映像を即興的に加工及び合成するための データフロービジュアル言語に関する研究

小 林 敦 友

システム情報工学研究科

筑波大学

2012年3月

概 要

本論文では、即興的に加工及び合成するデータフロービジュアル言語に関する研究について述べる。ここで加工とは映像に映像エフェクタを適用することであり、合成とは複数の映像を組み合わせることである。

本研究の対象はライブ映像パフォーマンスである。ライブ映像パフォーマンスとは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行われる、演者が映像を観客に提示し、同時に演者が映像を操作するパフォーマンスである。従来のライブ映像パフォーマンスでは、予め映像素材を用意しておき、本番では用意した映像素材を切り替えてゆくという手法が採られる。ライブ映像パフォーマンスはイベントの雰囲気にあったものである必要があるため、ライブ映像パフォーマンスの演者は、多くの種類の雰囲気に合わせるために、大量の映像素材を用意する。

本研究では、従来のライブ映像パフォーマンスで行われていた、映像の切り替えに加え、映像の加工及び合成もライブ映像パフォーマンスの最中に行うことを可能にすることにより、よりイベントの雰囲気にあったライブ映像パフォーマンスを行うことを可能にする。このために本研究では、データフロービジュアル言語を使って映像の加工や合成の方法を指定することとした。

ImproV は、映像処理と映像の流れをそれぞれ、ノードとエッジにて表すデータフロービジュアル言語である。映像再生中にもデータフロー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加工及び合成し、それらの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加工及び合成を、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、即興的に行うことが可能である。

また、ImproV のそれぞれのノードは、映像ソース、映像エフェクタ、映像ミキサ、映像出力を表し、これは映像機材と同等の水準である。ライブ映像パフォーマンスの演者は映像機材に関する知識を持っているため、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によって映像合成処理の流れを表現するといえる。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 ImproV のデータフローによる表現が理解できることを確かめた。

映像制作段階における映像の加工及び合成では、映像の一部のみを映像エフェクタによって加工することや映像エフェクタのパラメータをアニメーションさせることがある。これらをライブ映像パフォーマンスの最中に行うため、映像によってパラメータを指定する機構を考案し、実装した。

ライブ映像パフォーマンスの最中にデータフローを記述してゆくと、データフローが複雑になってしまうことがある。これを軽減するため、データの調整に着目し、この記述を簡潔にするシンタクティックシュガーを考案した。また、このシンタクティックシュガーを ImproV に実装し、有効性を確かめた。

目 次

図 目 次

表 目 次

第1章　序論

本章では、表題に現れる言葉である「データフロービジュアル言語」と「映像を即興的に加工及び合成する」とは何を指すのかについてそれぞれ説明する。次に、本研究が対象とするライブ映像パフォーマンスについて説明する。その後、本研究の目的とアプローチについて説明し、本研究の成果を述べ、本論文の構成について説明する。

1.1　データフロービジュアル言語

「ビジュアル言語」とは、「視覚的要素の組み合わせにより記述する言語」である。すなわち、図形や文字などの視覚的要素によって構成され、それらの組み合わせと配置に規則があり、意味づけが成されている言語である。例として、フローチャート、ER 図、UML などが挙げられる。図形言語と呼ばれることもある。「ビジュアル言語」は、「視覚的情報を処理するための言語」という意味でも使われる [Cha06]。この場合は、言語表現が視覚的であるかどうかは問わず、その言語が処理する対象が画像や図形といった視覚オブジェクトであるということを指す。通常は既存のプログラミング言語をライブラリによって拡張したものであり、例として、画像を処理するための Image Processing Language[CJL+85] や、SQL に図的検索機能を加えた Pictorial Structured Query Language[RFS88] が挙げられる。本研究で扱う言語も映像という視覚的情報を処理するが、本論文では「ビジュアル言語」を、上で述べた「視覚的要素の組み合わせにより記述する言語」としての意味で使用する。

「データフロー」とは、データの流れによってシステムを記述する計算モデルである [Sha87, JHM04]。データフローはノードとエッジからなる有向グラフ構造を持つ。ノードは入力ポートと出力ポートを持ち、エッジはあるノードの入力ポートから別のノードの出力ポートへ結線される。計算を実行する時には、ノードは入力ポートから入力されたデータを処理し、出力ポートへ出力する。また、エッジはデータの流れを表し、ノードからの出力を別のノードへ入力する。データフローでは、様々な計算処理を表すノードを、エッジによって結びつけ、計算処理間の通信を記述することによって、システムを記述する。

「データフロービジュアル言語」とは、データフローを記述するためのビジュアル言語である。「データフロービジュアルプログラミング言語」とも呼ばれる。データフローの記述は、テキストによる記述も存在するが [AW77, WC04]、ビジュアル言語による記述が人間にとってわかりやすく、これまでに多く研究されてきており [JHM04]、製品化もされてきた [Adv, Cycb]。

1.2 「映像を即興的に加工及び合成する」とは

ここで映像を「加工する」とは拡大縮小や回転といった変形操作、ぼかしや色の変更といったフィルタを適用することを指す。「合成する」とは複数の映像をアルファ合成や加算合成といった演算によって重ね合わせることである。また、「合成する」には「切り替える」という意味も含む。なぜならば、例えば二つの映像を合成しているとき、一方は濃く、もう一方は薄くといった合成する度合いを変化させることによって「切り替える」という操作が実現できるからである。

「即興」とは、あるパフォーマンスにおいて、そのパフォーマンスの内容をそのパフォーマンスの最中に決めることである。即興は演劇、音楽演奏、ダンスといったあらゆるパフォーマンスにおいて使われる。例えば、演劇は通常、その話の流れや台詞が台本としてあらかじめ決められており、演者がその台本に沿って演劇を進める。しかし即興劇では台本の全部、若しくは一部が決まっておらず、演者がパフォーマンスの最中に話しの流れや台詞を考え、それを演じる。パフォーマンスに即興を取り入れることの利点は、演者同士や観客を含むパフォーマンス環境とのインタラクションをパフォーマンスに取り入れること、柔軟にトラブルに対処することが可能になることなどが挙げられる。また即興には、「即興性」という度合いが存在する。演劇の例で言えば、全く台本が決まっておらず、全てをその場で考える場合もあれば、話の流れや台本の一部は決まっておりそれ以外の部分を即興で演じる場合もある。前者の方が後者と比べて、より「即興性が高い」と言える。

本研究では、映像を即興で観客に提示する、「ライブ映像パフォーマンス」を対象としている。「映像を即興的に加工及び合成する」とは、このライブ映像パフォーマンスにおいて、映像を加工や合成する方法をその場で決め、結果、新しい映像を作り出すことを指す。特にライブ映像パフォーマンスの最中に、複数の加工処理や合成処理を組み合わせることや、その組み合わせを変更することにより、新しい加工処理や合成処理を作り出し、結果、新しい映像を作り出すことを指す。

1.3 ライブ映像パフォーマンス

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、映像を提示し、その映像の生成や切り替え、加工、合成などの操作を人間がその場で行うというパフォーマンスが行われることがある。このパフォーマンスをライブ映像パフォーマンスと呼ぶ。

ライブ映像パフォーマンスを行う演者のことを VJ（Video Jockey または Visual Jockey の略)、パフォーマンス自体を VJing と呼ぶこともある。VJing は 1970 年代後半に、クラブシーンにおいて発祥し、発展してきた [Spi05]。Lew は VJing を、ライブ映像パフォーマンスの中でも、クラブ若しくは、レイブ（野外にて行われるダンスイベント）において行われるものと分類している [Lew04]。本論文では VJing も含め、「ライブ映像パフォーマンス」という呼び方を使う。また、Müeller らはライブビジュアルパフォーマンスと呼んでいる [MÖ7]。しかし、「ビジュアル」という言葉は映像以外の視覚、例えば照明や飾り付けも含む。このため、本論

文では「ライブ映像パフォーマンス」と、「映像」という言葉を含めた呼び方を使う。Müller らの文献内でも映像以外のことには触れられていない。

ライブ映像パフォーマンスの要件として以下の二つがあげられる。

要件 1 映像を途切れること無く流し続ける

要件 2 その場の雰囲気、すなわち、音楽、照明、観客の反応等にあった映像を流す

従来は要件 1 を満たすために、予め用意した映像を、映像ミキサを使って切り替えることを繰り返すという手法が取られる [Lew04]。また、要件 2 を満たすためには、予めイベントの主催者や出演者と打ち合わせることや、予め用意する映像を大量にすることによって対応される。

1.3.1 従来のライブ映像パフォーマンスの手法

ここでは、従来のライブ映像パフォーマンスの手法について述べる。ライブ映像パフォーマンスは、文献によって様々な呼ばれ方をするが、その手法に関しては多くの文献にて一致している [Spi05, Lew04, MÖ7, 木村 04, 木村 05]。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、観客の様子など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 VJ 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簡単な映像の加工、加工結果の確認などを行い、元の映像と切り替える。この切り替えによって、映像を途切れさせること無く、切り替えて行くことが可能となる。つまり、要件 1 を満たす。

Lew はこの手法における作業手順が、Disk Jockey（DJ）の作業手順と同じであることを指摘した [Lew04]。Lew の分析による 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。

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、エフェクタの調整を行う。

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。

ビデオミキサを使った機材構成

この作業手順を実現するため VJ は、図 1.1 に示す様なビデオミキサを中心とした機材構成を使う。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。VJ 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映像エフェクタの適用、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、映像エフェクタ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。

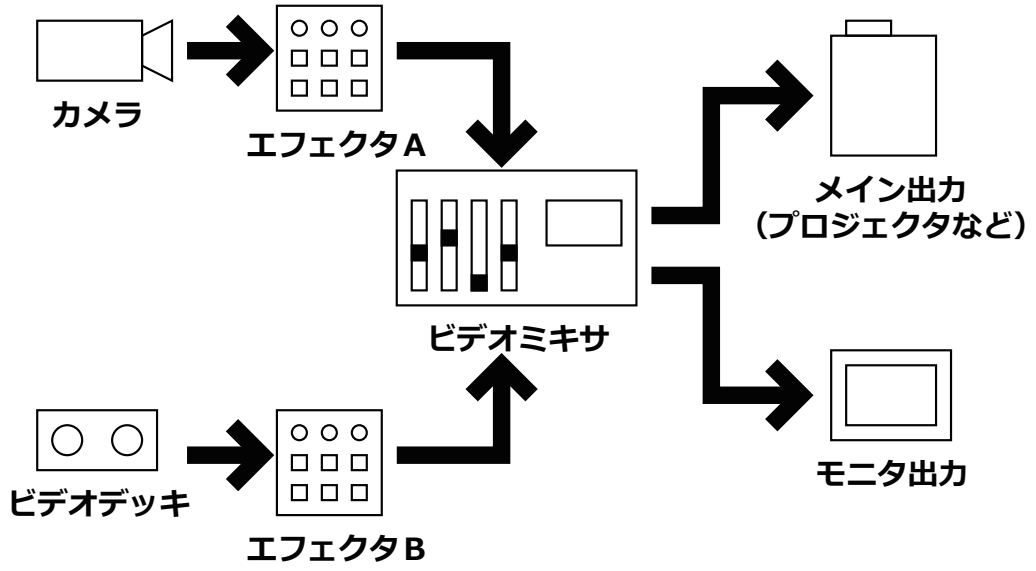


図 1.1: ライブ映像パフォーマンスのシステム構成例

このことから、ライブ映像パフォーマンスには、映像素材やそれに映像エフェクタ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。

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 VJ がライブ映像パフォーマンスの最中に、映像に対して行える操作を少なくしており、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。

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。既存のライブ映像パフォーマンスでは、VJ はその場の雰囲気に応じて映像を切り替えていく。VJ が意図しない映像が観客に見えてしまうのを防ぐため、VJ 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

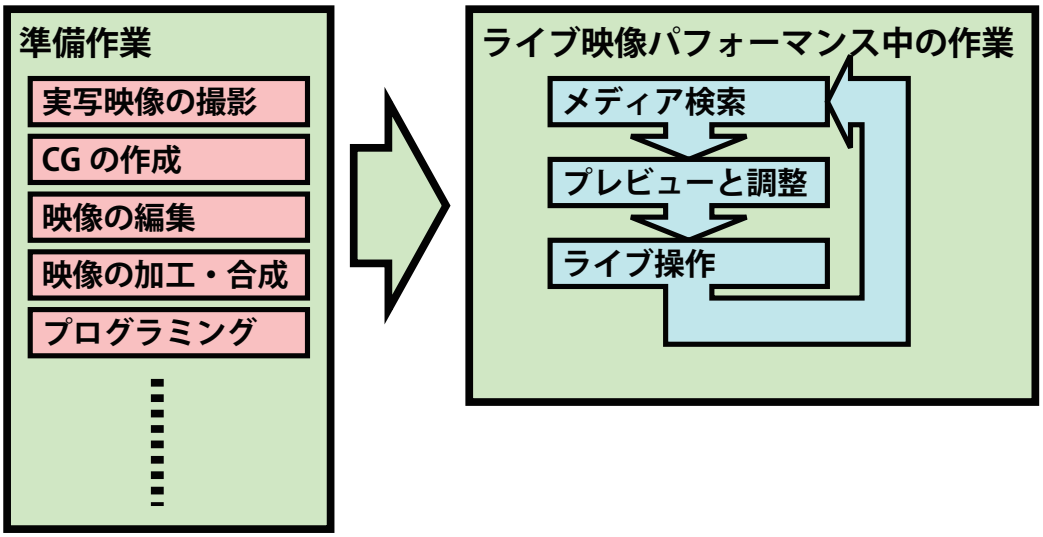


図 1.2: 準備作業を含めたライブ映像パフォーマンスの作業の流れ

材選別や映像エフェクタの適用や、合成結果のプレビューなどを行い、それらを切り替える。これら映像素材は、VJ が準備段階にてあらかじめ制作する。映像素材の作成には、ビデオカメラによる撮影、コンピュータグラフィクスのアニメーション、映像の編集、加工、合成など、あらゆる映像制作の手法が使われる。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。ライブ映像パフォーマンスの最中に行われている作業は映像の切替のみである。映像の切替とは複数の映像の時間軸上への配置であり、これは準備作業における映像の編集に相当する。

1.4 本研究の目的とアプローチ

従来のライブ映像パフォーマンスでは映像の編集のみが即興的に行われていたのに対し、この映像の編集に加え、映像の加工及び合成も即興的に行えるようにすることが本研究の目的である。これは上で述べた、準備作業の一部である映像の加工や合成を、「手順 2：プレビューと調整」にて行えるようにするものである。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により、VJ が自分の発想をその場で試すことを支援し、VJ の即興性を高める。

映像処理の組み合わせを記述するために、本研究ではデータフロービジュアル言語を採用するというアプローチを取る。このデータフロービジュアル言語の記述を映像再生中であっても行えるようにすることにより、ライブ映像パフォーマンスの最中であっても VJ は任意の映像処理の組み合わせを構築することが可能となる。これを検証するため、データフロービ

ジュアル言語 ImproV を設計、実装し、実運用試験や被験者実験を行い、有効性を確かめた。

同じくライブ映像パフォーマンスの即興性を高めるために、映像型によるパラメータ指定機構を提案し、ImproV へ実装した。映像エフェクタは映像とパラメータを入力として受け取る。映像制作においては、複雑なパラメータを指定するために様々な手法が使われる。例えば、映像の一部分にのみ映像エフェクタを適用するための、図形を描く事による範囲指定や、時間軸に沿ったパラメータの変化を指定するための、タイムライン上へのキーフレーム配置などが行われる。映像によるパラメータの指定機構は、このような複雑なパラメータ指定を、ライブ映像パフォーマンスの最中に行うための機構である。これにより、2 次元的な位置や時間軸に応じたパラメータの変化を指定することが可能となった。

ImproV はライブ映像パフォーマンスの最中に記述され続ける言語である。このため、ライブ映像パフォーマンスが進むに連れ、そのデータフローが複雑になることがある。この複雑さは、VJ がデータフローを理解することを阻害する。これを避けるため、データ調整インタフェースという、データフロービジュアル言語のシンタクティックシュガーを提案し、ImproV へ実装した。

1.5 本研究の成果

映像を即興的に加工及び合成するためのデータフロービジュアル言語：ImproV

ライブ映像パフォーマンスのシステムにおいて、データフロービジュアル言語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の加工や合成を行うことが可能となることを示した。また、ライブ映像パフォーマンスを行う VJ にとって理解しやすい水準のデータフロービジュアル言語を設計した。

複雑なパラメータ指定を行うための映像によるパラメータの指定機構

映像エフェクタのパラメータを映像によって指定する機構を考案した。これにより、フレーム内の位置によって異なるパラメータを同時に指定することや、パラメータをアニメーションさせるといった複雑なパラメータ指定が可能になることを示した。また、VJ は自身の専門分野である映像分野の語彙によってパラメータを加工することが可能となった。

データフロービジュアル言語のシンタクティックシュガーであるデータ調整インタフェース

データフロービジュアル言語において、ノードへの入力を調整するためのシンタクティックシュガーを考案した。このシンタクティックシュガーをデータフロービジュアル言語に導入することにより、ユーザはノードのパラメータ調整や、パラメータとして入力されたデータ値の変更を簡潔に記述できるようになり、データフローを把握し易くなる。本研究では、ImproV にデータ調整インタフェースに実装することによりその有効性を示した。

1.6 本論文の構成

第 2 章では、ImproV の設計、及び、ライブ映像パフォーマンスにおける ImproV の操作について説明し、データフロービジュアル言語を使うことによりライブ映像パフォーマンスが可能であることを示す。また、被験者実験を行ったのでそれについても述べる。第 3 章では、映像エフェクタのパラメータを映像によって指定する機構について説明する。また、ImproV の映像処理エンジンの実装方法について、映像をパラメータとして処理する機構の実装方法とあわせて説明する。第 4 章では、データフロービジュアル言語のシンタクティックシュガーであるデータ調整インタフェースについて説明する。データ調整インタフェースを ImproV に導入することによって、データ調整インタフェースの効果を明らかにする。第 5 章では関連研究について述べ、第 6 章にて結論を述べる。

第2章　映像を即興的に加工及び合成するためのデータフロービジュアル言語:ImproV

本章では、ImproV の設計について説明し、データフロービジュアル言語を動的に編集することによってライブ映像パフォーマンスを行う手法について述べる。また、実際のライブ映像パフォーマンスにおける実運用試験と、専門家による被験者実験を行ったのでこれらについても述べる。

2.1　ImproV の言語設計

ライブ映像パフォーマンスの最中に映像の加工及び合成を行うという目的を実現するために、映像の加工や合成の方法を指定する方法が必要である。VJ は映像の知識をもった専門家であるため、映像機材の接続を表すデータフロービジュアル言語を使って映像の加工や合成の方法を指定することとした。

ImproV は、映像処理の流れをデータフローによって表すデータフロービジュアル言語である。ユーザは、このデータフロービジュアル言語を動的に編集することにより、ライブ映像パフォーマンスの最中に、映像の加工及び合成を行い、多彩な映像を作り出すことが可能である [小林 08b, 小林 08a, KST09]。

2.1.1　VJ にとって理解ができる言語設計

ImproV の対象ユーザは、ライブ映像パフォーマンスを行う VJ である。VJ に理解が可能な言語にするため、ImproV の水準を映像機材の接続構成を表すものとした。映像機材とは、映像ソース、映像ミキサ、映像エフェクタ、映像出力の 4 つである。ImproV のノードはこれらの映像機材を表し、エッジはそれらの接続を表す。これは VJ、すなわち映像の専門家は、普段から複数の映像機材を接続して使っており、映像機材の接続を模したデータフロービジュアル言語を理解できるという仮説のためである。

この設計は、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ということを意図している。

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 のユーザは映像の専門家であり、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る。ImproV ではノードの水準を映像機材としユーザが数式やプログラミングについて意識しないよう配慮されている。

2.1.2 ImproV の文法

以下に例を示して、ImproV の文法を説明する。

ノードとエッジ

まず、単純な例を図 2.1 に示す。図 2.1 には Video File と Preview Screen の 2 つのノードが置かれている。この図を用いてそれぞれの視覚要素を説明する。各ノードの右上の円は出力ポートであり、Preview Screen の VideoIn と書かれた長方形は入力ポートである。

Video File の出力ポートから、Preview Screen の VideoIn へエッジが結線されている。エッジは、映像型のデータか、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。図 2.1 の場合は映像型のデータが流れている。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 Flow[HSMT06] と同様の、出力ポートから入力ポートへ細くなっていく形を採用した。

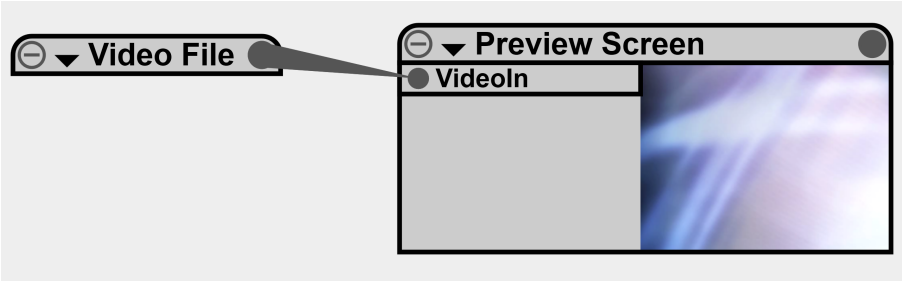


図 2.1: ImproV のデータフローの例

Video File は作成時に指定された映像ファイルをループ再生し、その映像を出力ポートから出力する。Preview Screen は VideoIn に入力された映像を表示するノードである。図 2.1 では Video File からの出力が Preview Screen の VideoIn に結線されている。すなわち、図 2.1 の意味は、Video File からの出力を Preview Screen で表示しているということになる。

次により実際の使用に即したデータフローの例を、図 2.2 に示す。この例は、二つの映像の合成とエフェクトの適用を含む。図 2.2 では二つの Video File によって読み込まれた二つ

の映像が Addition によって加算合成されている。また、二つの映像のうち一方、花の実写映像（図 2.2 左下段の Video File にて読み込まれる）には Addition に入力される前に Brightness が挟まれている。Brightness は明るさを調整するノードであり、図 2.2 では明るさが Slider によって半分程度に調整されている。

Slider や Preview Screen は、固有のユーザインタフェースを持つ。これらのノードの表示は常に更新され、ユーザからの操作も即時にデータフローに反映される。

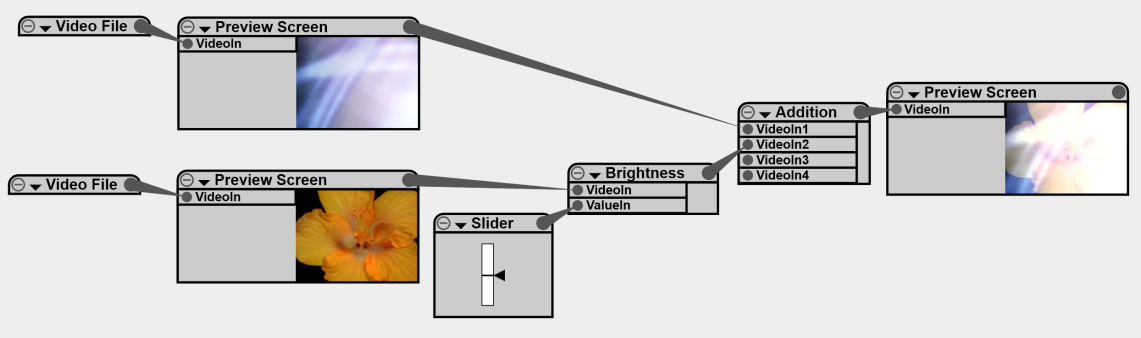


図 2.2: 実用に即したデータフローの例

プレビューと出力

Preview Screen はいくつでも作成でき、ユーザが表示部分をクリックすると表示ウィンドウを開き、ノード名が Output Screen に変更される。表示ウィンドウには表示部分と同じく Preview Screen に入力された映像が表示される。また表示ウィンドウをダブルクリックするとフルスクリーン表示になる。Preview Screen と Output Screen を図 2.3 に示す。図 2.3 上側では、Video File の出力を Preview Screen にて表示している。下側では、同じく Video File の出力を Output Screen にて表示しているが、同時に、“ImproV Output” とタイトルのついたウィンドウにも同じ内容が表示されている。

ライブ映像パフォーマンスにおいては、ImproV を実行する計算機にはデータフローエディタを表示するディスプレイと、観客に提示するためのディスプレイまたはプロジェクタを繋いでおき、Preview Screen の内一つを Output Screen に指定し、その表示ウィンドウを観客へ提示するための出力にてフルスクリーンにする。但し、Output Screen を複数作成することも出来るため、観客へ提示出力を複数にすることも可能である。

データの分岐と合流

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。入力は各入力ポートにつき一つずつしか受け付けない。合流に際しては、映

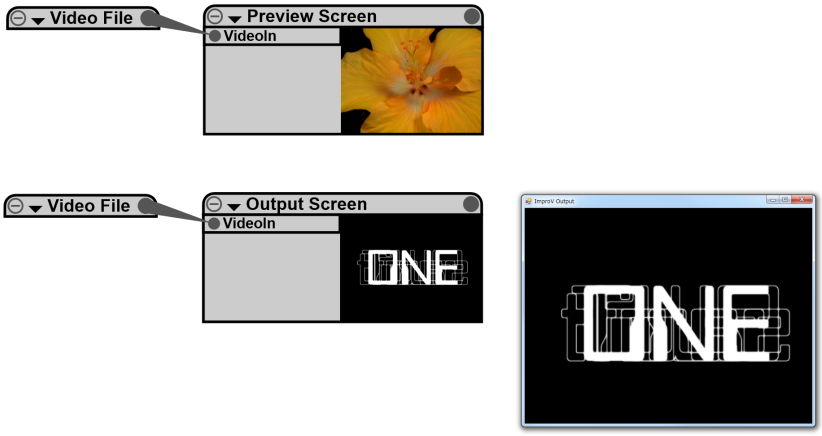


図 2.3: Preview Screen と Output Screen

像値同士の場合は、アルファ合成を行う Mixer や加算合成を行う Addition 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。

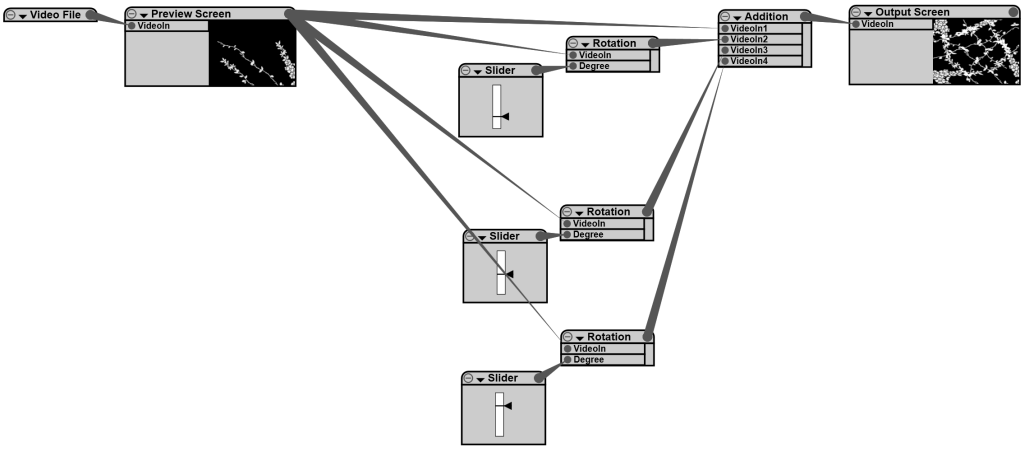


図 2.4: 分岐と合流を行なっている例

分岐と合流を行なっている例を図 2.4 に示す。図 2.4 では、Video File からの出力を Preview Screen にてプレビューしており、その Preview Screen からの出力が 4 つに分岐している。分岐した出力は、内 1 つはそのまま、残りの 3 つは、Rotation によって、約 90°、約 180°、約 270°、回転させられ、Addition によって合流している。

2.1.3 ImproV の編集操作

ノードの生成はウィンドウ上部のメニューバーの Create メニューから行う。ImproV データフローエディタのウィンドウ全体を図 2.5 に示す。またそれ以外にも、映像ファイルやシェーダファイルを別のアプリケーションから ImproV データフローエディタへ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ラベル部分をドラッグするとノードが移動する。

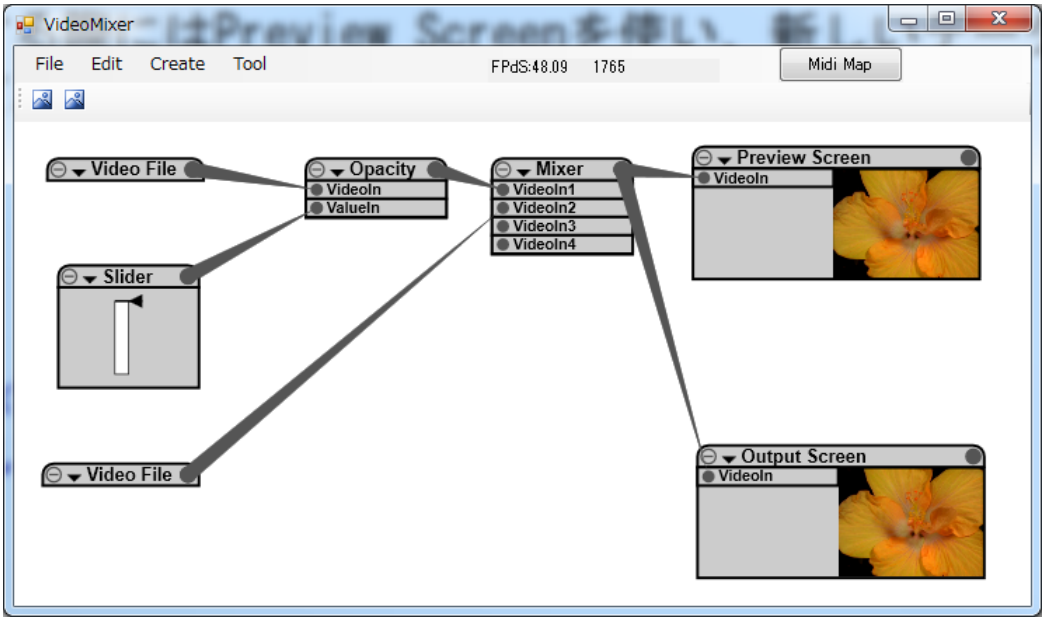


図 2.5: ImrpV のデータフローエディタウィンドウ

結線、すなわちエッジの生成は、マウスのドラッグによって行う。各ノードの出力ポート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、他のノードの入力ポートまでマウスカーソルを移動させ、その入力ポート上でマウスアップを行うと結線が行われる。結線する際、既にその入力ポートに別のノードからの出力が結線されている場合、そのエッジは削除され新しいエッジが生成される。

ノード及びエッジをクリックすると選択され、濃い色で表示される。選択されたオブジェクトは、メニューバーの編集メニューからコピーや削除することが可能である。但しコピーする際、選択したオブジェクト群からなるデータフロー内に、一方の端がノードに繋がっていないエッジが存在する場合、それらのエッジが削除されたデータフローがコピーされる。ImproV では、全てのエッジの両端が必ずノードに繋がっている。

データフローエディタ上のなにもないところをドラッグするとデータフローエディタがスクロールする。また、マウスホイールを動かすとデータフローエディタが拡大または縮小される。ユーザはこれらの機能を使い、データフローエディタを自分が編集しやすい位置や大きさにて表示させる事が可能である。

2.1.4 映像を即興的に加工及び合成する手法

ImproV では、ユーザは動的にデータフローを編集することが可能である。これによりユーザは、あるデータフローの実行中に、そのデータフローに影響を及ぼさずに別のデータフローを構築し、その後にそれらを切り替えることが可能となる。ImproV は映像処理データフロービジュアル言語であるので、前の文の意味するところはつまり、ある映像処理結果を表示中に別の映像処理を構築しそれらを切り替えることができる。この構築と切り替えを繰り返すことにより、新しい映像を次々と作り出し、観客に提示していくことが可能になり、要件 1 を満たすことができる。

また、動的にデータフローを構築し、映像の加工や合成を行うことで、ライブ映像パフォーマンスの最中に 1 つの映像から様々なバリエーションを作り出すことができる。これにより、従来よりその場の雰囲気に合った映像を提示することが可能となり、要件 2 を満たす。

ユーザがデータフローの構築と切り替えを確実に行うために、構築中のデータフローの処理結果をプレビューする機構が必要である。ImproV では、Preview Screen によって、データフローエディタ上にデータフローの処理結果を提示する。ノードの出力がいくつでも分岐可能であり、かつ、Preview Screen はノードであることから、データフロー上のどのノードの出力であっても Preview Screen に入力することが可能である。つまり、データフロー上のどのノードの出力も、データフローエディタ上にてプレビューすることが可能である。

ImproV によるライブ映像パフォーマンス中に VJ は、観客に提示している映像を処理するデータフロー、すなわち Output Screen に接続されているデータフローとは別に、新しいデータフローを構築し、それらのデータフローを映像ミキサノードによってマージすることによって切り替える。また、新しいデータフローを構築する際には Preview Screen を使い、新しいデータフローの処理結果を確認する。

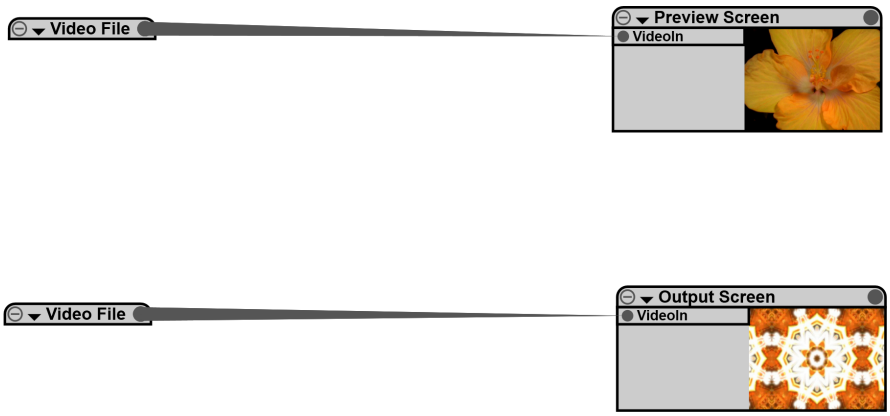


図 2.6: 切り替え手順:初期状態のデータフロー

映像ミキサを表すノードである Mixer を使って二つの映像を切り替える手順を示す。図 2.6 では二つの Video File によって二つの映像が再生されている。下側の Video File は Output Screen へ接続されており、観客に提示されている。VJ は上側の Video File を Preview Screen へ接続し、内容を確認した。この時 VJ は、観客に提示されている映像を、上側の Video File へフェードによって切り替えることを決めた。

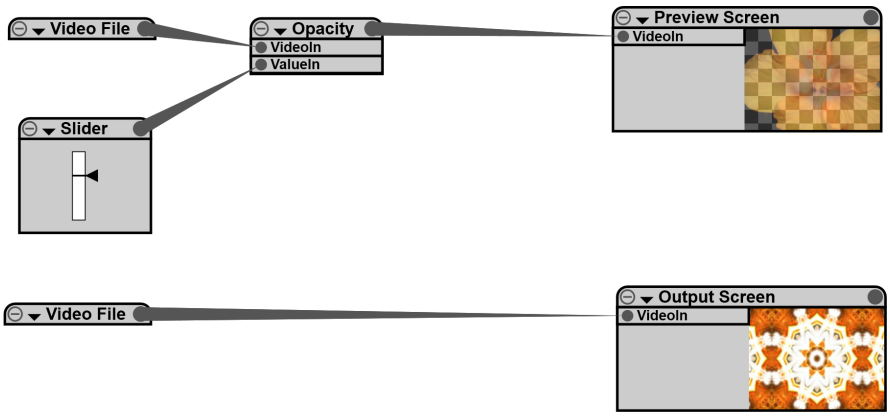


図 2.7: 切り替え手順:Opacity を加えたデータフロー

まず VJ は、図 2.7 に示すように、上側の Video File に Opacity を接続し、Slider によって不透明度を調節できるようにした。

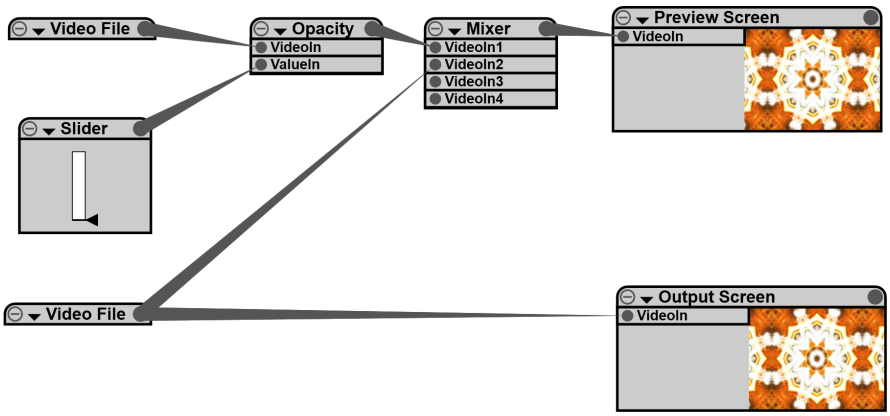


図 2.8: 切り替え手順:Mixer を加えたデータフロー

次に VJ は、図 2.8 に示すように、Mixer を使って二つの映像を重ねた。ここまでの手順の間、VJ は Preview Screen を確認に使っており、観客に提示されている映像に影響はない。

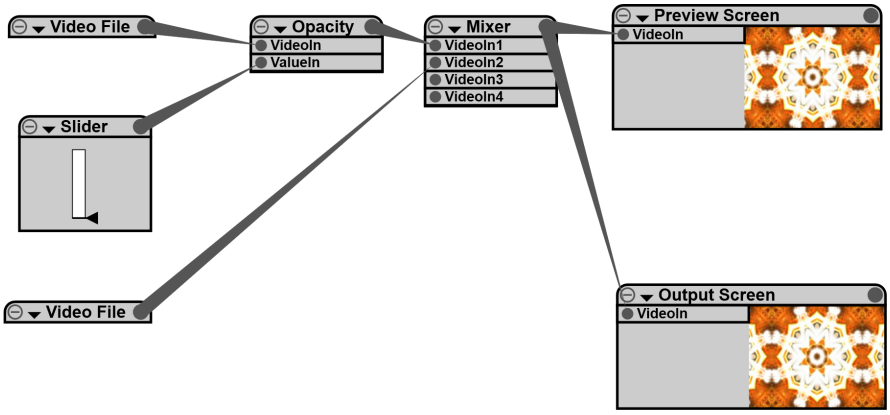


図 2.9: 切り替え手順:Mixer を Output Screen へ接続したデータフロー

最後に VJ は、不透明度を一旦 0 にし、Mixer の出力が観客に提示されている映像と同じことを確認してから Mixer の出力を Output Screen へ接続する。図 2.9 に Mixer の出力を Output Screen へ接続した状態を示す。

この状態から Slider 上のスライダを上に少しずつ上げていくと、上側の映像が少しずつ重なってゆき、最後には切り替わる。図 2.10 に切り替えが完了した状態を示す。

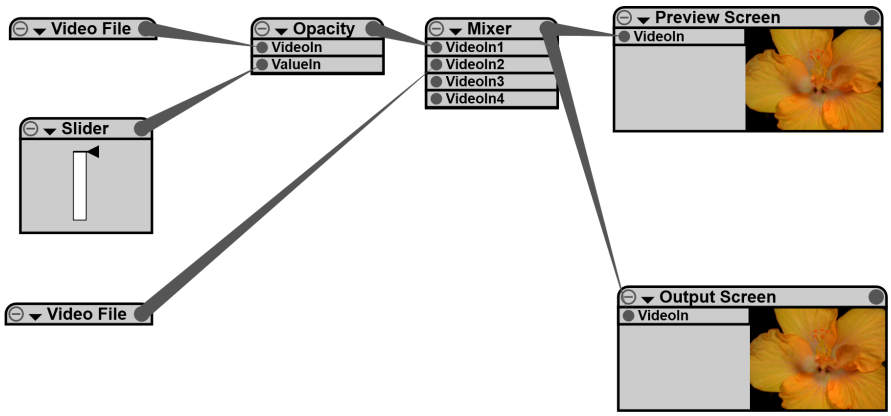


図 2.10: 切り替え手順:切り替え完了後のデータフロー

2.1.5 他のシステムとの連携について

ImproV は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 2：プレビューと調整」を拡張するものである。「手順 1：メディア検索」や「手順 3：ライブ操作」については、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。

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。

ImproV 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 のファイルエク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。

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、音楽の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上の個々の操作部品の種類、例えばノブ、スライダ、ボタン、またそれらの配置は様々である。例えば、シンセサイザを模した Korg 社による nano シリーズ [Avib] や、ミキサを模した M-Audio による TorqXponent[KOR] など、様々なハードウェアコントローラが開発されている。こ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操作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。

ImproV 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。

2.2 実運用試験

ImproV によって、実際にライブ映像パフォーマンスが行えることを確かめるために、実運用試験を行った。ライブ映像パフォーマンスは、筑波大学ジャズ研究会によるコンサートのうち 2 時間行われ、VJ は筆者が務めた。

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 操作用の液晶ディスプレイ、観客への提示用の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 2.11 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、幾つ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、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データフロー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、エッジの方向に沿って非対称な視覚デザインに改良した。

図 2.11: 実際に行われたライブ映像パフォーマンスの様子

2.3 被験者実験 1

VJ にとって ImproV が使いこなせるかどうかの確認、および、ImproV の操作性の評価を目的として被験者実験を行った。被験者はプロフェッショナル（出演料を受け取る）VJing の経験を持つ、22〜33 歳の男性 5 名である。また、実験では比較や確認のために Adobe After Effects を使用したため、被験者全員が Adobe After Effects の使用経験があることを確認した。被験者にはまず、映像制作、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。実験者は、被験者に付録 A の質問票を一枚づつ提示し、教示を行った。

実験 1 の目的は、VJ にとって ImproV が理解しやすいかどうかを確認することである。被験者には、ImproV について説明をせずに ImproV のデータフローを提示し、そのデータフロー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 にて同様の映像合成の構成を再現してもらった。結果は全員問題なく正答し、ImproV のデータフロー

を理解できていることが確認できた。さらに、事前アンケートでの「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。」という問いに対して付録 B に示すように被験者全員がデータフロー図を描いたこと、事後アンケートの結果、コメントからも、VJ が持っている映像機器に関する知識によって ImproV が十分使いこなせるものであることが分かった。

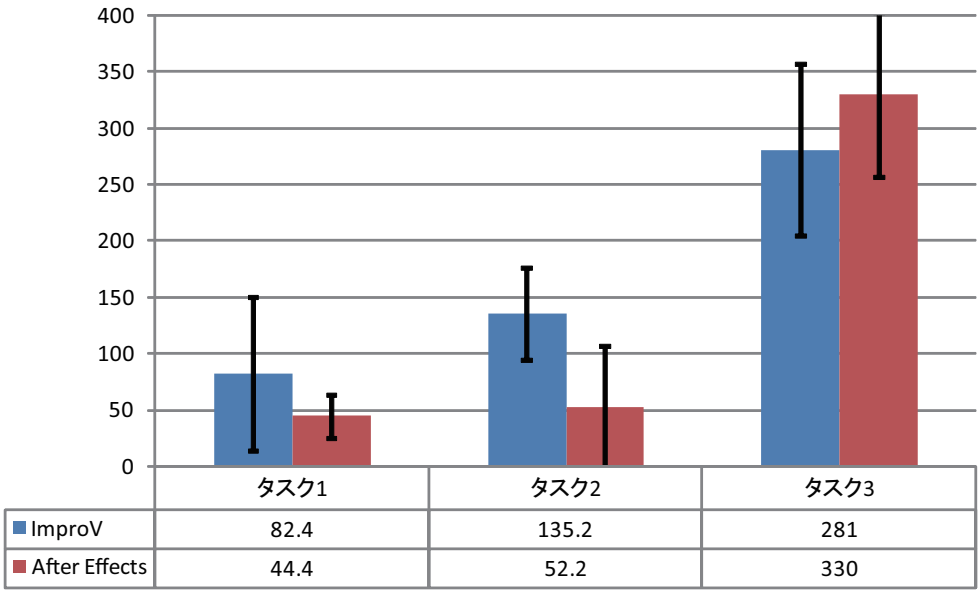
実験 2 の目的は、VJ にとって ImproV の操作を問題なく行えるかどうか、及び、操作性を確認することである。実験 2 では、Adobe After Effects 上の映像合成の構成を見て、同様の構成を ImproV にて再現する、または、ImproV 上の構成を Adobe After Effects 上で再現するというタスクを行ってもらった。タスクは以下に示す 3 種類を用意した。

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。

タスクを行う際、実際の VJing を想定し、メイン出力の映像が滑らかに切り替わるよう留意してもらった。結果、被験者全員が、メイン出力の映像が滑らかに切り替わるように全てのタスクを完了することができた。それぞれのタスクに要した時間を図 2.12 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 2.12: 実験 2 の各タスクに要した時間（秒）の平均、およびそれらの分散

表 2.1: 実験 2：タスクにかかった時間の分散分析

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			

タスクに要した時間に対して分散分析を行った結果を表 2.1 に示す。タスク 2 において、二つのシステムに有意差が見られた（$P = 0.0201 < 0.05$）。

ImproV と Adobe 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（$p < 0.1$）。

表 2.2: 実験 2：タスクの主効果における多重比較

	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

タスクの主効果についての多重比較を表 2.3 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（$p > 0.05$）。

表 2.3: 実験 2：交互作用における単純主効果

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（Adobe After Effects）	264668.4000000	2	132334.2000000	30.916	0.0000
error		16	4280.4083333		

交互作用における単純主効果を表 2.3 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、Adobe 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 2.13 に示す。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。Adobe 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。

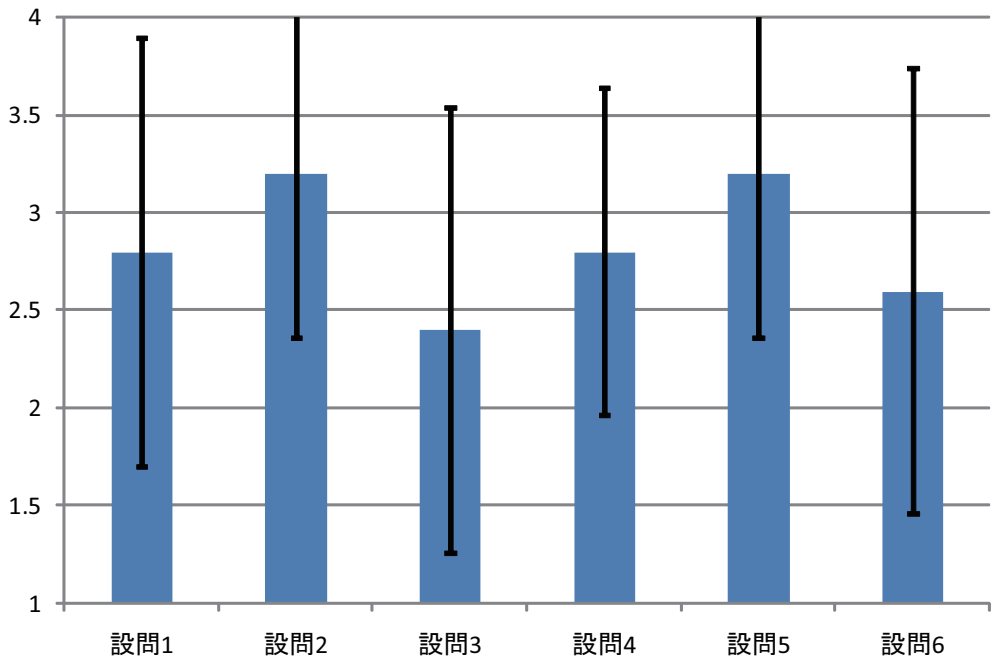


図 2.13: 事後アンケートの結果

これらの結果は、Adobe After Effects より ImproV のほうが、操作に時間を要することを示唆する。また、事後アンケートからも、操作性、特に結線時の操作に問題がある旨のコメントが寄せられた。この結果を踏まえ、現在の実装では、以下の修正を行った。

- ノードの出力ポートを大きくする
- 入力ポートの判定を広くし、視覚フィードバックを提示する
- 入力ポートを持つノードの生成時に選択中のノードがあれば、生成するノードの一つ目の入力ポートと、選択中のノード出力ポートを自動的に接続する

さらに今後は、システムをテーブルトップに移植し、マルチタッチやジェスチャなどを使った迅速な結線方法を導入することが考えられる。

第3章　複雑なパラメータ指定を行うための映像によるパラメータの指定機構

本章では、映像型と映像型によるパラメータ指定について説明し、映像型によるパラメータ指定によってどのようなことが可能になるかを、例を示して説明する。その後、映像型によるパラメータ指定の実装を含む、ImproV の映像処理エンジンの実装について説明する。最後に、この映像処理エンジンの処理速度について評価実験を行ったので、その評価実験について述べる。

3.1　映像型と映像型によるパラメータ指定

映像処理データフロービジュアル言語は、処理対象である映像型データと、映像エフェクタのパラメータを指定するためのパラメータデータを扱う。パラメータデータには多くの場合、スカラ型データやその組み合わせが使われる。ImproV では処理対象の映像を表す映像型と、映像エフェクタのパラメータを指定するためのスカラ型の二つのデータ型を持っていた。映像型のデータは実際には映像のフレーム画像であり、ImproV のデータフローは各フレームタイミングに要求駆動によって評価される。つまり、ImproV は画像の処理を連続して繰り返すことによって映像の処理を行う。スカラ型は 0 から 1 の浮動小数点であり、スカラ型のデータも映像型と同じタイミングで処理されていた。

著者は、映像エフェクタのパラメータを映像型によって指定する仕組みを開発し、ImproV 上に実装した [小林 11]。パラメータを映像型によって指定することにより、フレーム内の位置によって異なるパラメータを同時に指定することや、パラメータをアニメーションさせるといった複雑なパラメータ指定が可能になる。

また、映像型データを使ってスカラ型データを表現できるため、スカラ型を廃しデータ型を映像型のみにする事が可能である。データ型が映像型のみであることによって、ユーザの混乱が軽減される。プログラミング経験のない VJ にとって、ライブ映像パフォーマンスの最中に、複数種類のデータ型を区別して扱うのは難しい。これは被験者実験 1 を通して、被験者であった VJ から得られたコメントからわかった。ここから ImproV で扱うデータ型を映像型のみとした。

映像エフェクタのノードはパラメータとして入力された映像型データの各ピクセルの輝度値を取得し、その輝度値をそのピクセル位置におけるパラメータとして処理を行う。すなわち、映像エフェクタは式 3.1 における、関数 f として定義される。関数 f はフレーム画像サ

イズの全てのピクセル (x, y) に対し呼ばれる。Out_{xy} は結果画像の座標 (x, y) における色、n は入力ポートの数であり、V_n は n 番目の入力ポートに入力されたフレーム画像である。

$$Out_{xy} = f(x, y, V_1, V_2, V_3, \cdots, V_n) \tag{3.1}$$

映像型によって不透明度を指定する例を示す。図 3.1 では、透明度を変更するノードである Opacity の ValueIn にグレースケールの映像を入力している。Preview Screen の表示において、格子模様は透明であることを示す背景である。ValueIn に入力された映像の暗い部分が透明になっていることが確認できる。

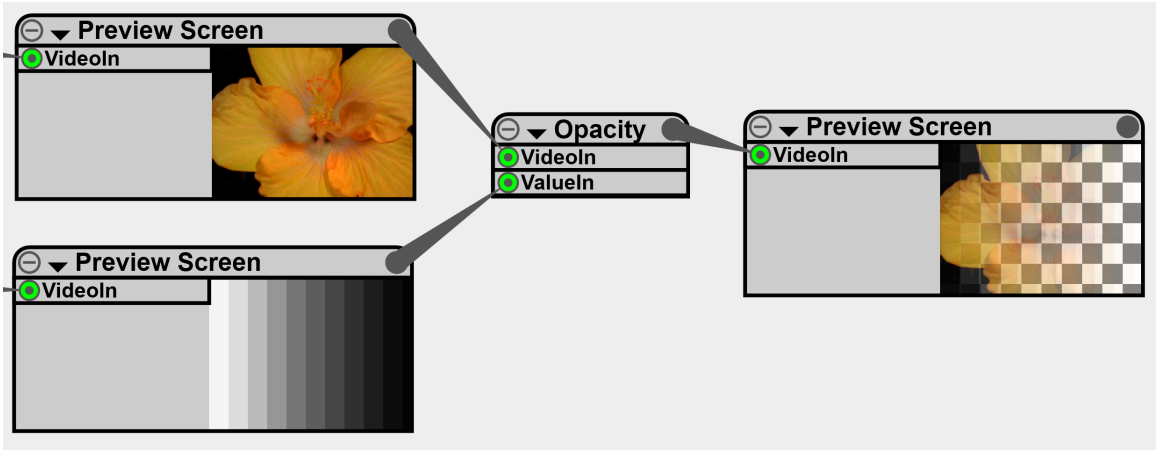


図 3.1: 映像型によって透明度を指定する例

別の例として、輝度や回転角度を映像型によって指定する例を示す。図 3.2 では、Brightness によって輝度の変更、Rotation によって回転がそれぞれ行われている。データは同じであり、VideoIn には三角形がグリッド状に配置された映像、ValueIn にはグレースケールの映像がそれぞれ入力されている。ValueIn に入力された映像の輝度によって、Brightness では輝度値、Rotation では回転角度が指定されていることが確認できる。

フレーム全体に同じパラメータでエフェクトを適用するには全ピクセルが同じ輝度値の画像を入力する。Slider は、全ピクセルが白（RGB=1.0,1.0,1.0) でありアルファ値がスライダーによって指定された値である画像を出力する。これにより従来のスカラ型と同等のデータを表す事ができ、スカラ型と同様の処理は映像型によって行える。

3.2 複雑なパラメータ指定の例

この節では映像型による複雑なパラメータ指定の例を 3 つ挙げる。

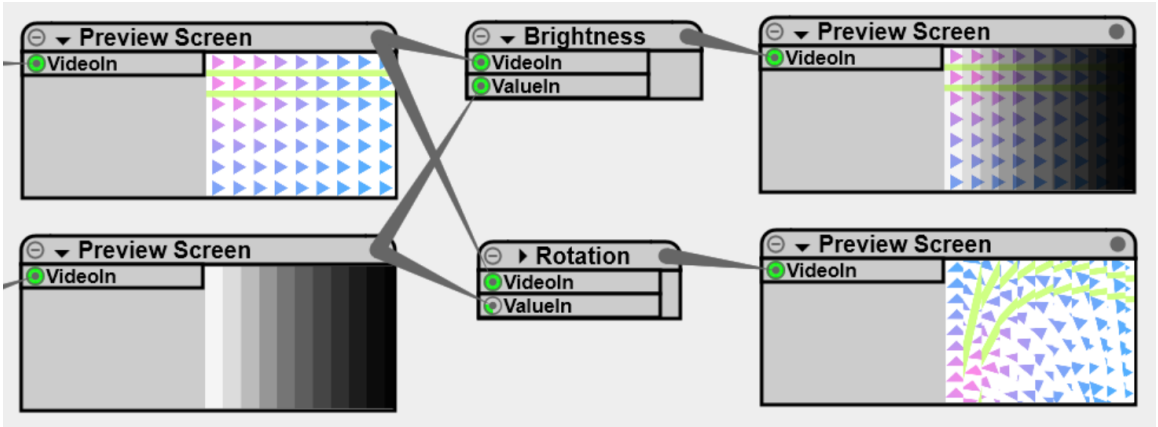


図 3.2: 映像型によって輝度と回転角度を指定する例

3.2.1 アニメーション

ある映像によってパラメータを指定することは、その映像の動きに沿ってパラメータをアニメーションさせることを可能にする。

例として、Rotation による回転角度のアニメーションを示す。図 3.3 では、Rotation の ValueIn にグラデーションが左から右へ移動する映像が入力されている。

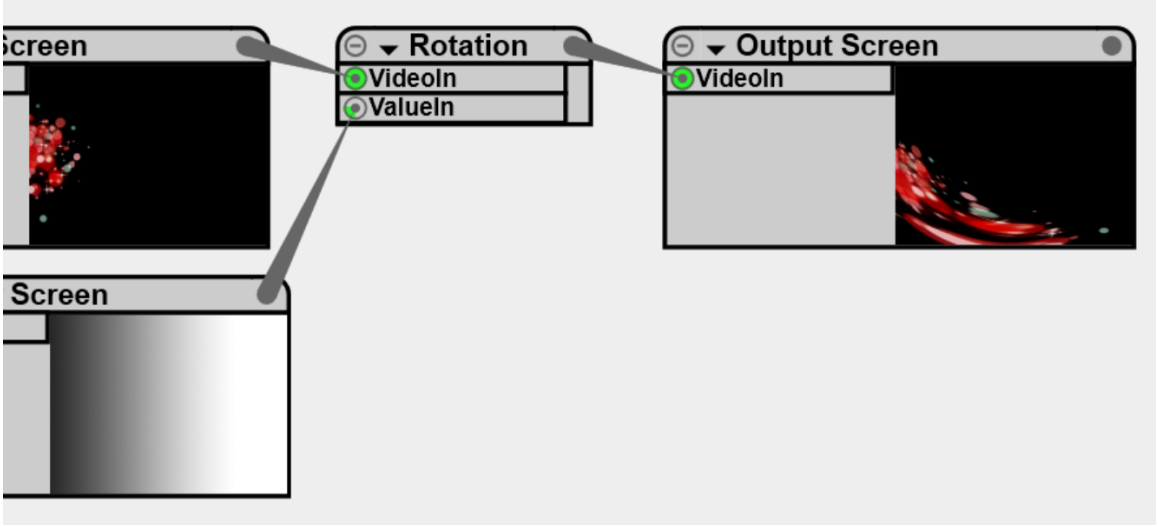


図 3.3: アニメーションの例

図 3.4 は図 3.3 における Rotation の入力と出力をそれぞれ 10 フレームを左から右に並べたものである。上から、VideoIn への入力映像、ValueIn への入力映像、そして出力映像のそれぞれ対応する。ValueIn への入力映像が変化するに従って出力画像の回転角度が変わってゆく

のが確認できる。

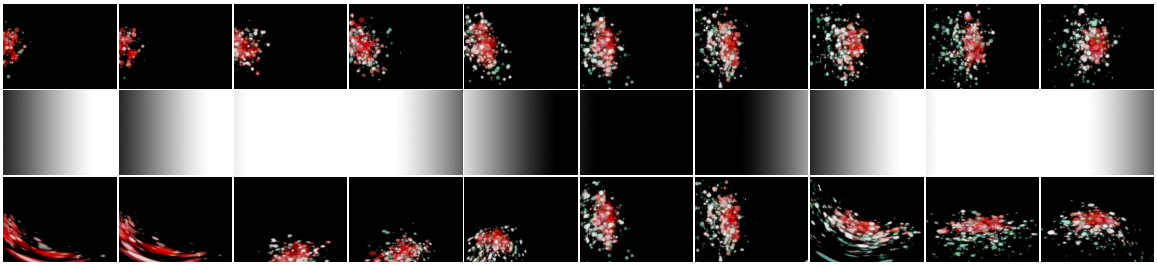


図 3.4: 図 3.3 の入力と出力

ここで ValueIn へ入力しているグレースケールの映像は依然としてあらかじめ用意する必要があるが、このようなグレースケールの映像は様々なパラメータを指定する用途に使え、汎用的であるといえる。したがって、パラメータを指定するためのグレースケール映像をライブラリとして提供することは実用的であるといえる。

3.2.2 音響信号の可視化

音響の可視化はライブ映像パフォーマンスの AudioIn はサウンドカードから入力された音響信号をグレースケールの映像として可視化するノードである。まず、AudioIn は固定幅の音響信号バッファを持っており、サウンドカードから入力された音響信号はこの音響信号バッファに蓄えられる。

入力ポート Value に入力された映像型データに応じて、音響信号バッファ内の数値をグレースケールのピクセル値として配置する。配置方法は、Value に入力された映像の輝度値が低いほど古い音響信号、輝度値が高いほど新しい音響信号の値をそれぞれのピクセルに配置する。

図 3.5 では、AudioIn に水平のグラデーションの映像が入力されているため、AudioIn は水平方向に音響信号を配置している。この AudioIn の出力を Translation の Y に入力する事により、水平縞模様の画像が y 軸方向に歪められ、波形を描画している。

図 3.6 では、AudioIn に円形グラデーションが入力されており、AudioIn は放射状に音響信号を配置している。AudioIn からの出力は環状の筋となる。Blur は画像をぼかすノードであり、この場合は AudioIn からの出力をなめらかなものにしている。これを Scale の ValueIn に入力することによって、波紋状の効果になる。

3.2.3 カメラによるライブ操作

Camera は USB カメラからの映像を取得するノードであり、この出力をパラメータを指定として用いることにより、VJ はカメラを通して直接パラメータをライブ操作することが可能である。

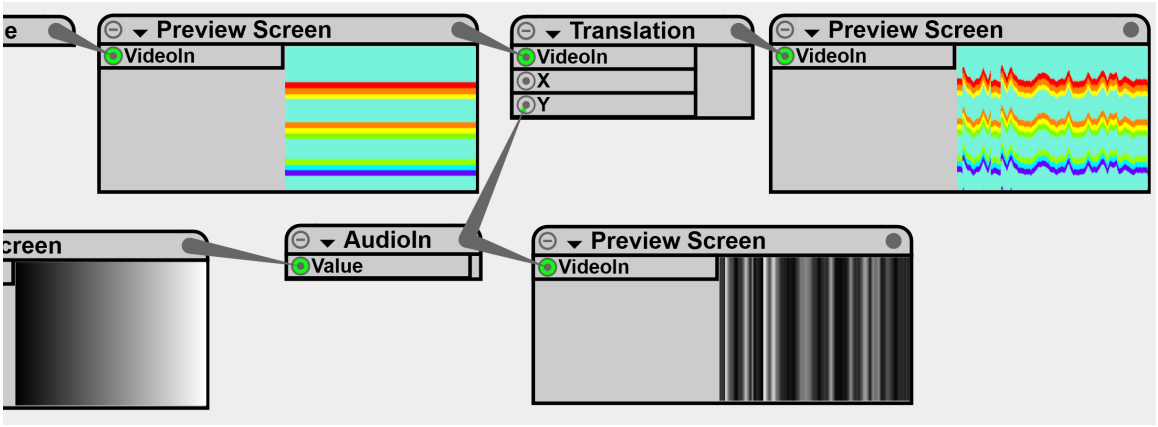


図 3.5: 音響信号の可視化の例

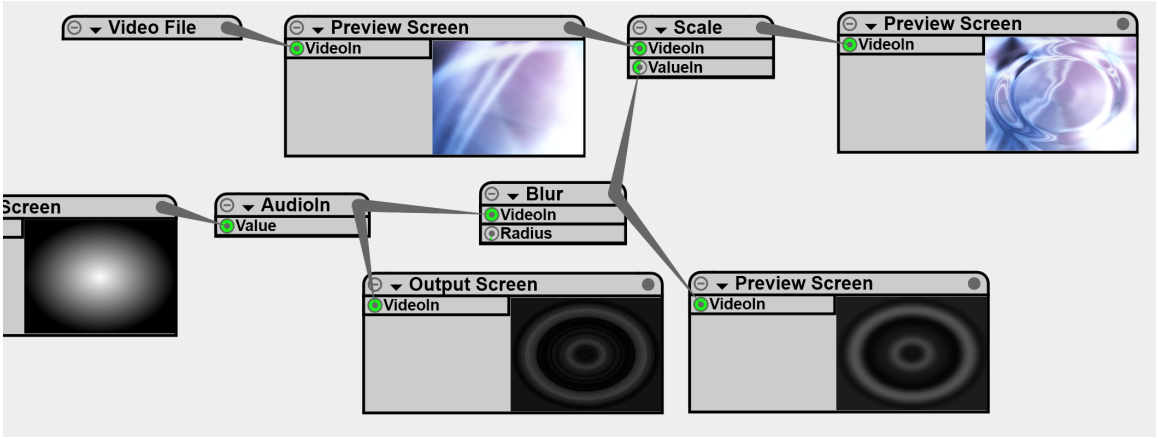


図 3.6: 複雑な音響信号の可視化の例

図 3.7 では、Camera の出力が、Scale の ValueIn に繋がれている。Scale は ValueIn に入力された映像の輝度値に応じて、VideoIn に入力された映像を拡大・縮小するノードである。VJ が指を使って USB カメラのレンズを半分ほど隠しており、Camera の出力には、指が半分ほどと、指がレンズに接触している部分が影となって写っている。Scale の出力は指が接触している部分が拡大され、それ以外の部分は縮小されている。

このように、VJ はカメラを使って映像エフェクタが適用される部分をその場で指定することが可能になる。また、指を動かすことによってパラメータをアニメーションさせることも可能であり、例えば音楽のリズムに合わせたアニメーションをその場で創りだすといったことが可能となる。

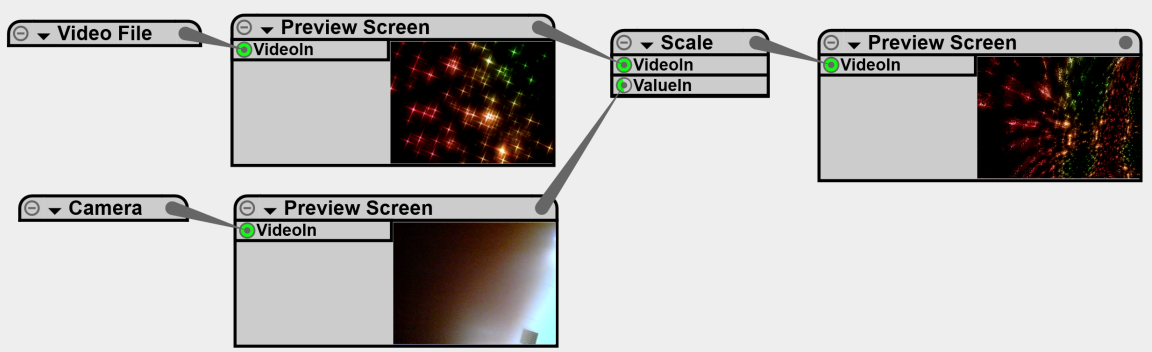


図 3.7: USB カメラによって縮尺を操作している様子

3.3 映像処理エンジンの実装

映像型によるパラメータ指定は、スカラ型よりも計算量が多くなる。さらに、ImproV は、ライブパフォーマンスで利用されるシステムであり、内部の映像合成処理はリアルタイムに行う必要があるこれらの理由から ImproV では、映像処理に GPU を用い高速化している。映像型のデータを GPU のテクスチャとしてグラフィックカード上に確保し、ノードの処理を GPU 上のピクセルシェーダにて行うことによって処理速度を確保している。以下ではまず、データフローの構造と評価について説明し、次に映像処理について述べる。

3.3.1 データフローの構造と評価

ImproV は C#言語によって Microsoft .NET Framework 上に実装されている。また、映像処理および、データフローエディタの描画には SlimDX を通して Direct3D 10 を使っている。このため、実行には Direct3D 10 をサポートするグラフィックカードが必要である。

Node クラスはデータフロー上の全てのノードの親となる抽象クラスである。このクラスを継承し、evaluationProcess というメソッドをオーバーライドすることにより、映像の処理を記述する。OutputNode クラスは Preview Screen を表すクラスであり、Node クラスを継承している。データフロー評価が OutputNode クラスを起点に始められるため、他の Node サブクラスとは区別して管理される。Input クラスは各ノードの入力ポートを表す。これは各 Node オブジェクトが必要に応じて生成する。Dataflow クラスはデータフローの構築や評価を行う。Edge クラスは結線の情報を管理する。これは Dataflow クラスが必要に応じて生成する。

Dataflow オブジェクトはそのデータフローに存在する全ての Node、Input、Edge、そして OutputNode オブジェクトを管理している。データフローの構築は、Dataflow オブジェクトの Add（ノードをデータフローに追加する）、Plug（あるノードの出力ポートと別のノードの入力ポートを結線する）、Merge（別のデータフローと併合する）などのメソッドによって行われる。

次にデータフローの評価について述べる。映像とはフレーム画像のシーケンスである。ImproV

ではデータフローの評価時に画像の処理が行われる。この評価を毎フレーム繰り返す事によって映像の処理を行う。この評価時に、ImproV はデータフローを、各 OutputNode オブジェクトをルートとしたツリーとみなし、それをポストオーダーによって走査する。このとき、データフローはツリーではなく有向グラフであるため、重複して評価が起こる。この評価の重複を回避するために各ノードは、自分が最後に評価を行った際のタイムスタンプと結果をキャッシュしてある。各ノードは、評価が要求されたときのタイムスタンプとキャッシュしてあるタイムスタンプを比較する。二つのタイムスタンプが同じであればキャッシュしてある内容を返し、それ以上は評価要求を伝播しない。

詳細を見ると、まず、Dataflow オブジェクトの Evaluate メソッドが呼び出される。そして、Dataflow オブジェクトの Evaluate メソッドはそのデータフローに存在する全ての OutputNode オブジェクトに対して Evaluate メソッドを呼び出す。この際、Dataflow オブジェクトは評価時点のフレーム番号を引数に渡す。OutputNode を含む、全ての Node オブジェクトの Evaluate メソッドでは、まず、渡されたフレーム番号が、最後に呼ばれた時点のフレーム番号 lastFrame と同じかどうかを調べる。同じである場合、キャッシュしてあるテクスチャを返す。異なる場合は lastFrame を更新し、次に、自分が保持している全ての Input クラスに結線されているノードに対して Evaluate メソッドを呼び出す。その後、つまり自分が処理すべき入力がそろってから、自分の evaluationProcess を呼び出し、結果のテクスチャを返す。

ImproV ではデータフローエディタによって、動的にデータフローが変更される。このため ImproV は、データフローエディタを含むアプリケーションのイベント処理、データフロー評価、データフローエディタの描画、という順序で処理し、ユーザの操作が次のフレームで必ず反映される。また、データフローを変更していく過程では、必要な入力が指定されていない状態が起こりうる。ImproV は、このような状態のデータフローを評価しても実行を止めないために、何も結線されていない入力ポートは、そこに透明なフレーム画像が入力されているものとして評価を行う。

3.3.2 映像処理

上述の通り、ImproV のデータフローで扱うデータ型は、我々が映像型と呼ぶ、映像のフレーム画像である。この映像型は実際には Direct3D 10 のテクスチャを使って実装されている。

そして、複数の映像を合成するミキサノードや映像を加工するエフェクトノードは、受け取ったテクスチャをポリゴンに貼り付け、それをそれぞれの方法でレンダリングし、そのレンダリング結果を別のテクスチャに描画する。このようにレンダリング結果をディスプレイに転送せず、テクスチャに保存するオフスクリーンレンダリングと呼ばれる方式を採ることにより、各映像エフェクト間のデータ受け渡しが GPU 内において行われ、メインメモリへの画像の転送が起きないため、高速に映像を合成することができる。

このデータフローに基づく映像処理エンジンは別の.NET Framework アプリケーションに組み込み、利用することが可能である。以下に、ImproV の映像処理エンジンの使い方を例を示して利用方法を説明する。

ソースコード 3.1:

```
1  //データフローの生成
2  Dataflow graph = new Dataflow();
3
4  //データフローに追加するノードの生成
5  Node source = new BitmapNode("test.bmp");
6  Node effect = new EffectNode("test.hlsl");
7  OutputNode output = new OutputNode();
8
9  //生成したノードをグラフに追加する
10 graph.Add(source);
11 graph.Add(effect);
12 graph.Add(output);
13
14 //source>>effect>>outputと結線する
15 graph.Plug(source,effect.Inputs[0]);
16 graph.Plug(source,output.Inputs[0]);
17
18 //outputの中身を表示するウィンドウを表示する
19 output.Show();
20
21 //0フレーム目としてデータフローを評価する
22 graph.Evaluate(0);
23
24 //effectの評価結果を取り出す。
25 Texture2D myTexture = effect.Evaluate(0);
```

ソースコード 3.1 において、BitmapNode は画像を読み込み出力するノード、EffectNode は High Level Shader Language（HLSL）のソースコードを映像エフェクトとして読み込むノードである。15 行目では、source からの出力を effect の 1 番目の入力ポートに結線し、effect からの出力を output の 1 番目の入力ポートに結線している。ソースコード 3.1 が実行されると、ウィンドウが生成され、“test.bmp” の内容に “test.hlsl” の画像エフェクトを適用した画像が表示される。また、Node クラスにも Evaluate メソッドが用意されており、25 行目のように任意のノードの評価結果をテクスチャとして取り出すことも可能である。

3.4 プラグインシステムの実装

ノードの種類は様々なものが考えられる。さらに、ある程度プログラミングの知識を持った VJ は自身の手によって新しいノードを作りたいと思うかもしれない。このため、ノードはプラグインとして追加可能とすることで拡張性を確保した。

多くの映像エフェクトは一回のレンダリングパスで実現できるので、そのような映像エフェクトノードであれば、HLSL のみによって実装することが可能になるように実装した。ImproV の映像エフェクトノードや映像ミキサノードの大半は、この方法で実装されている。

より複雑なノードは Node クラスのサブクラスとして実装することが出来る。そのため、.NET Framework の知識を持ったプラグイン開発者は、SlimDX を通して Direct3D 10 の全ての機能を利用することや、.NET Framework の全ての機能を利用したノードを実装することが出来る。

この方法では、独自の GUI をもった映像エフェクトや.NET の機能を全て活用した映像エフェクトを作り、ImproV を拡張することができる。例えば、映像ファイルを読み込むノードや、スライダを持つ UI ノードなどはこの方法を使って実装される。

さらに、ImproV は、VJ がライブ映像パフォーマンスを行っている最中に映像合成に手を加えられることが求められる。このため、動的にデータフローを変更することができ、その変更はリアルタイムに映像合成処理に反映される必要がある。データフローエディタ上で作成されたノードは ImproV に動的にリンクされる。HLSL にて実装されたノードは、データフローに追加された時点でコンパイルされる。Node クラスを継承して実装されたノードは予め.NET アセンブリとしてコンパイルしておき、ImproV の起動時に読み込まれる。

以下では、まず HLSL を使う実装方法について説明し、その次に、Node を継承する実装方法について説明する。

3.4.1 HLSL を使った動的なプラグイン

ソースコード 3.1 にて使われていた EffectNode クラスを使うと、HLSL ソースコードを動的に読み込み、コンパイルすることが出来る。ユーザは ImproV を起動したまま、任意のテキストエディタによって HLSL ソースコードファイルを作成し、そのファイルを ImproV のデータフローエディタ上にドラッグアンドドロップすることにより、EffectNode を生成することができる。このため、映像エフェクトの開発者は ImproV を起動したまま試行錯誤に基づいた開発を行うことができる。

EffectNode クラスのコンストラクタは、HLSL で書かれたエフェクトファイルのソースコードを動的にコンパイルし、Direct3D 10 のエフェクトオブジェクトを生成する。そして、読み込まれたソースコードファイルのグローバルスペースにテクスチャ型の変数が宣言されていると、それらの変数を入力ポートとして自動的に登録する。

EffectNode オブジェクトは評価時に、まず自分の入力ポートに入力されたテクスチャを、上記のテクスチャ型変数に代入する。次に、レンダリングする際にテクスチャを貼り付けるオブジェクトとして、四角形の平面ポリゴンを構成する頂点をエフェクトオブジェクトへ渡す。ここでテクスチャや頂点、エフェクトオブジェクトは Direct3D 10 で提供されるクラスであり、GPU 内に配置されているため、評価時のデータの受け渡しは GPU 内で行われる。その後、生成したエフェクトオブジェクトのレンダリングパスを実行する。ここで、レンダリングパスとは、頂点シェーダ関数、ジオメトリシェーダ関数、ピクセルシェーダ関数、ラスタライザの設定、出力マージャの設定の組み合わせである。このうち、頂点シェーダ関数、とピクセルシェーダ関数は必須であり、何らかの引数や返り値の型が合う関数を指定しなければならない。我々は HLSL ライブラリ ImproVCoreLib.hlsl を提供しており、その中で MapVertex 関数という頂点シェーダ関数を用意している。多くの映像エフェクトや映像ミキサは、MapVertex 関数を頂点シェーダ関数として指定し、ピクセルシェーダを記述するだけで実現できる。現在、ImproV では 10 種類のノードがこの方法で実装されている。

以下では、二つの入力を受け取る映像エフェクトの実装を例として説明する。この映像エフェクトは、一つの入力ポートに入力された映像の輝度を、もう一つの入力ポートに入力さ

れた映像に応じて変更する。

図 3.8: 例として実装する映像エフェクトの適用例 左：加工対象として入力された画像 中央：パラメータとして入力された画像 右：結果として出力された画像

図 3.8 にこの映像エフェクトの適用例を示す。この例では、図 3.8 左の画像を、図 3.8 中央に応じて、図 3.8 右の結果を出力している。図 3.8 中央の暗い部分が、図 3.8 右では暗くなっている。ソースコード 3.2 にこの映像エフェクトのソースコードを示す。

ソースコード 3.2:

```
1  #include "../system/ImproVCoreLib.hlsl"
2  Texture2D VideoIn;
3  Texture2D ValueIn;
4
5  float4 Effect( PipelineVertex input ) : SV_Target {
6
7    float4 color = VideoIn.Sample(TextureSampler, input.texCoord);
8    float value =
9      Luminance(ValueIn.Sample(TextureSampler, input.texCoord));
10
11   color.r *= value;
12   color.g *= value;
13   color.b *= value;
14
15   return color;
16 }
17
18 technique10 Render{
19   pass P0{
20     SetGeometryShader( NULL );
21     SetVertexShader( CompileShader( vs_4_0, MapVertex() ) );
22     SetPixelShader( CompileShader( ps_4_0, Effect() ) );
23   }
24 }
```

1 行目では、我々が提供するライブラリである ImproVCoreLib.hlsl がインクルードされている。ソースコード 3.2 の中にて使われているものとしては、Luminance 関数、MapVertex 関数、PipelineVertex 構造体がこのライブラリに含まれる。

2 行目と 3 行目では、Texture2D 型の変数が宣言されている。これらは EffectNode のコンストラクタが、HLSL ソースコードをコンパイルする際に入力ポートとして登録し、データフローエディタ上では変数名が表記される。図 3.9 にこの映像エフェクトのデータフローエディ

タ上における見た目を示す。ファイル名である “Sample” がノード名として、また、VideoIn、ValueIn がそれぞれ入力ポートとして表示されていることが確認できる。

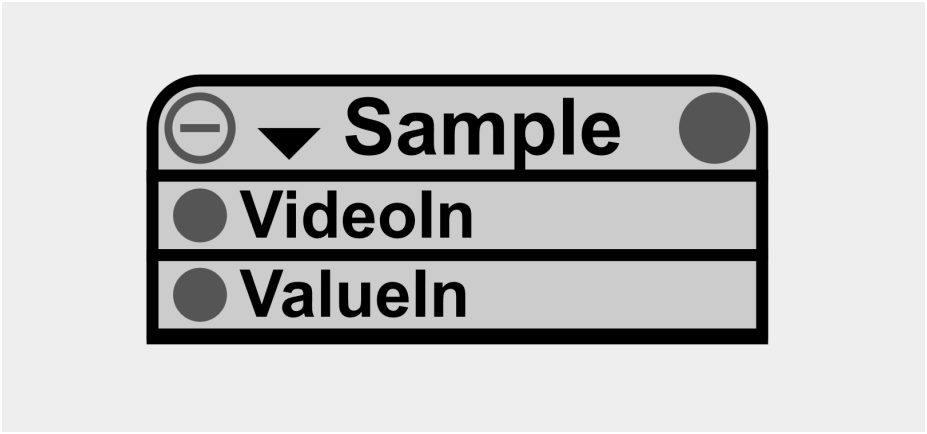


図 3.9: 映像エフェクト Sample のデータフローエディタ上の見た目

EffectNode クラスは、評価時に 20 行目で P0 として宣言されているレンダリングパスを 1 フレームに付き一回実行する。

23 行目では頂点シェーダとして MapVertex 関数が指定されている。この関数は ImproV-CoreLib.hlsl にて提供される関数であり、EffectNode クラスから渡される頂点データをピクセルシェーダに渡される PipelineVertex 構造体に変換する。PipelineVertex は頂点座標 pos と対応するテクスチャ座標 texCoord をプロパティとして持っている。MapVertex 関数は左上隅が (0.0,0.0)、右下隅が (1.0,1.0) とそれぞれなるように PipelineVertex を生成する。

24 行目ではピクセルシェーダとして 5 行目に宣言される Effect 関数が指定されている。GPU は出力画像の各ピクセルごとに、そのピクセルに対応する位置の頂点データを引数にして、ピクセルシェーダを呼び出す。ここでは頂点シェーダとして MapVertex 関数が指定されている。MapVertex 関数は画面の描画領域全体に四角形のポリゴンを投影する。この為、Effect 関数の引数 input の texCoord の x,y にはそれぞれ 0.0〜1.0 までのテクスチャ座標が入っている。

7 行目では、Effect 関数が呼び出されたピクセルに対応する、VideoIn テクスチャのピクセル値を取得し color に代入している。8、9 行目では同様に ValueIn テクスチャのピクセル値を取得し、Luminance 関数によってそのピクセルの輝度値を求め value に代入している。この Luminance 関数も ImproVCoreLib.hlsl にて提供される。このように、Luminance 関数によってピクセル値をスカラ値に変換することにより、映像型によるパラメータ指定を実現している。

11 行目から 13 行目ではピクセル値 color と、value 値を乗算している。ここでは、RGBA 全てに対して乗算を行うと不透明度まで変化してしまうため、説明のために RGB それぞれに対してのみ乗算を行っている。

この 7〜16 行目のピクセルシェーダ関数の内容を変更することで様々な映像エフェクトを実装できる。ソースコード 3.3 は図 3.2、および、図 3.3 にて示した回転エフェクトのピクセルシェーダ関数である。

ソースコード 3.3:

```
float4 Rotation(PipelineVertex input) : SV_Target
{

  float Val=
    Luminance(ValueIn.Sample(TextureSampler, input.texCoord)) * 2 * PI;
  float2 TCoord;

  input.texCoord -= 0.5;
  float3x3 rotationMatrix={
    cos(Val),-sin(Val),0,
    sin(Val),cos(Val),0,
    0,0,1
  };
  TCoord = mul(rotationMatrix,input.texCoord);
  TCoord += 0.5;

  return VideoIn.Sample(TextureSampler, TCoord);
}
```

ソースコード 3.3 では ValueIn の輝度値に応じて入力されたテクスチャ座標を変換し、VideoIn からピクセルを取得する際の座標値を変更することにより回転を実現している。

3.4.2 Node の継承によるプラグイン

HLSL のみでは実装できない機能や映像エフェクトに関しては、Node クラスを継承したクラスを作ることにより実装する。このようなノードの例としてすでに実装されているものとしては、映像ファイルを読み込み再生するノード VideoFileNode や、USB カメラからの映像をキャプチャするノード CameraNode クラスが挙げられる。

新しいクラスの定義に最低限必要なことは、evaluationProcess メソッドをオーバライドすることにより、データフロー評価時のふるまいを記述することである。このメソッドはデータフロー評価時に呼び出される。呼び出される際には、フレーム番号が整数型として渡され、テクスチャ型を返すことが求められる。

例としてソースコード 3.4 に、生成時にビットマップ画像を読み込み、評価時にその画像を出力する BitmapNode のソースコードを示す。

ソースコード 3.4:

```
using ImproV;
using SlimDX.Direct3D10;

namespace ImproVCoreTest {

  public class BitmapNode: Node {

    Texture2D BitmapTexture;

    public BitmapNode(string filename) : base()
    {
```

```
12        BitmapTexture = GlobalDevice.CreateTexture(filename);
13      }
14
15      protected override Texture2D evaluationProcess(int currentFrame)
16      {
17        return BitmapTexture;
18      }
19
20    }
21  }
```

10〜13 行目はコンストラクタである。GlobalDevice.CreateTexture は、ImproV の映像処理エンジンにて提供されるヘルパメソッドであり、ファイルパスを引数にテクスチャを生成する。15〜17 行目にて evaluationProcess メソッドがオーバライドされている。この例では、コンストラクタにおいて生成したテクスチャを返しているだけである。

この他に入力が必要なノードには、入力ポートの追加が必要である。入力ポートを追加するには、例えばコンストラクタ等において、入力ポートを表す Input オブジェクトを生成し、AddInput メソッドに引数として渡す。このメソッドは、Input クラスのコレクションである Inputs というプロパティに渡された変数を登録する。Inputs に登録された Input オブジェクトは、その入力ポートにエッジが結線されていてもそうでなくとも、evaluationProcess メソッドが呼び出される直前に評価を終えており、それぞれの Input オブジェクトの TextureValue というプロパティを参照することにより、入力されたテクスチャを参照できる。

またノードには、そのノード特有の GUI を持たせることが出来る。例えば “Slider” は、ユーザが値を指定するためのスライダを持っている。このような GUI を持たせるためには、CustomGUI クラスを継承したクラスを実装し、そのインスタンスを Node クラスの CustomGUI プロパティに設定する。CustomGUI クラスは、画面面積である Size プロパティや、OnMouseDown、OnDrawGUI といったイベントハンドラを提供する。

ImproV の画像処理ライブラリを使って実装されたアプリケーションは、起動時にアプリケーションと同じパスに置いてあるノードのアセンブリをすべて読み込みリンクする。リンクされたノードのリストが提供され、アプリケーションから利用することが可能になっている。

3.5 映像処理エンジンの性能評価

ImproV の映像処理エンジンがどれくらいの性能をもつかを確かめるための実験を行った。ImproV の映像処理エンジンは、映像ミキサや映像エフェクトの複雑な組み合わせを動的に構築することが目的である。映像ミキサや映像エフェクトはピクセルシェーダを使って実装されている。これらピクセルシェーダを使ったノードの処理性能を調べることにより、どれくらい複雑な組み合わせを処理できるかの目安になる。また、動的にデータフローを変更したときのオーバヘッドを計測し、データフローの動的変更が実用的かどうかを確かめた。

3.5.1 ピクセルシェーダを使ったノードの処理性能

ピクセルシェーダを使ったノードの処理は、DirectX SDK に付属の HLSL コンパイラを使うことにより、いくつのインストラクションスロットを消費するかおおよその数が計測できる。たとえば、二つの映像の合成及び切替の例では、Transparency と Mixer がピクセルシェーダを使ったノードである。Transparency は、7 スロット、Mixer は 23 スロットそれぞれ使用し、合計では 30 スロット使用する。また、エフェクト適用の例では、Transparency が同じく 7 スロット、Translate が 8 スロット、Addition が 17 スロットそれぞれ使用し、合計では 32 スロット使用する。

実際のライブ映像パフォーマンスを想定すると、エフェクト適用の例で見たようなデータフローの結果映像を流している間に、別の同様のデータフローを構築し、それらを切り替えるという作業が多くなる。ここから、$32 \times 2 + 30 = 94$ 程度のスロットを使用するといえる。

この実験では 3.4.1 節にて示した Sample を追加しながら評価し、評価にかかった時間を計測する。Sample は 6 スロットを使用する映像エフェクタである。

3.4.2 節にて例に示した BitmapNode を 2 個、3.4.1 節にて示した Sample を 130 個生成した。始めの Sample の二つの入力に、2 個の BitmapNode をそれぞれ繋ぐ。それ以降の Sample には、前の Sample の出力と、二つの BitmapNode のうち一つを繋ぎ、数珠繋ぎにした。

これを、それぞれの Sample につき評価を 1000 回ずつ行い、かかった時間の平均を計測した。n 番目の Sample を評価すると、n 個の Sample と 2 個の BitmapNode が評価されることになる。また、このとき使用されるインストラクションスロットは $6 \times n$ 個となる。

2 個の BitmapNode で読み込んだ 2 つの画像、及び出力画面サイズは 640×480 ピクセルであった。使用した計算機は、CPU は Intel Core2 Duo 2.67GHz、グラフィックカードは NVIDIA 社の GeForce 8800 GTX（グラフィックメモリ 768MB）を搭載していた。

結果のグラフを図 3.10 に示す。横軸は評価されたノード数、縦軸が評価に要した平均時間である。おおよそ線形になっていることが確認できる。最も時間を要した 130 ノード評価、780 スロット使用時でも、平均 22.8 ミリ秒しかかかっていない。これは FPS に換算すると 43FPS 出ており、十分実用的といえる。上で想定した 94 スロット使用するライブ映像パフォーマンスと比べても、より多くのノードを使用することや、より複雑なノードを使用することが可能と言える。

比較のため、同じデータフローを CPU にて 200 回づつ評価し、評価にかかった時間の平均を計測した。この CPU を使った結果と上述の GPU を使った結果のグラフを図 3.11 に示す。CPU では 130 ノード（780 スロット）使用時では 2207.1 ミリ秒かかった。1 ノード（6 スロット）評価時には 28.5 ミリ秒（35FPS）であり 30FPS を超えているが、2 ノード（12 スロット）評価時には 52.1 ミリ秒（19FPS）と 30FPS を下回る結果となる。したがって、CPU による実装は現実的ではないと言え、GPU を使った実装が必要であることがわかる。

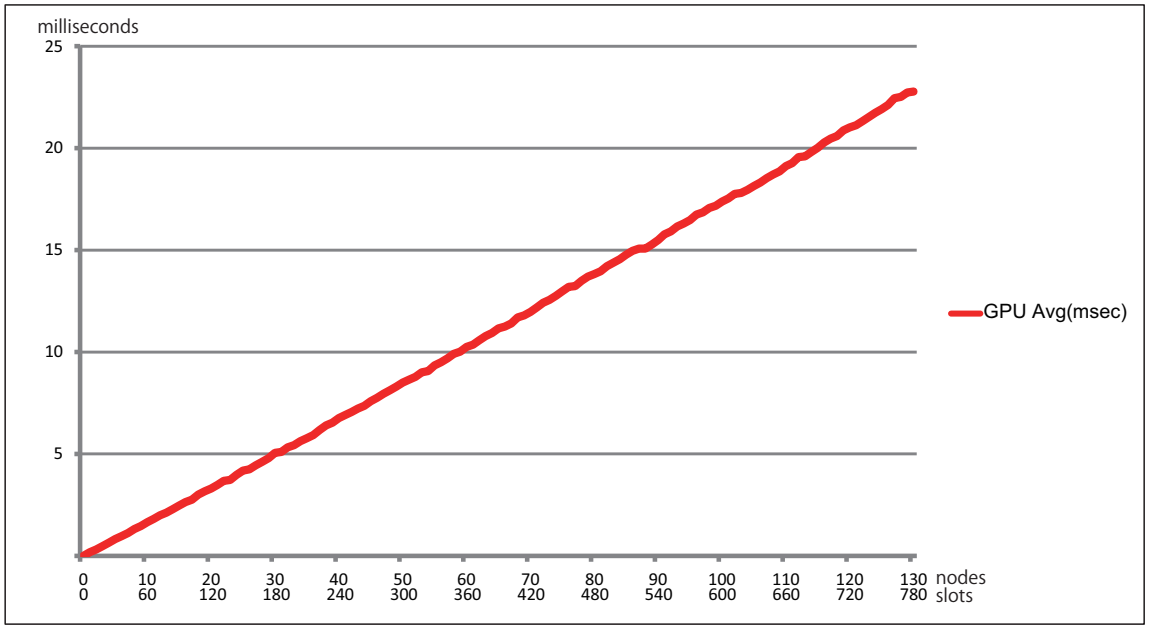


図 3.10: 性能評価実験の結果

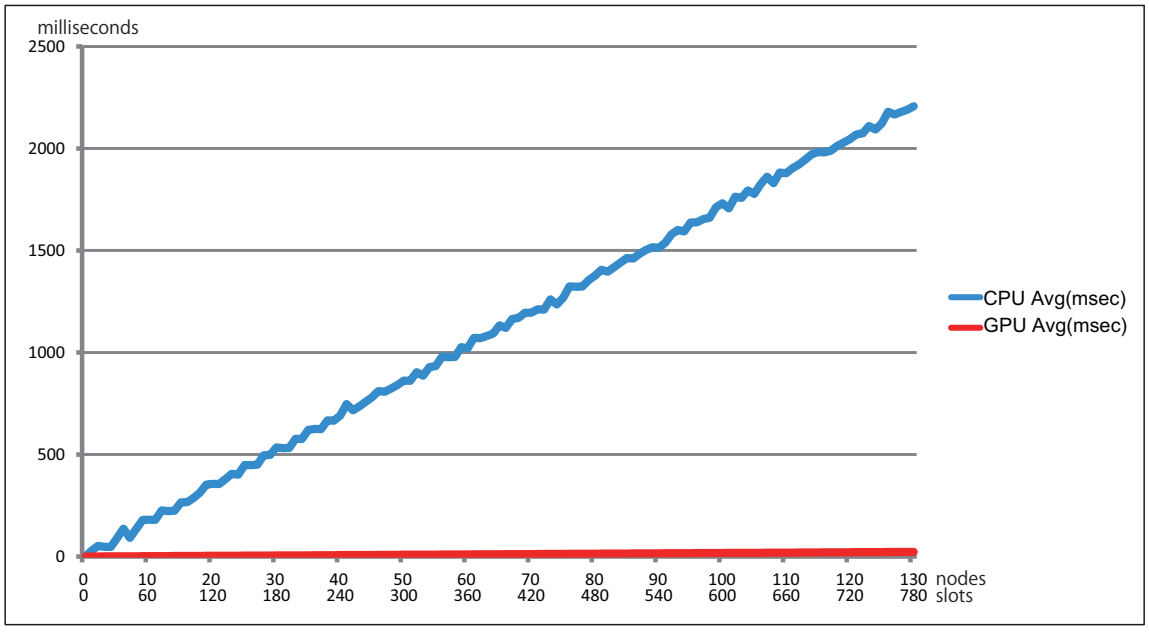


図 3.11: 性能評価実験を CPU と GPU それぞれによって実行した結果

3.5.2 データフロー変更時のオーバヘッド

データフロー変更時のオーバヘッドをプロファイラを使用し計測した。具体的には、ImproV のデータフローエディタを手動にて操作し、100 回の結線操作を行った。この時の結線操作完了時から、内部のデータフローを変更するまでの時間を計測した。結果、1 回あたり平均 0.26 ミリ秒であった。これは、30FPS における 1 フレーム（33.33 ミリ秒）において、ごく僅かを占める。

一方、ノードの生成に関しては、生成に時間を要するノードが存在する。例えば、ビデオファイルを再生する VideoFileNode は、生成時にビデオデータをバッファするのに数秒かかる。このようなノードに対しては、生成時に別スレッドを立ち上げ、そこで時間のかかる処理を行うという工夫をしている。（なお、このスレッドが終了するまでは、このノードの evaluationProcess メソッドは透明のテクスチャを返す。同時に、このノードは、データフローエディタ上でのノードをタイトルを “Loading” と変え、ユーザに読み込み中である事を知らせる。）

3.6 議論

データ型が映像型に統一されていれば、ユーザはデータフローエディタ上において、どの出力ポートからどの入力ポートへでも結線することが出来る。ただし、これはさらなる被験者実験をとおして検証すべき今後の課題である。二つ目は、エフェクトのパラメータを映像型によって指定することによって、画面上の位置や時間に応じてエフェクトのパラメータを変えることが出来る点である。これにより、ユーザはより複雑なエフェクトをより単純なデータフローによって構築できるようになった。

一方で、プログラミングの知識を持っていれば Node クラスを継承し、任意のノードを追加できるという柔軟な拡張性を備える。さらに、EffectNode クラスにより、簡単にノードの実装が行える。3.4.1 節にて述べた Sample エフェクトの例のように、ピクセルの色は RGBA の 4 つの数値から成ることや、ピクセルが 2 次元の平面に並んでいること、そして簡単な数学など、基本的な画像処理の知識のみで実装が可能である。そして、VJ を含む多くの映像制作者が、デジタル化された映像制作過程に慣れているため、このような画像処理の知識は持っていると考えられる。EffectNode クラスは映像制作者にとって、十分利用可能なツールとして、またはプログラミング学習の初めの題材として有用なものであると考えられる。

第4章　データフロービジュアル言語のシンタクティックシュガーであるデータ調整インタフェース

本章では、データフロービジュアル言語のシンタクティックシュガーであるデータ調整インタフェースについて述べる。まず、データフロービジュアル言語におけるパラメータの指定について説明し、データ調整インタフェースについて説明する。その後、ImproV に導入したデータ調整インタフェースの一種であるデータ調整ノブについて説明する。また評価実験を行ったので、それについても述べる。

4.1　データフロービジュアル言語におけるパラメータの指定と調整

広く使われているデータフロービジュアル言語は、特定のドメインに特化したものが多い。ドメイン特化型データフロービジュアル言語として、画像処理用の Koelma らによるシステム [KS94] や VIVA[Tan90]、信号解析のための LabVIEW[JK91]、音楽情報処理のための Max、音響処理のための MSP などが挙げられる。これらのドメイン特化型のデータフローで扱われるデータは 2 種類に分けることができる。すなわち、その言語の処理対象である主データと、ノードのパラメータ設定に使われる副データである。

例として MSP にて、ノコギリ波を生成し、その信号にローパスフィルタを適用する場合を考える。ノコギリ波を生成するノードは、周波数として実数値の入力を持ち、音響信号を出力する。また、ローパスフィルタのノードは、音響信号の入力と、カットオフ周波数として実数値の入力を持ち、ローパスフィルタの適用結果を音響信号として出力する。この例で主データとは音響信号であり、副データは実数値である。

MSP を含む、多くのデータフロービジュアル言語では、副データも、主データと同じデータフロー上で扱われる。このことは、データフローの見た目を複雑にする上に、操作における手間を増やす。上記の例では、実数値の定数ノードを 2 つ作成し、ノコギリ波生成ノードの周波数とローパスフィルタのカットオフ周波数にそれぞれ入力する必要がある。さらに、ノコギリ波生成ノードから出力される音響信号のボリュームを調整するとき、ユーザは乗算ノードと新しい定数ノードを作成し、それらを結線し、ノコギリ波生成ノードとローパスフィルタノードを結ぶエッジ上に挿入する。この音響信号のボリューム調整は頻繁に行う必要があるが、そのたびにユーザはこれらの操作を行うことになる。

視覚的な面においても、これらの副データのデータフローは MSP の主データである音響信

号のデータフローを把握しにくくする。このように、主データのデータフロー上に副データのデータフローが混在することは、主データのデータフローの把握を阻害する。

4.2 データ調整インタフェース

データフロービジュアル言語におけるパラメータの指定と調整を行うためのシンタクティックシュガーとして、データ調整インタフェースを定義する。

4.2.1 データ調整インタフェースの視覚表現

まず、データ調整インタフェースはその名の通り、データを調整するユーザインタフェースである。ここから、ユーザがデータを入力する何らかのユーザインタフェースを持つこととなる。このユーザインタフェースには様々なものが考えられる。数値であれば、スライダ、ノブ、数値入力テキストボックスなどが考えられる。

次にデータ調整インタフェースの配置について考える。データ調整インタフェースはあるノードのあるパラメータを指定、若しくは調整する。このため、データ調整インタフェースはどのパラメータを調整するのかが明らかになる視覚表現である必要がある。

更に、データフロービジュアル言語において、パラメータの指定と調整は、別のノード、別のノードからのエッジ、ノードの入力ポートにて行われる。このため、これらの視覚要素、若しくはそれら近い位置に配置されるのが適切である。

パラメータの指定については、パラメータが指定されるノードそのものにデータ調整インタフェースが組み込まれていることが望ましい。ノードそのものにデータ調整インタフェースが組み込まれていれば、そのノードのみでパラメータが指定が完結する。

これらの理由から、データ調整インタフェースはノード上の調整するパラメータの入力ポート、若しくはその近辺に配置され、その入力ポートに結び付けられた視覚表現を持つ。

4.2.2 データ調整インタフェースの機能

データ調整インタフェースはパラメータの指定と調整の二つの意味を持つ。パラメータの指定とは、新しいデータを生成し、そのデータをパラメータに設定することである。パラメータの調整とは、すでに存在するデータに対して変化を加え、その結果をパラメータに設定することである。

データ調整インタフェースは、結び付けられた入力ポートに入力が存在するかどうかに応じて、これら二つの意味を切り替える。データ調整インタフェースに関しては、パラメータに設定するとは、そのデータ調整インタフェースに結び付けられた入力ポートにデータを入力するということである。また、すでに存在するデータとは、そのデータ調整インタフェースに結び付けられた入力ポートに入力されたデータである。

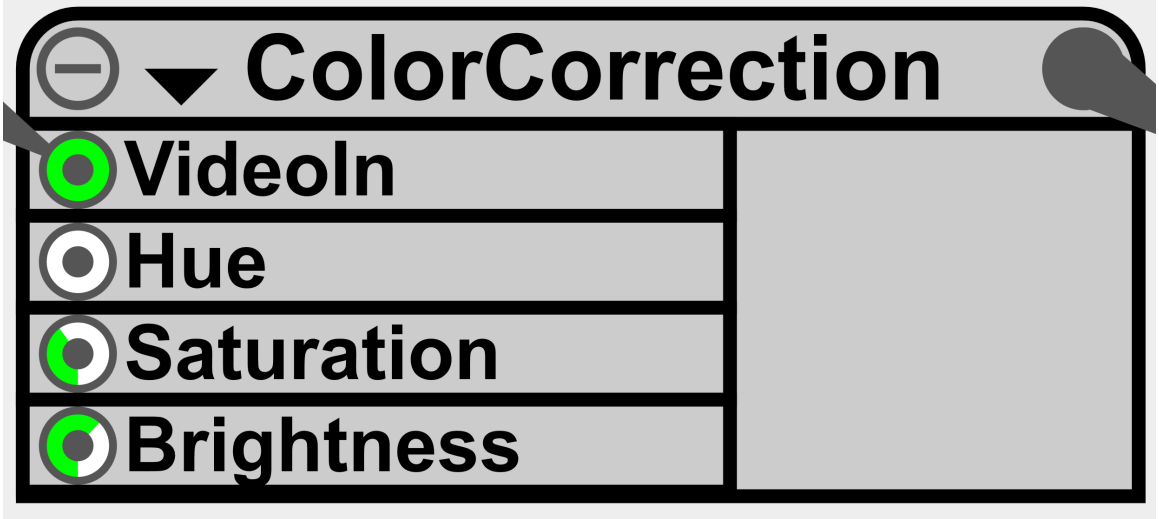


図 4.1: データ調整ノブ導入後のノード

データ調整インタフェースは、自身に結び付けられた入力ポートに別のノードからのエッジが結線されていない場合は、新しいデータを生成し、そのデータ入力ポートへ入力する。反対に、データ調整インタフェースに結び付けられた入力ポートに別のノードからのエッジが結線されてる場合は、そのエッジに流れるデータに対して、データ調整インタフェースによってユーザが指定したデータに応じた変化を加え、その変化を加えた結果を入力ポートへ入力する。

4.3 データ調整ノブ

ImproV にデータ調整インタフェースを導入するにあたって、インタフェースを ImproV の入力ポートに合わせてノブ型の視覚要素にした。これをデータ調整ノブ呼ぶ。

データ調整ノブは、入力ポートに配置されており、その入力ポートに何も結線されていない場合は定数ノードが結線されているのと同じ役割を果たし、その入力ポートにエッジが結線されている場合は入力される映像の透明度を変更する。データ調整ノブを導入することにより、定数ノードや透明度変更ノード、さらに、それらのノードを結線するためのエッジが必要なくなる。ユーザはノードのパラメータ調整や、透明度の変更を簡潔に記述できるようになり、主なデータ型のデータフローを把握し易くなる。

図 4.1 はデータ調整ノブ導入後のノードである。入力ポートの円が二重になり、二つの円の間に緑の扇型が描かれる。

データ調整ノブは実数値を持っており、その値はユーザが入力ポートを上下にドラッグすることにより 0.0 から 1.0 の範囲で変化する。緑の扇型の角度はこの値を示し、0°から 360°まで変化する。図 4.1 では、VideoIn、Hue、Saturation、Brightness のデータ調整ノブそれぞれが

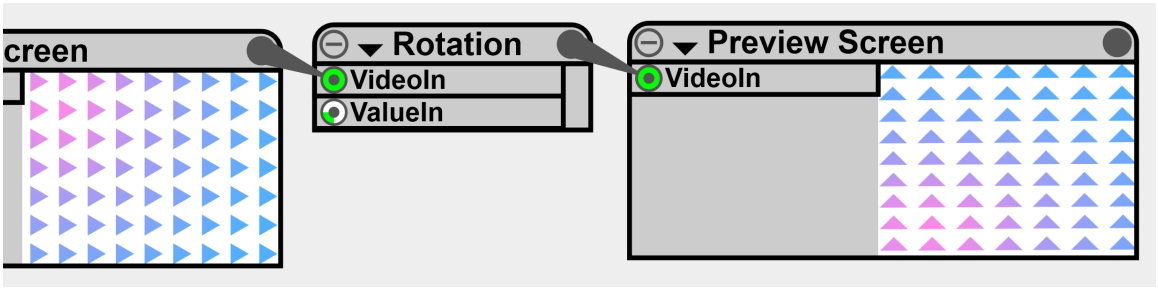


図 4.2: 入力ポートに何も入力されていないデータ調整ノブの例

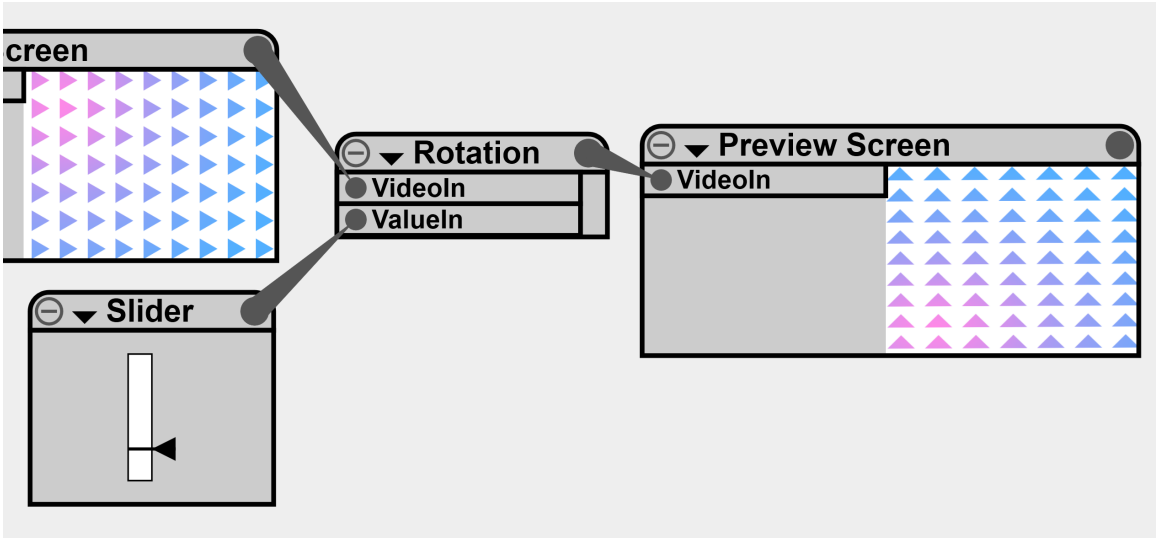


図 4.3: データ調整ノブ導入前の ImproV における図 4.2 と等価なデータフロー

1.0、0.0、約 0.33、約 0.66 を示す。

ImproV におけるデータ調整ノブは、映像エフェクタのパラメータ調整と、映像の透明度の調整の 2 つの役割を果たす。入力ポートに何も結線されていない場合は、データ調整ノブは全ピクセルが白（RGB=1.0,1.0,1.0) でありアルファ値がノブの値である画像をその入力ポートに入力する。図 4.2 に例を示す。Rotation は、VideoIn に入力された映像を ValueIn の値に応じて 0°から 360°まで回転し出力する。図 4.2 では、ValueIn が約 0.25 に合わせられているため、出力は約 90°回転している。

これはデータ調整ノブ導入前の ImproV において、約 0.25 の値に合わせた Slider を入力ポートに入力したことと同じ意味である。図 4.2 のデータフローは、データ調整ノブ導入前の ImproV における図 4.3 のデータフローと等価である。

データ調整ノブを導入することによって、Slider の作成と Slider から Rotaion の ValueIn への結線、すなわち 1 回のノード作成と 1 回の結線が削減された。この削減はパラメータ数の多い映像エフェクタに対して特に有効となる。図 4.1 をデータ調整ノブ導入前の ImproV で表

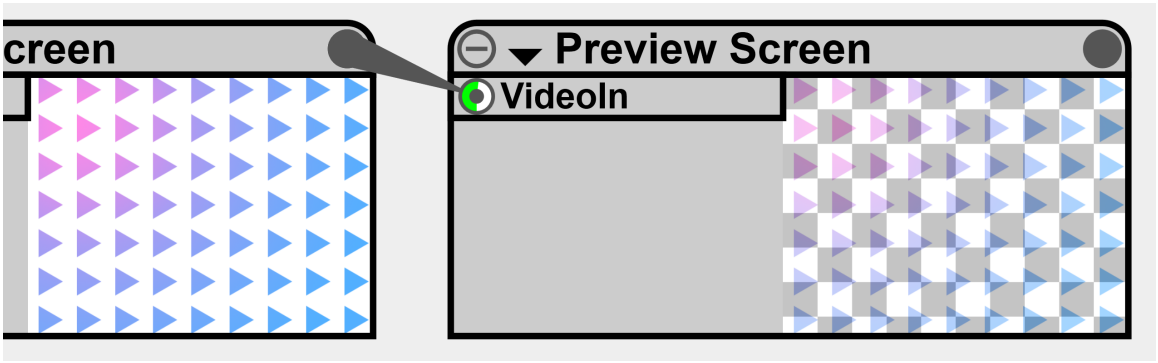


図 4.4: 入力ポートにエッジが結線されているデータ調整ノブの例

したものが図 4.5 である。

入力ポートにエッジが結線されている場合、データ調整ノブは入力された映像型のアルファ値とデータ調整ノブが持つ実数値を掛け合わせ、もとの映像型のアルファ値を更新し、入力ポートに入力する。図 4.4 では、右側の Previw Screen の VideoIn が約 0.5 に合わせられている。この結果、入力された映像が半透明になって表示されている。

この場合データ調整ノブは、データ調整ノブ導入前の ImproV において、結線されているエッジに透明度を変更する Opacity を挟み、Opacity の ValueIn に Slider を結線した事と同じ役割を果たす。図 4.6 はデータ調整ノブ導入前の ImproV における図 4.4 と等価なデータフローである。この場合、データ調整ノブを導入することによって、Slider と Opacity の作成、Slider から Opacity の ValueIn への結線、Opacity から元の入力ポートへの結線、すなわち 2 回のノード作成と 2 回の結線が削減された。

4.4 被験者実験 2

データ調整ノブの評価実験として被験者実験 2 を行った。この実験の目的は、データ調整ノブを導入することにより、データフロー記述の所要時間がどのように変わるかを確かめることである。

そのために、被験者にデータ調整ノブ有りと無しの 2 つの ImproV を使ってタスクを行ってもらい、それぞれのタスク所要時間を計測した。タスクでは、目標とするデータフローを被験者に提示し、同じデータフローを被験者に記述してもらった。図 4.7 はタスクに用いたソフトウェアのスクリーンショットである。計算機の画面が上下に赤い線によって分割され、下画面に目標とするデータフローが表示される。被験者はそれを見ながら上画面でデータフローを完成させる。

データ調整ノブはノード作成と結線の回数を削減するが、ノード作成の時間はシステムによって大きく異なる。例えば ImproV では、ノード作成をプルダウンメニューから選ぶことで作成するが、このときメニューに表示される項目が多いほどノード作成に時間がかかるだ

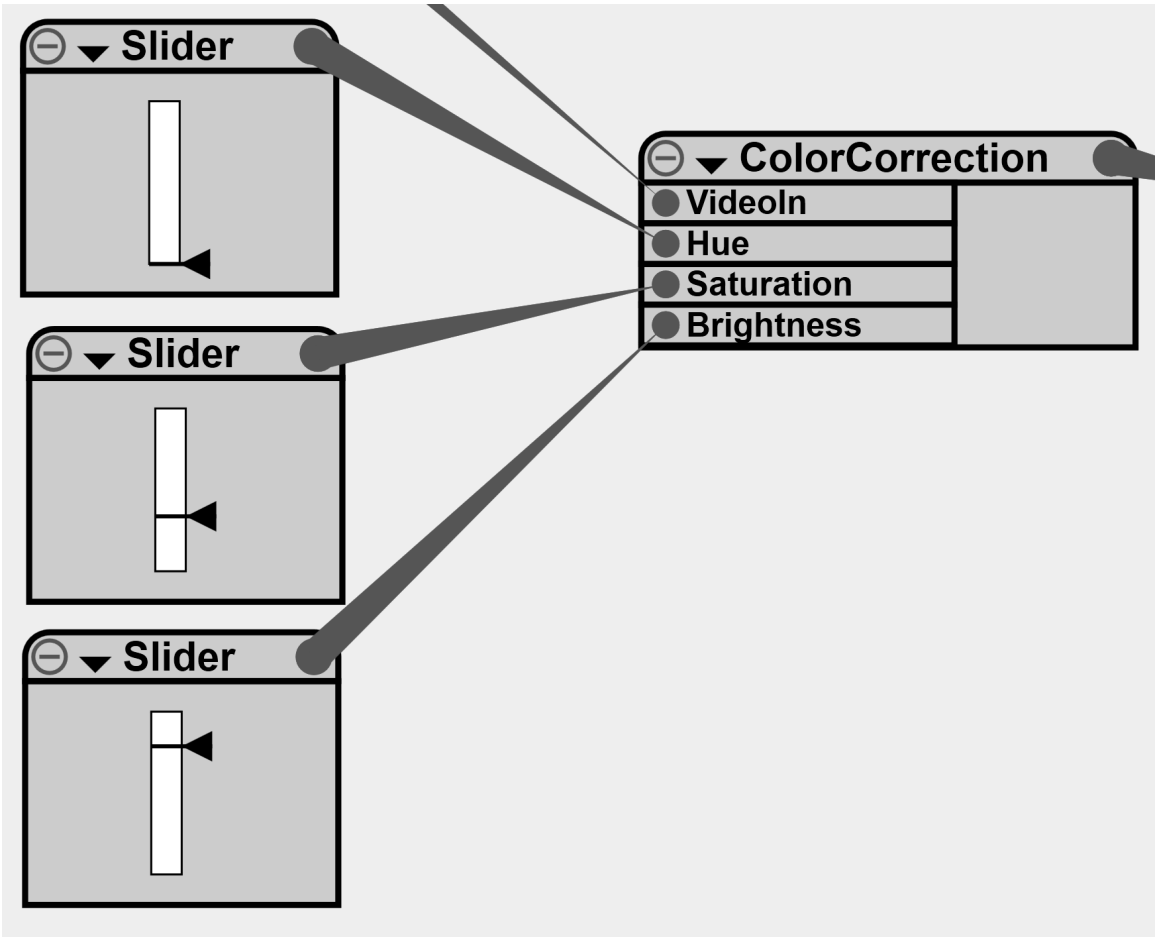


図 4.5: データ調整ノブ導入前の ImproV における図 4.1 と等価なデータフロー

ろう。このためタスク開始時に必要なノードは配置されていることとし、被験者には結線とデータ調整ノブ、若しくは Slider の値の調整を行ってもらった。

被験者は 21〜24 歳のコンピュータサイエンスを専攻する学生 8 人であった。被験者にはデータ調整ノブ有りの ImproV とデータ調整ノブ無しの ImproV の両方を説明し、十分に練習してもらった。その後被験者は、データ調整ノブ有り／無しの ImproV を使って 10 種類のタスクを行ってもらい、その後さらに、データ調整ノブ無し／有りの ImproV を使って 10 種類のタスクを行ってもらう。それぞれのタスク間では自由に休憩を取ってもらった。データ調整ノブ有りと無しの順番は被験者間でバランスをとり、タスクの順番はランダマイズした。データ調整ノブ有りと無しのそれぞれ 10 種類のタスクのうち 5 種類づつは等価である物を用意した。

この実験では、以下の 3 つの作業の所要時間をデータ調整ノブ有りと無しの両条件下で測定し比較することとした。

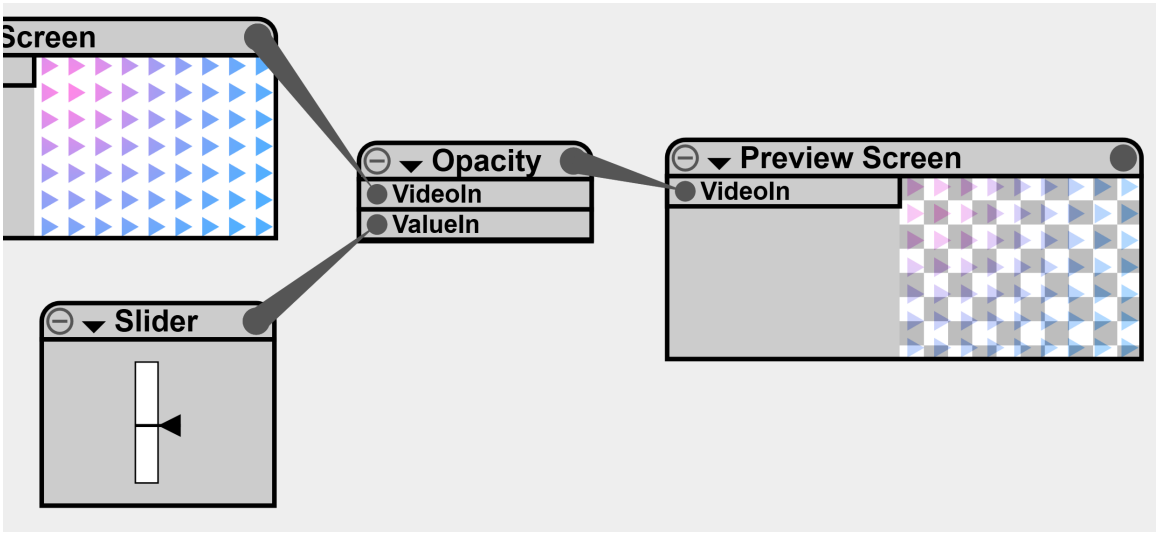


図 4.6: データ調整ノブ導入前の ImproV における図 4.4 と等価なデータフロー

作業 1 単純な結線

作業 2 入力ポートにエッジが結線されていない場合の値の編集

作業 3 入力ポートにエッジが結線されている場合の値の編集

このため、10 種類のタスクのうち 5 種類のタスク（タスク 1〜タスク 5）にこれらの作業を含めるようにした。それぞれのタスクに含まれる作業の回数を表 4.1 に示す。なお、他のタスク

表 4.1: それぞれのタスクに含まれる作業の回数

	作業 1	作業 2	作業 3
タスク 1	1 回	無し	無し
タスク 2	2 回	1 回	無し
タスク 3	1 回	無し	2 回
タスク 4	3 回	2 回	無し
タスク 5	4 回	2 回	無し

ク（タスク 6〜タスク 10）は、実験の意図を被験者から隠蔽するためのディストラクタであり、後の解析にも用いられていない。

作業 1 の所要時間を計測する目的は、単純な結線に要する操作が、データ調整ノブ無しの ImproV においては結線のみであるのに対して、データ調整ノブ有りの ImproV においては結線に加え、ノブを 0.0 から 1.0 に調整する必要があることから、データ調整ノブ有りの ImproV では単純な結線の所要時間が多くなると考えられ、その増分を調べるためである。図 4.8、図 11 それぞれにデータ調整ノブ無し、有りのタスク 1 の目標データフローを示す。図 4.8 に示

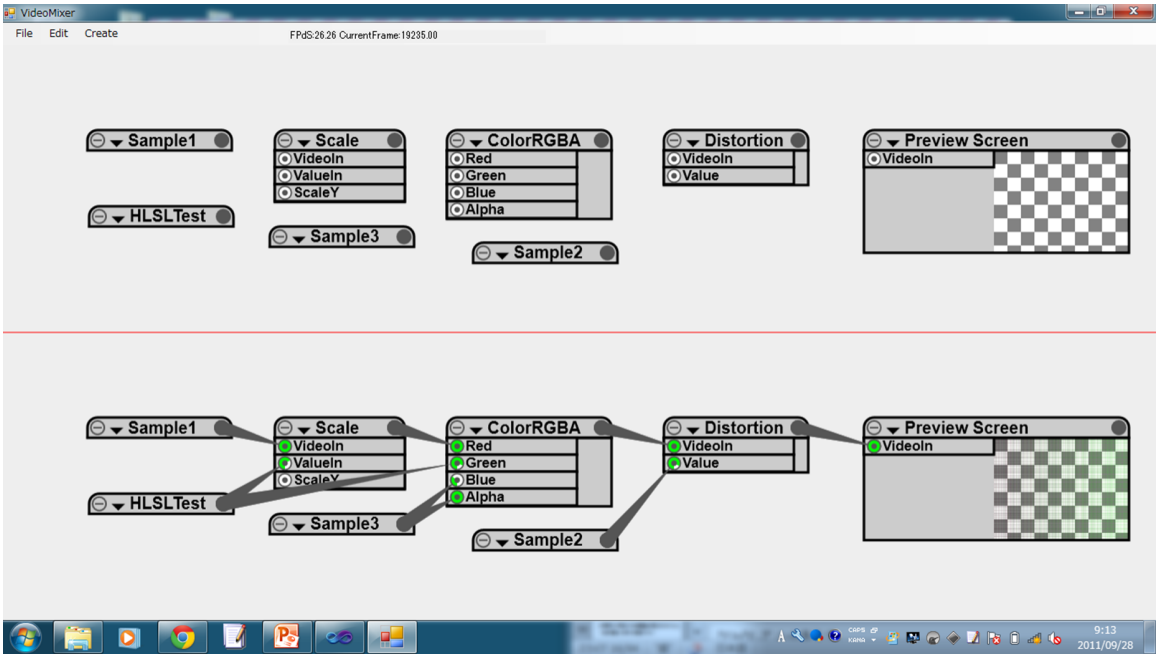


図 4.7: タスクに用いたソフトウェアのスクリーンショット

すデータ調整ノブ無しのタスク 1 を完遂するには結線が一回だけなのに対し、図 4.9 については結線とノブの調整が一回づつ必要である。

作業 2 と作業 3 の時間計測の目的は、データ調整ノブの有効性を検証するためである。作業 2 はデータ調整ノブ有りの場合はノブの調整が 1 回、データ調整ノブ無しの場合は結線とスライダーの調整が 1 回づつからなり、作業 3 はデータ調整ノブ有りの場合は結線とノブの調整が 1 回づつ、データ調整ノブ無しの場合は結線が 3 回とスライダーの調整が 1 回からなる。

実験結果を図 4.10 に示す。図 4.10 では、緑はデータ調整ノブ有り、橙はデータ調整ノブ無しのそれぞれのタスクにかかった平均時間をミリ秒で示している。エラーバーは標準偏差である。分散分析の結果、タスク 3 にのみ有意な差が見られた（$F(7,1) = 77.449, p < 0.01$）。

当初、我々はデータ調整ノブ導入によって作業 1 の所要時間が伸びると考えていた。しかし、作業 1 が 1 回のみからなるタスク 1 には有意な差は見られなかった（$F(7,1) = 0.637, p > 0.1$）。このことから、結線のみの操作と、結線に加えて 0.0 から 1.0 へのノブの調整を行う操作の間に有意差はなく、ノブの調整が加わる事によるオーバーヘッドは問題では無いと考えられる。

有意差が見られたタスク 3 は作業 3 を 2 回含んでおり、作業 2 は含んでいないことから、データ調整ノブは作業 3 において有効であるといえる。また平均値も、データ調整ノブ有りの場合 22.1 秒、データ調整ノブ無しの場合 13.7 秒と 10 秒近く差があり、データ調整ノブの有用性を示している。

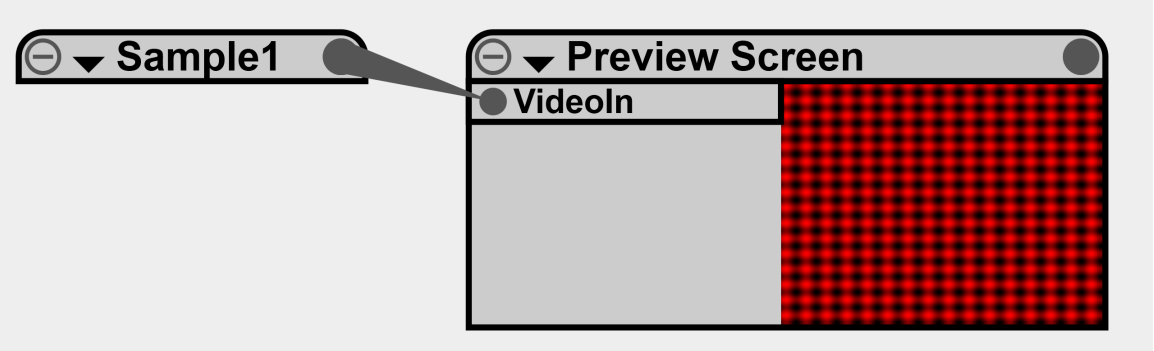


図 4.8: データ調整ノブ無しのタスク 1

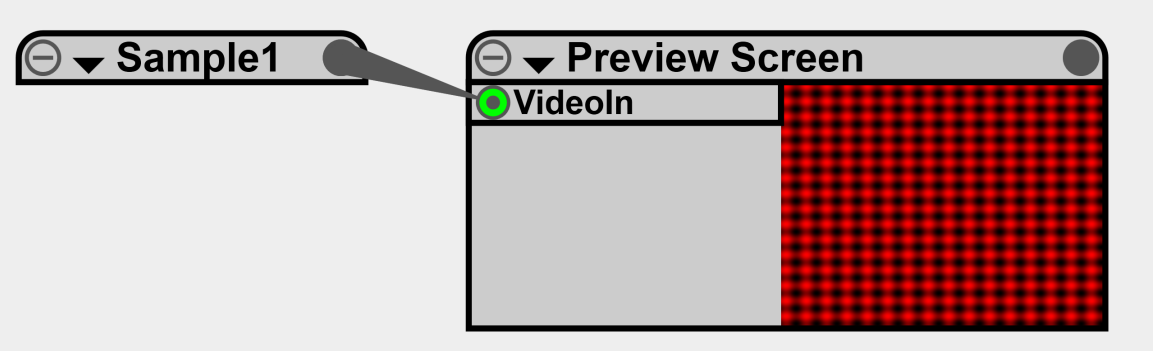


図 4.9: データ調整ノブ有りのタスク 1

4.5 議論

データ調整ノブの ImproV への導入は、データ調整インタフェースを実数値によって生成することが可能である映像型に適用した例と言える。データ調整ノブは、実数値を指定するノブを持つデータ調整インタフェースであり、実数値、または、実数値によって生成することが可能であるデータ型の指定や調整に有効である。

ドメイン特化型のデータフロービジュアル言語においては、エフェクタのパラメータ調整とは、定数ノードをエフェクタのパラメータに結線し、定数ノードの値を調整することである。そして、映像の透明度の調整とは、乗算ノードと定数ノードの組み合わせにおいてその定数ノードの値を調整することである。調整したいデータを乗算ノードの一項に結線し、定数ノードの出力をもう一方の乗算ノードの入力に結線する。この状態で定数ノードの値を変化させると、もとのデータと定数ノードの値が掛け合わされたデータが乗算ノードから出力される。例えば音響信号を扱うデータフロービジュアル言語では、乗算は音量の調整となる。この様にデータ調整ノブは、他分野のデータフロービジュアル言語においても有効であると考えられる。

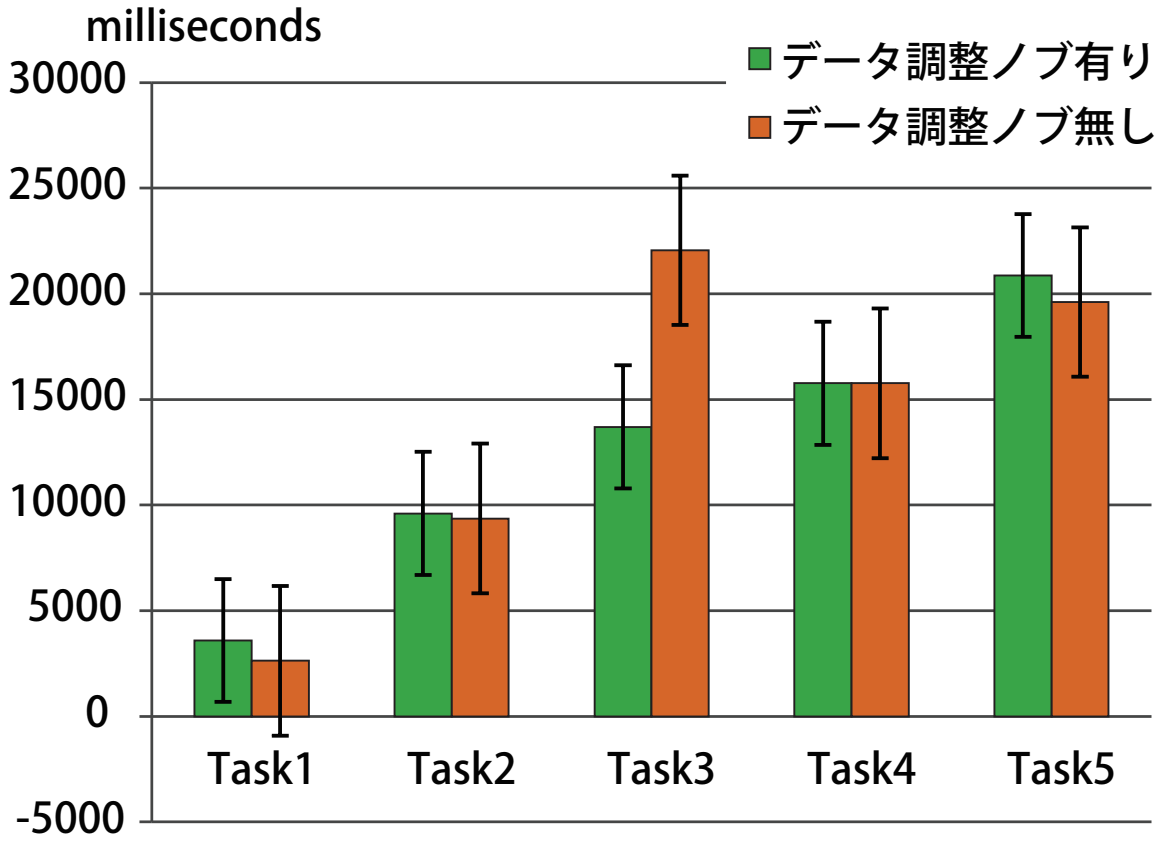


図 4.10: 実験結果：データ調整ノブ有りと無しのそれぞれのタスクにかかった平均時間

第5章　関連研究

5.1　データフロービジュアル言語

データフロービジュアル言語は、特定の分野に特化した場合において特に有効であり [JHM04]、そのようなデータフロービジュアル言語が多く研究されている。例えば、可視化の分野に特化したものとして、AVS/Express[Adv] や Hils による DataVis[Hil91]、信号処理に特化したものとして LabVIEW[JK91] などが挙げられる。

Hansaki らによる Find Flow[HSMT06] はデータベースの検索クエリをプログラミングするためのデータフロービジュアル言語である。Find Flow ではエッジが検索クエリを表し、ノードはデータソースやクエリ結果のデータ集合を表す。各ノード上にはデータ集合の内容が表として表示されており、データフローの変更は即時に反映される。ユーザはノードの表示を確認しながら、試行錯誤によってデータを絞り込んでゆく。

Puckette による、Patcher[Puc88] は楽譜情報を扱うデータフロービジュアル言語である。また、同じく Puckette による Pure Data[Puc96] は Patcher と同様の文法を持つ音声信号処理用のデータフロービジュアル言語である。その後、Patcher は Max として製品化され、Pure Data は MSP という名前で Max にプラグインとして組み込まれた [Cycb]。これらは、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。また、REAKTOR[Nata] も音声処理のデータフロービジュアル言語であり、Max/MSP と同様の使われ方をする。しかし、REAKTOR ではデータフロー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[Natb] 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、VJ にとって、理解しやすい事を目指している。

映像を扱うデータフロービジュアル言語としては Jitter[Cyca] が挙げられる。Max/MSP[Cycb] のプラグインであり、映像を扱うライブラリである。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）のデータフロー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。

映像の再生中にデータフローの変更を行うことが出来るデータフロービジュアル言語として、vvvv[vvv] や Quartz Composer[App] 等が挙げられる。しかし、これらのデータフロービジュアル言語は、プログラミング言語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。

汎用データフロービジュアル言語に関する研究も多く成されてきた。初期の例として Ichikawa らの HI-VISUAL[IH87] が挙げられる。HI-VISUAL は、処理とデータをアイコンによって表現

し、それらの関係を結線によって表現する。アイコン表現を使っている点や、データ型によって適用可能な処理の候補をユーザに提示する点など、ユーザに親しみやすいプログラミング環境を提供するための工夫が試みられている。また、プログラマ向けの汎用データフロービジュアル言語として Bernini らによる VIPERS[BM94] が挙げられる。VIPERS はプログラマ向けのデータフロービジュアル言語である。Tcl によって作られており、他の Tcl アプリケーションと通信することが可能である。ユーザは必要に応じてテキストのプログラミングと VIPERS によるプログラミングを使い分け、それらの成果を組み合わせる事が可能である。VIPERS は Tcl の知識を持つプログラマを対象とし、プログラミングを簡略化することを目標としている。これらに対して本研究は、ライブ映像パフォーマンスに特化しており、対象ユーザが普段使っている機材と同じ水準の語彙をノードとすることにより、対象ユーザに親しみやすさを提供する。

5.2 ライブパフォーマンスに関する研究

本研究はライブ映像パフォーマンスを対象としたユーザインタフェースの研究とも言える。そこでライブパフォーマンスに関する研究について紹介し、本研究と比較する。

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[Dig] 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[Der] 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ビジュアルプログラミング言語を備えるが、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[Mav] は ImproV 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データフロービジュアル言語である。しかし、データフローの編集に関しては、準備作業の段階で行うことを想定されており、リアルタイムにデータフロー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。

Müeller 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、上記の映像を扱うデータフロービジュアル言語同様、VJ 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。

5.2.1 ライブ操作に関する研究

DJ の作業手順における「手順 3：ライブ操作」に関しては、音楽を対象とした多くの研究が存在する。センサを取り付けたデバイスを使うことで、人間の体の動きを楽器の制御に使うという物が多い。例えば、Villar らによる ColorDex[VGJL07] は、加速度センサを取り付けたキューブ型デバイスである。ColorDex を演者が傾ける事により、音響ミキサのフェーダを操作する。他にも、Miyama による Peacock[Miy10] は、距離センサをマトリクス状に並べたデバイスである。予め各距離センサを音響シンセサイザや音響エフェクトのパラメータに割り当てる。演者は Peacock の上で手を動かすことにより、それらのパラメータを操作する。

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[TII07]、Nervixxx[Tok09] 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、VJ はそのマラカスを振ることによって映像を操作する。Nervixxx は VJ の筋電位を計測し、その筋電位によって映像の操作を行う研究である。

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」を対象とした本研究とは異なる。しかし、ImproV を使って「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を使って「手順 3：ライブ操作」を行うといった共存も可能である。

5.2.2 live coding に関する研究

live coding と呼ばれるライブパフォーマンスが存在する。live coding とは、（主に音楽や視覚の）デジタルコンテンツを生成するプログラムを、演者がその場でプログラミングする事により、デジタルコンテンツを生成するというパフォーマンス [Bro06, Nil07] である。この定義によると、ImproV によるライブ映像パフォーマンスも live coding の一種といえる。

live coding は、特に音楽の分野において盛んである。音楽 live coding のための言語として、テキスト言語によるものが多く研究されている。Mills らによる Minim[IFB10] は、音楽 live coding のための Processing のライブラリである。Sorensen による Impromptu[Sor05] は、音楽 live coding の Scheme 環境である。Impromptu には、コーディング環境、実行環境、ライブラリなどが含まれる。Wang らによる Chuck[WC04, Wan04]、および、Mccartney による SuperCollider[McC96, McC02] は、live coding のために設計されたデータフロープログラミング言語である。Minim や Impromptu が既存言語にライブラリを追加することにより live coding をサポートしているのに対して、Chuck や SuperCollider は言語自体が live coding のために新しく設計されている。ImproV はビジュアル言語である点がこれらの研究と大きく異なる。

また、live coding のためのビジュアル言語についても幾つか研究されている。McCormack らによる Nodal[MMLD07, MMLD06] は、live coding のためのビジュアル言語である。Nodal はビジュアル言語によって手続きフローを記述することにより、音楽の演奏情報を生成する。ImproV はデータフロービジュアル言語である点が異なっている。

Bencina による AudioMulch[Ben98] は live coding のための音響処理データフロービジュアル言語である。ImproV は映像を対象としている点や、プレビューをサポートしている点などが異なる。

5.3 ビジュアル言語の記述手法に関する研究

データ調整インタフェースはビジュアル言語の記述手法に関する研究と言える。ビジュアル言語の記述手法に関する研究を挙げ、データ調整インタフェースとの差異を明確にする。

Hirakawa らは、アイコンを重ねることによって二つのオブジェクトの関係を記述する手法を開発した [HTI90]。本研究ではデータフローのノード間の関連を記述するが、一つのノードに対して複数の関係を記述する必要があり、この手法を使うことは難しい。

Peacock は、画像処理用データフロービジュアル言語である [Avia]。ノードは四角形であり、その四角の各辺に異なるデータの入出力ポートが配置される。上辺には画像データの入力ポート、下辺には画像データの出力ポート、左辺にはノードのパラメータ信号の入力ポート、右辺にはパラメータ信号の出力ポートがそれぞれ配置される。記述されるデータフローでは、縦方向に画像データ、すなわち処理対象のデータ、の流れが表現され、横方向にノードのパラメータデータの流れが表現される。つまり、処理対象のデータフローとパラメータデータのデータフローを異なる視覚表現にて表現するというアプローチと言える。

平井らは、データフロービジュアル言語 Max において音声によってノードを作成する手法を開発した [平井 08, 田中 08]。データフローの編集効率の改善という点で本研究と類似しているが、本研究のデータ調整ノブはノードとエッジの作成の機会そのものを減らすというアプローチを採っている点で異なる。

5.3.1 ライブパフォーマンス向けのデータフロービジュアル言語

ライブパフォーマンス向けのデータフロービジュアル言語はいくつか研究されており、パラメータの調整やデータフローの編集において様々な工夫がなされている。

Jordà らの reacTable[JGAK07] は、ライブ音楽パフォーマンス向けのデータフロービジュアル言語である。デーブルトップ上のタンジブルオブジェクトがノードを表し、ノード同士を近づけると自動的にそれらを結線するというデータフロー編集手法を持つ。また、Taylor らの VPlay[TIK$^+$09] は、ライブ映像パフォーマンス向けデータフロービジュアル言語であり、ノードを近づけると自動で結線するというデータフロー編集手法を持つ。

これらは従来のポインタを使い入力ポートと出力ポートを指定するという結線手法より、手間が軽減されているが、一方でノードのレイアウトを制限する上、ユーザの意図しない結線

が生じてしまう。データ調整ノブはノードとエッジの作成の機会を減らすが、そのような副作用は生じない。

AudioMulch[Ben98] のデータフローが扱うデータ型は音響信号のみであり、パラメータは別のウィンドウにノブやスライダとして配置される。データ型を一種類に限定する点は本研究のアプローチと類似するが、ImproV の映像型とデータ調整ノブの組み合わせは、処理対象のデータとパラメータのデータを同じデータフロー上にて処理できる点が大きく異なる。

第6章　結論

ライブ映像パフォーマンスにおいて、即興的に映像を加工及び合成するためのデータフロービジュアル言語、ImproV について述べた。VJ は、ImproV のデータフローエディタにてプレビューが可能である。プレビューにより VJ は、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、即興的に行うことが可能となる ImproV の水準は、映像機材をノードとし、映像機材の接続関係をデータフローによって記述するものである。この言語は、被験者実験により、VJ にとって理解しやすいものであることが分かった。

ライブ映像パフォーマンスの即興性をさらに高めるために、映像エフェクタのパラメータを映像によって指定する機構を考案し、ImproV に実装した。映像によってパラメータを指定することにより、映像の位置によって異なるパラメータを指定することやパラメータをアニメーションさせるといった、複雑なパラメータの指定が可能になった。このように、処理対象のデータによってパラメータを指定することにより、ユーザは自身が専門とする処理対象と同じ語彙によってパラメータデータを加工することが可能となる。さらに、この機構の実装に GPU を利用することで、ライブ映像パフォーマンスにて必要な映像処理を実時間にて処理可能であることを示した。

データフロービジュアル言語において、ノードへの入力を調整するシンタクティックシュガー：データ調整インタフェースを考案し、データ調整ノブとして ImproV に導入した。このデータ調整インタフェースをデータフロービジュアル言語に導入することによって、視覚的複雑さ、及び、操作回数を削減することが出来る。データ調整ノブを導入した ImproV においてユーザは、エフェクトのパラメータ調整と映像の透明度の調整の 2 つの操作を、定数ノードや透明度変更ノードの組み合わせを使って調整する従来の手法よりも簡潔に行うことができる。また、我々は被験者実験を行い、その結果、映像の透明度の調整にはデータ調整ノブが特に有効であることを確かめた。このデータ調整ノブの導入は、一般のデータフロービジュアル言語においても、ノードの作成や結線の操作を行う機会を削減することができ、有効であると考えられる。

6.1　本研究の貢献

データフロービジュアル言語の研究において、ドメイン特化型データフロービジュアル言語が現在の主流になりつつある。本研究では、ノード水準を対象ユーザの興味水準に合わせ、さらに、ユーザの専門分野である映像操作と同じ方法によるパラメータデータ操作を可能と

する、映像型によるパラメータ指定機構を実現した。これらは、データフロービジュアル言語を、より特定のドメインの専門家にわかりやすくするための指針を示すものであり、本研究の貢献である。

理想的なデータフロービジュアル言語とは、ユーザが着目すべきデータの流れのみを提示し、ユーザは最低限の記述によってデータの流れを定義可能なものである。本研究では、データ調整インタフェースを導入することにより、ユーザが着目すべきであるデータの流れのみを提示することを実現した。これらにより、データフロービジュアル言語が本来持つ、見やすさ、及び、記述しやすさを強化した。

6.2 今後の展望

データフロービジュアル言語としての展望と、ライブパフォーマンスのためのユーザインタフェースとしての展望について、それぞれ以下に述べる。

6.2.1 データフロービジュアル言語としての展望

本研究の成果である ImproV は映像処理用データフロービジュアル言語であり、ライブ映像パフォーマンスという適用対象から、高いインタラクティブ性を持つ。つまり、ImproV にとって言語の記述時と実行時に差はなく、ユーザによる記述が即時に実行に反映される。このインタラクティブ性はライブパフォーマンスだけではなく、広い範囲のデータフロービジュアル言語において有効である。ユーザがデータフローの記述を行う際、ユーザが行った変更が即時にデータフローに反映され、計算結果が提示されることはユーザの試行錯誤を強力に支援する。

本研究で示した、GPU による実装はデータフロー計算モデルと相性が良い。GPU はもともと、3 次元コンピュータグラフィクスを実時間にて処理するためのハードウェアであった。このため、座標変換、ラスタライゼーション、ピクセル処理等の処理をパイプライン処理する。近年の GPU は、このパイプラインをプログラマが自由に実装できる様になっており、この性質を利用し、GPU を使ってグラフィクス以外の処理を行うという General-purpose computing on GPU（GPGPU）という技術も多く研究されている。GPU や GPGPU におけるパイプライン処理はデータフロー計算モデルによって記述可能であり、かつ、GPU によってデータフロービジュアル言語を実装可能であることは本研究にて示した。今後、GPU を使ったインタラクティブ性の高いデータフロービジュアル言語が、様々なドメイン、特に GPGPU によって解決されうる分野へ応用されることが期待できる。

今後はこれらの技術を利用し、高い即時性を持ち、データの流れを把握し記述しやすくするためのユーザインタフェースとして、データフロービジュアル言語が様々なドメインへ応用されることが期待できる。

6.2.2 ライブパフォーマンスのためのユーザインタフェースとしての展望

本研究では、映像の制作段階でしか成し得なかった映像の加工と合成を、即興的に行うことを実現した。しかし、映像の制作段階でしか成し得ない作業はまだ残されている。例えば、3D コンピュータグラフィクスに関わる作業については本研究では取り扱わなかった。これは、映像の加工と合成と、3D コンピュータグラフィクス制作ではデータの形式が大きく異なるためである。このような異なるデータを同時に扱え、かつ、ユーザが即興的に操作することが可能であるユーザインタフェースが実現可能かどうか、また、実現可能であればどのようなユーザインタフェースであるか、今後の研究が望まれる。

謝辞

本論文の執筆において、筑波大学システム情報系情報工学域の田中二郎教授、ならびに、志築文太郎先生には多くの御助言を頂きました。また、本研究を進めるにあたって博士課程 5 年間にわたり、研究の進め方、実験方法、論文執筆方法など、多大なる御指導を頂きました。

本論文の審査にあたっては、田中二郎教授、志築文太郎先生、島根大学総合理工学部数理・情報システム学科計算機科学の平川正人教授、筑波大学システム情報系情報工学域の西川博昭教授、福井幸男教授、知能機能工学域の星野准一准教授に審査委員を努めて頂きました。審査では、本論文をまとめるにあたって有用なご意見、ご指摘を頂きました。

筑波大学システム情報系情報工学域の三末和男准教授ならびに、高橋伸准教授にはゼミにてご意見、ご指摘を頂きました。

株式会社ババガンプ社には博士前期課程 2 年間にわたり経済的支援を頂きました。また、博士後期課程では文部科学省グローバルＣＯＥプログラム「サイバニクス：人・機械・情報系の融合複合」に研究補助員として雇用頂きました。これらの経済支援のおかげで研究活動を続けることが出来ました。

本研究は江藤貴康氏との VJ 活動を発端としています。江藤貴康氏には研究を進めるにあたって VJ の視点から様々なご意見を頂きました。また、被験者実験にあたっては、現場で活躍する VJ の方々に協力を頂きました。実験後も有用なご意見、ご議論を頂き、研究を進めるにあたり大変役立ちました。

筑波大学システム情報工学研究科コンピュータサイエンス専攻田中研究室 WAVE グループの学生には、本研究についての議論にお付き合いいただきました。また、日頃の研究活動の協力、支援を頂きました。

以上の方々に心より謝意を述べさせて頂きます。ありがとうございました。

参考文献

[Adv] Advanced Visual Systems Inc. AVS/Express. Retrieved 30 December 2011 from `http://www.avs.com/products/avs-express/index.html`.

[App] Apple Inc. Quartz Composer. Retrieved 30 December 2011 from `http://developer.apple.com/technologies/mac/graphics-and-animation.html`.

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

[Avia] Aviary, Inc. Effects Editor Peacock. Retrieved 30 December 2011 from `http://advanced.aviary.com/online/filter-editor`.

[Avib] Avid Technology, Inc. Torq Xponent. Retrieved 30 December 2011 from `http://www.m-audio.jp/products/jp_jp/TorqXponent.html`.

[AW77] Edward A. Ashcroft and William W. Wadge. Lucid, a nonprocedural language with iteration. *Communications of the ACM*, Vol. 20, pp. 519 –526, ACM, 1977.

[Ben98] Ross Bencina. Oasis rose the composition - real-time DSP with AudioMulch. In *Proceedings of the Australasian Computer Music Conference*, pp. 85–92. Australasian Computer Music Association, 1998.

[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. National University of Singapore, 2002.

[BM94] Massimo Bernini and Mauro Mosconi. Vipers: a data flow visual programming environment based on the tcl language. In *AVI '94: Proceedings of the workshop on Advanced visual interfaces*, pp. 243–245. ACM, 1994.

[Bro06] Andrew R. Brown. Code jamming. *Journal of Media and Culture*, Vol. 9, Retrieved 30 December 2011 from `http://journal.media-culture.org.au/0612/03-brown.php`, Media and Culture, 2006.

[Cha06] Shi-Kuo Chang. Visual languages: A tutorial and survey. *IEEE Software*, Vol. 4, pp. 29 –39, IEEE Computer Society, 2006.

[CJL+85] Shi-Kuo Chang, Erland Jungert, S. Levialdi, G. Tortora, and Tadao Ichikawa. An image processing language with icon-assisted navigation. *IEEE Transactions on Software Engineering*, Vol. 11, pp. 811–819, IEEE, 1985.

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

[Cyca] Cycling '74. Jitter. Retrieved 30 December 2011 from `http://www.cycling74.com/products/jitter`.

[Cycb] Cycling '74. Max/MSP. Retrieved 30 December 2011 from `http://www.cycling74.com/products/maxmsp`.

[Der] Derivative Inc. TouchDesigner . Retrieved 30 December 2011 from `http://www.touch077.com/Products/`.

[Dig] Digitalstage Inc. motion dive .tokyo. Retrieved 30 December 2011 from `http://www.digitalstage.net/en/`.

[FSE04] Kentaro Fukuchi, Mertens Sam, and Tannenbaum Ed. EffecTV: a real-time software video effect processor for entertainment. In *ICEC 2004: International Conference on Entertainment Computing*, pp. 602–605. Springer, 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. ACM, 1991.

[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. Australian Computer Society, Inc., 2006.

[HTI90] Masahito Hirakawa, Minoru Tanaka, and Tadao Ichikawa. An iconic programming system, HI-VISUAL. *IEEE Transactions on Software Engineering*, Vol. 16, pp. 1178–1184, IEEE Computer Society, 1990.

[IFB10] John Anderson Mills III, Damien Di Fede, and Nicolas Brix. Music programming in minim. In *NIME '10: Proceedings of the 10th international conference on New interfaces for musical expression*, pp. 37–47. ACM, 2010.

[IH87] Tadao Ichikawa and Masahito Hirakawa. Visual programming-toward realization of user-friendly programming environments. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, pp. 129–137. IEEE Computer Society, 1987.

[JGAK07] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction*, pp. 139–146. ACM, 2007.

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

[JK91] Gary Rymar Jeffrey Kodosky, Jack MacCrisken. Visual programming using structured data flow. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, pp. 34 –39. IEEE Computer Society, 1991.

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

[KOR] KORG, Inc. nano シリーズ 2. Retrieved 30 December 2011 from `http://www.korg.co.jp/Product/Synthesizer/nano2/`.

[KS94] Dennis Koelma and Arnold Smeulders. A visual programming interface for an image processing environment. *Pattern Recognition Letters*, Vol. 15, No. 11, pp. 1099–1109, ELSEVIER, 1994.

[KST09] Atsutomo Kobayashi, Buntarou Shizuki, and Jiro Tanaka. ImproV: A system for improvisational construction of video processing flow. In *HCI International 2009: Proceedings of 13th International Conference on Human-Computer Interaction*, Part IV, LNCS 5611, pp. 811–820. Springer, 2009.

[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. National University of Singapore, 2004.

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

[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. Springer, 2006.

[Mav] Mavrick Designs. VisualJockey. Retrieved 30 December 2011 from `http://www.visualjockey.com/`.

[McC96] James McCartney. Supercollider: a new real time synthesis language. In *Proceedings of the International Computer Music Conference 1996*, pp. 257–258. International Computer Music Association, 1996.

[McC02] James McCartney. Rethinking the computer music language: Supercollider. *Computer Music Journal*, Vol. 26, No. 4, pp. 61–68, MIT Press, 2002.

[Miy10] Chikashi Miyama. Peacock: a non-haptic 3d performance interface. In *NIME '10: Proceedings of the 10th international conference on New interfaces for musical expression*, pp. 380–382. ACM, 2010.

[MMLD06] Jon McCormack, Peter McIlwain, Aidan Lane, and Alan Dorin. Composing with nodal networks. In *Proceedings of the 2006 Australasian Computer Music Conference*, pp. 101–107. Australasian Computer Music Association, 2006.

[MMLD07] Jon McCormack, Peter McIlwain, Aidan Lane, and Alan Dorin. Generative composition with nodal. In *Proceedngs of Workshop on Music and Artificial Life*, 13 pages. Interdisciplinary Centre for Computer Music Research, 2007.

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

[Nata] Native Instruments GmbH. REAKTOR. Retrieved 30 December 2011 from `http://www.native-instruments.com/#/en/products/producer/reaktor-55/`.

[Natb] Native Instruments GmbH. REAKTOR Core Technology. Retrieved 30 December 2011 from `http://www.native-instruments.com/#/en/products/producer/reaktor-55/?page=1626`.

[Nil07] Click Nilson. Live coding practice. In *NIME '07: Proceedings of the 7th international conference on New interfaces for musical expression*, pp. 112–117. ACM, 2007.

[Puc88] Miller Puckette. The Patcher. In *Proceedings of the International Computer Music Conference 1988*, pp. 420–429. International Computer Music Association, 1988.

[Puc96] Miller Puckette. Pure data: another integrated computer music environment. In *Proceedings of the International Computer Music Conference 1996*, pp. 37–41. International Computer Music Association, 1996.

[RFS88] Nick Roussopoulos, Christos Faloutsos, and Timos Sellis. An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering*, Vol. 14, pp. 639–650, IEEE Computer Society, 1988.

[Sha87] John A. Sharp. データフロー・コンピューティング（富田眞治訳）. サイエンス社, 1987.

[Sor05] Andrew Sorensen. Impromptu : an interactive programming environment for composition and performance. In *Proceedings of the Australasian Computer Music Conference 2005*, pp. 149–153. Queensland University of Technology, 2005.

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

[Tan90] Steven L. Tanimoto. Viva: A visual language for image processing. *Journal of Visual Languages and Computing*, Vol. 1, pp. 127–139, Academic Press, Inc., 1990.

[TII07] Satoru D. 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. ACM, 2007.

[TIK+09] Stuart Taylor, Shahram Izadi, David Kirk, Richard Harper, and Armando G. Mendoza. Turning the tables: an interactive surface for VJing. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pp. 1251–1254. ACM, 2009.

[Tok09] Satoru Tokuhisa. Nervixxx: A video performance system with neural interfaces. In *ACHI '09: Proceedings of IEEE The Second International Conferences on Advances in Computer-Human Interactions*, pp. 156 –163. IEEE, 2009.

[VGJL07] Nicolas Villar, Hans Gellersen, Matt Jervis, and Alexander Lang. The colordex dj system: a new interface for live music mixing. In *NIME '07: Proceedings of the 7th international conference on New interfaces for musical expression*, pp. 264–269. ACM, 2007.

[vvv] vvvv group. vvvv. Retrieved 30 December 2011 from `http://vvvv.org/`.

[Wan04] Ge Wang. On-the-fly programming: Using code as an expressive musical instrument. In *NIME'04: Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 138–143. National University of Singapore, 2004.

[WC04] Ge Wang and Perry Cook. Chuck: a programming language for on-the-fly, real-time audio synthesis and multimedia. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pp. 812–815. ACM, 2004.

[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, IEEE Computer Society, 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（第 128 回ヒューマンコンピュータインタラクション研究会）, 第 2008 巻, pp. 1–8. 情報処理学会, 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. 情報処理学会, 2008.

[小林 11] 小林敦友, 志築文太郎, 田中二郎. GPU を利用したライブ映像パフォーマンス向け映像合成システム. 情報処理学会論文誌 プログラミング（PRO）, Vol. 4, No. 1, pp. 76–89, 情報処理学会, 2011.

[田中 08] 田中秀明, 平井重行. ビジュアルプログラミング環境向け音声入力支援インタフェースの改良-音声訂正および音声コマンドの導入. インタラクション 2008 論文集 CD-ROM（ポスター発表）, 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（第 75 回音楽情報科学研究会）, 第 2008 巻, pp. 19–24. 情報処理学会, 2008.

[木村 04] 木村健太. VJ シーンの構造 - 映像メディアにおけるモーション表現の可能性. 基礎造形, No. 013, pp. 71–77, 日本基礎造形学会, 2004.

[木村 05] 木村健太. VJ とは何か - 映像メディア表現の新たな可能性について. 大学美術教育学会誌第 37 号, pp. 463–470, 大学美術教育学会, 2005.

著者論文リスト

本論文に関する原著論文

論文誌論文

[1] 小林敦友, 志築文太郎, 田中二郎. GPU を利用したライブ映像パフォーマンス向け映像合成システム. 情報処理学会論文誌プログラミング（PRO）, Vol. 4, No. 1, pp. 76–89, 情報処理学会, 2011.

査読付き国際会議

[1] Atsutomo Kobayashi, Buntarou Shizuki, and Jiro Tanaka. ImproV: A system for improvisational construction of video processing flow. In *HCI International 2009: Proceedings of 13th International Conference on Human-Computer Interaction*, Part IV, LNCS 5611, pp. 811–820, Springer, 2009.

[2] Atsutomo Kobayashi, Buntarou Shizuki, and Jiro Tanaka. Data unification on a dataflow visual language for VJing. In *VLC2011: Proceedings of International Workshop on Visual Languages and Computing*, pp. 268–273, Knowledge Systems Institute, 2011. (Short Paper)

国内口頭発表等

[1] 小林敦友, 志築文太郎, 田中二郎. データフロービジュアル言語におけるデータ調整ノブ. 日本ソフトウェア科学会第 28 回大会（2011 年度）講演論文集, 7 Pages, September 2011.

[2] 小林敦友, 志築文太郎, 田中二郎. ImproV: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. 情報処理学会シンポジウムシリーズ（インタラクション 2009 論文集）,Vol.2009, No.4, in CD-ROM, 2 Pages, 情報処理学会, 2009. （ポスター発表）

[3] 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）, Vol.2008, No.50, 2008-HCI-128, pp. 1–6, 情報処理学会, 2008.

[4] 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 分冊 , pp. 157–158, 情報処理学会, 2008.

その他の論文

[1] Atsutomo Kobayashi, Buntarou Shizuki, and Jiro Tanaka. 3D direction display bracelet for wearable system. In *IWC2011: Proceedings of 4th International Workshop on Cybernics* , 4 Pages(in CD-ROM), Global COE Program: Cybernics, University of Tsukuba , 2011.

[2] 小林敦友, 志築文太郎, 田中二郎. ウェアラブルコンピューティングにおける情報提示様式の評価. 情報処理学会第 72 回全国大会講演論文集, 第 3 分冊 , pp. 71–72, 情報処理学会, 2010.

[3] 内藤真樹, 小林敦友, 志築文太郎, 田中二郎. 円筒型マルチタッチインタフェース. 情報処理学会研究報告（127 回ヒューマンコンピュータインタラクション研究会）, Vol.2008, No.11, 2008-HCI-127, pp. 37–43, 情報処理学会, 2008.

付 録A　被験者実験1に使った質問票

次ページ以降は被験者実験 1 の際、被験者に配布した質問票、及び教示である。

実験時のページ番号は、次ページが 1 ページ目として以降順番に記載されていた。本付録では、本論文を通して一貫したページ番号を記載した。

69–73 ページはアンケートの質問票である。実験者は、被験者に質問票を一枚づつ提示、及び説明し、アンケートに答えてもらった。74,75,76 ページは、それぞれ、実験 1 の教示、ImproV の説明と練習に関する補足、実験 2 の教示である。これらについても一枚づつ提示、及び説明し、被験者に操作を行なってもらった。77–82 ページは実験 2 のタスクごとの教示である。これらタスクは、被験者ごとに順番をランダム化される。実験者は、そのランダム化された順番に沿って、一枚づつ被験者に提示、及び説明し、タスクを行なってもらった。

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

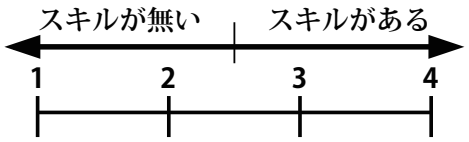
Q2. あなたがライブ映像パフォーマンス (VJ) の準備で行う作業について、作業の手順や気をつけている点、かかる時間など、図もしくは文章を使って教えてください。

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

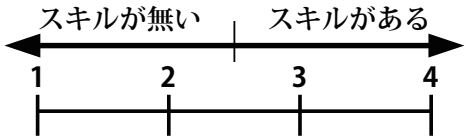
Q4. あなたの Adobe After Effects の使用経験とスキルについて教えてください。
使用経験：__年



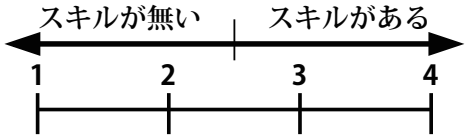
Q5. あなたの Adobe After Effects 以外の映像関連ソフトウェアの使用経験について教えてください。（複数ある場合はよく使う順に 3 つ）

- 経験がない

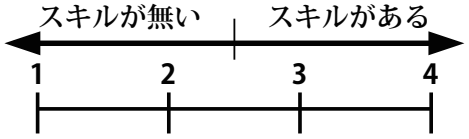
1. ソフトウェア名：
使用経験：__年



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

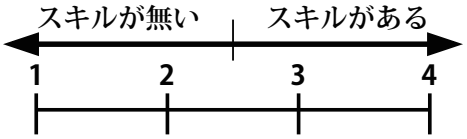


事前アンケート

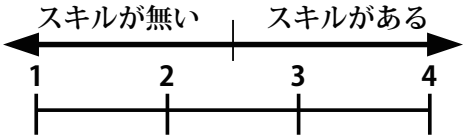
Q6. あなたのコンピュータプログラミングの経験について教えてください。（複数ある場合はよく使う順に 3 つ）

- 経験がない

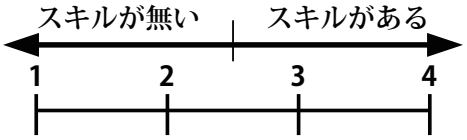
1. プログラミング言語名：
使用経験：__年



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。

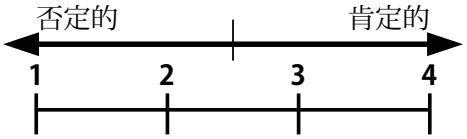
タスク

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

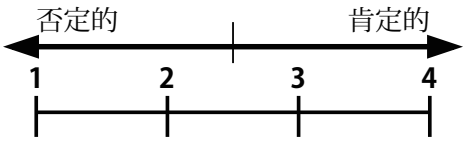
ImproV について、各質問に答えてください。出来れば理由や、コメントも記入してください。

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？



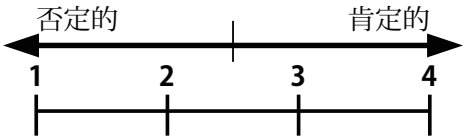
理由・コメント：

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？



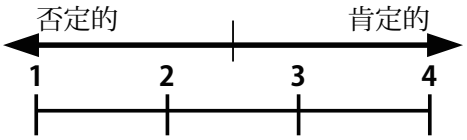
理由・コメント：

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？



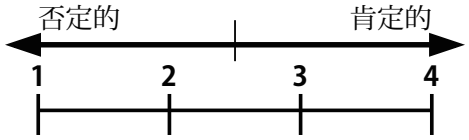
理由・コメント：

Q. プレビューは適切に行えましたか？



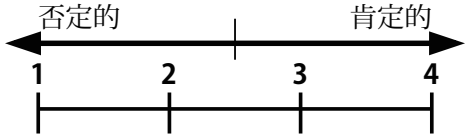
理由・コメント：

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。

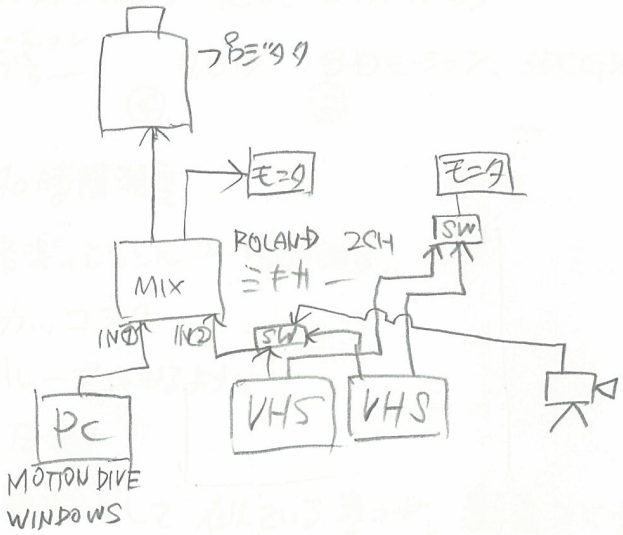
回答欄

付 録B　被験者実験1にて得られた回答

次ページ以降は被験者実験１の「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。」という問いに対して得られた回答を５人分全て示す。ただし、被験者が特定されそうな記述に関しては黒く塗りつぶした。

被験者５人全てが、映像機材をノードとしたデータフロー図を描いていることが確認できる。

回答欄



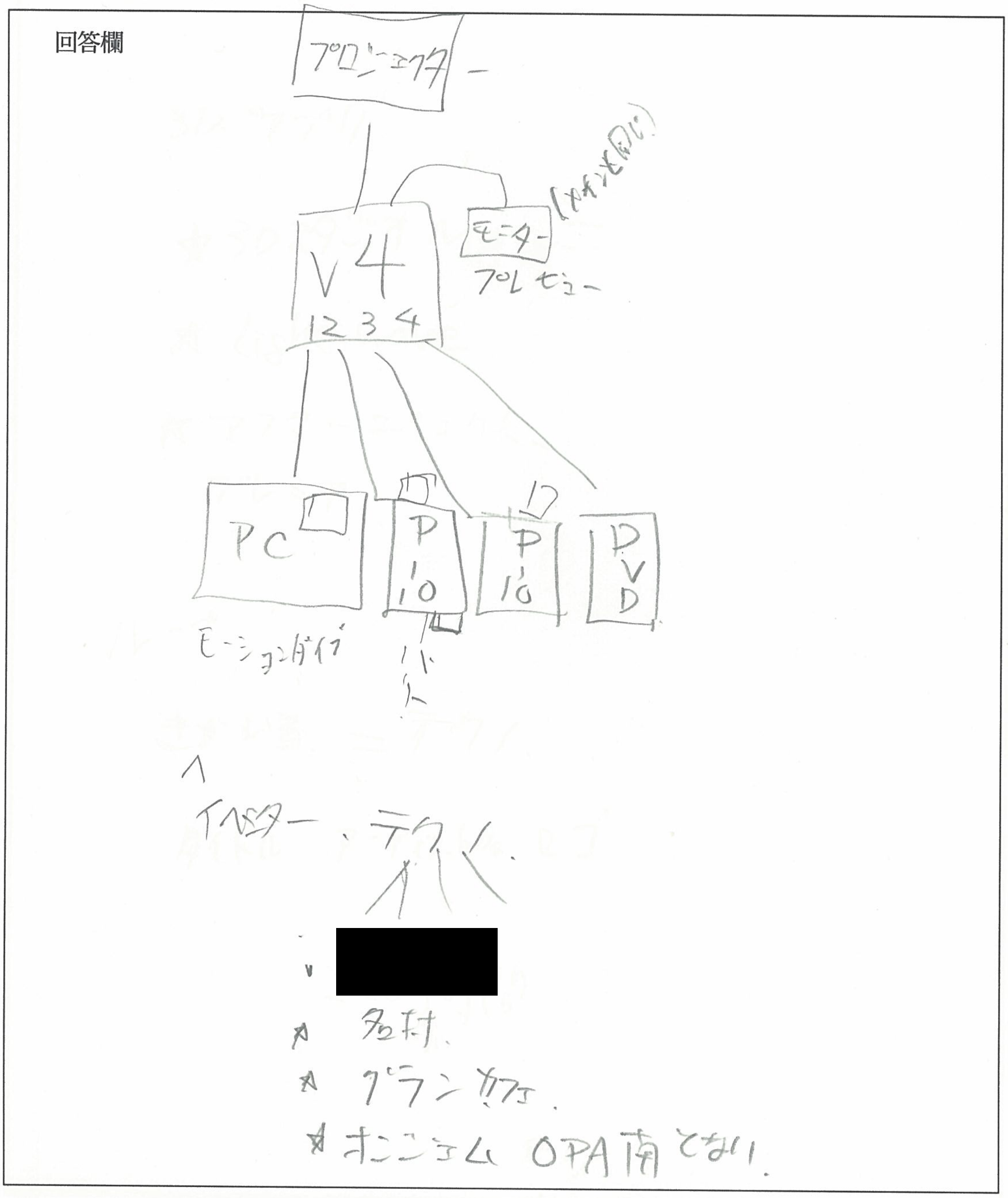
人数：2人 (ダンス、ライブ)

場所：神戸 Bebe1、OTOKA、神戸ポートピアランド[redacted]

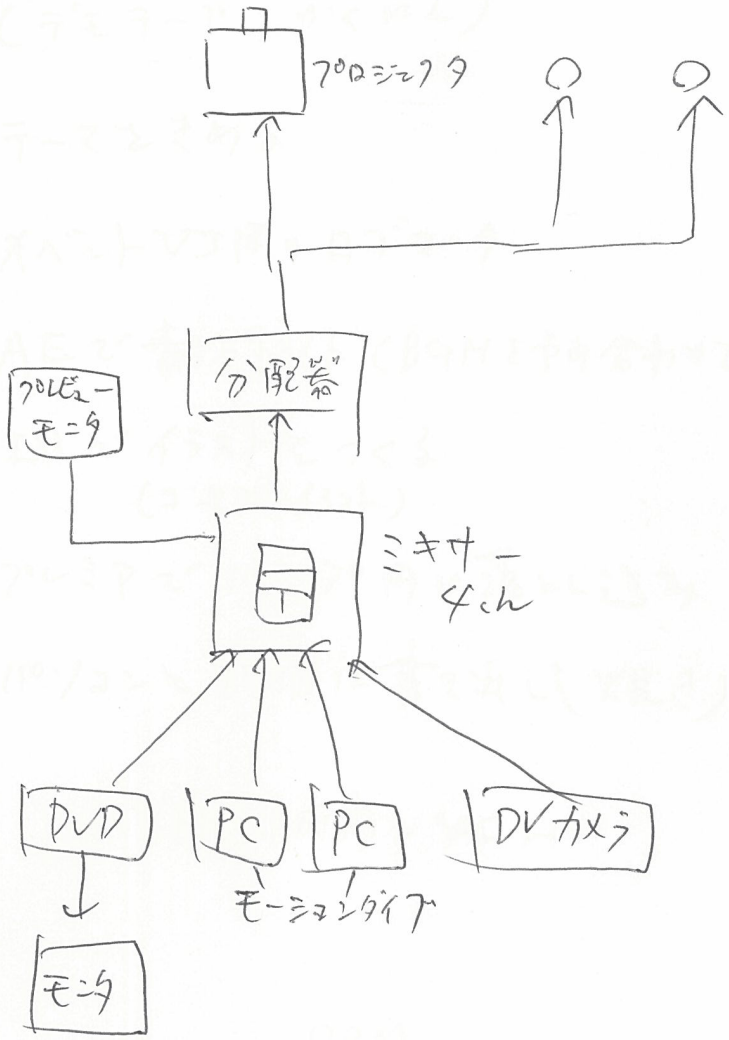
[redacted](大阪アイランド)

ヘアショウイベントVJ

回答欄

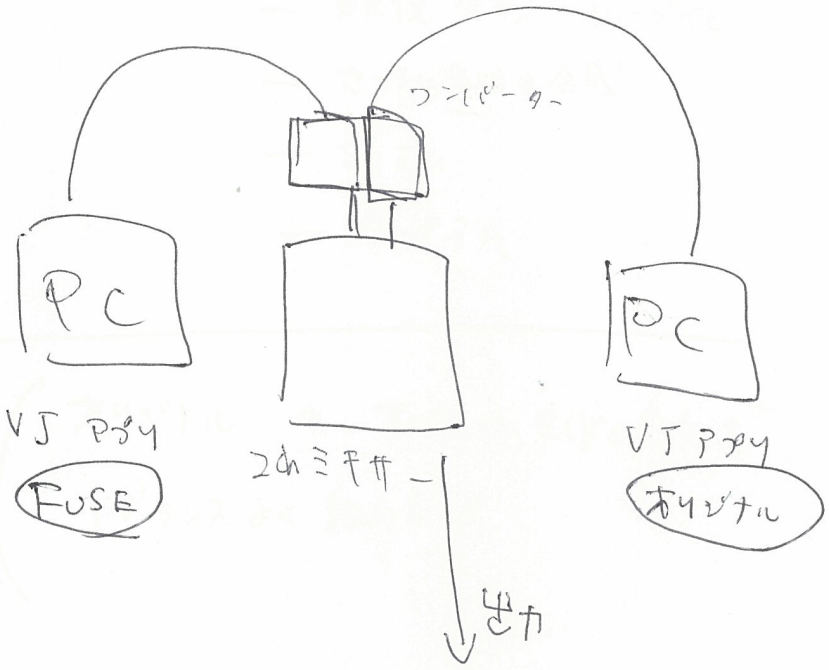

プロジェクター
モニター
プレビュー
V4
1 2 3 4
PC
P 10
P 10
DVD
モーションダイブ
[redacted]
グランカフェ

回答欄



①ヘアショー ②企業パーティー ③レク

回答欄



場所 / クラブ, カフェ, ホール など

内容 / ライブイベント, クラブイベント, 野外イベント, フェス

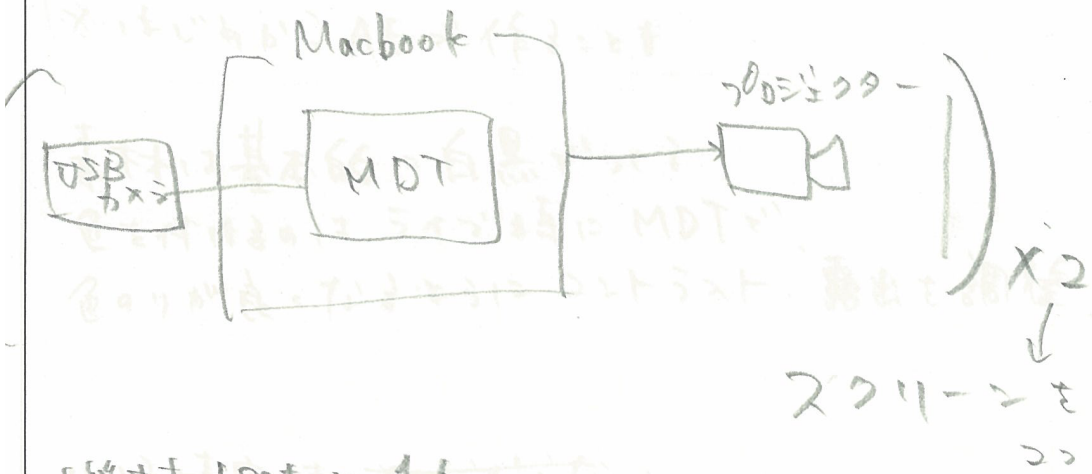
回答欄

機材：Macbook Pro, プロジェクター, USB webカメラ

ソフトウェア：Motion Dive Tokyo → ライブ用

After Effects
iMovie
} → とざいかのエ

・主にライブをやるのは，バーみたいなところ．(Frog)



・

機材1setに1人．

DJイベント，ライブイベント