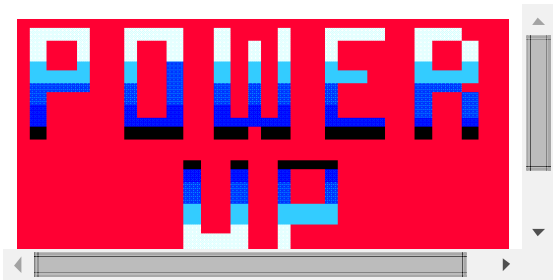


`requestAnimationFrame()` - 60 frames/second animation (best practice)!



INTRODUCTION: ADVANTAGES OVER PREVIOUS METHODS

The new `requestAnimationFrame(animationLoop)` is very similar to `setTimeout`:

- **It targets 60 frames/s:** `requestAnimationFrame` asks the browser to schedule a call to the `animationLoop` function passed as parameter in 1/60th of a second (equivalent to 16.6ms). Keep in mind that most monitors cannot display more than 60 frames per second (FPS). Note that whether humans can tell the difference among high frame rates depends on the application, most games are acceptable above 30 FPS, and virtual reality might require 75 FPS so to give a natural feel. Some gaming monitors go up to 144 FPS (pro players in e-sport train themselves at Counter Strike with a 150 frames/s rate).
- **It calls the function only ONCE**, so if you want a continuous animation, like with `setTimeout`, you need to call again `requestAnimationFrame` at the end of the `animationLoop` function.

It has however several advantages over `setInterval` and `setTimeout`:

- **The scheduling is much more accurate:** if the code inside the function can be executed in less than 16.6ms, then the average error between the scheduled time and the real time will be much smaller than with the old functions.
- **High resolution timer:** even if this difference is small, the function that is called after 16.6ms has an extra parameter that is a high resolution time, very useful for writing games that do [time-based animation](#). Time-based animation will be studied in detail in the HTML5 Part 2 course at W3Cx. It is a technique that consists in measuring the amount of time elapsed between two frames, then in computing the distance in pixels to move objects on screen so that the visible speed for a human eye remains constant, even if the frame rate is not.
- **Multiple animations are merged:** browsers can bundle animations happening at the same time into a single paint redraw (thus happening faster/with less CPU cycles), solving

the problems that can occur with simultaneous `setInterval` calls.

- **CPU/GPU optimization, battery saved on mobiles:** if the JavaScript execution is occurring in a tab/window which is not visible it doesn't have to be drawn. However the animation loop is still executed (objects will be moved, not drawn). This is the same when a mobile phone or tablet screen is black or if the application is put in background.

TYPICAL USE

You will note that `requestAnimationFrame(function)` is used like `setTimeout(function, delay)`. A call to `requestAnimationFrame` just asks the browser to call the function passed as a parameter **ONCE, and the target delay is fixed**, and corresponds to a 60 frames/s frame rate (16.6ms). Notice that an `id` is used for stopping an animation with `cancelAnimationFrame(id)`.

Source code:

```
<body onload="init();">
<script>
var canvas, ctx;
function init() {
    // This function is called after the page is loaded
    // 1 - Get the canvas
    canvas = document.getElementById('myCanvas');
    // 2 - Get the context
10.   ctx=canvas.getContext('2d');
    // 3 - start the animation
    startAnimation();
}

var id;
function animationLoop(timestamp) {
    // 1 - Clear
    ctx.clearRect(0, 0, canvas.width, canvas.height);
20.   // 2 Draw
    drawShapes(...);
    // 3 Move
    moveShapes(...);
    // call again mainloop after 16.6ms (corresponds to 60 frames/second)
    id = requestAnimationFrame(animationLoop);
}
```

```
30. function startAnimation() {  
    id = requestAnimationFrame(animationLoop);  
}  
function stopAnimation() {  
    if (id) {  
        cancelAnimationFrame(id);  
    }  
}  
</script>  
</body>
```

EXAMPLE: ANIMATE THE MONSTER WITH `requestAnimationFrame`

Online example at: <http://jsbin.com/jixuju/4/edit>



Extract from source code, compare to the previous example that used `setInterval()`

```

function animationLoop(timestamp) {
  // 1 - Clear
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // 2 - Draw
  drawMonster(monsterX, monsterY, monsterAngle, 'green', 'yellow');

  // 3 - Move
  monsterX += 10;
10.  monsterX %= canvas.width
  monsterAngle += 0.01;

  // call again mainloop after 16.6 ms (60 frames/s)
  requestId = requestAnimationFrame(animationLoop);
}
function start() {
  // Start the animation loop, targets 60 frames/s, this calls animationLoop
  // only ONCE!
  requestId = requestAnimationFrame(animationLoop);
}
21.
22. function stop() {
    if (requestId) {
      cancelAnimationFrame(requestId);
    }
  }
}

```

Notice that calling `requestAnimationFrame(mainloop)` at line 24, and after that from within the loop at line 14, asks the browser to call the `animationLoop` function so that the delta between calls will be **as close as possible to 16.6ms (this corresponds to 1/60th of a second)**.

IS THE 16.6MS DELAY REALLY ACCURATE? CAN WE TRUST IT?

This target may be hard to reach if:

- The animation loop content is too complex,
- The target device that runs the animation is a low end phone or an old computer,
- The scheduler may be a bit late or a bit in advance (even if this kind of error is much

smaller with `requestAnimationFrame` than with `setInterval` or `setTimeout`).

Many HTML5 games perform what we call a "time-based animation". For this, we need an accurate timer that will tell us the elapsed time between each animation frame.


Depending on this time, we can compute the distances each object on the screen must achieve in order to move at a constant speed (for a human eye), independently of the CPU or GPU of the computer or mobile device that is running the game.

The `timeStamp` parameter of the `animationLoop` function (line 1 in the above code) is useful exactly for that: it gives a high resolution time. By measuring deltas between two consecutive calls of the `animationLoop`, we will know exactly, with a sub-millisecond accuracy, the exact elapsed time between two frames.

Using time-based animation, and more generally, using the canvas element for writing HTML5 games, will be a chapter of the HTML5 Part-2 course to come.

CURRENT SUPPORT

Current support (as in April 2015) is really good and all modern browsers support this API.

requestAnimationFrame  - CR								
API allowing a more efficient way of running script-based animation, compared to traditional methods using timeouts.								Global 86.54%
								unprefixed: 85.38%
<div>Current aligned Usage relative Show all</div>								
IE	Firefox	Chrome	Safari	Opera	iOS Safari*	Opera Mini*	Android Browser*	Chrome for Android
		31						
		36						
		37					4.1	
8	31	38					4.3	
9	35	39	7				4.4	
10	36	40	7.1		7.1		4.4.4	
11	37	41	8	27	8.3	8	40	41
TP	38	42		28				
	39	43		29				
	40	44						

Up-to-date version of this table +

details: <http://caniuse.com/#search=requestAnimationFrame>.

KNOWLEDGE CHECK 4.2.5 (NOT GRADED)

```
requestAnimationFrame(function);
```

`requestAnimationFrame` asks the browser to call the function passed as a parameter only once, and tries to call it after 16.6ms:

- ☐
 - ☐ Yes, this is true
 - ☐ No, it will call the function automatically every 16.6ms, resulting in a 60 frames/second smooth animation. It works like `setInterval`, but is more efficient.