AND PARALLELISM

There is another drawing mode called "path drawing mode" where you first send drawing orders to the graphics processor, and these orders are stored in a buffer. Then you call methods to draw the whole buffer at once. There are also methods to erase the buffer's content.

Path drawing mode allows parallelism: if you need to draw 10.000 rectangles, it's better to store the orders in the graphics card, then execute the drawing all at once, rather than doing 10.000 immediate calls to `strokeRect(...)` for example. With the buffered mode, the Graphic Processing Unit (GPU) of the graphics card hardware will be able to parallelize the computations (modern graphics cards can execute hundreds/thousands of things in parallel).

Same example as before, this time using the buffered mode for drawing rectangles: http://jsbin.com/zanizu/5/edit

Extract from source code (the part that draws the rectangles):

```
for(var i=0; i < 1000; i++) {
    var x = Math.random() * w;
    var y = Math.random() * h;
    var width = Math.random() * w;
```

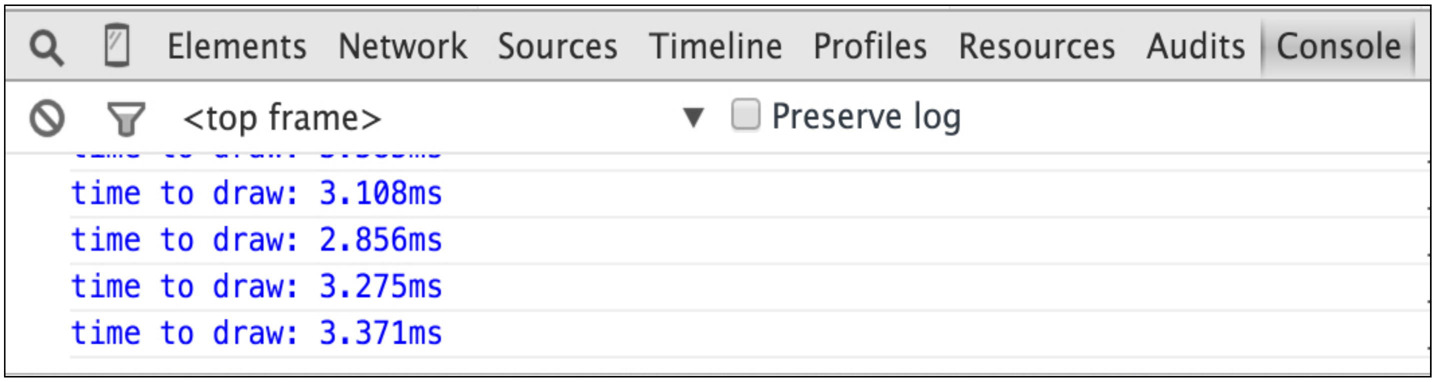
```
      var height = Math.random() * h;
     ctx.rect(x, y, width, height); // store a rectangle in path/buffer
  }
  ctx.stroke(); // draws the whole buffer (the 1000 rectangles) at once
```

Instead of calling `strokeRect(...)` or `fillRect(...)`, we just call the `rect(...)`method of the context (line 7). This is how we can delay the drawing of the rectangles. The 1000 rectangles are stored in a buffer in the hardware.

The call to `ctx.stroke()` (line 9) or to its sister method `ctx.fill()` will draw the entire buffer contents in fill or stroke mode.

And here is what the timer gives: a slightly faster execution time. Changing 1000 to 100.000 will give even larger differences.

Path mode is faster than immediate mode!



```
time to draw: 3.108ms
time to draw: 2.856ms
time to draw: 3.275ms
time to draw: 3.371ms
```

RESET THE PATH MODE BUFFER

A call to `ctx.beginPath()` will reset the buffer (empty its contents). We will see many more examples of using the Path drawing mode in another section later on this week.

```
 // start a new buffer / path
ctx.beginPath();
 // all these orders are in a buffer/path
ctx.moveTo(10, 10);
ctx.lineTo(100, 100);
ctx.lineTo(150, 70);
 // Draw the buffer
ctx.stroke();
```

SUMMARY OF PATH MODE PRINCIPLES

1. **Call drawing methods that work in path mode**, for example call `ctx.rect(...)` instead of `ctx.strokeRect(...)` or `ctx.fillRect(...)`
2. **Call `ctx.stroke()` or `ctx.fill()` to draw the buffer's contents,**
3. **Beware that the buffer is never emptied,** two consecutive calls to `ctx.stroke()` will draw the buffer contents twice! Instead, use `ctx.beginPath()` to empty it if needed.
4. It is possible to empty the buffer by calling `ctx.beginPath()`.
5. Path drawing is faster than immediate drawing (parallelization is possible).

KNOWLEDGE CHECK 3.4.1 (NOT GRADED)

The path drawing mode...

- ○ Enables parallelization by storing drawing instructions in a buffer of the graphics card.
- ○ Draws shapes that compose a path immediately after each instruction is executed.