筑波大学大学院博士課程

システム情報工学研究科修士論文

ライブ映像パフォーマンスのための
即興的映像合成システム

小林 敦友

(コンピュータサイエンス専攻)

指導教員　田中 二郎

2009年3月

概要

音楽イベントやファッションショーなどのイベントにおいて映像が提示され、その映像の制御を人間がパフォーマンスとして行う、ライブ映像パフォーマンスのためのシステム、ImproVの開発を行ってきた。本論文では、ライブ映像パフォーマンスのツールに求められる機能やユーザインタフェース、ImproV の詳細、及びその評価、そしてそれらの考察について述べる。

ライブ映像パフォーマンスの作業フローは、その準備段階の映像素材制作と、本番のライブ映像パフォーマンスに分けることが出来る。準備段階にてライブ映像パフォーマンスの演者は、イベントの性格をイベント主催者やイベント出演者と打ち合わせ、そこに演者自身の主観を加味し映像素材の制作を行う。制作で行われる具体的な制作作業は、実写映像の撮影、アニメーションの制作、映像の合成、編集などである。ライブ映像パフォーマンスの最中では、イベントが行われる会場の雰囲気、その時の音楽、観客の様子等に応じて映像の切り替えが行われる。この切り替えによって、ライブ映像パフォーマンスが途切れることがなく、イベント全体の雰囲気に沿ったものとなりえる。これを実現するため演者は、観客に映像を提示している間に、次に提示する映像のための映像素材選別や簡単な映像の加工等をプレビューしながら行い、元の映像と切り替える。

ImproV は、ライブ映像パフォーマンスの演者にとって分かりやすい水準のデータフロー図によって映像合成処理の流れを表現する。また、映像再生中にもその図を複数個編集可能にすることで、提示中の映像とは別の映像を容易に合成し、それらのプレビューや切り替えを可能にした。これにより演者は、従来は準備段階の制作の一部であった映像の合成を、ライブ映像パフォーマンスの最中に、提示中の映像に影響を及ぼすことなく試行錯誤を行いながら、即興的に行える。

被験者実験では、ライブ映像パフォーマンスの演者にとってデータフローによる表現は分かりやすいことや、操作性について改善すべきであることが分かった。

目 次

図目次

表 目 次

第1章　はじめに

音楽イベントやファッションショーなどのイベントにおいて映像が提示されることがあり、それらのイベントの性格によりその映像が人間によってリアルタイムに制御されることがある。特にダンスクラブやディスコなどで開かれる、Disc Jokey（DJ）の音楽パフォーマンスにて行われる事が多く、そのようなパフォーマンスやパフォーマンスを行う者を一般に、Visual Jockey（VJ）と呼ぶ場合があるが、本稿では VJ を含め、観客に提示する映像をリアルタイムに制御する事を「ライブ映像パフォーマンス」と呼び、ライブ映像パフォーマンスを行う者を「演者」と呼ぶ。

ライブ映像パフォーマンスの歴史や分類については Spinrad による [Spi05] が詳しい。また、Lew の [Lew04] や、特に VJ については Mueller による [MÖ7] にも短く述べられている。

ライブ映像パフォーマンスを行うにあたって、演者は、準備段階であらかじめ映像素材を制作し、ライブ映像パフォーマンスの最中にそれらを切り替えながら提示してゆく。

既存のライブ映像パフォーマンスでは、演者はその場の雰囲気に応じて映像を合成し切り替えていく。演者が意図しない映像が観客に見えてしまうのを防ぐため、演者は観客にある映像を提示している間に、次に提示する映像のための映像素材選別やエフェクト適用、合成結果のプレビュー等を行い、それらを切り替える。

このようなライブ映像パフォーマンスのためのシステムをソフトウェアで実現するには既存 の映像制作用アプリケーションとは異なるユーザインタフェースが必要となる。我々は、ライブ映像パフォーマンスに適したアプリケーションのユーザインタフェースの設計と実装を目的に、映像合成システム, ImproV の開発を行ってきた [小林 08b, 小林 08a, KST09]。

1.1　ライブ映像パフォーマンスにおける作業

まず、背景としてライブ映像パフォーマンスで行われる作業について述べる。ライブ映像パフォーマンスの最中では、イベントが行われる会場の雰囲気、その時の音楽、観客の様子等に応じて映像の切り替えが行われる。これを実現するため演者は、観客に映像を提示している間に、次に提示する映像のための映像素材選別や簡単な映像の加工、加工結果の確認等を行い、元の映像と切り替える。この切り替えによって、ライブ映像パフォーマンスが途切れることがなく、イベント全体の雰囲気に沿ったものとなりえる。

Lew は [Lew04] にて、DJ の作業手順を 3 段階に分解し、ライブ映像パフォーマンスに適用しようと試みた。その 3 段階の DJ の作業手順を以下に示す。

手順 1：メディア検索 レコードを選択する。

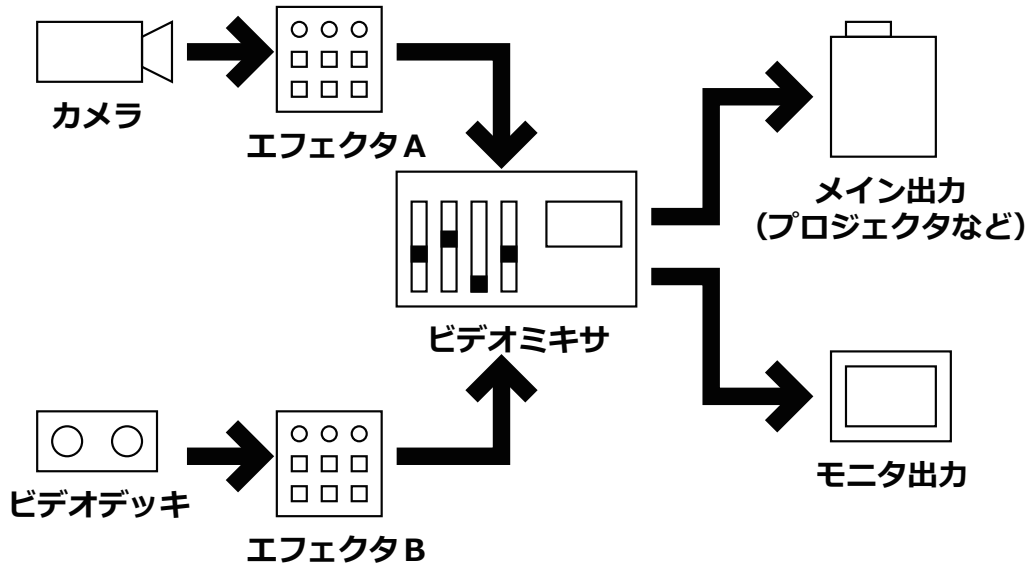


図 1.1: ライブ映像パフォーマンスのシステム構成例

手順 2：プレビューと調整 ヘッドホンでレコードをプレビューし、目的の曲やサンプルを見つけ、再生速度、フィルタ、エフェクトの調整を行う。

手順 3：ライブ操作 「手順 2：プレビューと調整」で調整された素材を、再生されている既存のストリームに組み入れ、スクラッチ、カット、逆再生によってその素材を操作する。

DJ パフォーマンスにおいて音楽が途切れず、音楽の切り替わりが自然であることが要求されることと同様に、ライブ映像パフォーマンスにおいては映像が途切れないことと、映像の切り替わりが自然であることが要求される。つまり上記の手順は、レコードを映像素材に、ヘッドホンをモニタ出力にそれぞれ置き換えるとライブ映像パフォーマンスの手順としても利用できる。事実、motion dive [Incd] や V-4[Cor] など、この手順を想定した多くのシステムや機材が存在する 。この手順を実現するため演者は、図 1.1 に示す様なビデオミキサを中心としたシステム構成を使う。

ビデオミキサは観客に提示するためのメイン出力とプレビュー用のモニタ出力を備えている。演者は、ある映像がメイン出力にて観客に提示されている間に、次に提示する映像素材選別やその映像素材に対するエフェクト適用、調整をモニタ出力でプレビューしながら行う。その後映像を切り替えたり、エフェクトのパラメータを操作する。これらを繰り返すことによって、その場で次々と映像を切り替えていく。このことから、ライブ映像パフォーマンスには、映像素材やそれにエフェクトを加えた映像経路が複数、それら複数の映像経路を切り替るビデオミキサ、及びプレビュー用のモニタ出力が必要であることがわかる。

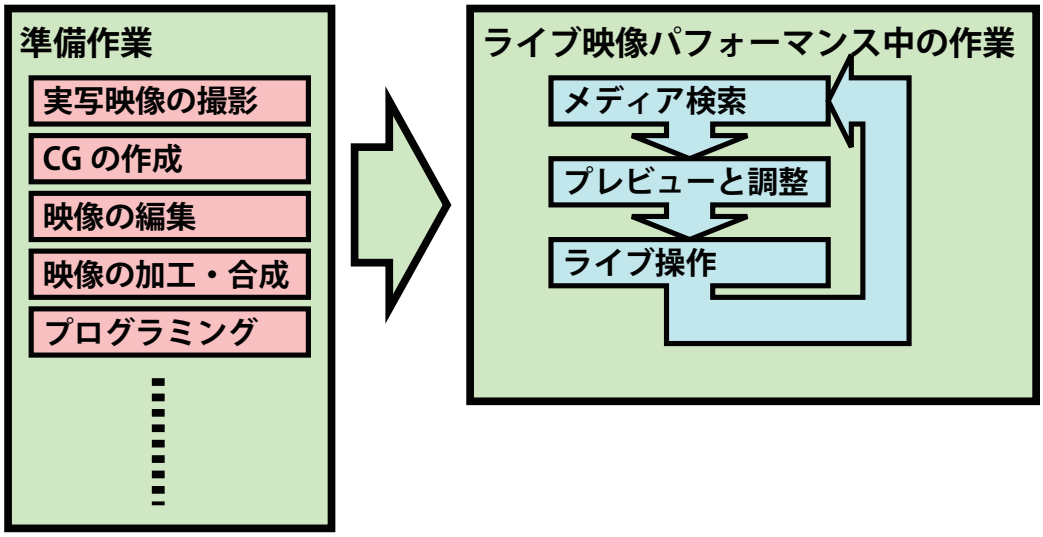


図 1.2: 準備作業を含めたライブ映像パフォーマンスの作業の流れ

1.1.1　ビデオミキサを使ったシステム構成

1.1 節で述べた、ビデオミキサを使ったシステム構成では、ライブ映像パフォーマンス中にシステム構成を変更する事ができない。例えば図 1.1 に示したシステム構成を使ってライブ映像パフォーマンスを行っている最中には、ビデオデッキからの映像にエフェクタ A を適用するように配線を繋ぎ換えることは出来ない。そのような繋ぎ換えを行うにはメインアウトプットへの出力を一旦止める必要があるからである。これは演者がライブ映像パフォーマンスの最中に手を加える余地を少なくしており、ライブ映像パフォーマンスに求められる即興性を制限している。

1.2　ライブ映像パフォーマンス前の準備作業

上記の作業手順はライブ映像パフォーマンス中の作業手順である。しかし、ライブ映像パフォーマンスを行う前には準備が必要である。

素材となる映像の撮影や作成、編集や合成、機材の設定等の準備を行う。演者によっては自作のコンピュータプログラムのみによってライブ映像パフォーマンスを行うものもいるが、これも事前にプログラミングを行う必要がある。多くのツールでは、これらの制作作業はライブ映像パフォーマンス前に行うものであり、ライブ映像パフォーマンスの最中の作業とは分けられている。このライブ映像パフォーマンス前の準備作業とライブ映像パフォーマンス中の作業を図式化したものを図 1.2 に示す。

1.3 本研究でのアプローチ

我々は、映像合成を行うユーザの脳内モデルはデータフロー（DF）型のモデルであると仮定し、アプリケーション上で DF 図によって映像処理の流れを表し、その DF 図の編集を映像再生中であっても行えるようにした。

DF 図の編集を映像再生中であっても行えるようにすることにより、演者は任意のシステム構成を構築でき、ライブ映像パフォーマンスの最中であってもそのシステム構成を変更できる。これは上で述べた、準備作業である映像の合成や機材の設置といった作業の一部を、「手順 2：プレビューと調整」で行えるようにするものである。さらに、従来は準備作業としてしか行えなかった作業をライブ映像パフォーマンスの最中に行えるようにすることで、演者が演者の発想をその場で試すことを支援し、演者の即興性を高める。

ライブ映像パフォーマンス中の作業の 3 つの作業のうち「手順 1：メディア検索」や「「手順 3：ライブ操作」については深く取り扱わず、既存のシステムとの連携を図れるようにした。それぞれについて、以下に述べる。

1.3.1 メディア検索

メディア検索はライブパフォーマンス以外のシステムにおいても盛んに研究される分野である。映像ファイルに関してはサムネイル表示やタグ付、及びメタデータ等の手法が利用できる。

本システムでは映像ファイル、一部のエフェクトファイルのドラッグアンドドロップによる読み込みを実装した。これにより上で述べた作業手順の「手順 1：メディア検索」については別のアプリケーションを利用することが可能である。例えば、Windows のファイルエキスプローラを使ってサムネイル表示が利用できることや、Adobe Bridge を使ってメタデータによる検索、サムネイル表示及び、プレビューを利用できることから、これらのシステムから直接ドラッグアンドドロップすることが考えられる。図 1.3 は Adobe Bridge によってサムネイル表示及び、プレビューを行っているところである。

1.3.2 ライブ操作

ライブ操作のためのインタフェースでは音楽の分野のシステムを参考にすることが出来る。電子楽器の制御のための通信規格として、Musical Instrument Digital Interface(MIDI) や Open Sound Control（OSC)[MW97] といった規格がある。これらにより、演者がライブ操作を行うハードウェアコントローラと、実際に発音するシンセサイザや実際に音声処理を行うエフェクタを分け、演者は自分が慣れたハードウェアコントローラを使いつつも、多様な音色を演奏することが可能である。ハードウェアコントローラの例として、UC-33e[Tec] を図 1.4 に示す。

ライブ操作のためのハードウェアコントローラは様々なものが開発されており、それらのハードウェアの多様性を吸収するために、ハードウェアコントローラ上のノブ、スライダ、ボ

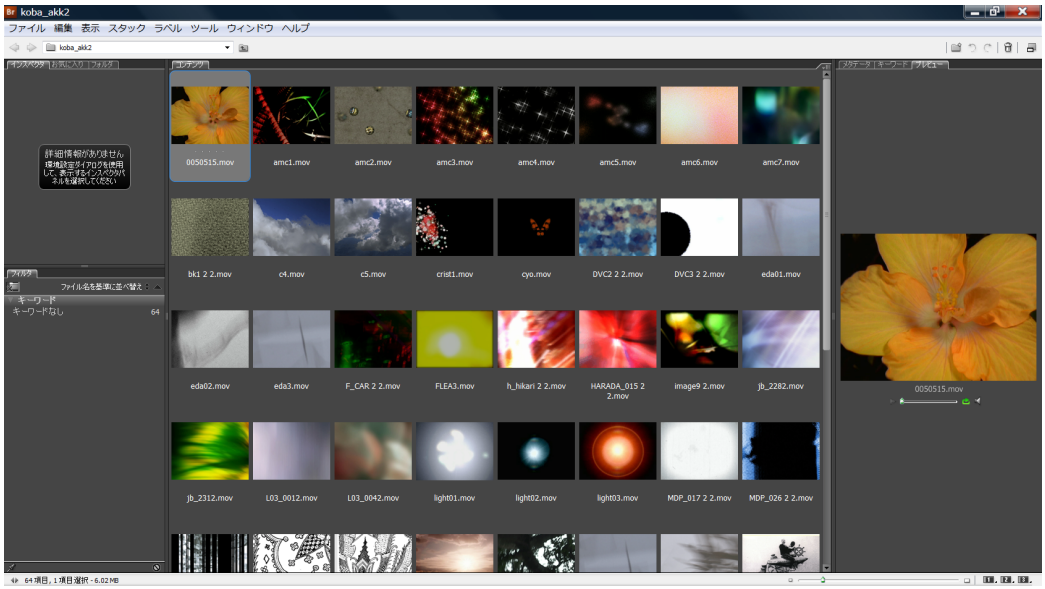


図 1.3: Adobe Bridge によるサムネイル表示及び、プレビュー

タン等の部品はソフトウェア上、もしくは別のハードウェアに組み込まれたシンセサイザやエフェクタ等の各パラメータに好きに割り当てることが出来る [Cro08]。

この割り当てや操作を実現するために、多くの音楽用ソフトウェアは、物理的なコントローラから送られる MIDI 信号とソフトウェア上のパラメータとのマッピングを行う機能を備えている。この機能は MIDI マッピングと呼ばれる。さらにこのマッピングの指定を簡潔にするための MIDI ラーニングと呼ばれる手法がある。以下に MIDI ラーニングにおけるユーザの手順を述べる。

1. GUI 上で「MIDI ラーニング」ボタンをクリックすると MIDI ラーニングモードに入る。
2. GUI 上でマッピングしたいパラメータを表す部分（多くの場合スライダーやノブ）を選択する。
3. 物理的なコントローラ上のマッピングしたいノブやスライダ等の部品を操作する。
4. GUI 上で「MIDI ラーニング」ボタンを再度クリックすると MIDI ラーニングモードを終わる。

本システムでもこれを参考に物理的なコントローラとのマッピング機能を実装した。これにより上で述べた作業手順の「手順 3：ライブ操作」については MIDI に対応した物理コントローラであればどのようなものでも対応することが可能である。

図 1.4: MIDI コントローラの例

1.4 本研究の貢献

本研究の目的はライブ映像パフォーマンスの最中に映像合成を行えるようにすることで、従来より即興性の高いライブ映像パフォーマンスを実現することである。

本研究では、ライブ映像パフォーマンスのシステムにおいて、DF に基づくユーザインターフェースを採用しすることにより、ライブ映像パフォーマンスの最中に映像合成を行い、その結果をプレビューし、映像を切り替える事が可能になった事を示す。さらにその過程において、ライブ映像パフォーマンスを行う演者にとって理解しやすい水準の DF 図を設計した。

また、DF 図の理解や DF 図編集の操作性について、ライブ映像パフォーマンスの経験者を対象に被験者実験を行った。そこで得られた実験結果、及びコメントも本研究の貢献である。

第2章　関連研究

本研究はライブ映像パフォーマンスの映像処理システムにDF型ユーザインタフェースを導入する試みである。映像処理のためのDF型システムは多くあるが、ライブ映像パフォーマンスの最中に操作するためのユーザインタフェースとして利用された例は少ない。

また、ライブ映像パフォーマンスの観点から見ると、本研究は、従来準備段階で行われていた映像合成作業や、器材構成の構築作業を、ライブ映像パフォーマンスの最中に行うことによって、ライブ映像パフォーマンスの即興性を向上を図るものである。

以下に映像を扱うDF型のシステムと、ライブ映像パフォーマンスに関する研究について述べる。

2.1　DF型のシステム

DF図とはデータの処理と流れを表す図であり、データの処理をノード、データの流れをエッジとした有向グラフとして表わされる。特に信号処理用のビジュアルプログラミング（VP）言語においてよく使われ[JHM04]、VP言語の特性として特定の分野に特化したシステムにおいて特に有効である。例えば、可視化の分野に特化したものとして、AVS/Express[Inca]やHilsによるDataVis[Hil91]などが挙げられる。

また、プログラミング以外の分野でも、視覚的わかり易さや、試行錯誤におけるインタラクティブ性によりエンドユーザ向けのシステムに利用されることがある。飯崎らによるFind Flow[HSMT06]はデータベース検索のためのインタフェースである。対象分野が全く異なるが、ユーザの試行錯誤を支援する点や、DFの途中をプレビュー出来るようにする点など本研究と類似している。

Max/MSP[Cyc]は、音楽、音声処理用VP言語である。もともとは楽譜情報を扱うMaxと、そのプラグインとして音声信号処理用のMSPが開発され、シンセサイザーや音声エフェクタの制作や作曲、メディアアート等に使用される。

REAKTOR[Insa]も音声処理のVP言語であるが、DF図における各ノードの抽象度が、「オシレータ」や「フィルタ」等、シンセサイザや音声エフェクタを扱うミュージシャンに分かる水準であり、一方で複雑なプログラミングは出来なかった。近年はREAKTOR Core[Insb]と呼ばれる、低水準のノードが追加され、MSPに近いプログラミング言語になりつつある。ImproVは、REAKTORと同様に高水準のノードを用意し、ライブ映像パフォーマンスの演者にとって、理解しやすい事を目指す。

映像を扱うDF型のシステムとしてはJitter[’74]が挙げられる。Max/MSP[Cyc]のプラグイ

ンで映像を扱えるもので、ライブ映像パフォーマンスで使われる例もあるが、再生中（実行中）の DF 構造の変更は想定されておらず、あらかじめ作成したプログラムを使う事しかできない。

映像の再生中に DF 構造の変更を行うことが出来るシステムとして、vvvv[vvv] や Quartz Composer[Incb] 等が挙げられる。しかし、Jitter を含むこれらの DF 型システムは、VP 環境として作られており自由度の高いプログラミングが可能な反面、ユーザには高度な知識を要求する。

DF 型のシステムの操作性に関する研究として、平井らの音声入力によるビジュアルプログラミング環境操作支援インタフェース [平井 08] が挙げられる。これは音声によって、Max/MSP の操作の一部を行うものであるが、これは制作効率の向上を目的とした物であり、ライブ映像パフォーマンスの最中に操作を行う物では無い。

2.2 ライブパフォーマンス用のシステム

ライブ映像パフォーマンス用のシステムに計算機を使おうという試みとしては、福地らの EffecTV[FSE04] がある。これはコンピュータをエフェクタとして使おうというものであるが、一台のコンピュータを一台のエフェクタとして使うのでそのシステム構成はハードウェアによる結線を前提としている。また、幾つかの商用ソフトウェアも発表されており、代表的なものとして、motion dive[Incd] があるが、これらのシステム構成は固定されておりエンドユーザが自由に変更できるようにはなっていない。

より自由なシステム構成を行えるとして、上に述べた Jitter、Quartz Composer、vvvv 等を使ってライブ映像パフォーマンスを行う者もいるが、これらについても、準備段階においてあらかじめプログラムを制作し、ライブ映像パフォーマンス最中ではその制作したプログラムを操作するという方法で使用される。

TouchDesigner[Incc] は、インタラクティブな 3D アニメーション制作を対象とした VP 言語を備えるが、これも、ライブ映像パフォーマンスで使用される際には、制作したプログラムを TouchPlayer という実行環境で実行する。

Visual Jockey[Des] は本システムと同じく。ライブ映像パフォーマンスにおいて使うことを目標とした DF 型システムである。しかし、DF 図の編集に関しては、準備作業の段階で行うことを想定されており、リアルタイムに DF を編集するということに対して、ユーザインタフェースとして配慮されていない。

Mueller らは Soundium[MASBS06] にて、ライブ映像パフォーマンス最中に、マルチメディア制作システム Soundium を使用することについて具体的に述べている。また、Arisona は [Ari07] にて、映像制作とライブ映像パフォーマンスの分離、及び、ライブ映像パフォーマンスの最中に制作の作業を Soundium を使って行うことについて述べている。本研究も、ライブ映像パフォーマンスの最中に制作の作業を行うことを試みるものである。しかし、Soundium はもともと汎用的なマルチメディア制作のためのシステムであり、上記の映像を扱う DF 型のシステム同様、ユーザに高度な知識を要求する。また、ライブ映像パフォーマンスの最中に

プログラミングを行える機構を備えているがテキストのプログラミング言語である。

ライブ映像パフォーマンスの最中の作業については、Lew による [Lew04] にて、DJ の手順を分析し、その手順をライブ映像パフォーマンスに適用することが試みられている。

ライブ操作に関する研究は殆どが音楽パフォーマンスの分野でなされている。ライブ映像パフォーマンスに関する物としては、Bongers らによる Video-Organ[BH02] や、徳久らによる Rhythmism[dTII07]、Nervixxx[Tok08] などがあげられる。Video-Organ では、音楽のシステムに用いられるライブ操作のためのハードウェアコントローラをその操作部品ごとに分解し、それらを映像の様々なパラメータにマッピングすることが試みられている。Rhythmism はマラカスにセンサーを取り付け、ユーザはそのマラカスを振ることによって映像を操作する。これらは、上で述べた、3 段階の DJ の作業手順での「手順 3：ライブ操作」を対象としており、「手順 2：プレビューと調整」を対象とした本システムとは異なる。しかし、本システムで「手順 2：プレビューと調整」を行い、これらの操作インタフェースによって「手順 3：ライブ操作」を行うといった共存も可能である。

第3章　ImproVのユーザインタフェース

ImproV では、映像処理方法が DF 図として表現される。DF 図上で扱うデータは映像と、パラメータとしての実数値である。

ImproV のユーザは、ライブ映像パフォーマンスを行う演者を想定しており、そのユーザにとって閾の低い UI を目指している。

3.1　映像機器の接続を模した UI 設計

ユーザは映像機材とその接続方法に精通しており、映像機材をノードとした DF 図を容易に理解できることが期待できる。そのため、各ノードがそれぞれ映像機材をひとつづつ表現するように配慮し、映像ファイル再生、実数値を指定するスライダー、出力画面、各種エフェクタやミキサ等のノードを実装した。これは、Andrew らによる 6 つの学習障壁 [KMA04] のうち、Design Barriers を既に乗り越えたユーザに対して、Selection Barriers を乗り越える支援を行うものである。Design Barriers とは、ユーザがプログラミング学習の際、設計を行うことを学ぶのが難しいという障壁である。ImproV のユーザは、自分の行いたい映像合成のプロセスを考えることができると仮定している。一方、Selection Barriers とは、ユーザがプログラミング学習の際に、設計は行えるがそれを実現するために何を使えばよいかを学ぶことが難しいという障壁である。これを下げるため、十分な映像に関する知識を持った、ImproV のユーザに必要以上の知識を追加で学習する必要がないよう、なるべく配慮している。

実数値を処理するノードを導入し、複雑な計算を行うことも可能である。しかし、Weis らの報告 [WKU+07] によると、プログラミング経験のないユーザには数式の作成が困難であり、ImproV では導入していない。

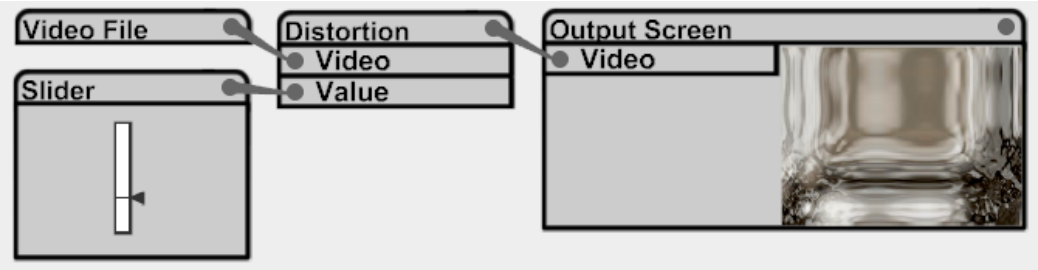


図 3.1: ImproV の DF 図の例

3.2 DF 図の文法

エッジは、映像値か、エフェクト等のパラメータのための実数値のどちらかの流れを表現している。全てのノードにはノード名と、その右に出力を表す円（出力円）が表示されている。入力を受け付けるノードは、ノード名の下に入力データの型名と、その左に入力を表す円（入力円）が表示される。図 3.1 中央の「Distortion」は映像値と実数値の入力をそれぞれ一つづつ持つノードの例である。また、図 3.1 の「Slider」や「Output Screen」のように、固有の UI を持つノードも存在する。エッジの視覚デザインは、データの流れの方向が分かるように、また一部が隠れていてもその方向が分かるように、飯崎らの Find Flo w [HSMT06] と同様の、出力円から入力円へ細くなっていく形を採用した。図 3.1 では、「Video File」に画像を歪ませる「Distortion」が適用され、「Output Screen」により表示されている。「Distortion」の歪み具合は「Slider」によって指定されている。

ImproV では、出力はいくつにでも分岐させる事ができ、その分岐では映像値や実数値のコピーが行われる。入力は各入力円につき一つづつしか受け付けない。合流に際しては、映像値同士の場合は、4.3.8.2 目にて述べる「Mixer」等のノードを使って、その合成方法を指定する必要がある。映像値と実数値の合流も、上記「Distortion」のようなエフェクタへの映像入力とそのパラメータの指定として、ノードを介する事によって行われる。実数値同士の合流についても、数値演算ノードとして実装することが考えられるが、3.1 節で述べた理由により、ImproV には導入していない。

3.3 DF 図の編集操作

ノードの生成は「Create」メニューから行う。またそれ以外にも、映像ファイルやシェーダファイルを Windows 上からドラッグアンドドロップすることにより自動的に適切なノードに変換し配置される。配置されたノードのノード名部分をドラッグするとノードを移動させる事が出来る。

各ノードの出力円上でマウスダウンを行いそのままドラッグすると、マウスカーソルに向かってエッジが伸びる。そのまま、他のノードの入力円までマウスカーソルを移動させ、その入力円上でマウスアップを行うと結線が行われる。

3.4 ライブ映像パフォーマンス向けの機能

ライブ映像パフォーマンス中に DF 図を編集するにあたって、ユーザは DF 図内の各ノードで何が起こっているのか把握する必要がある。また、ライブ映像パフォーマンスでは、観客に提示している映像とは別の映像を観客に見えないところで用意し、切り替えるといった手法がよくとられる。このためライブ映像パフォーマンスではプレビュー機能が重要になる。ImproV ではノードの出力部分、もしくはエッジ上へのマウスオーバーにより、それらを流れる映像がマウスカーソルの右下に表示される。図 3.2 中央では「Color」からの出力が表示され

ている。マウス操作しながらのプレビューも行う必要がある。そのようなときには、「Output Screen」に DF 上の任意のノードから結線することにより、そのノードの出力をプレビューできる。図 3.2 にこれらの様子を示す。

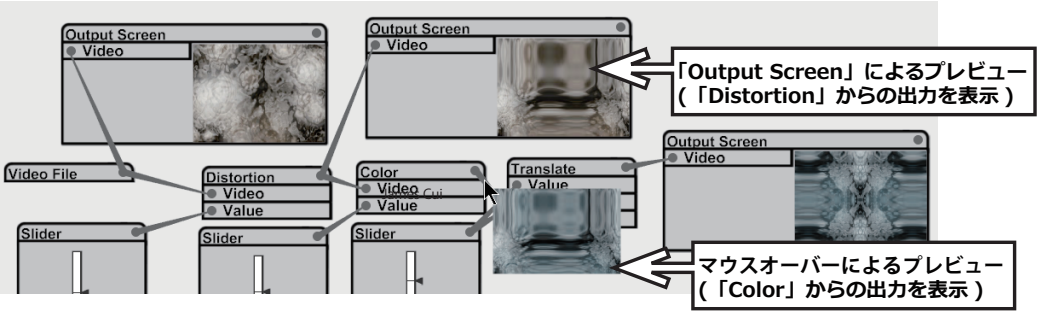


図 3.2: プレビューの機構

また、すでに結線済みの入力に対して、別の出力からのエッジを結線しようとすると、元々あったエッジは削除され新しいエッジが結線される。これにより、エッジを一旦削除する必要がなくなり、上で述べたプレビュー機能と組み合わせることで、観客に提示中の映像にさらにエフェクトを適用したり、別の映像と合成したりすることが可能である。

3.5 適用例

ImproV の典型的な使われ方について説明する。

3.5.1 設置

観客に提示する画面に加えて、ユーザ、つまりライブ映像パフォーマンスの演者が ImproV を操作するための画面が必要である。そのため、ImproV はデュアルスクリーンに対応したコンピュータ上で動かされる必要がある。ここでは、観客に提示する画面が一つの場合の設置について説明する。

一方のディスプレイ出力は通常のディスプレイを接続し、ユーザが DF 図を編集するために用いる。もう一方のディスプレイ出力はプロジェクターや大型ディスプレイに接続し、観客への提示用とする。ImproV には画面出力を行うノードとして「Output Screen」が用意されている。「Output Screen」をクリックすると「Output Screen」に入力された映像を表示するウィンドウが開く。このウィンドウを観客への提示用ディスプレイ出力にてフルスクリーン表示する。

図 3.3 ではノート PC 上で ImproV を動かし、液晶ディスプレイに「Output Screen」の表示を表示している。

図 3.3: ノート PC 上で ImproV を動かし、液晶ディスプレイに「Output Screen」の表示を表示しているところ

3.5.2　映像の切り替え

DF 図の編集はユーザの意図しない結果を引き起こすことがあるため、ImproV では観客に提示している映像には影響を与えずに DF 図の編集を行えるようにした。演者は観客に提示している映像を構成する DF とは別の DF をプレビューを行いながら作成し、その DF が満足いくものになってから観客に提示している映像と切り替える。

典型的な切り替え方法であるフェードインは、透明度を変更するエフェクト、Transparency とアルファ合成を行う Mixer によって可能である。以下にその手順を示す。図 3.4-A に示す状態では右上の Output Screen に表示されている映像が 観客へ提示されている。演者は左下の Video File によって再生される映像にフェードインしようとしている。

- 図 3.4-A: 演者は左下の Video File に透明度を変更するエフェクト、Transparency を繋ぐ。
- 図 3.4-B: 演者は Mixer を右上の Output Screen の前に挿入する。

- 図 3.4-C: 演者は Transparency によって一旦、左下の Video File からの映像を透明にし、Mixer によって重ねる。この時点で右上の Output Screen には左下の Video File の映像が重なっているが、透明なので見えない。
- 図 3.4-D: 演者が透明度を徐々に戻すと結果として、左下の Video File の映像がフェードインされる。

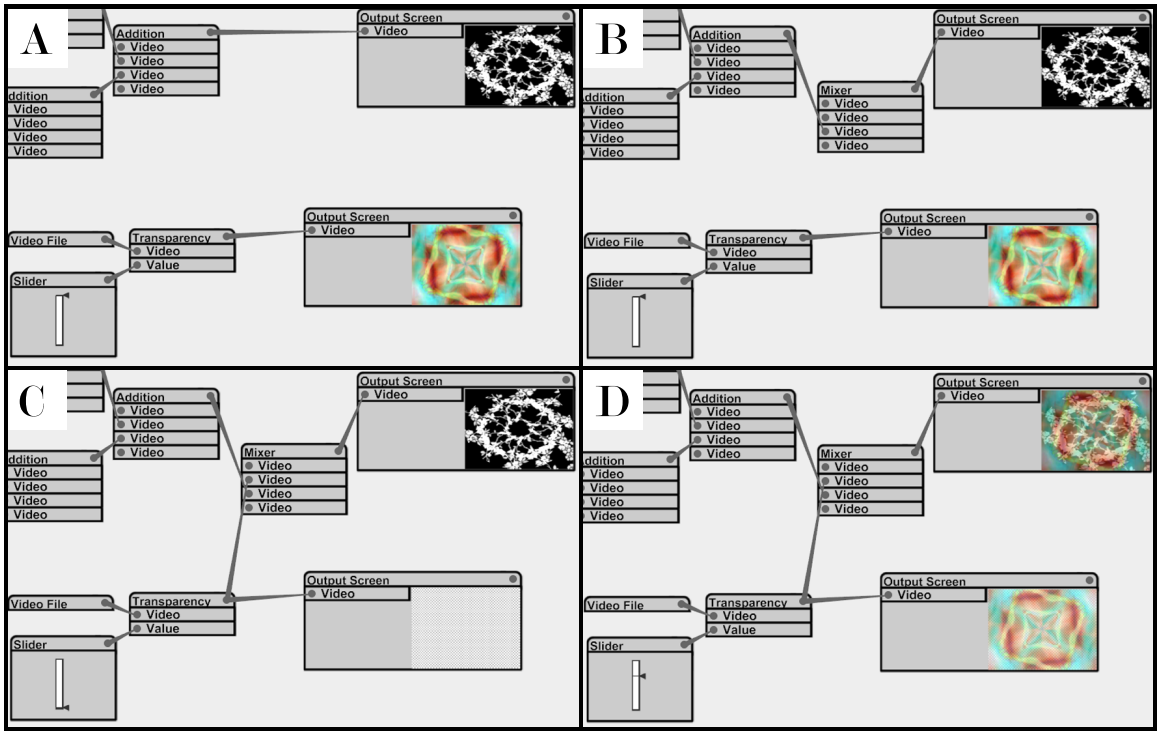


図 3.4: フェードインを行っているところ

第4章　ImproVの実装

ImproV の実装言語は C# と High Level Shader Language (HLSL)2.0 である。ImproV の DF 上の各ノードでは DirectX を使って映像が処理される。この処理は、ユーザが DF 図を編集している最中にも止まることなく行われる。

ImproV 本体の実装はシンプルに作られており、DF 図編集以外の機能はノードによって実装されている。

本章では ImproV の各ノードで行われる映像処理、DF 図の評価処理について説明し、現在実装されているノードについて述べた後、ノードの拡張について説明する。

4.1　映像処理

ImproV の映像処理は、映像のフレームを DirectX のテクスチャとして用意し、それぞれのノードの処理結果は再びテクスチャにレンダリングすることで行われる。これはノードの入出力を同じ形式で扱うことで処理の内容を DF 図で表現しやすくすることと、処理内容を GPU 上で行うことで高速化できることを期待したためである。扱う映像の形式は 640px X 480px のサイズで秒間 30 フレームとした。

ImproV 上のほとんどのエフェクタノードでは図 4.1 に示すように、四角形の平面を板状のポリゴンとして持っており、そのポリゴンに、入力されたテクスチャを張った物をピクセルシェーダへ渡すことで処理を行う。

ノードの種類に依っては、テクスチャの端を超えた座標を参照、もしくは表示することがある。例えば拡大縮小を行うノードを考えた場合、映像のサイズは 640px X 480px で固定されているので、縮小時は元の映像の範囲外の表示を行わなければならない。このような場合の処理方法が DirectX ではいくつか用意されているが、本システムでは元のテクスチャとその鏡像を交互に繰り返し敷き詰める処理を、全てのノードで採用している。

実数値は 0.0〜1.0 の float である。これは DirectX でこのデータ形式が多くつかわれる事による。例えば、DirectX でのテクスチャ画像はピクセル座標ではなく、テクスチャの幅、もしくは高さを 1.0 とした UV 座標で表される。

上記以外にも、ノードの種類によっては特殊なライブラリを使った処理が行われることがある。例えば、映像ファイルの読み込みを行うノード、Video File では DirectShow を使って映像ファイルの読み込み、再生が行われている。

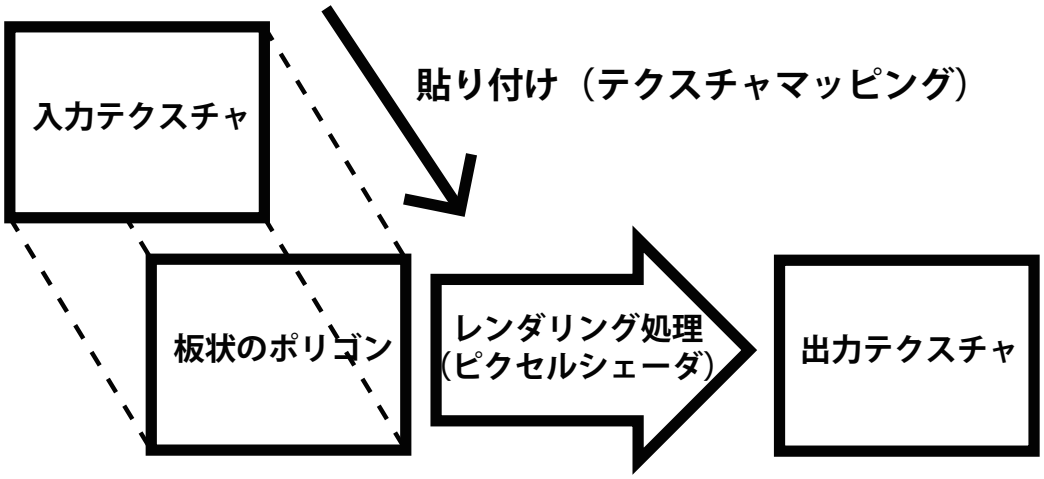


図 4.1: 各ノードで行われる典型的な映像処理

4.2 実行中リアルタイムに編集可能な DF 図

ImproV では、プログラムを実行しながらユーザがリアルタイムに DF 図を編集できることを実現するため、DF 図の評価は毎フレームごと常に行われる。しかし、ノードを生成し、それらを結線していくという作業の途中では、必ずまだ結線されていないノードや未完成の DF が発生する。

また、複数の映像ファイルプレーヤやカメラなどの映像ソースを配置したとき、それらのフレームレートが異なる場合や、フレームレートが同じでも再生が同期されない場合はそれら映像ソースのフレーム更新ごとに評価処理が発生してしまい、処理回数が多くなってしまう。

このため、DF 図の評価は映像表示のためのノードである、「Output Screen」からの要求駆動にてフレーム毎に行われることとした。出力が結線されていないノードは評価されない。結線がされていない入力に対しては、それらにデフォルト値を持たせてある。デフォルト値は、映像値の場合は透明な画像であり、実数値の場合は 0.0 である。またこれらの値は各ノードによってオーバーライドすることも可能である。

また、映像ソースのノードは Output Screen からの要求があった時点でのフレームを返すだけであり、個々の映像ソースのフレーム再描画にも影響されない。

また、ImproV では、DF が循環を構成することを許している。これは例えば「ディレイ」等の映像エフェクトにおいて、フィードバックを実現するためである。しかし、ImproV はフレーム毎に評価を行いその結果を表示するので、循環構成をもった DF をそのまま評価すると、評価が終わらず止まってしまう。これに対しては、各ノードの評価が終わるとその結果をキャッシュし、既にキャッシュされたフレーム番号での要求に対して再評価しない実装にすることにより解決した。

4.3 実装したノード

ImproV に実装したノードを以下に述べる。

4.3.1 Video File

Video File は映像ファイルを読み込みループ再生するノードである。読み込めるファイル形式は、AVI や WMV などに対応している。しかし、利用できるコーデックは、DirectShow を使っているため OS 環境に依存する。

このノードは他のノードと違い、「Create」メニューからは生成できず、「File」メニューの「Import Video」を選択することで生成する。「Import Video」を選択すると、ファイル選択ダイアログが開き、映像ファイルを選択すると、Video File がノードとして DF 図上に生成され、その映像ファイルを読み込みループ再生を開始する。また、他のアプリケーションから ImproV の DF 図上へ、映像ファイルをドラッグドロップすることでも生成することが可能である。

4.3.2 Camera

Camera はコンピュータに接続されたカメラからの映像を取得するノードである。筆者と同じ研究室の佐藤、村田による iCamera ライブラリを使っており、USB カメラと Artray のカメラに対応している。

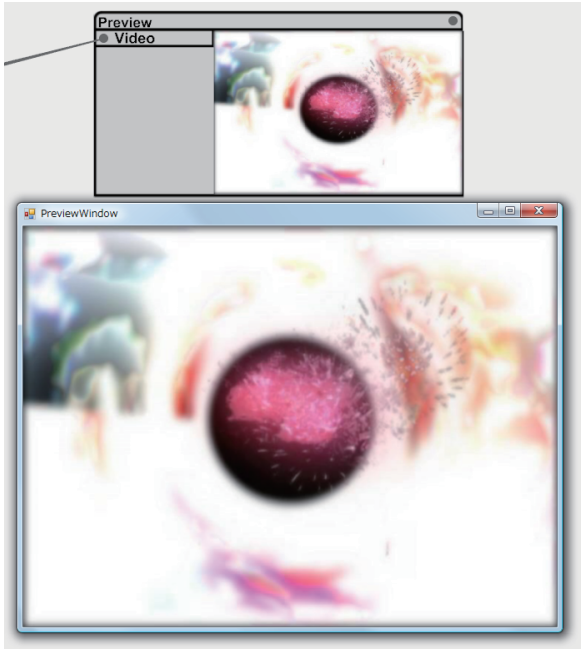


図 4.2: Output Screen と Output Screen によって開かれるウィンドウ

4.3.3 Output Screen

Output Screen は入力された映像を画面出力するノードである。ノード上に映像を表示する GUI を持っておりそこに入力された映像が表示される (図 4.2)。また、映像を表示する GUI をクリックすると、新しいウィンドウが開きそこに入力された映像が表示される。開かれる新しいウィンドウは観客への提示用を想定しているので二つの特徴をもつ。一つは別のディスプレイ上でフルスクリーン表示することが可能であることと、もう一つはマウスカーソルを表示しないことである。

4.3.4 Slider

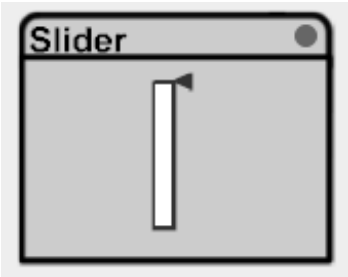


図 4.3: Slider

Slider は実数値を出力するノードであり、図 4.3 に示すように、スライダを配置した GUI を持っている。ユーザは、三角形で示されるインジケータをドラッグすることで、出力する値を変更できる。

4.3.5 MIDI In

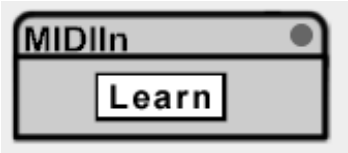


図 4.4: MIDI In

MIDI In は MIDI 信号を受信し、実数値として出力するノードである。「Learn」と書かれたボタンを配置した GUI を持っている (図 4.4)。MIDI 信号は 0 から 127 までの整数値を取りうるが MIDI In ノードはそれを 0.0 から 1.0 実数値へ変換する。受信する Control Change(CC) 番号の指定は MIDI ラーニングに基づく手法を採用した。以下にその手順をのべる。

図 4.5: 左：MIDI ラーニングモード中の MIDI In　中央：MIDI ラーニングモード中に MIDI 信号を受信したところ　右：MIDI 信号を受信したところ

1. ユーザが MIDI In 上の「Learn」と書かれたボタンをクリックするとその MIDI In が MIDI ラーニングモードに入り、Learn ボタンが赤くなってそのことを知らせる (図 4.5 左)。
2. ユーザが物理コントローラのマッピングしたい部分を操作すると、MIDI 信号が送信され Learn ボタンが点滅しそのことを知らせる (図 4.5 中央)。
3. ユーザが再度 Learn ボタンを押すとその MIDI In は MIDI ラーニングモードを終わり、Learn ボタンが白に戻ってそのことを知らせる (図 4.4)。この MIDI In は MIDI ラーニングモード中の最後に受信した CC 番号にマッピングされる。
4. MIDI In はマッピングされた CC 番号の信号を受信すると緑色に点滅し、そのことを知らせる (図 4.5 右)。

このノードにより、ImproV 内のエフェクタのパラメータを物理コントローラにマッピングすることができ、ライブ操作を可能にしている。

4.3.6 Delay

Delay は入力された映像を最大 2 秒遅らせて出力する。遅らせる時間は入力された実数値で指定される。内部では 60 フレームのバッファを持ち、そこに映像を蓄えている。

Delay を通した映像と元の映像を合成する、あるいは Delay の出力をフィードバックさせ、そのフィードバックの経路上に別のエフェクトを配置することにより残像のような効果が得られる。図 4.6 では Delay の出力の映像を Transparency によって透明度を変更し、Scale、Rotation によって拡大、回転させ、Addition により元の映像と加算合成させている。

4.3.7 Blur

Blur は入力された映像をぼかす。ぼかす量は入力された Value の実数値で指定される。Blur では、指定された Value の値 0.0〜1.0 を 0 ピクセル〜映像の幅と高さの最大としてぼかす。0.0 では全くぼかしがかからず、1.0 ではフレーム全ての距離をぼかす。一般的に、ぼかしは指定された範囲のピクセル値を平均することで実装されるが、この Blur ではリアルタイムに処理

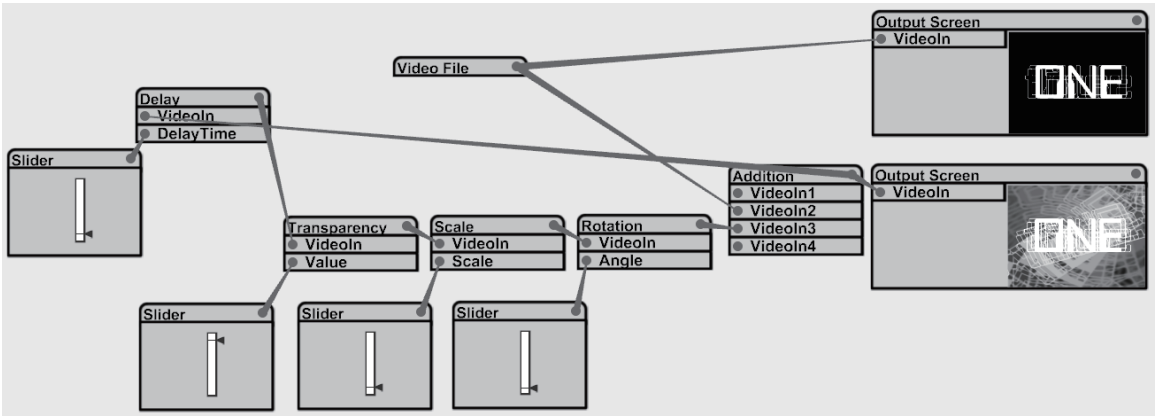


図 4.6: Delay の適用例

を行うことを優先するため、ピクセル値のサンプリング回数が固定されている。そのため正確なぼかしとはなっていない。

図 4.7 に Blur の Value をそれぞれ変化させた例を示す。一番上の Output Screen は元の映像を表示しており、それ以下の Output Screen は、それぞれスライダが示すとおり、約 0.05、約 0.3、1.0 の実数を指定したときの結果を表示している。

4.3.8 Effect

Effect は HLSL で書かれたシェーダによる処理を行うラッパーノードである。そのためこのノード自体を DF 上に配置することはできない。ImproV は起動時に特定のフォルダにある.fx の拡張子を持つシェーダファイルを調べ、ノード生成メニューに追加する。ユーザはそのメニューから選択するか、シェーダファイルをドラッグアンドドロップすることでそのシェーダファイルをノードとして DF 上に配置出来る。Effect は指定されたシェーダファイルをコンパイルし、シェーダファイル内の入力変数を調べノードの入力に変換し、ノードを追加する。入力変数の型は ImproV の映像値として HLSL の texture 、ImproV の実数値として HLSL の float がそれぞれ対応している。以下にシェーダファイルで実装されたノードを示す。

4.3.8.1 Transparency

Transparency は入力された映像の不透明度を変更し出力する。不透明度は入力された Value の実数値によって完全な透明と完全な不透明の範囲で指定される。

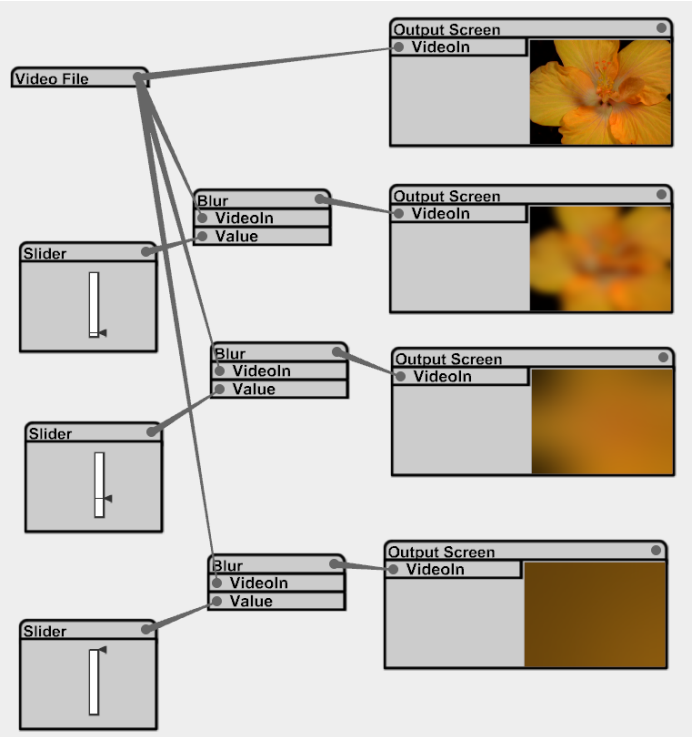


図 4.7: Blur の適用例

4.3.8.2 Mixer

Mixer は映像値の入力を 4 つ持っており、入力された映像をレイヤー状に重ねて合成する。下のレイヤーに配置された映像は上の映像にアルファ値を透明度として反映して上書きされる。このような合成方法はアルファ合成と呼ばれる。

Mixer は、4.3.8.1 目の Transparency と組み合わせることでフェードインを行う事や、アルファ値を持った映像を複数重ねる事に使う。

図 4.8 上から 3 つ目の Output Screen に表示されているフローでは Transparency と組み合わせており、一番下の Output Screen に表示されているフローでは 4.3.8.9 目にて述べる、Luminance Mask を使って暗い部分を透明にした映像を重ねている。

DirectX はアルファ合成をサポートしているが、主にゲームで使うことを目的として実装されているため、一度だけの合成しか想定されていない。そのため DirectX のアルファ合成では合成の結果が透明な場合、黒色として合成される（式 4.1）。

$$ResultColor = UnderneathColor \times (1 - TopAlpha) + TopColor \times TopAlpha \quad (4.1)$$

ImproV では Mixer の出力を別の Mixer の入力として使うこと、つまり、アルファ合成を複数回行うことがあるため別の計算方法が必要である。このため、HLSL を使って式 4.2 を実装

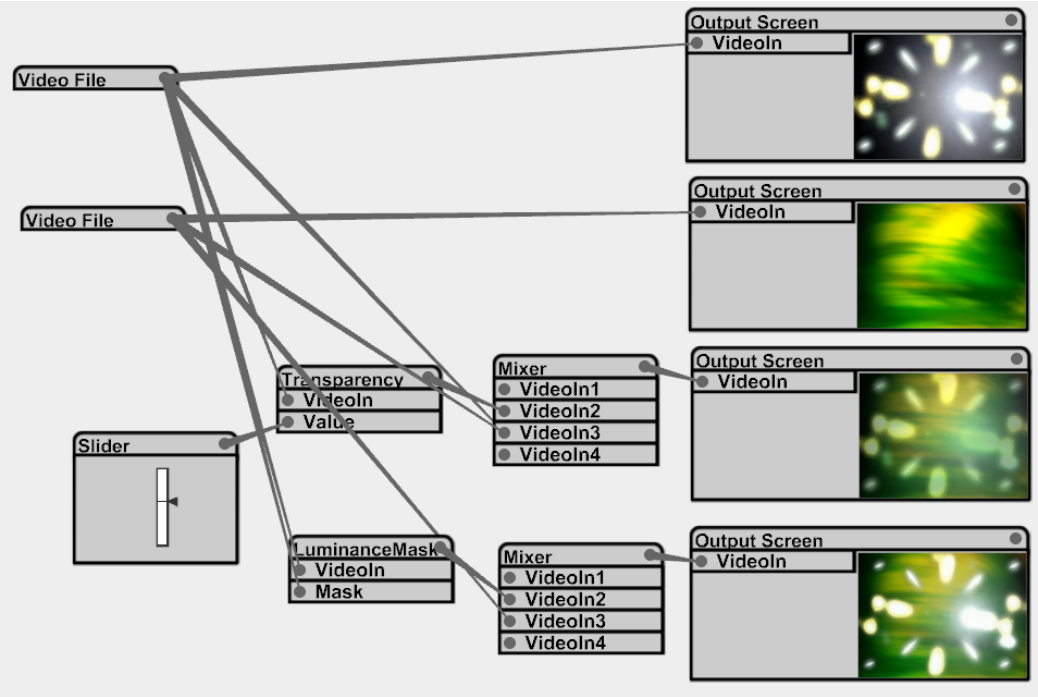


図 4.8: Mixer の適用例

した。

$$ResultColor = \frac{(UnderneathColor \times UnderneathAlpha) + (1 - UnderneathAlpha) \times TopColor \times TopAlpha}{(UnderneathAlpha + TopAlpha - UnderneathAlpha \times TopAlpha)} \tag{4.2}$$

4.3.8.3 Addition

Addition は映像値の入力を 4 つ持っており、入力映像の加算合成を行う（図 4.9）。

4.3.8.4 Screen

Screen は映像値の入力を 4 つ持っており、入力映像のスクリーン合成を行う。スクリーン合成は加算合成に似た効果を持つが、加算合成より暗い合成結果になる（図 4.9）。

4.3.8.5 Scale

Scale は入力された映像値を映像平面上で拡大、もしくは縮小させる（図 4.10）。拡大、縮小の大きさは入力された Scale の実数値によって指定される。

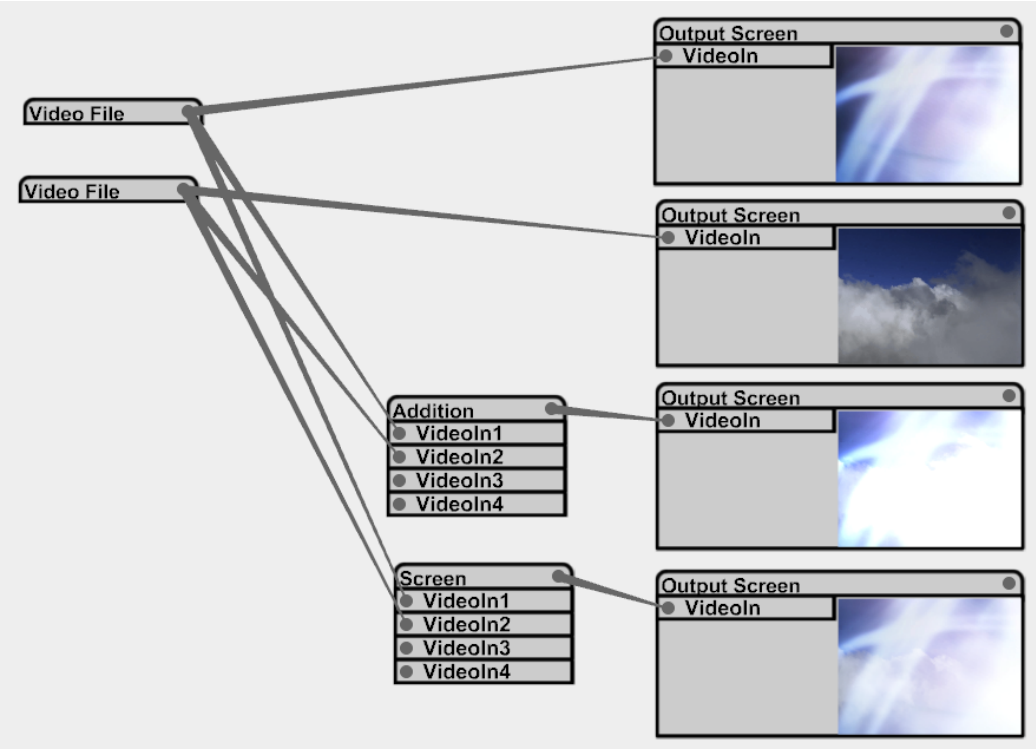


図 4.9: Addition と Screen の適用例

4.3.8.6 Rotation

Rotation は入力された映像値を回転させる（図 4.10）。回転角は入力された Angle の実数値によって、0〜360 °の範囲で指定される。

4.3.8.7 Translate

Translate は入力された映像値を映像平面上で平行移動させる（図 4.10）。移動量は入力された X、Y の実数値それぞれによって、UV 座標系の U、V が指定される。

4.3.8.8 Alpha Mask

Alpha Mask は映像値の入力を 2 つ持ち、Mask に入力された映像値のアルファ成分を VideoIn に入力された映像値のアルファ成分に書き込み出力する。これは Mask の映像の透明部分を VideoIn の映像でも透明にする。

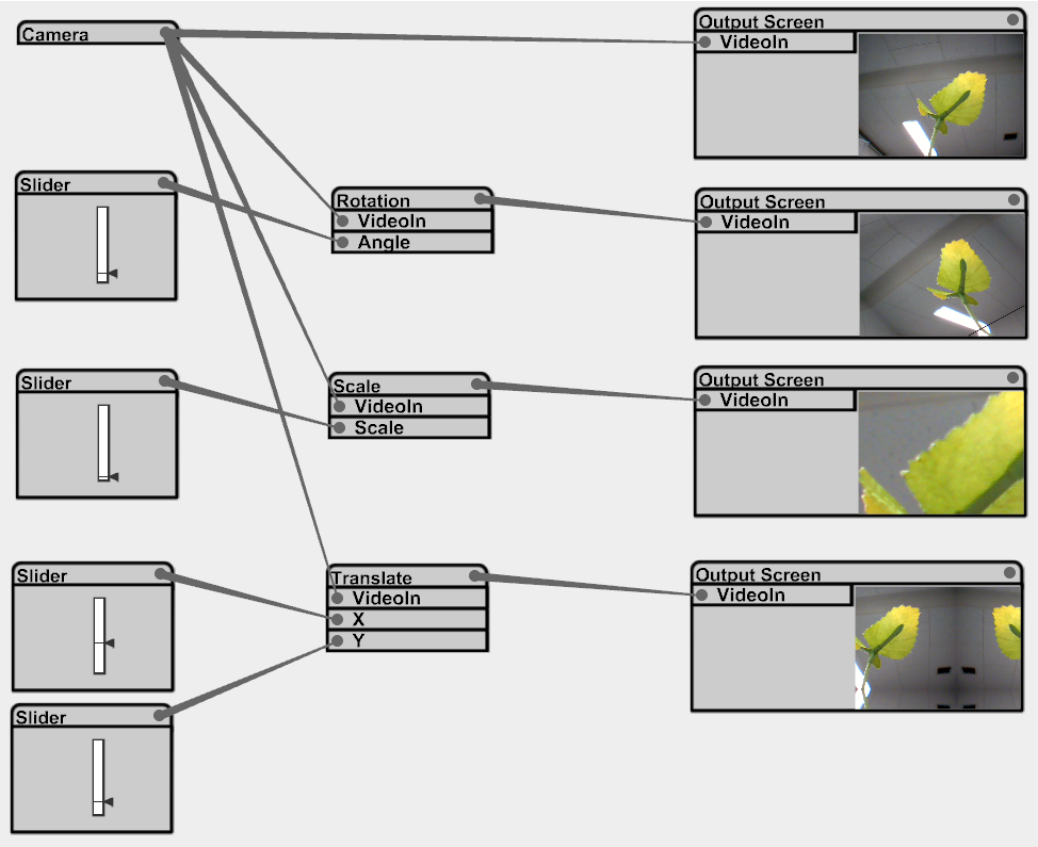


図 4.10: Rotation、Translate、及び Scale の適用例

4.3.8.9 Luminance Mask

Luminance Mask は映像値の入力を 2 つ持っている。Mask に入力された映像値の輝度を計算し、その輝度を VideoIn に入力された映像値のアルファ成分に書き込み出力する。図 4.11 に示すように Mask の映像の暗い部分を VideoIn の映像にて透明にする。

4.3.8.10 Luminance Mat

Luminance Mat は入力された映像値の輝度を計算し、その輝度を反転させたものを元の映像のアルファ成分に書き込む。その際アルファ成分は、Threasiold で指定された範囲に正規化され、Offset で指定された値が減算される。

図 4.11 に示すように、Luminance Mat によって元の映像の明るい部分を透明にすることが出来る。これは主に Mixer と同時に使い、複数の映像を重ねる効果に使用されることを狙っている。

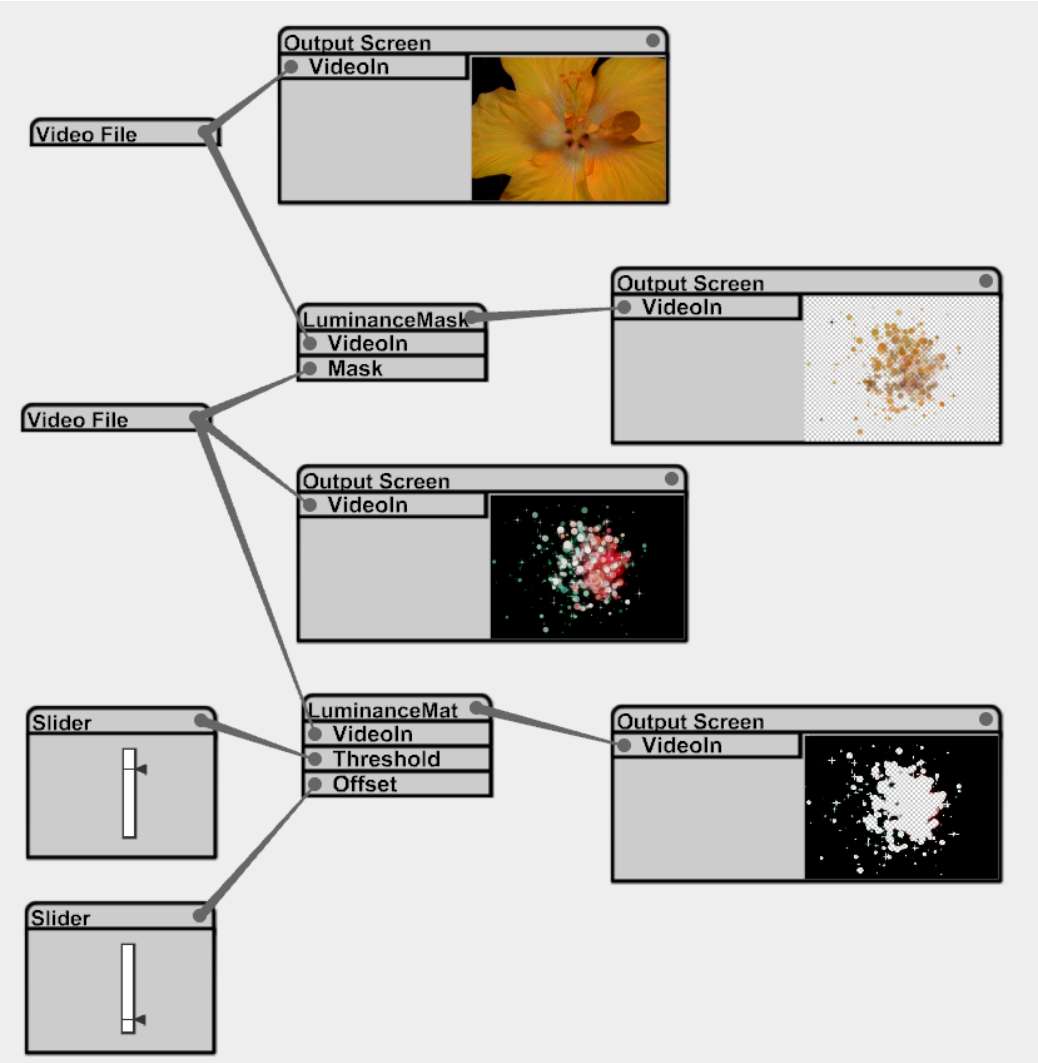


図 4.11: Luminance Mat 及び、Luminance Mask の適用例

4.3.8.11 Color

Color は入力された映像値の色を変える。色相、彩度、明るさの変化量は入力された Hue、Satuation、Brightness の実数値によってそれぞれ指定される。Hue は-180～180 °の範囲で色相を回転させる。Satuation は元の彩度を 0.0 から 2.0 倍する。明るさは 0.0 から 1.0 の範囲で加算される。

4.3.8.12 Distortion

Distortion は入力された映像値を歪める。歪み量は入力された実数値によって指定される。歪みの形は現在の実装では固定されている。図 3.1 では Distortion が適用されている。

4.4 ノードの拡張

ImproV はグラフィックプログラミングの知識があれば、プラグインとしてノードを作成できるようになっている。二通りの作成方法があり、一つは上で述べた HLSL によって記述したシェーダファイルを Effect により読み込む方法と、もう一つは Node クラスを継承した.Net アセンブリを作り、ImproV 起動時に読み込む方法である。Effect による方法は HLSL の知識のみで、特殊な環境も必要なく比較的に手軽に作成できるが、使える機能は HLSL の機能の一部に限られる。一方で Node クラスを継承する方法では.Net の機能を全て使えるが.Net の開発環境やそれらの知識が必要となる。

4.4.1 Effect による拡張

Effect による拡張は HLSL の Pixel Shader を使って行う。HLSL の Pixel Shader は、2 次元座標を引数とし、その座標の色を返す関数として書く。この関数内では引数以外に、グローバル領域の変数が扱える。ImproV ではこれらのグローバル変数の定義のうち、texture 型と float 型の変数を入力変数として解釈する。HLSL 内に複数のパスを用意し、DirectX 上で実行させることが可能であるが、これは実行順などを HLSL 外部に記述しなければならなくなるので Effect ではサポートしていない。

4.4.2 Node クラスを継承する

Node クラスを継承し.Net アセンブリを作成することでノードを作成できる。ImproV は起動時に特定フォルダの dll ファイルを読み込み、その中に Node クラスを継承したクラスがあればそれをノード生成メニューに追加する。図 4.12 に関連するクラスの関係をクラス図で示す。

ImproV はデータ型として、映像値と実数値を扱う。Node はこれらのうちどちらかを出力しなければならない。

これらの出力は、映像値を出力する Node の場合は Texture draw(int currentFrame) 関数、実数値を出力する Node の場合は float getValue(int currentFrame) 関数によって行われる。これらの関数はノードの評価時に呼び出され、引数の currentFrame は現在のフレームを表す。プラグイン開発者はこれらの関数をオーバーライドすることによって Node の処理を記述する。

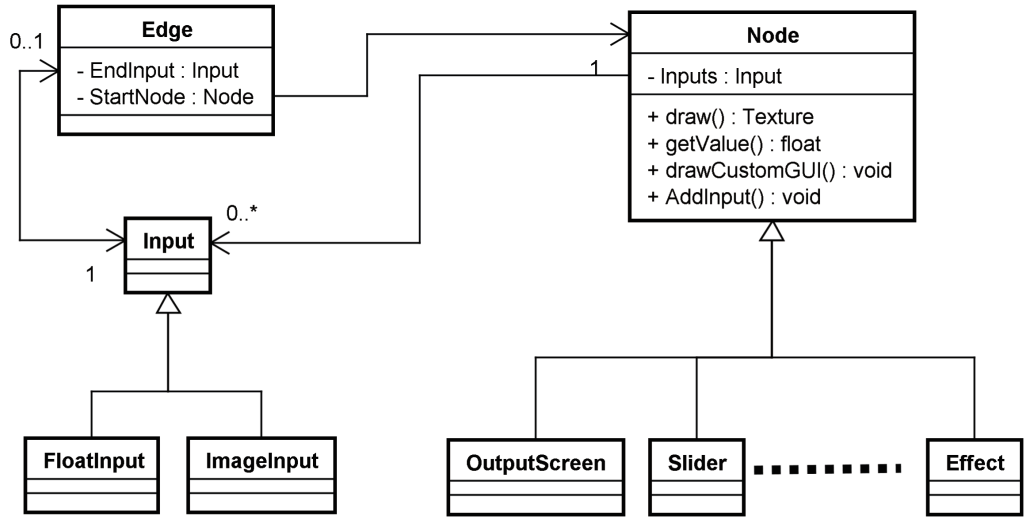


図 4.12: Node、Input、Edge の関係を表すクラス図

4.4.2.1 入力の追加

ノードへの入力を表すクラスとして Input クラスが用意されている。また、ImproV ではデータ型として映像値と実数値の 2 種類が用意されており、これらのデータ型に対応する Input クラスのサブクラスとして ImageInput クラスと FloatInput クラスがそれぞれ用意されている。これらの入力をノードに追加するには、ノードのコンストラクタで AddInput 関数を呼ぶ。

draw 関数や getValue 関数内で入力の値を使うにはその Input インスタンスの Texture Draw(int currentFrame) を呼ぶ。

4.4.2.2 独自 GUI の実装

ImproV のノードはそれぞれノード独自の GUI を持つことが出来る。Output Screen や Slider 等が独自の GUI を持つノードの例である。独自 GUI を実装するには、drawCustomGUI 関数をオーバーライドし、その関数内で GUI の描画を行う。その時、描画を行う範囲を CustomGUIRectangle で指定すれば、ImproV がノードの描画時にその範囲を含めたノードを描画する。そのほか、マウスインタラクションのための各種イベントハンドラも提供される。

第5章　評価

ImproV の開発初期に予備実験として、筆者がライブ映像パフォーマンスにて ImproV を使用し、実際に ImproV を使ってライブ映像パフォーマンスを行うことができるか確かめた。そこで得られた知見をもとに ImproV に改良を加えた後に、ライブ映像パフォーマンスの経験者 5 人による被験者実験を行った。

5.1　予備実験

予備実験では、筆者が実際に 2 時間程度のライブ映像パフォーマンスを行った。図 5.1 にその様子を示す。

イベント内容は音楽イベントで、ジャズバンドが演奏するという形式であった。機材として、Intel Core 2 Duo 2.6GHz の CPU、2GB の RAM、512MB のビデオメモリ NVIDIA GeForce 8800 GTS のグラフィックチップを持ったグラフィックカードを搭載したデスクトップコンピュータに USB カメラ、マウス、キーボード、ImproV 操作用に液晶ディスプレイ、観客への提示用にプロジェクタを接続し、使用した。筆者は、固定された USB カメラによって、ジャズバンドの演者らを撮影しており、そのカメラからの映像とあらかじめ用意しておいた映像素材を加工し組み合わせたものを提示した。この予備実験において、図 5.1 のような複雑な機能による映像が即興的に作成できること、また、そのような複雑な映像同士を複数重ねたり、切り替えられることが確認できた。

また、この時点での ImproV は一部の機能しか実装されていなかった。そのほかの機能はこの実験から得られた知見に基づいている。

Output Screen によるプレビューだけでは、いちいち Output Screen を生成し結線しなければならず、操作性が低かった。これに取り組むために、3.2 節で述べた、マウスオーバーによるプレビューを実装した。

また、ライブ映像パフォーマンスが進むにつれ DF 図が煩雑になることも挙げられる。とくにエッジの方向が把握しづらいため、3.2 節で述べた視覚デザインに改良した。

5.2　被験者実験

ライブ映像パフォーマンスの演者にとってデータフローが理解し易いことを確かめること、及び ImproV の操作性評価を目的とした被験者実験を行った。被験者はライブ映像パフォーマ

図 5.1: 実際に行われたライブ映像パフォーマンスの様子

ンスの経験者であり、Adobe After Effects の使用経験がある、22〜33 歳の男性 5 名をボランティアで募った。

被験者にはまず、映像オーサリング、及びライブ映像パフォーマンスの経験に関する、インタビューを交えた事前アンケートに答えてもらい、実験 1 を行ってもらう。その後、ImproV について説明した後、実験 2 を行ってもらい、事後アンケートに答えてもらった。

実験は、Intel Core 2 Duo 2.6GHz の CPU、4GB の RAM、256MB のビデオメモリと NVIDIA Quadro FX 570M のグラフィックチップを持ったグラフィックカードを搭載したノートパソコンで行われた。液晶ディスプレイを接続しており観客に提示しているメイン出力として使用した。マウスと MIDI コントローラが接続されており、被験者は自由に利用できた。

5.2.1 実験１

一つ目の実験では、映像オーサリングについて知識を持ったユーザにとって DF 図が理解し易いことを確かめることを目的とし、ImproV について説明をせずに、ImproV の DF 図を提示し、その DF 図が何を意味しているか答えてもらった。返答を正確にするため Adobe After Effects にて同様の映像処理を行う作業を行ってもらった。タスク 1 に使った DF 図を図 5.3 に示す。図 5.3 では二つの Video File からの出力がそれぞれ不透明度を変更する Transparency を通して Mixer に繋がれており、二つの映像が重ねられている。

図 5.2: 被験者実験の様子

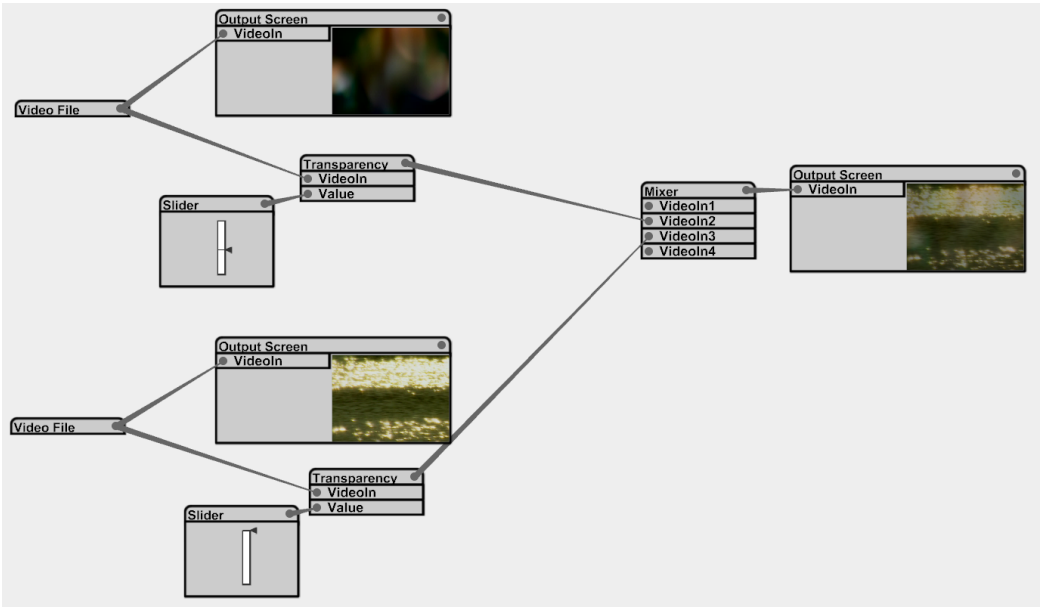


図 5.3: 実験 1 に使った DF 図

5.2.2　実験 2

実験 2 では以下に示す三つのタスクを、ImproV を使って行ってもらい、それぞれのタスクにかかる時間を計測した。

- タスク 1　メイン出力に流れている映像に Blur（ぼかし）のエフェクトを適用する。
- タスク 2　メイン出力に流れている映像を別の映像に切り替える。
- タスク 3　メイン出力に流れている映像の一部分にだけ Blur（ぼかし）のエフェクトを適用する。

被験者にはそれぞれの実験において、実際のパフォーマンスを想定し、観客に提示していると仮定した映像を、なるべく滑らかに切り替えることに留意してもらった。それぞれのタスクの開始時は、あらかじめ必要な映像ファイルが Video File として読み込まれており、観客に提示しているメイン出力、及びプレビュー用の Output Screen に配置された状態であった。

タスク 1 は最も単純なタスクである。指示されたエフェクトとスライダーをメニューから生成し、結線を行うだけである。タスク 2 はタスク 1 より若干複雑なタスクであり、3.5.2 項にて述べた手順を行う必要がある。タスク 3 は、かなり複雑なタスクである。タスク 1 とタスク 2 では、作業中一時的に、もしくはプレビューのためにしか、映像を分岐させることは必要としないが、タスク 3 では最終的に完成した DF 図に映像の分岐が含まれる。また、複数の映像を試行錯誤しながら合成し、その合成結果をさらにその時メイン出力に流れている映像と切り替えることが必要とされる。

また参考として、Adobe After Effects を使った同様のタスクを行ってもらい、それぞれのタスクにかかる時間を計測した。タスクの順番、及び使用するツールの順番はランダム化した。Adobe After Effects は、ライブ映像パフォーマンスのツールでは無く、リアルタイムに映像を再生する事や、表示している映像をリアルタイムに切り替える事はサポートされていない。そのため、Adobe After Effects では再生は停止したままとし、また切り替えについては、Adobe After Effects の合成セットであるコンポジションを複数作り、そのうち一つのコンポジションを観客に提示していると仮定し、他のコンポジションをプレビュー及び作業用としてタスクを行ってもらった。作業中は図 5.2 に示すように、実験者が隣に座っており、どちらのツールを使った場合も、被験者はツールの使い方について自由に実験者に質問することを許されていた。また、操作中はなるべく発話思考してもらい、操作の様子はビデオカメラで撮影した。

5.2.3　事後アンケート

事後アンケートでは以下に示す項目について 1〜4 の 4 段階で答えてもらった。

設問 1 ImproV の表示は、どのように映像が合成されているか適切に表現できていましたか？

設問 2 エフェクタやミキサ等を繋げて行って映像を合成していく操作方法は直観に沿っていましたか？

設問 3 上記の操作方法はライブ映像パフォーマンスの最中に行うにあたって現実的だと感じましたか？

設問 4 プレビューは適切に行えましたか？

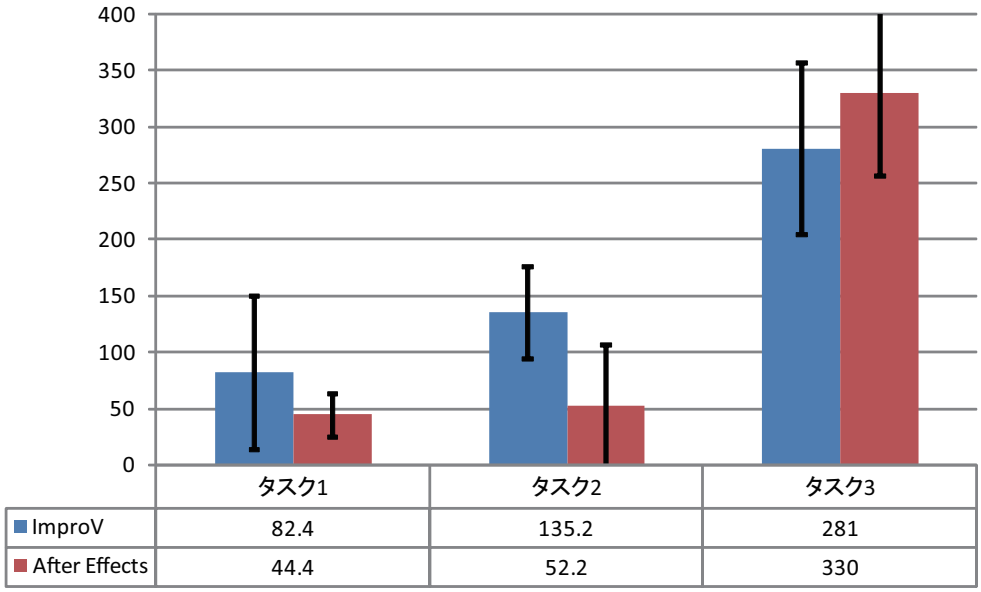
設問 5 エフェクト等のノードやそれらのパラメータの命名は分りやすいですか ？

設問 6 ノード同士を繋ぐ時の操作は直感的に行えましたか ？

これらの設問には、それぞれに自由に記述できる欄を設け、コメントを記述してもらった。また、アンケートの最後には ImproV に対する要望を自由に記述してもらった。

5.2.4 結果

実験 1 では被験者全員が正しい回答をした。
実験 2 において計測した時間を図 5.4 に、分散分析を行った結果を表 5.1 に示す。



	タスク1	タスク2	タスク3
ImproV	82.4	135.2	281
After Effects	44.4	52.2	330

図 5.4: 実験 2：タスクにかかった時間の平均

source	SS	df	MS	F	p
subject	30605.4666667	4	7651.3666667		
ツール	4320.0000000	1	4320.0000000	3.359	0.1408
error[AS]	5144.0000000	4	1286.0000000		
タスク	347965.8000000	2	173982.9000000	31.067	0.0002
error[BS]	44802.5333333	8	5600.3166667		
ツール × タスク	22515.0000000	2	11257.5000000	3.803	0.0691
error[ABS]	23684.0000000	8	2960.5000000		
Total	479036.8000000	29			

表 5.1: 実験 2：タスクにかかった時間の分散分析

ImproV と After Effects の二つのツールの主効果（$F = 3.359$）は有意でなかった（$p > 0.1$）。タスクの主効果（$F = 31.067$）は 1% 水準で有意だった（$p < 0.01$）。交互作用（$F = 3.803$）は 10% 水準で有意傾向であった（$p < 0.1$）。

	タスク 1	タスク 2	タスク 3
mean	63.400	93.700	305.500
n	10	10	10

pair	r	nominal level	t	p	sig.
タスク 3 - 1	3	0.0166667	7.234	0.0000894	s.
タスク 3 - 2	2	0.0333333	6.329	0.0002257	s.
タスク 2 - 1	2	0.0333333	0.905	0.3917249	n.s.

MSe=5600.316667, df=8, significance level=0.050000

表 5.2: 実験 2:タスクの主効果における多重比較

タスクの主効果についての多重比較を表 5.2 に示す。タスク 1 とタスク 3（$p < 0.05$）、タスク 2 とタスク 3（$p < 0.05$）はそれぞれ有意な差があったが、タスク 1 とタスク 2 には有意な差が見られなかった（$p > 0.05$）。

effect	SS	df	MS	F	p
ツール (タスク 1)	3610.0000000	1	3610.0000000	1.503	0.2438
ツール (タスク 2)	17222.5000000	1	17222.5000000	7.169	0.0201
ツール (タスク 3)	6002.5000000	1	6002.5000000	2.499	0.1399
error		12	2402.3333333		
タスク (ImproV)	105812.4000000	2	52906.2000000	12.360	0.0006
タスク (After Effects)	264668.4000000	2	132334.2000000	30.916	0.0000
error		16	4280.4083333		

表 5.3: 実験 2:交互作用における単純主効果

交互作用における単純主効果を表 5.3 に示す。タスク 2 において、ツールの主効果（$F = 7.169$）が有意であった（$p < 0.05$）。また、ImproV におけるタスクの主効果（$F = 7.169$, $p < 0.05$）及び、After Effects におけるタスクの主効果（$F = 7.169, p < 0.05$）も有意であった。

事後アンケートの結果について、各設問に対して縦軸を評価の点数の平均として図 5.5 に示す。

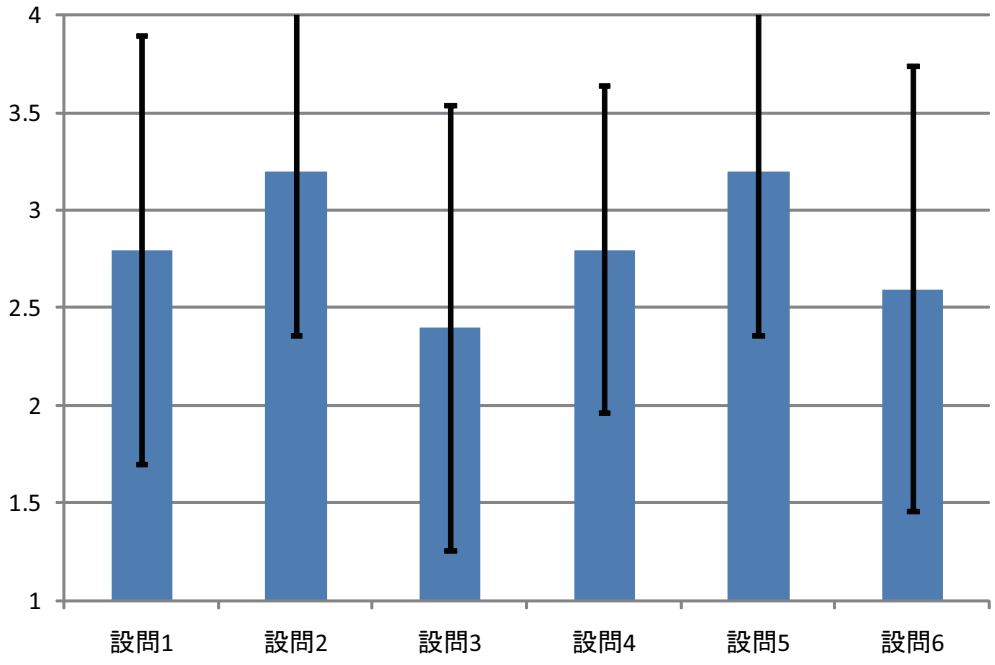


図 5.5: 事後アンケートの結果

実験 2 では、被験者をライブ映像パフォーマンスの経験者であり、Adobe After Effects の使用経験があるとし、母集団を絞ったにもかかわらず分散の大きい結果となった。After Effects

の使用方法や知識に大きなばらつきがあることが伺える。
　一方で、タスク達成にあたって試行錯誤が必要なタスク 3 は、それほど有意な差が出なかった。これは、タスクの順番及び、使用するツールの順番をランダム化したため、学習効果によるばらつきが大きくなってしまったことも原因と考えられる。

第6章　議論

本研究では、ライブ映像パフォーマンスの最中に操作可能なシステムの自由度を多くすることにより、即興性を向上させようとした。

被験者実験の結果からは、ImproV の DF 図表現がライブ映像パフォーマンスの演者にとって直観的であり、追加の学習がなくても理解することが可能であることが示された。

しかし、システムの操作性も即興性に大きく影響する要素であり、被験者実験の際に被験者から寄せられたコメントや意見では ImproV の操作性、特に DF 図編集時の操作性には改善の必要があることがわかった。

また、1.1 節で述べたように、ImproV は Lew による 3 段階の DJ の作業手順 [Lew04] のうち、「手順 2：プレビューと調整」のみを対象とした。しかし、今後 ImproV を統合的なライブ映像パフォーマンス環境として開発していくには、「手順 1：メディア検索」、「手順 3：ライブ操作」についても視野に入れなければならない。

6.1　高水準の DF 図設計

実験 1 の結果、事前アンケートで被験者全員が DF 図を描いた事、事後アンケートの設問 2 と設問 5 において特に良い結果が得られた事、及び、以下のコメントから、機材やエフェクター等をノードとする DF 図は、映像制作を行うユーザにとって理解が容易であると考えることが出来、1.3 節で述べた仮定も支持されたといえる。また、ImproV の DF 図及び、ノードの抽象レベルはライブ映像パフォーマンスの演者にとって直感的であったと言える。

1. 「映像制作の基礎があれば直観的にも使用可能。」
2. 「なんとなくわかるけど日本語がいい。」
3. 「複雑なエフェクトは無かったので分かりやすかった。」
4. 「直感的で分かりやすいが、操作性に問題があるような気がする。」

6.2　映像の切り替え

タスク 1 とタスク 2 に有意差が見られなかった。実験中撮影したビデオを分析すると、タスク 1 にてほとんどの被験者が、どちらのツールを使った場合も、メイン出力の映像へエフェ

クトを直接適用せず、一旦メイン出力と同じ映像を分岐させてからエフェクトを適用し、その映像をタスク 2 と同じ方法でメイン出力へフェードインさせていた。

これは ImproV、After Effects の両方で、被験者らが、プレビューを使って作業し、メイン出力への意図しない影響を避ける方法をとったと言える。つまり、ライブ映像パフォーマンスにおいて、メイン出力とは別の作業領域での作業が行える事、またその作業のプレビューが行える事が重要であることが分かった。

また、タスク 2 の切り替え、及び上記のようにタスク 2 と同じ方法でタスク 1 の切り替えを行った場合全てにおいて、被験者らがアルファ合成と透明度の変更によるフェードインを行っていた。

ImproV でフェードインを行うには、アルファ合成を行う Mixer、透明度の変更を行う Transparency を、他の DF 図編集操作と同様に、それぞれ適切に生成、結線する必要がある。しかし上記の結果からは、DF 図編集とは別の特別な機構として、アルファ合成と透明度の変更によるフェードイン機能を備えることが有効であると考えられる。

6.3 ライブ映像パフォーマンス最中の DF 図編集操作

交互作用について見てみると、タスク 2 では After Effects を使った場合と比べると、ImproV を使った場合、2 倍以上の時間がかかってしまっている。この結果により、ImproV の操作性は After Effects ほど良くないと考えられる。

また、ImproV の操作性について、以下に示すように改善につながるコメントが多く寄せられた。

1. 「シビアすぎる。現場は暗いのでマウスでシビア操作は厳しい。」
2. 「ひっつく動きが欲しい。線の数も無制限ぽくて混乱する。」
3. 「慣れれば強力だが、操作感などの改善が必要。」
4. 「直感的で分かりやすいが、操作性に問題があるような気がする。」
5. 「エフェクトのかかっていない映像にエフェクトをかけるときに繋ぎ直すのがめんどくさい。」
6. 「1 人（で作業することが）が多いのでフローを組んでられないと思う。もう一人専用に必要。」
7. 「エフェクトにスライドパラメータを後で付け加えるのではなくて、エフェクトを配置した瞬間パラメータを操作できるようにしてほしい。」
8. 「それぞれのエフェクトにスライダ、モニタは必ず付くのがいい。」

また、被験者実験 2 のタスク 2 での結果において、特に単純な映像の切り替えやエフェクトの適用の作業において、After Effects と比べ時間がかかったことも、これらのコメントと矛盾しない。

上記コメントの 1–3 は、視覚要素を大きくすることやスナップの機能によって改善出来ると考えられる。しかし上記コメントの 4–7 については、DF 図の編集時に結線という作業が発生することを軽減することが必要である。現在の ImproV で各ノードのパラメータを操作するには図 6.1 に示すように、Slider を作り、結線しなければならない。解決策として、実数値の入力を持つノードを生成したときに Slider を自動的に生成、及び結線することが考えられる。また別の解決策として、図 6.2 に示すように、各ノード上にそれぞれスライダーを持たせる事が考えられる。

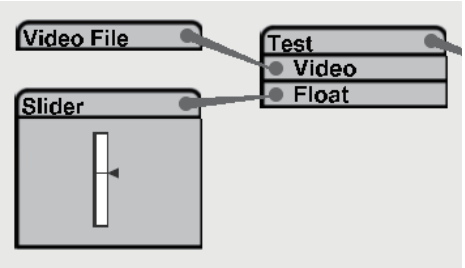


図 6.1: Test ノードにスライダを結線したところ

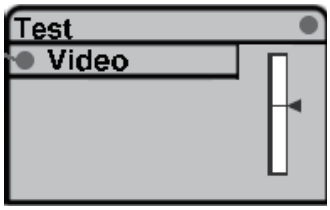


図 6.2: Test ノード自体にスライダを配置する

そのほかにも、ノード生成時に、選択中のノードや最後に生成されたノード等、その時の DF 図編集のコンテキストに基づいて、自動的に結線することも考えることが出来る。また、既に結線されたノードに対して、入力の個数や型が同様のノードを代替可能ノードとして提示することや、代替可能ノードに手早く置き換える機構を提供することも有効と考えられる。

複数のノードの組み合わせを保存する事や、そのような組み合わせを一つのノードとして扱う、パッケージ化という機能によっても、DF 図編集時の結線操作を軽減出来ると考えられる。以下のコメントからも、そのような機能が望まれていることが伺える。

1. 「サンプルエフェクトの充実（A のエフェクトをかける時はワンクリックみたいな）」
2. 「セットのアーカイブ化」

6.4　メディア検索

現在の ImproV では映像ファイルのメディア検索は他のアプリケーションに委ねている。これによりプレビューの機能を持ったアプリケーション、例えば Adobe Brigde を ImproV と併用し、映像ファイル選択時のプレビューを実現できる。しかし、カメラやエフェクタ等のノードを選択するときにプレビューを行うことは出来ない。今後、ノードの種類が増えてくると、ノード選択時のノードプレビューやノード検索が必要になってくると考えられる。例えば、現在の ImproV には、ぼかしエフェクタとして Blur が 1 種類しか実装されていないが、一般的な映像合成アプリケーションには、ガウスブラー、方向性ブラー、拡散ブラー等、ぼかしだけでも様々なエフェクタを持っている。今後、このようなエフェクタを実装し、ノードの種類が増えた時、現在の ImproV では一旦そのノードを生成し、何らかの映像を結線しその出力を確認しなければどのようなエフェクタかは分からない。この作業を軽減することはライブ映像パフォーマンス最中の試行錯誤を促進することに繋がる。また、ノード生成のメニューも整理する機能が無く、例のように微妙な差異をもった同系統のエフェクタを階層的に分ける事や、絞り込んで検索する事が必要となる。また、このようなノード検索機能やプレビュー機能は、映像ファイル検索機能と統合され、シームレスに扱えることが求められると考えられる。

6.5　ライブ操作

ライブ操作の手順では、多数のパラメータを変化させることが困難な点が問題となることがある。演者はエフェクタやビデオミキサに備え付けられたつまみやスライダなどを手で操作してパラメータを変化させる。これは直感的に操作できる一方、同時に変化させられるパラメータの数は限られてしまう。映像表現として時間的変化は重要であり、この問題を解決することは表現の幅をより広げることに繋がると考えられる。

以下では、時間的変化を自動的に行う、アニメーションについて考察する。

6.5.1　タイムラインノード

初期の ImproV では、キーフレームアニメーションを導入し、タイムラインノード図 6.3 と呼ぶ DF 図上のノードとして表現することによって解決を図った [小林 08a]。これは、「手順 3：ライブ操作」での作業の一部を「手順 2：プレビューと調整」の手順で済ませてしまおうとするアプローチであると言える。

中央部分はキーフレームを表示するタイムラインである。タイムライン上にある縦線は再生している時間を表すインジケータである。左下には再生ボタンを備えている。全てのタイムラインノードはインジケータと再生ボタンを一つづつ持っている。これら、タイムラインノードはそれぞれ再生時間軸を持っており、個別に再生、停止が出来る。これにより、あるタイムラインを再生中でもその再生に影響することなく、別のタイムラインを編集やプレビューのために再生、停止を行うことが出来る。

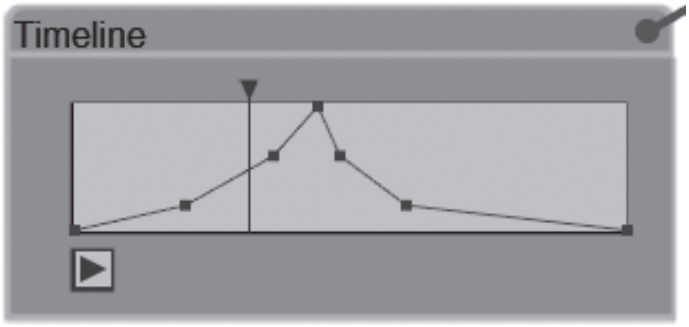


図 6.3: タイムラインノード

また、一つのタイムラインノードを別のタイムラインノードにドラッグアンドドロップすることで、複数のタイムラインノードを合成することが出来、合成されたタイムラインノードは再生時間軸を共有し同期される。同期化されたタイムラインノードの例を図 6.4 に示す。

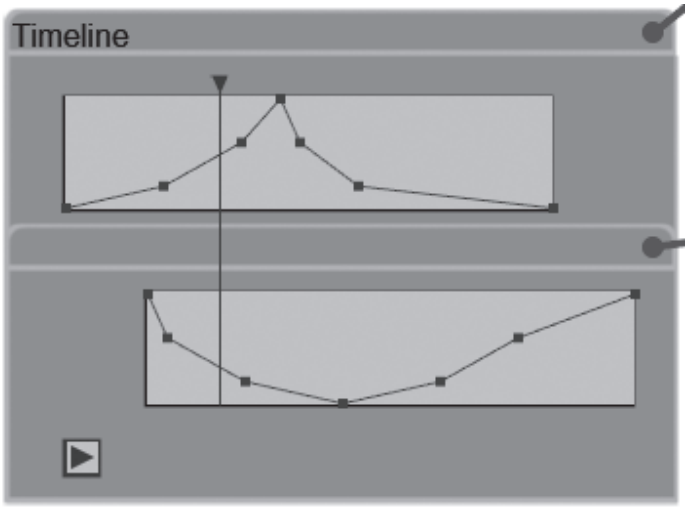


図 6.4: 同期化されたタイムラインノード

二つのタイムラインを持っているが、インジケータと再生ボタンは一つしか持っていない。これは二つのタイムラインが同じ時間軸上で再生されることを表している。

このタイムラインノードにより、複雑なアニメーションを行うことが可能である。図 6.5 では、一つの画像を 6 つに分岐させ、それぞれタイミングをずらした回転のアニメーションを適用し、ミキサによってふたたび一つの映像へ合成している（映像 A)。さらにその合成結果を別の映像 B と合成している。

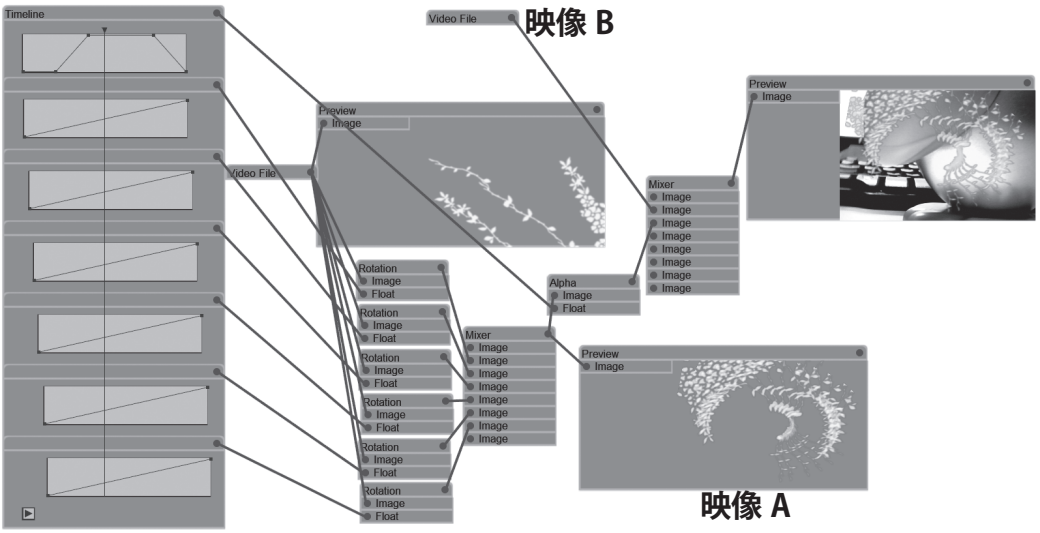


図 6.5: タイムラインノードを使った複雑なアニメーション

6.5.2 キーフレームアニメーション以外のアニメーション指定

実際にライブ映像パフォーマンスを行った予備実験で、タイムラインノードを使用するには、複雑なキーフレーム編集操作が必要であり、操作に時間がかかりすぎてしまうことが分かった。

アニメーション、つまり時間的変化について、音楽の分野から取り入れることが出来る手法がいくつかある。

一つは、Low Frequency Oscillator（LFO）である。LFO は通常のオシレータが音声波形を生成するのに比べ、低い周波数を生成する。シンセサイザによる音声合成では LFO によって、オシレータの音量を制御することでトレモロ効果を実現することや、オシレータの周波数を制御することでビブラート効果を実現すること、もしくは、他のパラメータを制御することで様々な効果を実現する。LFO に対する指定も、周波数、位相、波形の種類とそれほど複雑ではない。ImproV に導入するにあたっても、これらのパラメータを実数値の入力として備えたノードを実装できる。

また、音楽の分野ではパターンシーケンサやリズムマシンといった機構がある。これはノート信号のシーケンスを指定する機構で、ある時間範囲（一小節等）を 16 程度に分割し、ノートを指定する。図 6.6 は Reason に含まれるシンセサイザである。手早くシーケンスの指定が出来るのでライブパフォーマンスで使われることもある。

現在、この機構を取り入れたアニメーション指定方法を検討中である。升目に区切った箱の中にパラメータ指定のウィジェットを配置する（図 6.7）。

キーフレームタイムラインの方がより複雑なアニメーションを指定できるが、ライブ映像パフォーマンス中においては、より手早く出来ることが重要と考える。ImproV にパターンシーケンサを導入するにあたっては検討すべき事項がいくつかある。最も大きな問題は、ノード

図 6.6: propellerhead Reason の RPG-8

図 6.7: パターンシーケンサを取り入れたアニメーション指定

の一つとして取り扱うべきか、DF 図とは別の機構として取り扱うべきかである。画面面積も問題となることが予想される。

ただし、LFO、パターンシーケンサ両方に言える問題として、ライブ映像パフォーマンスの演者にとって、これらの機能は馴染みの無いものであることに留意しなければならない。ライブ映像パフォーマンスの演者が、これらの機能に適応し直観的に使えるかどうかは、慎重な議論を要する。

第7章　結論

本研究では、DF 図に基づいた映像合成システム、ImproV の設計、実装を行い、そのシステムによってライブ映像パフォーマンスにおいて必要となる、複数の映像経路の切り替えや、それらのプレビューを可能にした。これにより、従来は準備段階の制作の一部であった映像の合成を、ライブ映像パフォーマンスの最中に、提示中の映像に影響を及ぼすことなく試行錯誤を行いながら、即興的に行う事が可能となった。

被験者実験により、ImproV の DF 図は、ライブ映像パフォーマンスを行う演者にとって理解しやすいものであることが分かった。一方で、ImproV における DF 図編集操作は改善の余地があることも明らかとなった。

今後は、ImproV における DF 図編集操作の改善を中心に、メディア検索機能や、ライブ操作に関する機能の設計、試作を含めた、ImproV の更なる改良を進めてゆく。

謝辞

主任指導教員である田中二郎教授をはじめ、 三末和男准教授、 講師の志築文太郎先生と高橋伸先生には、大変貴重なご意見やご指導、激励を頂き、本論文執筆に際しても大変お世話になりました事、深く御礼を申し上げます。

田中研究室の佐藤大介氏、村田雄一氏による iCamera ライブラリには、ImproV 実装に際し、大変助けられました。ライブラリを制作し、使用出来る形で残して下さった事、ここにお礼申し上げます。

多忙の中、被験者実験に参加していただき、多くの有用なコメントを残して下さった被験者の皆様にも感謝の意申し上げます。

また、田中研究室の皆様にも、多くの意見、 アドバイスを頂きましたこと深く御礼を申し上げます。

参考文献

['74] Cycling '74. Jitter. http://www.cycling74.com/products/jitter.

[Ari07] Stefan Müller Arisona. Live performance tools: part II. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pp. 73–126, New York, NY, USA, 2007. ACM.

[BH02] Bert Bongers and Yolande Harris. A structured instrument design approach: the video-organ. In *NIME '02: Proceedings of the 2002 conference on New interfaces for musical expression*, pp. 1–6, May 2002.

[Cor] Roland Corporation. V-4 4 Channel Video Mixer. http://www.rolandsystemsgroup.net/en/0212.htm.

[Cro08] David Cronin. FEATURE into the groove: lessons from the desktop music revolution. *interactions*, Vol. 15, No. 3, pp. 72–78, May 2008.

[Cyc] Cycling '74. Max/MSP. http://www.cycling74.com/products/maxmsp.

[Des] Mavrick Designs. VisualJockey. http://www.visualjockey.com/.

[dTII07] Satoru dangkang Tokuhisa, Yukinari Iwata, and Masa Inakage. Rhythmism: a VJ performance system with maracas based devices. In *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*, pp. 204–207, June 2007.

[FSE04] Kentaro Fukuchi, Mertens Sam, and Tannenbaum Ed. EffecTV: a real-time software video effect processor for entertainment. In *Entertainment Computing - ICEC 2004*, pp. 602–605, September 2004.

[Hil91] Daniel D. Hils. Datavis: a visual programming language for scientific visualization. In *CSC '91: Proceedings of the 19th annual conference on Computer Science*, pp. 439–448, New York, NY, USA, 1991. ACM.

[HSMT06] Tomoyuki Hansaki, Buntarou Shizuki, Kazuo Misue, and Jiro Tanaka. FindFlow: visual interface for information search based on intermediate results. In *APVis '06: Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation*, pp. 147–152, February 2006.

[Inca] Advanced Visual Systems Inc. AVS/Express. http://www.avs.com/software/soft_t/avsxps.html.

[Incb] Apple Inc. Quartz Composer. http://developer.apple.com/graphicsimaging/quartzcomposer/.

[Incc] Derivative Inc. TouchDesigner . http://www.touch077.com/Products/.

[Incd] Digitalstage Inc. motion dive .tokyo. http://www.digitalstage.net/en/.

[Insa] Native Instruments GmbH. REAKTOR. http://www.native-instruments.com/index.php?id=reaktor5&L=0.

[Insb] Native Instruments GmbH. REAKTOR Core Technology. http://www.native-instruments.com/index.php?id=coretechnology&L=1.

[JHM04] Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. Advances in dataflow programming languages. *ACM Computing Surveys*, Vol. 36, No. 1, pp. 1–34, March 2004.

[KMA04] Andrew J. Ko, Brad A. Myers, and Htet H. Aung. Six learning barriers in end-user programming systems. In *Visual Languages and Human-Centric Computing, 2004. VL/HCC 2004. IEEE Symposium on*, pp. 199–206, Los Alamitos, CA, USA, September 2004. IEEE Computer Society.

[KST09] Atsutomo Kobayashi, Buntarou Shizuki, and Jiro Tanaka. Improv: A system for improvisational construction of video processing flow. In *International Conference on Human-Computer Interaction 2009*, 2009. (to appear).

[Lew04] Michael Lew. Live cinema: designing an instrument for cinema editing as a live performance. In *NIME '04: Proceedings of the 2004 conference on New interfaces for musical expression*, pp. 144–149, June 2004.

[MÖ7] Pascal Müeller. Live visuals tutorial: part III. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pp. 127–151, New York, NY, USA, 2007. ACM.

[MASBS06] Pascal Müeller, Stefan Müller Arisona, Simon Schubiger-Banz, and Matthias Specht. Interactive media and design editing for live visuals applications. In *International Conference on Computer Graphics Theory and Applications*, pp. 232–242, 2006.

[MW97] Adrian Freed Matthew Wright. Opensound control: a new protocol for communicating with sound synthesizers. In *International Computer Music Conference 1997*, pp. 101–104, Thessaloniki, Hellas, September 1997.

[Spi05] Paul Spinrad. *The VJ Book: Inspirations and Practical Advice for Live Visual Performance*, chapter History, pp. 17–24. A Feral House book, 2005.

[Tec] Avid Technology, Inc. USB MIDI Control Surface UC-33e. `http://www.maudio.com/products/en_us/UC33e-main.html`.

[Tok08] Satoru Tokuhisa. Nervixxx. Laval Virtual Award 2008, April 2008.

[vvv] vvvv group. vvvv. `http://vvvv.org/`.

[WKU+07] Torben Weis, Mirko Knoll, Andreas Ulbrich, Gero Mühl, and Alexander Brändle. Rapid prototyping for pervasive applications. *IEEE PERVASIVE COMPUTING*, Vol. 6, No. 2, pp. 76–84, 2007.

[小林 08a] 小林敦友, 志築文太郎, 田中二郎. データフロー図に基づくリアルタイム映像合成システム. 情報処理学会研究報告 (128 回ヒューマンコンピュータインタラクション研究会), No. 50 in 2007, pp. 1–8, May 2008.

[小林 08b] 小林敦友, 志築文太郎, 田中二郎. リアルタイム映像パフォーマンス向け映像合成システム. 情報処理学会第 70 回全国大会講演論文集, 第 4 巻, pp. 157–158, March 2008.

[平井 08] 平井重行, 田中秀明. 音声入力によるビジュアルプログラミング環境操作支援インタフェース. 情報処理学会研究報告 (128 回ヒューマンコンピュータインタラクション研究会), No. 50 in 2007, pp. 19–24, May 2008.

付 録A　評価に使った質問票

次ページ以降は被験者実験の際、被験者に配られた質問票である。実験者は、被験者に質問票を一枚づつ提示、及び説明した。11–16 ページのタスクは、被験者ごとに順番をランダム化され、その順番に沿って一枚づつ提示、及び説明された。

今回の実験について

この度は実験にご協力いただき、ありがとうございます。

本実験は、ライブ映像パフォーマンスのためのツール、ImproV とその基盤となった理論の有用性を調査するための実験です。

実験時に収集した個人情報は実験に関する連絡や報告以外の用途では使用いたしません。実験で得られたデータは個人が特定できない形で統計的に処理され、学内外で発表する論文などで利用します。また、研究遂行上の必要に応じて、研究室内の共同研究者とデータを共有することがあります。

本実験はいつでも中断・辞退できます．中断や辞退によって被験者の方が不利益を被ることはありません。

本実験についてのご質問・要望などがございましたら，実験コードと被験者 ID を明記の上、以下の連絡先までお気軽にご連絡ください。

実験担当者　小林敦友
メールアドレス　atsutomo@iplab.cs.tsukuba.ac.jp
Web　http://www.iplab.cs.tsukuba.ac.jp/~atsutomo/
研究室　総合研究棟 B 棟 (内線 5425)
筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
インタラクティブプログラミング研究室 (田中研究室)
http://www.iplab.cs.tsukuba.ac.jp/

実験手順

実験は以下の通り進められます。

1. 事前アンケート：あなたの映像オーサリングやライブ映像パフォーマンス (VJ) の経験についての質問に答えていただきます。
2. 実験１：こちらが出す問題について答えていただきます。一部、Adobe After Effects の作業を行っていただきます。必要であれば、Adobe After Effects の操作方法について説明します。
3. 説明：ImproV について説明します。
4. 実験２：Adobe After Effects や ImproV を使った簡単な作業を行っていただきます。
5. 事後アンケート：ImproV の操作感や感想について答えていただきます。

実験に際してのお願い

- 実験中にはあなたが考えていることを声に出しながら、作業を行っていただけると助かります
- 作業をこなしている様子をビデオカメラでの撮影と音声の録音をさせてください。

これらは、ツールの機能が我々の意図通りであるかを検証する調査等に使用します。

事前アンケート

Q1. あなたがライブ映像パフォーマンス (VJ) で使う機材やソフトウェアの構成について図もしくは文章を使って教えて下さい。

回答欄

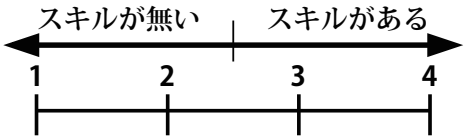
Q2. あなたがライブ映像パフォーマンス (VJ) の準備で行う作業について、作業の手順や気をつけている点、かかる時間など、図もしくは文章を使って教えてください。

回答欄

Q3. あなたがライブ映像パフォーマンス (VJ) の最中に行う作業について、作業の手順や気をつけている点、かかる時間など、図もしくは文章を使って教えてください。

回答欄

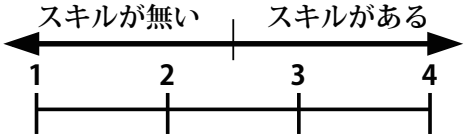
Q4. あなたの Adobe After Effects の使用経験とスキルについて教えてください。
使用経験：__年



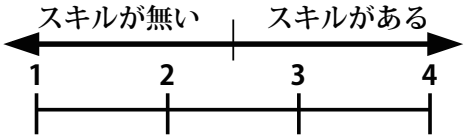
Q5. あなたの Adobe After Effects 以外の映像関連ソフトウェアの使用経験について教えてください。（複数ある場合はよく使う順に 3 つ）

- 経験がない

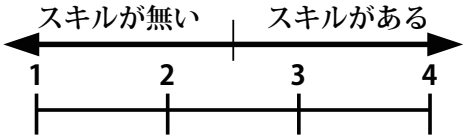
1. ソフトウェア名：
使用経験：__年



2. ソフトウェア名：
使用経験：__年



3. ソフトウェア名：
使用経験：__年

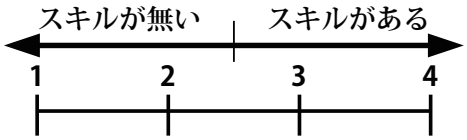


事前アンケート

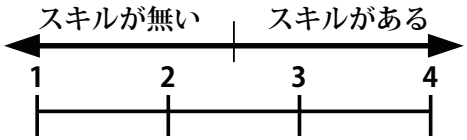
Q6. あなたのコンピュータプログラミングの経験について教えてください。（複数ある場合はよく使う順に 3 つ）

- 経験がない

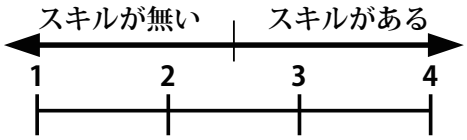
1. プログラミング言語名：
使用経験：__年



2. プログラミング言語名：
使用経験：__年



3. プログラミング言語名：
使用経験：__年



実験 1

必要があれば Adobe After Effects の操作方法について説明を受け、操作の練習を行ってください。

Q1.

提示される図について何を意味しているか口頭で答えてください。
あなたは、表記されている英単語の意味については質問できます。

Q2.

Q1 で提示されたものを Adobe After Effects で再現してください。必要な素材は Adobe After Effects に読み込まれています。
あなたは、表記されている英単語の意味や、Adobe After Effects の操作方法について質問できます。

ImproV の説明、練習

mproV について説明を受け、操作の練習を行ってください。

- 文法
- 操作方法
- 各ノード種類
- 作業フロー
- 使用例

実験 2

指定されるタスクを ImproV か Adobe After Effects で行ってください。それぞれ指定されたセーブファイルを開いてからの作業となります。また、ImproV と Adobe After Effects のどちらで作業を行うかについては指示があります。

実際のパフォーマンスを想定し、観客に見せている映像はなるべく滑らかに切り替えるようにしてください。Adobe After Effects では実際には滑らかに切り替えることは出来ませんが、「特定の操作を行えば切り替わる」という状態にしてください。

タスク

Adobe After Effects で指定された映像に Blur（ぼかし）エフェクトをかけてください。

タスク

ImproV で指定された映像に Blur（ぼかし）エフェクトをかけてください。

タスク

Adobe After Effects で指定された映像をもうひとつの映像にフェードインしてください。

タスク

ImproV で指定された映像をもうひとつの映像にフェードインしてください。

タスク

Adobe After Effects で指定された映像に Blur（ぼかし）エフェクトをかけてください。ただし、もうひとつの映像の白い部分にだけ Blur（ぼかし）エフェクトがかかるようにしてください。

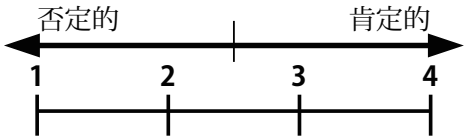
タスク

ImproV で指定された映像に Blur（ぼかし）エフェクトをかけてください。ただし、もうひとつの映像の白い部分にだけ Blur（ぼかし）エフェクトがかかるようにしてください。

事後アンケート

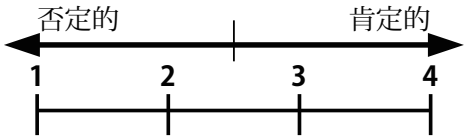
ImproV について、各質問に答えてください。出来れば理由や、コメントも記入してください。

Q. ImproV の表示は、どのように映像が合成されているか適切に表現できていましたか？



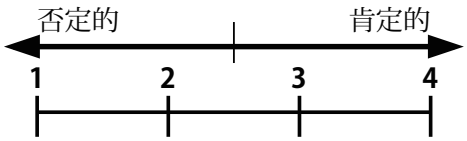
理由・コメント：

Q. エフェクタやミキサ等を繋げて行って映像を合成していく操作方法は直観に沿っていましたか？



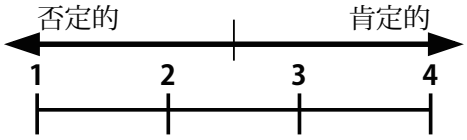
理由・コメント：

Q. 上記の操作方法はライブ映像パフォーマンスの最中に行うにあたって現実的だと感じましたか？



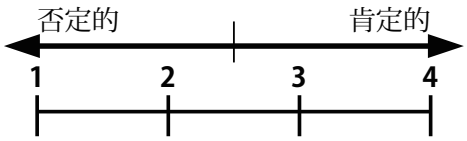
理由・コメント：

Q. プレビューは適切に行えましたか？



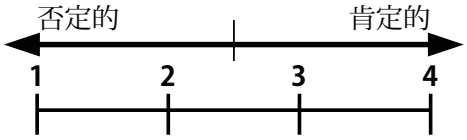
理由・コメント：

Q. エフェクト等のノードやそれらのパラメータの命名は分りやすいですか ？



理由・コメント：

Q. ノード同士を繋ぐ時の操作は直感的に行えましたか ？



理由・コメント：

Q. ImproV の改善すべき点や、ImproV に欲しい機能などあればご自由にご記入ください。

回答欄

Q. その他、今回の実験で感じたことがあればご自由にご記入ください。

回答欄