情報処理学会論文誌　プログラミング　Vol. 4　No. 1　76–89 (Mar. 2011)

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

小林敦友[†1]　志築文太郎[†1]　田中二郎[†1]

我々は，ライブ映像パフォーマンス（以降 VJing）のためのシステム，ImproV の開発を行ってきた．VJing 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，映像を提示し，その映像の生成や切替えなどの制御を人間がその場で行うというパフォーマンスである．ImproV のユーザインタフェースは，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エディタである．これにより，複数の映像を重ねる，複数の映像エフェクトを任意に組み合わせるといった操作を，ユーザが VJing の最中に行うことが可能になっている．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．合成対象の映像も，映像エフェクトのパラメータもすべて映像型で表現するように設計した．これにより映像エフェクトのパラメータとして映像を入力し，そのパラメータを入力された映像に沿って変化させることが可能になっている．この映像型は，GPU のテクスチャを利用して実装されている．このため，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．ImproV のデータフローで扱う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．ノードの実装は High Level Shader Language（HLSL）を使った動的なプラグイン開発と，Node クラスの継承によるプラグイン開発の 2 つを用意している．HLSL を使った動的なプラグイン開発では，プラグイン開発者は 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，新しいノードを生成することができる．このため，プラグイン開発者は 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．また，このようなノードを実装するために役立つライブラリが提供されており，簡単なエフェクトであればピクセルシェーダを記述するだけで実装が行える．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 クラスの継承によってプラグインを開発する．この場合では，独自の GUI を持った映像エフェクトや GPU の機能をすべ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．

A Video Compositing System Using GPU for Live Video Performance

Atsutomo Kobayashi,[†1] Buntarou Shizuki[†1] and Jiro Tanaka[†1]

We have been developed ImproV, a system for live video performance (VJing). VJing is a performance that humans control generation or switching of videos presented to the audiences on events, such as musical events or fashion shows. ImproV allows the users to mix multiple videos and to combine multiple video effects on VJing arbitrary by data flow editor. We employ a unified data type, we call, Video Type which is frame images of videos. We design ImproV to express compositing video and parameters of video effects in Video Type. This allows the user to input a video as a parameter of video effect and to animate the parameter along the video inputted. The Video Type is implemented with GPU textures so that the data passing between video effects is done inside the GPU and realize fast video compositing. We have made nodes in dataflow editor of ImproV to be developed as plugin. ImproV provides two way to imprement nodes, one is dynamic developement in HLSL and the other is developement with node inheritance. With dynamic developement in HLSL, a plugin developer can write an HLSL source code file on any editor while ImproV is running, and then drag and drop the HLSL source code file to create a new node. This enables the plugin developer to develop by trial and error. Also, ImproV provides the liblary for developing video effect. With the liblary, the developer can imprement simple effect only by writing pixcel shader. Features and video effect that can not be impremented only with HLSL, still can be impremented with node inheritance. In this case the developer can create video effects with its original GUI or video effects utilizing all GPU features to extend ImproV.

1. はじめに

ライブ映像パフォーマンス（以降 VJing）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映像が観客に提示され，その映像の生成，切替え，合成などの操作が，VJing の演者（以降 VJ）によってその場で行われるというパフォーマンスである．

ここではイメージをつかんでもらうために従来の典型的な VJing について説明する．**図 1**

†1 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
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba

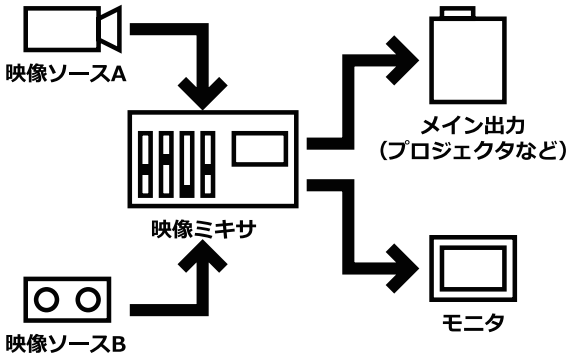


図 1　典型的な VJing の機材配置
Fig. 1　Equipment wiring on a typical VJing.

に典型的な機材の配置を示す．従来の VJing では機材として，複数，多くの場合 2 つの映像ソースとそれらを切り替える映像ミキサ，観客に映像を提示するためのプロジェクタ，VJ が映像を確認するためのモニタが用意される．ここで映像ソースとはライブカメラ，数秒程度のループ映像のプレーヤや，比較的長尺の映像を再生するための DVD プレーヤやビデオテーププレーヤなどである．

次にこれらの機材を使った VJing 中の作業を説明する．VJing では，映像を途切れさせないことと，VJ のセンスやその場の雰囲気に応じた映像を提示することが重要である．VJ は，1 つの映像ソース A によって映像を再生し，それを観客に提示している間に，別の映像ソース B に次に再生する映像を選定する．この際 VJ はモニタディスプレイによって，選定している映像を確認しながら選定を行う．そして，VJ のセンスに叶う映像が選定でき，かつ切り替えるのに良いタイミングであると VJ が判断したら，VJ は映像ミキサによって観客に提示している映像を映像ソース A から映像ソース B へ切り替える．この作業を繰り返すことによって，映像を途切れさせず，VJ のセンスやその場の雰囲気に応じた映像を提示し続ける．

実際には，映像経路の途中に映像エフェクトをつなげることや，入力チャネル数が 2 チャネル以上の映像ミキサを使って 2 つ以上の映像ソースを使うこともある．さらに，これらの構成の機能を 1 台のコンピュータ内において実現するソフトウェアも存在する．

しかし，ここにあげた構成を使ってできることは限られており，映像の切替えを主な目的としている．このため VJ は，VJing を行う前に準備として，事前に映像を用意しておく必要がある．たとえば，VJing において 2 チャネルの入力を持つ映像ミキサを使う場合，最大でも 2 つの映像を重ねることしかできない．2 つ以上の映像を重ねたい場合は，複数の映像を重ねた結果の映像を用意しておく必要がある．また，映像エフェクトに関しても，たとえば映像ミキサの出力に映像エフェクトを適用するように機材を構成することができるが，VJing の最中に別の映像エフェクトを追加することはできない．

これら，映像の重ね合わせ，映像エフェクトの適用は，映像制作の過程において任意に組み合わせて行われるものであるが，従来の VJing システムはそれらの組合せを変更できないという問題があった．

そこで，本研究では，従来 VJing の準備として行われていた映像の合成，つまり映像にエフェクトをかけ加工することや，複数の映像を重ねること，そしてそれらを任意に組み合わせることを VJing の最中に行えるようにすることにより，VJing の即興性を高めることを研究目的とする（即興性とは，その VJing と同時に行われる他のパフォーマンス，すなわち，楽器の演奏，照明，モデルや役者の動きなどと，観客の反応を VJ のセンスを通してパフォーマンスに反映することである）．

この研究目的を達成するために，我々は，VJing のためのシステム，ImproV の開発を行った[1)]．従来の VJing において VJ は，VJing の前に多くの映像を制作しておき，VJing の最中にはそれらを選定，切替えを行うだけであった（事前に行われる映像の制作には VJ のセンスは含まれるが，即興性が欠けている）．

本システムの主な特徴は以下にあげる 3 つである．

(1)　映像の流れをデータフローで表現し，ユーザはデータフローエディタを通してそのデータフローを動的に変更できる．
(2)　データフロー上で扱うデータ型は，我々が映像型と呼ぶ，映像のフレーム画像のみであり，映像型によってパラメータを指定することができる．
(3)　データフロー上で扱うノードは，プラグインとして拡張可能である．

次の 2 章ではまず，ImproV についてユーザの視点から述べ，データフローの構文と映像型によるパラメータの指定について説明する．続く 3 章では，ImproV の映像処理エンジンの実装について述べ，GPU を用いた映像処理や，複数のピクセルシェーダからなるデータフローを動的に変更するための仕組みについて説明する．その次の 4 章で，プラグインの実装方法について説明する．映像型によるパラメータ指定の実装方法もここで示す．

2. ImproV

ここではまず，ImproV についてユーザの視点から説明する．ImproV のユーザインタ

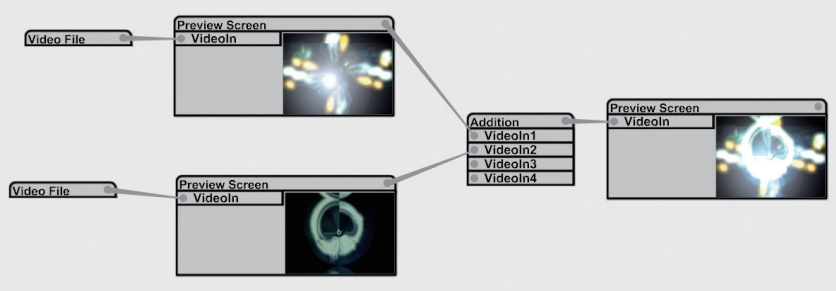


図 2　ImproV のデータフローエディタ
Fig. 2　The data flow editor of ImproV.

フェースは，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エディタである．ImproV のデータフローエディタでは，複数の映像を重ねる，複数の映像エフェクトを任意に組み合わせるといった操作を，ユーザが VJing の最中に行うことが可能になっている．

図 2 に ImproV のデータフローエディタを示す．ここでは 2 つのビデオファイルを加算合成している．

2.1　データフローエディタ

ImproV のユーザインタフェースはプログラミングの知識がなくても，複雑な映像合成を組み合わせることができるように設計されている．このため，ImproV のデータフローエディタ上のノードとエッジは，カメラや映像ミキサといったハードウェアの映像機器とそれらの接続を模している．

ImproV のデータフローエディタで扱うデータ型は映像型のみである．すべてのエッジは映像型の流れを表す．本研究の初期の段階では，映像エフェクトのかかり具合などのパラメータ指定に実数型を用意し扱っていた．しかし，被験者実験の際，実数型や映像型を混在して扱うことが混乱しやすいということが分かったことや，あらかじめ単純なアニメーション映像を用意しておくことにより，パラメータをそれらのアニメーションによって指定できるようになることなどにより，実数型を廃しすべて映像型とすることにした．映像型に含まれるデータは実際には映像のフレーム画像であり，毎フレームごとに評価が行われることにより映像となる．このように実際の処理という視点から見ると画像型やフレーム型と名付ける方が適切であるが，対象ユーザ，つまり VJ から見ると映像型と呼んだ方が分かりやすいと考えこのように名付けた．

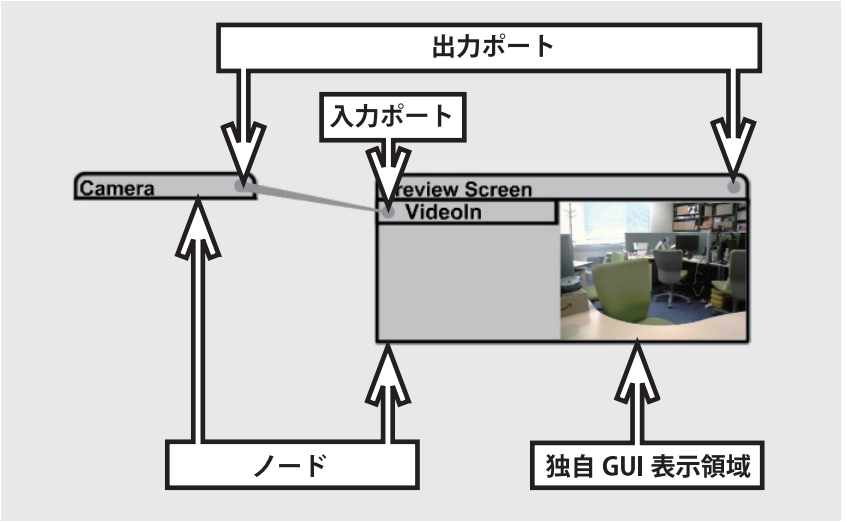


図 3　ImproV のノード
Fig. 3　Nodes of ImproV.

図 3 はノードの説明である．ImproV のノードは，必ず 1 つの出力ポートを持っている．入力ポートに関しては 1 つも持っていないノードや，1 つ，または複数持つノードが存在する．また，独自の GUI を持つノードも存在する．ノードの出力ポートをドラッグすると，出力ポートからエッジが伸び，別のノードの入力ポートにドロップすると，もとのノードの出力ポートとその入力ポートが結線される．出力ポートは複数の入力ポートと結線可能である．この場合は結線された入力ポートすべてに同じ映像が入力される．逆に 1 つの入力ポートは 1 つの出力ポートからのエッジしか結線できない．複数の映像を合成して処理するというデザインも考えられるが，そのようにユーザが複数の映像をまとめたときに意図する合成方法，たとえば加算合成なのかアルファ合成なのか，がユーザによってまちまちであると考えたためこのようにしている．

2.2　ノードの種類

ユーザから見ると，ImproV のノードには 4 種類ある．それぞれ，映像ソース，映像の出力，映像ミキサ，映像エフェクトを表す．以降で順番に説明する．

1 つ目は映像ソースを表す映像ソースノードであり，映像ファイルを再生する “Video File”

やカメラからの映像を取得する “Camera” などがある．これらのノードは入力ポートがなく，出力ポートのみを持っている．図 2 左には “Video File” が 2 つ配置されている．

2 つ目は映像の出力を表す映像出力ノードである．これは今のところ，“Preview Screen” の 1 種類のみである．このノードはユーザのためのプレビューと観客に提示する映像出力の 2 つの役割を持っている．このノードは 1 つの入力ポートと出力ポートを持っている．また，このノードはノード上に独自の GUI として，映像を表示する領域を持っており，入力ポートに入力された映像をそのままその領域に表示する．同様に出力ポートからも入力ポートからの映像がそのまま出力される．このノードをデータフロー上の任意のノードに結線することにより，そのノードからの出力を確認することができる．また，このノード上の表示領域をクリックすることにより表示用ウィンドウが作られる．この表示ウィンドウにも入力された映像が表示されており，このウィンドウを観客に提示するディスプレイやプロジェクタに表示する．また表示ウィンドウが表示されているときは，データフローエディタ上の表記が “Output Screen” と書き替えられ他の “Preview Screen” と区別される．

3 つ目は複数の映像を混ぜ合わせるミキサを表す映像ミキサノードである．アルファ合成を行う “Alpha” や加算合成を行う “Addition” などがある．これらのノードは複数の入力ポートを持ち，入力された映像すべてを混ぜ合わせ，結果を出力ポートから出力する．図 2 中央付近に “Addition” が配置されている．

4 つ目は映像の加工を行うエフェクトを表す映像エフェクトノードである．色調の調整を行う “Color” やぼかしをかける “Blur” などがある．これらのノードも複数の入力ポートを持っている．それらのうち 1 つは加工の対象となる映像であり，その他は加工の程度を示すパラメータである．ImproV では，パラメータとして入力された映像フレームの各座標のピクセル輝度値が各座標におけるパラメータとして処理される．ここで映像を回転させる “Rotation” を例に説明する．**図 4** では左上に表示されている映像が “Rotation” によって回転され右側に出力されている．このとき，回転の角度 “Angle” は “Slider” によって指定されている．**図 5** では左上に表示されている映像が “Rotation” へ入力されている点は同じであるが，回転の角度 “Angle” に左中央の白黒のグラデーション映像が入力されている．出力される画像は，“Angle” に入力された映像の輝度が高い部分ほど回転角度が大きく回転される．図 4 では “Slider” によってスカラ値が扱われているように見えるが，実際には “Slider” は，全ピクセルが指定された値の輝度値である映像フレームを出力する．

2.3 利　用　例

ImproV の典型的な利用例として，従来の VJing においてよく見られる 2 つの映像の重ね合わせおよび切替えと，映像エフェクトを適用する手順を示す．

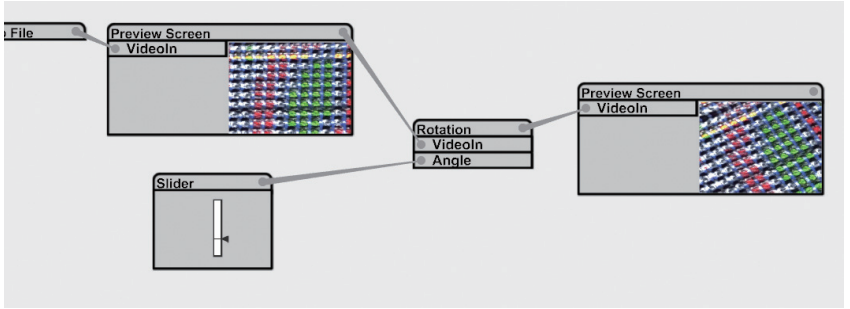


図 4　“Slider” によって “Rotation” の “Angle” を指定しているところ
Fig. 4　Setting “Angle” of “Rotation” with “Slider”.

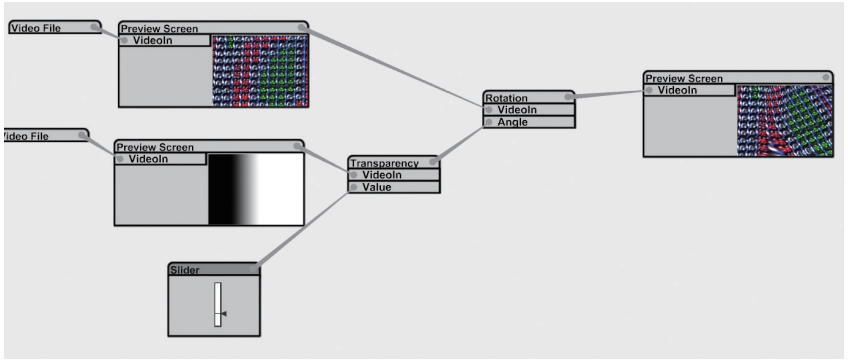


図 5　映像型によって “Rotation” の “Angle” を指定しているところ
Fig. 5　Setting “Angle” of “Rotation” with Video Type.

1 つ目は，2 つの映像の合成および切替えである．**図 6** a では 2 つの “Video File” によって 2 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．ユーザは上側の “Video File” を “Preview Screen” へ接続し，内容を確認した．このときユーザは，観客に提示されている映像を，上側の “Video File” へクロスフェードによって切り替えることを決めた．まずユーザは，図 6 b に示すように，上側の “Video File” に “Transparency” を接続し，“Slider” によって不透明度を調節できるようにした．次にユーザは，図 6 c に示すように，“Mixer” を使って 2 つの映像を重ねた．

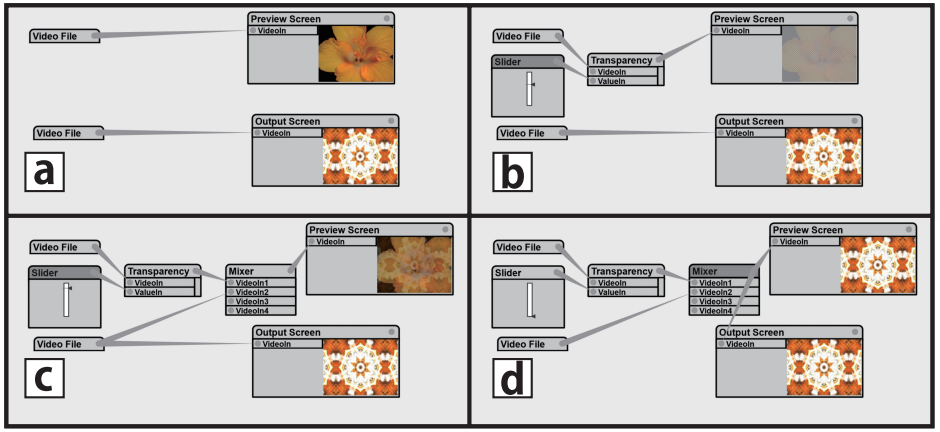


図 6　2 つの映像を切り替えているところ
Fig. 6　Switching two videos.

ここまでの手順の間，ユーザは “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．最後にユーザは，不透明度をいったん 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 6 d 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いき，最後には切り替わる．

2 つ目はエフェクトの適用である．ユーザは，**図 7** a 上側の花の映像に，花の形に沿った光が煌く効果を加えたいと考え，図 7 a 下側の，光芒をともなった細かい光が煌く映像を読み込んだ．しかし，このまま 2 つの映像を重ねても光の形は花の形に沿わない．このため，図 7 b に示すように，ユーザは “Translate” を使うことにより，光芒の映像を花の映像に沿って歪ませた．“Translate” は入力ポート “X”，および，“Y” に入力された輝度に応じて，入力ポート “VideoIn” に入力された映像の座標を移動させる．図 7 b では “X”，および，“Y” に花の映像を入力することにより，“VideoIn” に入力された光芒の映像の各ピクセルが花の映像の該当ピクセルの輝度に応じて移動している．これにより，光芒の映像は，花の形，すなわち花の映像の陰影に沿って歪んでいるように見える．次にユーザは，1 つ目の例と同様に，元の映像と “Translate” の出力を合成した映像を作成する．図 7 c では，加算合成を行う “Addition” を使って元の映像と “Translate” の出力を合成している．また，

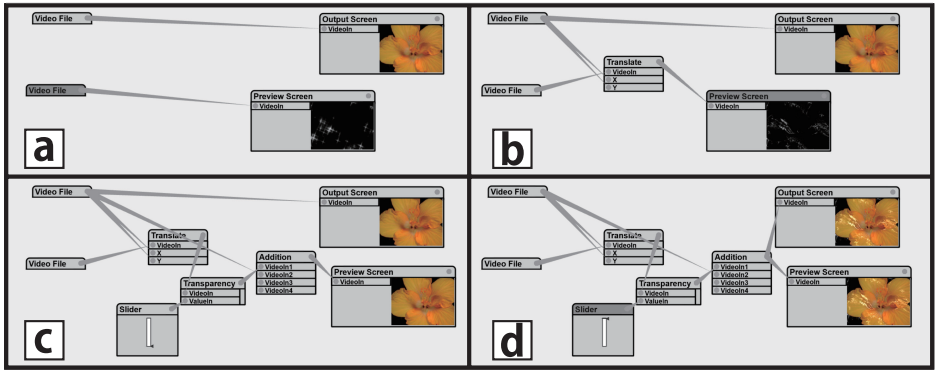


図 7　エフェクトを適用しているところ
Fig. 7　Applying effect.

“Translate” と “Addition” の間に，“Transparency” をはさむことにより，“Translate” の出力の透明度を “Slider” を使って調節できるようにしている．ユーザは，いったん “Slider” の値を下げ，“Addition” の出力を “Output Screen” へ接続する．そして，徐々に “Slider” の値を上げることにより，光が煌く効果がかかっていく．図 7 d は “Slider” の値を上げきった様子である．

ここまでで，ImproV を使って，2 つの映像の重ね合わせおよび切替えと，エフェクトの適用という VJing に必要な操作を行う手順を見てきた．加えて，ImproV ではこれらを実現する構造を動的に構成できることも確認できた．これに加え，ImproV ではすべてのノードの出力が映像型であるため，複数の映像を重ねた結果，または，エフェクトを適用した結果の映像に，さらに別の映像を重ねる，または，別のエフェクトを適用することが可能である．

3. 映像処理エンジンの実装

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．

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ため GPU を用い高速化している．さらに，本システムは，VJ が VJing 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．

また，今後ユーザインタフェース研究として，WIMP ベース以外のユーザインタフェースや，データフローエディタ以外のユーザインタフェースを探索するために GUI 部分の分離も行った．このため，映像合成処理部分はライブラリとして実装されており，.NET Framework アプリケーションに組み込んで利用することが可能である．

以下ではまず，データフローの構造と評価について述べ，次に映像処理について述べる．

3.1　データフローの構造と評価

Node クラスはデータフロー上のすべ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 クラスは前章での “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すべての 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こと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見なし，それをポストオーダによって走査する．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．2 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させない．

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すべての 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，すべての 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すべての 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．

3.2　映 像 処 理

2.1 節で述べたとおり，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とることにより，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を別の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説明する．

ソースコード 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(effect,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 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 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．

4. プラグインシステムの実装

2.2 節で述べたノード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．これらの理由から，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 1 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 のみで記述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ほとんどは，この方法で実装されている．

より複雑なノードは Node クラスサブクラスとして実装することができる．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 のすべての機能を利用することや，.NET Framework のすべての機能を利用したノードを実装することができる．この方法では，独自の GUI を持った映像エフェクトや GPU の機能をすべ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．

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．

4.1 HLSL を使った動的なプラグイン

ソースコード 1 で使われていた 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できる．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合せである．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

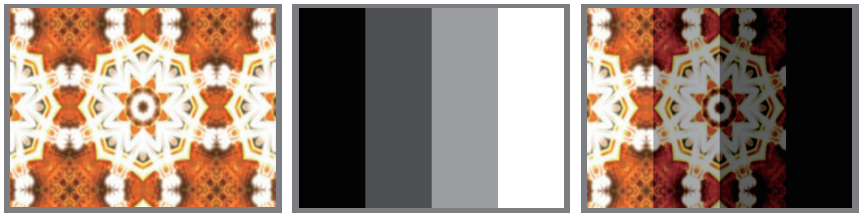


図 8　例として実装する映像エフェクトの適用例　左：加工対象として入力された画像　中央：パラメータとして入力された画像　右：結果として出力された画像
Fig. 8　Video effect implemented on the example. Left: The image input as a source. Center: The image input as a parameter. Right: The image output as the result.

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．

ここでは，2 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，1 つの入力ポートに入力された映像を，もう 1 つの入力ポートに入力された映像に応じて暗くする．

図 8 にこの映像エフェクトの適用例を示す．この例では，図 8 左の画像を，図 8 中央に応じて，図 8 右の結果を出力している．図 8 中央の明るい部分が，図 8 右では暗くなっている．ソースコード 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 2

```
1  #include "../system/ImproVCoreLib.hlsl"
2  Texture2D VideoIn;
3  Texture2D ValueIn;
4
5  float4 Darken( 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;
```

図 9　映像エフェクト Darken のデータフローエディタ上の見た目
Fig. 9　The node of video effect “Darken” in the data flow editor.

```
14
15   return color;
16 }
17
18 technique10 Render
19 {
20   pass P0
21   {
22     SetGeometryShader( NULL );
23     SetVertexShader( CompileShader( vs_4_0, MapVertex() ) );
24     SetPixelShader( CompileShader( ps_4_0, Darken() ) );
25   }
26 }
```

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 2 の中では，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 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．図 9 において，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．

EffectNode クラスは，評価時に 20 行目で宣言されているレンダリングパスを 1 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 ImproVCoreLib.hlsl で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 構造体は頂点座標 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 構造体を生成する．

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 Darken 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ため，Darken 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，Darken 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 で提供される．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 を，value 値分減算することにより「暗く」している．ここでは，RGBA すべてに対して減算を行うと図 8 右の結果が暗くなりすぎるため，説明のために RGB それぞれに対してのみ減算を行っている．

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 は図 5，図 4 で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

```
1  float4 Rotation(PipelineVertex input) : SV_Target
2  {
3
4    float Val=
5      Luminance(ValueIn.Sample(TextureSampler, input.texCoord)) * 2
           * PI;
6    float2 TCoord;
7
8    input.texCoord -= 0.5;
9    float3x3 rotationMatrix={
10     cos(Val),-sin(Val),0,
11     sin(Val),cos(Val),0,
12     0,0,1
13   };
14   TCoord = mul(rotationMatrix,input.texCoord);
15   TCoord += 0.5;
16
17   return VideoIn.Sample(TextureSampler, TCoord);
18 }
```

ソースコード 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．

4.2 Node クラス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 クラスや，USB カメラからの映像をキャプチャするノード CameraNode クラスがあげられる．

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，評価時の振舞いを記述することである．このメソッドは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 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 クラスのソースコードを示す．

ソースコード 4

```
1  using ImproV;
2  using SlimDX.Direct3D10;
3
4  namespace ImproVCoreTest {
5
6    public class BitmapNode: Node {
7
8      Texture2D BitmapTexture;
9
10     public BitmapNode(string filename) : base()
11     {
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で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 行目で 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．

このほか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，たとえばコンストラクタなどにおいて，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ても，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 を持たせることができる．たとえば 2.2 節であげた “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．

5. 議　　論

5.1 映像処理エンジンの性能

ImproV の映像処理エンジンがどれくらいの性能を持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合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合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．

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，2.3 節で述べた，2 つの映像の合成および切替えの例では，“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 VJing 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 4.1 節で示した Darken を追加しながら評価し，評価にかかった時間を計測する．Darken は 6 スロットを使用するエフェクトである．

4.2 節で例に示した BitmapNode を 2 個，4.1 節で示した Darken を 130 個生成した．はじめの Darken の 2 つの入力に，2 個の BitmapNode をそれぞれつなぐ．それ以降の Darken には，前の Darken の出力と，2 つの BitmapNode のうち 1 つをつなぎ，数珠つなぎにした．

これを，それぞれの Darken につき評価を 1,000 回ずつ行い，かかった時間の平均を計測した．n 番目の Darken を評価すると，n 個の Darken 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，およ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 2.67 GHz，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 768 MB）を搭載していた．**図 10** が結果のグラフである．

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 23.4 ミリ秒しかかかっていない．これは FPS に換算すると 42 FPS 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 VJing 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いえる．

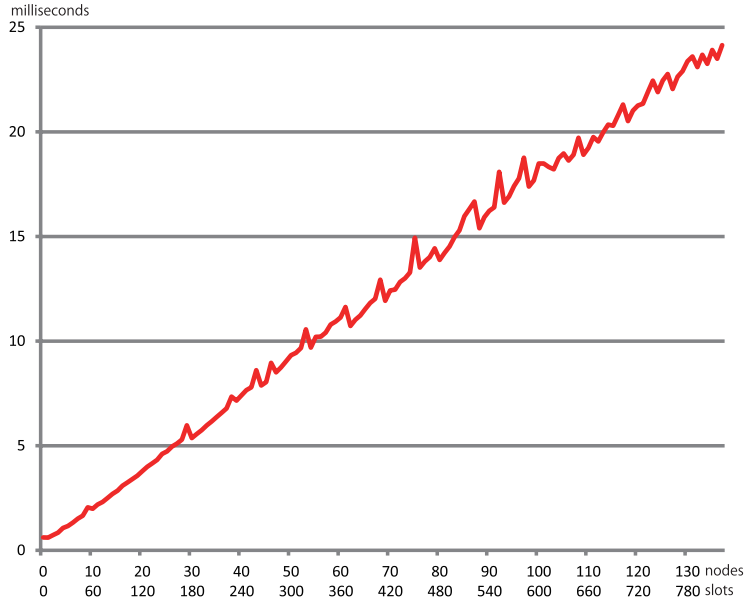


図 10　実験結果：横軸は評価されたノード数および，使用したスロット数，縦軸が評価に要した平均時間
Fig. 10　Average time of calculation for numbers of evaluated nodes and numbers of used slots.

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で操作し，100 回の結線操作を行った．このとき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，30 FPS における 1 フレーム（33.33 ミリ秒）において，ごくわずかを占めるにすぎない．

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．たとえば，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のタイトルを “Loding” と変え，ユーザに読み込み中であることを知らせる）．

5.2　ユーザインタフェース

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[1)]．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，および VJing 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．

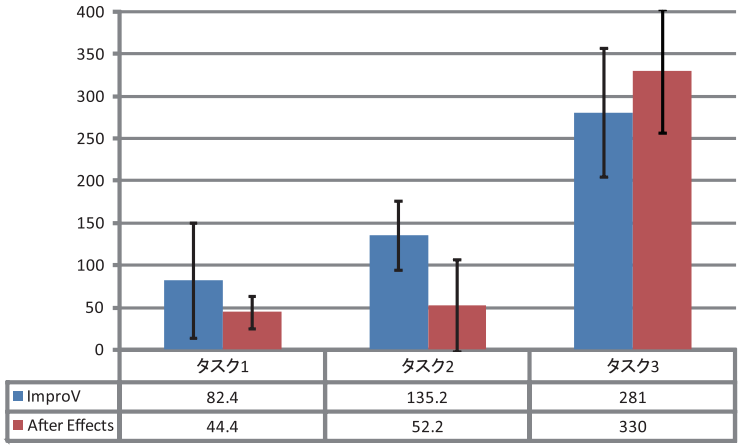
実験 1 の目的は，VJ にとって ImproV が理解しやすいかどうかを確認することである．被験者には，ImproV について説明をせずに ImproV のデータフローを提示し，そのデータフローが何を意味しているか答えてもらった．返答を正確にするため Adobe After Effects で同様の映像合成の構成を再現してもらった．結果は全員問題なく正答し，ImproV のデータフローを理解できていることが確認できた．さらに，事前アンケートでの「あなたがライブ映像パフォーマンス（VJ）で使う機材やソフトウェアの構成について図もしくは文章を使って教えて下さい」という問いに対して被験者全員がデー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，および，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 で再現する，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すべ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 11** に示す．

タスクに要した時間に対して分散分析を行ったところ，タスク 2 において，2 つのシステムに有意差が見られた（$P = 0.0201 < 0.05$）．これは，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



	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 11　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
Fig. 11　Average time (in seconds) and variance to complete each task on the second user study.

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．この結果をふまえ，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 1 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．

さらに，ImproV では，合成対象の映像も，映像エフェクトのパラメータもすべて映像型で表現するように設計した．映像エフェクトのパラメータとしての映像型は，その映像型の各ピクセル輝度値によって，各ピクセル位置のパラメータを指定するものである．データ型が映像型のみであることの利点は 2 つある．1 つ目はユーザの混乱を軽減するという点である．プログラミング経験のない VJ にとって，VJing 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ようである．これは被験者実験を通して，被験者であった VJ から得られたコメントから分かった．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できる．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．2 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できるという点である．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，簡単に実装が行える．4.1 節で述べた Darken エフェクトの例のように，ピクセルの色は RGBA の 4 つの数値からなることや，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，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．

5.3　他分野への応用

ImproV は，映像処理に特化したデータフロー言語といえる．しかし，ImproV で扱う映像型は実数値の配列であるため，ノードの実装を工夫することにより他分野への応用が可能である．たとえば，サウンドカードの音声入力からの信号をテクスチャに書き込むノード，および，テクスチャのピクセル値をサウンドカードへ出力するノードを実装すれば，音響処理に応用できるかもしれない．ImproV は，Single Instruction Multiple Data（SIMD）の処理に特化した GPU を使って処理を行うため，高速な処理や実時間性の高い処理が期待できる．ただし，テクスチャメモリとメインメモリの間でのデータ転送は比較的時間のかかる処理であることに留意する必要がある．音響処理の例でいえば，音響入力から出力までのレイテンシに大きく影響すると考えられる．

6. 関 連 研 究

データフロー型の映像処理システムとして，Cycling ’74 Jitter，Apple Quartz Composer などがあげられる．特に Quartz Composer は，Core Image と呼ぶ，GLSL をベースとしたシェーダ言語によるノードの実装もサポートしており，ImproV と類似している．しかし，Jitter と Quartz Composer は開発環境として設計されており，VJing の最中にデータフローを変更することは考慮されていない．たとえば，Jitter は動的なデータフローの変更に対応していない．また，VJing では観客に提示する前に，データフローに加えた変更を確認するためのプレビューの機構が必須であるが，Quartz Composer はこのような機構を備えていない．

VJing のためのユーザインタフェースの研究として，Bongers らによる Video-Organ [2] や，Tokuhisa らによる Rhythmism [3] 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，VJ が映像生成や合成などのパラメータをコントロールするためのユーザインタフェースの研究である．本研究は映像合成を変更するためのユーザインタフェースの研究であり，これらのシステムと併用することも可能である．

また，Lew は文献 4) において，ライブ音楽パフォーマンスの一種である DJing の作業を分析し，それに基づいた VJing のためのシステムを開発している．しかし，これは従来の VJing を行うためのユーザインタフェースとして研究されている．

Müller らは文献 5) で，VJing の最中に，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 は文献 6) で，映像制作作業と VJing 作業の分離，および，VJing の最中に制作作業の一部を Soundium を使って行うことについて述べている．本研究も，VJing の最中に制作作業の一部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，ユーザに高度な知識を要求する．ImproV は，より高水準のノードを提供することや，すべてのデータを映像型にすることなどにより，より容易なシステムを目指している．

7. ま と め

本論文では VJing のためのシステムである ImproV と，その映像処理部分の実装について述べた．ImproV では映像処理の流れがデータフローによって表され，そのデータフローをリアルタイムに変更することが可能である．データフロー上で扱うデータ型はすべて映像型であり，データフローをシンプルに保ちつつ複雑な映像合成を行うことができる．

ImproV の内部では，GPU を使うことにより，リアルタイムな映像処理を実現している．また，毎フレームごとにデータフローの評価と映像処理を繰り返すことによって，ユーザのデータフロー変更操作を即時に映像処理に反映させる．

また，映像処理を行うノードはプラグインとして拡張可能である．プラグインの実装方法は，.NET Framework のアセンブリとして実装する方法と，HLSL のみによって記述する方法の 2 つの方法を用意している．HLSL のみによって記述する方法は最低限の画像処理プログラミングの知識しか要さず，多くの映像制作者にプログラミングの門戸を開くことが期待できる．

参 考 文 献

1) 小林敦友：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，修士論文，筑波大学 (2008).
2) Bongers, B. and Harris, Y.: A structured instrument design approach: The video-organ, *NIME '02: Proc. 2002 Conference on New Interfaces for Musical Expression*, pp.1–6 (2002).
3) Tokuhisa, S.D., Iwata, Y. and Inakage, M.: Rhythmism: A VJ performance system with maracas based devices, *ACE '07: Proc. International Conference on Advances in Computer Entertainment Technology*, pp.204–207 (2007).
4) Lew, M.: Live Cinema: Designing an instrument for cinema editing as a live performance, *NIME '04: Proc. 2004 Conference on New Interfaces for Musical Expression*, Singapore, pp.144–149, National University of Singapore (2004).
5) Müller, P., Arisona, S.M., Schubiger-Banz, S. and Specht, M.: Interactive Media and Design Editing for Live Visuals Applications, *International Conference on Computer Graphics Theory and Applications*, pp.232–242 (2006).
6) Arisona, S.M.: Live performance tools: Part II, *SIGGRAPH '07: ACM SIGGRAPH 2007 Courses*, New York, NY, USA, pp.73–126, ACM (2007).

(平成 22 年 7 月 5 日受付)
(平成 22 年 11 月 15 日採録)

小林　敦友（学生会員）

2000 年近畿大学商経学部経済学科卒業．2009 年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博士前期課程修了．現在，同大学院同専攻博士後期課程に在学中．ヒューマンインタフェースやビジュアルプログラミング言語に関する研究に興味を持つ．

志築文太郎（正会員）

1971 年生．1994 年東京工業大学理学部情報科学科卒業．2000 年同大学大学院情報理工学研究科数理・計算科学専攻博士課程単位取得退学．博士（理学）．現在，筑波大学大学院システム情報工学研究科講師．ヒューマンインタフェースに関する研究に興味を持つ．日本ソフトウェア科学会，ACM，IEEE Computer Society，電子情報通信学会，ヒューマンインタフェース学会各会員．

田中　二郎（正会員）

1975 年東京大学理学部卒業．1977 年同大学大学院理学系研究科修士課程修了．1984 年米国ユタ大学大学院計算機科学科博士課程修了．ユタ大学では関数型プログラミング言語の並列実装に関する研究に従事．Ph.D. in Computer Science．1985 年から 1988 年に（財）新世代コンピュータ技術開発機構にて並列論理型プログラミング言語の研究開発に従事．1993 年から筑波大学に勤務．現在，筑波大学大学院システム情報工学研究科教授．最近の研究としてはヒューマンインタフェースやユビキタスコンピューティング関連が多い．ACM，IEEE，電子情報通信学会，日本ソフトウェア科学会各会員．2007 年から 2009 年まで本学会理事．