Lyric App Framework: A Web-based Framework for Developing Interactive Lyric-driven Musical Applications

Jun Kato
jun.kato@aist.go.jp
National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba, Ibaraki, Japan

Masataka Goto
m.goto@aist.go.jp
National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba, Ibaraki, Japan

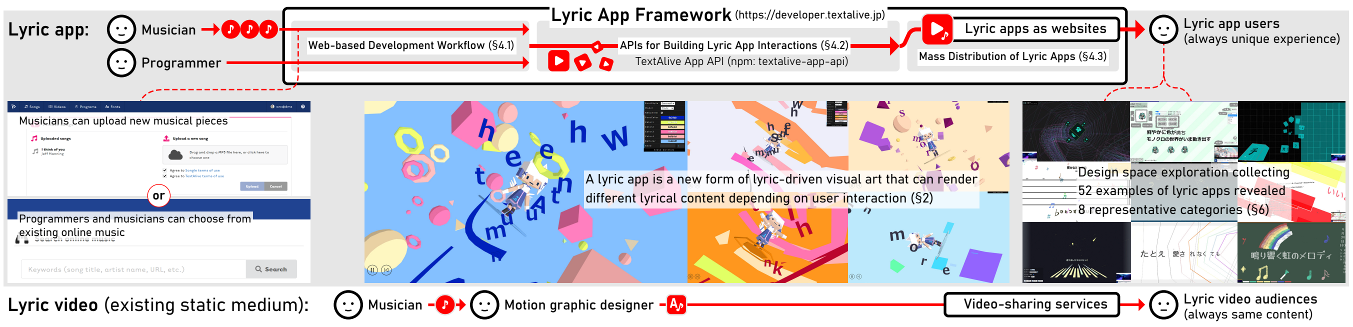
Lyric app:
Musician
Programmer
Lyric App Framework (https://developer.textalive.jp)
Web-based Development Workflow (§4.1)
APIs for Building Lyric App Interactions (§4.2)
TextAlive App API (npm: textalive-app-api)
Lyric apps as websites
Mass Distribution of Lyric Apps (§4.3)
Lyric app users
(always unique experience)
Musicians can upload new musical pieces
or
Programmers and musicians can choose from existing online music
A lyric app is a new form of lyric-driven visual art that can render different lyrical content depending on user interaction (§2)
Design space exploration collecting 52 examples of lyric apps revealed 8 representative categories (§6)
Lyric video (existing static medium):
Musician
Motion graphic designer
Video-sharing services
Lyric video audiences
(always same content)

Figure 1: The Lyric App Framework opens up a novel design space for programmers and musicians to develop *lyric apps*. These apps enable users to interact with lyric-driven visual art synchronized with music, addressing the limitations of lyric videos.

ABSTRACT

Lyric videos have become a popular medium to convey lyrical content to listeners, but they present the same content whenever they are played and cannot adapt to listeners' preferences. Lyric apps, as we name them, are a new form of lyric-driven visual art that can render different lyrical content depending on user interaction and address the limitations of static media. To open up this novel design space for programmers and musicians, we present Lyric App Framework, a web-based framework for building interactive graphical applications that play musical pieces and show lyrics synchronized with playback. We designed the framework to provide a streamlined development experience for building production-ready lyric apps with creative coding libraries of choice. We held programming contests twice and collected 52 examples of lyric apps, enabling us to reveal eight representative categories, confirm the framework's effectiveness, and report lessons learned.

CCS CONCEPTS

• **Human-centered computing** → **User interface toolkits**; *Web-based interaction*; • **Applied computing** → *Sound and music computing*.

KEYWORDS

toolkit, multimedia control, music synchronization

ACM Reference Format:
Jun Kato and Masataka Goto. 2023. Lyric App Framework: A Web-based Framework for Developing Interactive Lyric-driven Musical Applications. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), April 23–28, 2023, Hamburg, Germany.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3544548.3580931

1 INTRODUCTION

Lyrics in a musical piece have greater power to induce mood changes than music alone [41], and lyrics have been delivered to audiences in various ways that reflect the evolution of media technologies. Vinyl records and compact discs may come with *lyric sheets* on which graphic designers aesthetically place static lyrics text. Since music videos and video-sharing services appeared, *lyric videos* have become popular, with motion graphic designers implementing kinetic typography of lyrics synchronized with music playback. In the current era of media convergence [14], musicians use various media formats to deliver their musical pieces to audiences. The current era is also one of participatory culture [15], in which audiences not only listen to musical pieces but also actively participate in the lifecycle of media content. For example, they make posts on social networking services and share their experience with their favorite musical content. Moreover, the proliferation of smartphones has provided many people with casual means to experience novel interactions with musical pieces [22].

We foresee that an interactive medium incorporating lyrics, which we name a *lyric app* (Figure 1), will follow lyric sheets and lyric videos in providing a new way to enjoy musical pieces and lyrics audibly and visually. For instance, when an end-user runs a lyric app, he or she can listen to a musical piece and see a dynamically generated scene in three-dimensional (3D) computer graphics. Unlike in static lyric videos, it is possible to freely navigate a scene in which graphical objects based on lyric phrases appear at the timings when they are vocalized. Thus, users can not only read but also touch and place lyric phrases to build toward a one-time landscape at the end of the music piece, making the experience unique. Clearly there are far more interactive design opportunities here than in the static medium of lyric videos.

However, the development of lyric apps involves distinct challenges as compared to authoring lyric videos. Synchronization of lyrics with music playback requires laborious preparation to annotate the timings of lyrics and other musical elements. It is not straightforward to build time-sensitive user interactions for musical content, nor is it trivial to deliver such applications to end-users.

To aid the development and explore the design space of lyric apps, we propose *Lyric App Framework*, a web-based framework that provides a streamlined development experience for building production-ready lyric apps. After accepting music audio and corresponding lyrics text as inputs, the framework analyzes the audio to estimate the timings of lyrics and other musical elements. Then, it provides an application programming interface (API) to control the music playback and access relevant musical elements at the specified timings. The resulting lyric apps can be distributed as standard websites that can be executed by web browsers on smartphones and PCs. We built this framework on top of our current web service for authoring lyric videos, named *TextAlive* [34], and we released it to the general public (https://developer.textalive.jp).

Our contributions in this work are threefold: (1) We extend lyric videos by adding interaction capabilities around musical pieces and lyrics, thus yielding a novel media format called a "lyric app." (2) We built a novel framework comprising a web-based workflow and API (the *TextAlive App API*) to provide a streamlined development experience for building production-ready lyric apps. (3) We deployed the framework in the wild, conducted annual programming contests twice, and collected 52 examples of lyric apps (in addition to 11 apps that we developed); as a result, we could reveal eight representative categories, evaluate the framework's effectiveness, and obtain insights for future work.

2 LYRIC APPS

In this section, we explain our web service for authoring lyric videos, which led us to define a lyric app as *a novel media format that adds interaction capabilities to lyric videos*. Then, we provide a representative example to illustrate the interactive user experience, and we explain the distinctive development challenges.

2.1 From Lyric Videos to Lyric Apps

In 2015, we built TextAlive [34], a web service for authoring lyric videos that was based on the interaction design described in our CHI 2015 paper [16]. TextAlive allows programmers to interactively develop graphical algorithms for animating text and drawing graphics, and it allows musicians to choose and customize those algorithms with its intuitive user interface to create lyric videos. As of February 6, 2023, 1396 algorithm revisions and 18,867 video data entries had been created on TextAlive. Both amateur and professional musicians have used it to promote their musical pieces.

While programmers could use TextAlive to successfully define a variety of graphical algorithms, we observed that their creativity was limited by the tool and the target media format. First, they had to use the provided programming environment, which prevented them from using the creative coding libraries of their choice. Second, they had to follow a specific program structure that renders a static video frame without any interaction capabilities, other than defining typed parameters for musicians to customize the visual styles of lyric videos.

To address these limitations, we conceptualized a "lyric app" that adds interaction capabilities to lyric videos. Instead of providing a specific programming environment, we provide a framework for lyric app development that allows programmers to use their favorite tools and environments. Musicians can still customize applications to promote their musical pieces, and audiences can use those apps to listen to music, read lyrics, and interact with lyrics.

2.2 Representative Example of Lyric App

Here, given the benefits of lyric apps from the programmer and musician perspectives, we concretize the interactive user experience.

Suppose that Alice finds a new blog post by her favorite musician in the afternoon. When she opens a link in the post, the web browser executes a lyric app and starts playing a newly released musical piece. The piece's cover image and title appear on the wall in a space of three-dimensional computer graphics (3D CG).

Unlike with a static lyric video, Alice can touch the screen to navigate freely in the space. She enjoys finding various graphical objects, which reflect the song's theme and are animated in synchrony with the music's beat. Then, she notices lyrics popping out at the timings when they are vocalized. She can not only read the lyrics but also manipulate and place them in her own way to build a one-time landscape. When the musical piece reaches the chorus, the visual effects become more prominent to reflect excitement.

In the evening, Alice opens the lyric app again. It renders the same 3D CG space but in dark colors. She can now build a different landscape, thus making the same song and lyrics feel more familiar yet fresh. Within the lyric app, she can even open another musical piece by the musician. These interaction capabilities enable her to dive deeply into the virtual world of the musical pieces, making the experience unique and personal.

2.3 Development Challenges and Support Needs

Given the comparison to lyric video programming and the above lyric app example, we discuss three distinctive challenges in developing lyric apps that lead to the needs for development tools.

First, lyric-driven visual art, including lyric videos and apps, must be synchronized with the music playback. Such synchronization typically requires a lot of manual effort to annotate lyrics and other music-structure timings. Our prior work on lyric video authoring introduced automatic analysis and a user interface for correcting the analysis to reduce the amount of labor [16]. Such a semiautomated procedure would be beneficial, but it is not obvious how to integrate the procedure into the lyric app development workflow.

Second, it is not straightforward to build user interactions for musical pieces and lyrics. Compared to motion graphic designers, who elaborate a single version of a lyric video, programmers must come up with algorithms to generate visuals on the fly. Programmers can benefit from existing creative coding libraries (Section 3.3), but lyric app development is notably more challenging because it requires accounting for various musical timings, lyric meanings, and user inputs. It is crucial to offer tool support that does not conflict with creative coding practices.

Third, it is unclear how to deliver lyric apps to end-users. While typical research on creativity support tools often focuses on laboratory studies [8], we consider an engineering perspective for production use to be very important, particularly for research on the novel media format of lyric apps. Just as video-sharing services enable musicians to easily distribute lyric videos, there should be a way for programmers and musicians to distribute their lyric apps.

3 RELATED WORK

Before introducing our framework to support programmers, we review related work to provide context for our research contributions. First, we review early examples of lyric apps. Then, we cover prior work on automatic synchronization techniques for multimedia content and programming techniques for generating and interacting with multimedia content.

3.1 Early Examples of Lyric Apps

Synchronization of audio and visual media makes for an immersive experience. Karaoke allows people to watch simple lyric captions, sing a song without remembering the exact lyrics, and enjoy the musical piece. In some cases, karaoke machines play lyric videos and provide an immersive experience. Some rhythm games [39, 42] and games for practicing typing and foreign languages [26] can also show lyrics text synchronized with music playback.

These applications use the timing information of musical elements such as lyrics and beats and can be considered as early examples of lyric apps. However, they have a relatively narrow scope from the interaction design perspective. Karaoke involves little interactivity, and games focus on fun gameplay itself or gamified purposes. We expect that lyric apps have more potential than their predecessors, and we thus built a framework to aid lyric app development and explore the design space (see Section 6).

3.2 Automatic Synchronization Techniques

An experience synchronized with lyric and musical elements is integral to lyric app development. There are previous examples of automatically synchronizing and generating such time-series multimedia content.

Automatic transcription of speech videos and unified interactions for the resulting time-coded text, audio, and video has long been studied. PodCastle [11] applies automatic speech recognition to podcasts, accepts crowdsourced user edits for improved accuracy, and provides full-text search within the transcriptions. Our work uses the same crowdsourcing approach and enables revision control of user edits. Video Digests [31] defines a structured format for summarizing video content and provides a user interface for editing such summaries with transcript-based interactions. Our work deals with structured time-coded information, not for speech transcription but for lyrics and musical elements, and we propose a novel interactive media format using this information.

Synchronization of lyrics text with its corresponding musical audio signals is typically more challenging than synchronization of speech transcription with its corresponding speech signals, because singing voices have varying vocal styles and music signals contain accompaniment sounds. LyricSynchronizer [9] aims to solve this problem by applying a novel signal processing technique for audio-lyric synchronization. By leveraging this technique, it provides a user interface for controlling the music playback position by clicking lyrics text. TextAlive [16, 34] further enables interactive kinetic typography video authoring and live programming of graphic rendering algorithms for videos. SyncPower [4], Musixmatch [28], and LyricFind [25] provide APIs for retrieving lyrics text and rendering karaoke-style visuals with graphical components called PetitLyrics, FloatingLyrics, and Lyric Display, respectively. Our work differs from these prior studies (except for TextAlive, upon which we built this work) in two aspects. First, our intelligent web-based workflow enables registration of new musical pieces and corresponding lyrics text. Second, the existing commercial APIs focus on lyrics but do not provide direct access to timing information, whereas our APIs provide unified, direct access to lyrics text and timing information as well as other musical elements. These two novel aspects are the key enablers for programmers to develop lyric apps for musical pieces of their choice.

Automatic generation techniques and frameworks for creating synchronized audio and visual media content have also been studied. MusicStory [40] generates a video from a musical piece by retrieving its lyrics online, searching for relevant images in a specified dataset, and showing the images at intervals matching the music's pace. Crosscast [43] generates a video from an audio travel podcast by transcribing the content and finding relevant images via natural language processing (NLP) and text mining. A recent automatic generation technique [23] infers a preferred shot-type sequence from music, which can potentially be used for automatic concert video mashups. Songle Widget [13] and Songle Sync [17] are web-based platforms that support music playback synchronization with multimedia performances. Our work can be used in conjunction with these, thus opening up novel end-user interaction opportunities that go beyond static multimedia content.

3.3 Creative Coding and Interactive Multimedia

There are various programming tools for generating multimedia content and designing interactions with content. Such creative use of programming techniques for artistic purposes is often called "creative coding," or it may have a different name, depending on the application domain and context, such as generative art [32], algorithmic music [27], generative design, and procedural modeling.

Processing [37] and openFrameworks [44] are popular creative coding tools for prototyping interactive applications. They are often used to generate beautiful images, videos, and interactive graphics. Creative coding libraries for the web such as p5.js help build graphical applications. They have become popular by enabling users to easily run applications on smartphones and other devices with web browsers. Because such libraries are helpful for generating visuals in lyric apps, our framework is designed to work with a programmer's creative coding library of choice, while still providing support to address the distinctive challenges of lyric app development.

Researchers have also explored the web's potential to facilitate new media formats for creative use. D3.js [2] was developed for data visualization, and subsequent works such as Idyll [3] explored a workflow to build web articles with interaction capabilities. Webstrates [19] envisioned shareable dynamic media and enabled collaborative data authoring and interaction programming. Videostrates [20] and Codestrates [35] were built on top of Webstrates and explored domain-specific interactions for video authoring and literate programming. Our work is in line with these works, in that it defines a novel media format of lyric apps and explores the related design space by deploying a framework in the wild.

In the context of musical applications, commercial musical pieces have been released as smartphone applications that enable end-users to dynamically edit musical and visual content, such as Bjork's Biophilia [1] and Brian Eno's Reflection [29] (for more examples, see Levtov's article on algorithmic music for mass consumption [22]); however, these were built with general programming tools. In contrast, RjDj [36] was one of the first attempts to enable playing and editing algorithmic musical pieces based on Pure Data on smartphones, to explicitly support the production and distribution of interactive music. A more recent example is variPlay [30], which focuses on recorded music tracks instead of algorithmic music. Unlike these prior works, our framework does not allow editing of existing musical pieces, but it adds interaction capabilities around them to produce a novel lyric-driven artistic experience.

4 LYRIC APP FRAMEWORK

To address the distinctive challenges of developing lyric apps, we propose Lyric App Framework. In this section, we introduce its streamlined development experience, as illustrated in Figure 1, which is enabled by three core features that correspond to each of the three challenges discussed in Section 2.3.

4.1 Web-based Development Workflow

To address the first challenge of developing lyric-driven visual performances synchronized with music playback, we propose an intelligent, flexible, web-based development workflow.

4.1.1 Semiautomatic Analysis. Our framework analyzes musical pieces and lyrics text and provides useful information for lyric app development while allowing manual corrections. Table 1 lists the types of analysis results that are available.

Table 1: Types of analysis results available to lyric apps.

Analysis type	Explanation
Lyric timings	Timings when the vocalization of each phrase, word, and character starts and ends.
Part of speech	Part of speech of each word in the lyrics.
Vocal amplitude	Time-series changes in vocal amplitude.
Valence/arousal	Time-series changes in valence and arousal values indicating emotional mood transitions.
Musical beats	Timings of each musical beat, corresponding to a quarter note.
Chords	Series of chord progressions with timings.
Chorus segments	Timings when each repetitive chorus segment starts and ends.

First, programmers and musicians can use the framework's web interface to register pairs of musical pieces and lyrics text for use in lyric apps. They can either upload new music and lyrics or specify the URLs of existing online music and lyrics with an appropriate license for derivative use. After about ten minutes, automatic analysis results become available on the framework's servers, and the musical piece becomes available for use in lyric apps.

Then, not only programmers but also musicians can review the lyrics and other music-structure timings on the web interface and correct them if necessary. We previously examined such a crowdsourcing approach in a web service called Songle [12, 33]. Following its success, we added novel features including lyric timing analysis and corrections, part-of-speech (POS) estimation of lyric words, and revision control for timing information.

4.1.2 Musical Piece URL as Key. Our framework assumes that every musical piece has a canonical URL, which is the unique key to playing the piece and retrieving its lyrics text and analysis results. A programmer can implement a user interface for the end-user to choose a musical piece so that the user's lyric app runs with multiple musical pieces. Additionally, the programmer can specify optional numerical revision identifiers for each musical piece to retrieve a particular version of the analysis results, thus making the lyric app's behavior reproducible and resilient against malicious edits. This simple URL-based mechanism with revision control of automatic analysis and manual edits will be useful for future work on frameworks that add interactivity to static media.

Furthermore, our framework allows a musical piece to be registered at any time, thus contributing to a flexible development workflow. For instance, a programmer can start development with an existing musical piece and easily replace it later with a newly registered one. Similarly, a musician can first upload a musical piece with restricted access and later make it publicly available with a different URL to promote its release.

4.2 APIs for Building Lyric App Interactions

To address the second challenge of building user interactions in lyric apps, we developed the TextAlive App API, which comprises two APIs to address this challenge.

4.2.1 Event-driven API for State Management. While a lyric video is always tied to a single musical piece, a lyric app can be designed either for a particular piece (just like a lyric video), for a particular set of pieces (e.g., a rhythm game with a supported song list), or for any musical piece (e.g., a music player application with interactive

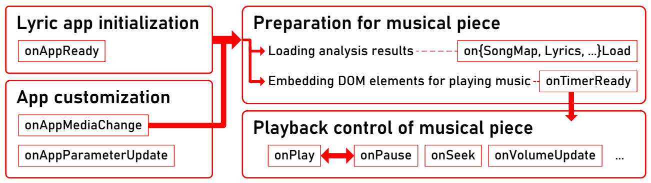

Figure 2: Our framework provides an event-driven API to help manage complex states in a lyric app.

graphics). In addition, a lyric app can be designed to allow or disallow particular end-user operations. For instance, whereas a music player allows pause and seek operations, a game typically disallows both. For narrative purposes, a lyric-video-like application might allow pausing but not seeking.

To help implement all of these interaction scenarios, we define a shared set of possible lyric app states as shown in Figure 2, and we provide an API for notification of every state transition. In particular, the JavaScript class `Player` serves as the entry point to access the framework's features.

Once an app instance successfully connects to a framework server, the `onAppReady` event is triggered. If a programmer wants to design a lyric app for a particular musical piece, it should be explicitly loaded here by calling `player.createFromSongUrl(songUrl)`. Otherwise, the framework looks for a fallback URL to load, which can be specified by the programmer or musician via a web interface on the framework called Lyric App Customizer (detailed in Section 4.3.2). If no fallback URL is found, the framework waits for a call to this API, which allows the programmer to show the end-user a list of musical pieces. An additional call to this API loads the newly specified musical piece, which allows the lyric app to switch between musical pieces.

After the loading process, the programmer is responsible for controlling music playback. It is up to him or her whether to implement a user interface to allow end-user operations of pausing and seeking, depending on the chosen interaction scenario. In any case, the framework provides notifications of various media player events, such as `onPlay`, `onPause`, `onSeek`, and `onTimeUpdate`.

4.2.2 Time-driven API for Time-sensitive Interactions. Various graphical user interface (GUI) frameworks and our prior toolkit work on time-coded musical elements [17] provide event-driven APIs, but for lyric apps, we suppress the support for such musical events for three major reasons. First, these events are not good for future planning because they are usually only triggered right after something happens. For example, they cannot make a lyric word start sliding into the screen *well before* it is vocalized. Second, they can only be tied to a single kind of musical element, whereas attractive visuals typically involve multiple kinds of musical elements. For instance, consider ripple animations that appear around *phrases* when they are vocalized. They might fade in and out in synchrony with the *beat* and change color during the *chorus*. Third, they tend to result in poor time precision. For our framework to emit musical events, it must implement an event loop that periodically checks whether the current playback position exceeds the timing of a musical event. Such an implementation may emit a musical event with a delay of nearly the event loop interval in the worst case[1]. In summary, APIs that do not guarantee time precision are not helpful and could be harmful, especially for novices who could be confused by imprecise behavior.

Thus, to support future planning, easily account for multiple kinds of musical elements, and achieve high timing accuracy, we propose a *time-driven API* that accepts a position argument in milliseconds and returns time-coded information (see Appendix A for illustrated explanations). For example, seeking a phrase that will be vocalized five seconds later is as easy as calling `player.getPhrase(player.position + 5000)`. Then, after beat information is retrieved by `getBeat` with the same argument, it can be combined with phrase and chorus information to generate visuals.

This API works well with existing graphics and creative coding libraries that help generate interactive visual content. These libraries typically have mechanisms with which a programmer defines a rendering function to draw visual content, including `requestAnimationFrame()` in the Web API, `Components.render()` in React, `loop()` in p5.js, and `renderer.render()` in Three.js. The programmer can then simply call the *time-driven API* within such rendering functions to guarantee that displayed content is always up to date with the music playback.

[1]In addition to this theoretical limitation, current browser implementations have a practical limitation in that execution timings can be affected by various factors such as the CPU overload, inactive browser tabs, and power saving on a low battery.

4.3 Mass Distribution of Lyric Apps

To address the third challenge of delivering lyric apps to end-users, our framework enables programmers and musicians to distribute a lyric app via a website, as shown in Figure 3.

4.3.1 Production-ready Deployment. The distribution of interactive media content has traditionally suffered from interoperability and scalability issues. We consider the web standard to be the most practical solution so far, as discussed in the context of mass distribution of interactive music [22].

With our framework, as long as a programmer can build a website (with client-side HTML and JavaScript code in a minimal case, up to a full-stack application including client-side stylesheets, static media content, and server-side code), a production-ready lyric app can be created for mass distribution. Thus, programmers do not have to deal with external resources, including musical pieces, lyrics text, and related analysis results, because music/video sharing services and our framework servers directly deliver them to the end-user. Nevertheless, these external resources can be optionally embedded in a website to avoid access to external services and servers, which enables deployment to a content distribution network (CDN) for even higher availability and performance.

4.3.2 Lyric App Customizer. Our framework has a special web interface named TextAlive Lyric App Customizer, through which programmers and musicians can customize an existing lyric app (Figure 3). The customizer page opens the lyric app in its `<iframe>`

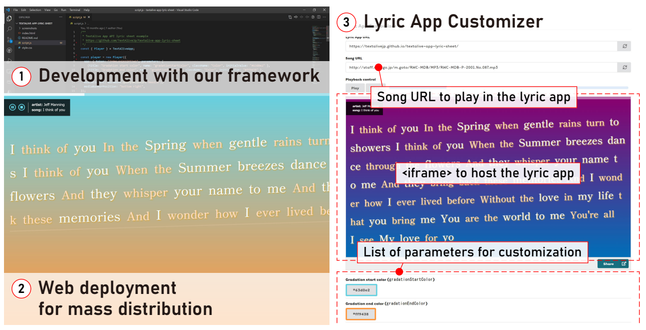
1 Development with our framework
2 Web deployment for mass distribution
3 Lyric App Customizer
Song URL to play in the lyric app
<iframe> to host the lyric app
List of parameters for customization

Figure 3: Our framework supports deployment and post-deployment customization to promote new musical pieces.

component and maintains communication with the app to enable interactive customization. When the creator is satisfied, the customization results can be stored in the app URL's query string, e.g., `?ta_song_url=NewUrl`. This enables musicians to use existing lyric apps to promote their musical pieces. Note that, as explained in Section 4.2.1, a lyric app may also be implemented to support specific musical pieces, without the customization capabilities.

In addition to the URL of a musical piece to play, the programmer can define a list of parameters with types (a number, color, string, or list of values with optional labels) when initializing the player instance; then, event listeners can be implemented to respond to customization events sent by the customizer. The list automatically appears on the customizer page and enables musicians to make their own custom versions of lyric apps for more effective promotion.

5 IMPLEMENTATION

In this section, we give a brief overview of the framework's implementation (publicly available at https://developer.textalive.jp).

5.1 Framework Web Servers

Two dedicated servers implement the web-based workflow described in Section 4.1, one for the web API based on representational state transfer (REST), and the other for client web applications. These servers are connected to a database that stores lyric app information such as authors and URLs. In addition, the REST-based API server is connected to external servers that are used to analyze and provide lyric timings and additional musical element information. We implemented the external analysis servers by using and extending our web service named Songle [12, 33] based on music understanding technologies. The extensions include lyric timing analysis, POS estimation of lyric words, revision control for timing information, and an online storage feature for uploading musical pieces and lyrics. The lyric timing analysis was initially based on a prior work [9] and later replaced by a machine-learning-based method with better performance. The POS estimation uses Mecab [21] for Japanese words and the Natural Language Toolkit (NLTK) [24] for English words.

The current implementation supports lyrics in Japanese, English, or a mixture of Japanese and English, and it supports musical pieces up to ten minutes long (longer than the duration of most songs with lyrics). Analysis results can be freely corrected by the users of client applications; the corrections are then recorded with unique numerical revision identifiers. As introduced in Section 4.1.2, programmers can specify revision identifiers to make their lyric apps' behaviors reproducible.

We initially released our framework for the Japanese community, and we assumed that lyrics would primarily be in Japanese or a mixture of Japanese and English. In Japanese writing, each character forms a complete syllable, and it is completely sensible to analyze the timing of each character. In contrast, in English writing, each character is part of a syllable, and people typically care about word-level timing rather than character-level timing. As such, our current implementation analyzes character-level timings for Japanese lyrics and word-level timings for English lyrics. Character-level timings for English are also provided by linearly interpolating between the timings of a word's start and end.

5.2 Framework Web Applications

The framework's client web applications have three roles. First, they provide user interfaces for the web-based development workflow, including registration of existing pairs of musical pieces and lyrics, uploading of new pairs, and viewing and correction of automatic analysis results (Section 4.1). Second, they provide tutorial content and introductory materials, which will be explained in Section 6.1.2. Third, they provide Lyric App Customizer (Section 4.3.2), which operates as follows.

First, a programmer can specify a lyric app's URL and an optional song URL to play in the app. Then, Lyric App Customizer loads the app's URL, with an optional query parameter specifying the song URL, in its `<iframe>` component. Next, the framework's client library in the lyric app triggers an `onAppReady` event (Figure 2), and the musical piece becomes playable in the lyric app. Lyric App Customizer keeps communicating with the lyric app to update information on the playback position and customization parameters (triggering `onAppParameterUpdate` events in the lyric app when necessary) through the Web Messaging API, which is a web standard for inter-frame communication.

5.3 Client Library for Building Lyric Apps

To use the TextAlive App API for building lyric apps, a programmer can use an "npm" package or load the library with a `<script>` tag.

The media information here comprises metadata that is specified upon registration to a framework server (e.g., the musical piece URL, lyrics URL, artist name, and song title) and runtime information retrieved from the embedded media. For playing audio, we currently support audio files (uploaded to a framework server or hosted by the musician or programmer) and videos uploaded to video-sharing services (YouTube and Niconico). Our client library wraps the original APIs (HTML DOM API for audio elements, YouTube Player API, and Niconico Video Player API) and provides a unified API for playback control. As the original APIs only report playback positions intermittently, the library stores the last-reported position along with the UNIX timestamp to enable the programmer to access the current precise playback position at any time.

The resulting time-coded information is represented as object instances of lyric and musical element classes. The framework's automatic analysis has varying time windows (e.g., lyric vocalization timings have a 10-ms precision, while valence and arousal analysis has a 15-s precision), and the classes have internal methods to appropriately interpolate values for analysis results with larger time windows. The *time-driven API* looks for the corresponding information in the objects, which are stored as sorted arrays, by using a fast binary search algorithm.

6 DESIGN SPACE EXPLORATION

To explore the design space of the novel media format of lyric apps, we made the framework publicly available on September 18, 2020 with 11 example applications and collected 52 new applications through an annual programming contest held twice. This section describes the procedure for this design space exploration. We also report the results, which revealed eight representative categories of lyric apps, and the usage statistics and a user study to briefly evaluate the framework's effectiveness.

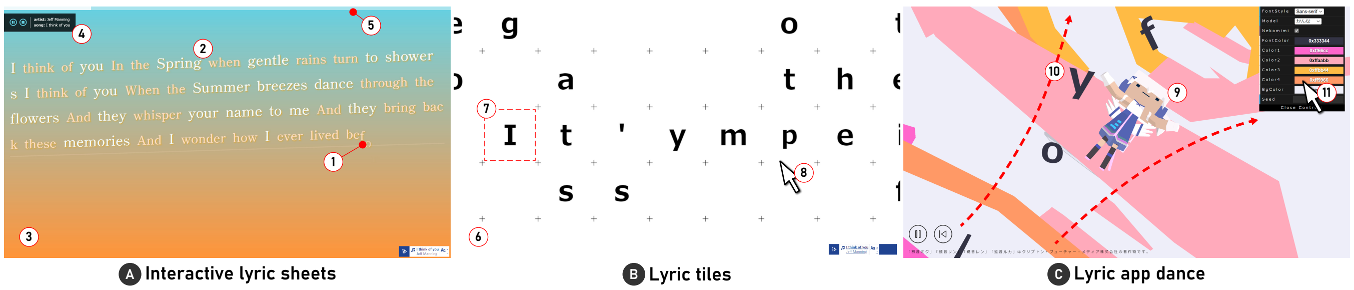
I think of you In the Spring when gentle rains turn to shower
s I think of you When the Summer breezes dance through the
flowers And they whisper your name to me And they bring bac
k these memories And I wonder how I ever lived bef
A Interactive lyric sheets
B Lyric tiles
C Lyric app dance

Figure 4: Three of the 11 example applications that we built and provided on the framework via open-source distribution (more details in Appendix B).

6.1 Programming Contests

We conducted annual programming contests in 2020 and 2021 in collaboration with Crypton Future Media, Inc., a music technology company famous for the singing voice synthesizer *Hatsune Miku*. The programming contests were held as part of a larger annual exhibition called Magical Mirai [7]. The exhibition included live music and stage performances that attracted people who enjoy music and participate in creative culture.

The contests had three primary goals: to examine the design space of lyric apps with open-minded programmers interested in creative culture, to verify the framework effectively supports non-author programmers with diverse expertise in programming, and to enliven the lyric app community.

6.1.1 Rules and Procedures. The contests were called the Magical Miral 2020 & 2021 Programming Contests [5, 6]. They were open to the public and held completely online, except for the winner announcements at the annual exhibition stages. We provided a dedicated set of musical pieces and lyrics with appropriate permission for online streaming so that the contestants could focus on development. We did not hold any in-person or online hands-on activities during the contests. Instead, we provided introductory materials as described in Section 6.1.2. We occasionally answered technical questions on Twitter and Gitter. We asked the contestants to submit the source code of static web applications and their build instructions so that the lyric apps could be safely hosted on our standard web servers and anyone could execute and play with them.

We selected the winners (one winning team and three honorable-mention teams per year) according to three evaluation criteria: the aesthetic quality of the lyric-driven performance, innovative use of technology, and the technical completeness and compatibility of the implementation. These criteria were listed on the programming contest websites from the beginning. The contestants voluntarily attended, and the winners received non-monetary awards: a certificate, assorted goods, and a ticket to enjoy a live music concert, with a market price of several hundred dollars.

After announcing the winners at Magical Mirai’s annual exhibition stage [7], we distributed an online questionnaire to the contestants that asked for feedback on the programming contest. The questionnaire results will be reported in Section 6.3.3.

6.1.2 Tutorial Content and Example Applications. When we made the framework available to the public, we also published tutorial content and introductory materials on the framework website (https://developer.textalive.jp). The tutorial provided an overview of the framework and a step-by-step walkthrough to start developing lyric apps. The introductory materials included API documentation, explanation videos, and open-source examples. We implemented 11 examples to demonstrate the framework’s use, available on GitHub (https://github.com/TextAliveJp) under an MIT License. Here, we briefly explain three of them, shown in Figure 4.

The first example, interactive lyric sheets, gradually shows lyrics text along with the music playback through CSS transitions ①. For improved aesthetics and readability, the text has varying font colors that depend on the POS ②. Besides reading the lyrics text, the user can click any part of it to jump to the corresponding position in the musical piece. With Lyric App Customizer, a user can customize the background gradation and share the resulting lyrics sheets with others ③. This lyric app also has basic playback control and copyright display ④, as well as a translucent seekbar ⑤. While we sought to write comprehensible code and inserted many comments, this app amounts to just 237 lines (160 excluding comments.)

The second example, lyric tiles, shows a two-dimensional grid of tiled squares via the HTML5 Canvas API ⑥. Each square can display a single character of lyrics text ⑦. The user can use mouse or touch interaction to navigate the grid smoothly, and the lyrics text gradually fills the neighboring tiled squares in response to navigation ⑧. Every time the user plays a musical piece, he or she can improvise new paths to navigate the grid and fill the squares, thus yielding a creative experience similar to drawing pixel art.

The third example, lyric app dance, shows a small, chubby (so-called “chibi”-style) 3D character model in a stage scene by using Three.js ⑨. The character dances to the music playback while particle effects and floating lyrics enliven the scene ⑩. The motion patterns are dynamically generated in response to various musical elements such as beats and repetitive segments in a musical piece. With Lyric App Customizer, a user can customize the character model and multiple color properties of the stage scene. Even when the customizer is not available (i.e., if the lyric app is executed by directly specifying its URL), the lyric app shows its own parameter tuning interface for interactive experience ⑪.

While we carefully designed these examples to demonstrate the framework’s features, we wanted them to be compatible with various browser environments on personal computers, tablets, and smartphones, as they served as introductory materials. This policy prevented us from using several interesting web-standard APIs for technologies such as geolocation, accelerometers, and Bluetooth.

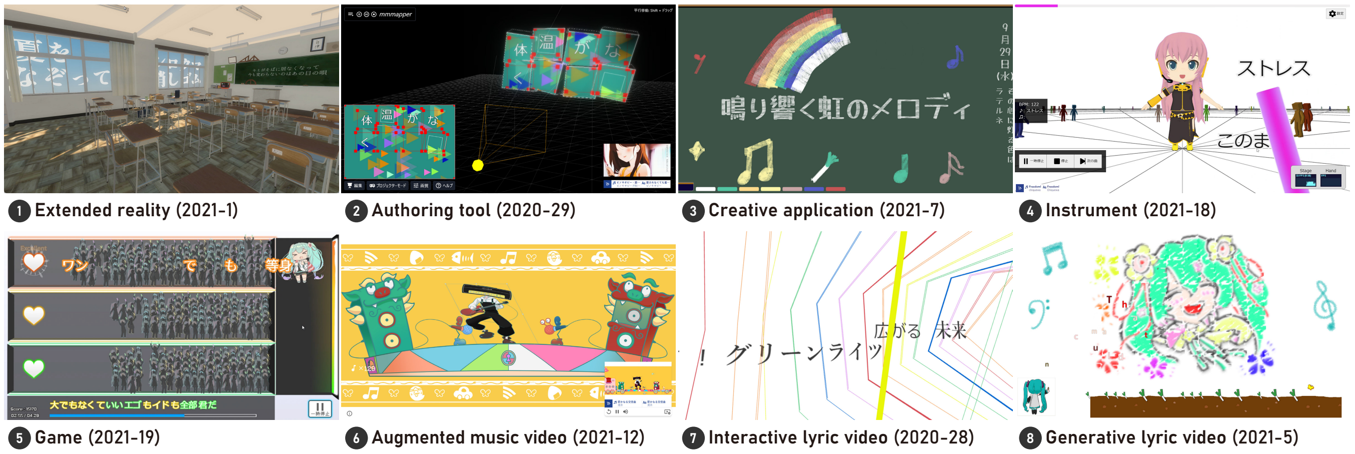
1 Extended reality (2021-1)
2 Authoring tool (2020-29)
3 Creative application (2021-7)
4 Instrument (2021-18)
5 Game (2021-19)
6 Augmented music video (2021-12)
7 Interactive lyric video (2020-28)
8 Generative lyric video (2021-5)
鳴り響く虹のメロディ
ストレス
このま
グリーンライツ
広がる
未来

Figure 5: Representative examples of the eight lyric app categories collected in the programming contests, with screenshots courtesy of Crypton Future Media, Inc. Details and more examples of the diversity of visual styles are given in Appendix C.

6.2 Eight Lyric App Categories

The two contests lasted 47 and 77 days and attracted 32 and 20 applications, respectively. It is unclear why the second year attracted fewer applications despite having a more extended development period. A possible explanation is that the second-year contestants saw the quality of the lyric apps by the first-year winners, which caused some novices to hesitate to submit their applications.

All of the submitted lyric apps were complete and executable. To understand the characteristics of the surprisingly diverse set of lyric apps, we wrote short text descriptions of them (see Appendix C). As a result, we revealed eight representative categories of lyric apps, which are explained below. Figure 5 shows notable examples of each category. Later, we provide a more detailed analysis of the source code (Section 6.3.2).

We generated the eight categories via the following three steps. First, by picking common keywords that appeared multiple times while excluding keywords that were too common (e.g., kinetic typography), we obtained three initial categories of "extended reality," "generative lyric video," and "game." Then, we carefully examined the lyric apps that did not fit into any of these categories to obtain the "creative application," "instrument," and "augmented music video" categories. While all examples fell into at least one of these six categories, we noticed that some categories could be better explained when they were divided into subcategories based on the interaction capabilities. Specifically, some generative lyric video applications did not accept user input, but other generative applications were interactive and thus fell into a new "interactive lyric video" category. In addition, some creative applications did not allow users to seek or rewind the playback position and instead made them focus on an improvisational experience. In contrast, other creative applications allowed users to repetitively listen to musical pieces and made them focus on content creation, thus yielding a new "authoring tool" category. In the end, there were nine extended reality applications, six authoring tools, six creative applications, three instruments, six games, two augmented music videos, and five interactive and 16 generative lyric videos.

6.2.1 *Extended (virtual or augmented) reality.*

This sort of application focuses on an immersive experience in a virtual 2D/3D scene, or it adds a visual layer of lyric-driven performance to the real world with a camera input. Users can thus feel as if they are in the world of a musical piece. One example was dedicated to a specific musical piece and showed a classroom in a school, where the user could freely navigate and find hidden perks related to the story in the lyrics (2021-1). Another example used the WebVR API and allowed the user to dive into a virtual space with supported VR headsets (2020-25). The augmented reality applications used various libraries, such as AR.js to overlay a cube with lyric characters in front of camera images (2020-8), and Tensorflow.js to detect the user's posture and improvise visual effects on a camera image (2020-31).

6.2.2 *Authoring tool.*

This sort of application allows users to create and elaborate a derivative work synchronized with a musical piece, and optionally to share it with others. One example (2021-3) allowed the user to place various visual components that comprised lyric videos on the screen, thus creating a music video on the fly. The components included placeholders for the lyrics text, particle effects, animated character images, and so on. A more complex example (2020-29) was an authoring environment for projection mapping. It allowed the user to place visual components on predefined surfaces in a virtual 3D space. It then showed new browser windows displaying animations for the predefined surfaces, which could be used for projection mapping in the real world. Another example (2020-11) showed a 3D singing character and allowed the user to configure various properties to animate her.

6.2.3 *Creative application.*

This sort of application allows the user to interact with a musical piece creatively and actively. Unlike an authoring tool, this category typically does not allow the seek operation and instead focuses on improvisational experiences. Such experiences included drawing illustrations (2020-26, 2021-7), touch interactions to put musical notes on the screen that would start dancing during the chorus (2021-9), and control of character movements in a virtual space, whose trajectories were visualized with specific brush-stroke styles (2021-11, 2021-15).

6.2.4 Instrument. Here, the application serves as an instrument that actively participates in the musical performance. One application captured the user's hand motions by analyzing camera images with MediaPipe.js, and it then rendered virtual glow sticks (2021-18). Another example used the Web MIDI API to receive MIDI signals from hardware devices and allowed the user to play musical instruments along with the music playback (2020-3). A more straightforward approach played prerecorded audio effects in response to user actions (2021-8).

6.2.5 Game. This sort of application has specific rules and scoring systems to encourage the user to achieve goals and get high scores. Through our framework, the user can choose the musical piece to play. All of the collected examples in this category implemented a musical rhythm game in which the user had to interact quickly with lyric characters. One example allowed a player character on a live performance stage to hit approaching lyric characters (2021-19). Another example showed lyric phrases on dynamically generated winding paths and allowed the player to touch lyric characters (2021-16). Every example had dedicated interaction between the user and lyric characters.

6.2.6 Augmented music video. This sort of application shows an original music video and augments its playback with an interactive experience. One example, which was dedicated to a specific musical piece, displayed the music video on the bottom right and reproduced almost identical scenes with dynamically rendered visual components (2021-12). The video and the reproduced scenes were mostly synchronized, and the user could interact with specific visual components related to the underlying lyrics. For instance, when the lyrics described a puppet and the video showed a morphing puppet, the reproduction showed a puppet that the user could morph by mouse or touch interaction. Another example could play any musical piece with the corresponding music video, and it offered a theater-like scene in which the lyrics and other visual components appeared and were animated in synchrony with the music and video playback (2020-6).

6.2.7 Interactive lyric video. This sort of application dynamically renders lyric videos and adds some interactivity. In one example, the lyrics scrolled while the song played, and the user could grab the lyrics with a mouse or touch operation to change the playback position (2020-28). In another example, the words of vocalized lyric characters gradually gathered together in the center of a 3D space, and the lyrics repeatedly came and went as the music playback progressed (2020-7). The user could rotate the gathered characters with a touch operation and view them from any angle.

6.2.8 Generative lyric video. This sort of application dynamically renders lyric videos and may support variable aspect ratios and frame rates, depending on the user's hardware. It typically allows the seek operation. Otherwise, the user cannot interact with the content. Many applications in this category provided a similar experience to lyric videos but in an artistic style that was intrinsic to the user. The artistic style reflected a tight relationship between the transformation of visible graphics and the timings of lyrics and other musical elements. Certain exceptional examples did not look like ordinary lyric videos. For instance, one example divided the musical piece and an illustration into equal numbers of segments; then, it gradually placed the illustration segments on the screen as the music playback progressed, and the illustration was complete when the playback ended (2021-5). Another example used an NLP technique to replace lyric words with different words that had the same phonetic counts, thus generating a parody song (2020-32).

6.2.9 Limitations of Programming Contests. Regarding the limitations of the programming contests, first, we only observed successful applications. They gave evidence that a certain number of programmers could use the framework to build lyric applications, but they did not necessarily indicate that novices could use it. Still, it is noteworthy that several contestants reported in the post-contest questionnaire that they were indeed novices, in some cases, without any prior experience in building web applications.

Second, the competitive format of the contests put significant pressure on the contestants to develop applications that would appeal to the jury. Here, we saw two potential concerns impacting the validity of this study: (a) such pressure could have prevented a fair evaluation of the framework's effectiveness; and (b) the pressure could have affected the variety of applications. Regarding issue (a), as the framework developers, we were members of the jury, which might have dissuaded the contestants from complaining about technical issues. We aimed to address this issue in three ways: (1) We ensured that half of the jury members were not framework developers. (2) We opened an online forum for open discussion of technical issues, and we clarified that the contest format did not evaluate the development process, thereby relieving the contestants from feeling reluctant to complain about technical issues. (3) We investigated the submitted source code and found no significant mitigations for technical issues.

Regarding issue (b), in comparison with immediate or short-term studies in controlled setups, which were criticized by a recent study on creativity support tools [38], our case potentially attracted a wider variety of participants and gave them more opportunity to master the framework. In other words, this can be considered a longitudinal study because the framework was publicly available for over two years. In addition, we carefully considered the evaluation criteria for the programming contest. If the criteria only focused on artistic quality, the contest would have been like a creative competition with the contestants focused on aesthetic qualities. On the other hand, if the criteria only focused on technical completeness, the contest would have been like a programming competition focusing on technical challenges. Accordingly, we specified the criteria (Section 6.1.1) to balance these two aspects. We believe this left the choice to the contestants on how to develop the aesthetic qualities of their apps and approach the creative and technical challenges.

6.3 Framework Evaluation

In this subsection, we report additional evaluations to understand the effectiveness of the framework.

6.3.1 Usage Statistics. The public release of our framework was on September 18, 2020. To measure its use for lyric app development, since September 22, 2021, we have asked users to generate an access key to identify each lyric app when they accessed the REST API server. As of September 13, 2022, the registration database counts

448 lyric apps developed by 396 unique users. To maintain compatibility, we allow REST requests without the access key, and as such, there could be more outdated apps in the wild.

GitHub reports that the framework's repository has 78 stars (programmers' reactions expressing their appreciation and interest), and that ≥ 67 projects with public code on GitHub depend on the framework. Moreover, 30 contestants open-sourced their lyric apps to enliven the community. Overall, these statistics indicate that the framework has been steadily used without significant issues.

6.3.2 Source Code Analysis. We analyzed the source code of all 52 contest submissions, of which 16 had codebases that were partially derived from our examples. Still, their diffs were so significant that, in most cases, they only shared the build pipeline (Parcel to pack the program) with the original examples. Everything else, such as the interaction design and visual styling of the lyric apps, was done by the contestants.

We could confirm that the framework's *time-driven API* worked with various graphics APIs and creative coding libraries, including p5.js, Three.js, Create.js, PixiJS, glslify, GSAP, PlayCanvas, and Phaser (see Appendix C for a complete list). In addition, many contestants used GUI libraries such as jQuery, React, Vue.js, and Svelte to help make their lyric apps interactive. Some also used user interface components (e.g., Tweakpane and dat.gui) to allow end-user customizations. Contestants also embedded static images, web fonts, and stylesheets to enrich their visual styles, and some used Lottie, a library to import animations based on Adobe After Effects. These diverse results demonstrate not only the framework's capability but also its low threshold and high ceiling in terms of development skill.

We investigated the source code further and found that many contestants implemented their own "scene manager." In a typical case, the manager was initialized upon loading a musical piece. It then analyzed the macro-level musical structure (e.g., intro, repetitive segments, and outro) and prepared for music playback by instantiating visual-effect components. During playback, the scene manager managed the state transitions between musical scenes and the switching between visual-effect components. We discuss future work to support this design pattern in Section 7.1.3.

6.3.3 Qualitative User Feedback. We collected qualitative user feedback from the contestants with a web-based questionnaire that asked for general comments in freeform text. We received 21 and 15 responses in the respective years. The contestants reported that they greatly appreciated the programming contests as opportunities to contribute to creative culture. They also provided positive feedback on the framework, as well as a small number of non-fatal bug reports that could easily be addressed.

The second year's questionnaire asked for more details about the contestants' application development experience. The answers were diverse, ranging from no prior experience to six years of professional JavaScript programming experience; thus, the answers aligned with our source code analysis, which indicated a low threshold and a high ceiling of the framework. Notably, one-third of the respondents had virtually no prior experience with JavaScript programming, and we examine their motivations in Section 7.2.

The second questionnaire also asked whether the contestants were satisfied with the *time-driven API*, and whether they felt the need for new event listeners for lyric and musical elements, such as `player.addEventListener("phraseEnter", listener)`. The answers were mostly positive regarding the current API design. While the contestants did use the event listeners for state management (Section 4.2.1), they considered the *time-driven API* reasonable and useful for real-time multimedia performance. One contestant considered more event types would be better, but the rest did not see this need. Another contestant mentioned wanting to plan complex scene changes, for which the current API was insufficient. This echoed the need for the scene manager, as revealed in the source code analysis.

7 DISCUSSION

As we are the first to propose lyric apps, this particular media format is currently a niche format; however, this work also contributes unique, generalizable knowledge to three diverse audiences.

First, music-related researchers and the music industry have discussed the importance of enabling interactive user experiences around musical content (e.g., [10]), and lyric apps are a promising solution. Our framework is the first to provide a streamlined lyric app development experience, which marks a significant technical milestone in pushing lyric apps toward the cultural mainstream. Second, we developed the time-driven API, thus providing insights for future work on general time-sensitive interactions, as explained below in Section 7.1. Third, our framework allows musicians to customize and distribute interactive musical applications. As this makes programming matter to them, it can lead to an exciting line of future work on connecting programmers and artists through application development, as discussed in Section 7.2.

7.1 Insights for Time-driven API

As briefly mentioned in Section 6.3.3, we received positive feedback on the *time-driven API*, with which programmers could successfully implement interactive effects along a musical piece's timeline. The source code analysis revealed an anti-pattern, a successful design pattern, and the importance of reusable components, as discussed below in Section 7.1.1. These observations led us to insights on potential improvements to support lyric app modularity, as explored in Section 7.1.2 and Section 7.1.3. These insights have the potential for application in programming general time-sensitive interactions, such as synchronized audio and visual performances and physical interactions using sensors and actuators.

7.1.1 Reusable Components in Lyric Apps. Almost all of the collected lyric apps generated visuals with rendering functions provided by graphics APIs and creative coding libraries (see Section 4.2.2). Such generative algorithms could be implemented via a single, huge rendering function composing various kinds of information, but we consider this an anti-pattern because it typically results in monotonous visuals. Among the analyzed codebases, we observed a successful design pattern that implemented that functionality through many subfunctions, which served as reusable components and improved the code's modularity.

7.1.2 Time-range-driven API. In implementing reusable components for lyric apps, it is noteworthy that a component's lifecycle is often tied to a specific time range when a musical element is audible, such as a beat or a singing voice in a phrase. In the submitted code, we found a recurring pattern of checking whether the playback position entered or left such ranges. In one example, a particle manager stored the result of a previous call of `player.findBeat` and compared it with the last call to decide whether to instantiate or destroy particle effects.

Hence, we can generalize this design pattern and support it in our framework with a new API to calculate changes like this:

```
player.findBeatChange(previousPosition, currentPosition);
// returns the following tuple:
// { currentBeat, enteredBeats, leftBeats }
```

While common event-driven APIs notify an application of the occurrence of a one-time event, this novel time-range-driven API would look for all the time-coded information relevant to the specified time range (see Appendix A for illustrated explanations).

Such an extension of the proposed *time-driven* API would have additional benefits for lyric apps. First, it would maintain the framework-agnostic nature of the getter methods, thus supporting the use of various graphics APIs. Second, it would naturally support querying of changes in a custom time range, which would be useful for implementing animated transitions. For instance, to start a transition, a query to search for a chorus 5 seconds after the current playback time could be written like this:

```
player.findChorusChange(previousPosition, currentPosition
      + 5000);
```

7.1.3 User-defined Time Ranges. Finally, many of the submitted applications divided a musical piece into multiple time ranges corresponding to specific scenes in a lyric-driven performance. Typically, a scene manager was implemented to manage the lifecycles of reusable components in response to scene changes. Our framework cannot predefine such time ranges, because they are the creative outcome of the programmer's effort to understand the structure of a musical piece. To aid such scene management, we could extend the framework with an API to define time ranges and to query changes in their status, e.g., `player.createTimeRange(startTime, endTime)` and `player.findTimeRangeChange(timeRange, curPos, prevPos)`.

7.2 Addition of Interactivity to Existing Media

Lyric apps are not built from the ground up but add novel interactive capabilities to existing musical pieces. Here, we discuss the benefits of this augmenting approach and interesting topics for future work.

7.2.1 Kickstarting of Creativity. We observed that the framework appealed to a wider community than the typical creative coding community, and it encouraged the contestants to publicly present their outcomes. One contestant (2020-10) commented that they usually did not (or even could not) participate in this kind of competition. However, their love for music drove them to develop and submit a lyric app. While there is increasing interest in creative coding culture, not all programmers feel confident that they are "creative" enough to enjoy creative coding.

Lyric apps allow such programmers to rely on existing musical pieces for the "creative" part and thus kickstart their programming activities. Programming itself is a creative activity, and we were pleased to see that lyric apps gave opportunities for more people to unleash their creativity. We believe that lyric app development could form a useful part of a computer science education curriculum. Furthermore, our future work could also investigate mixtures of creative coding and other kinds of static media content.

7.2.2 Programming as Communication. In developing lyric apps, programmers may listen to musical pieces and read lyrics many times. This experience may inspire them to come up with ideas for compelling, lyric-driven performances. Just as lyric videos enable collaborations between musicians and motion graphic designers, lyric apps present a novel opportunity for musicians to collaborate with programmers. Programming environments are often designed as mere computational artifacts, but they can be far more social and collaborative when programming activities are part of communication [18]. Indeed, one contestant who developed an augmented music video (2021-12) contacted the original songwriter and video author, and together, they enjoyed creating derivative work.

During one of the Magical Mirai's exhibition stages to announce the winners of the programming contest, Hiroyuki Itoh, the CEO of Crypton Future Media, Inc. and an expert on the music industry, commented on the tight relationship between music and technology: *The way musicians deliver music to audiences has always been affected by inventions, from woodworking technology to manufacture pianos to networking and encoding technologies to stream music. These technologies challenge musicians, and musical content has always adapted to technological shifts. I look forward to the future of music and lyric apps co-created by musicians, programmers, and audiences.* It is not uncommon for musicians today to take on the role of a producer who envisions the best way to deliver musical content, and we foresee that musicians will be positively involved in collaborative lyric app development.

7.2.3 Future Work on Creative Culture. In the age of media convergence [14], the value of media content is not self-contained but should be considered holistically, by including the value chain of relevant derivative work created by amateur artists and social engagement by audiences.

Research on creativity support has been long criticized for the lack of studies in the wild [8]. Our framework can potentially serve as a socio-technical testbed for such phenomena, but as discussed in Section 6.2.9, the programming contest format made the contestants conservative in their technical choices, such as avoiding the use of web APIs for social interaction. Additionally, they avoided experimental web APIs (there were a few exceptions, such as the Web MIDI API (2020-3) and WebVR API (2020-25)), and they could not use server-side JavaScript engines such as Node.js. We believe that keeping testing the framework in the wild would be an interesting future work, potentially leading to novel lyric app categories and new studies on creative culture.

8 CONCLUSION

This paper has proposed a novel media format called lyric apps, examined the format's characteristics, and highlighted the technical challenges in developing such apps. We developed a framework to address the technical challenges and deployed it in the wild through two programming contestants, which revealed eight representative categories of lyric apps and insights for future work.

ACKNOWLEDGMENTS

The programming contests were administered by Crypton Future Media, Inc. We sincerely appreciate our collaboration with them, which was based on a shared belief in the potential of novel technology to support creativity. The contests were only successful because of the passionate participants. We sincerely appreciate efforts and are happy to share their creations as part of this first academic exploration in the design space of lyric apps.

We thank our collaborators at AIST for their development that contributes to the web-based workflow of the framework: Tomoyasu Nakano for lyric timing analysis, Kento Watanabe for POS estimation, and Yuta Kawasaki and Takahiro Inoue for external servers for semiautomatic analysis.

This work was supported in part by JST CREST Grant Number JPMJCR20D4, JST ACCEL Grant Number JPMJAC1602, and JST ACT-X Grant Number JPMJAX22A3, Japan.

REFERENCES

[1] björk. 2011. björk: full biophilia app suite. https://www.youtube.com/watch?v=dikvJM__zA4. Accessed on September 13th, 2022.

[2] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185

[3] Matthew Conlen and Jeffrey Heer. 2018. Idyll: A Markup Language for Authoring and Publishing Interactive Articles on the Web. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 977–989. https://doi.org/10.1145/3242587.3242600

[4] SyncPower Corporation. 2022. SyncPower. https://syncpower.jp/en. Accessed on September 13th, 2022.

[5] Crypton Future Media, Inc. 2020. Hatsune Miku "Magical Mirai" Programming Contest 2020. https://magicalmirai.com/2020/procon. Accessed on January 23rd, 2023.

[6] Crypton Future Media, Inc. 2021. Hatsune Miku "Magical Mirai" Programming Contest 2021. https://magicalmirai.com/2021/procon. Accessed on January 23rd, 2023.

[7] Crypton Future Media, Inc. 2023. Hatsune Miku "Magical Mirai". https://magicalmirai.com/. Accessed on January 23rd, 2023.

[8] Jonas Frich, Michael Mose Biskjaer, and Peter Dalsgaard. 2018. Twenty Years of Creativity Research in Human-Computer Interaction: Current State and Future Directions. In *Proceedings of the 2018 Designing Interactive Systems Conference* (Hong Kong, China) *(DIS '18)*. Association for Computing Machinery, New York, NY, USA, 1235–1257. https://doi.org/10.1145/3196709.3196732

[9] Hiromasa Fujihara, Masataka Goto, Jun Ogata, and Hiroshi G. Okuno. 2011. LyricSynchronizer: Automatic Synchronization System Between Musical Audio Signals and Lyrics. *IEEE Journal of Selected Topics in Signal Processing* 5, 6 (2011), 1252–1261. https://doi.org/10.1109/jstsp.2011.2159577

[10] Masataka Goto and Roger B. Dannenberg. 2019. Music Interfaces Based on Automatic Music Signal Analysis: New Ways to Create and Listen to Music. *IEEE Signal Processing Magazine* 36, 1 (Jan. 2019), 74–81. https://doi.org/10.1109/MSP.2018.2874360

[11] Masataka Goto, Jun Ogata, and Kouichirou Eto. 2007. PodCastle: a web 2.0 approach to speech recognition research. In *Proceedings of the 8th Annual Conference of the International Speech Communication Association* (Antwerp, Belgium) *(INTERSPEECH '07)*. ISCA, 2397–2400. https://doi.org/10.21437/Interspeech.2007-183

[12] Masataka Goto, Kazuyoshi Yoshii, Hiromasa Fujihara, Matthias Mauch, and Tomoyasu Nakano. 2011. Songle: A Web Service for Active Music Listening Improved by User Contributions. In *Proceedings of the 12th International Society for Music Information Retrieval Conference* (Miami, Florida, USA) *(ISMIR '11)*, Anssi Klapuri and Colby Leider (Eds.). University of Miami, 311–316. https://ismir2011.ismir.net/papers/OS4-1.pdf

[13] Masataka Goto, Kazuyoshi Yoshii, and Tomoyasu Nakano. 2015. Songle Widget: Making Animation and Physical Devices Synchronized with Music Videos on the Web. In *2015 IEEE International Symposium on Multimedia (ISM)*. 85–88. https://doi.org/10.1109/ISM.2015.64

[14] Henry Jenkins. 2006. *Convergence Culture: Where Old and New Media Collide*. New York University Press.

[15] Henry Jenkins. 2009. *Confronting the challenges of participatory culture: Media education for the 21st century*. The MIT Press.

[16] Jun Kato, Tomoyasu Nakano, and Masataka Goto. 2015. TextAlive: Integrated Design Environment for Kinetic Typography. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3403–3412. https://doi.org/10.1145/2702123.2702140

[17] Jun Kato, Masa Ogata, Takahiro Inoue, and Masataka Goto. 2018. Songle Sync: A Large-Scale Web-Based Platform for Controlling Various Devices in Synchronization with Music. In *Proceedings of the 26th ACM International Conference on Multimedia* (Seoul, Republic of Korea) *(MM '18)*. Association for Computing Machinery, New York, NY, USA, 1697–1705. https://doi.org/10.1145/3240508.3240619

[18] Jun Kato and Keisuke Shimakage. 2020. Rethinking Programming "Environment": Technical and Social Environment Design toward Convivial Computing. In *Conference Companion of the 4th International Conference on Art, Science, and Engineering of Programming* (Porto, Portugal) *(<Programming> '20)*. Association for Computing Machinery, New York, NY, USA, 149–157. https://doi.org/10.1145/3397537.3397544

[19] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 280–290. https://doi.org/10.1145/2807442.2807446

[20] Clemens N. Klokmose, Christian Remy, Janus Bager Kristensen, Rolf Bagge, Michel Beaudouin-Lafon, and Wendy Mackay. 2019. Videostrates: Collaborative, Distributed and Programmable Video Manipulation. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 233–247. https://doi.org/10.1145/3332165.3347912

[21] Taku Kudo. 2013. MeCab: Yet Another Part-of-Speech and Morphological Analyzer. https://taku910.github.io/mecab/. Accessed on November 11th, 2022.

[22] Yuli Levtov. 2018. Algorithmic Music for Mass Consumption and Universal Production. In *The Oxford Handbook of Algorithmic Music*, Alex McLean and Roger T. Dean (Eds.). Oxford University Press, Chapter 34, 627–644. https://doi.org/10.1093/oxfordhb/9780190226992.013.15

[23] Jen-Chun Lin, Wen-Li Wei, Yen-Yu Lin, Tyng-Luh Liu, and Hong-Yuan Mark Liao. 2020. Learning From Music to Visual Storytelling of Shots: A Deep Interactive Learning Mechanism. In *Proceedings of the 28th ACM International Conference on Multimedia* (Seattle, WA, USA) *(MM '20)*. Association for Computing Machinery, New York, NY, USA, 102–110. https://doi.org/10.1145/3394171.3413985

[24] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. *CoRR* cs.CL/0205028 (2002). https://arxiv.org/abs/cs/0205028

[25] LyricFind Inc. 2022. LyricFind. https://www.lyricfind.com/. Accessed on September 13th, 2022.

[26] LyricsTraining.com. 2011. Learn Languages with Music Videos, Lyrics and Karaoke! https://lyricstraining.com/. Accessed on September 13th, 2022.

[27] Alex McLean and Roger T. Dean. 2018. *The Oxford Handbook of Algorithmic Music*. Oxford University Press. https://doi.org/10.1093/oxfordhb/9780190226992.001.0001

[28] Musixmatch s.p.a. 2022. Musixmatch Developer API. https://developer.musixmatch.com/. Accessed on September 13th, 2022.

[29] Opal Limited. 2016. Brian Eno : Reflection. https://apps.apple.com/us/app/brian-eno-reflection/id1180524479. Accessed on September 13th, 2022.

[30] Justin Paterson and Rob Toulson. 2015. Interactive digital music: enhancing listener engagement with commercial music. In *Innovation in Music II*, Justin Paterson, Rob toulson, Jay Hodgson, and Russ Hepworth-Sawyer (Eds.). KES Transactions on Innovation in Music, Vol. 2. Future Technology Press, Shoreham-by-Sea, UK, 193–209. https://repository.uwl.ac.uk/id/eprint/2526/

[31] Amy Pavel, Colorado Reed, Björn Hartmann, and Maneesh Agrawala. 2014. Video Digests: A Browsable, Skimmable Format for Informational Lecture Videos. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) *(UIST '14)*. Association for Computing Machinery, New York, NY, USA, 573–582. https://doi.org/10.1145/2642918.2647400

[32] Matt Pearson. 2011. *Generative Art: A Practical Guide Using Processing*. Manning Publications Co., Shelter Island, NY, USA.

[33] AIST Songle Project. 2023. Songle. https://songle.jp. Accessed on February 6th, 2023.

[34] AIST TextAlive Project. 2023. TextAlive – Create, Watch, and Write Code for Lyric Videos Online. https://textalive.jp. Accessed on February 6th, 2023.

[35] Roman Rädle, Midas Nouwens, Kristian Antonsen, James R. Eagan, and Clemens N. Klokmose. 2017. Codestrates: Literate Computing with Webstrates. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 715–725. https://doi.org/10.1145/3126594.3126642

[36] Reality Jockey Ltd. 2012. RjDj. https://web.archive.org/web/20120730133514/http://rjdj.me/. Accessed on September 13th, 2022 (Archived on July 30th, 2012).

[37] Casey Reas and Ben Fry. 2014. *Processing: A Programming Handbook for Visual Designers and Artists, Second Edition*. The MIT Press.

[38] Christian Remy, Lindsay MacDonald Vermeulen, Jonas Frich, Michael Mose Biskjaer, and Peter Dalsgaard. 2020. Evaluating Creativity Support Tools in HCI

Research. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference (DIS '20)*. Association for Computing Machinery, New York, NY, USA, 457–476. https://doi.org/10.1145/3357236.3395474

[39] SEGA Games Co., Ltd. 2012. MikuFlick Official Site | Hatsune Miku comes to the iPhone! https://miku.sega.jp/flick/en. Accessed on January 23rd, 2023.

[40] David A. Shamma, Bryan Pardo, and Kristian J. Hammond. 2005. MusicStory: A Personalized Music Video Creator. In *Proceedings of the 13th Annual ACM International Conference on Multimedia* (Hilton, Singapore) *(MM '05)*. Association for Computing Machinery, New York, NY, USA, 563–566. https://doi.org/10.1145/1101149.1101278

[41] Valerie N. Stratton and Annette H. Zalanowski. 1994. Affective Impact of Music Vs. Lyrics. *Empirical Studies of the Arts* 12, 2 (1994), 173–184. https://doi.org/10.2190/35T0-U4DT-N09Q-LQHW

[42] TuneWiki, Inc. 2011. Lyric Legend. https://web.archive.org/web/20110208002254/http://www.lyriclegend.com/. Accessed on September 13th, 2022 (Archived on February 8th, 2011).

[43] Haijun Xia, Jennifer Jacobs, and Maneesh Agrawala. 2020. Crosscast: Adding Visuals to Audio Travel Podcasts. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 735–746. https://doi.org/10.1145/3379337.3415882

[44] Theodore Watson Zach Lieberman and Arturo Castro. 2022. openFrameworks. https://openframeworks.cc/. Accessed on September 13th, 2022.

A EVENT-DRIVEN, TIME-DRIVEN, AND TIME-RANGE-DRIVEN APIS

To supplement Section 4.2.2 and Section 7.1.2, we show Figure 6 and provide a detailed comparison between the ⓐ event-driven (typical implementation), ⓑ time-driven (our proposal), and ⓒ time-range-driven (novel finding in our study) APIs in their ways of utilizing time-coded information for time-sensitive interactions.

A.1 Event-driven API

An event-driven API fires an *event* and notifies its listeners as soon as something happens. This *"push"* API design suits a simple use case in which animations are triggered by specific timings in music playback. For instance, programmers can define a musical beat listener that gets executed every time music playback reaches a musical beat, and that listener can trigger a ripple animation.

However, as discussed in Section 4.2.2, this design has difficulty in future planning, accounting for multiple types of musical elements, and achieving high timing accuracy. Furthermore, the time-coded information cannot always be represented well as events with specific timings. This representation-level inconsistency would force programmers to write verbose listeners. For instance, some analysis types in Table 1 represent specific time ranges, such as lyric timings and chords, and therefore need to split into two events per each (e.g., `onWordEnter` and `onWordLeave`, `onChordEnter` and `onChordLeave`). Other types change values along time, such as vocal amplitude, and therefore might need to trigger events frequently (e.g., `onVocalAmplitudeUpdate` firing every 10-ms).

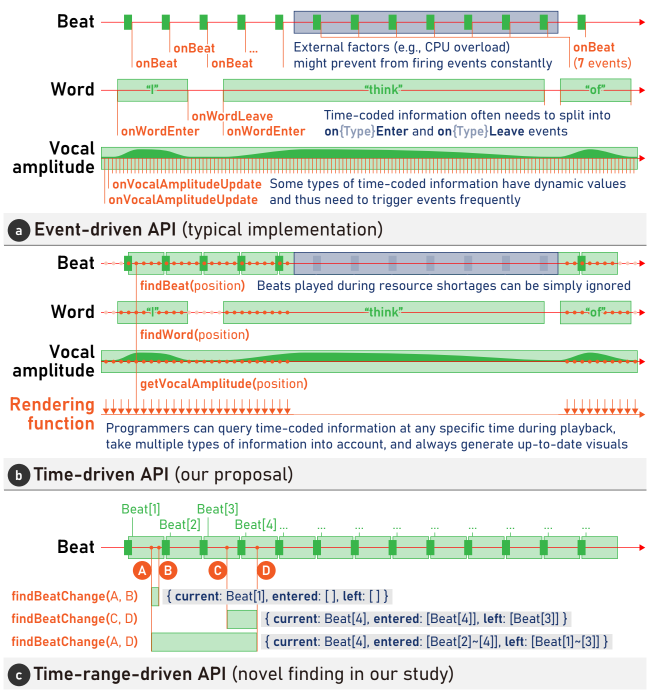
Beat
onBeat
onBeat
onBeat
...
External factors (e.g., CPU overload) might prevent from firing events constantly
onBeat (7 events)
Word
"I"
"think"
"of"
onWordLeave
onWordEnter
onWordEnter
Time-coded information often needs to split into on{Type}Enter and on{Type}Leave events
Vocal amplitude
onVocalAmplitudeUpdate
onVocalAmplitudeUpdate
Some types of time-coded information have dynamic values and thus need to trigger events frequently
a Event-driven API (typical implementation)
Beat
findBeat(position)
Beats played during resource shortages can be simply ignored
Word
"I"
"think"
"of"
findWord(position)
Vocal amplitude
getVocalAmplitude(position)
Rendering function
Programmers can query time-coded information at any specific time during playback, take multiple types of information into account, and always generate up-to-date visuals
b Time-driven API (our proposal)
Beat[1]
Beat[2]
Beat[3]
Beat[4] ...
Beat
A
B
C
D
findBeatChange(A, B)
{ current: Beat[1], entered: [ ], left: [ ] }
findBeatChange(C, D)
{ current: Beat[4], entered: [Beat[4]], left: [Beat[3]] }
findBeatChange(A, D)
{ current: Beat[4], entered: [Beat[2]~[4]], left: [Beat[1]~[3]] }
c Time-range-driven API (novel finding in our study)

Figure 6: Three different API designs to utilize time-coded information in time-sensitive interactions.

A.2 Time-driven API

In Section 4.2.2, we proposed a *time-driven API* that enables querying relevant time-coded information by passing a specific time argument. The framework stores the information so that it can return the query results quickly (e.g., binary tree sorted by timestamps in our implementation), and programmers write programs to *"pull"* the information at their discretion.

As opposed to the event-driven API that represents every time-coded information as discrete *events* with specific timings, this time-driven API design represents each type of information as a *segment* with a specific time range. This representation suites the nature of the time-coded information, addressing the representation-level inconsistency present with the event-driven API. For instance, a lyric word is represented as a segment with a start time (when the word starts being vocalized) and an end time (when it ends being vocalized). A musical beat is also represented as a segment with a start time (when the beat is hit) and an end time (when the next beat is hit). Other types with continuously changing values, such as vocal amplitude, can be considered as a segment that spans the entire playback of the music piece.

A.3 Time-range-driven API

Given the user feedback and source code analysis, we discussed a novel *time-range-driven API* in Section 7.1.2. While the *time-driven API* looks for a segment that overlaps with the given playback position, the *time-range-driven API* looks for all the segments that overlap with the given time range. In a typical case, programmers save the current playback position at the end of the rendering function, and in the next function call, they use this API to look for what have happened since the last call.

The result of the time-range-driven API is a tuple of `current`, `entered`, and `left`. Programmers can generate graphical objects corresponding to the `entered` segments, pass the `current` segment information to the graphical objects to render graphics, and destroy the graphical objects of the `left` segments. Note that the `current` property is either `null` or a single segment while the `entered` and `left` properties are variable-length arrays. For instance, in an extreme case, passing `(0, duration)` to the API returns a tuple like `{ current: null, entered: [all segments], left: [all segments] }`.

B DETAILS OF EXAMPLE APPLICATIONS

To supplement Section 6.1.2, Table 2 summarizes the 11 example applications that we developed with the framework. Three of the examples are illustrated in Figure 7.

Every example has a public demo website on GitHub Pages, through which programmers can instantly check the behavior. Some examples (e.g., interactive lyric sheets) are duplicated on CodePen, the web-based integrated development environment, to enable programmers to immediately edit the source code via a web browser.

C ALL PROGRAMMING CONTEST RESULTS

To supplement Section 6.2, Table 3 summarizes the 52 lyric app examples collected from the programming contests. For each example, the lyric app category, a short summary, any notable creative coding libraries used, and a link to the demonstration video on YouTube (if available) are reported.

To supplement Figure 5, more example screenshots are shown in Figure 8. The lyric apps in these figures can be seen in action on YouTube, via the links given in Table 3.

Table 2: Summary of the 11 example applications that we developed.

ID	Title and short summary	Dependencies
A	*Interactive lyric sheets.* (Section 6.1.2, Figure 4 Ⓐ, and GitHub:TextAliveJp/textalive-app-lyric-sheet)	No build tool; DOM operations and CSS transitions
B	*Lyric tiles.* (Section 6.1.2, Figure 4 Ⓑ, and GitHub:TextAliveJp/textalive-app-lyric-tiles)	Parcel (build tool) and HTML5 Canvas API
C	*Lyric app dance.* (Section 6.1.2, Figure 4 Ⓒ, and GitHub:TextAliveJp/textalive-app-dance)	Parcel and Three.js
D	*Lyric char cube.* (Figure 7 Ⓓ; GitHub:TextAliveJp/textalive-app-char-cube) A simple wireframe cube appears at the center and rotates, with the six faces showing the same animated character from the lyrics text being vocalized. This was intended to be the simplest demonstration of the framework’s use with a 3D CG creative coding library.	Parcel and Three.js
E	*Lyric mosaic.* (Figure 7 Ⓔ; GitHub:TextAliveJp/textalive-app-mosaic) The lyric character being vocalized is rendered in a mosaic-like pattern using the letters in the lyrics text. First, each character is drawn on a dedicated, square frame buffer, and the number of filled pixels is calculated. By using the calculated results, the characters are sorted in order of pixel fill density, and then the screen is divided into a grid of the specified size. Each grid cell is periodically updated to show a random character that can reproduce the density in the corresponding pixel of the vocalized character’s frame buffer.	Parcel and HTML5 Canvas API
F	*p5.js example.* (Figure 7 Ⓕ; GitHub:TextAliveJp/textalive-app-p5js) A canvas controlled by p5.js is populated at the center of the screen, and a simple animation of lyrics text is shown on the canvas. This was intended to be the simplest demonstration of the use of a popular creative coding library.	Parcel and p5.js
G	*Basic example.* (GitHub:TextAliveJp/textalive-app-basic) A huge DOM element is placed at the center of the screen and shows a lyric character being vocalized. Basic playback control and several other interesting controls are available, such as a button to jump to the position when the first lyric character starts being vocalized and a button to jump to the beginning of the chorus. This was intended to be the simplest demonstration for novices.	Parcel and DOM operations
H	*Lottie example.* (GitHub:TextAliveJp/textalive-app-lottie) This example was built on the previous basic example by adding an attractive animation in the background of the lyric character, which is drawn by loading a Lottie file. Lottie is an animation format that is compatible with Adobe After Effects, and Lottie files can be created by motion graphic designers. This was intended to be an example of the use of conventional motion graphic tools in lyric app development.	Parcel and Lottie
I	*Parameter customization example.* (GitHub:TextAliveJp/textalive-app-params) This example was built on the previous basic example by adding the capability for customization with Lyric App Customizer (Section 4.3.2).	Parcel and DOM operations with React
J	*Phrase and beat example.* (GitHub:TextAliveJp/textalive-app-phrase.) This example was built on the previous basic example by adding visual effects synchronized with musical beats.	Parcel and DOM operations
K	*Phrase and beat (script tag) example.* (GitHub:TextAliveJp/textalive-app-script-tag.) This example was the same as the previous one but was written in plain HTML/JavaScript/CSS so that no build tool was needed to edit and publish the lyric app.	No build tool; DOM operations

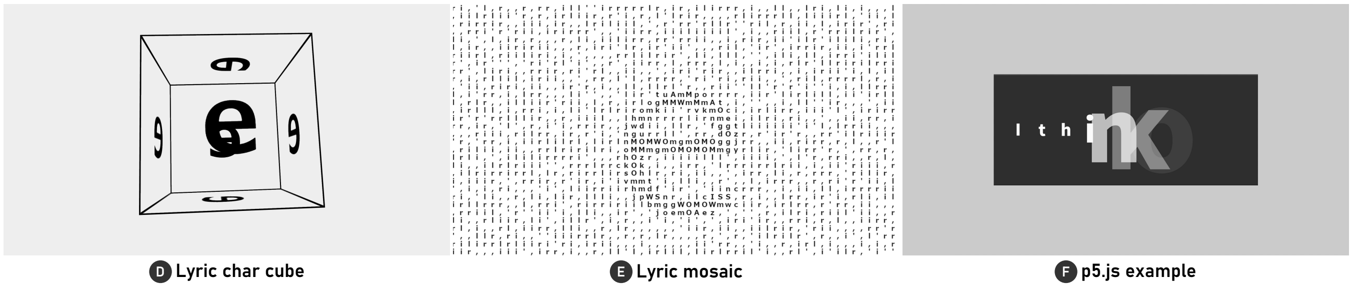

Figure 7: Screenshots of three example applications listed in Table 2.

Table 3: Summary of the 52 example applications collected from the programming contests (32 from 2020, 20 from 2021).

ID	Category	Short summary (notable creative coding libraries used and video link, if any)
2020-1	Generative lyric video	Generative kinetic typography and character performance with no interactivity (PixiJS, Three.js, three-vrm, ammo.js).
2020-2	Generative lyric video	Very simple kinetic typography video with glowing visual effects.
2020-3	Instrument	Connection to an external MIDI device via the Web MIDI API and synthesis of chords and drum sounds along with music playback.
2020-4	Creative application	Interactive fireworks animation generator synchronized with music playback, with various customization parameters (Three.js, anime.js; Figure 8 ⑬, https://youtu.be/KQc3FCelKNo).
2020-5	Generative lyric video	Very simple kinetic typography video using chord information (p5.js).
2020-6	Augmented music video	Theater-like space with the music video embedded at the center to provide a virtual party experience (React; Figure 8 ⑰, https://youtu.be/-t9AVVgZo5k).
2020-7	Interactive lyric video	Lyrics rendered in an immersive 3D space, with motions programmed to respond to the music structure and touch interaction (PlayCanvas; https://youtu.be/mfCFLvb9IS8).
2020-8	Extended reality	Augmented reality application to overlay lyrics on camera images (Three.js, AR.js).
2020-9	Extended reality	Virtual roller coaster whose course and stable camera control are dynamically generated (Three.js; Figure 8 ⑨, https://youtu.be/sYyGA_4YbwM).
2020-10	Generative lyric video	Colorful kinetic typography video in which lyrics gradually appear and disappear in synchrony with the music playback (Three.js).
2020-11	Authoring tool	A virtual singer on a stage sings a given musical piece, and various options (e.g., appearance, motion patterns) can be customized with a dedicated GUI (Three.js, three-vrm; Figure 8 ⑪, https://youtu.be/LiHmw7m5bCs).
2020-12	Generative lyric video	Simple karaoke-style kinetic typography video (jQuery).
2020-13	Generative lyric video	Various animations appear with lyrics text; the animation content is in the Lottie format, indicating that it was created with an external authoring tool (Lottie).
2020-14	Generative lyric video	Interactive lyric sheets with vibrant animation effects on an animated background where lyrics text flows from left to right (dat.gui; Figure 8 ⑱, https://youtu.be/W-Sb5FbnvZI).
2020-15	Authoring tool	Fully customizable interactive graphics; a piano keyboard allows the user to touch and trigger animating effects; default values for certain parameters are calculated from a musical piece's information (jQuery, Lottie; https://youtu.be/ON3lUgub8ws).
2020-16	Game	Shooting game to hit characters in the lyrics text (jQuery, Bootstrap; Figure 8 ⑮, https://youtu.be/10qkhB_NYEw).
2020-17	Interactive lyric video	Lyrics rendered in a 3D environment; various kinetic typography techniques; fully responsive design optimized for smartphones (Three.js, anime.js; Figure 8 ⑲, https://youtu.be/cs9qxULFARg).
2020-18	Generative lyric video	Simple kinetic typography video with the style, visual effects, and background graphics reflecting the stories and settings of a specific musical piece (jQuery).
2020-19	Authoring tool	Interactive kinetic typography generator with various customization parameters (Vue.js (Nuxt.js), jQuery, p5.js).
2020-20	Extended reality	Virtual roller coaster whose course is dynamically generated from the structure of a musical piece (React (Create React App), Babylon.js).
2020-21	Game	Musical rhythm game to control the horizontal movement of a player-controlled bar and collect characters dropping from the top of the screen (PixiJS).
2020-22	Generative lyric video	Various animations appear with lyrics text and chord information; animation content in the Lottie format (Lottie).
2020-23	Extended reality	Kinetic typography in an immersive 3D space where the camera moves forward and lyrics text is lit with glowing effects (glslify).
2020-24	Generative lyric video	Simple kinetic typography video; visual effects become vibrant during the chorus.
2020-25	Extended reality	Simple kinetic typography in an immersive 3D space; implemented with the WebVR API, allowing users with supported head-mounted displays to dive into the space (AFrame.js, Three.js).
2020-26	Creative application	Simple paint tool combined with a simple kinetic typography video.
2020-27	Generative lyric video	Simple kinetic typography in a deep sea; random bubbles occasionally float up from the bottom.
2020-28	Interactive lyric video	Lyrics move from right to left in an abstract 3D scene; grabbing lyrics triggers the seek operation, enabling the user to scroll the lyrics and easily navigate to a target position in a musical piece (Three.js; Figure 5 ⑦, https://youtu.be/_yAlLiIGByI).

(Continued from the previous page.)

ID	Category	Short summary (notable creative coding libraries used and video link, if any)
2020-29	Authoring tool	Authoring environment for projection mapping of kinetic typography videos onto virtual scenes with dedicated styles for a specific set of musical pieces; when the user prepares physical scenes with the same geometry as the virtual scenes, the tool can be used for actual projection (Svelte; Figure 5 ②, https://youtu.be/z_M1uBCV0tc).
2020-30	Generative lyric video	Very simple kinetic typography video with hard-coded visual effects (p5.js).
2020-31	Extended reality	Use of Tensorflow.js to detect human motion from a camera input in real time and overlay customizable generative visuals and kinetic typography of lyrics in front (Tensorflow.js).
2020-32	Interactive lyric video	Use of an NLP technique to automatically generate parody lyrics from the original lyrics (kuromoji.js).
2021-1	Extended reality	Immersive 3D environment in which many perks are hidden and various animations are played in synchrony with the music playback (PlayCanvas; Figure 5 ①, https://youtu.be/NqszHM1g9xE).
2021-2	Generative lyric video	Various kinetic typography techniques implemented in a 3D scene, with a binary toggle-switch user interface to customize the background animation (anime.js, glMatrix, glslify; Figure 8 ⑳, https://youtu.be/hvmDxCeyWU8).
2021-3	Authoring tool	Authoring environment for generative lyric videos; placeholders for lyrics text, anime-style human characters, and several other animated illustrations can be placed anywhere on the screen (Figure 8 ⑫, https://youtu.be/2dopnxQrWGc).
2021-4	Authoring tool	Authoring environment for simple generative lyric videos; only the kinetic typography part is customizable, and the parameters are very simple (React, tsParticles, Semantic UI React).
2021-5	Generative lyric video	Colored lyric characters gradually populate and jump into the canvas; the synthesized picture looks like it was drawn with a crayon, through algorithmic conversion from a specified raw illustration (PixiJS, Jimp; Figure 5 ⑧, https://youtu.be/bvRqBNwZBEM).
2021-6	Game	Simple game showing bubbles floating downward, each of which contains a single lyric character and is clickable to add to the user’s score (p5.js).
2021-7	Creative application	Customizable pen tool on a blackboard-like user interface showing lyrics text, allowing the user to share screenshots of lyrics and user-drawn illustrations (Three.js, opentype.js, Troika 3D Text for Three.js, Tweakpane; Figure 5 ③, https://youtu.be/XzYZr_urn3I).
2021-8	Instrument	Generative kinetic typography and animated human character performance with minor interactivity to click a button and generate sound effects (jQuery, Canvas Confetti).
2021-9	Creative application	The user is encouraged to touch the screen and put musical note icons in any location; the icons start dancing when the playback reaches the chorus; simple kinetic typography is displayed next to the notes (p5.js, Lottie; https://youtu.be/Cf5zhup_Bqo).
2021-10	Generative lyric video	Generative kinetic typography and animated character performance with no interactivity.
2021-11	Creative application	Improvisational experience to control horizontal character motion in a 3D scene, with grass gradually growing around the character, allowing drawing of simple illustrations (Three.js, Tween.js, XState; Figure 8 ⑭, https://youtu.be/hvmDxCeyWU8).
2021-12	Augmented music video	Reproduction of the same scene as the original music video, showing lyrics text and adding various interaction effects to visible components that can be triggered by touch or keyboard events (Next.js (React), clsx; Figure 5 ⑥, https://youtu.be/5BlFr6Ma70w).
2021-13	Generative lyric video	Very simple kinetic typography video with lyrics scrolling from right to left.
2021-14	Extended reality	Immersive 3D environment where an animated human character behaves in a lively way in response to musical elements; colors and animation patterns can be customized (Three.js; Figure 8 ⑩, https://youtu.be/vnginTMqg0Y).
2021-15	Creative application	Similar to 2021-11, but with a grid of squares on the ground, making the creation more like pixel art (Three.js).
2021-16	Game	Musical rhythm game highly optimized for touch operation; lyrics are drawn on dynamically generated paths, and the user traces them to achieve a high score (Next.js (React), PixiJS, GSAP; Figure 8 ⑯, https://youtu.be/VaCmJWSiNBg).
2021-17	Game	Musical rhythm game in which lyric characters are displayed in front of a music video; the arc around each character gets shorter, and the user touches it at the end of the arc animation (Create.js).

(Continued from the previous page.)

ID	Category	Short summary (notable creative coding libraries used and video link, if any)
2021-18	Instrument	Use of MediaPipe.js to track the user's hand motions and generate virtual glow sticks that the user can manipulate; the user can freely operate the sticks and cheer virtual human characters dancing to a musical piece (MediaPipe.js, React, Three.js, Konva, Tweakpane; Figure 5 ④, https://youtu.be/hvmDxCeyWU8).
2021-19	Game	Musical rhythm game in which the player moves a dancing character on a stage at the right side of the screen and aims to enliven the character's performance by catching lyrics text that flows from left to right at random heights (Phaser; Figure 5 ⑤, https://youtu.be/r26c6Og7r9k).
2021-20	Extended reality	Three completely different visual performances synchronized with musical pieces: a live stage, a computer console screen, and a world map (p5.js, ztext.js, Hammer.js; https://youtu.be/prDvCzQ1HfI).

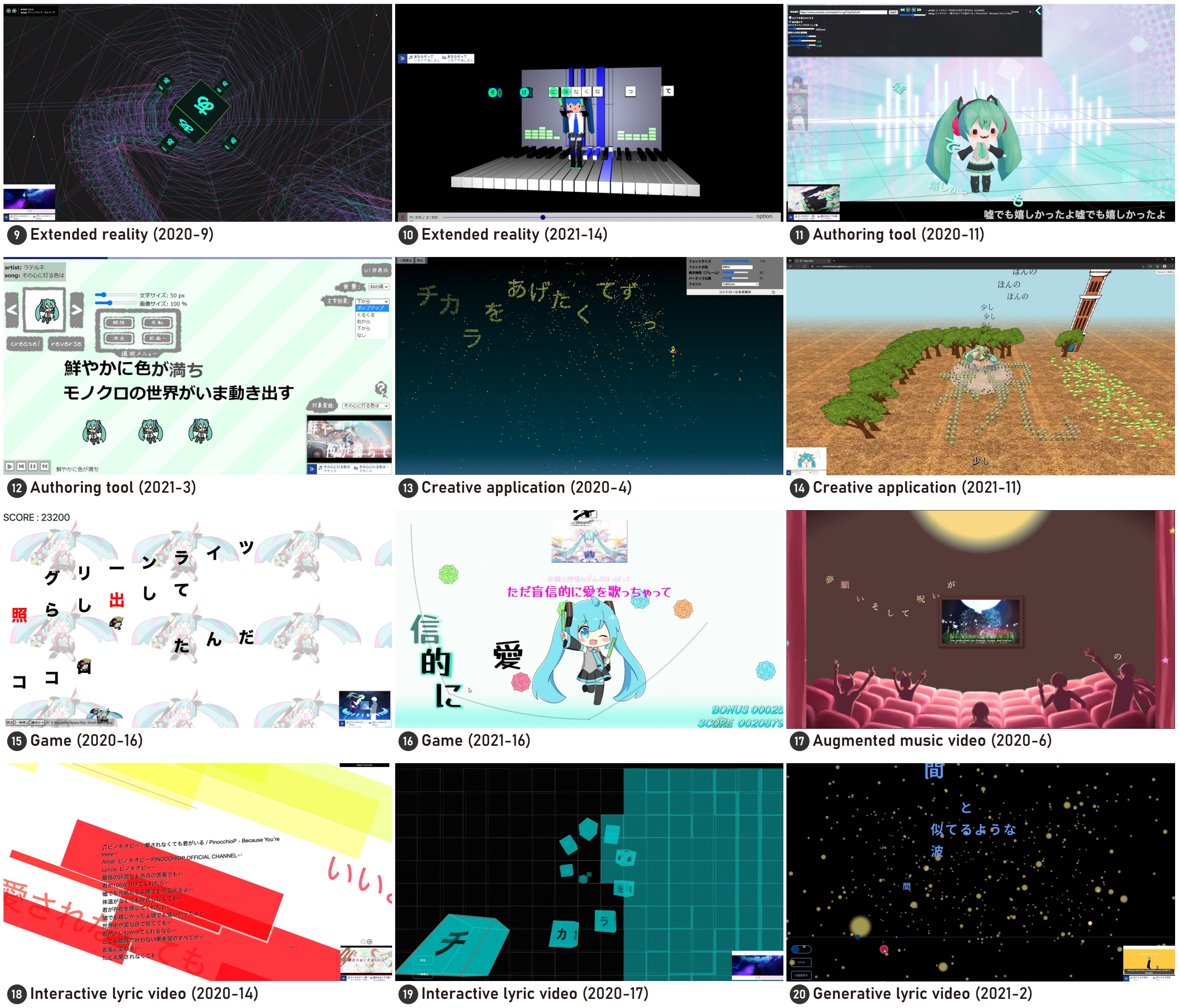
⑨ Extended reality (2020-9)
⑩ Extended reality (2021-14)
⑪ Authoring tool (2020-11)
⑫ Authoring tool (2021-3)
⑬ Creative application (2020-4)
⑭ Creative application (2021-11)
⑮ Game (2020-16)
⑯ Game (2021-16)
⑰ Augmented music video (2020-6)
⑱ Interactive lyric video (2020-14)
⑲ Interactive lyric video (2020-17)
⑳ Generative lyric video (2021-2)

Figure 8: Screenshots of 12 submitted applications to illustrate the diversity of visual styles found in the programming contests; screenshots courtesy of Crypton Future Media, Inc.