WebChucK IDE: A Web-Based Programming Sandbox for ChucK

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

Celeste Betancur
CCRMA, Stanford University
celeste@ccrma.stanford.edu

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

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

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

ABSTRACT

WebChucK IDE is a web-based integrated development environment for writing and running ChucK code. WebChucK IDE provides tools and workflows for developing and running ChucK on-the-fly and in any web browser, on desktop and mobile devices. This environment integrates ChucK development with visualization and code-based generative web UI elements to offer an accessible and playful way to program computer music. In this paper, we detail the design and implementation of WebChucK IDE and discuss its various affordances and limitations as a sandbox for learning, experimentation, and art making.

Try WebChucK IDE:
https://chuck.stanford.edu/ide/

1. INTRODUCTION

Advancements in the Web Audio API have enabled computer music synthesis languages to run with near-native performance on web browsers. Recently, ChucK [1]—a *strongly-timed* computer music language—was compiled to WebAssembly from its C/C++ source code with `emscripten`[1] and hosted in JavaScript through the Web Audio API, making it possible to embed the ChucK runtime system (compiler, virtual machine, synthesis engine) on a browser web page. Subsequently, a slew of creative web-based audio projects using WebChucK were developed, ranging from games to artwork, to tools.

One such tool, WebChucK IDE, provides a web-based programming sandbox for ChucK. Users familiar with ChucK will find that the WebChucK IDE provides many of the same features and functionalities as ChucK's long-standing desktop utility, miniAudicle [2]. These include the ability to:

- Start the ChucK virtual machine (VM)

[1] https://emscripten.org/

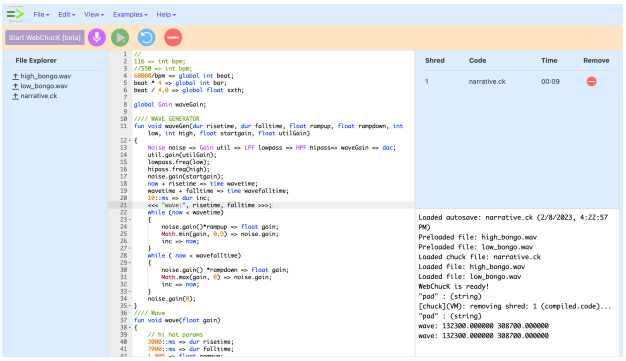

Figure 1. *WebChucK IDE* running a beach-themed ChucK musical composition.

- Access the vast majority of ChucK functionality
- Add, replace, and remove shreds (concurrent processes)
- Monitor the VM, currently running shreds, and program output
- Access ChucK's library of example code

WebChucK IDE combines the *on-the-fly* workflow of miniAudicle with the trappings of the web, such as streamlined access to the underlying audio system (i.e. no need to worry about selecting the correct audio device). Expanded by its proximity to UI/UX libraries of the web, entirely new ChucK development tools have been developed and integrated into the WebChucK IDE sandbox:

- A frequency and time-domain audio visualizer
- An auto-generative user interface
- Vim text-editing support
- Options for Light and Dark mode

Many of the benefits of WebChucK IDE follow from the inherent accessibility of the web. WebChucK IDE expands access to audio programming to anyone with a desktop computer or mobile device and an internet connection. It offers a compelling alternative to the desktop programming workflow for ChucK, avoiding the need to

download software and deal with file systems, audio interfaces, and other operating system overhead. To use WebChucK IDE, users can simply navigate to any web page hosting the IDE [2], start WebChucK (much like "starting the virtual machine" in miniAudicle), and begin writing and running ChucK code. Near ubiquitous access and a streamlined workflow make WebChucK IDE an attractive tool for use in education and research contexts. Furthermore, it serves as a playful sandbox for creative prototyping. In this vein, adopting the run-time and on-the-fly programming paradigms of its predecessor, miniAudicle, WebChucK IDE exposes a mode of immersive real-time audio coding on the web typically associated with live coding environments, transforming any browser into a potential musical instrument.

1.1 A Brief History of ChucK Tooling

2023 marks the 20th anniversary of the release of ChucK. The strongly-timed computer music language has been used in audio synthesis, musical instrument design, laptop orchestra, mobile music apps, and many other artistic, educational, and research contexts. Its notable features include:

- A sample-precise unified timing mechanism for multi-rate event and control processing
- A powerful concurrent programming model rooted in 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 [3]. ChucK exists in various forms including command-line `chuck`, miniAudicle (a graphical IDE; [4]), ChiP (ChucK on the iPhone), FaucK (Faust in ChucK; [5]), Chunity (ChucK in Unity; [6]), and most recently, WebChucK [7] and ChAI (ChucK for AI, in early development).

The first integrated ChucK development environment was the Audicle, in use and in development between 2004-2006 [8]. The Audicle featured an exclusively 3D audiovisual interface and was designed to be a fun and expressive way to write ChucK code *on-the-fly* while visualizing ChucK's inner workings as it compiled and executed the code. Created in C++ and OpenGL, The Audicle aimed to aid development and, potentially, to serve as an audiovisual live-coding performance tool. Through its various "faces," the Audicle provided visualizers for generated audio, timing, concurrency, shreds in the VM, and served as a live coding editor, and as a platform for building bespoke interfaces for three Princeton Laptop Orchestra works ("Non-Specific Gamelan Taiko Fusion", "On the Floor", and "ChucK ChucK Rocket") [3, 9]. While Audicle's video-game-esque audiovisual workflow was a radical experiment in IDE design, its implementation complexity (creating visual elements from the ground up in OpenGL) made it difficult to maintain and slow to develop new features. A simpler and more conventional ChucK IDE emerged in the form of miniAudicle [2], developed by Spencer Salazar in 2006. To this day, miniAudicle serves as the primary integrated development environment for audio programming in ChucK, running on Windows, MacOS, and Linux.

Various concerted efforts led to the creation of WebChucK IDE, including the following:

- Jack Atherton refactored and compiled the ChucK C/C++ source to WebAssembly to run on the web in 2020
- Mike Mulshine made tutorials for embedding WebChucK in a web page [3] and created audiovisual experiments, and demos [4] in 2021
- Chris Chafe taught CCRMA's introductory computer music course using WebChucK (before the IDE existed) in 2022 [5]
- Terry Feng prototyped and developed the interface and architecture of WebChucK IDE in 2022-2023
- Celeste Betancur joined the effort and experimented with code-based auto-generative GUI and graphics shaders in 2023

2. RELATED WORK

Over the years, lightweight desktop IDEs (like DrJava [6]) have been created to gently introduce new users to programming languages and workflows, providing interfaces for new programmers to quickly evaluate and experiment with code. Various online code editors have emerged to bring coding and educational tools to the masses across platforms. These include Online-Python [7], Ideone [8], and the *p5.js* Editor, which provide streamlined workflows to experiment, prototype, test, and learn, removing the technical overhead of installing and working with platform-dependent tooling. WebChucK IDE adds to this list, synthesizing the computer music features of ChucK with a fun and accessible interface.

WebChucK IDE is not alone as other computer music languages have introduced online editors. FAUST (a functional audio signal processing language) [10] has its own web IDE [11, 12] with code-based auto-generated UI tools for quick musical interface prototyping and DSP research [9]. Csound provides an online IDE featuring Csound script editing and a logging console, as well as a community forum for sharing Csound programs [13]. Additionally, efforts to port SuperCollider to the web have been explored, but no unified effort has persisted to bring it to the web in its full form, nor as an IDE.

[2] for example: `https://chuck.stanford.edu/ide/`

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

[4] `https://mikemulshine.com/webchuck-demo`

[5] `https://ccrma.stanford.edu/courses/220a/`

[6] `http://www.drjava.org/`

[7] `https://online-python.com`

[8] `https://ideone.com/`

[9] `https://faustide.grame.fr/`

Though not computer music languages, various audio libraries exist on the web, wrapping, abstracting, and extending the functionality of the Web Audio API. The popular JavaScript library *p5.js* [10] provides a limited set of audio functions (in *p5.sound*) to control the playback of audio files and perform basic sound synthesis. Paired with its online editor, p5.js provides a workflow for building audio-visual projects. Similarly, *gibber* [14] extends the Web Audio API and provides an interactive and fun live-coding interface with audio data visualization inline with JavaScript code.

3. DESIGN AND IMPLEMENTATION

WebChucK IDE offers an integrated development environment and graphical user interface for programming in ChucK. As a programming sandbox, WebChucK IDE contains a code editor, file explorer, output console, and virtual machine monitor. The workflows enabled by these elements are familiar to ChucK users, mirroring elements in the native IDE for ChucK, miniAudicle. Beyond these "standard" features, WebChucK IDE introduces an audio waveform and spectrum analyzer, as well as the ability to dynamically generate a controllable graphical user interface from ChucK code.

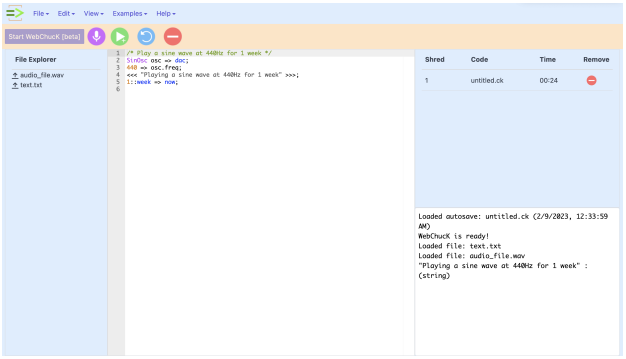

Figure 2. *WebChucK IDE* has 4 main sections. The WebChucK bar (in orange) lies across the top. Beneath that from left to right is the file explorer, code editor, and virtual machine monitor + output console (top and bottom).

The layout of WebChucK IDE is broken down into 4 sections (see Figure 2) The WebChucK bar across the top allows the user to start the ChucK Virtual Machine (VM) and synthesis engine. Once the machine is running, four buttons are enabled, enabling the ability to grant microphone access, compile a ChucK program and add it as a *shred* to the ChucK VM for immediate execution, replace the currently running shred, or remove the current shred. The buttons are designed to be click- and touch-friendly, accommodating usage across different platforms.

The code editor is powered by Ace [11] and incorporates basic editor features like ChucK syntax highlighting and Vim keyboard shortcuts. Users can write ChucK code here or browse for code examples to load into the editor. This makes it easy for users to build rapid prototypes or experiment with existing code from ChucK's extensive example library. To compile this code and run it in ChucK's virtual machine, the user navigates to the WebChucK bar and clicks the green "Compile and Add." This manner of running ChucK programs maintains miniAudicle's *on-the-fly* run-time nature, allowing for multiple programs to be monitored as they run concurrently. The editor's contents are auto-saved to the user's local browser cache to allow for continued development across browsing sessions.

Users can upload files to WebChucK IDE's virtual file system using the file explorer. Files are kept client-side, allowing users to upload and edit ChucK code directly in the browser. The virtual file system processes auxiliary audio and plain-text files uploaded through JavaScript into WebChuck's WebAssembly module instance for file-related APIs and unit generators to interface with the data. ChucK code can additionally be exported as a `.ck` file to be run in native ChucK or the miniAudicle.

The rightmost main section of the WebChucK IDE is comprised of two components: a virtual machine monitor (top), and an output console (bottom). The virtual machine monitor features a shred table, populated whenever a shred is created and running in ChucK's virtual machine. Each row of the table displays a unique shred identification number, filename, current running time, and a button to remove the program associated with the entry. This table shows all of the currently running programs. The output console directly beneath the shred table captures `stderr`/`stdout` from ChucK. This displays print and log messages directly in the IDE. If the output console is not needed, it can be swapped with the audio visualizer or auto-generated GUI.

WebChucK IDE is built with HTML and JavaScript components as well as the *Spectre.css* [12] framework. The IDE runs on top of the WebChucK API, in the form of `webchuck.wasm`, `webchuck.js`, and `webchuck_host.js`. The latter instantiates ChucK's runtime system as an `AudioWorkletNode` [15] within Web Audio, where the WebChucK WASM module acts as the node's `AudioWorkletProcessor` to process audio samples. The entire system runs on a browser client; no backend server is needed. This means that not only is it easy to start using WebChucK IDE from any modern browser, but it is also straightforward to deploy standalone and modified WebChucK IDEs (e.g. hosting a custom WebChucK IDE for use in a classroom setting).

3.1 Audio Visualizer

The Audio Visualizer (see Figure 3) in WebChucK IDE is a real-time visualization of the audio currently being generated by ChucK and displays a time-domain waveform and a short-time Fourier analysis spectrum. This reactive visualization can be helpful in debugging audio synthesis and additionally serves as a teaching tool that generates visual feedback of real-time audio.

The implementation of the audio visualizer takes advantage of the vast array of JavaScript tools and features.

[10] `https://p5js.org/`
[11] `https://ace.c9.io/`
[12] `https://picturepan2.github.io/spectre/`

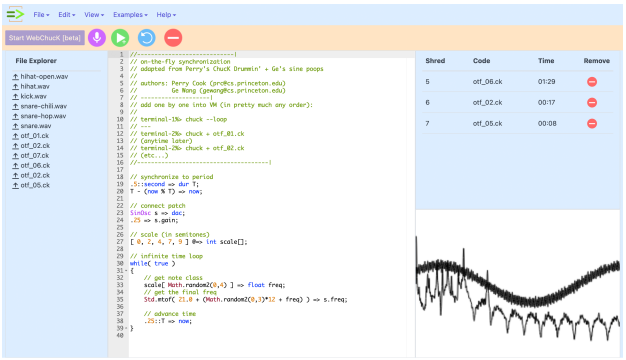

Figure 3. *WebChucK IDE's* audio visualizer pictured in the bottom right.

WebChucK's `AudioWorkletNode` provides direct access to the output audio generated by ChucK. The audio is analyzed and animated in JavaScript using Web Audio's built-in audio analyzer (`AnalyserNode`) and rendered to an HTML Canvas element. The visualizer lives as an external module in the same `AudioContext` as WebChucK, thereby visualizing audio without disrupting the processing and flow of sound.

3.2 Auto-generated GUI

```
global Event kick;
global Event snare;
global Event hihat;
global Event ride;

0 => global float gain;
0.5 => global float freq;

0 => global int octave;
100 => global int midi;

/* additional ChucK code here */
```

Figure 4. *WebChucK IDE's* auto-generated GUI processes global events, floats, and ints for user interaction

WebChucK IDE's auto-generated GUI implementation scans the current ChucK code to create an interactive GUI consisting of sliders and buttons that correspond one-to-one with the user's declared global ChucK variables (see Figure 4). Once generated, the GUI can be immediately used to manipulate corresponding ChucK variables. Presently, each global `int` is associated with a horizontal slider, each global `float` with a vertical slider, and each ChucK `Event` type with a button. By default, floating point values are normalized between 0 and 1, while integers are normalized from 0 to 127. Manipulating a GUI element will perform the appropriate action (e.g., update an integer, update a floating point value, or trigger an Event).

The method and semantics for creating the graphical user interface introduce a new philosophy from the native miniAudicle User Interface (MAUI), the GUI interface API for miniAudicle on MacOS [4]. Unlike MAUI, which presented an API for explicitly arranging and architecting the GUI, WebChucK IDE's approach is automatic and requires no explicit code. While MAUI is capability lies in sophisticated but intentional graphical user interface design, WebChucK IDE's auto-generated GUI elements provide immediately usable buttons and sliders, reducing code overhead to encourage experimentation with mapping different parameters. The automatic GUI generator performs text-preprocessing and is built using a *p5.js* canvas.

4. EVALUATION

While WebChucK has been used as a module embedded in web pages since 2020, and in a popular introductory computer music course at Stanford University in 2022, WebChucK IDE is very much in its infancy. Even so, its potential has become apparent, as programmers (novice and experienced) have quickly adopted the tool to learn ChucK, test its limits, and make music on the web. WebChucK IDE has performed well in preliminary tests, in which we ran a number of complex and computationally demanding ChucK programs without any modifications from their original desktop versions. Compared to desktops, WebChucK's performance varied between half to full efficiency (running in 32-bit WebAssembly), compared to the native command line version (64-bit). These tests included:

- An interactive multi-voice feature-based concatenative audio mosaic tool
- Computationally expensive physical models (Banded Waveguides)
- Audio-modulated WebGL shaders

We acknowledge it will take time, more testing, and more general usage to gain a clearer picture of both the performance and the overall usability of the system.

Seasoned ChucK users have remarked on the flexibility and portability of WebChucK IDE. Being able to not only run but also *code* ChucK programs on a mobile phone inside a browser transforms the world of ChucK development into a sandbox for experimentation. Ease of access and use has implications for communities far beyond the circle of ChucK power users. For example, students who primarily use portable devices in coursework will be able to work with ChucK on their phones or tablets, whereas previously, they would have needed a laptop.

5. FUTURE WORK

A team of active developers aims to bring WebChucK IDE to feature parity with miniAudicle. Additional efforts are underway to integrate tools like the Web MIDI API, libraries such as *osc.js*, and WebGL. These will supplement existing native ChucK implementations, making the IDE more than just a development environment but an expressive interface. While modular components like Chugins (dynamic ChucK libraries) introduce additional functionality and classes on top of the native ChucK language, they aren't yet available in WebChucK. Initial progress favorably suggests that WebChucK IDE will be an integral tool for ChucK development in general.

Of the many future extensions of WebChucK IDE envisioned, these are on the top of our list:

- A streamlined ChucK code-sharing system
- An "export" feature: save your project to run elsewhere or export it as a standalone WebChucK webpage
- Realtime collaborative online code-editing in WebChucK (akin to Google Docs or Overleaf)
- Code-inlined graphics for audio and event visualization (akin to *gibber* [13])
- Ability to write WebChucK, *p5.js*, shaders, and web pages side-by-side and render within WebChucK IDE
- A collection of live coding or performance interfaces (e.g. featuring full-screen graphics for audio visualization and a streamlined overlaid editor)
- Connect other Web Audio libraries or Web Audio implementations of audio programming languages like Faust [11] together

Much of the future work on WebChucK IDE harkens back to the aims of the original *Audicle*: to create a playful and immersive run-time coding environment—a sandbox in which to code, learn, and play.

Acknowledgments

The authors would like to thank Jack Atherton for bringing WebChucK to life in its earliest form, from modifying and compiling the ChucK source to WebAssembly to developing usage examples and basic IDE-like functionality. Special thanks also to Hongchan Choi and others on the Web Audio API team for implementing the `AudioWorklet` interface, enabling the low-level audio control required for programming languages like ChucK to run in conjunction with other Web Audio tools on the web.

6. REFERENCES

[1] G. Wang, P. R. Cook, and S. Salazar, "Chuck: A strongly timed computer music language," *Computer Music Journal*, vol. 39, no. 4, pp. 10–29, 2015.

[2] S. Salazar, G. Wang, and P. Cook, "miniaudicle and chuck shell: New interfaces for chuck development and performance," 01 2006.

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

[4] 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.

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

[6] J. Atherton and G. Wang, "Chunity: Integrated audiovisual programming in unity," in *New Interfaces for Musical Expression*. Virginia Tech, Jun. 2018, pp. 102–107.

[7] M. Mulshine, G. Wang, J. Atherton, C. Chafe, T. Feng, and C. Betancur, "Webchuck: Computer music programming on the web," in *New Interfaces for Musical Expression*, 2023.

[8] G. Wang and P. R. Cook, "The audicle: A context-sensitive, on-the-fly audio programming environ/mentality," in *International Conference on Mathematics and Computing*, 2004.

[9] G. Wang, "Some Principles of Visual Design for Computer Music," *Leonardo Music Journal*, vol. 26, pp. 14–19, 2016.

[10] Y. Orlarey, D. Fober, and S. Letz, "FAUST : an Efficient Functional Approach to DSP Programming," in *NEW COMPUTATIONAL PARADIGMS FOR COMPUTER MUSIC*, E. D. FRANCE, Ed., 2009, pp. 65–96. [Online]. Available: https://hal.science/hal-02159014

[11] S. Letz, Y. Orlarey, and D. Fober, "Faust domain specific audio dsp language compiled to webassembly," *Companion Proceedings of the The Web Conference 2018*, 2018.

[12] R. Michon and Y. Orlarey, "The faust online compiler: a web-based ide for the faust programming language," 2012.

[13] E. C. Steven Yi, Hlöver Sigursson, "Csound web-ide," *Proceedings of the International Web Audio Conference*, pp. 92–97, 2019.

[14] C. Roberts and J. Kuchera-Morin, "Gibber: Live coding audio in the browser," in *International Computer Music Conference*, 2012.

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

[13] `https://gibber.cc/`