WebChucK: Computer Music Programming on the Web

Michael R. Mulshine
CCRMA, Stanford University
Stanford, CA, United States
mulshine@ccrma.stanford.edu

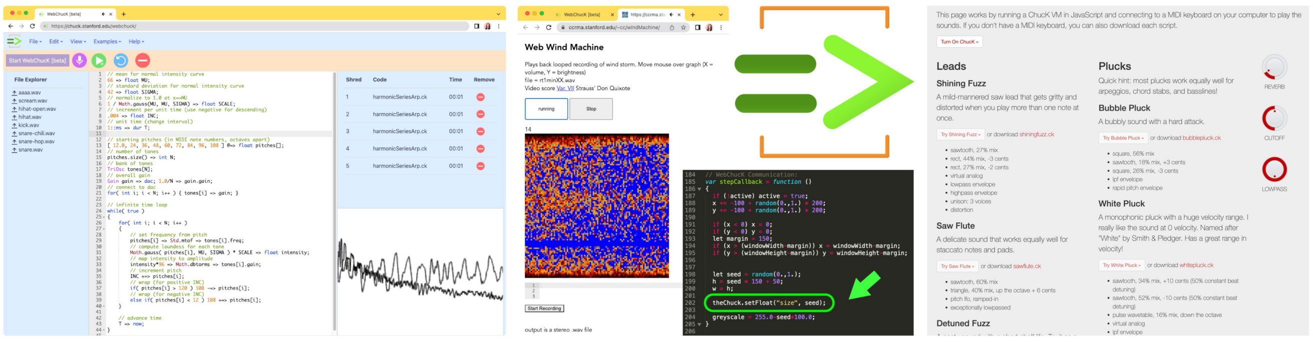
Ge Wang
CCRMA, Stanford University
Stanford, CA, United States
ge@ccrma.stanford.edu

Jack Atherton
CCRMA, Stanford University
Stanford, CA, United States
lja@ccrma.stanford.edu

Chris Chafe
CCRMA, Stanford University
Stanford, CA, United States
cc@ccrma.stanford.edu

Terry Feng
CCRMA, Stanford University
Stanford, CA, United States
tzfeng@ccrma.stanford.edu

Celeste Betancur
CCRMA, Stanford University
Stanford, CA, United States
celes@ccrma.stanford.edu



```
// mean for normal intensity curve
66 => float MU;
// standard deviation for normal intensity curve
42 => float SIGMA;
// normalize to 1.0 at x==MU
1 / Math.gauss(MU, MU, SIGMA) => float SCALE;
// increment per unit time (use negative for descending)
.004 => float INC;
// unit time (change interval)
1::ms => dur T;

// starting pitches (in MIDI note numbers, octaves apart)
[ 12.0, 24, 36, 48, 60, 72, 84, 96, 108 ] @=> float pitches[];
// number of tones
pitches.size() => int N;
// bank of tones
TriOsc tones[N];
// overall gain
Gain gain => dac; 1.0/N => gain.gain;
// connect to dac
for( int i; i < N; i++ ) { tones[i] => gain; }

// infinite time loop
while( true )
{
    for( int i; i < N; i++ )
    {
        // set frequency from pitch
        pitches[i] => Std.mtof => tones[i].freq;
        // compute loudness for each tone
        Math.gauss( pitches[i], MU, SIGMA ) * SCALE => float intensity;
        // map intensity to amplitude
        intensity*96 => Math.dbtorms => tones[i].gain;
        // increment pitch
        INC +=> pitches[i];
        // wrap (for positive INC)
        if( pitches[i] > 120 ) 108 -=> pitches[i];
        // wrap (for negative INC)
        else if( pitches[i] < 12 ) 108 +=> pitches[i];
    }

    // advance time
    T => now;
}
```

Shred	Code	Time	Remove
1	harmonicSeriesArp.ck	00:01	●
2	harmonicSeriesArp.ck	00:01	●
3	harmonicSeriesArp.ck	00:01	●
4	harmonicSeriesArp.ck	00:01	●
5	harmonicSeriesArp.ck	00:01	●

Web Wind Machine

Plays back looped recording of wind storm. Move mouse over graph (X = volume, Y = brightness)
file = rt1minXX.wav
Video score Var. VII Strauss' Don Quixote

output is a stereo .wav file

```
// WebChucK Communication:
var stepCallback = function ()
{
  if (!active) active = true;
  x += -100 + random(0.,1.) * 200;
  y += -100 + random(0.,1.) * 200;

  if (x < 0) x = 0;
  if (y < 0) y = 0;
  let margin = 150;
  if (x > (windowWidth-margin)) x = windowWidth-margin;
  if (y > (windowHeight-margin)) y = windowHeight-margin;


  let seed = random(0.,1.);
  h = seed * 150 + 50;
  w = h;

  theChuck.setFloat("size", seed);

  greyscale = 255.0-seed*100.0;
}
```

This page works by running a ChucK VM in JavaScript and connecting to a MIDI keyboard on your computer to play the sounds. If you don't have a MIDI keyboard, you can also download each script.

Leads

Shining Fuzz

A mild-mannered saw lead that gets gritty and distorted when you play more than one note at once.

or download shiningfuzz.ck

- sawtooth, 27% mix
- rect, 44% mix, -3 cents
- rect, 27% mix, -2 cents
- virtual analog
- lowpass envelope
- highpass envelope
- unison: 3 voices
- distortion

Saw Flute

A delicate sound that works equally well for staccato notes and pads.

or download sawflute.ck

- sawtooth, 60% mix
- triangle, 40% mix, up the octave + 6 cents
- pitch lfo, ramped-in
- exceptionally lowpassed

Detuned Fuzz

Plucks

Quick hint: most plucks work equally well for arpeggios, chord stabs, and basslines!

Bubble Pluck

A bubbly sound with a hard attack.

or download bubblepluck.ck

- square, 56% mix
- sawtooth, 18% mix, +3 cents
- square, 26% mix, -3 cents
- lpf envelope
- rapid pitch envelope

White Pluck

A monophonic pluck with a huge velocity range. I really like the sound at 0 velocity. Named after "White" by Smith & Pledger. Has a great range in velocity!

or download whitepluck.ck

- sawtooth, 34% mix, +10 cents (50% constant beat detuning)
- sawtooth, 52% mix, -10 cents (50% constant beat detuning)
- pulse wavetable, 16% mix, down the octave
- virtual analog
- lpf envelope

Figure 1: From left to right: WebChucK IDE, A browser-based "wind machine," controlling ChucK from Javascript code, web-hosted virtual instruments written in ChucK.

ABSTRACT

WebChucK is ChucK—a *strongly-timed* computer music programming language—running on the web. Recent advancements in browser technology (including WebAssembly and the Web Audio API's `AudioWorklet` interface) have enabled languages written in C/C++ (like ChucK) to run in web browsers with nearly native-code performance. Early adopters have explored the many practical and creative possibilities that WebChucK enables, ranging from a WebChucK integrated development environment to interactive browser-based audiovisual experiences. WebChucK has also been adopted as the programming platform in an introductory computer music course at Stanford University. Importantly, by running in any browser, WebChucK broadens and simplifies access to computer music programming, opening the door for new users and creative workflows. In this paper, we discuss WebChucK and its applications to date, explain how the tool was designed and implemented, and evaluate the unique affordances of combining computer music programming with a web development workflow.

WebChucK portal:
`https://chuck.stanford.edu/webchuck/`

NIME'23, 31 May–2 June, 2023, Mexico City, Mexico.

Author Keywords

WebChucK, ChucK, audio programming, web programming, web audio, browser IDE, accessibility

CCS Concepts

• **Applied computing** → **Sound and music computing;** • **Information systems** → *Web interfaces;* • **Human-centered computing** → *Systems and tools for interaction design;*

1. WHAT IS WEBCHUCK?

Computer music programming languages began on mainframe computers and evolved their way into desktops, laptops, mobile phones, and more. But until recently, fully-programmable audio synthesis on the web was not feasible, primarily due to limitations of the web platform. Browsers have not been powerful enough to tackle high CPU-load tasks like digital signal processing and other low-level audio code (often written in C/C++, which browsers don't natively support). On top of this, lack of cross-compatibility between browsers has limited the scope and potential impact of web-based audio development.

Fortunately, advancements and widespread adoption of key features of the Web Audio API[10] (such as the `AudioWorklet`[1] interface) and other tools (like `emscripten`[2], which enable C/C++ libraries to be compiled into We-

[1]`https://developer.mozilla.org/en-US/docs/Web/API/AudioWorklet`
[2]`https://emscripten.org/`

bAssembly[3] binaries) have opened a pathway for computer music languages written in C and C++ to run on the web.[8][9][7] Additionally, some intrepid exploration of what it means to do audio programming on the web has led to the creation of a slew of tools and web experiences.

With WebChucK, we merge the low-level control capabilities and expressiveness of the computer music programming language ChucK with the ease, accessibility, interactivity, portability, and ubiquity offered by the web. Specifically, this introduces several advantages:

1. WebChucK brings a fully-featured computer music programming to the web, and along with it capabilities such as unit generators, unit analyzers, audio DSP, music interaction design tools

2. It combines the sample-synchronous, *strongly-timed*, and concurrent features of the ChucK language with the workflow and trappings (web pages, web GUIs, high performance graphics, collaboration) of the web

3. Like other web-based programming tools, it broadens the reach and accessibility of audio and highly-synchronized audiovisual programming for educators, artists, and audiences around the world

This paper will be most useful for users of ChucK, but also highlights (via use-case examples) some of the key affordances that audio programming on the web, in general, may bring to the community. To learn more about ChucK, get started on the ChucK website.[4]

1.1 A Brief History of WebChucK

The ChucK programming language [17] was initially released in 2003 [16], marking this year as the 20th anniversary of ChucK. Over this period, ChucK has been used in sound synthesis, instrument design, laptop orchestras, mobile music apps, audiovisual and VR design, generative music systems, and many more artistic, educational, and research contexts. The defining features of ChucK are:

- A sample-precise unified timing mechanism for multi-rate event and control processing. This *strongly-timed* property of the language effectively removes the distinction between audio and control rate traditionally found in languages such as CSound, Max/MSP, and SuperCollider.

- A simple but powerful concurrent programming model based on time—allowing for audio programs to be expressed precisely and in parallel.

- A live coding environment (and way of thinking) that supports rapid experimentation, teaching/learning, and performance.

ChucK was designed with the idea that the form of a tool shapes the way we think [15]. ChucK exists in various forms including command-line `chuck`, miniAudicle (a graphical IDE; [13]), ChiP (ChucK on the iPhone; [14]), FaucK (ChucK in FAUST; [18]), Chunity (ChucK in Unity; [5]), and more recently, ChAI (ChucK for AI, in early development) and WebChucK. The development of the latter resulted from a collective effort:

- After a number of attempts by the ChucK community to compile the C++ code base from ChucK to WebAssembly using `emscripten`, Jack Atherton, then a Ph.D. candidate in Ge Wang's Music, Computing, Design research group, achieved this in 2020 and devised the core functionality of WebChucK.

- Soon after, Mike Mulshine began creating web-based audiovisual art, tutorials, and demos[5] using WebChucK and *p5.js*, and organized ambassadorial efforts to bring WebChucK to new audiences.

- In 2022, Chris Chafe transitioned from using ChucK to WebChucK as a teaching tool at Stanford University's popular Music 220a course "The Foundations of Computer-Generated Sound."[6]

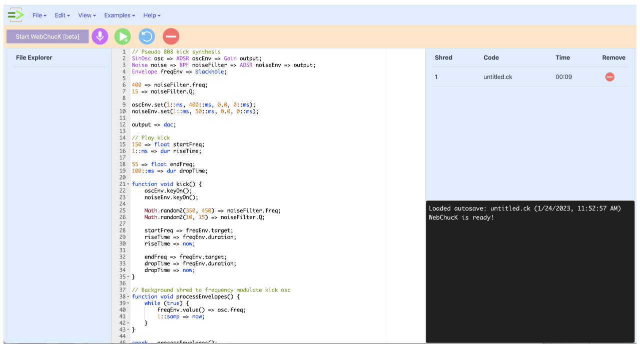
- Terry Feng and Celeste Betancur developed the WebChucK IDE in 2022, and designed tools for real-time visualization and code-based GUI-generation for the IDE.

1.2 Using WebChucK

There are two general approaches to using WebChucK:

1. As a **web-based programming tool** for sound design, interaction design, composition, and teaching

2. As a **component** in larger web-based artworks, tools, and games.



```
// Pseudo 808 kick synthesis
SinOsc osc => ADSR oscEnv => Gain output;
Noise noise => BPF noiseFilter => ADSR noiseEnv => output;
Envelope freqEnv => blackhole;

400 => noiseFilter.freq;
15 => noiseFilter.Q;

oscEnv.set(1::ms, 400::ms, 0.0, 0::ms);
noiseEnv.set(1::ms, 50::ms, 0.0, 0::ms);

output => dac;

// Play kick
150 => float startFreq;
1::ms => dur riseTime;

55 => float endFreq;
100::ms => dur dropTime;

function void kick() {
    oscEnv.keyOn();
    noiseEnv.keyOn();

    Math.random2(350, 450) => noiseFilter.freq;
    Math.random2(10, 15) => noiseFilter.Q;

    startFreq => freqEnv.target;
    riseTime => freqEnv.duration;
    riseTime => now;

    endFreq => freqEnv.target;
    dropTime => freqEnv.duration;
    dropTime => now;
}

// Background shred to frequency modulate kick osc
function void processEnvelopes() {
    while (true) {
        freqEnv.value() => osc.freq;
        1::samp => now;
    }
}

spork ~ processEnvelopes();
```

Shred	Code	Time	Remove
1	untitled.ck	00:09	⊖

```
Loaded autosave: untitled.ck (1/24/2023, 11:52:57 AM)
WebChucK is ready!
```

Figure 2: The WebChucK IDE running some 808 kick synthesis.

Using ChucK as a **web-based programming tool** entails simply navigating to the dedicated WebChucK IDE[7], pictured in Figure 2), and writing ChucK code. The IDE provides the ability to perform typical ChucK-environment actions like starting the virtual machine, adding and removing ChucK shreds, and printing to the ChucK console. This form factor is friendly to students, creative prototypers, and anyone wishing to "jump in" and start coding without installing any software. An activity monitor displays all currently running shreds and enables users to selectively add or remove them from the virtual machine. Audio and other data files can be loaded into WebChucK for use from code.

The second approach, using WebChucK as a **component** within a larger web-based project, involves embedding the ChucK WebAssembly (`Wasm`) module and JavaScript interface. The latter provides a suite of "getter" and "setter" methods to modify or retrieve global variables in ChucK from a browser script, enabling a two-way workflow in which web page elements can both control or be controlled by

[3]`https://webassembly.org/`
[4]`https://chuck.stanford.edu/`

[5]`https://mikemulshine.com/webchuck-demo/`
[6]`https://ccrma.stanford.edu/courses/220a/`
[7]`https://chuck.stanford.edu/webchuck/`

ChucK code. For example, it is possible to control audio synthesis parameters using GUI elements, mouse movement, keyboard input, and MIDI controllers. Working in tandem with JavaScript libraries (like *p5.js* or *Three.js*) and other web UI development platforms, WebChucK supports a tightly-synchronized audiovisual workflow for web-based art and other creative applications. Lastly, it is important to note that WebChucK runs entirely on the client-side browser, requiring no backend server-side support other than a static HTML web page.

1.3 Example Use Cases

To give a sense of WebChucK in action, here are some example use cases:

1. **Quick prototyping:** anyone with a supported browser can navigate to the online IDE
2. **Sharing:** Upon being asked to produce some wind sound for the Stanford Symphony Orchestra's performance of Richard Strauss's *Don Quixote*, Chris prototyped a generative wind machine with a scrolling interactive spectrogram with WebChucK and basic HTML/JavaScript. He simply sent the link to the Orchestra.[8] For context, compare these two methods:
   - **Existing method of sharing ChucK code**:
     - "Here is a ChucK file..."
     - "Go to this URL, download and install ChucK..."
     - "Open miniAudicle, start Virtual Machine, open the file, hit 'plus' (or run it in a terminal emulator)..."
     - "If it doesn't work, let us know!"[9]
   - **New method of sharing with WebChucK**:
     - "Here's a URL."
3. **Creating virtual instruments:** Connect a MIDI keyboard, and start playing immediately. Jack's Timbre Library[10] for ChucK (hosted on the web) provides several examples of web-based virtual synths in action.
4. **Teaching:** Students can start coding audio right away. WebChucK removes much of the technical overhead of operating systems, software installation, terminals, GUIs, and audio interfaces.
5. **Doing computer music in new ways:** In our experience, WebChucK isn't just a more portable version of ChucK, but affords some new functionalities and ways of working that didn't exist or weren't feasible before. We describe some examples of this in Section 4.

To learn more about how to embed WebChucK in your own web project, work through a few simple tutorials here: `https://chuck.stanford.edu/webchuck/tutorial/`

2. RELATED WORK

Three primary approaches to programming with audio on the web exist. The first approach involves embedding audio on a web page via the HTML `<audio>` element. This enables simple playback of audio files, triggered by user interaction with UI elements like buttons or `autoplay`. Dynamic real-time control of this audio is limited.

The second approach is to use and extend the Web Audio API. The *p5.js* library has implemented their own JavaScript wrapper around Web Audio called `p5.sound`[11], which provides a set of easy-to-use audio playback, synthesis, and analysis functions to web developers, like `p5.SoundFile`, `p5.Oscillator`, `p5.Envelope`, and more. *Three.js* provides an analogous tool in their `Audio` object, which primarily provides functions to manipulate the playback of audio files. Similarly, Tone.js[12] extends the Web Audio API and offers handy methods (e.g. via `Tone.Transport`) for dealing with musical timing in JavaScript.

More advanced and musically-tailored web-based audio platforms and live-coding languages have been developed by extending the Web Audio API, including Tidal Cycles[13] (a live-coding tool for music generation developed in Haskell and compiled into JavaScript[12]), Gibber[14] (a live audio programming language with a great depth of sound and music generation capabilities and unique text-animated graphical UI), CodeCircle[20] (shown to be a useful tool in prototyping instruments on the web), Estuary[3] (a live-coding platform that enables the simultaneous use of many live coding languages at once, as an "ensemble"), and Strudel[4] (an alternative to Tidal Cycles with an algorithmic composition focus). These useful and expressive tools each offer unique affordances for programming audio in a web-based IDE, live coding and performance, and audio development projects. Built in JavaScript around the Web Audio API, they make audio control more accessible for web developers and artists, despite lacking some of the low-level audio programming control (e.g. sample-by-sample synthesis) and flexibility of a full-fledged computer music programming languages like ChucK, Super Collider, Csound, and others.

The third approach involves porting an entire computer music language to the web using tools (e.g. `emscripten`) to create a WebAssembly binary and run it via Web Audio's `AudioWorklet` interface[6]. ChucK is not the only computer music programming language that now runs in this manner in the browser. Others include the functional audio signal processing language FAUST[1], with their custom IDE[2] tailored for compiling and exporting FAUST code as audio programs ready for use in a variety of other environments (from iOS apps to firmware on microcontrollers like Teensy). The Csound IDE[19][15] brings Barry Vercoe's Csound to the web as a user-friendly social audio programming environment. The graph-oriented live coding language (nicknamed Glicol[16]) compiles desktop Glicol into WebAssembly running via the AudioWorklet interface and provides a sleek and modern IDE. Additionally, there have been community efforts bring SuperCollider and Pure Data [17] to the web. The developers of ChucK are excited to join the fold in bringing audio programming languages to the web.

3. DESIGN AND IMPLEMENTATION

At its core, WebChucK is ChucK running as a WebAssembly binary in a `AudioWorkletNode`[18], inheriting the ben-

[8] `https://ccrma.stanford.edu/~cc/windMachine/`
[9] One could imagine a similar workflow for languages like Max/MSP or SuperCollider.
[10] Timbre Library link: `https://ccrma.stanford.edu/~lja/timbre-library/`
[11] `https://p5js.org/reference/#/libraries/p5.sound`
[12] `https://tonejs.github.io/`
[13] `https://tidalcycles.org/`
[14] `https://gibber.cc/`
[15] `https://ide.csound.com/`
[16] `https://glicol.org/`
[17] `https://github.com/sebpiq/WebPd`
[18] `https://developer.chrome.com/blog/audio-worklet-design-pattern/`

efits of WebAssembly's high-performance virtual machine as well as its managed container and fine-grained security model. Compiled directly from the ChucK source (C/C++) using `emscripten`, `webchuck.wasm` contains ChucK's compiler, virtual machine, class libraries, unit generators, and unit analyzers.

Next, an accompanying `webchuck.js` component provides JavaScript bindings to approximately 50 entry points into the C++ ChucK runtime as embedded in the `Wasm` module. It also handles messages passed through the port message interface between the browser thread and an `AudioWorkletNode`. `webchuck.js` also defines the `ChuckNode` and `ChuckSubNode` classes, each of which extends Web Audio's `AudioWorkletProcessor` and manages the ChucK core from within the `AudioWorklet` thread.

A WebChucK host (`webchuck_host.js`) provides helper functions for starting the Web Audio environment, initializing a ChucK instance within that environment as well as messaging mechanisms. The host associates `webchuck.js`'s `ChuckNode` to an `AudioWorkletNode` that can be modularly connected to the audio routing graph in the browser's audio context (`AudioContext`). In general, this host serves as a API for the ChucK runtime.

`webchuck.wasm`, `webchuck.js`, and some form of `webchuck_host.js` exist in all WebChucK projects; the latest released version is made accessible online[19]. Embedding these components makes it straightforward to do the following in a web development context:

1. **Write normal ChucK code**, passed as strings or files to the ChucK runtime. This enables the dynamic addition of ChucK shreds during the lifetime of the web page. Each shred has an ID enabling the removal or replacement of shreds with some simple data management.

2. **Communicate between ChucK and JavaScript**. Running ChucK on the browser can useful as a self-contained audio engine, but further control and flexibility is possible via a collection of "getter" and "setter" methods that allow developers to modify variables or trigger events between ChucK and web-based elements in real-time. This two-way communication enables the creation of web pages with synchronized and controllable audio and UI events (e.g., a slider on a web page can directly control a synthesis parameter in ChucK; meanwhile, precisely timed events in ChucK can drive web-based graphics and visualization).

3. **Take advantage of familiar workflows**. Audio, MIDI, and other data files can be loaded into the ChucK node for playback, sonification, or other creative uses. The WebChucK JavaScript API provides a few simple methods for file data to be accessed from within ChucK. Furthermore, a `chuckPrint()` method pipes anything printed in ChucK code to the browser console, for debugging and general output.

Preparing the ChucK code base for compilation to WebChucK involved some reworking of the source. While the vast majority of ChucK functionality is present in WebChucK, at the time of this writing the following native ChucK functionalities are selectively disabled:

- HID devices (keyboard, mouse, joystick)
- serial communication
- network communication (including Open Sound Control)
- OTF server (used to receive commands like "shred" over the network)
- ChucK watchdog (detects possible time-less infinite loops)
- ChucK shell (interactive prompt mode)

While these functionalities are currently not supported in WebChucK itself, several of these (HID, serial, network communication, OSC) can be implemented in JavaScript and communicated to ChucK.

[19]https://chuck.stanford.edu/webchuck/src/

4. APPLICATIONS AND EVALUATION

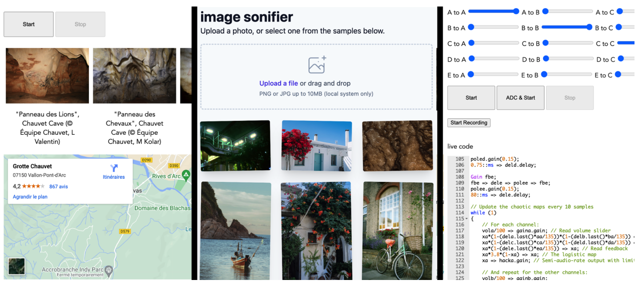
We feel that an evaluation of a tool like WebChucK is most meaningfully carried out by an examination of what people have been able to build and do with it, considering both the unique affordances and limitations encountered as result of using the tool.

Many students and several artists have begun to explore and demonstrate the varied uses of WebChucK. Chris Chafe taught CCRMA's popular "Introduction to Computer-Generated Sound" course entirely with WebChucK in the fall quarter of 2022. Student projects ranged from compositional exercises in data sonification to full-fledged browser-based instruments. Some notable student projects (pictured in Figure 3) include:

1. **Chauvet Cave Beep Ploc Symphony**[20], by Luna Valentin: A data sonification project examining the effect of increased extreme weather phenomena on the ecological balance of caves in France, previously inhabited by our Paleolithic ancestors 36,000 years ago.

2. **Image Sonifier**[21], by Cole Simmons: An image-to-sound converter that uploads an image from the user, retrieves various parameters like minimum and maximum hue and saturation, and sonifies the calculated data with ChucK.

3. **Barrel Synth: A Feedback-Based Browser Instrument**[22], by Josh Mitchell: A unique virtual instrument taking influence from Tom Mudd's Gutter Synthesis algorithm[11]. A bank of sliders allows you to control the gain (and, therefore, feedback) between a set of chaotic duffing oscillators and modal resonators routed to and from one another.



```
poled.gain(0.15);
0.75::ms => deld.delay;

Gain fbe;
fbe => dele => polee => fbe;
polee.gain(0.15);
80::ms => dele.delay;

// Update the chaotic maps every 10 samples
while (1)
{
    // For each channel:
    vola/100 => gaina.gain; // Read volume slider
    xa*(1-(dela.last()*aa/135))*(1-(delb.last()*ba/135))
    xa*(1-(delc.last()*ca/135))*(1-(deld.last()*da/135))
    xa*(1-(dele.last()*ea/135)) => xa; // Read feedback
    xa*3.8*(1-xa) => xa; // The logistic map
    xa => hacka.gain; // Semi-audio-rate output with limi

    // And repeat for the other channels:
    volb/100 => gainb.gain;
```

Figure 3: Left to right: Chauvet Cave Beep Ploc, Image Sonifier, Barrel Synth: A Feedback-Based Browser Instrument.

[20]https://ccrma.stanford.edu/~luna/220a/fp
[21]https://image-sonification.vercel.app/
[22]https://ccrma.stanford.edu/~jomitch/220a/fp

These projects all combined real-time audio generation in ChucK with web-based UI elements like buttons, sliders, graphs, images, and more, and can run in browser environments across platforms. Typically, ChucK developers need to go through a substantial process to incorporate ChucK natively in audiovisual environments such as Unity, or set up an Open Sound Control pipeline between ChucK code and graphics generated in another program (C++, Processing, OpenFrameworks etc,). Instead, WebChucK places ChucK squarely in the world of WebAssembly and web UI/UX, immediately making WebChucK programs compatible across operating systems. This allows ChucK to be extended in web plugins, extensions, modules, web games, or audiovisual creations, as the course projects above demonstrate. Reflecting on his own experiences in the same course, co-author Terry Feng[23] notes that WebChucK is a "one size fits all" tool, specifically in the way it unifies audio and visual development workflows into one environment to serve a user base across many platforms.

Celeste Betancur[24], an avid ChucK programmer and live-coding performing artist, suggested that one of the core advantages of using ChucK on the web is in the powerful timing and scheduling workflow it enables. By contrast, using JavaScript alone, the timing of events is achieved via functions like `setTimeout()` and `setInterval()`, which delay or perform the execution of a script at specific time intervals. Similarly, the Web Audio API provides timing mechanisms that specify when a node starts or stops its processing via the parent class `AudioScheduledSourceNode`'s `start([time])` and `stop([time])` functions, or schedule parameter automation via `AudioParam`'s `setValueAtTime()`, `linearRampToValueAtTime()`, and other functions. These methods require a rather clunky mode of development and no granular control over timing and scheduling. ChucK, on the other hand, provides sample-accurate control of how time progresses on any number of threads ("shreds"). This allows powerful schedulers and the ability to communicate Events and variables to the browser-based UI (e.g., musical events in ChucK triggering graphical events on the web page).

Celeste also reflects that WebChucK is a helpful tool for prototyping live computer-music performance. One can quickly trial creative ideas with ChucK code in the browser without having to manage a desktop development workflow (e.g., command line, audio settings, file systems etc.).

5. CONCLUSION AND FUTURE WORK

5.1 What have we learned so far?

WebChucK is a viable (and often fun) way to ChucK. There are some limitations that come with the medium of web browsers (e.g., network security, which so far has precluded native networking features). At the same time, WebChucK has added to the ecosystem in terms of what one can do with ChucK and expanded *who* can work—and has access to—the medium of programmable sound and music. WebChucK lets users code with ChucK on their favorite modern browsers, both on desktop platforms (macOS, Windows, Linux) and on mobile phones (iOS and Android). It leverages the sonic and musical affordances of ChucK with the many features of the web, making possible new and more accessible modes of audiovisual application development.

As discussed in Section 2.1, there exist many different ways to ChucK. The choice of tools (WebChucK, command line ChucK, miniAudicle, Chunity, etc.) depends on the task at hand. In other words, WebChucK is not intended to be a replacement for any existing tool, but rather an expansion to the collective toolbox.

5.2 Future Work

Momentum favors the development of WebChucK at this time, with ongoing efforts to ensure full-feature parity with the ChucK on the desktop (e.g., including network communication like OSC, multichannel audio support, and variable sample rate) and with miniAudicle and its varied IDE features. Also, more tutorials are being created help users incorporate ChucK into their web projects.[25]

A fully usable WebChucK IDE, similar to *p5.js*'s online editor, has already been developed by Terry Feng, Celeste Betancur, Mike Mulshine, and others. More work will be done to improve the IDE, including implementing a caching file system to organize full projects and enable users to return to all of their previous work between browser sessions. An audiovisual IDE combining WebChucK and *p5.js* is also in the works, opening the door for students and artists to freely and easily prototype highly synchronized audiovisual systems.

Modeled after Google Docs, Overleaf, or Jupyter Notebook, the authors would like to develop a collaborative online WebChucK editor (codenamed **WebChucK Notebook**), to enable users to work on the same ChucK project together simultaneously and remotely, embedding runnable ChucK code inline with text and figures. This could be especially useful for educators teaching audio programming, ChucK development teams, or artistic collaborators.

WebChucK will continue to be co-developed via its various uses in computer music courses and artistic applications. As more people adopt the web as an accessible creative-computing medium, one might ponder:

*How much Web could a WebChucK ChucK, if we **all** could ChucK on the Web?*

6. ACKNOWLEDGMENTS

The authors want to acknowledge and thank CCRMA (at Stanford University) for supporting its students and researchers who have contributed to the development of ChucK and WebChucK over the years. Special appreciate goes to the slew of web audio programmers, including especially Hongchan Choi, for developing the AudioWorklet interface, thus enabling ChucK and other computer music programming languages (Csound, FAUST, Glicol) to run on the web.

7. ETHICAL STANDARDS

WebChucK and Chuck has been developed with the support of CCRMA's departmental funding, curricular student research, and volunteer contributions. The authors are aware of no potential conflicts of interest. The students using early-stage WebChucK were doing so to learn audio programming in a curricular context and were never treated as test subjects.

8. REFERENCES

[1] Faust, a functional programming language for sound synthesis and audio processing. `https://faust.grame.fr/`.

[23] `https://fenglyfe.com/`

[24] `https://www.celestebetancur.com/`

[25] `https://chuck.stanford.edu/webchuck/tutorial/`

[2] Faust, web-based functional sound sythesis programming ide. `https://faust.grame.fr/`.

[3] *Estuary 0.3: Collaborative audio-visual live coding with a multilingual browser-based platform*. Zenodo, June 2022.

[4] *Strudel: Algorithmic Patterns for the Web*. Zenodo, June 2022.

[5] J. Atherton and G. Wang. Chunity: Integrated audiovisual programming in unity. In *New Interfaces for Musical Expression*, pages 102–107. Virginia Tech, June 2018.

[6] H. Choi. Audio worklet: The future of web audio. In *International Conference on Music and Computing*, 2018.

[7] Q. Lan and A. R. Jensenius. Glicol: A graph-oriented live coding language developed with rust, webassembly and audioworklet. In *Web Audio Conference*, 2021.

[8] V. Lazzarini, E. Costello, S. Yi, and J. Fitch. Csound on the web. In *Linux Audio Conference*, 2014.

[9] S. Letz, S. Denoux, and Y. Orlarey. Faust audio dsp language in the web. 2015.

[10] Mozilla. Web Audio API. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API`.

[11] T. Mudd. Gutter synthesis. `https://github.com/tommmmudd/guttersynthesis`, 2018.

[12] C. Roberts and M. Pachon-Puentes. Bringing the tidalcycles mini-notation to the browser. In *Web Audio Conference*, 2019.

[13] S. Salazar, G. Wang, and P. R. Cook. miniaudicle and chuck shell: New interfaces for chuck development and performance. In *International Computer Music Conference*, 2006.

[14] G. Wang. Ocarina: Designing the iPhone's Magic Flute. *Computer Music Journal*, 38(2):8–21, 06 2014.

[15] G. Wang. *Artful Design: Technology in Search of the Sublime*. Stanford University Press, 2018.

[16] G. Wang and P. R. Cook. Chuck: A concurrent, on-the-fly audio programming language. In *International Computer Music Conference*, 2003.

[17] G. Wang, P. R. Cook, and S. Salazar. Chuck: A strongly timed computer music language. *Computer Music Journal*, 39(4):10–29, 2015.

[18] G. Wang and R. Michon. Fauck!! hybridizing the faust and chuck audio programming languages. In *Sound and Music Computing*, 2016.

[19] S. Yi, H. Sigurðsson, and E. Costello. Csound web-ide. In *Web Audio Conference*, 2019.

[20] M. Zbyszynski, M. Grierson, and M. J. Yee-King. Rapid prototyping of new instruments with codecircle. In *New Interfaces for Musical Expression*, 2017.