TextAlive: Integrated Design Environment for Kinetic Typography

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 presents TextAlive, a graphical tool that allows interactive editing of kinetic typography videos in which lyrics or transcripts are animated in synchrony with the corresponding music or speech. While existing systems have allowed the designer and casual user to create animations, most of them do not take into account synchronization with audio signals. They allow predefined motions to be applied to objects and parameters to be tweaked, but it is usually impossible to extend the predefined set of motion algorithms within these systems. We therefore propose an integrated design environment featuring (1) GUIs that designers can use to create and edit animations synchronized with audio signals, (2) integrated tools that programmers can use to implement animation algorithms, and (3) a framework for bridging the interfaces for designers and programmers. A preliminary user study with designers, programmers, and casual users demonstrated its capability in authoring various kinetic typography videos.

Author Keywords

Kinetic typography; animation; creativity support tool; integrated design environment; live programming.

ACM Classification Keywords

H.5.2. Information interfaces and presentation (e.g., HCI): User interfaces – GUI; D.2.6. Software Engineering: Programing Environments – Integrated environments.

INTRODUCTION

Kinetic typography, a technique for moving and reshaping text in a video, is widely used in music, narrative, and commercial videos to deliver text information in a dynamic way. It can deliver the emotional feeling associated with a text in addition to the text's literal meaning [17]. When it is used in music and narrative videos, text is animated in synchrony with vocal music or speech. The animated text augments the listening experience with visual effects, helping listeners better comprehend what is being vocalized.

Kinetic typography videos have been produced with general-purpose tools for creating animations (e.g., Adobe Flash or After Effects [1]). While such tools provide a fine granularity of control over the animations, producing the final result takes long time even for the experienced user because manual synchronization of the text animation and audio is tedious. In addition, general tools do not provide any guidelines to help novice users produce convincing effects. Tools specifically designed for making kinetic typography videos (e.g., ActiveText [18] or Kinedit [7]) provide predefined templates of effective animations, from which the user can choose. Such domain-specific tools, though, have had a certain limitation in their flexibility. That is, the programming needed to create new animation templates has to be done in separate development environments.

With the goal of pushing forward the state of the art in kinetic typography design, we present TextAlive, a graphical tool that allows interactive editing of kinetic typography videos (Figure 1). While tools for designers and programmers have usually been designed separately, TextAlive is designed to benefit both. We therefore call it an integrated design environment (an integrated *development* environment is one designed only for the programmer). Thanks to its duality for the designer and programmer, its research contribution is three-fold. First, from an input pair consisting of audio and text information, TextAlive can automatically compose a playable kinetic typography video that serves as a concrete starting point for the designer. It also provides interactive user interfaces for editing the video in synchrony with the audio signals. Second, TextAlive allows live programming of animation templates with its built-in development environment that provides continuous graphical feedback to the programmer. Third, its unique framework allows the programmer to easily extend its GUI so that the programmer or designer can intuitively debug or customize the parameters of animation templates.

The remainder of this paper is organized as follows. Related work is introduced to highlight our research contribution. Next, the design and implementation of TextAlive's interaction for the designer and programmer are introduced.

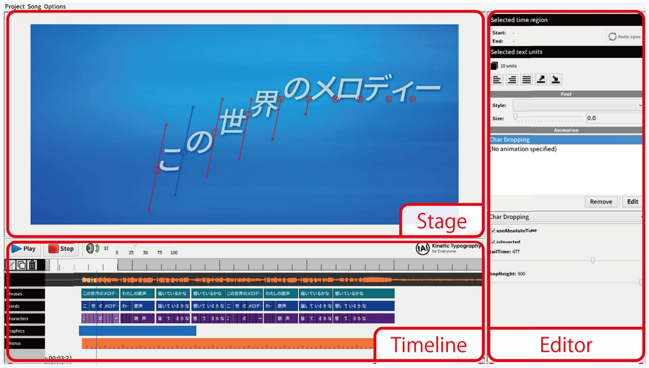


Figure 1. Overview of the TextAlive system.

CHI 2015, April 18 - 23 2015, Seoul, Republic of Korea

ACM 978-1-4503-3145-6/15/04…$15.00
http://dx.doi.org/10.1145/2702123.2702140

Then, the results of a preliminary user study are discussed. Finally, the conclusion and future work are presented. For better understanding, readers are encouraged to see the demonstration video and/or visit http://textalive.jp to try out the system before further reading.

RELATED WORK

Tools for Creating Kinetic Typography

Various tools for creating kinetic typography videos have been proposed. However, they focus on graphical aspects of creating videos. To our knowledge, none of them deals with audio signals synchronized with the videos. This is because their primary use of kinetic typography is for augmenting text-based communication. TextAlive, in contrast, aims to augment audio listening experience with dynamic graphical representation of text information. Prior work on editing time-series multimedia includes synchronized editing of transcription and narration [21]. This is a complementary technique and can be integrated into TextAlive.

The existing tools for kinetic typography videos target programmers, designers, or casual users, not all three. TextAlive is built on top of these. For instance, a framework to help programmers create kinetic typography videos by writing Java source code has been proposed [16]. TextAlive shares some concepts with this framework, such as the hierarchical structures of phrases, words, and characters for representing text information. The ActiveText [18] and Kinedit [7] systems allow designers and casual users to input text, choose animations from several templates, and edit their parameters by using sliders and other standard GUIs. TextAlive also provides similar features. A combination of natural language processing (NLP) and heuristics enables the extraction of emotional information from text and automatic generation of the associated videos [20]. While the existing techniques benefit from NLP, they do not allow user interaction, which is essential if the user is to express his/her creativity.

The existing tools have limitations because of their targeting certain users. While tools for designers and casual users provide interactive GUIs, they typically force the user to use a predefined set of motion algorithms and do not allow any extensions to be implemented. The tools for programmers, on the other hand, have extendibility based on their APIs, but their development environments are usually equipped with text-based user interfaces and do not provide interactive feedback on the source code changes.

There is a visual programming environment that overlays code elements on top of the animation and allows binding a set of elements to text units [3]. Each element represents one typography property (font size, color, style, or scaling), and the set of elements affects the properties of nearby text in a way that depends on the distance between a set element and a text unit. It does not allow precise control of properties, nor does it provide any abstraction, preventing reuse of the created animations. Instead of using visual programming, TextAlive integrates text-based programming into a graphical tool. It provides more precise and flexible control over the text animation and an abstraction of an animation template that is reusable not only by the programmer but also by designers and casual users.

Tools for Creating Animations

While this work focuses on text animation, it is closely related to prior work on tools for general animation. Adobe Flash and After Effects [1] are professional tools used to create various kinds of animation. The user specifies key frames for objects, and the system smoothly interpolates the motion between the frames. These tools have good support for direct manipulation, such as visualizing object trajectories and using drag-and-drop to change the motion or seek time periods. Compared with these professional tools, TextAlive has a limited, or rather focused, GUI for editing text animations synchronized with audio signals, including a timeline interface. The user does not specify key frames, but rather timings of each text unit being vocalized. The timing information is then passed to animation templates that output fluid motions. TypeMonkey [2] is a plug-in for After Effects that has additional user interfaces for choosing text animations from templates and editing their parameters for the purpose of making kinetic typography videos. While it reduces the pain of key-framing, it still requires manual synchronization of the animation and audio. In addition, it has the same issue as other tools for kinetic typography: the limited number of templates restricts the user's expressivity.

Tools that let casual users make animations, such as Microsoft PowerPoint and Movie Maker, do not require key-framing. However, the resulting motions are usually limited to those following simple trajectories. In particular, various sketching interfaces allowing the casual user to create animations have been proposed. K-Sketch [4] allows users to specify the motion of objects by drawing trajectories, which is implemented as our "Trace Path" animation. Draco [15] is a sketch-based tool to animate multiple objects. We also target casual users who make animations of multiple objects (text units) but with more finely grained control for each object. In addition, TextAlive integrates a programming environment that allows authoring of reusable animation templates.

Tools for Live Programming

Live programming [19] is a technique to provide the programmer with continuous feedback about the program being developed. It aims to get rid of the gulf of execution by providing a seamless programming experience with regard to writing code, executing the program, and debugging it.

Live programming in TextAlive is inspired by Victor's demo [23] that allows the programmer to dynamically rewrite parameters of graphical applications during their execution. In our system, the programmer can simply click the "update" button, and there is no need to re-launch the entire program and reconfigure the parameters. It allows the programmer to rewrite the logic of animation templates without losing their parameter information. VisionSketch [13] uses a similar technique, where the logic of image processing algorithms can be dynamically replaced, but it does not preserve the state information when the logic is updated. Subtext [6] continuously revalidates the program without explicit compilation. It updates the program output upon the typing of each character in the source code editor. We intentionally avoided such implicit revalidation and kept the “update” button because the implicit revalidation flushes the screen very often and is distracting.

Professional tools for animation are typically equipped with scripting engines. Adobe Flash and Adobe After Effects, for example, have a built-in scripting engine to control the properties of objects. Some tools meant for casual users, such as Microsoft PowerPoint, also allow such scripting. While most scripting engines allow conversational programming, these tools do not have decent support for bridging the activities of writing code and running the program. On the other hand, Juxtapose [12] is capable of generating a slider bound to a variable value, and Unity allows the programmer to extend its GUI for runtime parameter tuning. Within these environments, GUI widgets bridge that gap between the static definition and the runtime behavior of the program. TextAlive takes the same approach, balancing the variety of widgets (more than Juxtapose) and the lines of code (less than Unity). Moreover, TextAlive is specialized for animation authoring. For instance, a set of APIs is provided for easily animating text units. The Stage interface can visualize object trajectories calculated from the user code. Whereas the prior work intended for game development has such visualizations [23, 9], TextAlive uses the trajectories not only for visualization but also for seeking time periods.

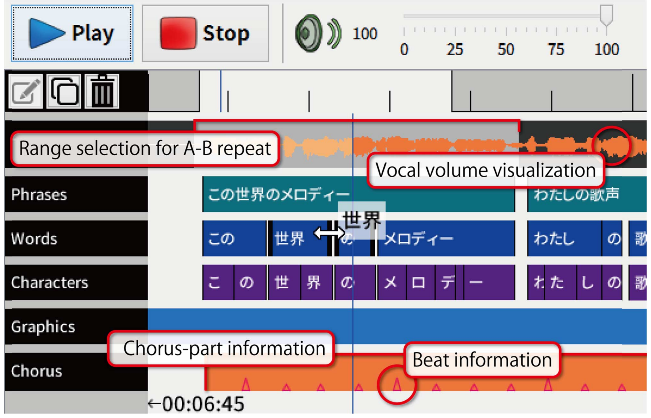
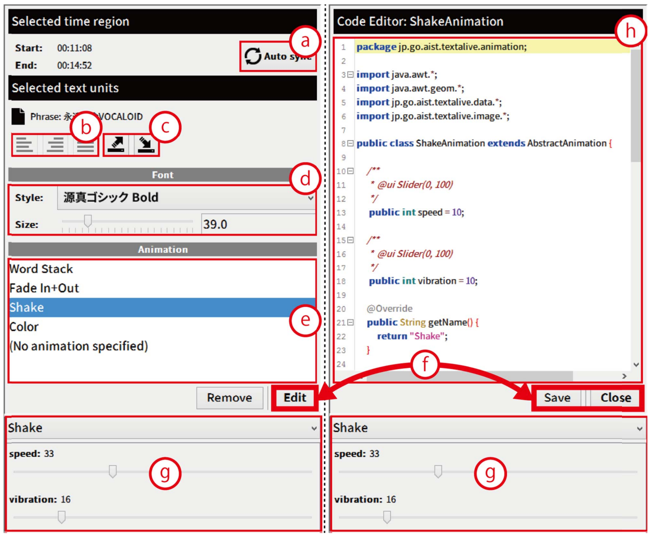


Figure 2. Timeline interface.



Selected time region

Start: 00:11:08
End: 00:14:52

Selected text units

Phrase: 永遠 VOCALOID

Font

Style: 源真ゴシック Bold
Size: 39.0

Animation

Word Stack
Fade In+Out
Shake
Color
(No animation specified)

Shake
speed: 33
vibration: 16

Code Editor: ShakeAnimation

```
package jp.go.aist.textalive.animation;

import java.awt.*;
import java.awt.geom.*;
import jp.go.aist.textalive.data.*;
import jp.go.aist.textalive.image.*;

public class ShakeAnimation extends AbstractAnimation {

   /**
    * @ui Slider(0, 100)
    */
    public int speed = 10;

   /**
    * @ui Slider(0, 100)
    */
    public int vibration = 10;

   @Override
   public String getName() {
      return "Shake";
   }

```

Shake
speed: 33
vibration: 16

Figure 3. Editor interface and integrated source code editor; the user can switch between them by clicking the buttons (f).

TEXTALIVE: AN INTEGRATED DESIGN ENVIRONMENT

This section introduces the interaction design of TextAlive. The first subsection describes interactive editing of kinetic typography videos by designers or casual users. This sort of editing does not require prior knowledge of kinetic typography or programming. The second subsection describes live programming of animation templates by novice programmers. These days, it is not uncommon for the designer to have programming skills. For instance, programming skills in scripting are required to precisely control many objects when using existing tools such as Adobe After Effects [1]. In TextAlive, programming abilities enable designers and casual users as well as programmers to control not only the currently present objects on the screen but also their later reuse and customization.

Interactive Editing of Kinetic Typography Videos

TextAlive can automatically create a kinetic typography video when given an audio file and its transcription. Whereas the existing tools require the user to create a video from scratch and to spend much time on manual synchronization of audio and text, TextAlive significantly reduces the effort required. The composed video can be interactively edited so that it matches the user’s style.

Automatic Video Composition and Timing Correction

The user first inputs a pair consisting of a song and its lyrics or a narration and its transcription. Then, the system estimates the timings of each *character* in the text being vocalized [8]. Through this process, not only *characters* but also *words* that consist of characters and *phrases* that consist of words and correspond to one line in the original text come to have timing information about the starting and ending of the vocalization. The system then assigns a default animation template to each *text unit* (a character, word, or phrase) and synthesizes a playable video. If a song is provided, its chorus parts are detected and assigned a more gorgeous animation template (explained in the implementation section). Beat information is also analyzed, and the result is visualized on the Timeline (Figure 2). After this automatic synthesis, the video is shown on the Stage interface (Figure 1), and it will always be playable by clicking the “play” button in the Timeline interface. Any operation described hereafter can be carried out interactively so that the video can be instantly updated.

TextAlive cannot always estimate correct parameters. If there is an error in the estimation, the timing information can be interactively corrected in many ways before being saved for use the next time the same pair of audio and text is loaded. For instance, text units in the Timeline can be drag-and-dropped to instantly fix their timing information. Drawing a rectangle in the Timeline selects multiple text units that overlap the rectangle. Chorus parts are highlighted in the bottom row of the Timeline, and clicking them selects the text units vocalized during the chorus. Once the text units are selected, the “align left,” “center,” “right,” and “justify” buttons in the Editor interface (Figure 3b) can be used to fix their timings collectively. Timing information can also be copied and pasted (Figure 3c), and this is quite useful for groups of text units with the same structure, such as repeating chorus parts.

The seek bar in the Timeline visualizes the sequential changes in vocal volume information extracted from the original audio signals, thereby helping the user to estimate correct timings visually. The seek bar allows one to select a specific time region. While a region is selected, the “play” button plays the corresponding part repeatedly, allowing one to concentrate on correcting the timing and editing the animation of the specific part. Automatic synchronization estimating the timings of the text units contained in the selected region can be applied again by clicking “auto-sync” in the Editor interface (Figure 3a).

Font and Animation Template Assignment

An automatically created video does not at all reflect the user’s intention. To make it his/her own, the user selects text units in the Timeline and changes their properties, including fonts, and animation templates with the Editor interface (Figure 3d, e). As in timing correction, property changes can be collectively made on multiple text units. For instance, when a phrase is selected, changes in the font style and size are applied to all of the characters in the phrase.

Predefined animation templates allow novice users to create aesthetic kinetic typography. To avoid implausible visual effects, they are not always applicable to every type of text unit: phrases, words, and characters have different sets of assignable animation templates. In addition, multiple templates can be assigned to a single text unit at the same time. For example, “Sliding Animation” can be assigned to a phrase, and “Hopping Animation” can be assigned to the characters in that phrase. “Karaoke Animation” that changes the text color before/after vocalization can also be assigned to the characters (Figure 4). Combinations of different templates that can be assigned to suitable types of text units allow flexible yet effective kinetic typography design.

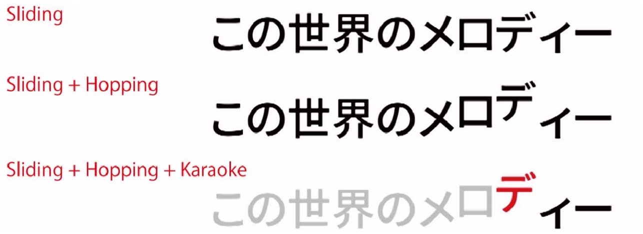


Figure 4. A simple example of kinetic typography effects generated from different sets of preset animation templates.

Animation Customization

Each animation template has parameters for customizing its appearance, and these parameters can be interactively manipulated with the GUI widgets shown on the Editor (Figure 3g) or the Stage. These widgets are dynamically generated by the template implementation, which is described in detail in the next subsection. They appear when the corresponding template is selected in the Editor interface.

While the Stage can usually show only the current frame of interest in the video, it would be desirable that the user can instantly see how an object’s motion is affected by parameter tuning. To make this possible, the trajectories of the selected text units can be visualized on the Stage (Figure 1). The trajectories can be used to seek time periods in the way shown in prior work [5].

Live Programming of Animation Templates

While each animation template can be customized to meet the user’s needs, it still limits the resulting animation to a certain extent. General tools for creating animations are often equipped with scripting engines with which the animation of lots of objects can be controlled in a mathematically precise way. Such scripting, though, typically does not allow later reuse because it lacks an abstraction mechanism. There is not much graphical feedback while the user is writing code either.

TextAlive aims to address this issue by enabling “live programming” of animation templates: the user can seamlessly go back and forth between editing animation and implementing templates (Figure 3f). The implementation can be updated by just one click and there is no notion of compilation or execution. The program that creates the animations continues running virtually. The update process updates not only the behavior of the text units but also the GUI widgets specific to the template (Figure 3g). These widgets help the programmer debug the template and help the user interactively customize it.

Implementation of Animation Templates

To read the implementation of an existing animation template, the user selects the template and clicks the “edit” button in the Editor interface. TextAlive then shows the code editor, with which the source code written in Java can be edited (Figure 3h). Each template is defined as a Java class that implements a specific Java interface. The template implementation can be updated by clicking the “update” button, which instantly updates animations of the text units that use the template. This update can be done at any moment, such as during the playing of a song or when the user is observing the motion of text units. To create a new template, the user can either click the “new” button or change the class name in the source code of any existing template and click the “update” button.

Updating running software is generally called a hot swap. It replaces existing objects using old code with dynamically generated new objects using new code. Since such a process simply discards the state information of the old objects, the state information (in our case, customized parameters of an animation template) usually needs special care, such as by the programmer writing some code to store and restore it. TextAlive handles this transparently, which frees the programmer from having to write boilerplate code and allows him/her to concentrate on the algorithm.

Implementation of GUI Widgets for Each Template

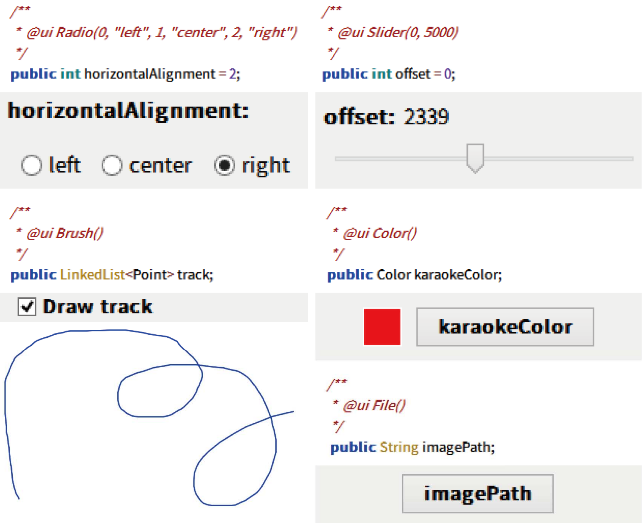
As discussed in *Animation Customization*, each template has dedicated GUI widgets to customize its appearance. "Sliding Animation," for instance, has a box of radio buttons to select its horizontal alignment and a slider to control its sliding angle. In order to implement a GUI widget, the programmer should declare a Java public field and write a comment block right before the field declaration in the source code. The comment should be written in a simple format: "*@ui WidgetName([options])*." The generated GUI widget is bound to the public field; GUI operations on the widgets change the field values.

The widgets are not only meant to help designers and casual users manipulate the parameters intuitively. They can also help the programmer debug and test the template during the development process. For instance, the programmer can temporarily create a box of radio buttons for switching between multiple implementations having a similar effect and check which one works best. Once he/she knows which one is best, he/she removes the radio buttons and publishes it for other users.

Below is the list of preset GUI widgets available in the current implementation of TextAlive (Figure 5). Other widgets can be easily added by writing Java source code.

@ui Slider(min, max) shows a slider to change the value of the corresponding integer field between *min* to *max*.

@ui Check() shows a check box to switch between true and false values of the corresponding boolean field.



```
/**
 * @ui Radio(0, "left", 1, "center", 2, "right")
 */
public int horizontalAlignment = 2;
```

```
/**
 * @ui Slider(0, 5000)
 */
public int offset = 0;
```

```
/**
 * @ui Brush()
 */
public LinkedList<Point> track;
```

```
/**
 * @ui Color()
 */
public Color karaokeColor;
```

```
/**
 * @ui File()
 */
public String imagePath;
```

Figure 5. Comments and GUI widgets generated from them.

@ui Radio("label1", v1, "label2", v2, …) shows a box of radio buttons used to select a value.

@ui Color() shows a button with a colored rectangle so that the user can choose a color with the color picker dialog.

@ui File() shows a button used to choose a file whose path is stored in the corresponding String field.

@ui Track() shows a check box used to enable/disable drawing a path by dragging the mouse on the Stage. When a path is drawn, its data is represented as a list of (x, y) coordinates and stored in the corresponding List field. This is used to implement the "Follow Path" animation template that makes text move along the specified path.

IMPLEMENTATION

This section introduces the current implementation of the TextAlive system. It first gives an overview of the system along with the techniques used in the implementation, and then it explains the system in more detail. The first subsection explains how the creation of plausible kinetic typography videos is supported, and the second explains how a live programming experience is provided.

Overview

The architecture of TextAlive is shown in Figure 6. It is currently implemented with Java 7 (64-bit) and tested on a personal computer with a Windows 8.1 (64-bit) OS. Input sound files should use the MPEG-2 Audio Layer III format for playback in the system and are converted to the monaural MS WAV format using FFmpeg for further audio signal processing. Input text is written in a natural language (either Japanese or English) and is converted into a series of phoneme information by using MeCab, open-source software for morphological analysis and part-of-speech tagging.

The audio signal is synchronized with the corresponding phoneme information [8] by attaching timing information to the phrases, words, and characters of the original text. If the input sound file is a song, the beat information and the chorus

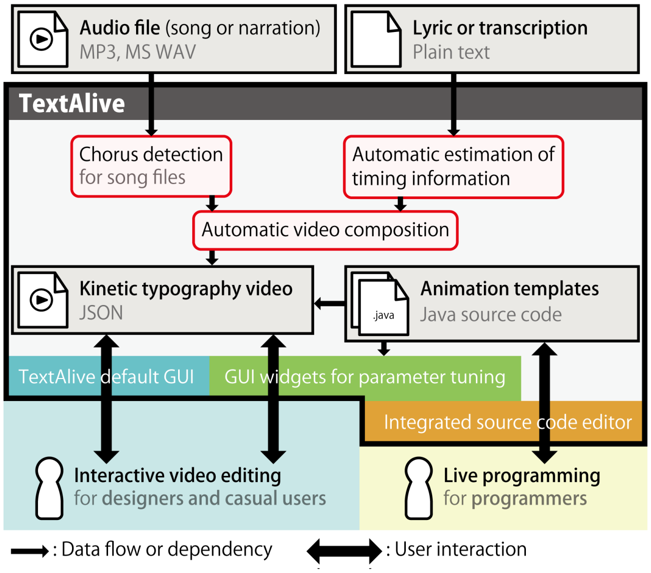


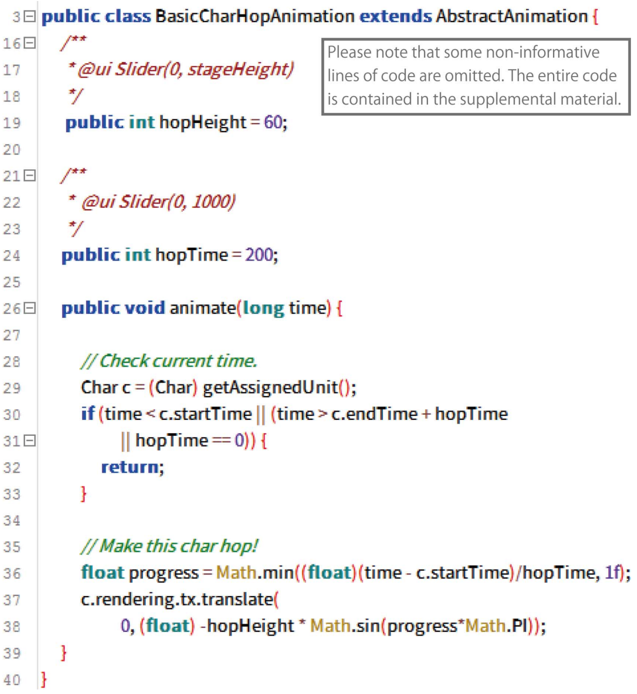
Figure 6. Software architecture of the TextAlive system.

part are detected [10]. This information is referenced for automatic video composition. The Java source code of animation templates is dynamically compiled with the Eclipse Compiler for Java and loaded into the Java VM with the Java Reflection API. In order to store and restore the state information of the animation templates, a text-based JSON format is used as a temporary backup. The animation data is also saved as a JSON file (Figure 9).

Implementation for Kinetic Typography Design

Automatic Video Composition

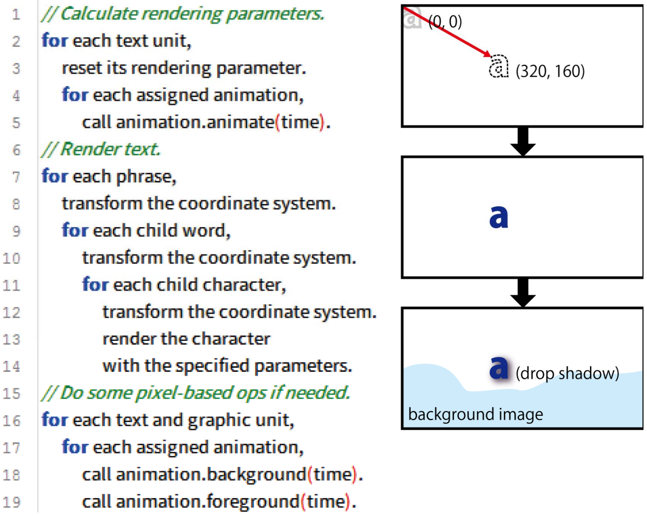
Given the automatically estimated parameters, TextAlive automatically assigns animation templates to text units and generates a playable kinetic typography video. By default, it simply assigns the "Sliding Animation" to all phrases and the "Karaoke Animation" to all characters. If the input sound is a song, phrases in the chorus part are assigned the "Sliding and Rotating Animation" and characters in the chorus part are additionally assigned "Hopping Animation" so that the chorus part looks more prominent than the other parts.



```
public class BasicCharHopAnimation extends AbstractAnimation {
  /**
   *@ui Slider(0, stageHeight)
   */
   public int hopHeight = 60;

  /**
   * @ui Slider(0, 1000)
   */
  public int hopTime = 200;

  public void animate(long time) {

    // Check current time.
    Char c = (Char) getAssignedUnit();
    if (time < c.startTime || (time > c.endTime + hopTime
          || hopTime == 0)) {
      return;
    }

    // Make this char hop!
    float progress = Math.min((float)(time - c.startTime)/hopTime, 1f);
    c.rendering.tx.translate(
        0, (float) -hopHeight * Math.sin(progress*Math.PI));
  }
}
```

Please note that some non-informative lines of code are omitted. The entire code is contained in the supplemental material.

Figure 7. Example implementation of an animation template.



```
// Calculate rendering parameters.
for each text unit,
  reset its rendering parameter.
  for each assigned animation,
    call animation.animate(time).
// Render text.
for each phrase,
  transform the coordinate system.
  for each child word,
    transform the coordinate system.
    for each child character,
      transform the coordinate system.
      render the character
      with the specified parameters.
// Do some pixel-based ops if needed.
for each text and graphic unit,
  for each assigned animation,
    call animation.background(time).
    call animation.foreground(time).
```

Figure 8. Pseudocode for rendering each frame.

The strategy that defines how to assign animation templates is currently written as Java source code and can be easily customized. We plan to implement a GUI for choosing and implementing new strategies.

Text Animation

In a lot of kinetic typography videos, phrases, words, and characters have their own motion algorithms independent from the others. In order to animate text information in a human-readable way, characters often keep their positions relative to each other and are aligned horizontally or vertically. Multiple words are often organized into a phrase that is prevented from overlapping with other phrases. Such motion algorithms can be described in a simple and clean way by using the coordinate system relative to the parent text units (e.g., characters to words and words to phrases).

Given the characteristics of text animation, each text unit holds its timing information (when it starts and ends being vocalized), a list of assigned animation templates, a font name, and one or more characters contained in the unit. Each animation template is required to implement a method named *animate(time)* whose argument is the current time. An example implementation is shown in Figure 7. The template has access to its assigned text unit, and the *animate* method modifies the rendering parameters of the text unit. The parameters include font color, an alpha compositing rule (how to blend the text graphic with the existing pixel values), and an affine transform matrix (a three-by-three matrix that defines how to transform the text graphic and includes translation, rotation, and scaling) relative to the parent text unit. Since the *animate* method can be called with any *time* value at any time, the method is usually expected to be a pure function – in other words, it should return a value without modifying the state of the animation template object.

While changing the rendering parameters can reshape the appearance of the text unit as a vector graphic, there are kinetic typography effects that cannot be supported by such reshaping and instead demand pixel-based operations. They include adding blur or shadows to the text graphics. To support such effects, an animation template can implement *background(graphic context, time)* and *foreground(graphic context, time)* methods that render graphics behind and in front of the text units. Since these methods are called after rendering all text units in the off-screen buffer by using the *animate(time)* method, they have access to the pixel-based representation of the text units.

TextAlive plays and renders the video in a 30-frames-per-second format. The pseudocode for rendering each frame is shown in Figure 8. Graphic units are optional units that hold the start and end times as well as a list of animation templates mainly used for rendering the background or foreground to complement or emphasize text information.

Trajectory Rendering

Trajectory rendering allows the user to see the past and future motion at a glance. The same procedure as used for rendering frames is repeatedly called without actually using rendering graphics, and the absolute time-series positions of the selected text units are visualized as their trajectories.

The trajectory can also be used as an interface to seek time periods. Since the system knows the correspondence between a specific point in the trajectory and timing, the current mouse position can easily be converted into timing information by finding the closest point in the trajectory. When the trajectory is not straight-forward, though, following the closest point sometimes results in a big jump in time, which is not desired. To avoid this problem, the distances between the mouse position and the next or previous point in the trajectory are calculated.

Implementation for Live Programming Experience

Hot-swap for Updating Animation Templates

Animation templates are represented by Java classes that are compiled and dynamically loaded into the Java VM when TextAlive launches. After the user edits their source code, the corresponding classes are recompiled and the resulting Java bytecode is reloaded into the VM.

Such a process, i.e., replacing classes without restarting the VM, is called a hot-swap. A general hot-swap simply replaces an old class definition with a new one and is not sufficient for updating animation templates. If the updating process only updated the class definition, existing objects with old definitions would not be updated. To see the updated motion, the user would need to remove the objects, re-assign the animation template, and reconfigure parameters.

To avoid such hassle, TextAlive automatically extracts state information from old objects, creates new objects with the updated implementation, and makes its best effort to restore the state information. Before a hot-swap, a backup of the state information of the existing objects is created by calling the Java Reflection API to serialize all public fields of each object. It is used after the hot-swap to construct objects with the updated class definition, in effect replacing the old objects. The process of restoring state information is identical to that of loading videos, which is described later.

Static Code Analysis for Generating GUI Widgets

The GUI widgets for customizing animation templates are generated by static code analysis upon compilation of the class definition. The widgets can collectively change the values of multiple variables (i.e., the same parameters of multiple animation template objects) when multiple text units are selected in the Timeline.

Each comment that begins with *@ui* and is located right before a public field declaration is parsed and divided into the widget class name and its options. The option values are evaluated using a subset of the Java programming language, thereby providing dynamic parameters to widgets. The language supports simple mathematical calculations and read-only access to useful parameters such as the resolution of the video (the size of the Stage). For instance, a slider with the maximum value of the height of the Stage can be generated by writing *"@ui Slider(0, stageHeight)"* as shown on line 17 of Figure 7.

Saving and Loading Videos

TextAlive saves a kinetic typography video in plain text JSON format. It consists of a hierarchical architecture of phrases, words, and characters. Each text unit has the vocalization start and end times and a list of information about the assigned animation templates. Unlike phrases and words, characters are also saved with their font name and size. Each animation template is represented by its name and the customized parameters. Figure 9 shows an example of a JSON object of a video.

Loading videos is exactly the reverse of saving them, except for the chance of animation templates being updated. If an animation template has an updated definition, one or more public fields might be added or removed. To construct an animation template object with an updated definition, TextAlive makes its best effort to maintain the state information; newly added fields have their default values, and old values of removed fields are simply discarded.

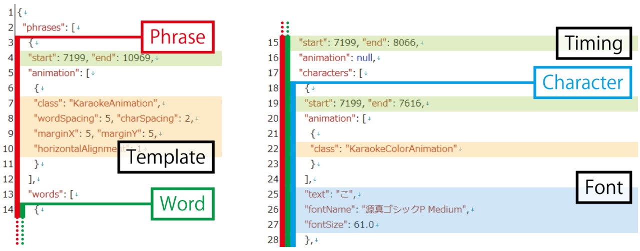
PRELIMINARY USER STUDY

We conducted a preliminary study to gain user feedback on the TextAlive system and to investigate the capabilities, limitations, and potential of our interaction design. Seven users with varying levels of expertise in authoring videos and programming graphical applications participated in the study. They were asked to use the system and to create a kinetic typography video with one favorite song or narration. They were asked to compare their use of the system against their prior experience on tools for creating videos and graphical applications.

Participants

Seven participants aged 23 to 30 years old (mean 27) took part in the study. Each was paid $30 for their participation.

P1-4 are designers and casual users (two females and two males). P1 is an amateur singer who has used Adobe After Effects [1] to create a simple slide show for her music video; P2 is an amateur illustrator who has no prior experience in



```
{
 "phrases": [
  {
  "start": 7199, "end": 10969,
  "animation": [
   {
   "class": "KaraokeAnimation",
   "wordSpacing": 5, "charSpacing": 2,
   "marginX": 5, "marginY": 5,
   "horizontalAlignment": 1
   }
  ],
  "words": [
   {
```

```
  "start": 7199, "end": 8066,
  "animation": null,
  "characters": [
   {
   "start": 7199, "end": 7616,
   "animation": [
    {
    "class": "KaraokeColorAnimation"
    }
   ],
   "text": "こ",
   "fontName": "源真ゴシックP Medium",
   "fontSize": 61.0
   },
```

Figure 9. JSON-formatted text for saving the animation.

creating videos; P3 is an amateur disk jockey who has extensive experience in mixing music and recording narrations for Rakugo (a traditional form of Japanese storytelling) but no video authoring experience; P4 is an interaction designer who has used Motion, Final Cut Pro, openFrameworks, and Unity [22] for creating videos.

P5-7 are programmers and are all male; P5 has a small amount of experience creating demonstration videos for his software; P6 is an experienced user of Adobe After Effects (> 2 years) and has created music videos; P7 has a hobby of writing songs and has created music videos with AviUtl, which is freeware for casual video authoring.

Experimental setup

All participants used a standard set of laptop computers with a full HD (1920 x 1080 pixels) display and a mouse (Figure 10). Some of the participants brought their headphones or earphones; others used earphones we provided. The TextAlive system occupied the entire screen during the study. The experiment followed the steps described below and took 2-3 hours for each participant.

Pre-experiment questionnaire: Each participant was asked to fill out a form asking his/her experience in authoring songs, narrations, videos, and programming.

Introduction and demonstration: Each participant was given a 5-15 minute introduction to the TextAlive system by the instructor. The instructor then demonstrated the timing correction, font and animation template assignment, and animation customization using the GUI widgets. For the programmers (P5-7), the instructor also demonstrated the use of the integrated source code editor to modify and create an animation template.

Kinetic typography design: Each participant was given 2 hours maximum to create a kinetic typography video of his/her favorite subject with the system. The participant was allowed to bring data of a song, narration, or image to be used in the video or could choose from a list of the songs we prepared. Since it is usually impossible to create a video for an entire song or narration in 2 hours, the participant was asked to work on his/her favorite part of the song or narration. During this open-ended task, the participant was free to ask any questions about the usage of the system, including how to use the preset animation templates and APIs for implementing animation templates. The participant was allowed to end the study when he/she was satisfied playing with the system.

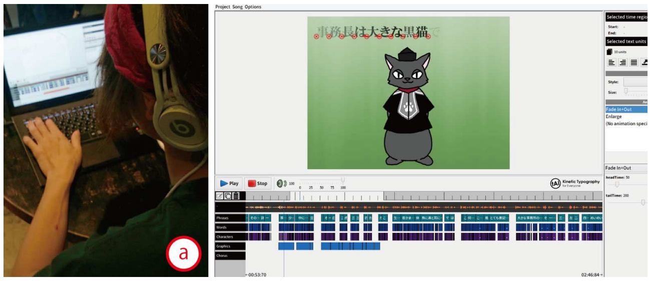


Figure 10. Setup of the user study. (a) A participant using our system. (b) A screenshot taken during the study.

Post-experiment questionnaire: Each participant was asked to fill out a form asking five questions about the usability of the system (selected answers on a 7-point Likert scale), and three questions about the good and bad points of the user interface.

Results and Lessons Learned

All participants could successfully create kinetic typography videos for their favorite part of a song or narration, some of whose screenshots are shown in Figure 11. Two participants used a song we prepared, while the others used their favorite song or narration. The results of the post-experiment questionnaire consisting of the mean, standard deviation, and percentage of positive responses (>4 on a 7-point Likert scale) are shown in Table 1.

Everyone who participated in the study was positive about how suitable TextAlive is for authoring kinetic typography videos (Q5). There were no answers representing negative impressions of the TextAlive system on Q1-3, but there were several neutral answers (= 4 points). The neutral answers were found to come from non-critical reasons such as the participant not being enthusiastic enough in creating kinetic typography videos (P1) and practical reasons that can be addressed by making minor revisions to the system such as adding keyboard shortcuts to frequently used commands (P7). Those who used the live programming feature (P5 and P7) indicated they felt the need for technical support in order to use the system. Their negative answers stem from the programming experience described later.

Automatic video composition was useful for both novice and experienced users: All participants liked the automatic video composition feature a lot. P7 commented *"automatically estimating timing information and synthesizing a playable video is extremely useful, reducing the effort always needed for existing tools I have used."* P3 commented *"it is nice that we no longer need to create but just edit the video."*

The abstraction of phrases, words, and characters was welcomed: Participants with prior experience of creating videos with text information such as subtitles (P1, P4, P6) especially favored the hierarchical abstraction, enabling assignment of different animation templates to each phrase, word, and character. P1 noted that *"the capability of modifying the properties of each character is very useful."* This feature is often missing from existing video authoring tools.

The abstraction of animation templates suggested potential applications: While animating text in a specific time period requires manual parameter tuning in existing authoring tools, animation templates in TextAlive adapt the animation to fit in the period. This allows the user to almost instantly animate text in synchrony with sound. P3 and P4 expressed their desire to play "text jockey" during their live disc jockey performances. P3 also foresees using it for Rakugo storytelling.

No need to stop the video, enabling a fluid experience: Many users appreciated that almost every operation on the system interactively updates the resulting video. P5 commented that *"there was no need to pause or stop the video. With TextAlive, I could select the time region of interest and play the video continuously while editing it. The only exception was when I got too tired to listen to the same part of the song."* P6 appreciated that *"animation templates can be edited in-place (without launching external editors.)"*

Having a variety of animation templates counts: P6 appreciated that the system allows novice users to create good looking videos simply by choosing animations from a list of preset templates. P1 and P7 requested more variety of templates. P4 wrote that *"more variety of animation templates would make TextAlive more appealing, but too many templates might overwhelm the user."* Actually, P3 and P7 complained that the current combo box for choosing an animation template is not informative enough. They commented that more detailed explanations or graphical previews would be desirable. In addition, P6 pointed out the difficulty of predicting the result of combining multiple animation templates, suggesting that we provide preset combinations of animation templates.

The Timeline interface is at the heart of synchronizing audio and video: All users appreciated the usability of the Timeline interface dedicated to timing correction. P1, P2 and P5 especially appreciated the wave-form visualization of vocal sound and the visualization of beat information. In addition, P1 and P5 requested snapping to the beat information while moving or resizing text units in the Timeline.

The Stage interface should allow (more) direct manipulation: P3 appreciated that most of the interfaces for customizing appearance of the selected units are gathered in the Editor interface. On the other hand, P2 commented that most of the preset animation templates only use the widgets in the Editor interface and the animation templates should make more use of the Stage interface for their customization. Currently, the "Follow Path" animation is the only preset template that allows the user to interact with the Stage to interactively update the tracing path. P1, P3 and P4 requested that simple animations should be customized by direct manipulation on the Stage, e.g., allowing drag-and-drop of text units to fix their positions at a certain timing. However, it is not straightforward to support direct manipulation of objects whose motion is defined by a combination of multiple algorithms. Since each animation template transforms the coordinates and does not know how the other templates are implemented, the parameters for the template to move the assigned unit to the mouse cursor position cannot be easily retrieved. The process is identical to finding a set of parameters for an unknown function that returns a specific value. Similar problems in physical simulations have been addressed by introducing novel interaction techniques that might be applicable to our case [11]. More improvements to the sketching interaction might also be helpful, as has been indicated in prior work [15].

#	Question	The TextAlive system Mean	SD	%
1	I would like to use it frequently.	5.57	1.20	5/7
2	I found it unnecessarily complex.	2.71	1.02	0/7
3	I thought it was easy to use.	4.86	0.75	4/7
4	I needed technical support to use it.	3.29	1.74	2/7
5	I thought it is suitable for authoring kinetic typography videos.	5.57	0.75	7/7

Table 1. Results of the post-experiment questionnaire.

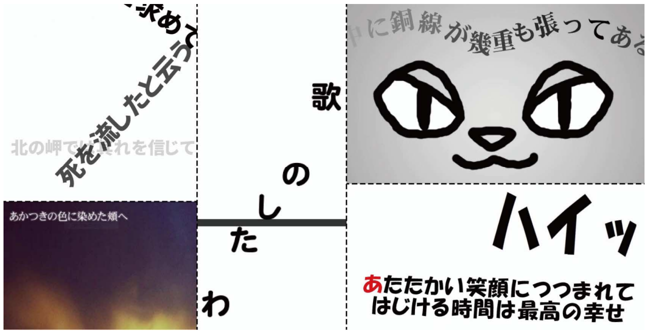


Figure 11. Screenshots of kinetic typography videos in various styles created by the participants (selected and trimmed.)

Programming ensures extendibility but needs training: Users who tried the programming interface (P5-7) all appreciated the extendibility of TextAlive, but at the same time, found it difficult to learn in 2-3 hours. P5 hesitated to use the feature. P7 commented that *"the live programming feature allows fine adjustments of animation but requires prior knowledge, making me procrastinate in its use for some time."* However, he was the most active person to create animation templates by the end of the experiment and also commented that *"the changes in code could be instantly applied to the video, which is nice."* P6 appreciated how easy it is to provide GUI widgets for tweaking parameters. Given that the live programming feature itself is not considered harmful, more software engineering effort to support more "learnable programming" [24] is desired.

CONCLUSION

We have described TextAlive, a tool for creating kinetic typography videos. It can automatically estimate the timing of each character being vocalized. Instead of providing a blank canvas and forcing the user to create videos from scratch, TextAlive analyzes a given pair of audio and text information and creates a playable and editable video. The video serves as a foundation on which the user can create his/her own video. The user interface for interactive editing of the video synchronized with audio signals was highly appreciated by the participants of the user study who had various professional backgrounds. The capability of editing animation templates by writing programs ensures the extendibility of the TextAlive system as a tool that can continue to evolve through use. While the user study confirmed its usefulness, its learning curve should be gentler, and more software engineering help provided.

Although the number of websites for sharing videos and audio files is increasing and more people are getting involved in creating new media content, it is still not easy to create videos synchronized with audio. Music videos on such websites typically feature the artists and illustrations. The lyrics are often placed in the safe area near the edges of the video or are not rendered on the video at all. Kinetic typography is effective at expressing the message of a song or narration visually. The tool described in this paper enhances the expressiveness of casual users by enabling them to create kinetic typography videos. While its current implementation is limited to placing objects on a two-dimensional plane, we are working on an extension to three-dimensional space.

TextAlive is available online at http://textalive.jp. Currently, many songs and narrations and their lyrics and transcriptions are available on the Internet. Preprocessing such information allows the user to create kinetic typography videos without first wading through the synchronization process. Corrected timing information can be shared with others, enabling crowdsourcing of such timing information. Animation templates can also be easily shared with others, meaning that the integrated design environment can be extended cooperatively. While this study focused on the experience of users with different backgrounds, the proposed interaction design enables various forms of collaboration (e.g. programmer with designer, designer with end user, etc.). Investigating such interactions would make an interesting follow up.

ACKNOWLEDGMENTS

This work was supported in part by JST, CREST.

REFERENCES

1. Adobe After Effects. http://www.adobe.com/products/aftereffects.html.
2. aescripts + aeplugins. TypeMonkey. http://aescripts.com/typemonkey/.
3. Chao, C. M. and Maeda, J. Concrete programming paradigm for kinetic typography. In *Proc. IEEE VL 1997*, 446-447.
4. Davis, R. C., Colwell, B. and Landay, J. A. K-Sketch: a “kinetic” sketch pad for novice animators. In *Proc. CHI 2008*, 413-422.
5. Dragicevic, P., Ramos, G., Bibliowitcz, J., Nowrouzezahrai, D., Balakrishnan, R. and Singh, K. Video browsing by direct manipulation. In *Proc. CHI 2008*, p.237-246.
6. Edwards, J. Subtext: uncovering the simplicity of programming. In *Proc. OOPSLA 2006*, p.505-518.
7. Forlizzi, J., Lee, J. and Hudson, S. The Kinedit system: affective messages using dynamic texts. In *Proc. CHI 2003*, 377-384.
8. Fujihara, H., Goto, M., Ogata, J., and Okuno, H. Lyric Synchronizer: automatic synchronization system between musical audio signals and lyrics. *IEEE Journal of Selected Topics in Signal Processing*, 5(6), 2011.
9. Fukahori, K., Sakamoto, D., Kato, J. and Igarashi, T. CapStudio: an interactive screencast for visual application development. *Ext. Abstracts CHI 2014*, 1453-1458.
10. Goto, M, Yoshii, K., Fujihara, H., Mauch, M. and Nakano, T. Songle: A web service for active music listening improved by user contributions, In *Proc. ISMIR 2011*, 311-316.
11. Ha, S., McCann, J., Liu, C. K. and Popovic, J. Physics storyboards. *Computer Graphics Forum 32*, 2 (*Proc. EG 2013)*, 133-142.
12. Hartmann, B., Yu, L., Allison, A., Yang, Y. and Klemmer, S. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *Proc. UIST 2008*, 91-100.
13. Kato, J., Sakamoto, D. and Igarashi, T. VisionSketch: Integrated support for example-centric programming of image processing applications. In *Proc. GI 2014*, 115-122.
14. Kazi, R., Chevalier, F., Grossman, T. and Fitzmaurice, G. Kitty: Sketching dynamic and interactive illustrations. In Proc. UIST 2014, 395-405.
15. Kazi, R., Chevalier, F., Grossman, T., Zhao, S. and Fitzmaurice, G. Draco: bringing life to illustrations with kinetic textures. In *Proc. CHI 2014,* 351-360.
16. Lee, J., Forlizzi, J. and Hudson, S. The kinetic typography engine: an extensible system for animating expressive text. In *Proc. UIST 2002*, 81-90.
17. Lee, J., Jun, S., Forlizzi, J. and Hudson, S. Using kinetic typography to convey emotion in text-basesd interpersonal communication. In *Proc. DIS 2006*, 41-49.
18. Lewis, J. and Weyers, A. ActiveText: a method for creating dynamic and interactive texts. In *Proc. UIST 1999*, 131-140.
19. McDirmid, S. Living it up with a live programming language. In *Proc. OOPSLA 2007*, 623-638.
20. Minakuchi, M. and Tanaka, K. Automatic kinetic typography composer. In *Proc. ACE 2005*, 221-224.
21. Rubin, S., Berthouzoz, F., Mysore, G. J., Li, W. and Agrawala, M. Content-based tools for editing audio stories. In *Proc UIST 2013,* 113-122.
22. Unity. http://unity3d.com/.
23. Victor, B. Inventing on Principle (2012). http://vimeo.com/36579366.
24. Victor, B. Learnable Programming (2012). http://worrydream.com/LearnablePrgramming/.