Data Unification on a Dataflow Visual Language for VJing

Atsutomo Kobayashi, Buntarou Shizuki and Jiro Tanaka
Department of Computer Science
University of Tsukuba
Tennodai 1-1-1, Tsukuba, Ibaraki, Japan
{*atsutomo,shizuki,jiro*}*@iplab.cs.tsukuba.ac.jp*

***Abstract*—We have been developing ImproV, a dataflow visual language system for VJing, which is a form of live performance involving simultaneous manipulation of imagery. The dataflow of ImproV represents a processing flow of video data and video effects that are familiar to performers. ImproV is designed to construct a combination of several video effects on the fly, enabling performers to create new video effects while VJing. We previously conducted a user study for the first prototype of this system. The results show that two data types of dataflow confuse performers and that connecting nodes is time consuming. Therefore, we unified the data types into a single type called `Video` to simplify dataflow and developed adjustment knobs to reduce the frequency of node connecting.**

The `Video` type, which is a sequence of images, is used to represent not only processing the target video but also parameters of video effects such as radius of blur and distance of translation. An adjustment knob on the input port is a new dataflow editing user interface for adjusting an effect parameter or opacity of a video flowing through the dataflow. In ImproV, all input ports on all nodes have an adjustment knob. These adjustment knobs allow performers to quickly fade video and adjust parameters.

***Keywords*-VJ; VJing; live video performance; DJ; DJing; image processing; end-user programming;**

I. Introduction

VJing is a form of live video performance and played mostly in night clubs. Performers of VJing are called VJs. Similar to DJing in a night club, VJing requires a VJ to keep playing videos improvisationally without interruption during the performance. The task includes choosing a video suitable to the mood and music, which dynamically changes during the performance, and seamlessly replacing the current video to the new one. Additionally, some visual effects (e.g., blur, and transformation) might be applied to the video before replacing.

To this end, a VJ uses one (or more in some cases) video mixer with multiple video inputs. When the VJ wants to change a video to another, he/she connects the new video source to an unused video input of the mixer while playing the original video with another video input, and seamlessly replaces the video with the new one by adjusting the knobs on the mixer and effectors. Lew [1] analyzed this task and found that it consisted of three steps:

STEP 1: Media retrieval
STEP 2: Preview and adjustment
STEP 3: Live manipulation

In traditional VJing, STEP 1 involves just choosing a pre-composed video the VJ prepared before the performance, and pre-defined video effects are chosen and applied in STEP 2. Then, the VJ starts showing the new video in STEP 3.

In contrast to such traditional VJing, we wanted to extend STEP 1 to enable a VJ to construct a new video effects structure by combining several video effects on the fly, giving him/her more chances to have the performance be truly improvisational. For this purpose, we developed ImproV, a datafow visual language system for VJing. With ImproV, the sole requirement for a VJ before a performance is to prepare simple videos. During the performance, the VJ can improvisationally create various effects from the prepared videos using ImproV's visual language.

We developed the first prototype of ImproV, whose language uses two data types, `Video` and `Value` [2]. To evaluate the first prototype, we conducted a user study with five professional VJs. For operational tests, the first author, who is a professional VJ, and an amateur VJ performed at actual musical events. Figure 1 shows one of the VJs using the first prototype at a jazz concert.

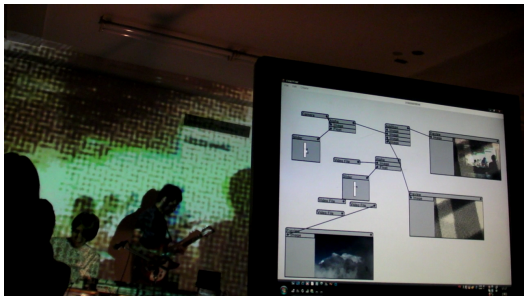


Figure 1. VJing with ImproV

As a result of the user study and operational tests, we found that while the dataflow diagram of ImproV is understandable for VJs, using two data types increases visual complexity of dataflow editing; thus, making the editing interface confusing. Moreover, connecting nodes is time consuming. To simplify dataflow editing, we unified the two data types into a single type called `Video`. To reduce

the frequency of node connecting, we developed adjustment knobs located on each input port that enable VJs to quickly adjust the input values.

After a brief description of the first prototype of ImproV, we explain the user study of the prototype. We then describe the unification of the two data types into a single type called `Video` and give examples that characterize this type. Next, we explain the adjustment knobs. Finally, we discuss possibilities of applying the unification of data types and adjustment knobs to dataflow visual language systems for other domains.

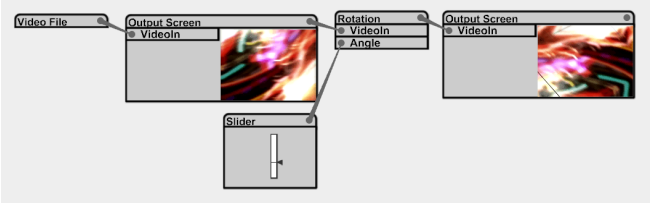


Figure 2. Dataflow of first prototype

II. First Prototype

ImproV is a video composing system. It provides a dataflow visual language, in which a node represents a video source, a video effect, or an output (or preview) screen. ImproV supports video files (including still images in Jpeg, GIF, and PNG), USB cameras, and audio inputs from sound cards. After data, video and audio, are captured, they are translated into images. All effects to the data are then processed in the graphics processing unit. Figure 2 shows an example of the dataflow composed on the first prototype of ImproV. All nodes have an output port on the top right and some have input port(s) listed on the left side.

In Figure 2, a video is rotated. The video file is imported using the `Video File` node, which reads its video file and repeatedly outputs the video stream frame-by-frame to its output port. The video is previewed using the `Preview Screen` node, which shows the content of the video on the dataflow editor, rotated using the `Rotation` node, and the rotation angle is specified using the `Slider` node. The `Angle` input port on the `Rotation` node accepts the `Value` type, which is a sequence of scalar values, and the other input port accepts the `Video` type, which is a sequence of images. Finally, the result of the `Rotation` node is output using the `Output Screen` node, which represents the display for the audience.

To create a node, a VJ chooses a node type from the GUI menu. Dragging from an output port to an input port connects the nodes, i.e., the action creates an edge between the nodes. Immediately after they are connected, the data begin to flow from the output port to the input port. In ImproV, output can fork to multiple inputs, and multiple outputs can be merged through mixer nodes corresponding to a blending mode, for example, alpha blending or addition blending.

III. User Study for First Prototype

We have conducted a user study for the first prototype of ImproV. The purpose of the user study was to confirm that the dataflow diagram is understandable for VJs, or those who have knowledge of video authoring, and to evaluate the operability of ImproV.

Five professional VJs (with experience of paid VJing) who all have knowledge of Adobe After Effects (AE) participated. The participants were first interviewed about their experience in VJing and skills with video authoring tools. Then, they participated in Experiment 1. After Experiment 1, they were told about ImproV and freely used ImproV for practice. Then, they participated in Experiment 2. Finally, they filled in a questionnaire.

Experiment 1

In Experiment 1, the participants were shown a dataflow diagram of ImproV and asked what the dataflow diagram expressed. Since Experiment 1 was conducted to confirm that a dataflow diagram is understandable for VJs, no information about ImproV was provided to the participants. To make the answers accurate, the participants were asked to construct a corresponding video processing structure using AE. Figure 3 is the dataflow diagram from Experiment 1.

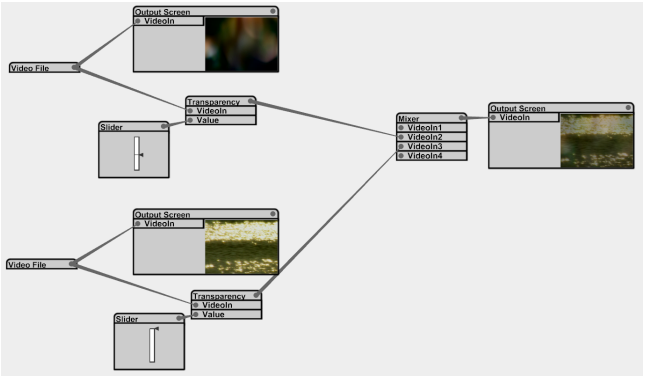


Figure 3. Dataflow diagram from Experiment 1

Experiment 2

In Experiment 2, the participants performed the following three tasks in both ImproV and AE:

- Task 1 Apply blur effect to the video displayed in the main output.
- Task 2 Merge two paths of video streams and switch them.
- Task 3 Apply blur effect to a part of the video displayed in the main output.

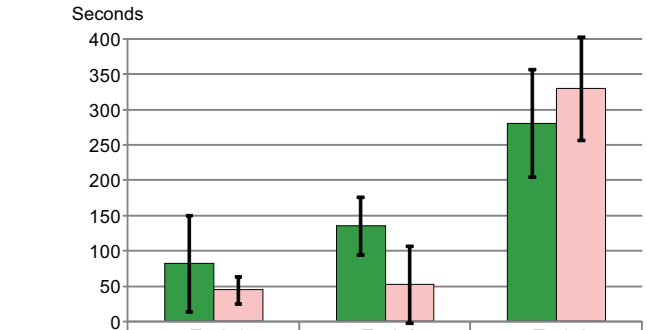
Task 1 is the simplest task that often used in VJing for applying an effect. Task 2 is the task that most often used in VJing for switching two videos. Task 3 is a complex task for creating a new effect for which Improve was developed.

The participants were able to ask questions about operating ImproV during the tasks. The completion times for the

tasks were measured to compare the operability of ImproV and AE. The participants performing the tasks were recorded for later analysis.

Results

In Experiment 1, all participants answered correctly. This result indicates that the dataflow diagram of ImproV is understandable for VJs.



	Task 1	Task 2	Task 3
ImproV	82.4	135.2	281.0
After Effects	44.4	52.2	330.0

Figure 4. Result of Experiment 2

Figure 4 shows the completion time of each task using the two systems in Experiment 2. We performed an analysis of variance to examine the difference between ImproV and AE. There was a significant difference between ImproV and AE in Task 2 ($F = 7.169, p < 0.05$), and there were no significant differences in Tasks 1 and 3. The participants completed Task 3 slightly faster with ImproV than with AE. However, it was not significant. These results indicate that ImproV has near equivalent operability to AE in complex tasks, but exhibits problems in simple tasks. Therefore, we analyzed the video we recorded during the tasks and found that connecting nodes, dragging from output port to input port, took too much time.

The comments from the participants on the questionnaire also support the above results. The participants gave many positive comments regarding understandability, e.g., “It is intuitive with basic knowledge of video creation” and “The nodes are easy to understand”. Most of the negative comments mentioned the difficulties in connecting nodes, e.g., “I was having trouble connecting when I inserted an effect” and “I want sliders to be pre-attached to each new node”. The comments regarding complexity of the dataflow editor of ImproV were, e.g., “The dataflow becomes too complex while VJing” and “Managing the two data types is difficult”.

We analyzed these results and decided to unify the data types and to address the connecting problem.

IV. Video Type

From the results of the user study, we unified the data types into a single type called `Video`, which is a sequence of images. The `Video` type provides simple yet flexible ways to specify parameters of effects, still enabling represents a video as the processing subject of the dataflow.

In Figure 5, the video with triangles in the upper left is rotated at the center of the frame by using the `Rotation` node and its brightness is changed by using the `Brightness` node. Note that the rotation angle and brightness to be changed are designated by the *gray scale video* input to the `ValueIn` of both the `Rotation` and `Brightness` nodes. The resulting video from the `Rotation` and `Brightness` nodes is more rotated and darkened where the gray scale video is brighter, respectively.

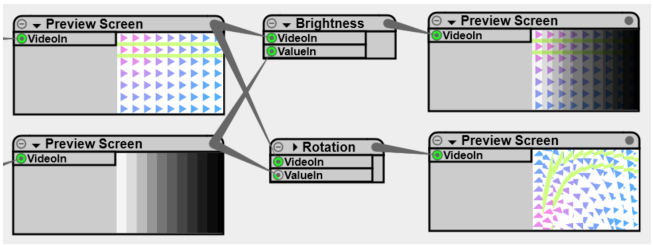


Figure 5. Setting parameters with `Video` type

The semantics is as follows. Each effect extracts a value for each pixel of each frame of the `Video`-type data and uses that value as the luminance value on the pixel position. More concretely, an effect is defined as f in Eq. (1). Out_{xy} is the pixel color of the resulting image at (x, y), n is the number of input ports, and V_n is the frame of nth input port.

$$Out_{xy} = f(x, y, V_1, V_2, V_3, \cdots, V_n) \tag{1}$$

For example, f of the `Rotation` node is defined using Eq. (2).

$$f(x, y, V_1, V_2) = V_{1_{RxRy}} \tag{2}$$

where

$$\begin{pmatrix} Rx \\ Ry \end{pmatrix} = \begin{pmatrix} \cos V_{2_{xy}} & -\sin V_{2_{xy}} \\ \sin V_{2_{xy}} & \cos V_{2_{xy}} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3}$$

Giving effect parameters with `Video`-type values enables different values for different positions on the same frame to be set simultaneously. It also enables animation of parameter values according to the video that is input as the parameter.

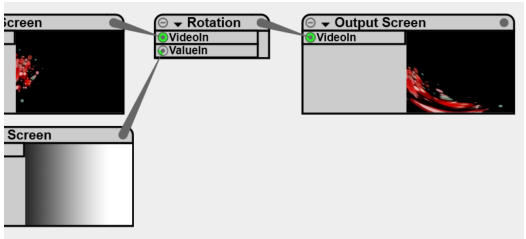


Figure 6. Dataflow of animation example

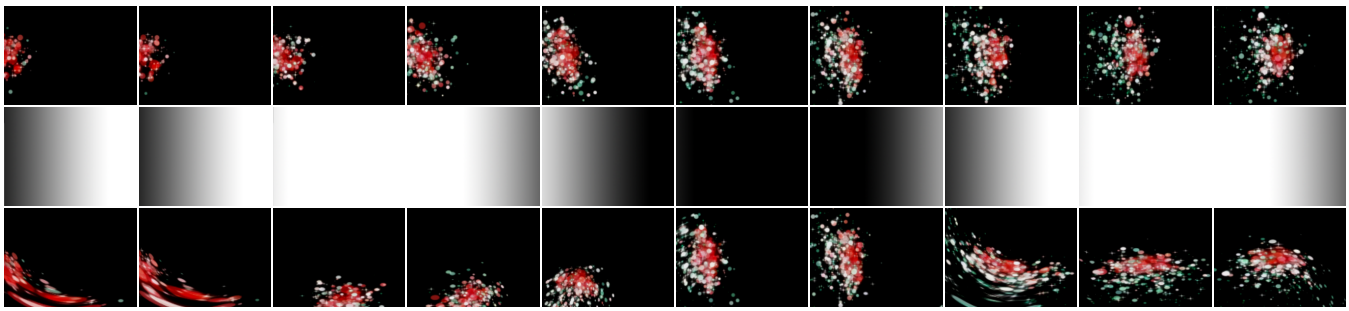


Figure 7. Example of animation

V. Examples using Video Type

In this section, we demonstrate three examples using the `Video` type for parameter animation, audio, and camera.

A. Animating Parameters

Designating a video effect parameter with a video can animate that parameter along with the input video's movement. Figure 7 shows ten frames of the input and output of the `Rotation` node shown in Figure 6. The top, middle, and bottom strips show the image sequence of `VideoIn`, `ValueIn`, and the output, respectively. In Figure 6, the rotation angle is designated by the gray scale video.

The resulting image sequence shows that the original video is modulated by the gray scale video. Although VJs still have to prepare some gray scale animated videos, those videos are versatile. Therefore, providing such videos as libraries is practical.

B. Audio Input

An `AudioIn` node captures audio signals from a sound card using PortAudio [11] and outputs a gray scale audio visualization. A VJ can designate the layout of the audio visualization by inputting `Video`-type data. An `AudioIn` node captures audio signals from the sound card then arranges the newer samples to where the pixels of the input video are brighter. Finally, it outputs the gray scale image according to the audio level.

For example, an `AudioIn` node outputs the waveform as vertical streaks since the horizontal gray scale video is input to the `AudioIn` node, as illustrated in Figure 8. Translating the horizontal color bars' y-axes according to the vertical streaks results in visualization of the waveform.

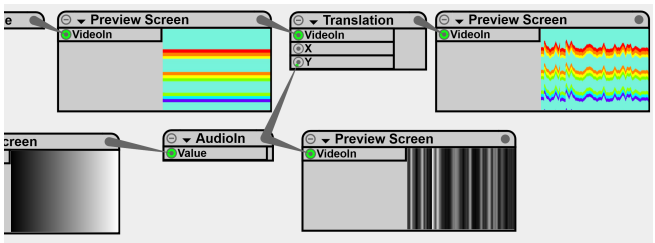


Figure 8. Example of audio visualization

Figure 9 shows a more complex example of audio visualization. By inputting circle gradation, an `AudioIn` node arranges the samples from center to outside. The output image from the `AudioIn` node results in ring streaks. The `Blur` node smoothes the streaks. The `Scale` node changes the scale of the video input in `VideoIn`. In Figure 9, the `Scale` node, by inputting the smoothed streaks to `ValueIn`, works as a ripple-like effect.

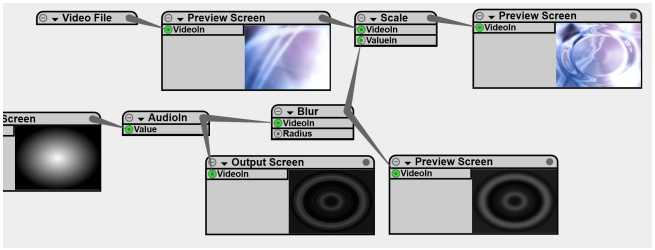


Figure 9. Example of complex audio visualization

C. Camera Input

With the `Video` type, a VJ can use a `Camera` node to control parameters. A `Camera` node captures the video from a USB camera. It was originally designed to be used to record an audience or DJs (or other performers). However, we found that using a captured video as another node's input parameter enables a VJ to use a USB camera as an instrument for controlling parameters.

The bottom left video of Figure 10 was captured using a USB camera. This video can be used as the parameter of the `Scale` node, which enlarges the original video where the parameter video is dark and shrinks the original video where the parameter video is bright. In the downer left of Figure 10, A VJ is rubbing the lens of the USB camera with his/her thumb. This produces a video similar to the animation of the gray scale video shown in Figure 6. In this way, the VJ can interactively control the area that he/she wants to enlarge by moving his/her thumb.

VI. Adjustment Knobs

A VJ can quickly set a parameter and adjust an input video value with an adjustment knob placed on each input port. Figure 11 shows what the adjustment knobs look like.

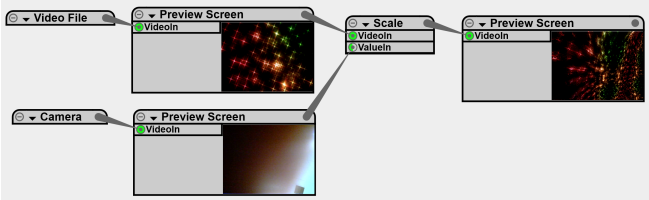


Figure 10. Controlling scale with USB camera

The `Rotation` node has two input ports, each of which is represented as double circles. The green fan between the two circles is the port's adjustment knob. The central angle of the fan indicates the value of the adjustment knob. A VJ can increase or decrease the value by dragging the mouse from the input port to up or down.

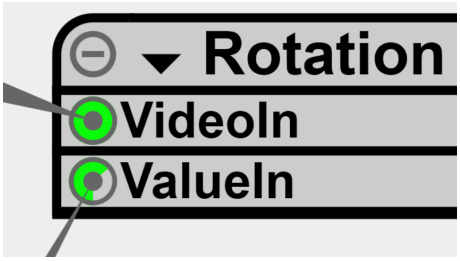


Figure 11. Adjustment knobs

An adjustment knob works differently depending on whether video input is given or not. In both cases, the value of the adjustment knob is used as opacity. If a video input is given to the input port, the port multiplies the value of its adjustment knob and the alpha channel value of the input video in each frame. In Figure 5, the rotation angle is restricted to 90° by designating the value of the adjustment knob to 25%.

If no video input is given to the input port, the port is treated as if a video of white frames is input. In this case, all pixels on the frame uniformly have the value of its adjustment knob. Figure 12 shows an example of such input ports. In this example, the `Color` node corrects the color of the input. The `Saturation` and `Brightness` are set around 50%, which means the same saturation and brightness as the original video. The `Hue` is set around 25%, which means the hue is 90°.

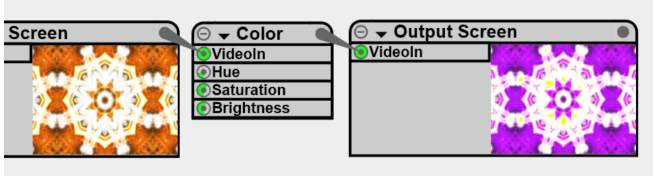


Figure 12. Correcting color with adjustment knobs

Adjustment knobs are designed as syntax sugar for the two idioms that an unconnected adjustment knob works as a `Slider` node which is a constant node in ImproV, and a connected adjustment knob works as a combination of `Slider` and `Transparency` nodes. Before the adjustment knob is developed, he/she has to create a `Slider` node and sometimes a `Transparency` node, and connect them. Creation and connection occurs many times when using nodes that have several input ports as parameters, such as the `Color` node. The same combination of `Slider` and `Transparency` nodes is also used for fading a video, which is often used in VJing.

To show the effectiveness of adjustment knobs, Figure 13 shows the corresponding dataflow of Figure 12 on an older version of ImproV. The meanings of these two dataflows are identical. The `Slider` node outputs a gray plane according to the value indicated by the slider on the node.

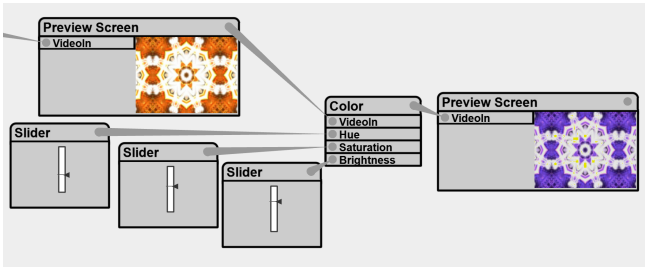


Figure 13. Correcting color with sliders in older version of ImproV

Figure 14 is a screenshot of the older version of ImproV. The transparency of the gray scale video is adjusted by a combination of `Transparency` and `Slider` nodes before the gray scale video is input to the `Rotation` node. The rotation angle is restricted by the `Slider` node as Figure 5.

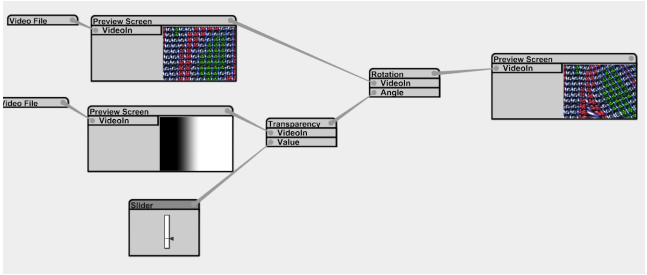


Figure 14. Adjusting video with Transparency node in older version of ImproV

The advantage of adjustment knobs is not only the reduction in the number of node creating and connecting operations, but is also the fluidness of the interaction flow, i.e., a VJ drags the mouse from an output port to an input port and then drags the mouse from the input port up or down.

VII. Discussion

In this section, we discuss the possibility of applying features of ImproV to other domains, i.e., audio and music.

It might be able to unify data types of a dataflow visual language for processing audio. An audio processing dataflow requires two data types, audio signal and parameter controlling signal. Both data types are sequences of linear values. Therefore, the abstract concept of data unification on a dataflow visual language can be applied to audio processing. However, note information in processing a musical score is difficult to represent by a sequence of linear values. We can define 0 as note-off and other values as note-on for the pitches of the values. However, such a definition may not suite a musician's intuition.

From the user interface point of view, adjustment knobs are also efficient for audio processing. Adjustment knobs in a dataflow visual language system for processing audio can be used to set linear values and adjust the value input to the input port. This reduces edge connection frequency and visual complexity. However, it depends on the characteristics of the data type. Adjustment knobs will work for many data types in a dataflow visual language system for any domain.

VIII. Related Work

There have been several dataflow visual language systems for supporting live performances. Editing dataflow during a performance is recognized as an established performance method in music. Bencina developed the audio processing system AudioMulch [3], which is based on a similar concept to that of ImproV in which the performer constructs effects on the fly. We are aiming at a similar abstraction level to AudioMulch with our system, in which professionals can understand and construct the dataflow without programming knowledge. The dataflow of AudioMulch also handles single data type, audio signal. However, parameter controlling is located in the user interface panel outside the dataflow editor. The `Video` type of our system can control parameters and be processed in the dataflow. It is also clear which knob corresponds to which parameter.

Dataflow visual language systems have been developed for VJing. Müeller et al. developed Soundium [5], a dataflow visual language system for VJing and explain the VJing method using Soundium [6]. Soundium has large function libraries and several data types which make dataflow too complicated. Parameter controlling is located outside the dataflow editor, as in AudioMulch.

ReacTable [4] by Jorda et al. uses tangible objects on a touch panel to construct an audio synthesizer's structure on the fly. VPlay [7] by Taylor et al. is a dataflow visual language system designed for multi-touch panels. Both systems have similar dataflow editing methods that bringing two nodes close together connects the nodes, and turning a node changes a parameter of the node. However, these editing methods sometimes result in an unintended dataflow structure. Moving a node can connect or disconnect edges.

Several systems, such as SPATIAL POEM [8], Rhythmism [9], and video-organ [10] for controlling the visual attributes in VJing have been developed. While our study mainly focused on STEP 1 of Lew's analysis, these studies addressed STEP 2 and can be used together with our system. Moreover, the `Video` type of our system enables VJs to use camera input to control parameters on the fly. This makes a camera a useful new instrument, which shows the flexibility of our system.

IX. Conclusions

ImproV is a dataflow visual language for VJing. We have conducted a user study for the first prototype of ImproV and found that two data types of dataflow confuse VJs and that connecting nodes is time consuming. To address these problems, we unified the data types into a single type called `Video` and developed adjustment knobs. This unification simplifies dataflow and makes various applications such as parameter animation, audio visualizing, and using cameras as controllers possible. The adjustment knobs enable quick interaction by reducing the frequency of constant node creation and connection. Dataflow is also easy to understand due to the reduction in visual complexity.

References

[1] M. Lew, "Live Cinema: designing an instrument for cinema editing as a live performance," in *NIME '04*, pp. 144–149.

[2] A. Kobayashi, B. Shizuki, and J. Tanaka, "ImproV: A system for improvisational construction of video processing flow," in *HCI International 2009 Part IV*, pp. 534–542.

[3] R. Bencina, "Oasis Rose the composition - real-time DSP with AudioMulch," in *Proceedings of the Australasian Computer Music Conference*, 1998, pp. 85–92.

[4] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner, "The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces," in *TEI '07*, pp. 139–146.

[5] P. Müeller, S. M. Arisona, S. Schubiger-Banz, and M. Specht, "Interactive media and design editing for live visuals applications," in *GRAPP '06*, pp. 232–242.

[6] P. Müeller, "Live visuals tutorial: part III," in *SIGGRAPH '07 courses*, pp. 127–151.

[7] S. Taylor, S. Izadi, D. Kirk, R. Harper, and A. G. Mendoza, "Turning the tables: an interactive surface for VJing," in *CHI '09*, pp. 1251–1254.

[8] J. Choi and S. H. Hong, "SPATIAL POEM: A new type of experimental visual interaction in 3D virtual environment," in *APCHI '08*, pp. 167–174.

[9] S. D. Tokuhisa, Y. Iwata, and M. Inakage, "Rhythmism: a VJ performance system with maracas based devices," in *ACE '07*, pp. 204–207.

[10] B. Bongers and Y. Harris, "A structured instrument design approach: the Video-Organ," in *NIME '02*, pp. 1–6.

[11] R. Bencina and P. Burk, "PortAudio - an open source cross platform audio API," in *ICMC '01*, pp. 263–266.