Making Music With Shaders

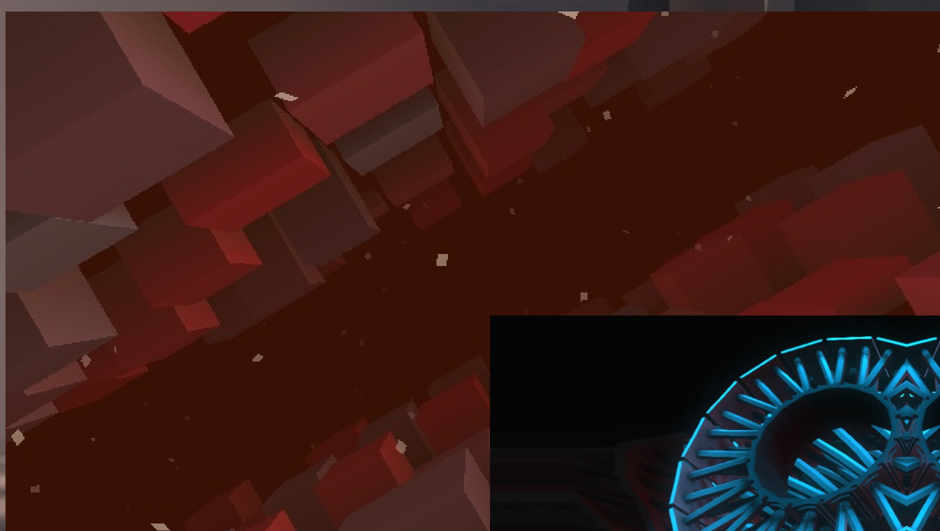
Practical additive GPU audio synthesis

@seecce
Pekka Väänänen

Who I Am

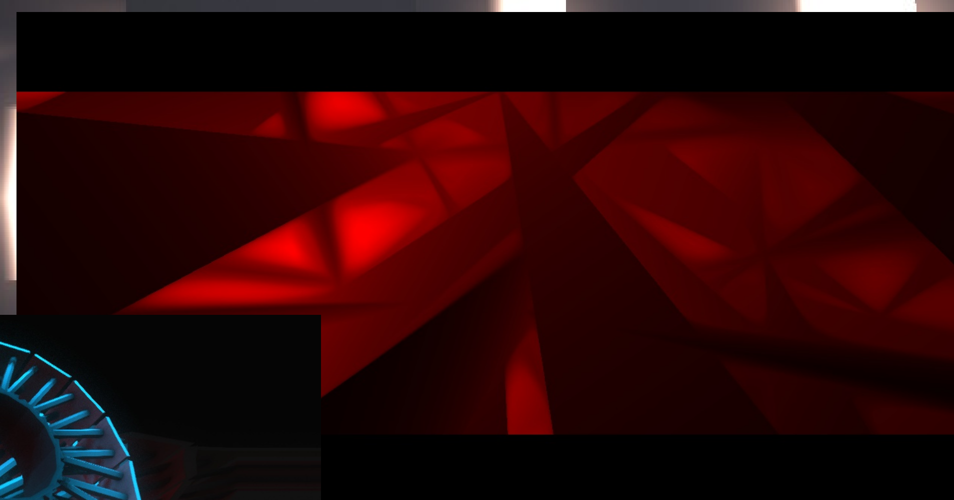
Pekka Väänänen a.k.a. cce/Peisik

PC Demoscener since ~2010



Archytas (2011)

Järjen Valo (2014)



Crimson (2014)

Who Are You?

- This talk is from a practical perspective.
  - The synth can be coded with just basic math skills.
- You should already be familiar with shaders :)
- Example code is written for ShaderToy.

Talk Structure

1. Motivation & Problem Statement
2. The Sine Wave
3. Harmony
4. Making Music & More Waveforms
5. The Phat Pad
6. Q&A

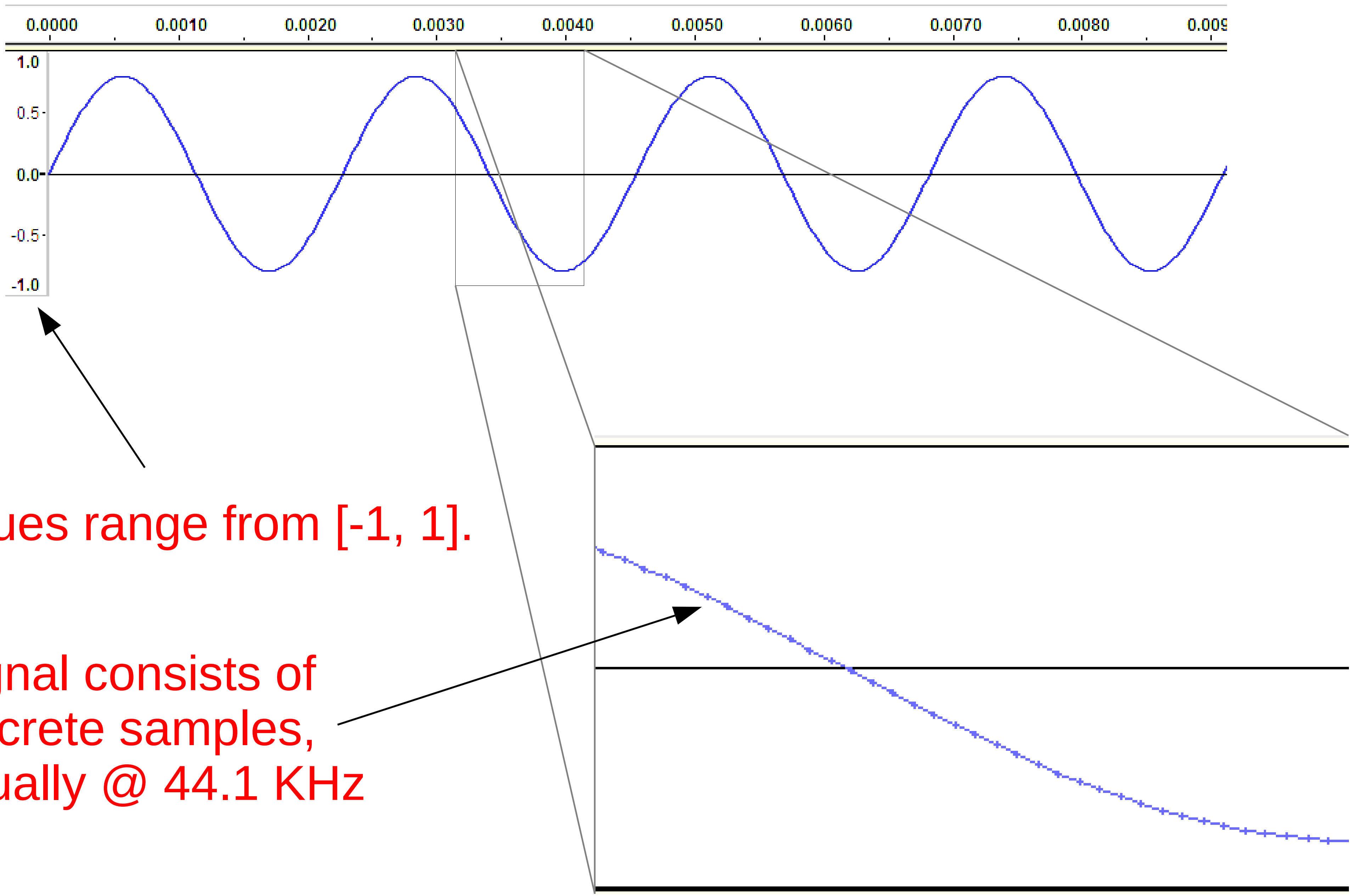
Problem Statement

- Write a simple synthesizer and a song that fit in small size.
- Do everything inside a single fragment shader.
  - This is exactly how ShaderToy does it!

(The final intro file size was ~22 KiB)

Pheromone (2016)

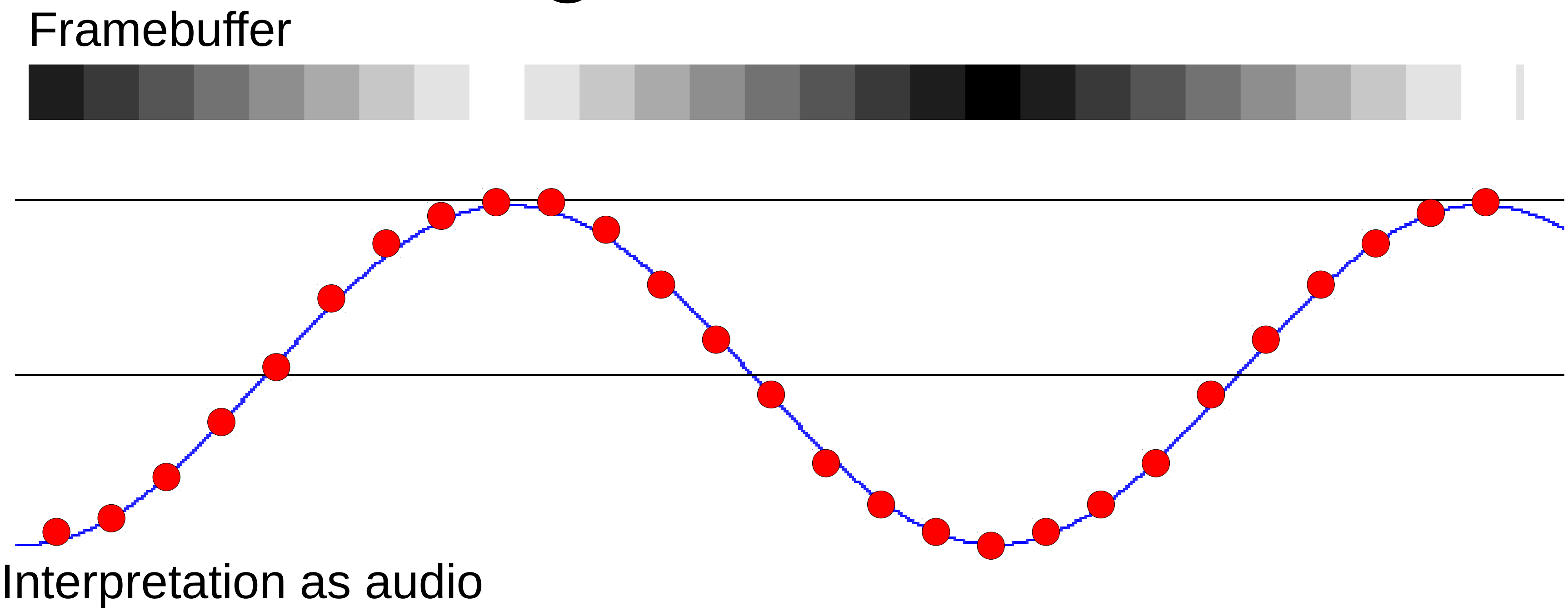
A Digital Waveform



Values range from [-1, 1].

Signal consists of discrete samples, usually @ 44.1 KHz

Fragment Shader

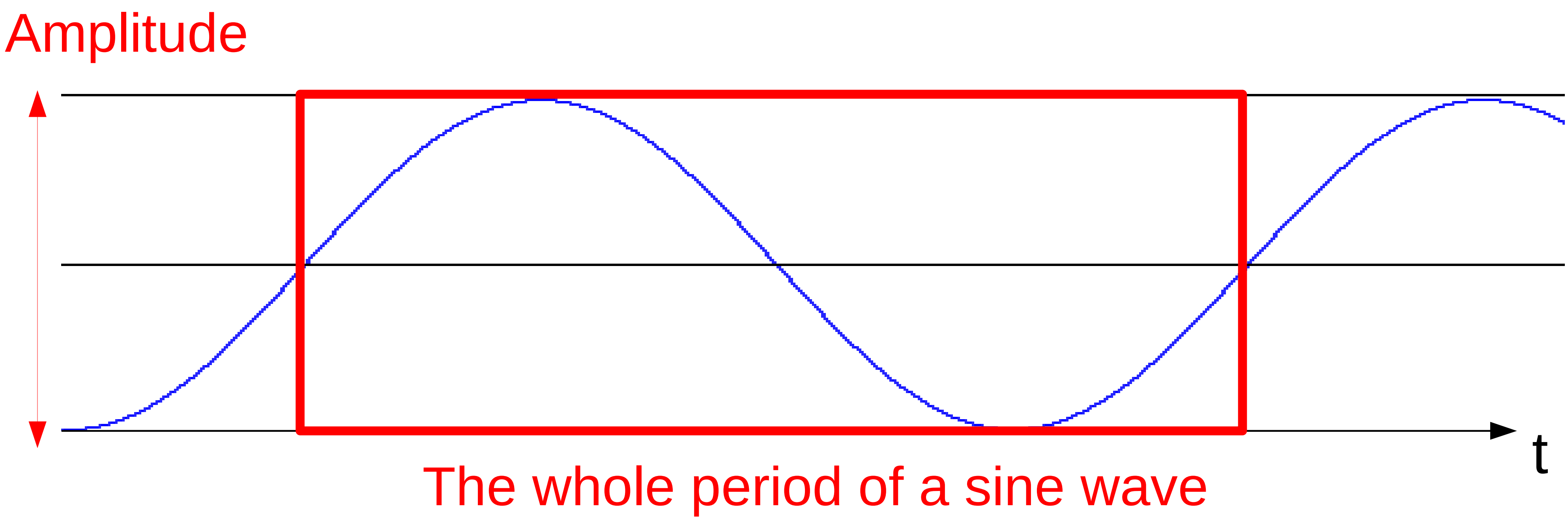


- Each pixel corresponds to one sample in the signal.
- Write to a 32-bit float image and play it as audio.

$$\sin(2\pi * t)$$

The period of a sine function

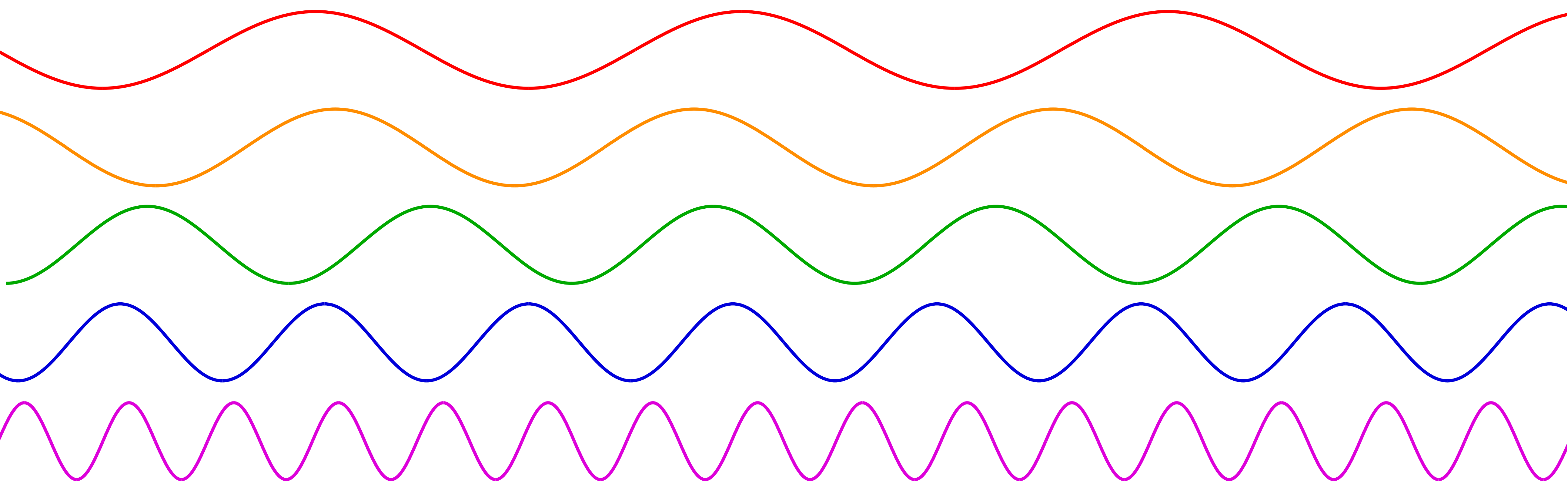
Time in seconds



A Sine Wave

Multiply time to get any frequency f (in hertz):

$$\sin(2\pi * t * f)$$



The Simplest Audio Shader

```
#define PI 3.1415926536
vec2 mainSound(float t)
{
    float s = sin(2.0*PI *440.0*t);
    return vec2(s);
}
```

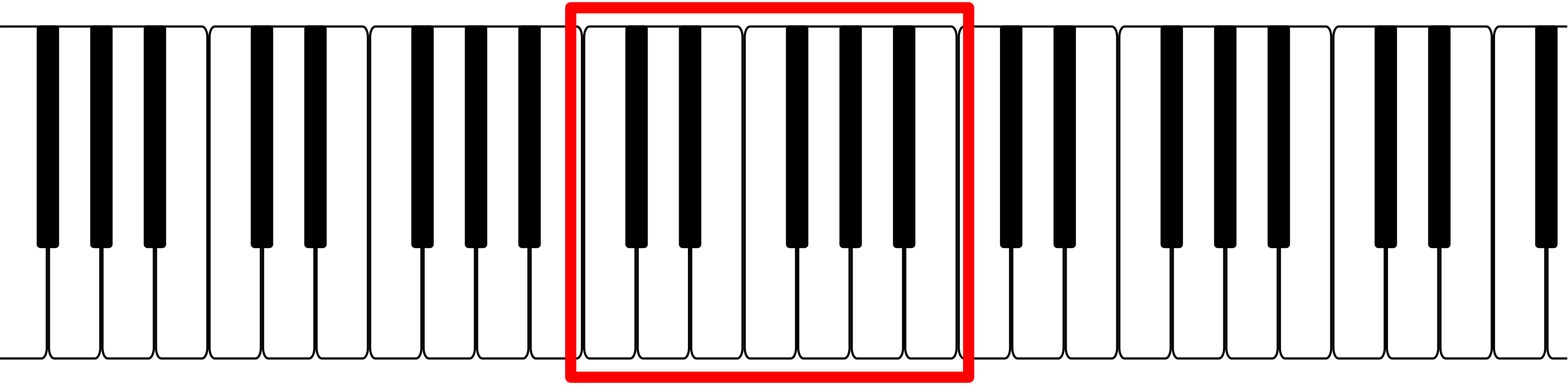
Time in seconds

Tone frequency in hertz

ShaderToy expects stereo audio

Source code on ShaderToy

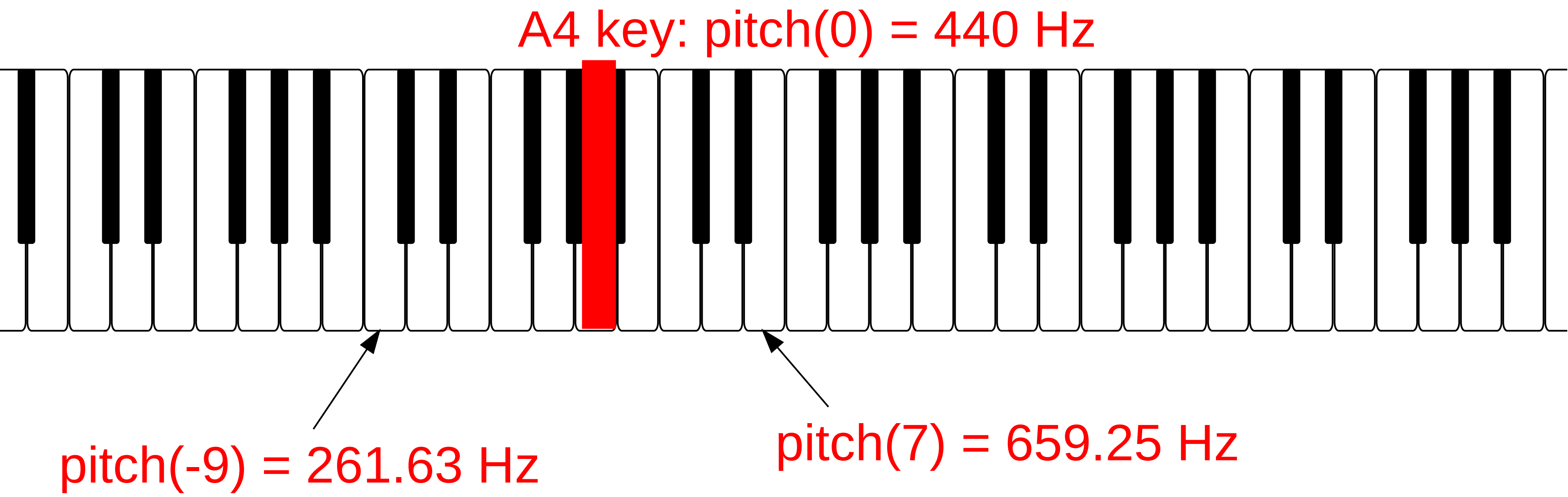
Measuring Music



- Each octave is divided into 12 *semitones*.
- Frequency doubles every octave.

Pitch To Frequency

- pitch(*p*) returns the frequency of the note *p.*
  - *p* is in semitones relative to A4.



Pitch To Frequency Formula

- pitch(*p*) returns the frequency of the note *p.*
  - *p* is in semitones relative to A4.

$$\text{pitch}(p) = \left(2^{\frac{1}{12}}\right)^{p} \cdot 440\,Hz$$

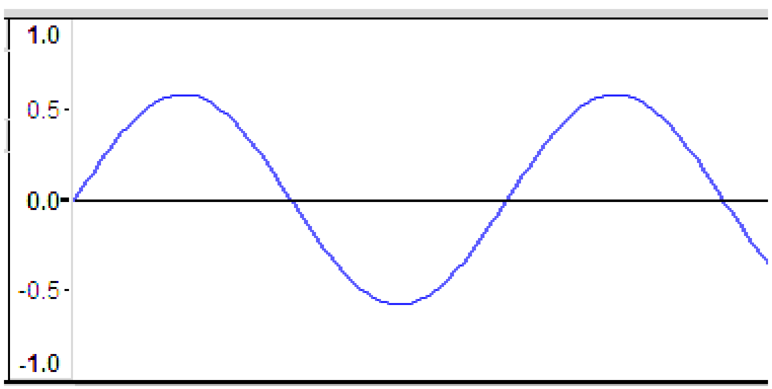
The 2nd Simplest Audio Shader

```
float pitch(float p) {
return pow(1.059460646483, p) * 440.0;
}

vec2 mainSound(float t) {
    float f = pitch(0.0); // Play A4 note.
    float s = sin(2.0*PI * f * t);
    return vec2(s);
}
```

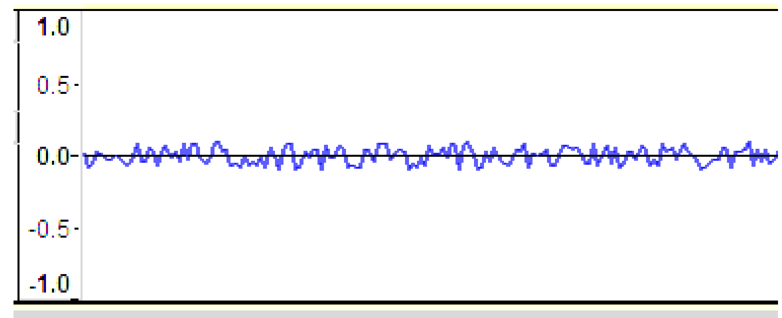
Source code on ShaderToy

Mixing Signals

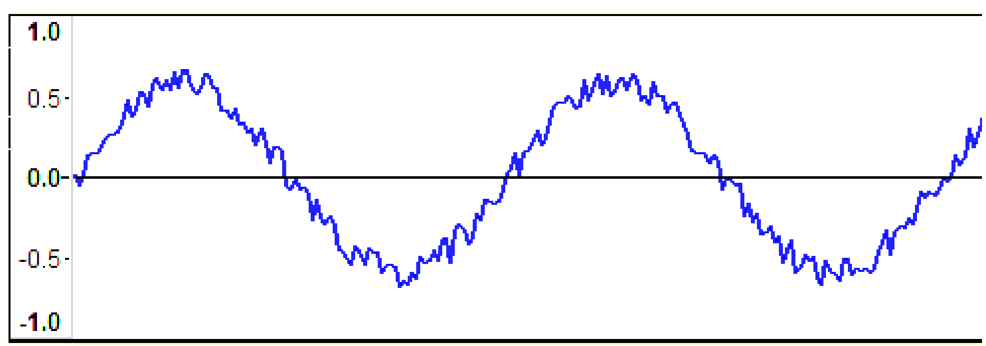
Mixing two signals means just adding them together.



+

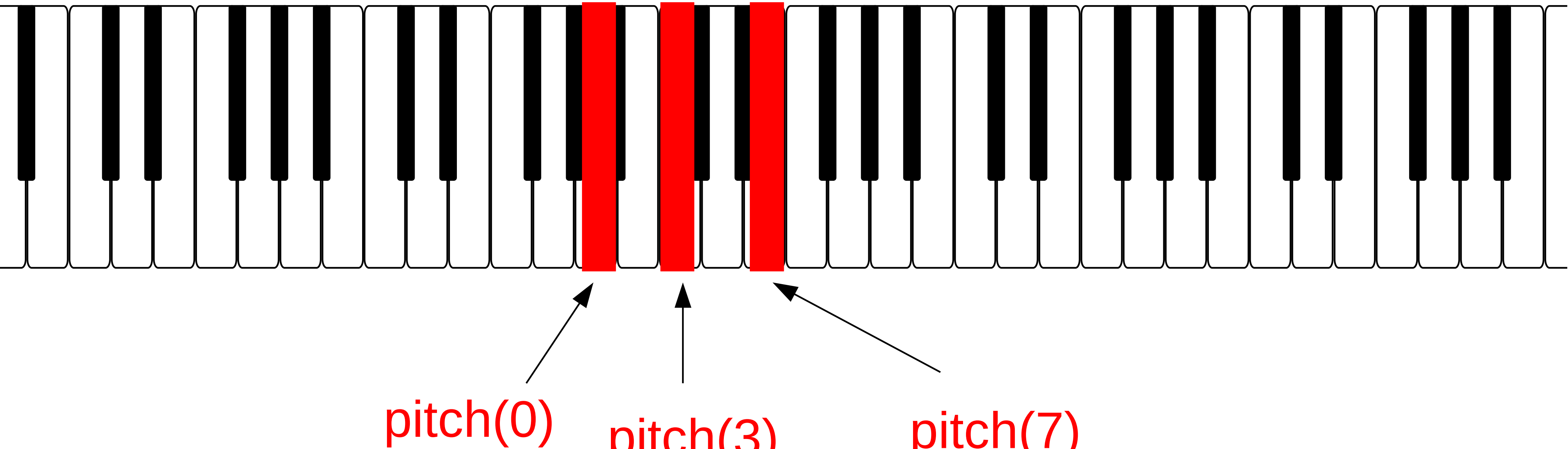


=



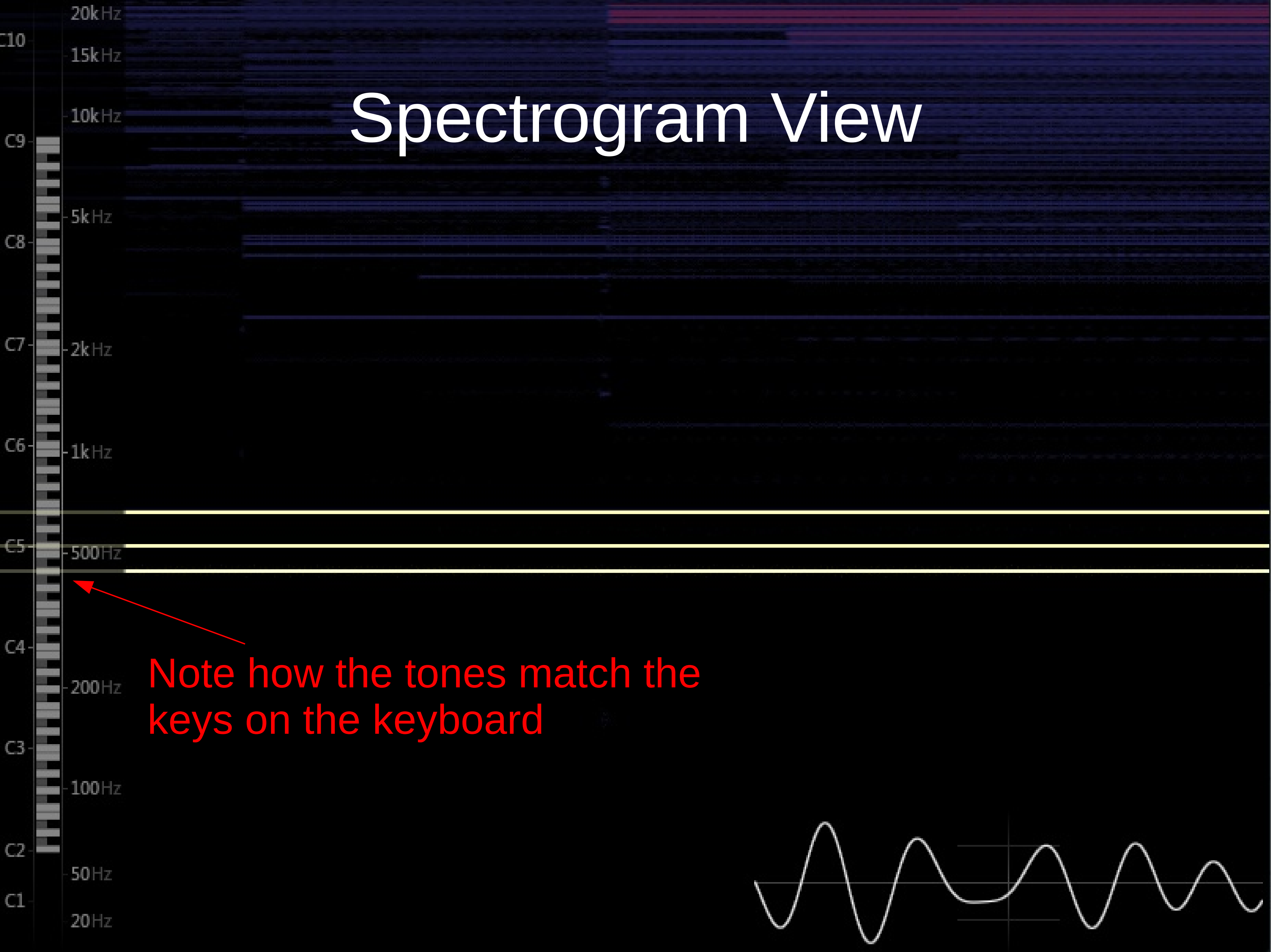
Semitones of a Chord

Let's see how an A-minor chord is played...



Aha! Let's just play tones 0, 3 and 7 at the same time.

Spectrogram View



Note how the tones match the keys on the keyboard

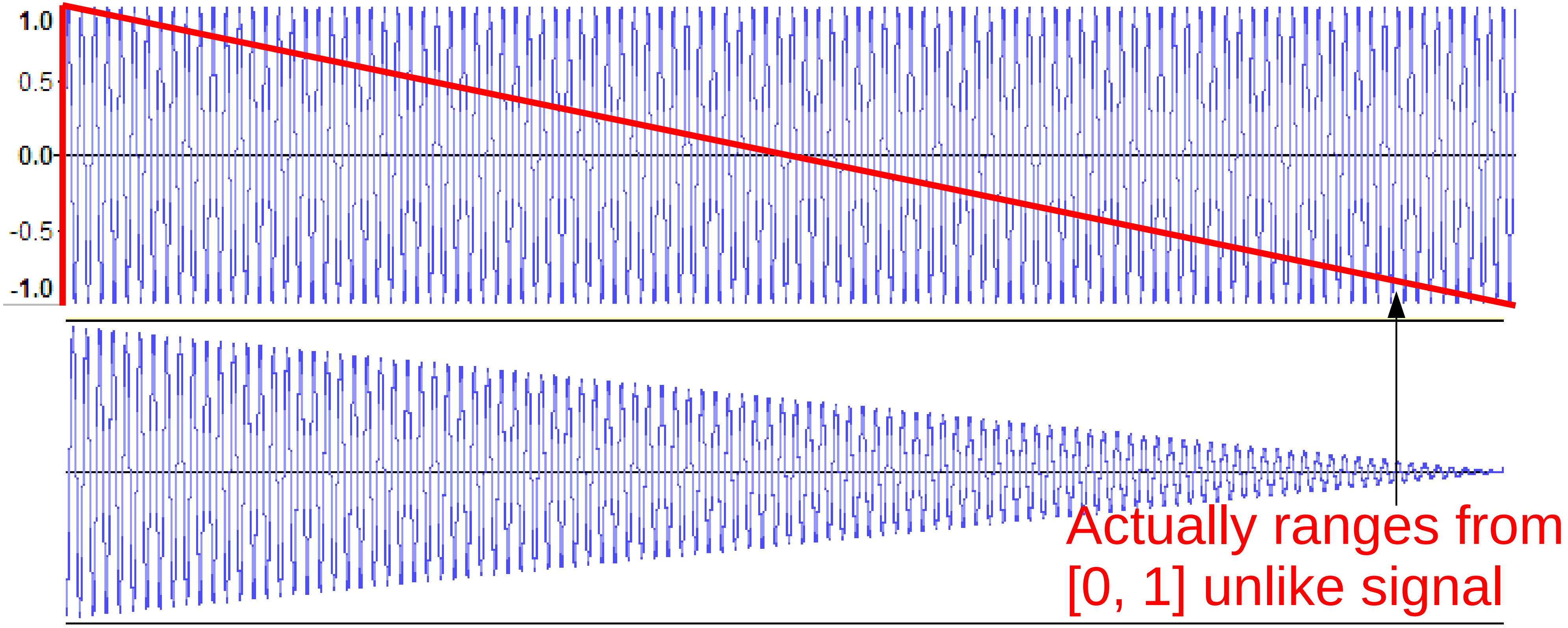
A-minor chord in GLSL

The three semitone offsets from last slide

```
vec2 mainSound(float t) {
    float s =
          sin(2.0*PI * pitch(0.0) * t)
        + sin(2.0*PI * pitch(3.0) * t)
        + sin(2.0*PI * pitch(7.0) * t);

    return vec2(s * 0.3); // lower volume
}
```

Source code on ShaderToy

Linear Envelopes

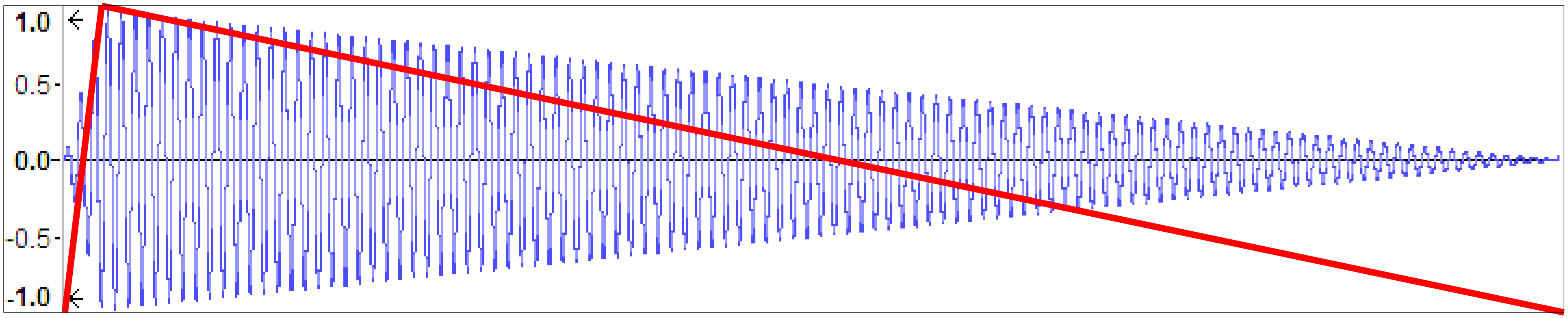


`max(0, 1 – t) * signal`

- To repeat once per second use `fract`:

  `max(0, 1 – fract(t)) * signal`

Two Linear Envelopes



- Add some fade-in:

```
min(1, t) * signal
```

- Add a multiplier for faster rise:

```
min(1, t * 40) * signal
```

Envelopes Example

```
vec2 mainSound( float t ) {
    float f = pitch(0.0); // Play A4 note.
    float s = sin(2.0*PI * f * t);
    // Decay (fade out)
    s *= max(0., 1.0 - fract(t));
    // Attack (fade in)
    s *= min(1.0, fract(t)*40.0);
    return vec2( s );
}
```

Can be used to fade in/out individual notes, instrument tracks or whole song parts

Source code on ShaderToy

Additive Synthesis

- Fourier theorem: We can construct complex periodic functions by just summing sines.
- Classic examples:

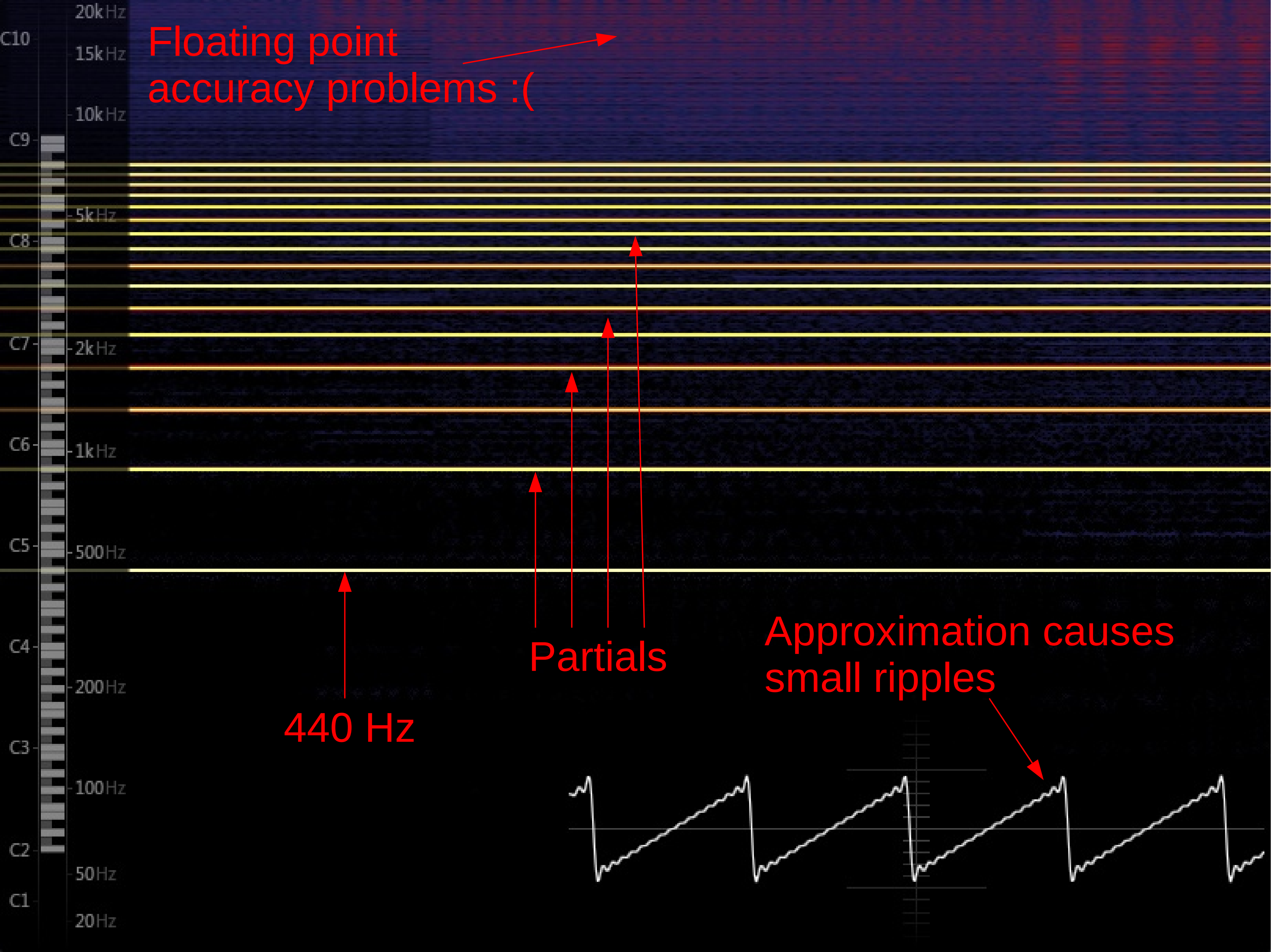
  triangle, sawtooth, square

A Boring Saw Wave

Only eight partials here, add more for a closer approximation.

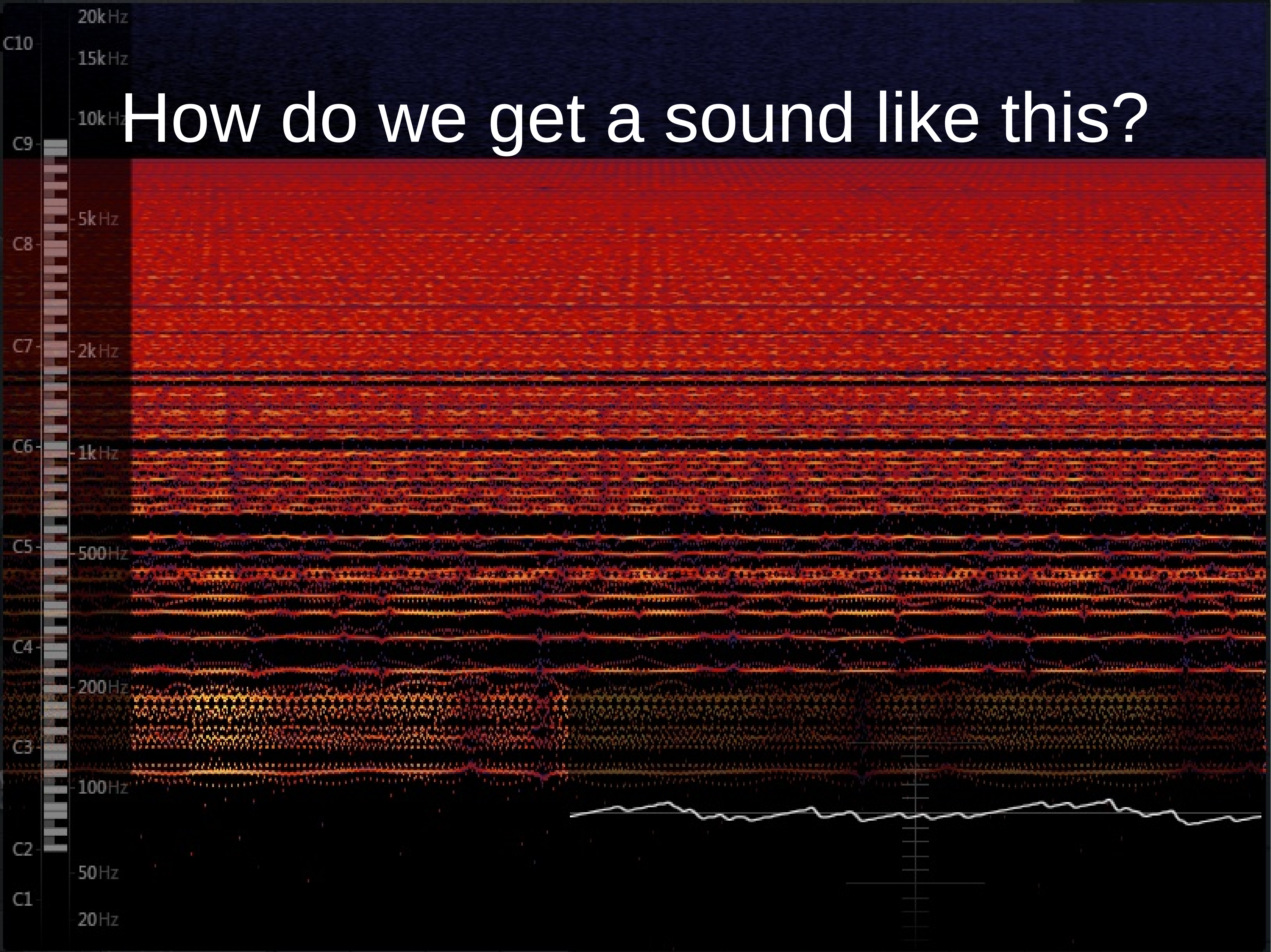
```
float saw(float phase) {
    float s = 0.0;
    for (int k = 1; k <= 8; k++) {
        s += (sin(2.0*PI*float(k)*phase) / float(k));
    }
    return (1.0/2.0) - (1.0/PI)*s - 0.5;
}
vec2 mainSound(float t) {
    float s = saw(t*440.0) * 0.8;
    return vec2(s);
}
```

Source code on ShaderToy



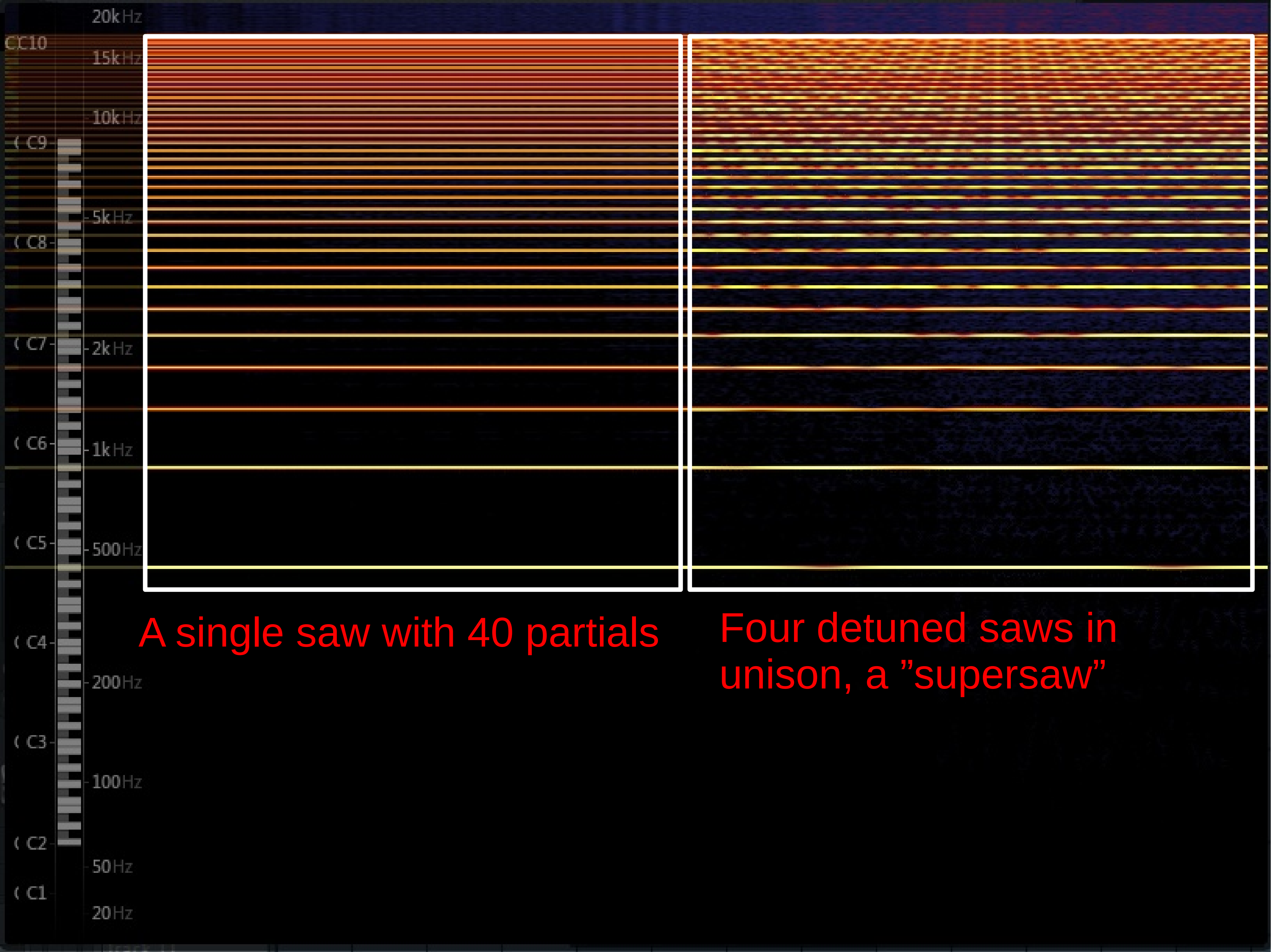
Floating point
accuracy problems :(
Partials
440 Hz
Approximation causes
small ripples
20k Hz
15k Hz
10k Hz
5k Hz
2k Hz
1k Hz
500 Hz
200 Hz
100 Hz
50 Hz
20 Hz
C10
C9
C8
C7
C6
C5
C4
C3
C2
C1

How do we get a sound like this?



Plan of Attack

- Play multiple detuned sawtooths in *unison*.
  - The slight frequency difference gives a warm oscillation effect.
- Just a few of them is enough though.


20k Hz
C10
15k Hz
10k Hz
C9
5k Hz
C8
C7
2k Hz
C6
1k Hz
C5
500 Hz
C4
200 Hz
C3
100 Hz
C2
50 Hz
C1
20 Hz
A single saw with 40 partials
Four detuned saws in unison, a ”supersaw”

The Phat Pad

```
vec2 mainSound(float t) {
    float s = 0.0; float semitones[4];
    semitones[0] = 0.0; semitones[1] = 4.0;
    semitones[2] = 7.0; semitones[3] = 9.0;

    const int VOICES = 4;
    for (int i = 0 ; i < 4; i++) {
        float f = pitch(-24.0 + semitones[i]);
        const int UNISON = 4;
        for (int u = 0; u < UNISON; u++) {
            float fu = float(u);
            float new_f = f + fu * sin(fu);
             s += saw(t * new_f) * (1.0/float(UNISON))
                                  * (1.0/float(VOICES));
        }
    }
    return vec2(s);
}
```

Source code on ShaderToy

The Phat Pad

```
vec2 mainSound(float t) {
    float s = 0.0; float semitones[4];
    semitones[0] = 0.0; semitones[1] = 4.0;
    semitones[2] = 7.0; semitones[3] = 9.0;

    const int VOICES = 4;
    for (int i = 0 ; i < 4; i++) {
        float f = pitch(-24.0 + semitones[i]);
        const int UNISON = 4;
        for (int u = 0; u < UNISON; u++) {
            float fu = float(u);
            float new_f = f + fu * sin(fu);
             s += saw(t * new_f) * (1.0/float(UNISON))
                                  * (1.0/float(VOICES));
        }
    }
    return vec2(s);
}
```

Store four semitone offsets that make up a chord

Source code on ShaderToy

The Phat Pad

```
vec2 mainSound(float t) {
    float s = 0.0; float semitones[4];
    semitones[0] = 0.0; semitones[1] = 4.0;
    semitones[2] = 7.0; semitones[3] = 9.0;

    const int VOICES = 4;
    for (int i = 0 ; i < 4; i++) {
        float f = pitch(-24.0 + semitones[i]);
        const int UNISON = 4;
        for (int u = 0; u < UNISON; u++) {
            float fu = float(u);
            float new_f = f + fu * sin(fu);
             s += saw(t * new_f) * (1.0/float(UNISON))
                                  * (1.0/float(VOICES));
        }
    }
    return vec2(s);
}
```

Loop through the four voices

Read pitch from array

Source code on ShaderToy

The Phat Pad

```
vec2 mainSound(float t) {
    float s = 0.0; float semitones[4];
    semitones[0] = 0.0; semitones[1] = 4.0;
    semitones[2] = 7.0; semitones[3] = 9.0;

    const int VOICES = 4;
    for (int i = 0 ; i < 4; i++) {
        float f = pitch(-24.0 + semitones[i]);
        const int UNISON = 4;
        for (int u = 0; u < UNISON; u++) {
            float fu = float(u);
            float new_f = f + fu * sin(fu);
             s += saw(t * new_f) * (1.0/float(UNISON))
                                  * (1.0/float(VOICES));
        }
    }
    return vec2(s);
}
```

Process the unison voices

A hacky detune term

Source code on ShaderToy

The Phat Pad

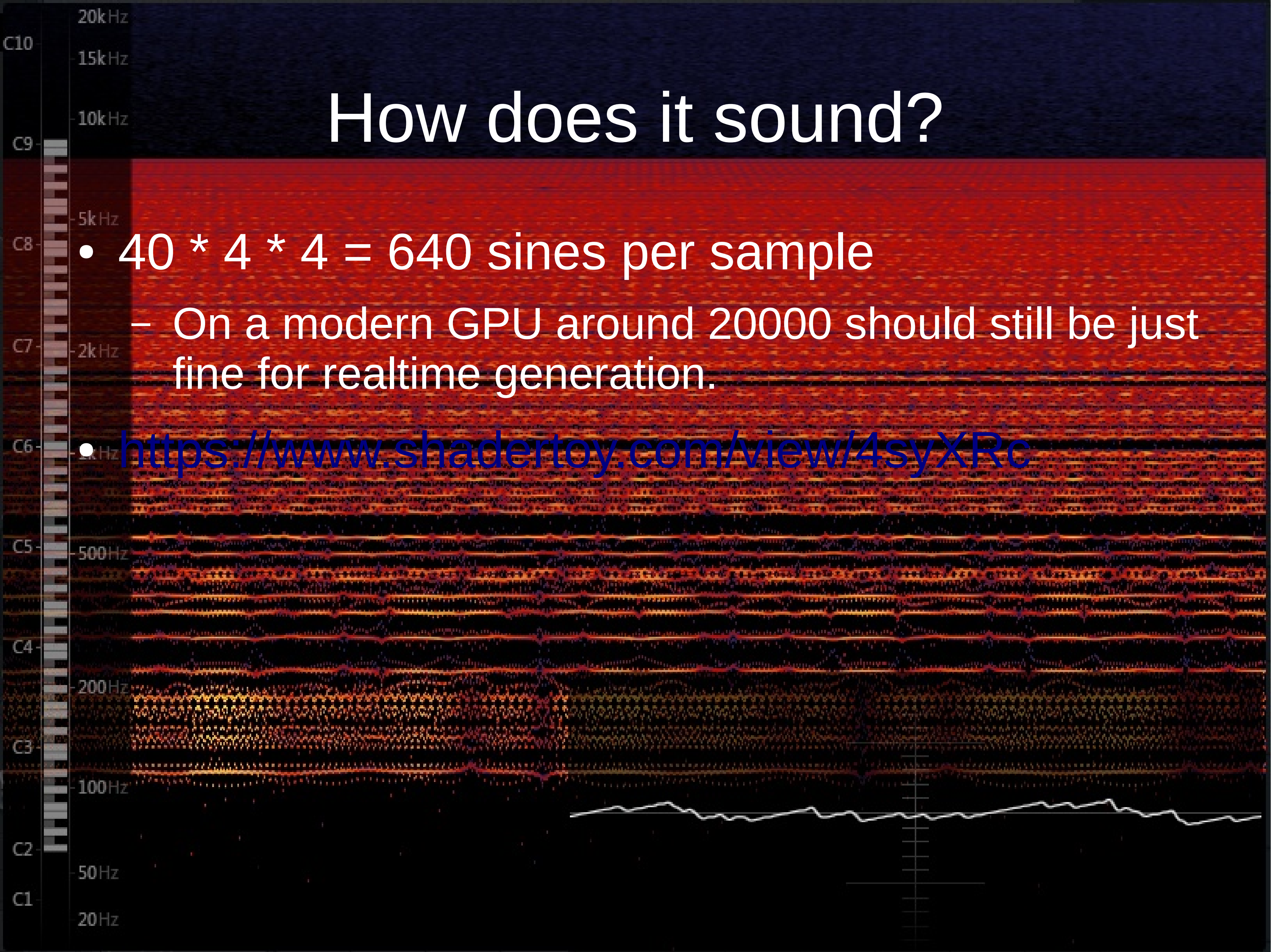
```
vec2 mainSound(float t) {
    float s = 0.0; float semitones[4];
    semitones[0] = 0.0; semitones[1] = 4.0;
    semitones[2] = 7.0; semitones[3] = 9.0;

    const int VOICES = 4;
    for (int i = 0 ; i < 4; i++) {
        float f = pitch(-24.0 + semitones[i]);
        const int UNISON = 4;
        for (int u = 0; u < UNISON; u++) {
            float fu = float(u);
            float new_f = f + fu * sin(fu);
             s += saw(t * new_f) * (1.0/float(UNISON))
                                 * (1.0/float(VOICES));

        }
    }
    return vec2(s);
}
```

Normalize so amplitudes add up to one, a bit hacky

Source code on ShaderToy

How does it sound?

- 40 * 4 * 4 = 640 sines per sample
  - On a modern GPU around 20000 should still be just fine for realtime generation.
- https://www.shadertoy.com/view/4syXRc



Practical Problems

- Floating point accuracy!
  - Once the timer gets big enough there just aren't enough bits left in the significand and we get noise.
  - Bearable for < 3 minutes of audio.
- Hard to compose an elaborate song with GLSL.
  - Easier to stick with repeating patterns.
- Destructive interference
  - Can't just stack a ton of unison voices, all you get is noise...

Things to Add

- Many subjects I didn't cover:
  - reverbation & stereo audio
  - percussion
  - more elaborate waveforms
    - (sawtooth waves are kinda lame)
  - multiple passes to combat floating point errors
  - human voice synth with formants
  - perfect filters

Thanks!

Any questions?

Bonus Slides

A Boring Square Wave

```
float square(float phase) {
    float s = 0.0;
    for (int k=1; k<8; k++) {
        s += sin(2.0 * PI * (2.0*float(k)-1.0) * phase)
            / (2.0 * float(k) - 1.0);
    }
    return (4.0 / PI) * s;
}
vec2 mainSound(float t) {
    float s = square(t*440.0) * 0.8;
    return vec2(s);
}
```

Source code on ShaderToy

Pheromone Synth GLSL Source

- Pretty messy but it's built on the basic concepts shown here in this presentation.
- Requires desktop OpenGL, doesn't work with WebGL.
- Writes to a RGB8 framebuffer so packing is needed at output.
- synth.glsl