LINKÖPINGS UNIVERSITET

PITCHONARY™

Guess what your friend is drawing with their voice

Utbildning: Civilingenjör i medieteknik

Kurs: TFYA65, Ljudfysik

Författare: Fredrik Johansson, Pontus Arnesson, Vera Fristedt Andersson

Examinator: Peter Andersson och Per Sandström

Datum: 2018-10-14

Institutionen för teknik och naturvetenskap

Fredrik Johansson, frejo851
Pontus Arnesson, ponar842
Vera Fristedt Andersson, veran877

Fredrik Johansson, frejo851
Pontus Arnesson, ponar842
Vera Fristedt Andersson, veran877

Inledning

Denna rapport sammanfattar utförandet och resultatet av ett projekt i kursen Ljudfysik, TFYA65. Projektet ska vara relaterat till ljud, hålla tillräckligt hög nivå och vara genomförbart inom kursens ramar. Ämnet är valfritt. Eftersom projektet handlar om ljud, kommer endast projektdelen som handlar om ljudanalys förklaras ingående i denna rapport.

Mål

Målet med det valda projektet är att göra det möjligt för en användare att rita en sammanhängande figur, på en datorskärm, med hjälp av sin röst. Via analys av volym och frekvens på det ljud som datorns mikrofon upptar, kommer penselns position bestämmas och måla med avseende på dessa två parametrar. En hemsida, innehållande en kanvas där det går att rita, kommer programmeras i språken HTML, CSS och JavaScript, och den ska vara någorlunda estetiskt tilltalande. Kanvasens y-axel ska representera frekvensen och x-axeln ska representera volymen.

Bakgrund

Web Audio API

Applikationsprogrammeringssgränssnitt, förkortat API, är ett slags bibliotek som vanligen utgörs av en samling funktioner som är anropbara via en applikation. Web Audio API, skrivet i JavaScript, är ett verktyg som gör det möjligt att kontrollera ljud på webben genom att hantera ljudoperationer inuti en *audio context*. *Audio context*-gränssnittet representerar en ljudprocessorsgraf som är uppbyggd av länkade *audio nodes*. Det är *audio context* som sköter skapandet av noderna och bearbetningen av ljudet. *Audio node*-gränssnittet representerar en ljudprocessorsenhet, till exempel en ljudkälla. Det är dessa *audio nodes* som utför själva ljudoperationerna i *audio context*.

Analyser Node

Analyser node-gränssnittet representerar en *audio node* som gör det möjligt att erhålla information om frekvens- och tidsdomän i realtid. En *analyser node* skickar ljudströmmen från insignal till utsignal utan att ändra den, men sparar ljuddatan med hjälp av fourier transform, vilket gör det möjligt att jobba med ljudströmmen.

Funktioner

Funktioner tillhörande Web Audio API som kommer till stor nytta vid bearbetning av ljud:

- *getUserMedia()* - ber användaren om tillåtelse för att ta upp ljud från mikrofonen. Om tillåtelse ges, skickas ljudströmmen vidare till specificerad plats. Om tillåtelse inte ges, alternativt om något annat fel inträffar, skickas information om felet vidare till annan specificerad plats.
- *createMediaStreamSource()* - lagrar ljudström i en *audio node*.
- *createAnalyser()* - skapar en *analyser node*.

Autokorrelation

Autokorrelation är en metod som kan användas för att beräkna frekvensen hos en signal. Det görs genom att jämföra signalen med en förskjuten version av sig själv. De två identiska signalerna färdas förbi varandra i motsatt riktning och för varje steg beräknas korrelationen mellan dessa signaler. Korrelationen är ett värde mellan -1 och 1 beroende på om korrelationen är minimal respektive maximal. Dessa värdena skapar en ny våg där periodtiden avläses mellan globala maxima och närmaste lokala maxima med högst amplitud. Frekvensen hos den ursprungliga signal fås därefter genom att dividera 1 med periodtiden hos den nyskapade vågen.

Metod

Projektet inleddes med efterforskning kring Web Audio API eftersom ingen av gruppmedlemmarna hade använt det som verktyg tidigare. Efterforskningen ledde till tre kodexempel, se Bilaga 1, 2 och 3, med funktioner som skulle komma att behövas för att nå

projektets mål. Då dessa exempel innehöll mycket som var irrelevant för detta projekt, rensades och modifierades koden för att vara mer specifikt inriktad på projektets mål.

Fem filer skapades för att kunna strukturera upp koden mer. En HTML-fil för hemsidans struktur, en CSS-fil för hemsidans utseende, en JavaScript-fil som utgör grunden med funktionsanrop och ritning på en kanvas, en JavaScript-fil för att analysera ljudvolym, och en JavaScript-fil för att analysera ljudfrekvens.

Ljudanalys

För att läsa in ljud från mikrofonen används funktionen *getUserMedia()* som skickar den inlästa ljudströmmen till *createMediaStreamSource()* för lagring. Därefter anropas *createAnalyser()* för att få tillgång till ljuddatan. De två kodfilerna för analys av volym och frekvens används sedan för att beräkna vilken koordinat i vår kanvas som ljudet motsvarar. Som nämnt tidigare, representerar x-axeln volym och y-axeln frekvens, där origo är i det övre vänstra hörnet.

Ljudvolym

I JavaScript-filen som analyserar ljudvolym, delas signalen upp i sampel som vi summerar och tar kvadratroten ur för att få fram ljudstyrkan.

För att underlätta kontrollen av penseln, görs rörelsen i negativ x-led långsammare. Detta genom att ta ett värde mindre än 1, i vårt fall 0.95, som multipliceras med värdet på den aktuella volymen för att få ett jämförelsevärde. I nästa steg jämförs vilket av det aktuella värdet och det beräknade jämförelsevärdet som är störst. Har aktuella volymen gått ner till 0 är då jämförelsevärdet större, och detta målas ut som penselns position istället. Denna uträkning gör att ifall ljudvolymen sjunker drastiskt kommer penseln röra sig mjukt mot det nya värdet.

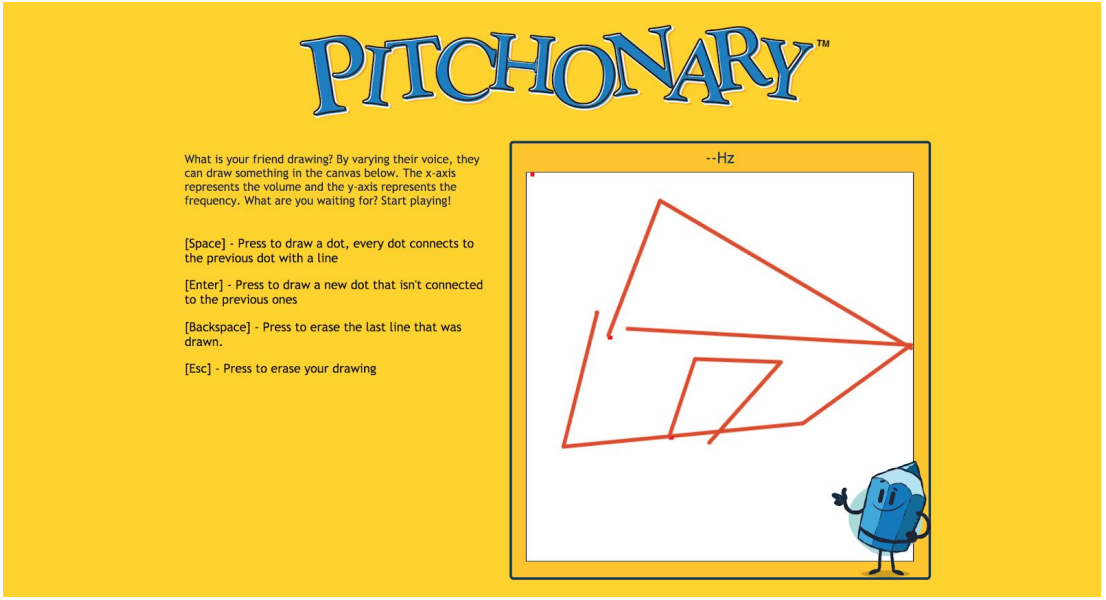
Ljudfrekvens

Till att börja med definieras ett värde för tillräckligt bra korrelation som i detta fall sätts till 0.9. Denna gräns innebär att när korrelationen når 0.9 avslutas funktionen och på så vis blir

funktionen snabbare samtidigt som den behåller noggrannhet. Däremot om korrelationen inte uppnår 0.9 returneras det högsta värdet som beräknats fram till och med sista sampel. Detta innebär att funktionen har två olika fall beroende på om 0.9 uppnås eller inte. I det första fallet när 0.9 uppnås, interpoleras de lokala maxima till höger och vänster kring den maximala korrelationen samt multipliceras med en konstant, och divideras med den maximala korrelationen. Detta värde multipliceras sedan med sampeltakten dividerat med den maximala korrelationen för att få ut frekvensen. I andra fallet, då 0.9 inte uppnås, returneras sampeltakten dividerat med den maximala korrelationen för att få ut frekvensen. Skillnaden är att detta inte multipliceras med en interpolerad faktor eftersom den maximala korrelationens position är okänd.

Resultat

Resultatet blev ett spel liknande Pictionary. Skillnaden är att användaren istället för att rita för hand använder sin röst för att rita. Logotypen och stilen på hemsidan är hämtad från Pictionary, men logotypen är istället ändrad till “Pitchonary” för att passa applikationen bättre. I Figur 1. nedan visas hemsidan med spelet.



Figur 1. Hus gjort med hjälp av en användares röst

För att rita på canvasen sparas positionen av markören varje gång mellanslagstangenten trycks ned. Alla sparade positioner sammanlänkas av linjer och det är dessa som utgör teckningen. Kanvasen suddas ut och ritas upp igen för varje frame med de sparade positionerna samt linjerna mellan dem. För att spara en ny position som inte är sammankopplad med de tidigare kan användaren trycka ner returtangenten istället för mellanslagstangenten. För att sudda det senaste strecket kan man trycka på backstegstangenten och för att sudda hela teckningen trycker man på escapetangenten.

Diskussion

Anledningen till att det blev ett spel istället för den ursprungliga idén med att endast ha en person som ritar för sig själv, var att det blev svårare att rita än vad vi trodde och applikationen var därför inte tillräckligt belönande. Genom att göra det till ett spel mellan flera personer så blir det mycket roligare och då gör det inte så mycket att det är svårt att rita med rösten.

Penselns rörelse är rätt hoppig och vi tror att det beror på att våra röster är polyfoniska. Vår kod är specificerad för en monofonisk ljudkälla och därmed hade vår ritningsfunktion nog fungerat bättre om koden var skriven åt en polyfonisk ljudkälla. Monofonisk betyder en ljudkälla som endast har en ton och våra röster har flera (mono = singel, poly = flera). Det blir oftast lättare att rita när man visslar vilket kan ha att göra med att det blir en mer ren ton. Vi utförde ett hörtest på vår applikation med en ljudfil som sakta ökade från låg till hög frekvens och där volymen varierade lite. Penseln höll sig väldigt stabil och hängde med i frekvensökningen och volymändringen så det fungerar felfritt för monofoniskt ljud. Alltså är det inte optimalt att applikationens syfte är att rita med en persons röst.

Från början var det tänkt att man skulle kunna hålla ned mellanslagstangenten för att få mjuka kurvor när man ritar, men eftersom det då sparas oerhört många positioner samt att kanvasen suddas ut och målas på igen hinner inte datorn rita ut allt utan att programmet fryser sig. Istället ger vi användaren instruktioner på hemsidan att endast klicka på mellanslagstangenten, så att programmet inte behöver spara för många positioner och därmed råka frysa sig. Detta leder till att kurvorna i teckningen blir kantiga men programmet

snabbare. Detta skulle kunna lösas genom att optimera ritningsfunktionen men vi valde att lägga energi på annat då denna del inte har med ljud att göra och därmed inte är särskilt relevant för projektet.

Hemsidan hade kunnat se mycket bättre ut om vi hade lagt ner mer tid på den. Eftersom kursen inte gick ut på att designa hemsidor, valde vi istället att fokusera på funktionaliteten hos applikationen istället för det visuella. Ljudet var vårt fokus i detta projekt och därför var det viktigast att få den delen att fungera bra.

Sammanfattning

Målet med projektet nåddes och i viss mån överträffades, då det blev mer än att bara rita på skärmen. Däremot blev det inte så lätt att rita som det var tänkt från början. Om det hade funnits mer tid hade ritningsfunktionen kunnat förbättras med hjälp av andra funktioner. Allt som allt, blev slutresultatet en färdig produkt som arbetar med ljud och som är fullt duglig att användas till det syfte som det var skapt för.

Referenser

https://developer.mozilla.org/en-US/docs/Web/API/AnalyserNode
https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
https://github.com/cwilso/pitchdetect
https://github.com/cwilso/volume-meter/
https://npwi.com/npwiAccount/courses/assignments/course8/pitch/
https://www.youtube.com/watch?v=ErtPBvEnQZc

Fredrik Johansson, frejo851
Pontus Arnesson, ponar842
Vera Fristedt Andersson, veran877

Bilagor

Bilaga 1. “main.js”

```
var audioContext = null;
var meter = null;
var canvasContext = null;
var WIDTH=500;
var HEIGHT=50;
var rafID = null;

window.onload = function() {

    // grab our canvas
    canvasContext = document.getElementById( "meter" ).getContext("2d");

    // monkeypatch Web Audio
    window.AudioContext = window.AudioContext || window.webkitAudioContext;

    // grab an audio context
    audioContext = new AudioContext();

    // Attempt to get audio input
    try {
        // monkeypatch getUserMedia
        navigator.getUserMedia =
            navigator.getUserMedia ||
            navigator.webkitGetUserMedia ||
            navigator.mozGetUserMedia;

        // ask for an audio input
        navigator.getUserMedia(
        {
            "audio": {
                "mandatory": {
                    "googEchoCancellation": "false",
                    "googAutoGainControl": "false",
                    "googNoiseSuppression": "false",
                    "googHighpassFilter": "false"
                },
                "optional": []
            },
        }, gotStream, didntGetStream);
    } catch (e) {
        alert('getUserMedia threw exception :' + e);
    }
```

```
    }

}


function didntGetStream() {
    alert('Stream generation failed.');
}

var mediaStreamSource = null;

function gotStream(stream) {
    // Create an AudioNode from the stream.
    mediaStreamSource = audioContext.createMediaStreamSource(stream);

    // Create a new volume meter and connect it.
    meter = createAudioMeter(audioContext);
    mediaStreamSource.connect(meter);

    // kick off the visual updating
    drawLoop();
}

function drawLoop( time ) {
    // clear the background
    canvasContext.clearRect(0,0,WIDTH,HEIGHT);

    // check if we're currently clipping
    if (meter.checkClipping())
        canvasContext.fillStyle = "red";
    else
        canvasContext.fillStyle = "green";

    // draw a bar based on the current volume
    canvasContext.fillRect(0, 0, meter.volume*WIDTH*1.4, HEIGHT);

    // set up the next visual callback
    rafID = window.requestAnimationFrame( drawLoop );
}
```

Fredrik Johansson, frejo851
Pontus Arnesson, ponar842
Vera Fristedt Andersson, veran877

Bilaga 2. “volume-meter.js”

```
function createAudioMeter(audioContext,clipLevel,averaging,clipLag) {
    var processor = audioContext.createScriptProcessor(512);
    processor.onaudioprocess = volumeAudioProcess;
    processor.clipping = false;
    processor.lastClip = 0;
    processor.volume = 0;
    processor.clipLevel = clipLevel || 0.98;
    processor.averaging = averaging || 0.95;
    processor.clipLag = clipLag || 750;

    // this will have no effect, since we don't copy the input to the output,
    // but works around a current Chrome bug.
    processor.connect(audioContext.destination);

    processor.checkClipping =
        function(){
            if (!this.clipping)
                return false;
            if ((this.lastClip + this.clipLag) < window.performance.now())
                this.clipping = false;
            return this.clipping;
        };

    processor.shutdown =
        function(){
            this.disconnect();
            this.onaudioprocess = null;
        };

    return processor;
}

function volumeAudioProcess( event ) {
    var buf = event.inputBuffer.getChannelData(0);
    var bufLength = buf.length;
    var sum = 0;
    var x;

    // Do a root-mean-square on the samples: sum up the squares...
```

```
    for (var i=0; i<bufLength; i++) {
        x = buf[i];
        if (Math.abs(x)>=this.clipLevel) {
            this.clipping = true;
            this.lastClip = window.performance.now();
        }
        sum += x * x;
    }

    // ... then take the square root of the sum.
    var rms =  Math.sqrt(sum / bufLength);

    // Now smooth this out with the averaging factor applied
    // to the previous sample - take the max here because we
    // want "fast attack, slow release."
    this.volume = Math.max(rms, this.volume*this.averaging);
}
```

Fredrik Johansson, frejo851
Pontus Arnesson, ponar842
Vera Fristedt Andersson, veran877

Bilaga 3. “pitchDetect.js”

```
window.AudioContext = window.AudioContext || window.webkitAudioContext;

var audioContext = null;
var isPlaying = false;
var sourceNode = null;
var analyser = null;
var theBuffer = null;
var DEBUGCANVAS = null;
var mediaStreamSource = null;
var detectorElem,
    canvasElem,
    waveCanvas,
    pitchElem,
    noteElem,
    detuneElem,
    detuneAmount;

window.onload = function() {
    audioContext = new AudioContext();
    MAX_SIZE = Math.max(4,Math.floor(audioContext.sampleRate/5000));    // corresponds to a 5kHz signal
    var request = new XMLHttpRequest();
    request.open("GET", "../sounds/whistling3.ogg", true);
    request.responseType = "arraybuffer";
    request.onload = function() {
      audioContext.decodeAudioData( request.response, function(buffer) {
            theBuffer = buffer;
        } );
    }
    request.send();

    detectorElem = document.getElementById( "detector" );
    canvasElem = document.getElementById( "output" );
    DEBUGCANVAS = document.getElementById( "waveform" );
    if (DEBUGCANVAS) {
        waveCanvas = DEBUGCANVAS.getContext("2d");
        waveCanvas.strokeStyle = "black";
        waveCanvas.lineWidth = 1;
    }
    pitchElem = document.getElementById( "pitch" );
    noteElem = document.getElementById( "note" );
    detuneElem = document.getElementById( "detune" );
    detuneAmount = document.getElementById( "detune_amt" );

    detectorElem.ondragenter = function () {
        this.classList.add("droptarget");
        return false; };
    detectorElem.ondragleave = function () { this.classList.remove("droptarget"); return false; };
    detectorElem.ondrop = function (e) {
        this.classList.remove("droptarget");
        e.preventDefault();
        theBuffer = null;

        var reader = new FileReader();
        reader.onload = function (event) {
            audioContext.decodeAudioData( event.target.result, function(buffer) {
                theBuffer = buffer;
            }, function(){alert("error loading!");} );

        };
        reader.onerror = function (event) {
            alert("Error: " + reader.error );
        };
        reader.readAsArrayBuffer(e.dataTransfer.files[0]);
        return false;
    };



}

function error() {
    alert('Stream generation failed.');
}

function getUserMedia(dictionary, callback) {
    try {
        navigator.getUserMedia =
            navigator.getUserMedia ||
            navigator.webkitGetUserMedia ||
            navigator.mozGetUserMedia;
        navigator.getUserMedia(dictionary, callback, error);
```

Fredrik Johansson, frejo851
Pontus Arnesson, ponar842
Vera Fristedt Andersson, veran877

```
            navigator.mozGetUserMedia;
        navigator.getUserMedia(dictionary, callback, error);
    } catch (e) {
        alert('getUserMedia threw exception :' + e);
    }
}

function gotStream(stream) {
    // Create an AudioNode from the stream.
    mediaStreamSource = audioContext.createMediaStreamSource(stream);

    // Connect it to the destination.
    analyser = audioContext.createAnalyser();
    analyser.fftSize = 2048;
    mediaStreamSource.connect( analyser );
    updatePitch();
}

function toggleOscillator() {
    if (isPlaying) {
        //stop playing and return
        sourceNode.stop( 0 );
        sourceNode = null;
        analyser = null;
        isPlaying = false;
        if (!window.cancelAnimationFrame)
            window.cancelAnimationFrame = window.webkitCancelAnimationFrame;
        window.cancelAnimationFrame( rafID );
        return "play oscillator";
    }
    sourceNode = audioContext.createOscillator();

    analyser = audioContext.createAnalyser();
    analyser.fftSize = 2048;
    sourceNode.connect( analyser );
    analyser.connect( audioContext.destination );
    sourceNode.start(0);
    isPlaying = true;
    isLiveInput = false;
    updatePitch();

    return "stop";
}

function toggleLiveInput() {
    if (isPlaying) {
        //stop playing and return
        sourceNode.stop( 0 );
        sourceNode = null;
        analyser = null;
        isPlaying = false;
        if (!window.cancelAnimationFrame)
            window.cancelAnimationFrame = window.webkitCancelAnimationFrame;
        window.cancelAnimationFrame( rafID );
    }
    getUserMedia(
        {
            "audio": {
                "mandatory": {
                    "googEchoCancellation": "false",
                    "googAutoGainControl": "false",
                    "googNoiseSuppression": "false",
                    "googHighpassFilter": "false"
                },
                "optional": []
            },
        }, gotStream);
}

function togglePlayback() {
    if (isPlaying) {
        //stop playing and return
        sourceNode.stop( 0 );
        sourceNode = null;
        analyser = null;
        isPlaying = false;
        if (!window.cancelAnimationFrame)
            window.cancelAnimationFrame = window.webkitCancelAnimationFrame;
        window.cancelAnimationFrame( rafID );
        return "start";
    }
```

Fredrik Johansson, frejo851
Pontus Arnesson, ponar842
Vera Fristedt Andersson, veran877

```
        return "start";
    }

    sourceNode = audioContext.createBufferSource();
    sourceNode.buffer = theBuffer;
    sourceNode.loop = true;

    analyser = audioContext.createAnalyser();
    analyser.fftSize = 2048;
    sourceNode.connect( analyser );
    analyser.connect( audioContext.destination );
    sourceNode.start( 0 );
    isPlaying = true;
    isLiveInput = false;
    updatePitch();

    return "stop";
}

var rafID = null;
var tracks = null;
var buflen = 1024;
var buf = new Float32Array( buflen );

var noteStrings = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"];

function noteFromPitch( frequency ) {
    var noteNum = 12 * (Math.log( frequency / 440 )/Math.log(2) );
    return Math.round( noteNum ) + 69;
}

function frequencyFromNoteNumber( note ) {
    return 440 * Math.pow(2,(note-69)/12);
}

function centsOffFromPitch( frequency, note ) {
    return Math.floor( 1200 * Math.log( frequency / frequencyFromNoteNumber( note ))/Math.log(2) );
}

var MIN_SAMPLES = 0;  // will be initialized when AudioContext is created.
var GOOD_ENOUGH_CORRELATION = 0.9; // this is the "bar" for how close a correlation needs to be
```

```

function autoCorrelate( buf, sampleRate ) {
    var SIZE = buf.length;
    var MAX_SAMPLES = Math.floor(SIZE/2);
    var best_offset = -1;
    var best_correlation = 0;
    var rms = 0;
    var foundGoodCorrelation = false;
    var correlations = new Array(MAX_SAMPLES);

    for (var i=0;i<SIZE;i++) {
        var val = b (local var) val: any
        rms += val*val;
    }
    rms = Math.sqrt(rms/SIZE);
    if (rms<0.01) // not enough signal
        return -1;

    var lastCorrelation=1;
    for (var offset = MIN_SAMPLES; offset < MAX_SAMPLES; offset++) {
        var correlation = 0;

        for (var i=0; i<MAX_SAMPLES; i++) {
            correlation += Math.abs((buf[i])-(buf[i+offset]));
        }
        correlation = 1 - (correlation/MAX_SAMPLES);
        correlations[offset] = correlation; // store it, for the tweaking we need to do below.
        if ((correlation>GOOD_ENOUGH_CORRELATION) && (correlation > lastCorrelation)) {
            foundGoodCorrelation = true;
            if (correlation > best_correlation) {
                best_correlation = correlation;
                best_offset = offset;
            }
        } else if (foundGoodCorrelation) {
```

```
        } else if (foundGoodCorrelation) {
            var shift = (correlations[best_offset+1] - correlations[best_offset-1])/correlations[best_offset];
            return sampleRate/(best_offset+(8*shift));
        }
        lastCorrelation = correlation;
    }
    if (best_correlation > 0.01) {
        // console.log("f = " + sampleRate/best_offset + "Hz (rms: " + rms + " confidence: " + best_correlation + ")")
        return sampleRate/best_offset;
    }
    return -1;
//  var best_frequency = sampleRate/best_offset;
}

function updatePitch( time ) {
    var cycles = new Array;
    analyser.getFloatTimeDomainData( buf );
    var ac = autoCorrelate( buf, audioContext.sampleRate );
    // TODO: Paint confidence meter on canvasElem here.

    if (DEBUGCANVAS) {  // This draws the current waveform, useful for debugging
        waveCanvas.clearRect(0,0,512,256);
        waveCanvas.strokeStyle = "red";
        waveCanvas.beginPath();
        waveCanvas.moveTo(0,0);
        waveCanvas.lineTo(0,256);
        waveCanvas.moveTo(128,0);
        waveCanvas.lineTo(128,256);
        waveCanvas.moveTo(256,0);
        waveCanvas.lineTo(256,256);
        waveCanvas.moveTo(384,0);
        waveCanvas.lineTo(384,256);
        waveCanvas.moveTo(512,0);
        waveCanvas.lineTo(512,256);
        waveCanvas.stroke();
        waveCanvas.strokeStyle = "black";
        waveCanvas.beginPath();
        waveCanvas.moveTo(0,buf[0]);
        for (var i=1;i<512;i++) {
            waveCanvas.lineTo(i,128+(buf[i]*128));
```

```
        }
        waveCanvas.stroke();
    }

    if (ac == -1) {
        detectorElem.className = "vague";
        pitchElem.innerText = "--";
        noteElem.innerText = "-";
        detuneElem.className = "";
        detuneAmount.innerText = "--";
    } else {
        detectorElem.className = "confident";
        pitch = ac;
        pitchElem.innerText = Math.round( pitch ) ;
        var note =  noteFromPitch( pitch );
        noteElem.innerHTML = noteStrings[note%12];
        var detune = centsOffFromPitch( pitch, note );
        if (detune == 0 ) {
            detuneElem.className = "";
            detuneAmount.innerHTML = "--";
        } else {
            if (detune < 0)
                detuneElem.className = "flat";
            else
                detuneElem.className = "sharp";
            detuneAmount.innerHTML = Math.abs( detune );
        }
    }

    if (!window.requestAnimationFrame)
        window.requestAnimationFrame = window.webkitRequestAnimationFrame;
    rafID = window.requestAnimationFrame( updatePitch );
}
```