Animate volume meters

EXAMPLE 1: ADD A SINGLE VOLUME METER TO THE AUDIO PLAYER

Try it at JSBin:

In order to have a "volume meter" that will move upward/downward with the intensity of the music, we will compute the average intensity of all frequencies, and draw this value using a nice gradient-filled rectangle.

Here are the two functions we will call from the animation loop (borrowed and adapted from http://www.smartjava.org/content/exploring-html5-web-audio-visualizing-sound):

```
function drawVolumeMeter() {
  canvasContext.save();
 analyser.getByteFrequencyData(dataArray);
  var average =getAverageVolume(dataArray);
  // set the fill style to a nice gradient
  canvasContext.fillStyle=gradient;
  // draw the vertical meter
  canvasContext.fillRect(0,height-average,25,height);
  canvasContext.restore();
}

function getAverageVolume(array) {
  var values = 0;
```

```
    var average;
20. var length = array.length;
    // get all the frequency amplitudes
    for (var i = 0; i < length; i++) {
      values += array[i];
    }
    average = values / length;
    return average;
  }
```

Note that we work in the intensity domain (*line 4*) and once we've got the array with all frequency analysis data, we call (*line 5*) the `getAverageVolume` function that will compute the average value we will draw as a volume meter.

This is how we created the gradient:

```
  // create a vertical gradient of the height of the canvas
gradient =canvasContext.createLinearGradient(0,0,0,height);
gradient.addColorStop(1,'#000000');
gradient.addColorStop(0.75,'#ff0000');
gradient.addColorStop(0.25,'#ffff00');
gradient.addColorStop(0,'#ffffff');
```

And here is what the new animation loop looks like (for the sake of clarity, we have moved the code that draws the signal waveform to a separate function):

```
function visualize() {
  clearCanvas();
  drawVolumeMeter();
  drawWaveform();

   // call again the visualize function at 60 frames/s
  requestAnimationFrame(visualize);
}
```

Notice that we used the best practices seen in week 3 of the HTML5 part 1 course: we saved and restored the context in all functions called from this animation loop, that changed

something in the canvas context (see function`drawVolumeMeter` and `drawWaveForm` in the source code).

Example 2: draw two volume meters, one for each stereo channel

This time we need to split the signal and create a different analyser for each output channel. We also keep the analyser node that is being used to draw the waveform, as this one works on the stereo signal (and is connected to the destination in order to hear the signal).

We added a `stereoPanner` node right after the source and a left/right balance slider to control its `pan` property. Use this slider to see how the left and right volume meter react.

In order to isolate the left and the right channel (for creating individual volume meters), we used a new node called a Channel Splitter node. From this node, we created two routes, each going to an analyser (*lines 46 and 47* of the example below)

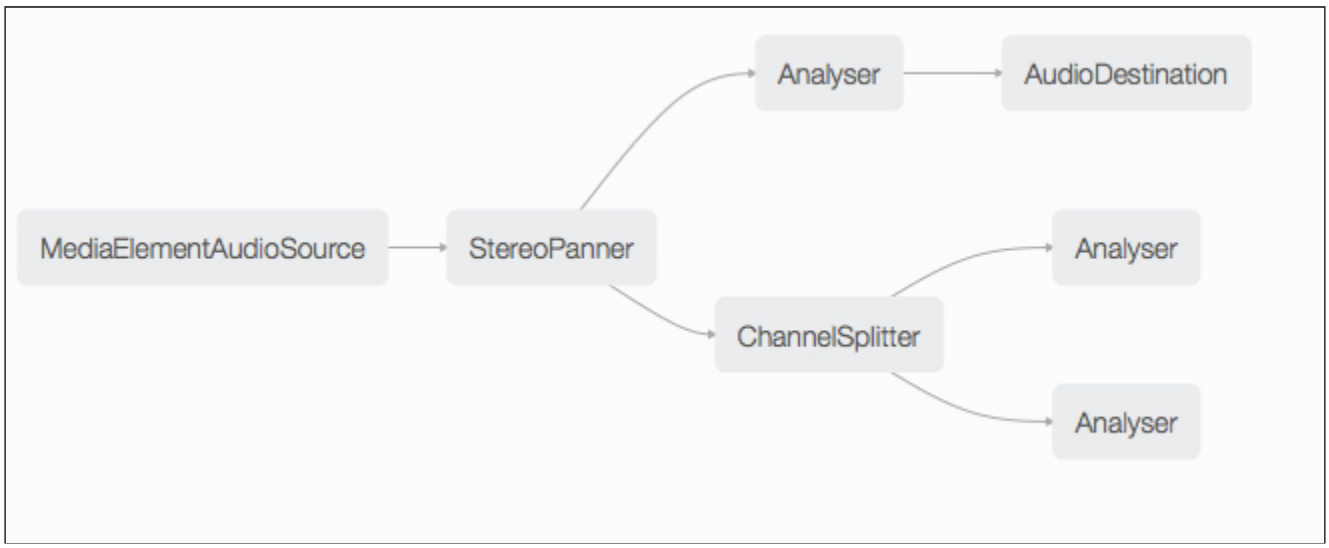
- See the documentation of the ChannelSplitterNode. Node that there is also a ChannelMergerNode for merging multiple routes into a single stereo one.

Use the `connect` method with extra parameters to connect the different outputs of the channel splitter node:

- `connect(node, 0, 0)` to connect the left output channel to another node,
- `connect(node, 1, 0)` to connect the right output channel to another node,

Example at JSBin:

This is the audio graph we built:



As you can see we've got two routes: the one on top sends the output signal to the speakers and uses an analyser node for animating the waveform, the one at the bottom splits the signal and send its left and right parts to separate analyser nodes used for drawing the two volume meters. Just before the split we added a stereoPanner so that we can adjust the left/right balance with a slider.

Source code extract:

```
function buildAudioGraph() {
  var mediaElement =document.getElementById('player');
  var sourceNode =audioContext.createMediaElementSource(mediaElement);
  // connect the source node to a stereo panner
  stereoPanner =audioContext.createStereoPanner();
  sourceNode.connect(stereoPanner);
  // Create an analyser node for the waveform
  analyser =audioContext.createAnalyser();
  // Use FFT value adapted to waveform drawing
  analyser.fftSize = 1024;
  bufferLength =analyser.frequencyBinCount;
  dataArray = newUint8Array(bufferLength);
  // Connect the stereo panner to the analyser
  stereoPanner.connect(analyser);
   // and the analyser to the destination
  analyser.connect(audioContext.destination);
  // End of route 1.  We start another route from the
  // stereoPanner node, with two analysers for the meters
  // Two analysers for the stereo volume meters
  // Here we use a small FFT value as we're gonna work with
```

```
    // frequency analysis data
    analyserLeft =audioContext.createAnalyser();
    analyserLeft.fftSize = 256;
    bufferLengthLeft =analyserLeft.frequencyBinCount;
    dataArrayLeft = newUint8Array(bufferLengthLeft);

    analyserRight =audioContext.createAnalyser();
    analyserRight.fftSize = 256;
    bufferLengthRight =analyserRight.frequencyBinCount;
    dataArrayRight = newUint8Array(bufferLengthRight);

    // Split the signal
    splitter =audioContext.createChannelSplitter();
    // Connect the stereo panner to the splitter node
    stereoPanner.connect(splitter);

    // Connect each of the outputs from the splitter to
    // the analysers
    splitter.connect(analyserLeft,0,0);
    splitter.connect(analyserRight,1,0);
    // No need to connect these analysers to something, the sound
    // is already connected through the route that goes through
    // the analyser used for the waveform
}
```

And here is the new function for drawing the two volume meters:

```
function drawVolumeMeters() {
    canvasContext.save();
    // set the fill style to a nice gradient
    canvasContext.fillStyle=gradient;
    // left channel
    analyserLeft.getByteFrequencyData(dataArrayLeft);
    var averageLeft =getAverageVolume(dataArrayLeft);

    // draw the vertical meter for left channel
    canvasContext.fillRect(0,height-averageLeft,25,height);
    // right channel
    analyserRight.getByteFrequencyData(dataArrayRight);
    var averageRight =getAverageVolume(dataArrayRight);
    // draw the vertical meter for left channel
```

```
    canvasContext.fillRect(26,height-averageRight,25,height);
21.   canvasContext.restore();
  }
```

The code is very similar to the previous one. We draw two different rectangles side to side, corresponding to two analyser nodes, instead of one in the previous example.