OCC

A browser live-coding environment connecting generative visuals and sample-level audio synthesis

Guillaume Piccarreta (guillaume.piccarreta@ircam.fr) - Developer / IRCAM

Introduction

OCC is a browser-native live-coding environment that lets performers script **both** 3D graphics **and** per-sample audio with the same terse DSL. Running entirely in the browser removes installation barriers, encourages audience participation on personal devices and makes every performance instantly shareable via a single URL.

The project builds on the spirit of *::colon::* (2017), an earlier browser based live-coding tool I developed for the Tokyo Algorave, but was rewritten from scratch for maintainability and to embed a native audio engine.

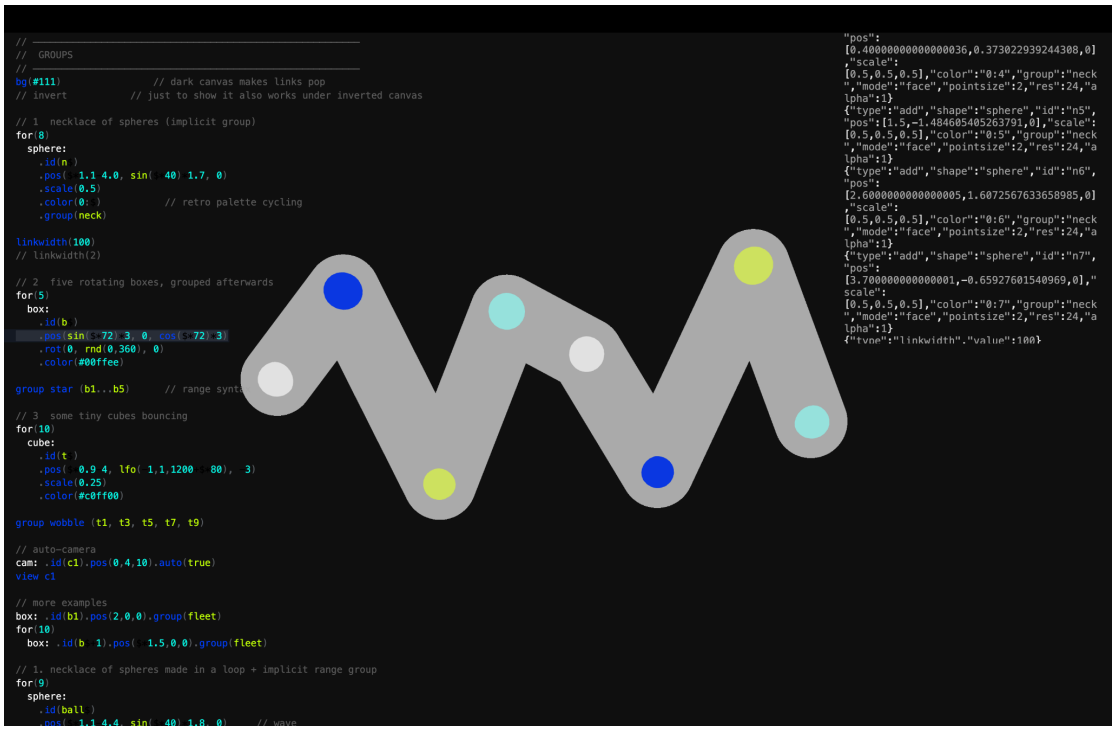

Fig. 1-1 Screenshot of OCC in dark mode: a loop scripts a zig-zag chain of coloured spheres while the editor pane (left) and JSON log (right) reveal the concise DSL syntax and per-operation trace.

Description

System Architecture

- **Frontend stack.** OCC is coded in modern JavaScript/JSX with React 19, React-Three-Fiber and Drei for scene management, Three.js for realtime WebGL rendering, CodeMirror 6 for the ergonomic in-browser editor, Zustand for global state, and Vite for fast reloadspackage.

- **DSL & parser.** A 500-line vanilla-JS parser translates one-liners like `box: id(b1).pos(0,0,0)` or indented blocks into a list of ops processed each animation frame. Expression helpers (`random`, `ramp`, `lfo`, math) are evaluated at run-time to keep visuals and sound perfectly in sync.

- **Audio layer.**

    - **genish.js in an AudioWorklet** provides sample-accurate synthesis, enabling chaotic oscillators, FM feedback networks and audio-rate modulation that are infeasible with block-based nodes.

    - **RNBO modules** generated from Max patches can be loaded as WebAssembly to integrate the DSP in the web browser.

    - **OSC gateway** remains available for hybrid setups with external DSP.
      A thin bridge maps every visual object to an optional *sound-twin*, so position, scale or custom parameters can modulate audio trivially.

Why the Web & Web Audio API?

- **Zero-install** for performers and audiences.
- **Security sandbox** versus native live-coding tools.
- **Uniform APIs** for graphics (WebGL 2) and sound (Web Audio), soon complemented by WebGPU and WASM-SIMD for further speed-ups.
- Growing **community & standards** ensure long-term longevity.

Why genish.js ?

Web Audio's node graph is block-based (128 samples by default), which limits feedback networks, chaotic oscillators and audio-rate scheduling. genish.js compiles *per-sample* DSP graphs into highly optimised callbacks:

- Sample-accurate feedback > FM feedback, physical modelling, feedback delay networks.
- Branch-free codegen > critical for JavaScript's JIT; memory is pooled in a single Float32Array heap.
- Unit-generator vocabulary (cycle, adsr, history, delay, phasor, etc.) familiar to Max/MSP users.

> *"Process a sample in unit-generator A, feed it into B, then feed the result back into A a sample later." (genish.js tutorial)*

In OCC the genish graph for each sound object is compiled once, then driven by the global animation clock. Latency ≈ 1 ms median on M1 / Chrome with a 128-sample hardware buffer.

Domain Specific Language

OCC's domain-specific language keeps the mental load low by exposing a handful of orthogonal verbs. Scene graph editing centres on add, edit, rm, clear, list, and view, while val declares live variables and for (…) builds range-based structures in one line. Global environment tweaks such as bg, invert, palettes, theme, fps or info alter rendering or UI state instantly. A tiny camera grammar extends shapes with .lookat(id), .auto(true) for autonomous roaming, and two flavours of .chain(…) for cinematic whip-pans—either uniform-time (chain(a,b,c,1200)) or per-segment timing (chain((a,500),(b,2000))). Inside any numeric slot the author can embed expression helpers: random()/rnd() for reproducible stochasticity, map() for linear remapping, and the call-by-reference generators ramp(min,max,ms[,env]) and lfo(min,max,ms[,shape]). The latter return callables that the renderer evaluates every frame, meaning you can write box:.scale(lfo(0.3,1,1500,sine)) and get a breathing cube without declaring update loops. Arithmetic (+ - * /) and trig (sin(), cos()) compose naturally with those helpers, giving performers a full-stack audiovisual toolkit in fewer than twenty keywords.

Code Examples

A single global clock (`t`) drives both LFOs and spatial movement; the collision helper fires envelopes exactly when the sphere hits the virtual floor:

```
/* visual object------------------------------------------------- */
sphere:
  .id(orb)
  .pos( sin(t)*2 , 1.2 , cos(t)*2 )
  .scale(0.75)
  .color(#0041ff)

/* sound object (per-sample genish) ---------------------------------- */
sound:
  .id(orbSnd)
  .source( fm(
        carrier : 220 * lfo(0.5,1.5,800,sine),
        mod     : 330,
        index   : ramp(0,8, 4000, easeout)
  ))
  .env( adsr(0.01,0.15,0.7,0.4) )
  .trigger( onCollision(orb) )
  .pan( map( orb.pos.x , -2 , 2 , -1 , 1 ) )
```

Granular rain, falling cubes that fire FM grains on impact:

```
/* GRANULAR RAIN */

val dropN = 24            // how many cubes
for(dropN)
  box:
    .id(drop$)
    .pos( rnd(-4,4), 6 + rnd(0,3), rnd(-4,4) )
    .scale(0.25)
    .color(#00ffee)
    .mode(face)
    .vel( 0 , rnd(-0.02,-0.04) , 0 )

/* short FM "plink" per cube */
for(dropN)
  sound:
    .id(sound$)
    .source( fm(
        carrier : 400 + rnd(-60,60) ,
        mod     : 1.5 ,
        index   : 2.2
    ))
    .env( adsr(0.001,0.04,0,0.12) )
    .trigger( onCollision(drop$) )
    .pan( map(drop$.pos.x,-4,4,-1,1) )
```

- Each cube accelerates downward (`vel` is an OCC helper that adds velocity each frame).
- When `drop$` hits **Y = 0**, `onCollision()` fires a *plink* envelope compiled by genish.js.
- Panning maps X-position > stereo field so the rain “moves” across headphones.

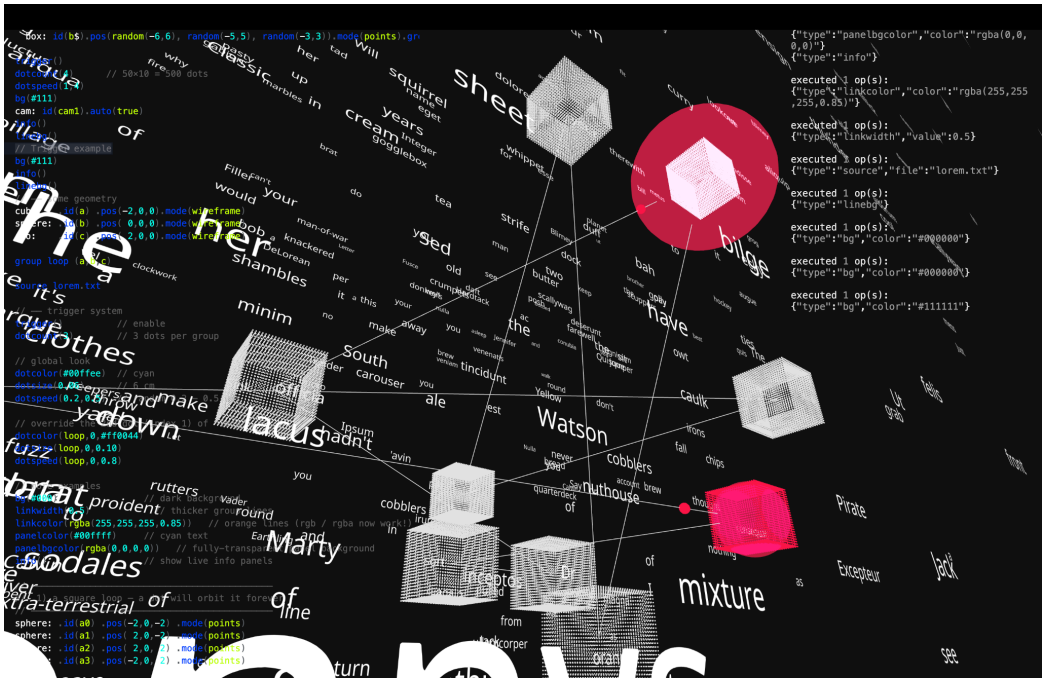

Fig. 1-2 A more complex scene combining a word cloud, wireframe cubes connected by links, and a red “halo” highlighting the audio trigger. The visible code shows how OCC mixes rendering modes (wireframe, points, face) and loads text corpora for live parametric control.

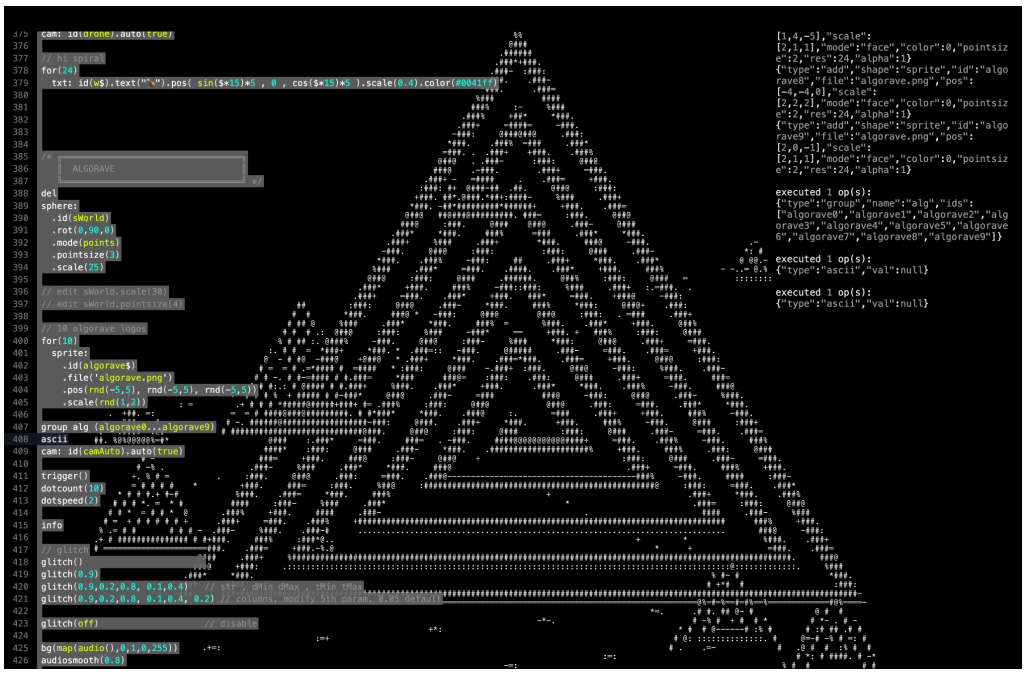

Fig. 1-3 The Algorave logo rendered in ASCII with the OCC editor on the left and the output logs on the right.

Previous Performances

OCC has already proven stage-ready in three major French live-coding events:

Venue & date	Context	Notes
Musée d'Art Contemporain de Lyon (17th May 2025)	*Live coding* evening for Nuit des Musées, organized by GRAME and MAC Lyon	first time using OCC for an audiovisual live performance
Algorave at **Ground Zero** (24th May 2025)	Main live-coding event held in France	live coding set with 3 performers
Le Sucre (28th June 2025)	Closing event of Journées d'Informatique Musicale × Linux Audio Conference (GRAME)	live coding set with 2 performers, club oriented

Fig. 1-4 Raphaël Forment (left) and Guillaume Piccarreta (right) during a live-coding performance at the Musée d'Art Contemporain de Lyon (17th May 2025). OCC code projected behind them drives the audiovisual scene in real time.

Proposed WAC 2025 Contribution

Track: *Artwork & Demo*

- **10 min live-coding set**: build an evolving audiovisual world where visuals directly modulate audio synth graphs.

- **15 min practitioner talk**: dive into DSL design, the concept behind the “sound object” interface, technical details about the implementation such as AudioWorklet tricks, real time code execution and preliminary benchmarks.

- **Equipment**: single laptop, Chrome 118+, stereo mini-jack, HDMI.

Acknowledgments

Special thanks to Raphaël Forment, Rémi Georges and **GRAME** for performance opportunities that drove many iterative fixes; to Charlie Roberts for *genish.js*; and to the OCC beta-testers.

Biography

Guillaume Piccarreta is a programmer and digital artist focusing on the interaction between sound and visual. He holds a Master’s degree in digital art and multimedia interaction from Paris Panthéon-Sorbonne University and spent four years in Tokyo developing creative applications, audiovisual performances and interactive installations. He is currently a developer at IRCAM’s Innovation and Research Means department, working on the Forum platform and R&D projects in audio software. Through his *gcode* project he explores generative algorithms and process-based aesthetics via performances, installations and workshops within the live-coding and digital-art communities.