WebAudio Deep Note

Stoyan Stefanov
@stoyanstefanov
DeveloperWeek
Nov 7, 2018 Austin, TX

youtube.com/results?search_query=simpsons+thx

Story time

- THX
- James Andy Moorer (interview)
- ASP
- Chaos to order inspiration:
  - J.S.Bach: Toccata and Fugue in D minor
  - The Beatles: A Day In the Life

reddit.com/r/movies/comments/8m7rpd/the_score_of_deep_note_thxs_audio_trademark/

What do we know?

- 30 voices, 11 notes
  - Guess: 8 x 2, top note x 6, 2 bottom ones x 4
- One D cello sample C3
- D = 150Hz (note frequencies)
- Just tuning (perfect ratios)

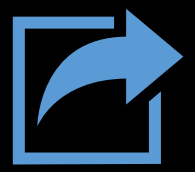
Load and play a sound

```
const actx = new AudioContext();
let sample;
```

Load and play a sound

```
function play() {
  fetch('Roland-SC-88-Cello-C3-glued-01.wav')
    .then(response => response.arrayBuffer())
    .then(arrayBuffer => actx.decodeAudioData(arrayBuffer))
    .then(audioBuffer => {
      sample = actx.createBufferSource();
      sample.buffer = audioBuffer;
      sample.connect(actx.destination);
      sample.start();
    })
    .catch(e => console.error('uff'));
}
```

Load and play a sound



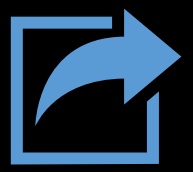
```
function stop() {
  sample.stop();
}
```

Nodes

bing.com/images/search?q=guitar+pedalboard

bing.com/images/search?q=modular+synth

Loop the sound

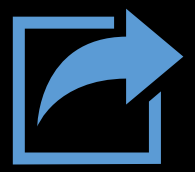


```
const sample = actx.createBufferSource();
sample.buffer = audioBuffer;
sample.loop = true;
sample.connect(actx.destination);
sample.start();
```

Side note: HTML audio is an option

```
<audio src="sound.mp3" autoplay="1" loop="1">

// or
new Audio('sound.mp3').play();
```

Repitch the sound

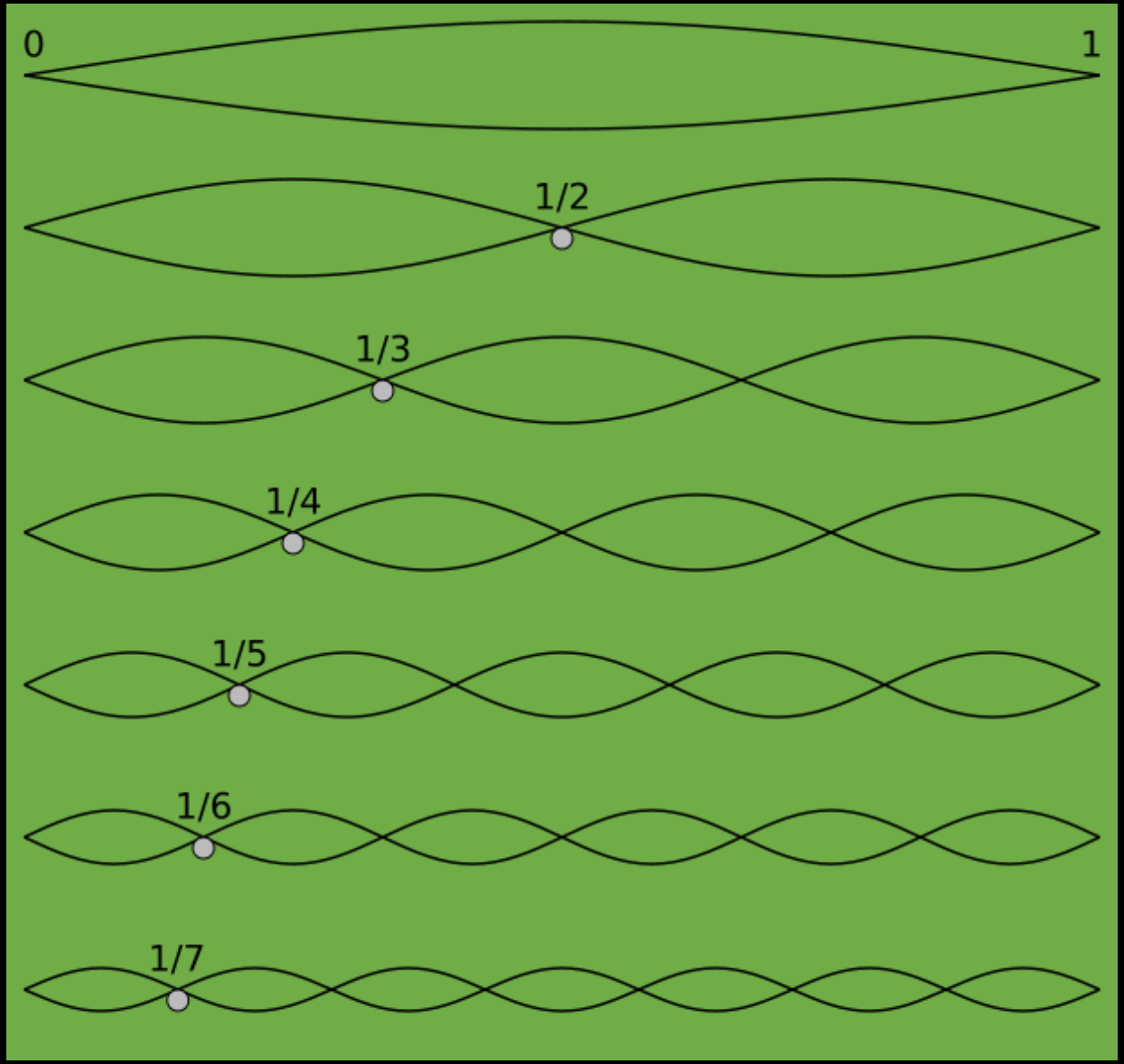


```
const C3 = 130.81;
const c3d150 = 150 / C3; // 1.1467013225;

const sample = actx.createBufferSource();
sample.buffer = audioBuffer;
sample.playbackRate.value = c3d150;
sample.connect(actx.destination);
sample.start();
```

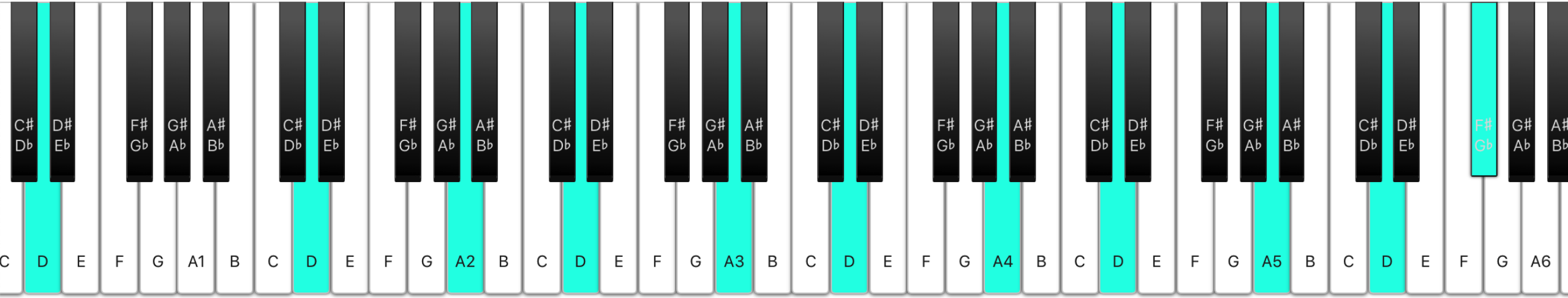
Try Web MIDI keyboard

Tuning

- Just intonation – perfect ratios
- Equal temperament - $\sqrt[12]{2}$


0
1
1/2
1/3
1/4
1/5
1/6
1/7

Final chord


C♯ D♭
D♯ E♭
F♯ G♭
G♯ A♭
A♯ B♭
C♯ D♭
D♯ E♭
F♯ G♭
G♯ A♭
A♯ B♭
C♯ D♭
D♯ E♭
F♯ G♭
G♯ A♭
A♯ B♭
C♯ D♭
D♯ E♭
F♯ G♭
G♯ A♭
A♯ B♭
C♯ D♭
D♯ E♭
F♯ G♭
G♯ A♭
A♯ B♭
C♯ D♭
D♯ E♭
F♯ G♭
G♯ A♭
A♯ B♭
C D E F G A1 B C D E F G A2 B C D E F G A3 B C D E F G A4 B C D E F G A5 B C D E F G A6

Final chord

```
const notes = {
  D1: {rate: 1/4, voices: 4},
  D2: {rate: 1/2, voices: 4},
  A2: {rate: 3/4, voices: 2},
  D3: {rate: 1,   voices: 2},
  A3: {rate: 3/2, voices: 2},
  D4: {rate: 2,   voices: 2},
  A4: {rate: 3,   voices: 2},
  D5: {rate: 4,   voices: 2},
  A5: {rate: 6,   voices: 2},
  D6: {rate: 8,   voices: 2},
  Fs: {rate: 10,  voices: 6},
};
```

Perfect octave:

D = D * 2

Perfect fifth:

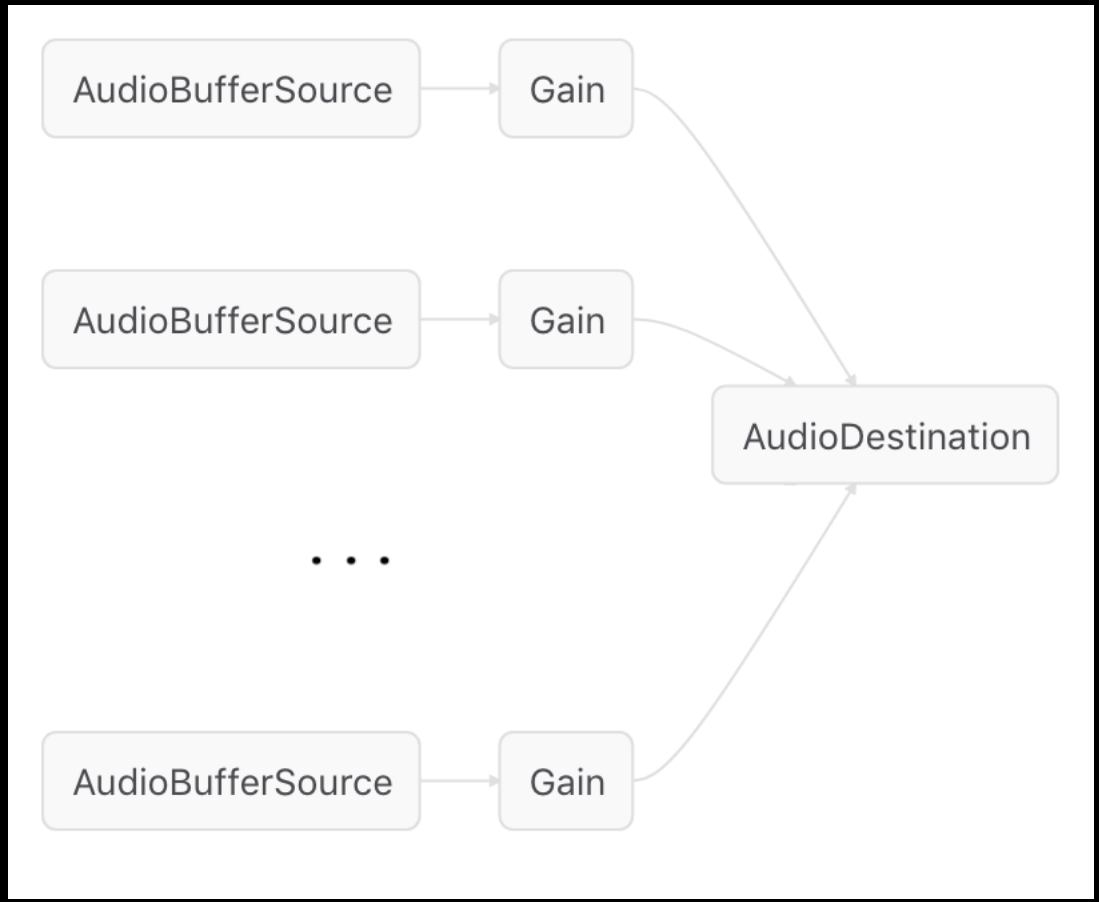
A = D * 3/2

Major third:

F sharp = D * 5/4;

```
load(['Roland-SC-88-Cello-C3-glued-01.wav']).then(buffers => {
  for (let note in notes) {
    const source = actx.createBufferSource();
    source.buffer = buffers.get('Roland-SC-88-Cello-C3-glued-01.wav');
    source.loop = true;
    source.playbackRate.value = c3d150 * notes[note].rate;

    const volume = actx.createGain();
    volume.gain.value = 0;
    source.connect(volume).connect(actx.destination);
    source.start();

    const range = document.createElement('input');
    range.type = 'range';
    range.oninput = (ev) => {
      volume.gain.value = Number(ev.target.value);
    };
    buttons.appendChild(range);
  };
});
```

Chord + sliders

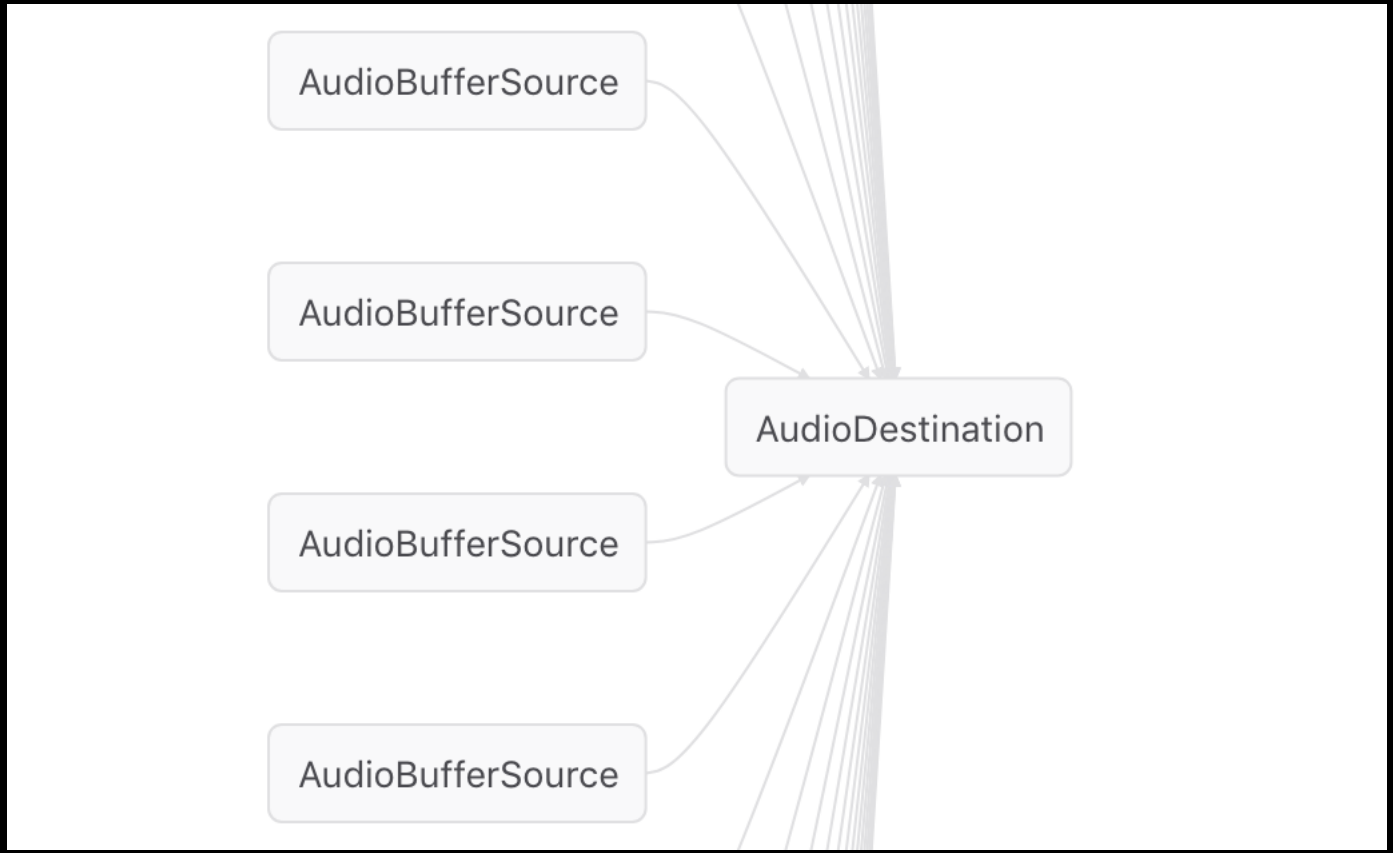


All the notes

```
const sources = [];

function stop() {
  for (let i = 0; i < sources.length; i++) {
    sources[i] && sources[i].stop();
    delete sources[i];
  }
}
```

```
function play() {
  load([SAMPLE]).then(buffers => {
    for (let note in notes) {
      for (let i = 0; i < notes[note].voices; i++) {
        const source = actx.createBufferSource();
        source.buffer = buffers.get(SAMPLE);
        source.loop = true;
        source.playbackRate.value =
          c3d150 * notes[note].rate;
        source.connect(actx.destination);
        source.start();
        sources.push(source);
      }
    };
  });
}
```

All the notes

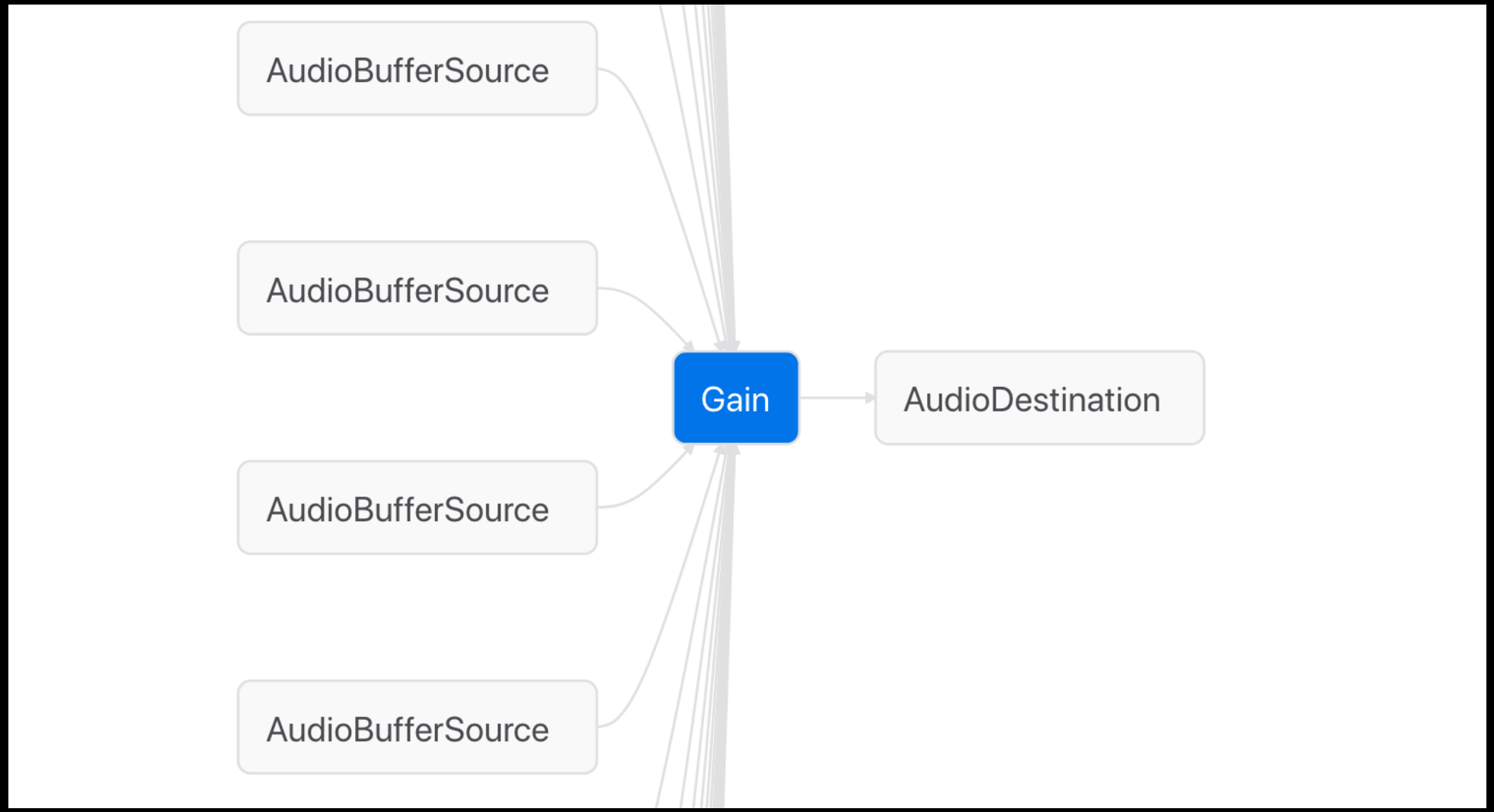


All the notes + gain

```
const releaseTime = 0.1;
const volume = actx.createGain();
volume.connect(actx.destination);

function stop() {
  volume.gain.linearRampToValueAtTime(
    0,
    actx.currentTime + releaseTime);
  for (let i = 0; i < sources.length; i++) {
    sources[i] && sources[i].stop();
    delete sources[i];
  }
}
```

```
function play() {
  load([SAMPLE]).then(buffers => {
    volume.gain.setValueAtTime(0, actx.currentTime);
    volume.gain.setTargetAtTime(1, actx.currentTime, 1);
    for (let note in notes) {
      for (let i = 0; i < notes[note].voices; i++) {
        // ...
        source.connect(volume);
        source.start();
        sources.push(source);
      }
    };
  });
}
```

All the notes + gain



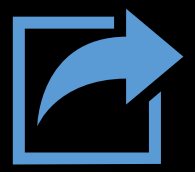
Automation/scheduling

- `setValueAtTime(value, start)`
- `linearRampToValueAtTime(value, end)`
- `exponentialRampToValueAtTime(value, end)`

- `setTargetAtTime(value, start, constant)`

- The second clock: `setTimeout()/requestAnimationFrame()`

Automate volume



```
function play() {
  stop();
  load([SAMPLE]).then(buffers => {

    // schedule volume automation
    volume.gain.setValueAtTime(0,    actx.currentTime);
    volume.gain.setTargetAtTime(0.1, actx.currentTime, 1);
    volume.gain.setTargetAtTime(0.5, actx.currentTime + 10, 4);
    volume.gain.setTargetAtTime(1.0, actx.currentTime + 14, 2);
    volume.gain.setTargetAtTime(0,   actx.currentTime + 20, 0.5);

    for (let note in notes) {
      for (let i = 0; i < notes[note].voices; i++) {
        // ...
    };
  });
}
```

Automate pitch

- Start with all notes randomly pitched 200 - 400Hz
- Update them approximately every second
- … but don’t let them drift too much
- At 10th second assign the target pitches
- … and allow 4 seconds to get there
- “slightly” detune the top note

Automate pitch

- Start with all notes randomly pitched 200 - 400Hz
- Some utilities:

```
function randRate200to400() {
  const freq = rand(200, 400);
  return freq / C3;
}

function rand(min, max) {
  return Math.random() * (max - min) + min;
}
```

Automate pitch

- Start with all notes randomly pitched 200 - 400Hz:

```
for (let note in notes) {
  for (let i = 0; i < notes[note].voices; i++) {
    const source = actx.createBufferSource();
    // ...

    const initial = randRate200to400();
    source.playbackRate.setValueAtTime(initial, actx.currentTime);

    // ...
    source.start();
  }
}
```

Automate pitch

- Update approximately every second:

```
for (let i = 1; i < 9; i++) {
  source.playbackRate.setTargetAtTime(
    initial + rand(-0.5, 0.5),
    actx.currentTime + rand(i - 0.5, i + 0.5),
    2,
  );
}
const finalRandom = initial + rand(-0.5, 0.5);
source.playbackRate.setTargetAtTime(finalRandom, actx.currentTime + 9, 2);
source.playbackRate.setValueAtTime(finalRandom, actx.currentTime + 10);
```

Automate pitch

- At 10th second assign the target pitches, detune the top note:

```
setTimeout(() => {
  source.playbackRate.exponentialRampToValueAtTime(
    c3d150 * notes[note].rate,
    actx.currentTime + 4,
  );
  if (note === 'Fs' && i > 1) { // 2 "correct" and 4 detuned
    source.detune.value = rand(-33, 33); // 100 cents = semitone
  }
}, 1000 * 10); // 10 seconds
```

`detune` vs `playbackRate`?

same thing

Slow down, same pitch? Autotune?

maybe one day

Sweetening

- Compression: `actx.createDynamicsCompressor()`
- Panning
- Reverb
- EQ

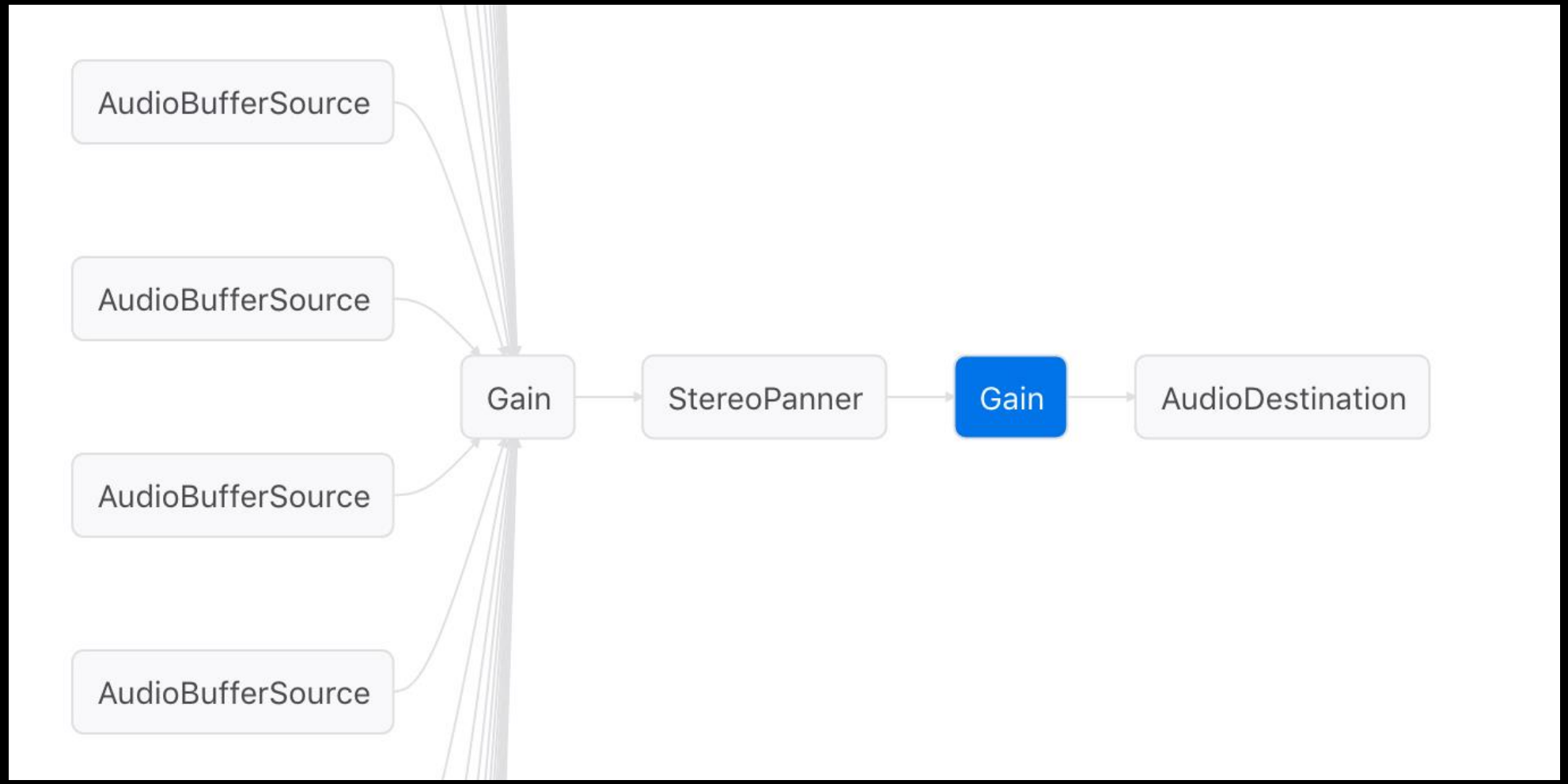
Panning: move in the stereo field

```
const panner = actx.createStereoPanner(); // stereo

panner.pan.setValueAtTime(0, 0);
panner.pan.setTargetAtTime(-1, actx.currentTime, 1);
panner.pan.setTargetAtTime(0.5, actx.currentTime + 4, 2);
panner.pan.setTargetAtTime(0, actx.currentTime + 8, 2);
```

Signal flow

```
const volume = actx.createGain(); // automation
const panner = actx.createStereoPanner(); // stereo
const master = actx.createGain(); // master volume

volume.connect(panner);
panner.connect(master);
master.connect(actx.destination);

// each note in the for-loop...
source.connect(volume);
```

Signal flow


AudioBufferSource
AudioBufferSource
AudioBufferSource
AudioBufferSource
Gain
StereoPanner
Gain
AudioDestination

Reverb: make it big

- Split the signal and route through reverb
- Automate the volume into the reverb
- EQ the signal going to the reverb
- Impulse response (IR) files

Reverb

```
load([SAMPLE, VERB]).then(buffers => {
  const reverb = actx.createConvolver();
  reverb.buffer = buffers.get(VERB);
  // ...
  reverb.connect();
  // ...
});
```

Reverb volume automation

```
const verbGain = actx.createGain();
const reverb = actx.createConvolver();
verbGain.gain.setValueAtTime(0.7, 0);
verbGain.gain.setTargetAtTime(
  0.1,
  actx.currentTime + 6,
  4,
);
```

Reverb EQ

```
const verbLowPass = actx.createBiquadFilter();
verbLowPass.type = 'lowpass';
verbLowPass.frequency.value = 3000;

const verbHighPass = actx.createBiquadFilter();
verbHighPass.type = 'highpass';
verbHighPass.frequency.value = 300;
```

Reverb: signal flow

```
// parallel verb
volume
  .connect(verbGain)
  .connect(verbHighPass)
  .connect(verbLowPass)
  .connect(reverb)
  .connect(panner);
// stereo panning
volume.connect(panner);
```

Reverb: signal flow

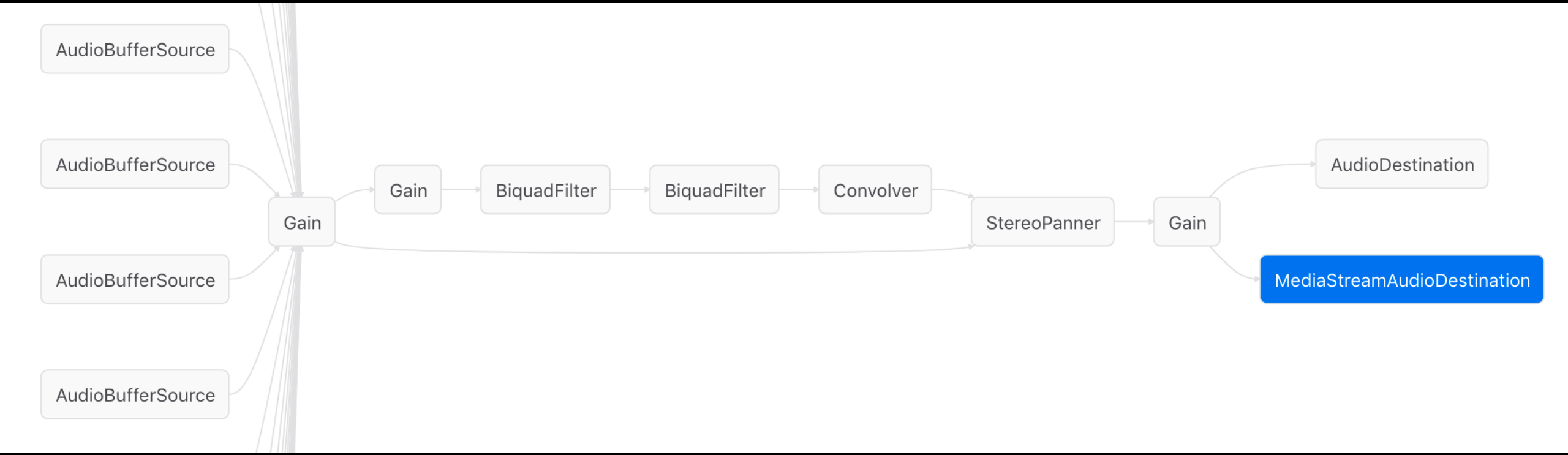
Recording

```
const mediaStream = actx.createMediaStreamDestination();
const recorder = new MediaRecorder(mediaStream.stream);
master.connect(mediaStream);

// when ready
recorder.start();

// when done
recorder.stop();
```

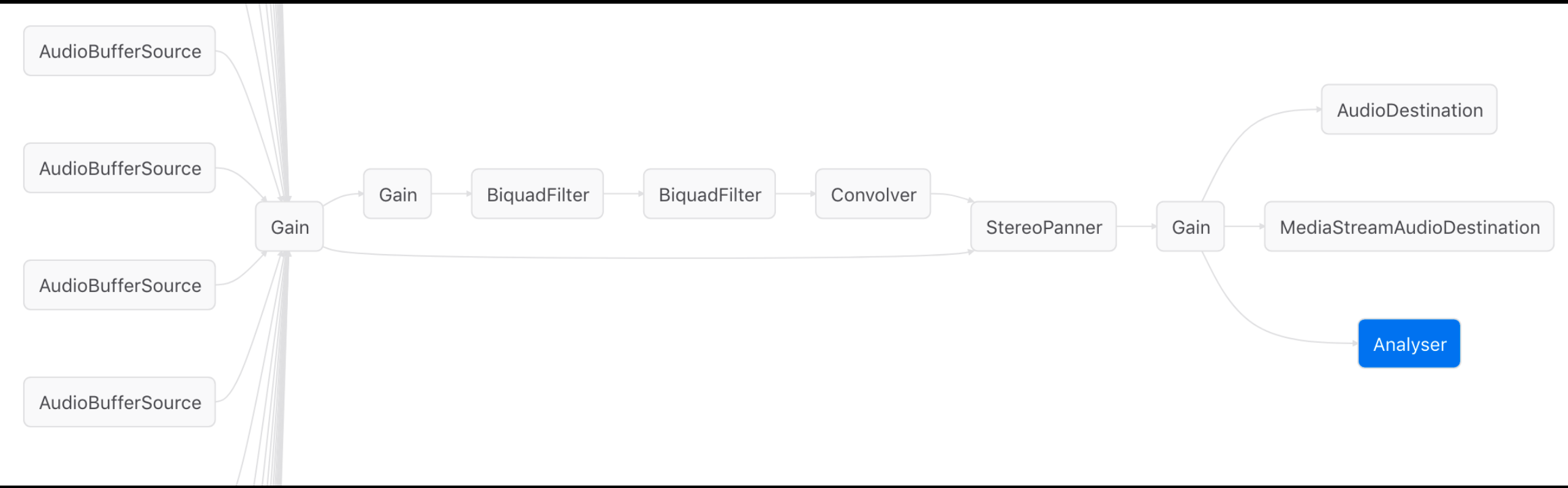
Recording

```
const chunks = [];
recorder.ondataavailable = evt => chunks.push(evt.data);
recorder.onstop = evt => {
  const blob = new Blob(chunks, {type: 'audio/ogg'});
  const audio = document.createElement('audio');
  audio.controls = true;
  audio.src = URL.createObjectURL(blob);
};
```

Armed for recording



Visualizations



Visualizations

```
const analyser = audioContext.createAnalyser();
analyser.fftSize = 1024;
const bufferLength = analyser.frequencyBinCount;
let dataArray = new Float32Array(bufferLength);

// plug
master.connect(analyser);

// in a drawing function
analyser.getFloatFrequencyData(dataArray);
```

Final result

More to explore

- Spatial panning
- Offline context
- Compression
- OscilatorNode
- Distortion: WaveShaperNode

Thank you!

github.com/stoyan/deepnote
phpied.com/files/thx
@stoyanstefanov