

# p5.spatial.js: Accessible Multichannel Sound Composition in the Browser

Tommy Martinez  
New York University  
370 Jay Street  
tjm9695@nyu.edu

## ABSTRACT

This paper introduces p5.spatial.js, an open source JavaScript library for creating multichannel sound works in the web browser. Designed to extend the popular creative coding environment p5.js, p5.spatial.js adds multichannel audio output support to the popular, interactive, artist-friendly toolkit. p5.spatial.js is also compatible with existing community-made examples, many of which include stylized motion, or particle systems, that can be readily adapted to animate the trajectory of sound sources in spatial audio contexts. The library implements Distance-Based Amplitude Panning (DBAP), a technique used for irregular speaker configurations typically found in art installation and performance environments. It supports quadraphonic, octophonic, and 5.1 surround setups, and can be modified to address custom loudspeaker arrays via a user-defined JSON object. Designed to be accessible to a community of artists and designers who use p5.js, the library serves non-specialists and experienced composers of computer music alike.

## 1. INTRODUCTION

The use of multichannel loudspeaker arrays for sound diffusion dates back to the early days of electronic audio exploration. From Leopold Stokowski’s ‘three-dimensional’ stereo works in the 1940s[1] to the quadraphonic performances of Pierre Schaeffer and engineer Jacques Poulin in the early 1950s[2], multichannel sound design has continued to be an active field of research and experimentation in the sonic arts, but remains underexplored in web contexts.

Although the ability to output multichannel sound with the Web Audio API’s ChannelMergerNode[3] has been supported across web-browsers since 2014, implementations of spatial diffusion for loudspeaker arrays has largely addressed standard speaker configurations (5.1, 7.1, quadraphonic, etc...) with little to no configurable parameters. Many other tools for web-based sound spatialization have focused on ambisonics for headphone spaces leaving a need for an adaptable JavaScript framework that connects web-apps to multi-speaker sound systems in more flexible ways.

p5.spatial.js is an open source extension that adds multi-

channel audio support to the popular creative coding framework p5.js[4]. Designed with artists in mind, the library simplifies spatialization by automating loudspeaker configuration and abstracting away gain calculations for virtual sound sources. Its modular design supports arbitrary loudspeaker layouts, giving sound artists greater freedom when creating immersive audio works in the browser. Additionally, the library allows for a direct one-to-one mapping of sound sources to output channels.

### 1.1 Visual Design and Control

Graphical user interfaces for multichannel sound design tools often use a Cartesian coordinate system for the control and placement of audio sources in the sound-field. Taking inspiration from landmark systems like Spat[5], the library separates audio processing from graphical interface logic, allowing composers to combine p5.spatial.js with alternative rendering engines[6], such as Three.js or the vanilla Canvas API.

Using the sophisticated graphics engine of p5.js as its host environment for core examples, the library employs a graphical control schema similar to other spatial audio tools using an X and Y plot. Therefore, p5.spatial.js addresses cumbersome workflows arising from managing disparate APIs by offering seamless integration with p5.js, a library that supports compositional flexibility and a rich visual design paradigm in a unified framework—concerns that have been a direct subject of spatial audio research for some time[7].

### 1.2 The p5.js Ecosystem and Community

p5.js has a community of 1.5 million users[8], is widely used for interactive web-based art works, and is taught in creative engineering programs such as ITP/IMA (Tisch School of the Arts, New York University) and IDM (Tandon School of Engineering, New York University). The library provides an easy-to-use API for creating interactive graphics, games, and web-based art. Designed with accessibility in mind, p5 makes a considerable effort to lower the barrier of entry to coding while providing a sophisticated real-time media environment necessary for working across expanded practices like multichannel audio.

The p5.js community use the associated online Web Editor[9] and the OpenProcessing platform[10], to share a vast repository of user-generated and reusable example projects called “sketches.” These sketches vary in application and demonstrate a wide variety of concepts in interactive and generative art, many of which are relevant to the task of designing trajectories for sound objects in 2D and 3D space.

There are also an abundance of learning materials for p5.js such as Daniel Shiffman’s Nature of Code[11], which offers easy-to-follow blueprints for simulating natural phenomena such as physics systems and flocking. These behaviors have been extensively explored in spatial sound design[12], making p5.js an optimal environment to port existing research for further exploration.

The p5.js framework is further enhanced by numerous community-built libraries. These extensions, such as p5.ble.js (for wireless communication with serial devices like Arduino)[13], p5.party (for networked multiplayer experiences)[14], and ml5.js (for gestural input using machine learning)[15], work seamlessly with p5.spatial.js, adding to the breadth of its capabilities as an extensible and modular tool for spatial audio.

## 2. LIBRARY DESCRIPTION

In addition to the robust graphics engine embedded in p5.js, p5.spatial.js is enhanced by its compatibility with p5.sound.js[16], an official audio library for the framework. Built using Tone.js as the primary dependency[17], with a leaner feature set geared towards beginners, p5.sound.js contains a powerful collection of audio nodes capable of sample playback, synthesis and much more.

Like Tone.js and the underlying Web Audio API, p5.sound.js allows users to design custom audio graphs by composing networks of virtual sound source and effects nodes. The library has been used to create everything from complex generative audio works to interactive web-based instruments and real-time visualizers for live performance. For more information on the current status of the p5.sound.js library, see this write-up detailing a redesign I did for the project at this URL[18].

p5.spatial.js provides two new classes, **p5.AudioSource**, and **p5.ChannelOut**, for use within the existing p5.sound.js ecosystem. The classes allow users of the library to connect a p5.sound.js audio-graph to an external multichannel audio interface and offers a number of strategies for spatialization one might expect from a tool of this kind. They are discussed in the following chapters.

### 2.1 The p5.AudioSource Class

The **p5.AudioSource** class spatializes sound in a multi-channel speaker array based on its x and y position in the p5 canvas coordinate space. The class implements a simplified DBAP (Distance-Based Amplitude Panning) algorithm, calculating the loudness of a sound source in each loudspeaker as a function of its proximity to virtual output nodes [19] specified at initialization. Because the algorithm does not depend on the listener’s position (like ambisonics for example), it is well suited for performance or installation settings where speakers might be configured in asymmetrical layouts with moving audience members.

The **p5.AudioSource** class can be chained to the end of any p5.sound.js node using the standard **connect()** method provided by the p5.sound.js library. The class accepts two arguments, the first of which is a string or object defining a speaker layout. Accepted strings include ‘quad’, ‘octo- phonic’, and ‘5.1’ and configure the algorithm to match the speaker array in the real environment. A JSON object defined by the user may be provided instead of the default string parameters to specify a custom speaker layout, and is the subject of a later section. An optional second **number**

argument can be provided to determine how far a sound source may be placed from an output node to be audible. Upon initialization, **p5.AudioSource** creates a number of Web Audio API GainNodes equal to the number of output nodes in the specified layout. The GainNodes are routed in sequence to the input channels of a ChannelMergerNode and pushed into an array for later modification. The ChannelMergerNode is then connected to an **AudioContext** which routes the GainNodes to discrete output channels on the attached output hardware.

```

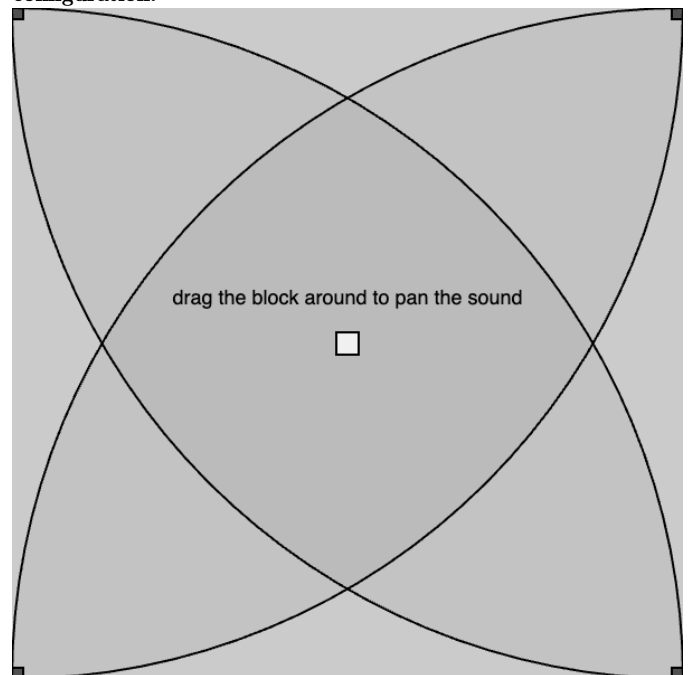
this.merger =
  ↪ this.context.createChannelMerger(this.outputs);
this.gains = [];
this.maxDistance = maxDistance;
for (let i = 0; i < this.outputs; i++) {
  let speaker = this.context.createGain();
  this.audioSource.connect(speaker);
  speaker.gain.value = 0;
  speaker.connect(this.merger, 0, i);
  this.gains.push(speaker);
}
this.merger.connect(this.context.destination);

```

#### 2.1.1 Usage Example

The example sketch below demonstrates basic usage for the **p5.AudioSource** class. A prototypical p5 sketch is created to ‘bounce’ a cube off of the canvas boundaries. A sound source correlates with the cube’s movements in a quadraphonic speaker space.

**Figure 1: The p5.AudioSource in its default, quadraphonic configuration.**



1. In the `setup()` function an `Oscillator` is connected to a new `p5.AudioSource`. It's default state is to represent a quadraphonic speaker space.
2. During each draw loop the `move()` method calculates the Euclidean distance between the sound source and each loudspeaker in the array, using those distances as a gain scalar for the connected audio source. The calculated gain value is applied using the Web Audio API's `AudioParam` `setTargetAtTime()` method which produces a smooth transition between target values. As a result, when the sound source approaches a virtual speaker node, its amplitude in the corresponding output channel grows louder, while the sound's amplitude is attenuated in speaker nodes which are further away.
3. Bouncing Cube Audio Source: Moves the cube by updating its position using the velocity vector and reverses the direction of the cube if it collides with the canvas boundary. The position of the audio source is updated to match the position of the cube using the `move()` method in the `p5.AudioSource` class. This is simply one approach to animating `p5.AudioSource` trajectories and may be defined using other generative or interactive algorithms.
4. Rendering the GUI: We render the speaker layout, the speaker overlap radius and the audio source as a square with the respective `renderLayout()`, `renderDistance()`, and `renderSource()` methods. These methods are optional however and are provided to build quick user interfaces and helpful visualizations for spatial audio projects. Users may choose to create their own visualizations with the tools provided by the `p5.js` core library.

```
//IMPORTANT! Configure your multichannel sound
↳ card in the system preferences and refresh
↳ this page. Watch your Master Volume level
↳ before starting.
let osc;
let spatSource;

let pos, speed;

function setup() {
  createCanvas(100, 100);
  pos = createVector(0, 0);
  speed = createVector(1, 1.3);

  osc = new p5.Oscillator('sine');
  osc.disconnect();

  /*
  Create a multichannel audio node and connect a
  ↳ sound source to it!
  */
  spatSource = new p5.AudioSource();

  osc.connect(spatSource);
```

```
background(220);
textSize(10);
textAlign(CENTER);
textWrap(WORD);
}

function draw() {
  background(220);
  pos.x += speed.x;
  pos.y += speed.y;
  if (pos.x < 0 || pos.x > width) {
    speed.x *= -1;
  }
  if (pos.y < 0 || pos.y > height) {
    speed.y *= -1;
  }
  //Update the rectangle position and "attach" a
  ↳ sound source to it
  rect(pos.x, pos.y, 10, 10);
  spatSource.move(pos.x, pos.y);
  spatSource.renderLayout();
  spatSource.renderDistance();
  text('click to start the sound', 0, 20, 100);
}

function mousePressed() {
  osc.start();
}
```

### 2.1.2 Advanced Usage

The above source code can be modified to work with octophonic speaker arrays by passing in the 'octophonic' string when initializing the `p5.AudioSource` class. The three built-in layouts supported by `p5.spatial.js` are 'quad', 'octophonic' and '5.1', although custom layouts can be defined through a process described in the following section.

```
/*
creates a spatial audio source that is decoded to
↳ an octophonic array
*/
spatSource = new p5.AudioSource('octophonic')
```

In many cases, especially when the `p5` canvas is large, it may be necessary to adjust the `maxDistance` property of the `p5.AudioSource` to ensure sufficient overlap between adjacent speakers. This parameter defines the maximum distance from which a sound source will begin to be audible, with gain scaled according to proximity. For smooth spatialization, where perceived loudness remains constant as the source moves across the field, `maxDistance` should be set so that the ranges of neighboring virtual speakers overlap. Ideally, this overlap matches the distance between speakers, allowing one speaker signal to fade out as another fades in, maintaining uniform power across the sound field.

If the range is too small, gaps in coverage may occur, leading to dropouts in the audio signal. In contrast, larger overlaps reduce spatial specificity by distributing the sound

Figure 2: Rendering of the `p5.AudioSource` class initialized using an octophonic array

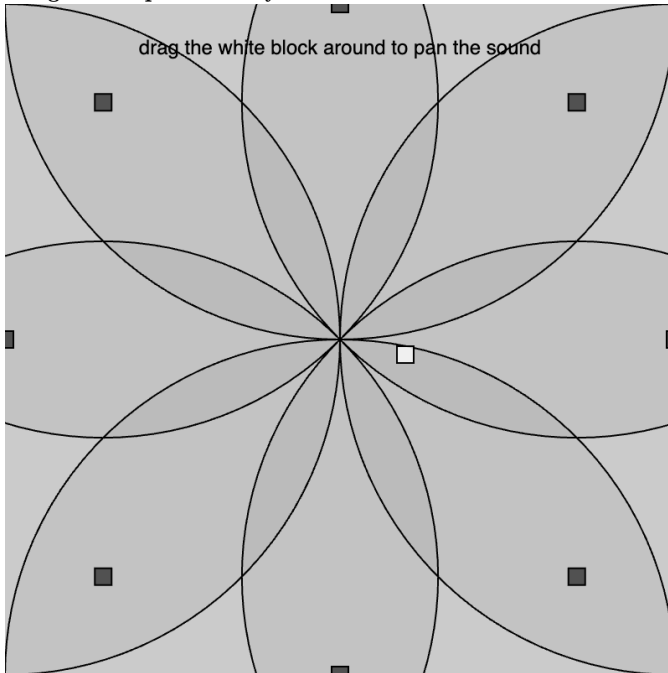
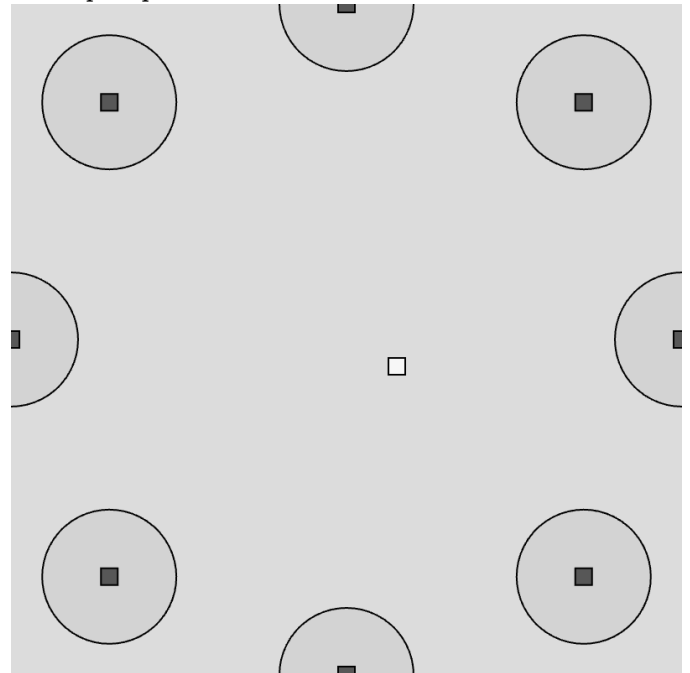


Figure 3: Rendering of the `p5.AudioSource` class with smaller source pickup radiuses



too widely.

Shown in Figure 3 is a creative use of the `maxDistance` property, creating more narrowly focused sound-producing regions.

```
//creates a smaller overlap between speakers
↳ causing dips in audio as the source travels
↳ the speaker-space
spatSource = new p5.AudioSource('octophonic',
↳ 40).
```

## 2.2 Addressing Custom Speaker Arrays

Custom loudspeaker arrays can be defined in `p5.spatial.js` by passing a specially formatted JSON object to `p5.AudioSource` when initializing the class. The object specifies the location of each output node using p5 coordinates. The JSON object can be declared globally or within the `p5.setup()` function, as long as it is defined before `p5.AudioSource` is initialized.

Currently, `p5.spatial.js` only supports two-dimensional layouts and has been tested with up to 16 output channels in Safari, Firefox, and Chrome. Each speaker is represented as a key-value pair, where the key is a unique name and the value is an object with x and y coordinates, with w and h specifying dimensions for rendering.

```
function setup() {
  createCanvas(400, 400);
  //define a three-channel output layout
  //the output is defined after the canvas is
  ↳ created to allow us to use width and height
  ↳ variables
```

```
let triple = {
  out_1: { x: (width/2) - 100, y: height/2, w:
    ↳ 10, h: 10 },
  out_2: { x: width/2, y: height/2, w: 10, h:
    ↳ 10 },
  out_3: { x: (width/2) + 100, y: height/2, w:
    ↳ 10, h: 10}
}
osc = new p5.Oscillator();
osc.disconnect();
spatSource = new p5.AudioSource(triple, 100);
osc.connect(spatSource);
}
```

See Figure 4 for a visual representation of this excerpt.

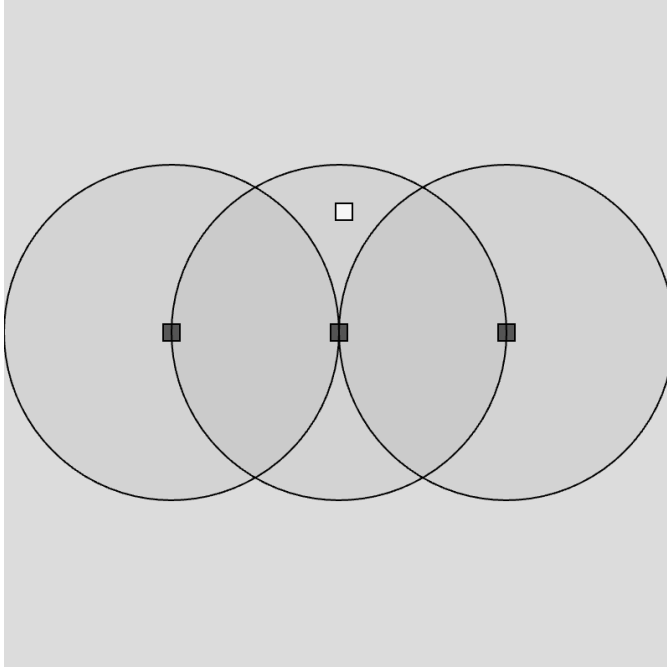
## 2.3 The `p5.ChannelOut` Class

Unlike the `p5.AudioSource` class, `p5.ChannelOut` does not implement any panning logic; instead, it creates a one-to-one mapping between a sound source and a specified output channel on the hardware. This makes it useful for tasks such as routing pre-mixed stems to specific loudspeakers or sending a low-pass filtered signal to an LFE (Low Frequency Effects) channel in 5.1 layouts (as the LFE channel is ignored in `p5.AudioSource` spatialization). The class accepts a single zero-based integer which determines the output channel. You can change the output channel by calling the `switch()` method and passing in a new channel index.

### 2.3.1 Usage Example

An example sketch demonstrating the `p5.ChannelOut` class and its associated `switch` method is analyzed below. A live

Figure 4: Rendering of a custom three-channel speaker layout



example can be viewed here[20].

1. p5.sound.js and p5.spatial.js in the `setup()` function: A basic audio graph consisting of an oscillator node from p5.sound.js and the `p5.ChannelOut` class from p5.spatial.js is initialized.
2. Channel switching in the `draw()` function: For each draw loop, the counter is raised by 1. If the counter value is divisible by 10, the frequency of the oscillator is changed and the output channel is modified by passing a random integer into the switch method.

```
let counter = 0
let channel = 0

function setup() {
  createCanvas(400, 400);
  textAlign(CENTER);
  osc = new p5.Oscillator('sawtooth');
  osc.disconnect();

  //create a discrete channel output and connect
  ↪ our audio source to it
  chanOut = new p5.ChannelOut();
  osc.connect(chanOut);
}

function draw() {
  background(220);

  counter++;
  if (counter > 100) {
```

```
    counter = 0
  }

  if (counter % 10 == 0) {
    osc.freq(440 + counter * 100);
    channel = floor(random(8))
    chanOut.switch(channel);
  }

  text('click to start', 0, 180, width)
  text(`sending to channel ${channel + 1}`, 0,
    ↪ height/2, width);
}

function mousePressed() {
  osc.start();
}
```

### 3. AVAILABILITY AND DOCUMENTATION

p5.spatial.js can be added to your p5.js sketch by including the library in the HTML head tag after p5.sound.js.

```
<script src="https://cdn.jsdelivr.net/gh/ogbabyd ↪
  ↪ iesal/p5.spatial.js@latest/dist/p5.spatial.j ↪
  ↪ s"></script>
```

Alternatively, the library can be built from the source by cloning the GitHub repository[21] and running `npm install` and `npm run build`.

Usage examples for p5.spatial.js can be found in a 'collection' of sketches using the p5.js Web Editor[22] and are also linked to in the GitHub repo.

### 4. FUTURE DEVELOPMENT

Future versions of p5.spatial.js will introduce the ability to address 3-Dimensional speaker arrays with the WebGL renderer in p5. Additionally, many multichannel panners allow the operator to change speaker positions in real-time allowing for greater flexibility when calibrating or improvising with the system. The `maxDistance` property of the `p5.AudioSource` class can at the moment only be set as a group. Independent control of those pickup radii should be introduced to allow for more fine-grained tuning.

### 5. CONCLUSION

p5.spatial.js provides a fast and convenient way to create browser-based sound work with multichannel loudspeaker arrays. With its built-in support for standard speaker layouts, and an intuitive workflow for connecting the library to more specialized systems, p5.spatial.js introduces multichannel spatialization to web audio in a flexible new way.

Through its native compatibility with p5.js, it is welcoming to new programmers yet provides unique capabilities for experienced computer musicians. In some cases p5.spatial.js has been an artist's first experience with spatial audio, gently introducing them to the notion of multichannel panning through the accessible nature of p5 learning materials. In

other cases, p5.spatial.js will be the first time an experienced composer of spatial audio work will be able to create something that lives online. I hope that welcoming members from both sides of this spectrum will bridge a gap in a highly specialized practice and allow for more nuance and variety in future multichannel audio works.

## 6. REFERENCES

- [1] G. Milner, *Perfecting Sound Forever: The Story Of Recorded Music*. Granta Publications, 2011.
- [2] J. HARRISON, “Sound, space, sculpture: some thoughts on the ‘what’, ‘how’ and ‘why’ of sound diffusion,” *Organised Sound*, vol. 3, no. 2, p. 117–127, 1998.
- [3] “ChannelMergerNode, Web Audio API.” <https://developer.mozilla.org/en-US/docs/Web/API/GainNode>. Accessed: 2025-06-24.
- [4] “Official website of the p5.js library.” <https://p5js.org/>. Accessed: 2025-06-24.
- [5] IRCAM, “Spat.” <https://forum.ircam.fr/projects/detail/spat/>.
- [6] T. Carpentier, M. Noisternig, and O. Warusfel, “Twenty Years of Ircam Spat: Looking Back, Looking Forward,” in *41st International Computer Music Conference (ICMC)*, (Denton, TX, United States), pp. 270 – 277, Sept. 2015.
- [7] J. Garcia, T. Carpentier, and J. B. and, “Interactive-compositional authoring of sound spatialization,” *Journal of New Music Research*, vol. 46, no. 1, pp. 74–86, 2017.
- [8] “Lauren McCarthy’s Website, creator of p5.js.” <https://get-lauren.net/p5-js>. Accessed: 2025-06-24.
- [9] “The p5 Web Editor.” <https://editor.p5js.org/>. Accessed: 2025-06-24.
- [10] “OpenProcessing.” <https://openprocessing.org/>. Accessed: 2025-06-24.
- [11] D. Shiffman, *The Nature of Code: Simulating Natural Systems with JavaScript*. No Starch Press, 2024.
- [12] D. Kim-Boyle, “Spectral and granular spatialization with boids,” in *International Conference on Mathematics and Computing*, 2006.
- [13] T. I. Yining Shi, Jingwen Zhu, “p5.ble.js.” <https://itpnyu.github.io/p5ble-website/>.
- [14] J. Bakse, “p5.party.” <https://p5party.org/>.
- [15] “ml5.js.” <https://ml5js.org/>.
- [16] “The p5.sound.js GitHub Repo.” <https://github.com/processing/p5.sound.js>. Accessed: 2025-06-24.
- [17] Y. Mann, “Interactive music with tone.js,” in *Proceedings of the International Web Audio Conference* (S. Goldszmidt, N. Schnell, V. Saiz, and B. Matuszewski, eds.), WAC ’15, (Paris, France), IRCAM, January 2015.
- [18] T. Martinez, “Announcing The New p5.sound.js Library.” <https://medium.com/processing-foundation/announcing-the-new-p5-sound-js-library-42efc154bed0>. Accessed: 2025-06-24.
- [19] T. Lossius and P. Baltazar, “Dbap – distance-based amplitude panning,” 01 2009.
- [20] T. Martinez, “p5.spatial.js, discrete channel output with p5.channeloutput.” <https://editor.p5js.org/>
- thomasjohnmartinez/sketches/dPtKvHqOm.
- [21] T. Martinez, “p5.spatial.js, a multichannel sound library for the browser.” <https://github.com/ogbabydiesel/p5.spatial.js>.
- [22] T. Martinez, “p5.spatial.js example sketches.” <https://editor.p5js.org/ogbabydiesel/collections/...> Accessed: 2025-09-30.