ABSTRACT
This paper presents MVEVE (Modular Visual Environment for Visual Effects), a visual programming environment for creating new visual effects. For intuitive visual programming, MVEVE has several high-level data types, e.g., ImageList and 3DObjectList. Processing modules in MVEVE has input and output ports of these data types, and typically performs image/video processings. Users connect input and output ports among modules, and then create new visual effects easier than conventional textual programming. Sample effects of afterimages, speedlines, and particles are presented in the paper.

KEY WORDS

1 Introduction
In recent years, video hosting services including YouTube and Niconico have become popular worldwide, and various visual effects are added to many video clips posted to those services.

Most of the added effects are, however, simple ones prepared in video cameras or movie editing softwares, and few clips have dynamic effects according to the motion of objects in the scene. Studies on semi-automatic dynamic effects have thus been conducted.

In these studies, however, each effect is fixed and all the users can usually do is to choose an effect type and then to adjust its parameters. If they want to add their original visual effects, they have to program the effects by themselves with a conventional programming language. Such textual programming is difficult and time-consuming for most users.

This paper presents MVEVE (Modular Visual Environment for Visual Effects), a visual programming environment for creating new visual effects. Assuming that any visual effect can be built by combining basic image processing units, MVEVE enables users to intuitively program visual effects with high-level data types instead of programming the effects textually.

2 Related Work
First of all, we present previous studies on semi-automatically making visual effects according to the motion of objects in the scene. Obayashi et al.[1] proposed a method which adds a motion effect to a moving object for 3D animation. The effects are given on the opposite side of the moving direction in three ways: lines, afterimages, or deformation. Goldman et al.[2] proposed a system which makes a static image expressing object’s motions with 3D arrows when the user selects the object’s domain on some frames in a video. Teramoto et al.[3] proposed a system which adds motion expressions to a static image when the user selects an object domain, its motion trajectory and a type of motion effect for the image. As presented in the first section, the visual effects in these previous researches are fixed and prepared in advance.

To enable users to create new visual effects, visual programming is a promising approach. Many
visual programming environments have already been proposed for various application areas including instrumentation[4], simulation[5], visualization[6], and media processing[7][8].

Even for video processing, many systems have appeared including Cantata[7][9], Jitter[10], VJing[11], and ImproV[12]. These systems can handle video mixing, transformation, blur, and more, and are used even for interactive art and entertainment[10][11][12]. From the viewpoint of programming, however, on these visual environments, users can hardly make motion-dependent visual effects mentioned above[1][2][3].

To mix/transform/blur videos, the program refers to a current frame only, while to represent motion in videos, the program should refer to adjacent frames1. In the existing visual environments, users do not have a convenient way to refer/represent these adjacent frames, and this is why motion-dependent visual effects can hardly be programmed in the existing systems.

To enable users to build new visual effects referring to adjacent frames, we introduce a new high-level data type called 3D Object List, a sequence of sequences of images. The use of 3D Object Lists and its results are presented in later sections.

3 System Overview

Our method for creating effects is similar to existing visual programming environments, such as LabVIEW[4], Simulink[5] and AVS[6], where the order of execution of modules is defined by data flow. A screenshot of MVEVE is shown in Figure 1. The green boxes on the screen are modules, most of which execute video processing, and the rests are for video input/output and control structures. Each module has input ports on its top and/or output ports on its bottom. The color of each port represents its data type. Red, blue, and gray correspond to RgbImageList, BitmapImageList, and GrayScaleImageList, respectively. Features of these data types are described in the next section. Each connection from an output port to an input port represents data flow, where the data type (colors) of the input port should coincide with that of the output port. As an exception, a GenericType port can be connected with a port of any data type. To create a visual effect on MVEVE, users put various modules on the window, and then make connections between ports. To modify an effect, users open its file, add/delete/move the modules and connections, and then save the edited effect.

As mentioned in Section 2, 3D Object List is one of the most important features in MVEVE. To avoid complex examples here, the use of 3D Object List is discussed in later sections.

4 Visual Language for Creating Effects

4.1 Data Types

MVEVE has several data types including ImageList, 3DObjectList, Value, Text, and Generic, and its modules have I/O ports of these data types.

4.1.1 Image List

ImageList is a sequence of images, typically an image time series. It is either RGBImageList, GSImageList, or BMImageList depending on its component image type. (GS and BM stand for Grayscale and Bitmap, respectively.) In MVEVE, the input and output are usually ImageLists as a sequence of video images, and most intermediate results are also treated as ImageLists.

If a module has multiple ImageList input ports and some of them should generate data by its initialization process, the size of the input image is determined by the other input ImageList(s).

4.1.2 3D Object List

To generate visual effects for a single frame, its adjacent frames are often required. In MVEVE, the sequence of adjacent frames/images is called 3DObject, and 3DObjectList, a sequence of 3DObjects, is also a data type.

Figure 2 illustrates how a 3DObjectList is generated from a ImageList. For example, an object is tracked and represented as ImageList (Figure 2 Left). To make an afterimage for each frame, its adjacent frames are combined.

---

1Motion is represented by a sequence of frames, not by a single frame.
The name of 3DObjectList comes from a combination of two dimensions of an image plane and a dimension of time $t$. 3DObjectList is either 3DObjectListRGB, 3DObjectListGS, or 3DObjectListBM depending on its source ImageList.

### 4.1.3 Other Data Types

Value is a scalar value (integer or real), and it is set as 0 by an initialization process. Text is a text data, and it is set as empty by an initialization process. GenericType can be connected with any port of all data types.

### 4.2 Processing Modules

MVEVE has various processing modules and they are classified into several categories as shown in Table 1. The modules and their categories are introduced below, and how they are connected is shown in the next section.

**Tracking Category.** Modules in Tracking category track an object specified by a user. ObjectTracking tracks the object and outputs its areas for subsequent processing, and TrackingText tracks the object and outputs a specified text near the object.

**Pixelwise Operation Category.** Pixelwise Operation category covers logical operations. PixelwiseSum, PixelwiseProduct, and PixelwiseDifference perform logical sum, product, and difference (subtraction) pixel by pixel, respectively. PixelwiseOverlay overpaints a (typically GS or BM) ImageList on a (typically RGB) ImageList. It is often used to add some effects represented by GS/BM ImageList to the original ImageList.

**Morphology Category.** MVEVE has morphological operators for creating various effects. Dilation dilates a binary area and Erosion erodes it, respectively, in a mathematical morphology fashion.

**Image Filter Category.** Modules in Image Filter category perform conventional image filtering operations including blurring and contour extraction.

**Data Conversion Category.** 3DSweep in Data Conversion category generates a 3DObjectList from an ImageList as shown in Figure 2. On the other hand, Projection projects all the 3DObjects in a 3DObjectList onto an image plane, and then generates an ImageList.

**Effect Category.** Modules in Effect category are substitutes of combinations of some modules. For efficiency, they are implemented as single modules. Speedline module generates speedlines, and Particle scatters particles.

**Control Structure Category.** MVEVE has two major control structures: For and Subsystem. For is a construct for repetition of fixed/predefined times, and Subsystem is a construct for making a subroutine/subprogram.

**Auxiliary Control Category.** Modules in Auxiliary Control category are used for supporting the above control structures. SubsystemIn and SubsystemOut modules function as input and output of Subsystem and For, while Memory is typically used for storing intermediate results in For.

**Input & Output Category.** VideoInput and VideoOutput are used for input and output of the whole program. ImageSeqOutput outputs a sequence of images instead of video format.

### 5 Combinations of Modules

This section shows some combinations of modules and their resultant effects (Figure 3).

#### 5.1 Afterimages

This subsection creates an effect of afterimages of object contours (Figure 3(a)). They are represented by drawing the (previous) contours on the opposite side of its motion direction. To create the effect, our process has three steps: (1) making a ripple shape around the object (Figure 4(a)), (2) making a track shape of the object depending on its motion (Figure 4(b)), and then (3) making the target effect with (1) and (2) (Figure 4(c)).

In Figure 4(a), VideoInput module receives an ImageList, and ObjectTracking module tracks a specified object in the ImageList. In the next For, Dilation and ContourExtraction modules extract multiple contours with different scales. The contours are accumulated with Memory and PixelwiseSum modules, and then the ripple shape around the object is generated.

Similarly, the object is tracked in Figure 4(b) \(^2\).

---

\(^2\)VideoInput and ObjectTracking modules in Figure 4(a) and (b) are identical in MVEVE. They are shown separately only for clarification in Figure 4.
Table 1. Processing modules and their categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracking</td>
<td>ObjectTracking</td>
<td>track a specified object</td>
</tr>
<tr>
<td></td>
<td>TrackingText</td>
<td>track a specified object and attach a text to it</td>
</tr>
<tr>
<td>Pixelwise Operation</td>
<td>PixelwiseSum</td>
<td>perform logical sum pixel by pixel</td>
</tr>
<tr>
<td></td>
<td>PixelwiseProduct</td>
<td>perform logical product pixel by pixel</td>
</tr>
<tr>
<td></td>
<td>PixelwiseDifference</td>
<td>perform logical difference pixel by pixel</td>
</tr>
<tr>
<td></td>
<td>PixelwiseOverlay</td>
<td>overlay an image onto another image</td>
</tr>
<tr>
<td>Morphology</td>
<td>Dilation</td>
<td>dilate an area</td>
</tr>
<tr>
<td></td>
<td>Erosion</td>
<td>erode an area</td>
</tr>
<tr>
<td>Image Filter</td>
<td>Blur</td>
<td>blur an image</td>
</tr>
<tr>
<td></td>
<td>ContourExtraction</td>
<td>extract contours</td>
</tr>
<tr>
<td></td>
<td>PerlinNoise</td>
<td>add Perlin noise</td>
</tr>
<tr>
<td>Data Conversion</td>
<td>3DSweep</td>
<td>generate a 3DObjectList from an ImageList</td>
</tr>
<tr>
<td></td>
<td>Projection</td>
<td>generate an ImageList from a 3DObjectList</td>
</tr>
<tr>
<td>Effect</td>
<td>Speedline</td>
<td>draw rough speedlines</td>
</tr>
<tr>
<td></td>
<td>Particle</td>
<td>scatter particles</td>
</tr>
<tr>
<td>Control Structure</td>
<td>Subsystem</td>
<td>subroutine/subprogram</td>
</tr>
<tr>
<td></td>
<td>For</td>
<td>repetition</td>
</tr>
<tr>
<td>Auxiliary Control</td>
<td>SubsystemIn</td>
<td>input of Subsystem/For</td>
</tr>
<tr>
<td></td>
<td>SubsystemOut</td>
<td>output of Subsystem/For</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>temporal image storage</td>
</tr>
<tr>
<td>Input &amp; Output</td>
<td>VideoInput</td>
<td>input video file</td>
</tr>
<tr>
<td></td>
<td>VideoOutput</td>
<td>output video file</td>
</tr>
<tr>
<td></td>
<td>ImageSeqOutput</td>
<td>output a sequence of image files</td>
</tr>
</tbody>
</table>

Figure 3. Sample effects in MVEVE

(a) Afterimages  (b) Speedlines  (c) Surrounding particles
3DSweep module then makes a 3DObjectList, and Projection module makes a sequence of the track shape.

In Figure 4(c), PixelwiseProduct module inputs the results from the above two (sub)processes, and then obtains a sequence of the afterimages. Finally, the obtained sequence is overlayed onto the original image by PixelwiseOverlay module.

### 5.2 Speedlines

In this subsection, speedlines are represented by drawing multiple lines of (mostly) fixed length in opposite direction of object motion (Figure 3(b)). To create the effect, our process has three steps: (1) making rough speedlines of the object depending on its motion (Figure 5(a)), (2) making an enlarged area of the object (Figure 5(b)), and then (3) making the target effect with (1) and (2) (Figure 5(c)).

In Figure 5(a), Speedline module generates rough speedlines of the object, and Dilation module obtains the enlarged area in Figure 5(b). In Figure 4(c), PixelwiseDifference module generates a sequence of the speedlines from the above two (sub)processes, and finally the obtained sequence is overlayed onto the original image.

### 5.3 Surrounding Particles

This subsection illustrates how to create an effect of particles surrounding a moving object (Figure 3(c)). To create the effect, our process has two steps: (1) make a probability distribution of particle positions (Figure 6(a)), and then (2) generate particles depending on the above distribution (Figure 6(b)).

In Figure 6(a), from the tracked object image and its dilated image, PixelwiseDifference module generates a donuts shape surrounding the object, and the donuts shape is further blurred by Blur module. In Figure 6(b), particles are generated depending on the probability distribution of the above blurred donuts shape, and finally the particles are overlayed onto the original image.

### 6 Implementation

We implemented MVEVE on a PC with Intel Core i7 CPU and 24GB RAM using Microsoft Visual C++ with OpenCV and OpenGL libraries.

### 7 Results

In Figure 7, two types of visual effects are shown on a video where a man is running. The faster the man runs, the more contours appear in Figure 7(a) and the longer the speedlines get in Figure 7(b). Such speed-dependent changes of effects make a video clip lively. The speedlines in Figure 7(b) are different from those in Figures 3(b) and 5. They are made of many particles following the object (running...
Figure 5. Module arrangements for speedlines

(a) Process(1) (b) Process(2) (c) Whole process

As shown in these examples, MVEVE enables users to create various effects by changing the arrangements of modules.

8 Conclusion

This paper presents MVEVE (Modular Visual Environment for Visual Effects), a visual programming environment for creating new visual effects. With MVEVE, even naive users can intuitively program visual effects with high-level data types instead of programming the effects textually. In particular, a high-level new data type, 3D Object List, provides a convenient way to refer to adjacent frames for representing motions.

Our future work includes: (1) preparing a richer set of modules for creating more innovative effects, (2) improving the processing speed with, for example, GPU programming, and (3) providing preview functions for interactive use.

References


Figure 7. Results

(a) Afterimages

(b) Speedlines (Following particles)


