DSL (Data Shaping Lines)

version: v.pre-alpha-0.1
author: innsbluck

定義

DSL (Data Shaping Lines) とは、2次元の画像データを高速かつフレキシブルに処理するためのsledge独自の言語および記法です。

主に以下のような用途で使用されます。

1. 各レイヤーの色調補正・エフェクト追加。
2. インポートした画像の色調補正・エフェクト追加。
3. 別のレイヤーから参照した情報(side-chain)を用いたエフェクト処理。
4. 各レイヤーを合成し、 1 枚の画像にまとめる。

1、2，3 のような要素ごとに対するエフェクト処理のほか、4のようなペイントソフトの根幹とも言える**レイヤー合成と出力画像の生成処理**であっても、DSLを用いて記述できます。

> 1、2，3 のような各レイヤーや要素のみに機能するDSLを`Layer-level DSL (LDSL)`と呼びます。
> 4 のように、レイヤー等を取りまとめて一枚の画像に集約するDSLを`Image-level DSL (IDSL)`と呼びます。

記法

> ここでは、DSL＝`Layer-level DSL`であるものとして解説します。

概念

DSLは、一言で表すなら **「2次元画像データのストリーム」** です。

```
in
> contrast(50%)
> invert()
> brightness(-40%)
> jpeg_glitch(40, 90, 0.001) > out;
```

入力ポート(`in`)から入力された画像データは様々なエフェクトを通過し、 最終的にエフェクトが適用された画像データが出力ポート(`out`)に送信されます。

この`in`と`out`にあたる入力元と送信先は、画像データでさえあれば**なんでも**指定できます。

例えばレイヤーの加工においては、
入力ポート(`in`)はエフェクト適用前のレイヤーの画像データであり、
出力ポート(`out`)はプレビューやエクスポートに使用されるレイヤーの「表示先」を意味します。

> DSLの思想や構文は、以下の概念に大きく影響を受けています。

> - Linuxのshellにおけるパイプライン
> - DAW(digital audio workstation)におけるFXチェーン、サイドチェイン

基本構造

DSLの記法には、関数・クラスの定義やブロック、スコープといった概念は存在しません。

ユーザーは`node`と呼ばれるエフェクトコマンドを選択し、それらをレイヤーの入出力の間に追加していくことで、エフェクトの逐次処理を簡潔かつ直感的に記述できます。

例えば、「レイヤー0の画像のコントラストを50%上げて、色を反転させる」というエフェクトは、以下のような`DSLScript`で記述できます。

```
inout layer_0;

in > contrast(50%) > invert() > out;
```

この`DSLScript`は

- 入出力レイヤーの指定(`inout`, `in`, `out`)
- エフェクトの`node`(`contrast(50%)`, `invert()`)
- 各`node`間の区切り文字 `>`
- 各行の区切り文字 `;`

で構成されています。

本章で登場する`DSLScript`は、この4要素を覚えておけば直感的に理解できます。

即時性と追従性

上記の例で注目すべき点は、`>`で繋がれたパイプラインが示す通り、この記述は一回適用して終わる使い捨てのエフェクトではない、という点です。

> `DSLScript`**はレイヤー内容の更新に即座に追従して描画状態を更新します。**
> また、`DSLScript`における`node`の追加/編集/削除も、エフェクトの変更として即座に描画に反映されます。

`inout`は、エフェクトの入力元レイヤーおよび出力先レイヤーを示します。

`inout`は`in`と`out`宣言をまとめたものであり、主に色調補正などの**あるレイヤーの情報が、同レイヤーの出力を決定する**ような効果を記述する場合は、`inout`が使えます。

レイヤー間のやりとり

入力元と出力先のレイヤーが異なる場合、`inout`の代わりに`in`と`out`宣言を分けて使用できます。以下は、**元絵に追従するグロー効果**のサンプルです。

```
in layer_main;  # 元絵レイヤー
out layer_blur; # ぼかし用レイヤー(元絵の下)
```

```
blank() > out; #グロー効果用のレイヤーを初期化
in > blur(3px) > out; #元絵を3pxぼかした画像をlayer_blurに出力
```

このDSLScriptにより、ユーザーがlayer_mainに描いた絵は即座に処理され、 ぼかしを入れた元絵がlayer_blurに入ります。

このように、sledgeのDSLでは**あるレイヤーから得た情報を別のレイヤーに出力することもできます。**

グロー効果の他にも、

- 自由なレイヤーでのクリッピング
- レイヤーから抜き出した主要な色をカラーパレットとして別レイヤーに出力

など、他のペイントソフトでは手動で行うであろう複雑な操作も、DSLScriptで記述することで保守性を保ちつつも自動化することができます。

> これらの複雑なエフェクトも、**後から変更可能**かつ**関係するレイヤー・画像の変化に自動で追従**して適用されます。

各種ノード: node

> 各nodeの詳細については DSL_nodes_doc_jp.md を参照して下さい。

一時メモリ: subout

概要

suboutは、一言で表せばDSLにおける「変数」です。 もう少し詳しく言うならば、入出力の結果を一時的に保存する「画像のメモリ」ともいえます。

suboutは主にデータの分配や分岐に使用されます。 そのほか、範囲分割系のnodeにおいては multi(subout_area, subout_outside)のように**複数の分割された画像の一時保存先**としても使用できます。

宣言

suboutの宣言はin, outの宣言とパイプライン記述の間で行われます。initステートメントの後にsubout名を記述することで**キャンバスと同サイズの画像データがメモリ上に確保されます**。

```
inout layer_0;
init temp_out;

in > temp_out;
temp_out > out;
```

上記のDSLScriptは実際のところ何の影響も及ぼしません。すなわち、この処理におけるin, temp_out, outは全て全く同じピクセルデータを持つ画像データを示します。 initialization)のための構文として、実装が検討されています。

高度な入出力: in(layer_x), multi(*o1, *o2, ...)

> in(layer_x)もしくはout(layer_x)は`in`, `out`ステートメントで宣言されたレイヤー**以外**のレイヤーの入出力を実現するノードです。**現行のバージョンでは実装の予定はありません**。代わりに、複数の空レイヤー及びDSLを用いて擬似的に表現することを検討して下さい。

multi(*o1, *o2, ...)

`multi`ノードは出力先として機能するという点で`out`や`subout`に近いですが、 **入力された複数の画像データを複数のsuboutに出力する**役割を持ちます。

`multi`ノードを使用するケースとして最も一般的なのは、`splitV`といった**分割系**のnodeが出力する、範囲/範囲外の2つの出力をそれぞれキャッチし、別々にエフェクト処理を行うケースです。

```
inout layer_1;

init merged;

init upper;
init lower;

# 画像の上半分がupper、下半分がlowerにそれぞれ格納される
in(layer_1) > splitV(50%) > multiout(upper, lower);

upper > jpeg_glitch(9, 72) > merged;
lower > invert() > merged;

merged > out(layer_1);
```

アサーションとディレクティブ

アサーション

> 現行バージョンでは実装されません。

アサーションは、主にモジュール化を目的とする`DSLScript`コードの保守、テストのために用いられます。例えば、subout等がinitステートメントによって正常に初期化されているかをテストし、実行用とは異なるテスト用のインタプリタ(DSL Test Runner)でこれを実行することで、`DSLScript`が望んだ動作を行うことを検証できます。

```
@exist subout1;
```

ディレクティブ

ディレクティブが提供する機能は、保守・テスト用のアサーションに比べて実際の実行に寄った役割があります。具体的には、

- 全体、行、ノードごとでのログ出力（sledge内のデバッグコンソールで閲覧可能）

- 既に内容があるsubout にさらに出力を送信した際の挙動(`combine mode`)の定義

などがあります。以下は、ディレクティブを使ったログ出力と挙動設定の例です。

```
inout layer_1;

[combine=stack-up] # 既存のピクセルの上に上書き
# [combine=stack-down] # 既存のピクセルの下に上書き
init merged;

init upper;
init lower;

[log_trace_line] # 下の1行の中間ログを出力
in(layer_1) > splitV(50%) > multiout(upper, lower);
upper > jpeg_glitch(9, 72) > merged;
lower > invert() > merged;
merged > out(layer_1);
```

上記の例における`log_trace_line`が出力するログには各ノードの出力結果が**画像として**含まれますが、sledgeの内部デバッグコンソールは文字情報と同時にこれらの画像を出力できるよう最適化されています。

検討されている機能

unused subout omitter

`DSLScript`には`subout`が本来不要か、もしくは短縮できるケースが存在します。

```
inout layer_0;
init temp_out;

in > temp_out;
temp_out > out;
```

そこで、省略可能な部分を自動で検知し、`in > out`のような形へ最適化する機能(unused subout omitter)が検討されています。

instant subout initialization

`*subout`は`init`ステートメントを使わない即時宣言(instant subout initialization)の方法として検討されています。

```
in > splitV(30%) > multi(*upper ,*lower);
```