# グラフィックデザイナーのための視覚化プログラミング環境に関する基礎検討

P00085 高畠 綾子　　指導教授：藤吉弘亘

## 1. はじめに

近年, 個人が扱う情報の多様化, 量の膨大さから文書や画像等の情報を視覚化する技術の需要が高まっている. このような情報の視覚化においては, プログラミングが必要とされている. しかし, 既存のプログラミング環境では, 動的情報の視覚化により出力された画像を印刷物等の高解像度の媒体で二次利用することができない問題がある. そこで本研究では, グラフィックデザイナーのための視覚化プログラミング環境に関する基礎検討として, 出力画像の二次利用を可能とするマウスによる動的情報の視覚化を実現するプログラミング環境の構築を目的とする.

## 2. 視覚化を目的とした既存システム

プログラミングにより情報を視覚化したグラフィック作品やデザイナーが注目を集めている. プログラムを用いると繰り返し処理や外部データを利用した処理により, 今迄に無い表現が可能となる. これらを実現する既存の環境として, 以下の 3 種類が挙げられる.

**DBN** Maeda によって開発された Design By Numbers (DBN) はデザイナーがプログラムを学ぶ際に有効とされるシステムである. DBN はアニメーションの作成やマウスイベントに反応する平面図形の表示が可能である. しかし, コマンド数が少ないため他のシステムに比べ表現能力が劣るという欠点がある.

**Processing** Fry と Reas は, DBN の欠点の多くを改善したシステム Processing を開発した. このシステムは, DBN において表現不可能であった図形の 3 次元表示や外部データ (JPEG, GIF イメージ, カメラ映像等) の取扱いが実装されている.

**Adobe Scripting** Adobe Scripting は Adobe Systems 社が開発・販売を行っている Illustrator, Photoshop 上で動作するスクリプト言語である. アニメーションの再生やマウスイベントへの対応等の動的情報を用いる描画を扱うことができない. しかし, 静的なイメージを作成する際に Illustrator, Photoshop の機能を利用することができるため, 複雑な図形の加工を容易に行うことができる. また, 出力したデータはベクター形式である.

上記 3 種類のシステムの比較を表 1 に示す.

表 1：既存システム比較

| システム | 出力形式 | 静的な描画 | 動的な描画 | 二次利用の可能性 |
|---|---|---|---|---|
| DBN | ラスター | ○ | ○ | × |
| Processing | ラスター | ◎ | ◎ | × |
| Adobe Scriptng | ベクター | ◎ | × | ◎ |

表 1 より, DBN や Processing の出力はラスター形式である. そのため, 拡大縮小や変形の際に生じるジャギが発生し, 紙媒体等での二次利用が困難となる場合が多い. 一方, Adobe Scripting の出力はベクター形式であるため, 二次利用が可能である. しかし, 動的な要素を含むグラフィックの作成ができないという問題点がある.

## 3. 視覚化のためのプログラミング環境

本研究では, 図 1 に示すような動的な入力情報をプログラムを用いて視覚化し, そのイメージを手作業の介入により二次利用・加工することが可能となるプログラミング環境を実現する. 入力には, 動的情報としてマウス動作や音声, 動画等を扱い, ユーザの作成したプログラムの実行により情報の視覚化を行う. プログラムの実行結果はベクター形式に出力され Illustrator 上に表示される. 次に, Illustrator 上で手作業による加工を施すことで, プログラミングにより出力された図形に人間の曖昧な感覚を付加することができる. このように, プログラミングと手作業による加工を融合することで新しい表現が可能となる.

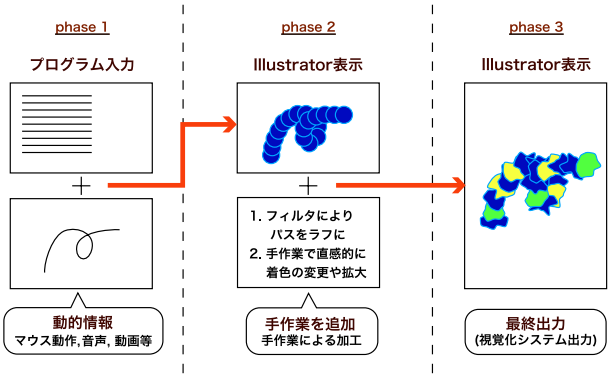
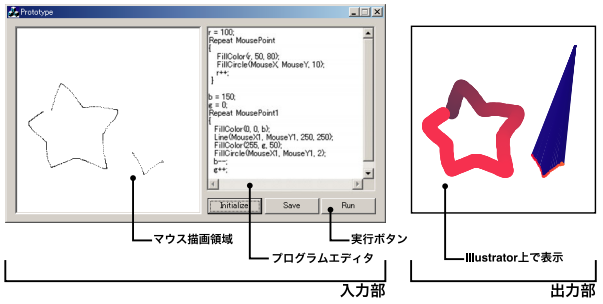



図 1：提案する視覚化プログラミング環境

## 4. プロトタイプシステムの作成

今回は, 動的情報としてマウス座標を基に図形を描画するプロトタイプシステムを作成した (図 2 参照). ユーザは動的情報としてマウス描画領域内のドラッグによりマウス座標を入力し, その座標を利用するプログラムを入力部のプログラムエディタに入力する. 次に, 実行ボタンを押すとプログラムの構文チェックを行い, 描画結果を Illustrator 上に表示する.




プログラム

```
r = 100;                                  変数 r の設定
Repeat MousePoint                         1回目のドラッグ時に得た座標数分処理を繰返す
{
  FillColor(r, 50, 80);                   塗り潰し時の色の設定
  FillCircle(MouseX, MouseY, 10);         塗り潰された円の描画設定
  r++;                                    変数 r のインクリメント
 }

b = 150;                                  変数 b の設定
g = 0;                                    変数 g の設定
Repeat MousePoint1                        2回目のドラッグ時に得た座標数分処理を繰り返す
{
  FillColor(0, 0, b);                     塗り潰し時の色の設定
  Line(MouseX1, MouseY1, 250, 250);       線を描画する設定
  FillColor(255, g, 50);                  塗り潰し時の色の設定
  FillCircle(MouseX1, MouseY1, 2);        塗り潰された円の描画設定
  b−−;                                    変数 b のデクリメント
  g++;                                    変数 g のインクリメント
}
```

図 2：プロトタイプシステム

図 2 に示すユーザが記述したプログラムは, 大きく 2 つの処理に分けられる. 1 つは 1 回目のドラッグ時に得たマウス座標に対して色調を変化させた円を描画する. もう 1 つは 2 回目のドラッグ時に得たマウス座標からある点まで直線を描く. このように, 本プロトタイプでは動的情報としてマウス座標を容易に扱うことができ, 直感的に理解し易いコマンドを用意している. さらに, ベクター形式の出力を得ることができるため, プログラムにより表示した図形を Illustrator 上で手作業による加工が容易である.

## 5. まとめ

本研究では, グラフィックデザイナーのための視覚化プログラミング環境に関する基礎検討として, マウスによる動的情報の視覚化を目的としたプログラミング環境のプロトタイプを構築した. 今後は, 動的情報の入力として音声, 動画に対応するシステムを構築する予定である.