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り替え等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 の最中に行う事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全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 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 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 をもった映像エフェクトや GPU 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。

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 developping 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 developper 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、切換え、合成などの映像に対する操作が、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

†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

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 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、多くの場合二つの映像ソースとを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が提示されていることが重要である。VJ は、一つの映像ソース 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、かつ切り替えるに能うと 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。

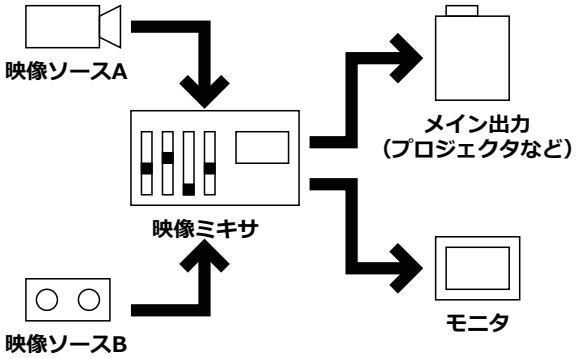

図 1 典型的な VJing の機材配置

我々は、VJing のためのシステム、ImproV の開発を行ってきた[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、切り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、即興性にかけるものである。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等と、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。本研究では、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目的としている。

本システムの主な特徴は以下にあげる三つである。

(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出来る
(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出来る
(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

次の第 2 節ではまず、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。続く第 3 節では、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、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。その次の第 4 節にて、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明らかになる。続いて簡単な評価を含む議論、関連研究を述べ、最後にまとめる。

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 の最中に行う事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。ここでは二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出来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。全てのエッジは映

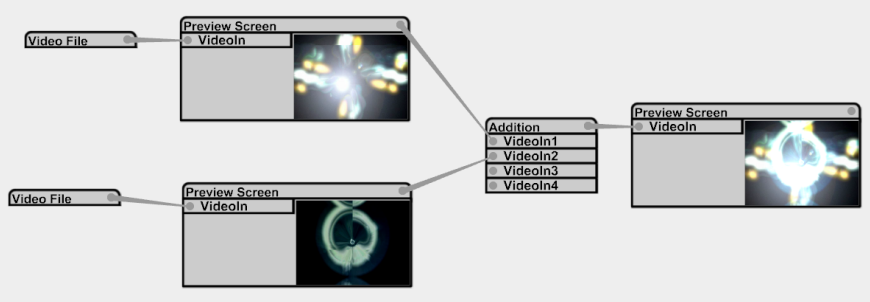

図 2 ImproV のデータフローエディタ

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等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、予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事などにより、実数型を廃し全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わかりやすいと考えこう名付けた。

図 3 はノードの説明である。ImproV のノードは、必ず一つの出力ポートを持っている。入力ポートに関しては一つも持っていないノードや、一つ、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全てに同じ映像が入力される。逆に一つの入力ポートは一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

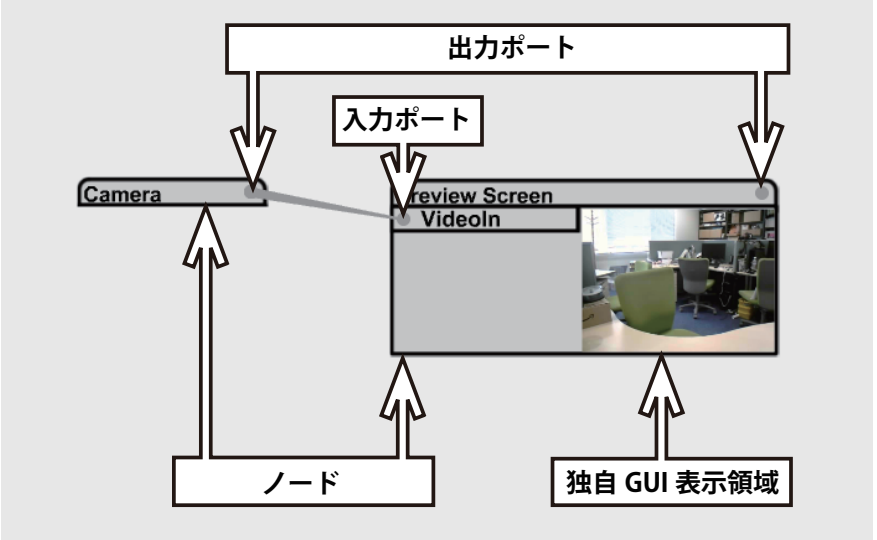

図 3 ImproV のノード

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。

一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無く、出力ポートのみを持っている。図 2 左には “Video File” が二つ配置されている。

二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” の一種類のみである。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二つの役割を持っている。このノードは一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出来る。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” と書き換えられ他の “Preview Screen” と区別される。

三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、入力された映像全てを混ぜあわせ、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。

四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。それらのうち一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” 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” によってスカラー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。

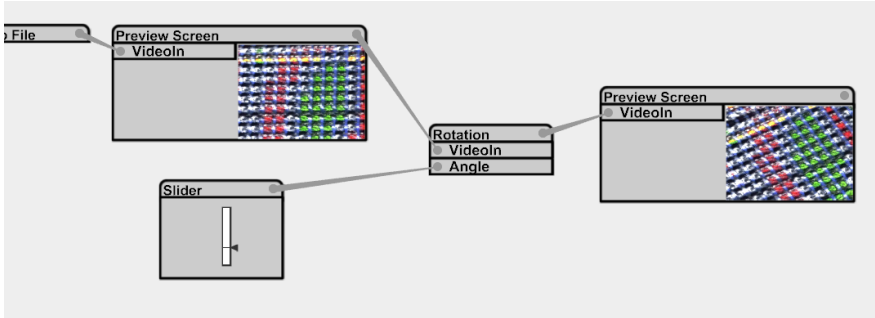

図 4 “Slider” によって “Rotation” の “Angle” を指定しているところ

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。

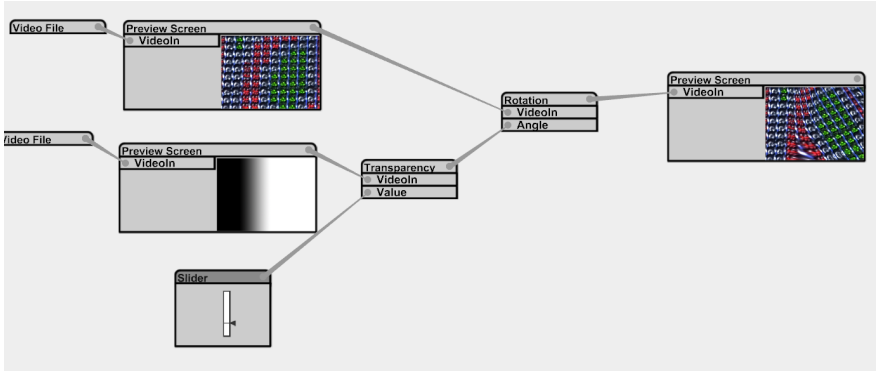

図 5 映像型によって “Rotation” の “Angle” を指定しているところ

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 クラスは前節での “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される事によって映像の処理を行う。ImproV は各フレームごとに、各 OutputNode オブジェクトをルートとしたツリーとして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。

まず、Dataflow オブジェクトの Evaluate メソッドが呼び出される。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の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

第 2.1 節にて述べた通り、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ことは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。

この際留意すべき点は、アルファ合成の方法である。Direct3D 10 はアルファ合成の機構をもっているが、主にゲーム等において使われるものである。これは、例えば背景とオブジェクトの重なり等を表現するための物であり、式 1 のように計算される。

$$ResultColor = UnderneathColor \times (1.0 - TopAlpha) + TopColor \times TopAlpha \quad (1)$$

式 1 の各変数は 0.0 から 1.0 の値をとる。$UnderneathColor$ は下に重ねられる画像の色である。1.0 から上の画像の不透明度 $TopAlpha$ を引いた値、すなわち、上の画像から透けて見える色の濃さが $UnderneathColor$ に掛けられ下の画像の最終的な色が決定されている。これに、上の画像の色 $TopAlpha$ にその画像の不透明度 $TopAlpha$ をかけた、上の画像の最終的な色が足しあわされることで合成結果の色 $ResultColor$ を求めている。

この計算方法を使って複数の画像に対してアルファ合成を行う場合は、アルファ合成を始める前に全ての画像を用意し、それらの重なり順を定め、重なり順が下の画像から順にアルファ合成を行う必要がある。ImproV では、そのデータフローによって合成順序が決定されるため、アルファ合成結果の画像が、次のアルファ合成では上側に重ねられる事もありうる。このようなアルファ合成は式 2、式 3 によって求められる。

$$ResultAlpha = UnderneathAlpha + TopAlpha - UnderneathAlpha \times TopAlpha) \quad (2)$$

$$ResultColor = \frac{UnderneathColor \times (1 - TopAlpha) \times UnderneathAlpha + TopColor \times TopAlpha}{ResultAlpha} \quad (3)$$

まず、アルファ合成が複数回行われるためには、一回のアルファ合成の結果画像にアルファ値 $ResultAlpha$ が含まれている必要がある。これを、式 2 によって求める。式 2 では、上下の画像のアルファ値の合計から、それらを掛け合わせたものを引いている。この掛け合

わせたものを引いている項は、例えば半透明（アルファ値 0.5）の板を 2 枚重ねても不透明度は上がるが、依然透けて見えるという現象を再現するための物である。単に合計だけであると、この場合では $0.5 + 0.5 = 1.0$ と完全に不透明となってしまう。式 2 では、これが合計（$0.5 + 0.5 = 1.0$）から、半分のさらに半分 $(0.5 \times 0.5 = 0.25)$ 透けて見える分を引いている。

次に、式 3 によって合成結果の色 *ResultColor* を求める。式 3 の分子は式 1 と殆ど同様である。ただし、アルファ合成が複数回行われるということから、下の画像もアルファ値 *UnderneathAlpha* を持っているため、下の画像の最終的な色にこれを掛けている。そして最後に、この結果の色を *ResultAlpha* で割っている。こうすることにより、次回アルファ合成が行われる時には、上に重ねられるときには *TopAlpha* として、下に重ねられるときには *UnderneathAlpha* として、*ResultAlpha* が掛けられ、望んでいた色となる。ImproV では、このアルファ合成がなんか以下繰り返され、最終的には黒色 $(RGBA = (0.0, 0.0, 0.0, 1.0))$ の画像とアルファ合成が行われる。

このデータフローに基づく映像処理エンジンは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利用方法を説明する。

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(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 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

第 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がるかもしれない。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 のみで記述することが可能になるようにした。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、こちら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 GPU 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。

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 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。

ここでは、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を、もう一つの入力ポートに入力された映像に応じて暗くする

図 6 にこの映像エフェクトの適用例を示す。この例では、図 6 左の画像を、図 6 中央に応じて、図 6 右の結果を出力している。図 6 中央の明るい部分が、図 6 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;
```

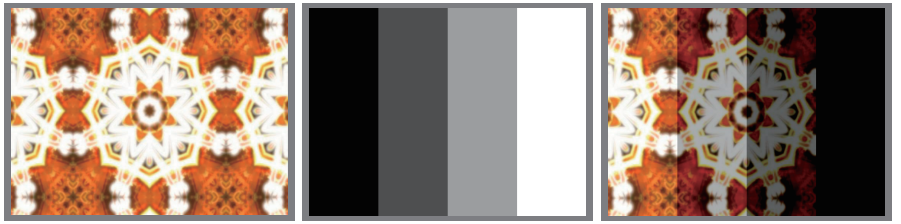

図 6 例として実装する映像エフェクト 左:加工対象として入力された画像 中央:パラメータとして入力された画像 右:結果として出力された画像

```
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;
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 7 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 7 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。

図 7 映像エフェクト Darken 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 行目で宣言されているレンダリングパスを一回実行する。

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 ImproVCoreLib.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 (0.0,0.0)、右下隅 (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 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 を、value 値分減算することにより「暗く」している。ここでは、RGBA 全てに対して減算を行うと図 6 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 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、評価時のふるまいを記述することである。このメソッドは評価時に呼び出され、その時の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 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。例えば第 2.2 節で挙げた “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

ImproV 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簡単な実験を行った。第 4.2 節にて例に示した BitmapNode を 2 個、第 4.1 節にて示した Darken を 130 個生成した。始めの Darken の二つの入力に、2 個の BitmapNode をそれぞれ繋ぐ。それ以降の Darken には、前の Darken の出力と、二つの BitmapNode のうち一つを繋ぎ数珠つなぎにした。これを、それぞれの Darken につき評価を 1000 回づつ行い、かかった時間を計測した。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。

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。図 8 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 1000 回評価するのにかかった時間である。おおよそ線形になっていることが確認できる。

最も時間がかかった 130 ノード評価時でも、23377 ミリ秒しかかかっていない。これは FPS に換算すると 42FPS 出ており、十分実用的といえる。評価時間はノードの種類に大きく依存するが、筆者らが ImproV を使って VJing を行った際には多くても 20 ノード程度であったことを考えると、十分な性能である。

また、ImproV の映像処理エンジンは映像処理のデータフローを動的に変更可能である。結線の変更に関して、エッジ生成、消滅やエッジの入力/出力ポートの参照変更などのオーバヘッドを含んでいるが、筆者が実際に VJing を行った限りにおいては体感できる物ではなかった。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

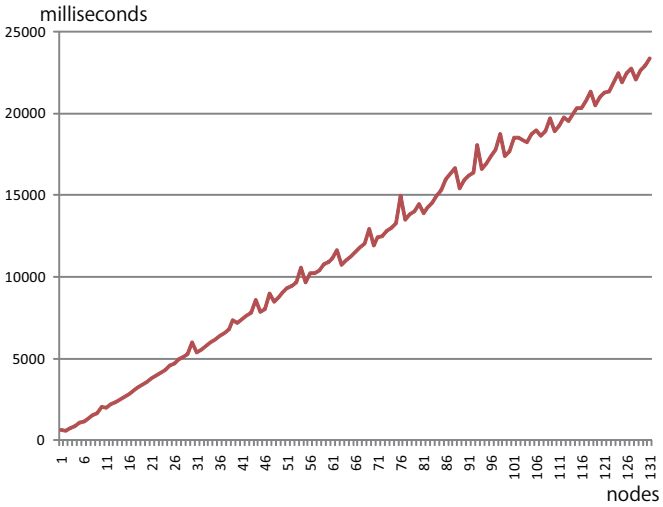

図 8 実験結果:横軸は評価されたノード数、縦軸が 1000 回評価するのにかかった時間

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 “Loding” と変え、ユーザに読み込み中である事を知らせる。)

ImproV は対象ユーザを VJ とし、そのデータフローエディタは映像機器の接続を模した高水準なものとなっている。被験者実験を通して、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ている[2)]。

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全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二つある。一つ目にはユーザの混乱を軽減する点である。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わかった。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。もう一つの利点は、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。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、簡単に実装がおこなえる。第 4.1 節にて述べた Darken エフェクトの例ように、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、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。

6. 関 連 研 究

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[3)] や、徳久らによる Rhythmism[4)] などがあげられる。Video-Organ では、音楽のシステムに用いられるライブ操作のためのハードウェアコントローラをその操作部品ごとに分解し、それらを映像の様々なパラメータにマッピングすることが試みられている。Rhythmism はマラカスにセンサーを取り付け、ユーザはそのマラカスを振ることによって映像を操作する。これらは、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 5) 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 6) にて、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 7) にて、映像制作作業と VJing 作業の分離、及び、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、全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全て映像型であり、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出来る。

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二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) Kobayashi, A., Shizuki, B. and Tanaka, J.: ImproV: A System for Improvisational Construction of Video Processing Flow, *HCI International 2009: Proceedings of 13th International Conference on Human-Computer Interaction*, Part IV, LNCS 5611, pp.811–820 (2009).
2) 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).
3) Bongers, B. and Harris, Y.: A structured instrument design approach: the video-organ, *NIME '02: Proceedings of the 2002 conference on New interfaces for musical expression*, pp.1–6 (2002).
4) Tokuhisa, S.D., Iwata, Y. and Inakage, M.: Rhythmism: a VJ performance system with maracas based devices, *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*, pp.204–207 (2007).
5) Lew, M.: Live Cinema: designing an instrument for cinema editing as a live performance, *NIME '04: Proceedings of the 2004 conference on New interfaces for musical expression*, Singapore, Singapore, National University of Singapore, pp. 144–149 (2004).
6) Müeller, 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).
7) Arisona, S.M.: Live performance tools: part II, *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, New York, NY, USA, ACM, pp.73–126 (2007).