Javascript Animation: Tutorial, Part 1

You're reading one of a three-part series on Javascript animation. Also see part 2: Smooth javascript animations, and part 3: Creating motion tweens in javascript.

Animation Theory - The Basics

Fireworks.js

A javascript animation experiment

What's this all about? (AKA, "Holy pointless lightshow,

Fireworks.js is a bit of Javascript that creates starburst-type explosions in
on, who doesn't want something like that? As far as appropriateness is co
so popular on the web in 1997. Exploding firework animations are hot, the

Nonetheless, Fireworks has been published here for fun, experimenting a
object-oriented code or script-driven sound. It also serves as a dirty brow
destroyed on this page as the script runs.

Fireworks.js: OO-JS animation

I have been refining a number of object-oriented javascript animationtechniques recently, so it seemed right to write some notes, explanations and comments about my approach to animation based on various findings made in previous years. Javascript/DHTML animation seems to be a rather niche topic but one of occasional interest, so I am hoping this will come in handy to someone, somewhere.

How does it work?

The idea behind Javascript-based animation is fairly simple; a number of DOM elements (`<img />`,`<div>` or otherwise) are moved around the page according to some sort of pattern determined by a logical equation or function.

To achieve the effect of animation, elements must be moved at a given interval or frame rate; from a programming perspective, the simplest way to do this is to set up an animation loop with a delay. Javascript doesn't have a "pause" feature, unlike some earlier higher-level languages such as Basic (later QuickBasic and Visual Basic) which had directives such as `sleep n`, n being a number of ticks/"jiffies" or miliseconds.

There is no way in JS-land to specify an inline delay in this fashion:

```
doSomething();
delay(500); // pause execution for 500 msec
doSomethingElse();
```

So we are left instead with `setTimeout` and `setInterval`, which when called as `setTimeout(c,t)`and `setInterval(c,t)` allow for execution of javascript code *c* at *t* miliseconds from now in the case of `setTimeout`, and *every t miliseoncds* in the case of setInterval.

```
setTimeout(doSomething,500); // calls doSomething() 500 miliseconds
from now
```

```
setInterval(doSomething,500); // calls doSomething() every 500
miliseconds until stopped
```

The code that is called can be a reference (ie. passing a function object directly) or an inline statement which is evaluated, such as `setTimeout("alert('ding')",1000);` - in object-oriented cases though, it is useful to take advantage of references to maintain scope when the code is executed.

Basic animation

Let's say that we have an object called `foo` which refers to a `<div>` element; we are going to move this with a function that is called every 20 msec via `setTimeout`*. This object is within a function called `doMove`.

```
function doMove() {
foo.style.left = (foo.style.left+10)+'px'; // pseudo-property code:
Move right by 10px
}

doMove(); // animate the object
```

This will move `foo` 10 pixels to the right of its current position. Not bad, but this is only one frame. You may think a `for` loop would work for animation, to call this function 100 times for example, expecting to see `foo` to slowly move 1000px to the right - in fact, you would see two "frames" here: the before (at 0px), and after (1000px) because the browser is able to skip the display of items during the loop (presumably for efficiency, or because it cannot keep up with that rate.) Therefore, you need a delay. A recursive function works well here. *"Move the object, and then call this function again 20 msec from now."*:

```
function doMove() {
foo.style.left = (foo.style.left+10)+'px'; // pseudo-property code:
Move right by 10px
setTimeout(doMove,20); // call doMove() in 20 msec
}
```

.. And there is the basic theory behind Javascript-based animation.

View the functional demo animation code.

Next up in Javascript Animation, Part II: Efficiency considerations.

* A 20 msec delay does not necessarily infer the call (and thus a frame/movement) is made every 20 miliseconds, but rather 20 miliseconds *from the time the call was made* - and in this case, the action of moving a DOM node generally has the most potential to cause the biggest delays between calls (and therefore, the slowest frame rate.)

Related links

- Javascript Animation Demo 1: setTimeout method
- Javascript Animation: Tutorial, Part 2
- Javascript fireworks effect API

- Lots of Javascript animation effects: scottschiller.com