Time-based animation

INTRODUCTION

Let's study an important technique known as *"time-based animation"*, that is used by nearly all "real" video games.

This technique is useful when:

- **Your application runs on different devices, and where 60 frames/s are definitely not possible. More generally, you want your animated objects to move at the same speed on screen, regardless of the device that runs the game.**

  For example, imagine a game or an animation running on a smartphone and on a desktop computer with a powerful GPU. On the phone, you might achieve a maximum of 20 frames/s with no guarantee that this number will be constant, and on the desktop, you will reliably achieve 60 frames/s. If your application is a race game, for example, your car will take 30s to make a complete loop on the race track, while on your smartphone it will take 5 minutes.

  The way to address this is to have fewer frames/s on the phone, enabling the car to take the same time to race around the track as it does on a powerful desktop computer.

  Solution: you need to compute the time elapsed between the last frame that has been drawn and the current one, and depending on this delta of time, adjust the distance you must move your car on the screen. We will see several examples of this later.
- **You want to perform some animations only a few times per second.** For example, in sprite-based animation (drawing different images as a character moves, for example), you will not change the images of the animation 60 times/s, but only ten times per second. Mario will walk on the screen in a 60 f/s animation, but his posture will not change every 1/60th of second.
- **You may also want to accurately set the framerate**, leaving some CPU time for other tasks. Many games on console limit the*framerate* to 1/30th of a second to allow time for other sorts of computations (physic engine, artificial intelligence, etc.)

HOW TO MEASURE TIME WHEN WE USE REQUESTANIMATIONFRAME?

Let's take a simple example with a small rectangle that goes from left to right. At each

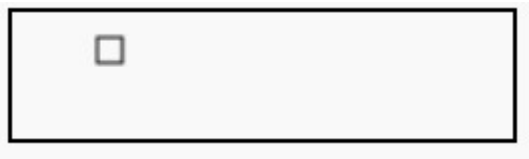
animation loop, we erase the canvas content, we move the rectangle, we draw the rectangle and we call the animation loop again. So you animate a shape as follows (note: steps 2 and 3 can be swapped):

1. erase the canvas,
2. draw the shapes,
3. move the shapes,
4. go to step 1.

When we use requestAnimationFrame for implementing such an animation, as we did in the previous lessons, the browser tries to keep the framerate at 60 frames/s, meaning that the ideal time between frames will be 1/60 = 16.66 ms.

Naive example that does not use time-based animation

Online example at JSBin



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset=utf-8 />
  <title>Small animation example</title>
  <script>
    var canvas, ctx;
    var width, height;
    var x, y;
    var speedX;
    // Called after the DOM is ready (page loaded)
    function init() {
       // init the different variables
       canvas =document.querySelector("#mycanvas");
       ctx = canvas.getContext('2d');
       width = canvas.width;
       height = canvas.height;
       x=10; y = 10;
```

```
        // Move 3 pixels left or right at each frame
        speedX = 3;
        // Start animation
        animationLoop();
    }
    function animationLoop() {
        // an animation involves: 1) clear canvas and 2) draw shapes,
        // 3) move shapes, 4) recall the loop with requestAnimationFrame
        // clear canvas
        ctx.clearRect(0, 0, width, height);
        ctx.strokeRect(x, y, 10, 10);
        // move rectangle
        x += speedX;

        // check collision on left or right
        if(((x+5) > width) || (x <= 0)) {
            // cancel move + inverse speed
            x -= speedX;
            speedX = -speedX;
        }
        // animate.
        requestAnimationFrame(animationLoop);
    }
 </script>
</head>
<body onload="init();">
 <canvas id="mycanvas" width="200"height="50" style="border: 2px solid black">
 </canvas>
</body>
</html>
```

If you try this example on a low end smartphone (use this URL for the example in stand alone mode: http://jsbin.com/dibuze), and if you run it at the same time on a desktop PC you will certainly notice that the rectangle moves faster on the desktop computer screen than on your phone screen.

This is because the frame rate differs between the computer and the smartphone: perhaps 60 frames/s on the computer and 25 frames/s on the phone. As we only move the rectangle in

the`animationLoop`, in 1s the rectangle will be moved 25 times on the smartphone compared with 60 times on the computer! Since we moved it the same amount of pixels each time, the rectangle moves faster on the computer!

EXAMPLE THAT FAKE WHAT WE WOULD EXPERIENCE ON A LOW END DEVICE

Here is the same example in which we added a loop that takes time to execute, right in the middle of the animation loop, that will artificially increase the time spent to draw in the animation loop, making the 1/60th of second impossible to reach.

Try it on JsBin and notice that the square moves much slower on the screen. Indeed, its speed depends ont the time spent in the animation loop.

```
function animationLoop() {
    ...
    for(var i = 0; i < 50000000; i++) {
       // slow down artificially the animation
    }
    ...
    requestAnimationFrame(animationLoop);
}
```