1. Tại sao lại cần chúng?

Trước đây, nếu người lập trình web muốn đặt video vào trang của họ, họ phải sử dụng đến <object>. Do sự khác biệt của trình duyệt, họ phải dùng đến các phần tử <embed> và lặp lại nhiều thông số. Điều này dẫn đến một đoạn mã rắc rối như thế này:

*<object width=”425” height=”344”>*

*<param name=”movie” value=”http://www.youtube.com/9sEI1AUFJKw&hl=en\_GB&fs=1&”></param>*

*<param name=”allowFullScreen”*

*value=”true”></param>*

*<param name=”allowscriptaccess”*

*value=”always”></param>*

*<embed src=”http://www.youtube.com/9sEI1AUFJKw&hl=en\_GB&fs=1&”*

*type=”application/x-shockwave-flash”*

*allowscriptaccess=”always”*

*allowfullscreen=”true” width=”425”*

*height=”344”></embed>*

*</object>*

Như trên, chúng ta có thể thấy đoạn mã đó khá là “xấu xí” và “phức tạp”. Tệ hơn nữa, thực tế là các trình duyệt vượt qua nhiều điều nữa, đó là người dùng có cài plugin video hay chưa, rồi phải hy vọng là plugin đó đúng phiên bản để nó có thể làm việc tốt. Mà không phải người dùng nào cũng có kiến thức đó (lựa chọn đúng plugin, tải plugin về và cài đặt,…), rồi hàng loạt các rắc rối khác kéo theo điển hình như người dùng phải thường xuyên cập nhật plugin, quá nhiều rào cản cho đến khi trình duyệt có thể chạy được một video. Thật rắc rối và phức tạp cho cả người lập trình lẫn người sử dụng.

Một trong những lợi thế chính của HTML5 là phần tử video, cuối cùng video là một thành phần chính thức trên trang web. Không cần phần tử <object> đại diện, cũng không cần các yếu tố <embed> nữa.

Khả năng hỗ trợ đa phương tiện trong HTML5 được hiện thực qua việc bổ sung hai thẻ mới là <video> và <audio>. Hãy thử tưởng tượng đây là tất cả những gì cần làm để thêm một đoạn phim vào trang web:

*<video src=“myVideo.mp4”></video>*

Vì vậy, phần tử <video> có thể định dạng theo kiểu CSS. Chúng có thể được thay đổi kích cỡ bằng cách chuyển đổi CSS. Chúng cũng có thể được tinh chỉnh và hiển thị lại vào <canvas> với JavaScript.

Các thẻ video và audio của HTML5 được thiết kế để làm cho việc nhúng một video dễ dàng như việc nhúng một hình ảnh. Chúng cũng được thiết kế để cung cấp cho người dùng một trải nghiệm nhanh hơn so với các plugin như Adobe Flash.

1. Các thành phần đa phương tiện và phát triển game
2. Thẻ <audio>

Âm thanh là góp phần rất lớn tạo nên cái hay của các bộ phim cũng như các trò chơi. Hãy thử coi một bộ phim hay chơi một trò chơi mà tắt âm thanh đi, chúng ta sẽ nhận ra được vai trò quan trọng của nó.

1. Web Audio API.

Trước đây, Flash hoặc các plugin khác được sử dụng để tạo ra âm thanh trên các trang web. Từ khi tag <audio> HTML5 ra đời, âm thanh trên web không còn đòi hỏi plugin nữa, tuy nhiên tag <audio> cũng có những hạn chế đáng kể để thực hiện các trò chơi phức tạp và các ứng dụng tương tác.

Web Audio API là JavaScript API cấp cao để xử lý và tổng hợp âm thanh trong các ứng dụng web. Mục tiêu của API này là bao gồm khả năng nhầm thay thế các công cụ âm thanh hiện đại cũng như xử lý, pha trộn, và lọc các nhiệm vụ được tìm thấy trong các ứng dụng về âm thanh của máy tính hiện nay. Sau đây là một hướng dẫn cơ bản để sử dụng API mạnh mẽ này.

* Bắt đầu với AudioContext

AudioContext để quản lý và chơi tất cả các âm thanh. Tạo ra một âm thanh bằng cách sử dụng các trang Web Audio API nghĩa tạo ra một hoặc nhiều nguồn âm thanh và kết nối chúng đến đích âm thanh được cung cấp bởi thể hiện AudioContext. Kết nối này không cần trực tiếp, và có thể đi qua bất kỳ số lượng AudioNodes trung gian nào thứ mà thực hiện xử lý các mô-đun cho các tín hiệu âm thanh. Phần này sẽ được mô tả chi tiết hơn ở các đặc điểm kỹ thuật của Audio Web.

Một thể hiện AudioContext có thể hỗ trợ nhiều âm thanh đi vào và đồ thị âm thanh phức tạp, vì vậy chúng ta chỉ cần một thể hiện duy nhất cho mỗi ứng dụng âm thanh chúng ta tạo ra.

Sau đây là cách tạo ra một AudioContext:

var context;

window.addEventListener('load', init, false);

function init() {

try {

context = new webkitAudioContext();

}

catch(e) {

alert('Web Audio API is not supported in this browser');

}

}

* Tải âm thanh

Web Audio API sử dụng một AudioBuffer cho âm thanh có độ dài ngắn đến trung bình. Cách cơ bản là sử dụng XMLHttpRequest để lấy các tập tin âm thanh.

API hỗ trợ chạy các tập tin âm thanh ở định dạng khác nhau, chẳng hạn như WAV, MP3, AAC, OGG và một số định dạng khác. Các trình duyệt cũng có một vài khác biệt trong việc hỗ trợ các định dạng âm thanh khác nhau.

Đoạn sau đây là ví dụ cho việc tải một âm thanh:

var dogBarkingBuffer = null;

var context = new webkitAudioContext();

function loadDogSound(url) {

var request = new XMLHttpRequest();

request.open('GET', url, true);

request.responseType = 'arraybuffer';

*// Decode asynchronously*

request.onload = function() {

context.decodeAudioData(request.response, function(buffer) {

dogBarkingBuffer = buffer;

}, onError);

}

request.send();

}

Các tập tin âm thanh là dạng nhị phân (không phải văn bản), vì vậy chúng ta thiết lập các responseType là 'arraybuffer'.

Một khi dữ liệu tập tin âm thanh đã được nhận, nó có thể được giữ lại giải mã sau, hoặc nó có thể được giải mã ngay lập tức bằng cách sử dụng phương thức decodeAudioData() của AudioContext. Phương thức này lấy ArrayBuffer dữ liệu tập tin âm thanh được lưu trữ trong request.response và giải mã không đồng bộ.

Khi decodeAudioData () được hoàn thành, nó gọi một hàm callback thứ mà cung cấp các gải mã PCM dữ liệu âm thanh thành một AudioBuffer.

* Phát âm thanh



Một biểu đồ âm thanh đơn giản

Khi một hoặc nhiều AudioBuffers được nạp, sau đó chúng đã sẵn sàng để phát âm thanh. Giả sử một AudioBuffer với âm thanh của một con chó sủa đã tải hoàn tất. Sau đó, chúng ta có thể phát nó với mã sau đây.

var context = new webkitAudioContext();

function playSound(buffer) {

var source = context.createBufferSource(); *// creates a sound source*

source.buffer = buffer; *// tell the source which sound to play*

source.connect(context.destination); *// connect the source to the context's destination (the speakers)*

source.noteOn(0); *// play the source now*

}

Hàm playSound() có thể được gọi là mỗi khi có sự kiện từ bàn phím hay nhấp chuột.

Hàm noteOn() để dễ dàng sắp xếp phát lại âm thanh một cách chính xác. Tuy nhiên, để nó làm việc chính xác, cần phải đảm bảo rằng bộ đệm âm thanh đã được nạp sẵn.

Một lưu ý quan trọng nữa là đối với iOS, Apple hiện nay đã tắt tất cả âm thanh phát ra cho tới khi có sự tương tác của người dùng. Nghĩa là khi gọi hàm playSound() khi mới vào game có thể nó sẽ ko phát âm thanh, để khắc phục điều này, chúng ta có thể hướng người dùng vào sự tương tác sớm ví dụ như tạo 1 button “Chạm vào đây để chơi game”.

* Trừu tượng hóa

Tất nhiên, sẽ hay hơn nếu chúng ta tạo ra một hệ thống load tổng quát nhiều âm thanh để sử dụng sau này, đặc biệt là với game, có rất nhiều âm thanh cần quản lý. Một trong những cách giải quyết là sử dụng một lớp BufferLoader.

Sau đây là một ví dụ về cách sử dụng lớp BufferLoader. Trước tiên, tạo ra hai AudioBuffers, và ngay sau khi chúng được nạp, chúng ta có thể chơi chúng cùng một lúc.

window.onload = init;

var context;

var bufferLoader;

function init() {

context = new webkitAudioContext();

bufferLoader = new BufferLoader(

context,

[

'../sounds/hyper-reality/br-jam-loop.wav',

'../sounds/hyper-reality/laughter.wav',

],

finishedLoading

);

bufferLoader.load();

}

function finishedLoading(bufferList) {

*// Create two sources and play them both together.*

var source1 = context.createBufferSource();

var source2 = context.createBufferSource();

source1.buffer = bufferList[0];

source2.buffer = bufferList[1];

source1.connect(context.destination);

source2.connect(context.destination);

source1.noteOn(0);

source2.noteOn(0);

}

* Xử lý thời gian: phát âm thanh có nhịp điệu

Bằng việc xử lí thời gian, chúng ta có thể tạo ra 1 bài nhạc phức tạp hợp thành từ những âm thanh đơn giản.

Để chứng minh điều này, chúng ta hãy thiết lập một ca khúc nhịp điệu đơn giản. Đó là mẫu Drumkit khá phổ biến:



Một mẫu drum đơn giản

trong khi hihat được chơi tất cả các nốt thứ tám, kick and snare được chơi xen kẽ mỗi quý, 4/4 thời gian.

Đoạn mã này thực hiện điều đó:

var RhythmSample = {};  
  
RhythmSample.play = function() {  
    function playSound(buffer, time) {  
        var source = context.createBufferSource();  
        source.buffer = buffer;  
        source.connect(context.destination);  
        source.noteOn(time);  
    }  
  
    var kick = BUFFERS.kick;  
    var snare = BUFFERS.snare;  
    var hihat = BUFFERS.hihat;  
  
    // We'll start playing the rhythm 100 milliseconds from "now"  
    var startTime = context.currentTime + 0.100;  
    var tempo = 80; // BPM (beats per minute)  
    var eighthNoteTime = (60 / tempo) / 2;  
  
    // Play 2 bars of the following:  
    for (var bar = 0; bar < 2; bar++) {  
        var time = startTime + bar \* 8 \* eighthNoteTime;  
        // Play the bass (kick) drum on beats 1, 5  
        playSound(kick, time);  
        playSound(kick, time + 4 \* eighthNoteTime);  
  
        // Play the snare drum on beats 3, 7  
        playSound(snare, time + 2 \* eighthNoteTime);  
        playSound(snare, time + 6 \* eighthNoteTime);  
  
        // Play the hi-hat every eighthh note.  
        for (var i = 0; i < 8; ++i) {  
            playSound(hihat, time + i \* eighthNoteTime);  
        }  
    }  
};

* Thay đổi âm lượng

Một trong những thao tác cơ bản nhất mà người dùng muốn làm với một âm thanh thay đổi âm lượng của nó. Sử dụng API Audio Web, chúng ta có thể dùng AudioGainNode để điều khiển âm lượng:



Đồ thị âm thanh với GainNode

Thiết lập kết nối này có thể đạt được như sau:

*// Create a gain node.*

var gainNode = context.createGainNode();

*// Connect the source to the gain node.*

source.connect(gainNode);

*// Connect the gain node to the destination.*

gainNode.connect(context.destination);

tiếp theo, chúng ta đã có thể thay đổi âm lượng bởi thao tác gainNode.gain.value như sau:

*// Reduce the volume.*

gainNode.gain.value = 0.5;

Ngoài ra, chúng ta cũng có thể sử dụng AudioGainNode để trộn các âm thanh lại với nhau (tạo ra ứng dụng chơi nhạc cho DJ trên ngay trình duyệt).

Chúng ta cũng có thể lọc và tạo ra các hiệu ứng âm thanh đơn giản bằng đối tượng BiquadFilterNode



Biểu đồ âm thanh với BiquadFilterNode

Đoạn mã sau đây là ví dụ đơn giản để lọc low-bass từ một âm thanh:

*// Create the filter*

var filter = context.createBiquadFilter();

*// Create the audio graph.*

source.connect(filter);

filter.connect(context.destination);

*// Create and specify parameters for the low-pass filter.*

filter.type = 0; *// Low-pass filter. See BiquadFilterNode docs*

filter.frequency.value = 440; *// Set cutoff to 440 HZ*

*// Playback the sound.*

source.noteOn(0);

1. Nhạc nền

Khi tạo một trò chơi, một trong những yếu tố quan trọng đó chính là nhạc nền, chúng ta có thể vận dụng những gì đã tìm hiểu về Web Audio API ở phần trước đề tạo ra nhạc nền hấp dẫn, không đơn điệu. Nhạc nền, không có nghĩa chỉ là lặp đi lặp lại đoạn nhạc trong suốt trò chơi, nó dễ dàng tạo nên cảm giác nhàm chán cho người chơi.

Trong từng hoàn cảnh khác nhau, đòi hỏi nhạc nền cũng khác nhau, ví dụ lúc chiến đấu bình thường, lúc đánh với boss, lúc sắp mất hết máu,… Để giải quyết điều này, ta có thể dùng lớp BufferLoader để load các nhạc nền sẵn sàng lúc bắt đầu game.

function BufferLoader(context, urlList, callback) {

this.context = context;

this.urlList = urlList;

this.onload = callback;

this.bufferList = new Array();

this.loadCount = 0;

}

BufferLoader.prototype.loadBuffer = function(url, index) {

var request = new XMLHttpRequest();

request.open("GET", url, true);

request.responseType = "arraybuffer";

var loader = this;

request.onload = function() {

loader.context.decodeAudioData(request.response, function(buffer) {

if (!buffer) {

alert('error decoding file data: ' + url);

return;

}

loader.bufferList[index] = buffer;

if (++loader.loadCount == loader.urlList.length) loader.onload(loader.bufferList);

}, function(error) {

console.error('decodeAudioData error', error);

});

}

request.onerror = function() {

alert('BufferLoader: XHR error');

}

request.send();

}

BufferLoader.prototype.load = function() {

for (var i = 0; i < this.urlList.length; ++i)

this.loadBuffer(this.urlList[i], i);

}

Tiếp theo, mỗi nhạc nền ta tạo cho nó một GainNode tương ứng, ví dụ:

*// Assume gains is an array of AudioGainNode, normVal is the intensity*

*// between 0 and 1.*

var value = normVal \* (gains.length - 1);

*// First reset gains on all nodes.*

for (var i = 0; i < gains.length; i++) {

gains[i].gain.value = 0;

}

*// Decide which two nodes we are currently between, and do an equal*

*// power crossfade between them.*

var leftNode = Math.floor(value);

*// Normalize the value between 0 and 1.*

var x = value - leftNode;

var gain1 = Math.cos(x \* 0.5\*Math.PI);

var gain2 = Math.cos((1.0 - x) \* 0.5\*Math.PI);

*// Set the two gains accordingly.*

gains[leftNode].gain.value = gain1;

*// Check to make sure that there's a right node.*

if (leftNode < gains.length - 1) {

*// If there is, adjust its gain.*

gains[leftNode + 1].gain.value = gain2;

}

Như vậy, chúng ta có thể dễ dàng quản lý toàn bộ nhạc nền, rộng hơn nữa là quản lý các âm thanh của toàn bộ trò chơi một cách dễ dàng.

1. Hiệu ứng âm thanh

Một phần quan trọng của trò chơi nữa là hiệu ứng âm thanh, hàng loạt các hiệu ứng như vụ cháy nổ, tiếng súng, tiếng bước chân, chạy,… Nó cũng phải phát đi phát lại tương tự như âm thanh nền, nhưng có vài điểm khác cơ bản là:

+ Có hàng loạt các hiệu ứng tương tự nhau, chỉ khác nhau một chút về âm thanh cũng như tốc độ phát ra như tiếng các loại súng, tiếng bước chân từ chậm đến nhanh dần. Nói chung, chúng là những biến thể nhẹ của nhau.

+ Các âm thanh này phải lặp đi lặp lại nhiều lần và có thể phát cùng lúc hàng loạt các hiệu ứng khác nhau, trong thời gian chính xác. Hãy tưởng tượng đến một cuộc đấu súng giữa một nhóm người, tiếng các loại súng, tiếng người, tiếng bước chân, tiếng va chạm,…

Hiệu ứng âm thanh của Web Audio API thì thật sự tỏa sáng trong việc giải quyết các vấn đề trên.

Ví dụ sau đây tạo ra một vụ bắn súng máy từ một tiếng đạn được phát lại dồn dập trong một thời gian.

var time = context.currentTime;

for (var i = 0; i < rounds; i++) {

var source = this.makeSource(this.buffers[M4A1]);

source.noteOn(time + i \* interval);

}

Sau khi tạo ra được các tiếng súng đơn giản, ta nghĩ đến việc tiếng súng thay đổi tùy vào khoảng cách từ các mục tiêu cũng như vị trí tương đối. May mắn thay, Web Audio API cung cấp cho chúng ta có thể tùy chỉnh nó một cách dễ dàng theo hai cách:

+ Với một sự thay đổi tinh tế trong thời gian bắn ra của các viên đạn.

+ Bằng cách thay đổi playbackRate của âm thanh một cách ngẫu nhiên giống như ngoài đời thực.

Ví dụ:

function MachineGun(context) {

var ctx = this;

var loader = new BufferLoader(context, ['sounds/m4a1.mp3', 'sounds/m1-garand.mp3'], onLoaded);

function onLoaded(buffers) {

ctx.buffers = buffers;

};

loader.load();

}

MachineGun.prototype.shootRound = function(type, rounds, interval, random, random2) {

if (typeof random == 'undefined') {

random = 0;

}

var time = context.currentTime;

for (var i = 0; i < rounds; i++) {

var source = this.makeSource(this.buffers[type]);

source.playbackRate.value = 1 + Math.random() \* random2;

source.noteOn(time + i \* interval + Math.random() \* random);

}

}

MachineGun.prototype.makeSource = function(buffer) {

var source = context.createBufferSource();

var compressor = context.createDynamicsCompressor();

var gain = context.createGainNode();

gain.gain.value = 0.2;

source.buffer = buffer;

source.connect(gain);

gain.connect(compressor);

compressor.connect(context.destination);

return source;

};

1. Định vị âm thanh 3D

Khi tạo ra một trò chơi, ta thường phải tuân thủ các tính chất vật lý, hình học, của không gian 2D cũng như 3D. Nếu ta cũng tuân thủ điều đó đối với âm thanh thì quả là một điều tuyệt vời. Hãy nghĩ đến việc tiến lại gần hoặc ra xa mục tiêu, người chơi nhận được các âm thanh lớn nhỏ khác nhau một cách chân thật. Và tất nhiên, Web Audio API hỗ trợ ta làm điều đó một cách dễ dàng thông qua việc sử dụng AudioPannerNode. Ý tưởng cơ bản là hoạt động bằng cách thay đổi thiết lập vị trí các nguồn âm thanh như sau:

PositionSample.prototype.changePosition = function(position) {

*// Position coordinates are in normalized canvas coordinates*

*// with -0.5 < x, y < 0.5*

if (position) {

if (!this.isPlaying) {

this.play();

}

var mul = 2;

var x = position.x / this.size.width;

var y = -position.y / this.size.height;

this.panner.setPosition(x \* mul, y \* mul, -0.5);

} else {

this.stop();

}

};

1. Kết luận

Đó là tất cả những khía cạnh quan trọng, cần thiết để phát triển âm thanh cho trò chơi bằng cách sử dụng Web Audio API. Với kĩ thuật này, chúng ta có thể tạo ra những trải nghiệm âm thanh thật sự hấp dẫn ngay trên trình duyệt. Nhưng trước khi kêt thúc phần này, tôi đưa ra một lưu ý quan trọng mà người lập trình cần phải chú ý, đó là phải đảm bảo rằng âm thanh phải tạm dừng khi người dùng chuyển tab, chuyển trang hay cho trình duyệt chạy nền. Nếu không làm như vậy, chắc chắn nó sẽ tạo ra cảm giác vô cùng bực bội cho người dùng.

1. Thẻ <video>

Các thẻ <video> là một trong những tính năng của HTML5 được rất nhiều sự chú ý. Thường được sử dụng thay thế cho flash, bên cạnh đó, nó còn có những tính năng thật sự mạnh mẽ hơn rất nhiều. Mặc dù gần đây đã tham gia phần còn lại của các thẻ HTML phổ biến, khả năng của mình và hỗ trợ trên các trình duyệt đã tăng lên tại một tốc độ đáng kinh ngạc. Nó có thể tích hợp chồng với các lớp khác của web như CSS và JavaScript cũng như các thẻ HTML khác điển hình như thẻ <canvas>.

1. Các đánh dấu.

Để nhúng video vào trang web, chỉ cần các dòng sau đây là đủ:

<video>

<source src="movie.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"' />

<source src="movie.webm" type='video/webm; codecs="vp8, vorbis"' />

</video>

Đoạn này sử dụng thẻ <source> cho phép bao gồm nhiều định dạng khác nhau phòng trường hợp trình duyệt của người dùng trình duyệt không hỗ trợ một trong số đó.

Tất nhiên, chúng ta vẫn có thể sử dụng một định dạng video duy nhất làm cho cú pháp giống với các thẻ hình ảnh. Cú pháp này sẽ được sử dụng phổ biến hơn trong tương lai, khi mà tất cả các trình duyệt đều hỗ trợ các định dạng video phổ thông.

<video src="movie.webm"></video>

Để cải thiện hiệu suất phía người dùng điều quan trọng là xác định các loại định dạng trong các thẻ <source> trước tiên. Bằng cách này, trình duyệt có thể quyết định có nên tải về và chơi một video nào đó hay không. Nghĩa là, nó sẽ không tải về những loại video mà nó không thể chơi, bằng cách đó, chúng ta có thể cải thiện được hiệu suất của trang web.

1. Các định dạng Video

Hãy nghĩ một định dạng video như là một tập tin zip chứa các dòng video mã hóa và dòng âm thanh. Ba định dạng chúng ta cần quan tâm để phát triển web là (webm, mp4 và ogv):

* .mp4 = H.264 + AAC
* .ogg/.ogv = Theora + Vorbis
* .webm = VP8 + Vorbis

Tốc độ phát triển của thẻ <video> thật sự rất mạnh mẽ. Chỉ hai năm trước, khi mà thẻ <video> chỉ được hỗ trợ trên trình duyệt Safari, thì giờ đây hầu như tất cả các trình duyệt hiện tại đều hỗ trợ nó. Ngoài ra định dạng Webm càng trở nên phổ biến hơn thông qua dự án WebM. Internet Explore cũng hỗ trợ nó nếu codes này được cài đặt trên máy tính.

1. Các tính năng quan trọng với phát triển game

Như đã nói trong phần giới thiệu lợi thế chính của các thẻ <video> là một thành phần của HTML5, vì vậy nó có thể tích hợp với các lớp khác của web. Sau đây là những tính năng quan trọng đó:

* Video + các thành phần HTML khác

Tất cả các thuộc tính HTML thông thường bây giờ đã có thể được sử dụng trong thẻ <video>. Ví dụ, trong đoạn mã sau, sử dụng tabindex để bắt sự kiện từ bần phím. Có những thuộc tính mới có thể được sử dụng trong thẻ <video> cũng như thẻ <audio> chẳng hạn như “loop” và “autoplay”. Thuộc tính “poster” xác định hình ảnh nào sẽ được hiện khi video đang được load và khi đã kết thúc, “control” được sử dụng để chỉ ra rằng thay vì xây dựng các điều khiển tùy chỉnh, chúng tôi muốn trình duyệt tự động phát sinh nó. Ngoài ra còn có thuộc tính “preload”, chúng ta có thể sử dụng để tải về các video ở nền bên dưới ngay khi tải trang, ngay cả khi nó chưa bắt đầu phát.

<video *poster="star.png" autoplay loop controls tabindex="0"*>

<source src="movie.webm" type='video/webm; codecs="vp8, vorbis"' />

<source src="movie.ogv" type='video/ogg; codecs="theora, vorbis"' />

</video>

Kể từ khi video không còn là một đối tượng nhúng từ bên ngoài, có một số lợi ích khác cho trải nghiệm người dùng. Ví dụ, ngay cả khi nó đang được “focus” thì người dùng vẫn có thể scroll trang web hay sử dụng bàn phím bình thường.

* Video + JavaScript

Các thẻ video đi kèm với một tập hợp các thuộc tính và các phương thức, cho phép chúng ta có điều khiển tốt video từ mã JS.

Như bất kỳ phần tử HTML khác, chúng ta có thể gắn các sự kiện phổ biến cho thẻ <video> như “drag”, “mouse”,” focus”,… Tuy nhiên, điều này kèm theo hàng loạt các sự kiện mới phát sinh khi video đang phát, tạm dừng hoặc kết thúc. Khi bắt đầu tải một tài nguyên video về thì có rất nhiều thứ cần phải cẩn trọng, cũng như có rất nhiều sự kiện cần phải kiểm soát tốt trong quá trình tải, ở tầng mạng (loadstart, progress, suspend, abort, error, emptied, stalled) lẫn ở bộ đệm (loadedmetadata, loadeddata, waiting, playing, canplay, canplaythrough).

Ở cấp độ đơn giản nhất, bạn có thể gắn sự kiện “canplay” để bắt đầu làm việc với video.

video.addEventListener('canplay', function(e) {

this.volume = 0.4;

this.currentTime = 10;

this.play();

}, false);

* Video + CSS

Các thẻ <video> có thể được sử dụng theo CSS truyền thống (ví dụ như “border”, “opacity”,…) vì nó là một phần tử class đầu tiên trong DOM. Nhưng điều thú vị là ở bản mới nhất CSS3 thành phần video cũng có các thuộc tính như “reflections”, “masks”, “gradients”, “transforms”, “transitions” and “animations”.

*#video-css.enhanced {*

border: 1px solid white;

-moz-box-shadow: 0px 0px 4px *#ffffff; /\* FF3.5+ \*/*

-webkit-box-shadow: 5px 44px 28px *#333; /\* Saf3.0+, Chrome \*/*

box-shadow: 0px 0px 4px *#ffffff; /\* Opera 10.5, IE 9.0 \*/*

-moz-transform: translate(0, 10px); */\* FF3.5+ \*/*

-o-transform: translate(0, 10px); */\* Opera 10.5 \*/*

-webkit-transform: translate(0, 10px); */\* Saf3.1+, Chrome \*/*

}

* Video + Canvas

Canvas là một thành phần của HTML5 có nhiều tiềm năng khi được sử dụng kết hợp với các thẻ <video>.

Chúng ta có thể tận hai tính năng của phần tử <canvas> là nhập và xuất hình ảnh. Đầu tiên là phương thức drawImage cho phép nhập hình ảnh từ ba nguồn khác nhau: phần tử hình ảnh, phần tử <canvas> và phần tử <video>! Điều này có nghĩa rằng mỗi lần chúng ta chạy phương thức này, frame hiện tại trong đoạn video sẽ được nhập và trả lại vào canvas.

Đặc trưng thứ hai của thẻ <canvas> là chúng ta có thể sử dụng phương thức toDataURL để xuất nội dung của canvas thành một hình ảnh.

Đoạn mã sau đây sử dụng hàm drawImage cứ sau một khoảng thời gian 1,5 giây sẽ vẽ một hình ảnh từ video nguồn.

video\_dom.addEventListener('play', function() {

video\_dom.width = canvas\_draw.width = video\_dom.offsetWidth;

video\_dom.height = canvas\_draw.height = video\_dom.offsetHeight;

var ctx\_draw = canvas\_draw.getContext('2d');

draw\_interval = setInterval(function() {

*// import the image from the video*

ctx\_draw.drawImage(video\_dom, 0, 0, video\_dom.width, video\_dom.height);

*// export the image from the canvas*

var img = new Image();

img.src = canvas\_draw.toDataURL('image/png');

img.width = 40;

frames.appendChild(img);

}, 1500)

}, false);

* Video + SVG

SVG giúp chúng ta render và thao tác đồ họa vector, ngoài ra nó cũng đi kèm với nhiều tính năng như hiệu ứng lọc SVG. Với các bộ lọc, bạn có thể nhắm mục tiêu một phần tử DOM cụ thể và áp dụng một số các hiệu ứng như cảnh mờ, pha trộn, gạch,...

Ví dụ:

<svg id='image' version="1.1" xmlns="http://www.w3.org/2000/svg">

<defs>

<filter id="myblur">

<feGaussianBlur stdDeviation="1" />

</filter>

</defs>

</svg>

<style>

video { filter:url(#myblur); border: 2px solid red; }

</style>

1. Thu âm và quay phim bằng HTML5

Trước đây, để truy cập tới máy ảnh hay micro của người dùng chúng ta đã phải dựa vào các plugin (Flash hoặc Silverlight).

Giờ đây nhờ có HTML5 mang lại một sự đột phá về quyền truy cập vào thiết bị phần cứng cũng như định vị (GPS), API định hướng (gia tốc), WebGL (GPU), và Web Audio API (phần cứng) là những ví dụ điển hình.

Một API mới, navigator.getUserMedia(), cho phép các ứng dụng web để truy cập vào máy ảnh và microphone của người dùng.

1. Cơ bản

Giờ đây, với navigator.getUserMedia(), chúng ta đã có thể truy cập vào webcam và micro của người dùng mà không cần đến plugin. Thực hiện một cuộc gọi ngay trên trình duyệt, không cần phải cài đặt thêm gì cả, thật tuyệt vời!

* Kiểm tra sự hỗ trợ của trình duyệt

Tính năng này cho phép người lặp trình kiểm tra trình duyệt của người dùng có hỗ trợ chức năng này hay không:

function hasGetUserMedia() {

*// Note: Opera is unprefixed.*

return !!(navigator.getUserMedia || navigator.webkitGetUserMedia ||

navigator.mozGetUserMedia || navigator.msGetUserMedia);

}

if (hasGetUserMedia()) {

*// Good to go!*

} else {

alert('getUserMedia() is not supported in your browser');

}

* Tiếp cận với thiết bị đầu vào

Để sử dụng webcam hay microphone, chúng ta cần phải yêu cầu sự cho phép. Tham số đầu tiên getUserMedia () là một đối tượng xác định kiểu của thiết bị mà chúng ta muốn truy cập. Ví dụ, nếu muốn truy cập vào webcam, tham số đầu tiên là {video: true}. Để sử dụng cả hai micro và máy ảnh, pass {video: true, audio: true}:

<video autoplay></video>

<script>

var onFailSoHard = function(e) {

console.log('Reeeejected!', e);

};

*// Not showing vendor prefixes.*

navigator.getUserMedia({video: true, audio: true}, function(localMediaStream) {

var video = document.querySelector('video');

video.src = window.URL.createObjectURL(localMediaStream);

*// Note: onloadedmetadata doesn't fire in Chrome when using it with getUserMedia.*

*// See crbug.com/110938.*

video.onloadedmetadata = function(e) {

*// Ready to go. Do some stuff.*

};

}, onFailSoHard);

</script>

* Bảo mật

Một số trình duyệt trực tiếp hoạt động khi có lệnh gọi getUserMedia(), mà trước tiên nó cho phép người dùng tùy chọn cấp quyền hoặc từ chối quyền vào máy ảnh / micro của họ. Ví dụ, đây là hộp thoại confirm của Chrome:



* Xử lí khi trình duyệt không hỗ trợ

Đối với trường hợp trình duyệt người dùng không có hỗ trợ getUserMedia(), dưới đây là một trong những cách xử lí trường hợp này:

*// Not showing vendor prefixes or code that works cross-browser:*

function fallback(e) {

video.src = 'fallbackvideo.webm';

}

function success(stream) {

video.src = window.URL.createObjectURL(stream);

}

if (!navigator.getUserMedia) {

fallback();

} else {

navigator.getUserMedia({video: true}, success, fallback);

}

1. Chụp hình

Với phương thức ctx.drawImage(video, 0, 0) của thẻ <canvas> ta có thể dễ dàng vẽ frame <video> lên <canvas>. Chúng ta cũng đã có đầu vào video thông qua phương thức getUserMedia(), chúng ta có thể tạo ra hàng loạt các ứng dụng cũng như các trò chơi từ những thứ trên:

<video autoplay></video>

<img src="">

<canvas style="display:none;"></canvas>

var video = document.querySelector('video');

var canvas = document.querySelector('canvas');

var ctx = canvas.getContext('2d');

var localMediaStream = null;

function snapshot() {

if (localMediaStream) {

ctx.drawImage(video, 0, 0);

// "image/webp" works in Chrome 18. In other browsers, this will fall back to image/png.

document.querySelector('img').src = canvas.toDataURL('image/webp');

}

}

video.addEventListener('click', snapshot, false);

// Not showing vendor prefixes or code that works cross-browser.

navigator.getUserMedia({video: true}, function(stream) {

video.src = window.URL.createObjectURL(stream);

localMediaStream = stream;

}, onFailSoHard);

1. Sử dụng hiệu ứng

* CSS Filters

Sử dụng bộ lọc CSS, chúng ta có thể áp dụng một số hiệu ứng cho <video>, ví dụ:

<style>

video {

width: 307px;

height: 250px;

background: rgba(255,255,255,0.5);

border: 1px solid #ccc;

}

.grayscale {

[+filter: grayscale(1);](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html#including_a_mixin)

}

.sepia {

[+filter: sepia(1);](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html#including_a_mixin)

}

.blur {

[+filter: blur(3px);](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html#including_a_mixin)

}

...

</style>

<video autoplay></video>

<script>

var idx = 0;

var filters = ['grayscale', 'sepia', 'blur', 'brightness', 'contrast', 'hue-rotate',

'hue-rotate2', 'hue-rotate3', 'saturate', 'invert', ''];

function changeFilter(e) {

var el = e.target;

el.className = '';

var effect = filters[idx++ % filters.length]; *// loop through filters.*

if (effect) {

el.classList.add(effect);

}

}

document.querySelector('video').addEventListener('click', changeFilter, false);

</script>

* WebGL Textures

Chúng ta có thể chụp hình ảnh video và render nó vào texture WebGL bằng cách sử dụng getUserMedia() và Three.js.

Hướng dẫn: <http://learningthreejs.com/blog/2012/02/07/live-video-in-webgl/>

Demo: <http://learningthreejs.com/data/live-video-in-webgl/>

1. Thu âm

Sử dụng getUserMedia() + Web Audio API chúng ta hoàn toàn có thể thu lại âm thanh ngay trên trình duyệt

Với phiên bản Chrome 24, chúng ta có thể kích hoạt cờ "Web Audio Input" trong tab about:flags để thử nghiệm với getUserMedia() + Web Audio API. Tuy nó vẫn trong giai đoạn phát triển, nhưng hiện đang hoạt động khá tốt.

var context = new window.webkitAudioContext();

navigator.webkitGetUserMedia({audio: true}, function(stream) {

var microphone = context.createMediaStreamSource(stream);

var filter = context.createBiquadFilter()

*// microphone -> filter -> destination.*

microphone.connect(filter);

filter.connect(context.destination);

}, onFailSoHard);