TextAlive Online: Live Programming of Kinetic Typography Videos with Online Music

Jun Kato, Tomoyasu Nakano, Masataka Goto
National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan
{jun.kato, t.nakano, m.goto}@aist.go.jp

ABSTRACT

This paper introduces a web-based integrated design environment named "TextAlive Online" that supports creating Kinetic Typography videos synchronized with songs available online. It is the hybrid of a content authoring tool and a live programming environment. Through its development, we investigate the interaction design that most benefits from the interactive user interfaces used by designers and programmers, as well as the collaborative nature of the open-source culture on the web. This system is accessible at textalive.jp and the interactive "live" version of this paper is available at textalive.jp/paper.

1. Introduction

The Internet has allowed many people to express their creativity in the connected world. The open-source culture has altered the way of content authoring. For instance, the Creative Commons license made it easy for creators to author derivative content from existing one. Many songs and their derivative content are uploaded onto websites like SoundCloud and YouTube. However, they are designed as a platform for distributing content, not for authoring it.

Programmers live in a far more advanced world in terms of content authoring. They can easily collaborate with others by forking repositories, sending pull requests, pushing back changes, etc. Recent Web-based Integrated Development Environments (WIDEs) integrate such online activities into the IDEs, supporting the users' collaborations. Live Programming is another important technique that has recently attracted attention and been implemented in various IDEs, helping the programmers' content authoring. It provides a fluid programming experience by continuously informing the user of the state of the programs being developed.

We foresee that this programmers' way of content authoring is powerful yet general enough to be applied to various types of content authoring. At the same time, the typical workflow of today's programmer is not only focused on designing algorithms but also involves more data organization and manipulation. There is prior research on development environments with support for such workflow. Representative examples include Gestalt for machine learning (Patel et al. 2010), DejaVu for interactive camera applications (Kato, McDirmid, and Cao 2012), VisionSketch for image processing applications (Kato and Igarashi 2014), Unity for game applications, and web-based live coding environments for music and video improvisation such as Gibber (Roberts et al. 2014) and LiveCodeLab (Della Casa and John 2014). Content authoring and programming environments are colliding with each other. Therefore, it is important to design a content authoring environment that supports both data manipulation and live programming.

This paper introduces TextAlive Online, which allows a user to create Kinetic Typography videos interactively with live programming on a standard web browser. Kinetic Typography is a technique for animating text, which is known to be a more effective way to convey emotional messages compared to static typography. Our system supports the creation of Kinetic Typography videos that show the lyrics of a song synchronized with its audio content accessible on the web. While many live coding environments focus on improvisation of music, our system focuses on providing a platform on which the user can elaborate on creating videos.

Thanks to its web-based nature, the system can retrieve the specified audio content, analyze and estimate vocalized timing of all characters in the lyric text. Then, the estimation results can be cached and corrected collaboratively by several different people. A JavaScript code editor is embedded in the system along with the general graphical user interfaces (GUIs) for video authoring. JavaScript programmers can create, edit, and share algorithms for text animation, which are called "templates," with others. Please note that many designers are now casual JavaScript programmers as well.

In this paper, the core concept behind TextAlive Online is explained. Before we developed TextAlive Online, we had prototyped a Java-based desktop application named TextAlive Desktop that follows the concept and runs on a local machine. The next section introduces the interaction design shared between TextAlive Desktop and Online. Then, the unique feature of TextAlive Desktop with the results of the preliminary user study are briefly explained. This experience led us to believe in the potential impact of making the system web-based. Although rewriting a standalone desktop application to run on a web browser supported by a server-side backend required tremendous engineering effort, TextAlive Online opens up new content authoring possibilities for the web on the web. The following section explains its unique user experience. Then, related work is introduced, followed by a discussion and our conclusion.

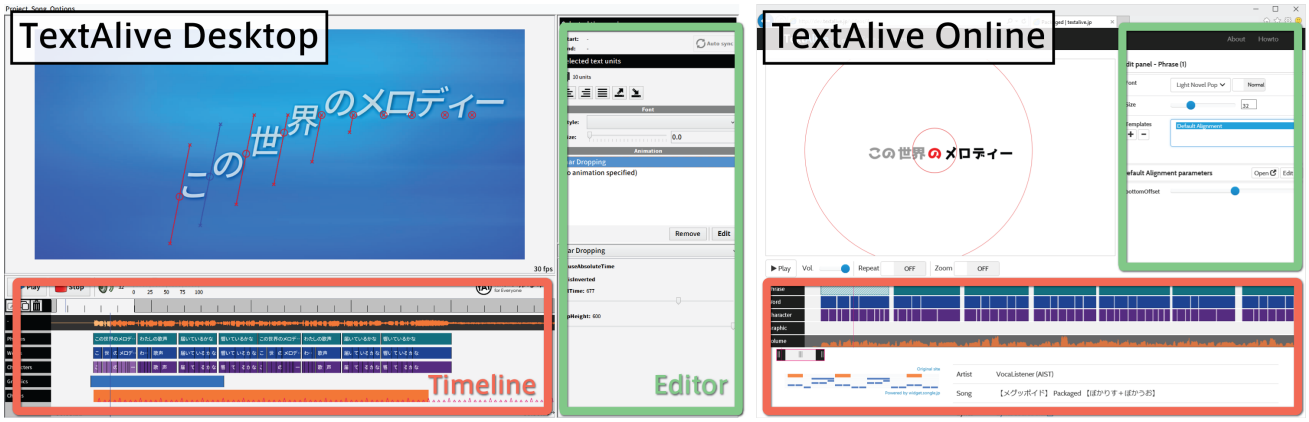

Figure 1: *TextAlive Desktop and TextAlive Online are both live programming environments for creating Kinetic Typography videos that share most of their GUIs. TextAlive Online enables a more collaborative way of content authoring, requires a server-side backend, and raises interesting research questions.*

2. Videos as Pure Functions of Time

Digital videos are usually created with content authoring tools such as Adobe After Effects, Adobe Premier and Windows MovieMaker. A content authoring tool saves information of a video as a project file that has references to external files including images, videos, and audio. When a video project is exported as a rendered video file, the tool generates tens of images per second with a particular resolution by calling rendering routines for every frame.

While ordinary video files have particular resolution in time and space, there are videos that potentially have an infinite resolution in time and space; object positions can be acquired by calculating interpolation between key frames and object shapes can be specified as vector graphics. In other words, a certain kind of videos can be represented by a pure function without any side effects that takes the current time as its argument and returns the rendering result. The demoscene and megademo are prior attempts at creating fascinating videos from code, but their code usually has heavy side effects over time and cannot jump to arbitrary time. Various live coding environments allow the programmer to define a function of time that outputs a visual scene, but most of them focus on improvisation. As a result, the scene is often ephemeral and transformed along time through live coding.

In TextAlive, a kinetic typography video is represented by a collection of pure functions of time with several parameters that can be customized through interactive GUIs. Since lyric text contains repetitions and a video contains multiple parts with similar effects, it is desired that a function for a text unit can be reused for another text unit. Therefore, instead of defining one huge function that renders everything on the video, we introduce a structure of functions – one function per each text unit (a phrase, word, or character) of lyric text. The functions are represented by instances of classes, which we call "templates," with particular parameters. As a concrete example of how these information is stored, please see textalive.jp/videos/20?format=json, which contains links to external resources, including songs, lyrics, and versioned template definitions and their parameters. For each timing, the video image is rendered by calling a method for all of the template instances. Most of the instance methods immediately return without doing anything since they are assigned to text units that are hidden at the specified timing. The rest of the methods modify the rendering parameters of text units to place and reshape them. An example of template definition can be found at textalive.jp/templates/KaraokeColor, which changes the color of assigned characters. If a method call takes too long time, the corresponding template is disabled and will be re-enabled when the programmer updates the template definition.

3. Interaction Design

The user first inputs a song and lyric text to the system. Then, the system estimates the vocalized timing (the starting and ending of the vocalization) of every character in the lyric text. With this information, a playable kinetic typography video is automatically composed and opened in the graphical video editor. Thus, this system provides the user with this concrete example to start working on at the beginning instead of the empty canvas provided by the general video authoring tools.

As shown in Figure 1, the main window of TextAlive consists of 1) a Stage interface that shows the editing video, 2) a Timeline interface specifically designed for the intuitive timing correction of the given text units (phrases, words, and characters), and 3) an Editor interface that allows for choosing and tweaking the templates that define the motion of the selected text units. Casual users and designers can create videos in their style using these GUIs. Though, choosing the text animation from the existing templates limits the flexibility of the resultant videos to a certain extent. TextAlive provides a text-based code editor for live programming of the template definitions to overcome such a limitation.

The text-based code editor is tightly coupled with the aforementioned graphical interfaces. The user can select text units using the Timeline interface, click on one of the assigned templates, edit its motion algorithm, and update the video without needing to pause the music playback. Moreover, part of the Editor interface is dynamically generated from the results of static code analysis of the source code. For instance, when the user annotates a member variable with the comment `@ui Slider(0, 100)`, a slider is populated. It can dynamically change the field values of the template instances assigned to the selected text units. Such GUI components make it easy for the programmer to develop and debug templates iteratively. Moreover, it also helps other people who use the template, including designers and casual users who cannot or do not want to read nor write code.

4. TextAlive Desktop

TextAlive Desktop is the first system that follows the core concept and implements the interaction design described in the previous section. As a preliminary user study, we provided TextAlive Desktop to seven participants with different backgrounds (designers, programmers, and casual users). Then, we asked them to create Kinetic Typography videos using their favorite songs. While the prior publication introduces the system and the study in detail (Kato, Nakano, and Goto 2015), this section summarizes the lessons learned that motivated us to develop the Online version.

4.1. Programming Needs Structured Data

The user needs to wait for the completion of the timing estimation for the song and lyrics pair before they can start creating a new video. If the pair has been previously analyzed, it is possible to reuse the cached data. When the analysis is complete, the estimated result typically contains errors that need to be corrected manually. Such timing correction is important for the Kinetic Typography videos that should be well synchronized with the songs.

The data resources used in the programming should always be clean, structured, and easy to manipulate from the programmer's perspective. The synchronization of a song and its lyrics is considered data manipulation that creates structured data from the raw materials, which is referenced by the template source code. In addition, creating high-quality videos in TextAlive requires an iterative cycle of the data manipulation, programming, and playback. While live programming techniques can reduce the programming burden, this kind of data manipulation might remain as the bottleneck in the iterative cycle. Therefore, it is considered important to make *integrated design environments* in which the user interfaces are tightly coupled for the data manipulation and live programming.

4.2. Designers Want to Design, Programmers Want to Code

TextAlive Desktop is the hybrid of an interactive graphical tool and a live programming environment. We assumed that many designers are now capable of writing code, and thus, can benefit from the live programming part of TextAlive. However, during the user study, we observed a clear tendency for the designer participants (and, of course, casual users without prior knowledge of programming) to hesitate opening up the code editor. They wanted more predefined and ready-for-use templates. In addition, combining multiple templates for a single text unit seem to be difficult for the users. It is because the results are often unexpected and there is no hint to look for visually pleasing combinations.

On the other hand, the programmers could write various algorithms for text animation but were not always interested in creating a complete video. While all the participants welcomed the capability of the integrated environment, collaborations between people with different backgrounds are found to be the key to making the most use of it.

5. TextAlive Online

Given the lessons learned from the deployment of TextAlive Desktop, we built TextAlive Online, a web-based application with which people can collaboratively author kinetic typography videos. The resulting system is available at textalive.jp. While the main interface consists of almost identical user interface components to the desktop version (Figure 2), there are online-specific features and improvements as explained in this section.

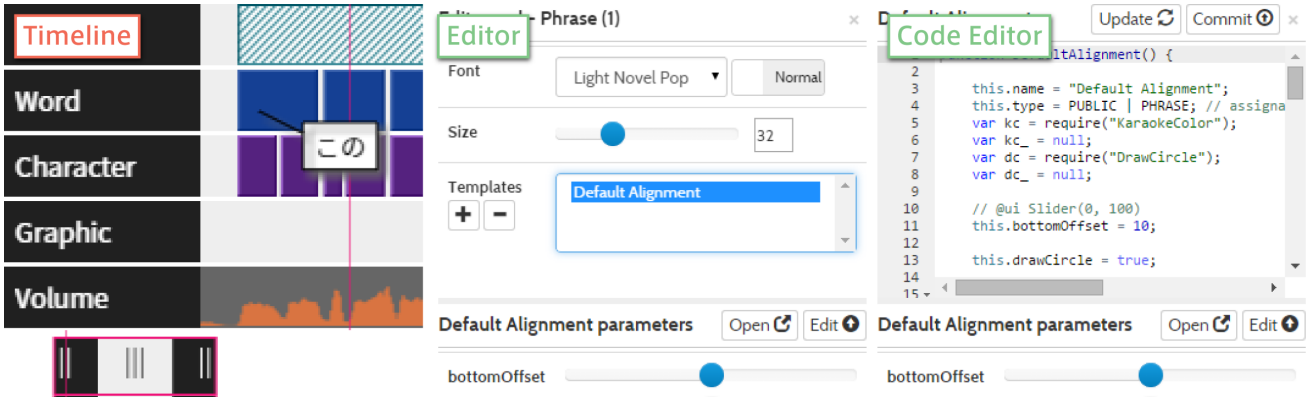

Figure 2: *TextAlive Online provides three user interfaces for editing Kinetic Typography videos.*

5.1. Everything is Shared and Version-Controlled

With TextAlive Online, the user just needs to search for and choose a song from one of the music streaming services at textalive.jp/songs (currently supporting SoundCloud, YouTube, a MP3 file on the web, and other several music streaming services) and provide its lyrics by pointing to a URL (currently supporting a TXT file and some websites with specific HTML formats) to start creating a Kinetic Typography video. If the song was not previously analyzed, the user is told to wait for several minutes. Otherwise, the user can instantly see an automatically-generated Kinetic Typography video.

Timing information about the vocalization of phrases, words, and characters can be corrected at any time, even during video playback. Then, the user can commit the timing information to share it with other users. In this way, TextAlive Online supports creating structured data collaboratively that is used for programming text animation.

A Kinetic Typography video on TextAlive Online consists of the following information: 1) URLs to the song and lyrics, 2) the structure of the lyrics, which is a tree of the text units (phrases, words, and characters), and 3) a list of assigned templates and their parameter values for each text unit. These set of information is represented by a JSON object and is version-controlled on the website. In fact, not only the videos but also the estimated timing information, correction history, and template definitions are all version-controlled, enabling to fork derivative work and to rollback flawed edits. Saving a new template definition is as easy as clicking the "Commit" button beside the update button that locally updates the definition (Figure 2, Code Editor).

5.2. Some Templates Make Use of Other Templates

In the Desktop version, the user is allowed to assign multiple templates to a text unit for flexible design of kinetic typography. For instance, a template for fading in and out text and a template for moving text can be assigned to the same phrase to combine both effects. However, it seemed difficult for the user to find a good combination of multiple templates and the feature was not used frequently. To simplify the interface, TextAlive Online currently limits the number of templates that can be assigned to a text unit to one. In addition, to maintain the reusability of templates, it allows the programmer to import and use other templates in a template definition.

textalive.jp/templates/DefaultAlignment is one example that calls a method of another template. The `KaraokeColor` is dynamically loaded with the `require(templateName)` function (which is similar to the popular RequireJS library), instantiated, and its `highlight` method is called on for coloring each character.

6. Related Work

This section mainly covers the prior work in the literature for human-computer interaction and live coding.

6.1. Tools for Content Authoring

Various tools have already been proposed for creating Kinetic Typography videos, such as toolkits for programmers (J. C. Lee, Forlizzi, and Hudson 2002) and GUI applications for end-users (Forlizzi, Lee, and Hudson 2003). However, most of them aim at augmenting text-based communication, and none of them provide support for synchronizing text with audio. Therefore, they rarely have the timeline interface necessary to seek the time and edit timings precisely.

More general video authoring tools are equipped with a timeline interface such as Adobe AfterEffects. They typically contain scripting engines in which the user can write code (called "Expression" in AfterEffects) to control the property changes over time. Although, it is just one way of specifying the visual properties, and thus, do not output video as a clean, structured object. There are also tools for interactive VJing, which is creating videos in real time synchronized with music. For instance, Quartz Composer and vvvv have been used for such purposes. Other examples are introduced in the next subsection. While the current implementation of TextAlive focuses on authoring videos and does not support improvisation of visual content, it is an interesting future work.

Live coding environments for music typically focus on improvising music and the resulting performance can only be recorded as static audio or video. On the other hand, Threnoscope (Magnusson 2014) is a live programming environment to describe musical scores. It is similar to TextAlive in that both help the user to create structured data (score or video) which can be modified later, either by the original author or other people.

6.2. Live Programming of Immediate Mode GUIs

TextAlive renders each frame of the video by repeatedly calling on the `animate(now)` method of all template instances. The programmer's code is responsible for building the scene from scratch for every frame. This type of user interface rendering is called immediate mode GUIs (related discussion at Lambda the Ultimate), while the frameworks for retained GUIs such as Java Swing expose APIs that essentially manipulate a scene graph composed of GUI widgets with hidden state information inside.

Typical GUI applications adopt the retained GUIs since they are considered convenient, but they have certain downsides such as difficulty in tracing the state transitions. This is particularly problematic for live programming. As a result, many of the existing live programming environments only support immediate GUIs. Notable examples that run on a standard web browser include Gibber (Roberts et al. 2014), LiveCodeLab (Della Casa and John 2014) and TouchDevelop (Burckhardt et al. 2013).

6.3. Live Programming with Parameter Tuning Widgets

Editing a text-based source code is often cumbersome since the programmer needs to type the parameter values many times to find the best value in order to make the full use of live programming. Providing sliders for this purpose is a common practice in this field as seen in recent Processing extension.

Impromptu (Sorensen and Gardner 2010), Overtone (Aaron and Blackwell 2013), and Threnoscope (Magnusson 2014) are live coding environments for music which can bind parameter values not only to GUI sliders but also to sliders on a physical control board. Unity is a game engine that is capable of instantiating widgets without killing the entire application. VisionSketch (Kato and Igarashi 2014) takes a similar approach and allows the programmer to draw shapes on the input images to interactively change the parameters passed to the computer vision algorithms.

7. Conclusion

We propose TextAlive Online, an online content authoring environment with support for live programming. It is the hybrid of a content authoring tool and a live programming environment, each of which has been separately evolved and is good at interactive data manipulation and intuitive programming, respectively. It is the successor of TextAlive Desktop, which is a desktop application and whose user study revealed the need for a web-based backend that features cooperative data manipulation, live programming, and video authoring.

The current implementation allows for embedding a created video into a web page, as demonstrated in the top page. Kinetic Typography is applicable not only for songs and lyrics but also for more general text information. It might be interesting to allow external developers to import motion algorithms from TextAlive Online and use them for animating text on their web pages. In addition, our future work includes improvisation of kinetic typography videos that enables a "text jockey" who creates text animation on-the-fly similar to a disc jockey and visual jockey.

While there have been proposed many live coding environments that output a visual scene as well as audio, existing environments typically put emphasis on improvising the scene from scratch in front of the audiences. Such environments are optimized for real-time live performance that is ephemeral and can only be recorded as videos with a static resolution in time and space. On the other hand, TextAlive Online is a platform on which the user can create and play videos that are essentially a collection of pure functions of time and potentially have an infinite resolution in time and space. The videos are always rendered on-the-fly by calling the functions. Since their information is kept structured, anybody can create derivative work from them, either by GUIs or by writing code. We believe that there is much potential in live coding techniques not only for live performance but also for collaboratively elaborating on creating various kinds of media content.

8. Acknowledgements

This work was supported in part by JST, CREST. TextAlive Online uses Songle Widget API for audio playback, whose main developer is Takahiro Inoue. It also relies on Songle web service for retrieving vocalized timing of lyric text and other information regarding the audio track, whose main developer is Yuta Kawasaki.

References

Aaron, Samuel, and Alan F. Blackwell. 2013. "From Sonic Pi to Overtone: Creative Musical Experiences with Domain-Specific and Functional Languages." In *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling & Design*, 35–46. FARM '13. New York, NY, USA: ACM. doi:10.1145/2505341.2505346. http://doi.acm.org/10.1145/2505341.2505346.

Burckhardt, Sebastian, Manuel Fahndrich, Peli de Halleux, Sean McDirmid, Michal Moskal, Nikolai Tillmann, and Jun Kato. 2013. "It's Alive! Continuous Feedback in UI Programming." In *Proc. of PLDI 2013*, 95–104. PLDI '13. Seattle, Washington, USA. doi:10.1145/2491956.2462170. http://doi.acm.org/10.1145/2491956.2462170.

Della Casa, Davide, and Guy John. 2014. "LiveCodeLab 2.0 and Its Language LiveCodeLang." In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design*, 1–8. FARM '14. New York, NY, USA: ACM. doi:10.1145/2633638.2633650. http://doi.acm.org/10.1145/2633638.2633650.

Forlizzi, Jodi, Johnny Lee, and Scott Hudson. 2003. "The Kinedit System: Affective Messages Using Dynamic Texts." In *Proc. of CHI 2003*, 377–384. Ft. Lauderdale, Florida, USA. doi:10.1145/642611.642677. http://doi.acm.org/10.1145/642611.642677.

Kato, Jun, and Takeo Igarashi. 2014. "VisionSketch: Integrated Support for Example-Centric Programming of Image Processing Applications." In *Proceedings of the 2014 Graphics Interface Conference*, 115–122. GI '14. Toronto, Ont., Canada, Canada: Canadian Information Processing Society. http://dl.acm.org/citation.cfm?id=2619648.2619668.

Kato, Jun, Sean McDirmid, and Xiang Cao. 2012. "DejaVu: Integrated Support for Developing Interactive Camera-Based Programs." In *Proc. of UIST 2012*, 189–196. Cambridge, Massachusetts, USA. doi:10.1145/2380116.2380142. http://doi.acm.org/10.1145/2380116.2380142.

Kato, Jun, 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*, 3403–3412. CHI '15. New York, NY, USA: ACM. doi:10.1145/2702123.2702140. http://doi.acm.org/10.1145/2702123.2702140.

Lee, Johnny C., Jodi Forlizzi, and Scott E. Hudson. 2002. "The Kinetic Typography Engine: an Extensible System for Animating Expressive Text." In *Proc. of UIST 2002*, 81–90. Paris, France. doi:10.1145/571985.571997. http://doi.acm.org/10.1145/571985.571997.

Magnusson, Thor. 2014. "Improvising with the Threnoscope: Integrating Code, Hardware, GUI, Network, and Graphic Scores." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, edited by Baptiste Caramiaux, Koray Tahiroglu, Rebecca Fiebrink, and Atau Tanaka, 19–22. London, United Kingdom: Goldsmiths, University of London. http://www.nime.org/proceedings/2014/nime2014_276.pdf.

Patel, Kayur, Naomi Bancroft, Steven M. Drucker, James Fogarty, Andrew J. Ko, and James Landay. 2010. "Gestalt: Integrated Support for Implementation and Analysis in Machine Learning." In *Proc. of UIST 2010*, 37–46. doi:10.1145/1866029.1866038. http://doi.acm.org/10.1145/1866029.1866038.

Roberts, Charles, Matthew Wright, JoAnn Kuchera-Morin, and Tobias Höllerer. 2014. "Gibber: Abstractions for Creative Multimedia Programming." In *Proceedings of the ACM International Conference on Multimedia*, 67–76. MM '14. New York, NY, USA: ACM. doi:10.1145/2647868.2654949. http://doi.acm.org/10.1145/2647868.2654949.

Sorensen, Andrew, and Henry Gardner. 2010. "Programming with Time: Cyber-Physical Programming with Impromptu." In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, 822–834. OOPSLA '10. New York, NY, USA: ACM. doi:10.1145/1869459.1869526. http://doi.acm.org/10.1145/1869459.1869526.