ビットと身体をつなぐクリエイションツール研究

櫻井 稔

Creation Tool Research for Connect Between Bit and Body

Minoru Sakurai

近年、本やポスター、CM など様々なコンテンツでコンピュータグラフィックスを用いることが一般的となり、デザインの制作現場ではコンピューターが無くてはならない存在となった。コンピューターを使用した制作では、コンテンツの種類により各々に適した編集のためのアプリケーション・ソフトウェアが存在し、クリエイターにとっての “道具” としての認識が定着している。しかし、コンピューター上の道具であるソフトウェアは使用方法や表現手法が限定されているものが多く、デザイナーの発想や表現の限界が道具によって規定されてしまう可能性がある。2008 年以降はスマートフォンやタブレットなどの小型のコンピューター端末と、高速なインターネット環境の普及により、コンテンツが直接消費者の持つデバイスに配信されることが増えた。これにより、コンピューター環境は制作のための道具としてだけではなく、消費のためのメディアとしても認識されるようになった。現在、デザイナーの関わるコンテンツの幅は画像や映像だけではなく、ツールやサービスなど様々な分野に広がり、求められる発想の幅も意匠だけではなく、機能や概念まで多岐にわたっている。本研究では、表現のための道具となる 3 次元ペイントソフトウェア “AirStroke” をゼロから制作し、出来上がったソフトウェアを用いて絵画表現を行った。本稿は、ソフトウェアの開発記録を元に、コンピューターを活用するデザインにおいて重要となる事項の言語化を試み、結果として “ゼロベースでの制作”、“素材からの発想”、“道具の身体化” という 3 つの重要な概念を抽出するに至った。

1. はじめに

1946 年、現在のコンピューターの原型となるノイマン型コンピューター “ENIAC” が Mauchly と Eckert によって生み出された。当時の民生用コンピューターは CUI[1] が主流であったが、その後 GUI[2] が普及し、1980 年には現在の GUI を搭載したパーソナルコンピューターの原型である “smalltakl-72” が作られた。1984 年に Apple Computer から GUI を基本とした OS(Operating System)、Macintosh 128K が発売されると、デザインの制作現場には DTP[3] が急速に普及した。Macintosh 128K で採用されたディスプレイの解像 (512x342 pixel) は、後の DTP の規格である 72PPI[4] の元となっている。現在では、CM や映画などの映像編集はもちろんのこと、本やポスターなど、ほぼ全てのコンテンツはその製作過程でコンピューターが使用され、デザイナーにとってコンピューターの活用は避ける事が出来ない道となった。

2007 年時点において、日本におけるコンピューターの普及率は 80%(85%) を超え、家にコンピューターがある生活が一般的となった。同時にコンピューター同士を接続するインターネットの普及率も 70%(73%) を超えている [5]。2007 年 6 月に Apple Computer が iPhone を発売すると、据置型のコンピューターを持たない人であってもいつでも気軽にデジタルコンテンツを楽しめるようになり、その結果コンピューターはコンテンツを制作するための道具としてだけではなく、インターネットを介して

[1] Character User Interface
[2] Graphical User Interface
[3] Desk Top Publishing
[4] Pixel Per Inch

[5] 総務省統計局調査

コンテンツを発信／受信することのできるメディアとしても広く認識されるようになった。この状況をReas はその著書『FORM+CODE』[1] において “コンピューターが「ツール」ではなく、「メディア」になる” と表現している。

iPhone を皮切りに、インターネットに接続された小型のコンピューター端末が “スマートフォン” として普及すると、消費されるコンテンツはこれまでの静止画や動画に加え、メモや音楽再生、ミニゲームなどの小規模なソフトウェアがネットワーク経由で配信されるようになった。それらは “App” と呼ばれ、スマートフォンの持つカメラや GPS、加速度センサといった様々な機能が使えることからコンテンツの自由度が大きく広がった。様々なセンサーを持った小型コンピューターのコンテンツは、持ちうる可能性はこれまでのものとは大きく異なる。例えば何かの実行をキャンセルするための UI[6] をデザインをする場合においても、意匠的にキャンセルボタンを配置するのではなく、端末を振るという動作をキャンセルボタンの代わりにすることが可能だ。このように、メディアの自由度が増す中で、開発に関わるデザイナーは意匠だけではなく、機能までも包括して提案ができるようになり、より幅広い発想をすることが求められている。

企業では、コンピューターというメディアの持つ可能性をより引き出すために、エンジニアとデザイナーの混成チームを作り、ディスカッションの回数を増やすなどの対策をしている。しかし、デザイナーがエンジニアリングにおける限界を理解出来ないために無理な提案を強いていたり、エンジニアが意図を汲みきれずに当初デザイナーが想定していたものと大きくかけ離れたものが生まれてしまうといった状況をよく目にする。デザイナーが機能を包括してアイデアを生み出そうとする場合、デザイナーはエンジニアリングの特性をよく理解する必要がある。これは実空間に粘土などの素材を使い、モノを作る時生まれる問題によく似ている。粘土という素材の特性を知らずに設計をすると、重力に耐えられないほど細い足を作り潰れてしまったり、量のバランスを間違えて倒れてしまったりする。ソフトウェアはその裏側で大量の 0 と 1（以下、ビット）を条件によって書き換えながら動いている。これらをエンジニアリングにおける素材と考えた場合、その特性を知らずに設計をすると、重すぎて動かなかったり、逆にキャパシティの大半を使わずに終わってしまうこととなる。物理的にモノを作る場合、デザイナーは自身の手で直接素材に触れ、特性を肌で感じながらアイデアを練る。コンピューター上でデザインをする時も同様に、デザイナーはプログラムコードに触れ、素材であるビットの特性を感じるべきである。

紙や粘土に限らず、表現活動を行う際には様々な種類の素材を扱うが、自由に加工するためにはそれぞれに適した道具が必要となる。さらに、種々の道具を通して素材の特性を深く理解するためには、道具を身体の一部のように捉え、扱えることが望ましい。このような状況を堀内らは “道具が身体化している” と表現している [2]。現在、コンピューター上での道具として数多くのソフトウェアが販売／配布されている。例えば絵を描くような用途であれば Adobe 社の Photoshop、3 次元の映像を作るのであれば Autodesk 社の Maya などが挙げられる。これらのソフトウェアは目的とする用途の範囲内において非常に柔軟な発想を行うための手助けをしてくれる。しかし、その多くが特定の用途に特化して作られており、定められた範囲をこえて表現を行うことは難しい。そのため、デザイナーは道具がもつ表現力の限界に囚われ、アイデアの発想が限られた範囲内で閉ざされてしまう可能性がある。

前述のようにコンピューターを活用したデザインには既存の概念にとらわれない幅広い発想が求められている。そこで、本研究では表現のための道具と

[6] User Interface

なる 3 次元ペイントソフトウェア “AirStroke” の開発からはじめ、完成したソフトウェアを用いて描画表現を行う事にした。デザイナーである著者自らプログラミング技術を深く習得し、コンピューターの素材となるビットを自由に扱うことで、試行錯誤を繰り返す中から発想を行うことの重要性を示し、言語化することを試みる。

2. 基本思想

本章では、著者が研究を進める上での背景となった基本的な思想についてまとめる。

2.1 デザインとエンジニアリング

昨今のデザイン制作において、デザイナーはコンピューターが関わる製品に携わる機会が増え、ビジュアルだけではなく機能をも視野にいれたモノづくりが求められている。デザインにはグラフィックやプロダクト、テキスタイル等様々な専門領域が存在し、エンジニアリングにおいても同様に非常に細かく領域がわかれている。各領域では各々の専門性を高めることにより、優れたビジュアルやシステムが生み出されてきた。しかし、プロダクトにコンピューターが埋め込まれる現代、形状による身体的な使いやすさだけではなく、ユーザーインタフェース等による視覚的な手びきや、アニメーションや状態遷移などの動きによる誘導が必要となり、各専門分野を横断した視点を持ちながら最終的なプロダクトを生み出す能力が不可欠になりつつある。多くの企業がこの問題に対し、各専門分野の人間がより密接に関わりながら制作をすすめる方法を模索してきた。IDEO[7] の CEO の Tim Brown は，デザインのプロセスを用い、クリエイティブなアプローチを活用して様々な業種の人間を巻き込みながらより革新的なプロダクトを生み出す手法を “デザイン思考” として提唱している [3]。しかしながら、未だに多くの会社ではデザインとエンジニアリングを部署などにより分割し、それぞれの得意分野の範疇でのみ活動を許される状況が見受けられる。今後、コンピューターを媒体としたデザインはハードウェアやサービスの仕組みそのものにまでその活動範囲を広げる。そのためにも、デザイナーは意匠だけではなく、いかにしてエンジニアリングに関わるかを意識する必要があるだろう。

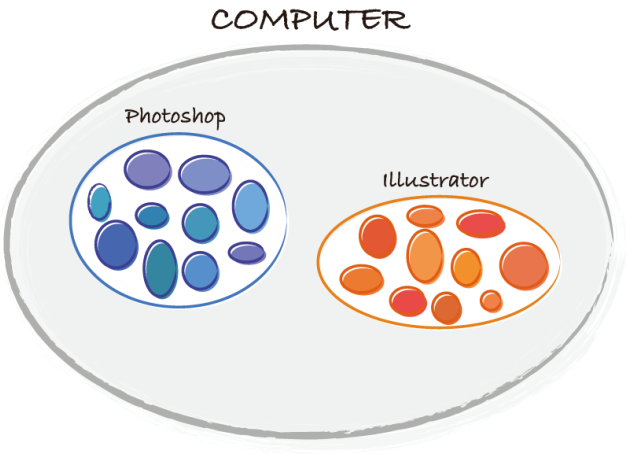


図 1 コンピューター上での表現活動の領域

コンピューターを媒体とした制作をするためには必ずプログラミングの能力が必要になるわけではない。例えば絵を描くには Photoshop、3D モデリングをするには Maya[8]、作曲ならば Cubase[9] といった具合に、世の中にはコンピューターで使うことのできる様々な編集用ソフトウェアが出回っている。デザイナーは頭の中にあるイメージを具現化するのに適したソフトウェアを選択し、使用方法を習得することで目的の表現をコンピューター上で行うことができる。しかし、全ての表現が可能なソフトウェアは存在せず、それぞれには規定された表現の幅が存在する。例えば Photoshop であれば色のついたピクセルを配置することが、Maya なら 3 次元画像をあたかも本物のようにリアルに表現することが可能だ。そのため、図 **1** のように、表現者はそれぞれのソフトウェアの持つ表現空間の中で表現活動を行い、その枠から外れることは極めて難しい。ソフトウェア

[7] http://www.ideo.com/

[8] http://www.autodesk.co.jp/products/autodesk-maya/
[9] http://japan.steinberg.net/jp/products/cubase/

の限界を既存のソフトウェアの組み合わせにより規定しているデザイナーは、新しく考えだす提案も既存の技術で実現可能な範囲からのみ発想せざるをえない。

コンピューターの世界では、ゲームのように現実世界には存在しないインタラクションを生み出すことができる。例えばソフトウェアの動きそのものを自由にデザインしようと考えた場合、図 **1** 上の Computer という大きな円で囲われた部分に創作をする必要がある。この領域では 0 と 1 で画像や音声の表現や、インタラクティブな動作の定義などをすることが可能だ。例えるのであれば自由に造形できる粒子が敷き詰められた砂場のようなものだ。コンピューターやモバイルデバイスなどによってこの領域の広さはまちまちではあるが、全てのコンピューティング環境でこの 0 と 1 を使用した表現活動が行われており、この領域で自由にデザインを行うことが今後の課題と考えられる。

2.2 素材としてのビット

コンピューターの利用において、何を素材と捉えるのかはその目的によって大きく異なる。

例えばゲームのプログラムを構築する場合などはビット [10] を素材として扱い、様々な状況を四則演算 [11] と条件分岐によって決定する。ビットを扱うための道具としてプログラミング言語が挙げられるが、その種類には C 言語や Java,Ruby など様々な言語がある。これらは同じ計算をする場合も記述方法が異なったり、得意とする計算の種類に違いがあったりする。これに対し、デザイナーがコンピューター上で画像を扱う場合、最小単位であるピクセル (Pixel) [12] が素材となる。ディスプレイはこのピクセルを X 軸（横）と Y 軸（縦）に敷き詰め、画像や映像

図 2 ビットによるカラー表現

を画面上に表示する。コンピューターを使って絵を書いたり、写真を加工したりする場合、このピクセルを素材として扱う。これを扱うための道具として Photoshop や Painter[13], GIMP[14] など様々なペイントソフトウェアが存在するが、それぞれにおいてピクセルをどのような質感で並べることができるかに違いがある。他にも、3DCG を扱う場合は頂点 (Vertex)[15] や面（Surface）、質感 (Texture) といったものが素材となり、製作者はこれらを使用して仮想空間に組み上げて 3 次元オブジェクトを制作する。この場合、Maya や XSI、3dsMax などが扱うための道具となり、様々な質感を生み出す。

しかし、Pixel も Vertex も、コンピューター上で扱われる全てのデータは 0 と 1 というビットを原材料とした素材である。例えば図 **2** は写真がピクセルから構成され、そのピクセルのカラーがビットで表現されていることを示している。ピクセルは左上から順に一本の糸を折り返すように敷き詰められ、0 と 1 の羅列で表現される。図 **2** の場合、画像のサイズが 400x75 ピクセルなので合計 30,000 ピクセルが並んでいることになる。ピクセルの色情報は光の三原色である赤 (R) 緑 (B) 青 (G) の混合によって表現される。16 進数では 00〜FF、10 進数では 0〜255、2 進数（ビット）では 00000000〜11111111 で指定

[10] デジタルコンピューターの扱う最小単位で、binary digit を略したもの。2 進数一桁である 0 と 1 で表現され、先も述べた通り現在のコンピューターはこの 2 つの信号によって音声や画像等を表現している
[11] 加算 (+) 減算 (−) 乗算 (×) 除算 (÷) を指す
[12] 一般的にドットとも呼ばれるが、ピクセルが色情報の最小単位であることに対し、ドットは物理世界での表示の最小単位を指している。

[13] http://corel.e-frontier.co.jp/products/illustrator/painting/
[14] http://www.gimp.org/
[15] 3D 空間上の x,y,z の値を持った点情報

```
int i;
for(i=0; i<100; i++){
  printf("%d\n",i);  //この行が100回繰り返される
}
```

図 3 C++言語における for 文による繰り返し処理

する。この例の場合、16 進数では FD,43,32、10 進数では 253,067,005 が色の情報となる。ホームページを記述する HTML 言語では背景や文字の色の指定にこの 16 進数の表現が使われている。各ピクセルが 8 ビット 3 原色の 24 ビットで構成されているため、図 **2** の写真ではには 30,000 ピクセル 24 ビットの合計で 720,000 個の 0 と 1 が並んでいることとなる。ここではコンピューター上での画像の扱いに触れたが、映像や音声、プログラムについてもこれと同様にビットで構成されており、その並びによって様々な表現がなされている。

素材を何と捉えるかは目的によって異なる、しかし 2.1 節で述べた通り、コンピューター上で道具にとらわれずに自由な発想をするためには可能な限り原材料に近いビットを素材と考えて発想をし、既存の道具を有効に活用できる場合は適切な道具を用いて加工をするべきである。

コンピューター上でビットという原材料を素材として扱う場合、その特性には向き不向きが明確に存在する。現実世界の素材を考えた時、繊維や紙、粘土など、それぞれが線材、面材、量材といった特性を持ち、素材に適した方法で使用することで、それぞれの持つ強度や美しさを引き出すことができる。線材を並べて面材として使用したり、束ねて量材として扱うことも可能だが、素材の特性から逸れた使用方法は物理的な制約からの乖離により脆性を生むことになるり、これはビットにおいても同様である。

例えば得意な処理として “繰り返し” が挙げられる。これは for 文 (図 **3**) と呼ばれる反復の処理がほぼすべての言語において基本概念として用意されていることからも分かる。例えば 100 人の身長の平均を求めたい場合、全ての処理を記述すると足し算を 100 行記述した後、割り算を記述する必要がある。しかし、繰り返しの記法を用いることで、100 行の記述を 1 行で行うことが可能となる。これは単純に記述労力の削減だけではなく、同じ処理を複数の要素に対して繰り返し行うという概念として、ソフトウェア開発に広く取り入れられている。対して、美術的表現など、環境により常に値の変化するような記述は不得意な処理として挙げられる。これは先に挙げた “同じ処理 ’ ’ を繰り返すものではないためだ。

ビットを素材と考えて使用するためには、扱うための道具となるプログラミングを習得する必要がある。プログラミング言語の習得には時間がかかり、デザイナーが自由に扱えるようになるためには相応の努力が必要となる。しかしながら、デザイナー自らが言語を学び、自身の手でプログラミングを行うことは非常に重要な条件といえる。

アイデアを練る段階から素材に触れ、試行錯誤を繰り返すことで洗練されたアイデアが生まれると言われている。奥出直人は、プロトタイプの制作とその検証を繰り返す**プロトタイプ思考** (build to think) を提唱している [4]。素早く大量のプロトタイプをつくり上げる手法をダーティプロトタイプやラピッドプロトタイプと呼び、最低限の機能をもった動くモデルを使い、そこから見つけた改良のアイデアをフィードバックしながら、機能追加や機能拡張を行う。これはつまり、試行錯誤を繰り返す制作手法である。この手法はデザインの現場ではごく当たり前に行なわれていることだ。鉛筆や絵の具、粘土など、どのような素材を扱う場合でも自らの手で素材にふれながら試作品をつくり、素材の持つ魅力を最大限に引き出すために設計から修正しながら制作をおこなう。プロトタイプを自身の手で触れて確認することは、新しい楽器やゴルフクラブを手にとって感触を確かめる行為に近い。アイデアの結果を実際に手に持ち、自身の身体の延長に配置できたとき、初期の段階では思いつかないアイデアがてから生まれる。奥出は

図 4 タビュレーティングマシン Hollerith Census Tabulator[5]

これを“エンボディメント (embodiment)”と呼んでいる。デザイナーがプログラミングを学ぶことには多大な努力が必要となるが、自身の手で素材に触れながら考えるという所作をコンピューター上で行うためには避けて通ることの出来ない道といえる。

2.3 道具の制作

ビットという素材を扱うためにはプログラミング言語という道具が必要となる。しかし、原材料に近い存在であるビットを素材として扱う場合、感性を元にした変則的な表現をすることが難しい。そのため、既存の編集ソフトウェアのように、直感的に扱える独自の道具を作り、ビットから新たな素材を生み出さなければならない。現実世界で感性を元に表現をする際、表現者の身体は常に周囲の環境の影響をうけ、その時々に肉体を通してイメージを出力する。アイデアの結果をつねに具現化し、自身の躯体の延長に配置することで、頭だけではなく身体でも思考する。コンピューター上で表現を行うためには、プログラミング言語で数値を扱うだけではなく、より感覚的に表現ができる道具が必要となる。

ビットをより感覚的に扱う試みはコンピューターが生まれた時代から常に試みられてきた。当初コンピューターは 0 と 1 をパンチカードで並べて計算をさせ、結果を導き出していた。これは“タビュレーティングマシン”(Tabulating machine)(図 4) と呼ばれ、アメリカでは最初 1890 年の国勢調査に用いられた。日本では“パンチカードシステム”と呼ばれていたこのシステムは、パンチカード (図 5) と呼ばれる紙に穴を開けてコンピューターに通すことで、穴が開いた部分のみ通電してデータの入出力をすることができる。

図 5 パンチカード[5]

```
MOV  EAX,  3
MOV  EBX,  5
ADD  EAX,  EBX
```

図 6 アセンブリ言語で 3 と 5 を加算する例

しかし、この方法は数値の羅列によるものであったため、1950 年代にはより人間に理解しやすい記述方法として“ニーモニック”と呼ばれる命令でプログラムを記述するアセンブリ言語が生まれた (図 6)。その後、ゲームなどで複雑な表現をするために、さらに人間に理解しやすい記述言語として C 言語[16] が開発され、ファミコンなどの家庭用のゲームの開発に使われ、現在も多くのソフトウェアにおいてこの C 言語が活用されている。当時、グラフィックを画面に表示するために Photoshop のようなペイントソフトウェアを使わずに、0 と 1 を画面上に並べることで画像を表現していた。図 7 は画像を数値によって表現しており、当時のプログラムではソースコード内に画像データとしてこのような数字が羅列されて

[16] 1972 年にアメリカ AT&T 社のベル研究所で D. M. Ritchie 氏と B. W. Kernighan 氏によって開発されたプログラミング言語。1986 年にアメリカ規格協会 ANSI によって標準化され、現在最も普及しているプログラミング言語である。

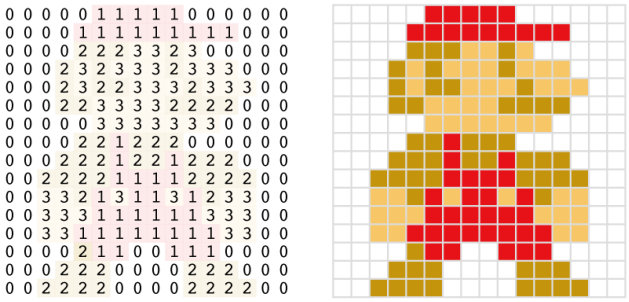


図 7 文字列による画像の表現例

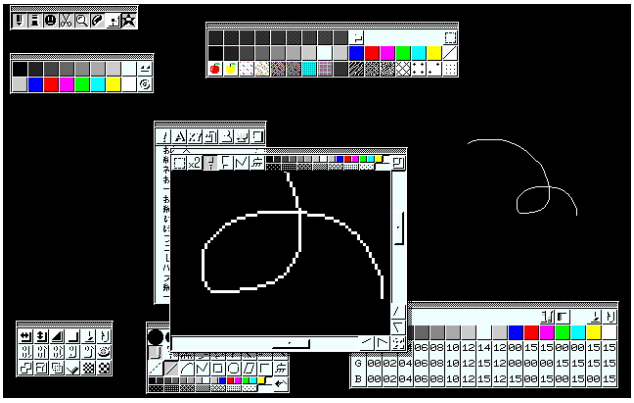


図 8 マルチペイント

いた。上記の例でも分かる通り、当時のコンピューターで行われる表現活動において、プログラムとコンテンツの明確な差は存在せず、全てを粘土だけで造形するように 0 と 1 を並べることによって表現を行なっていた。

その後、パーソナルコンピューターの普及やコンピューターゲームの進化により、一度に処理できるデータ量が爆発的に増え、表現の幅も大きく広がった。それに伴い、編集用ソフトウェアが開発され、プログラムとそこで扱われる画像などの要素は区別して制作されるようになった。コンピューターの制約の変化により、当初 8 色、16 色、256 色、と進化とともに増えてきた色数も、人間の目で判断できる限界であるフルカラーが 32 ビットで表現されるようになり、それを扱うソフトウェアもより感覚的に扱えるように進化を続けた。

当初は図 **7** のように数字で形を並べていたが、その後 16 色や 256 色といった色数が使えるようにな

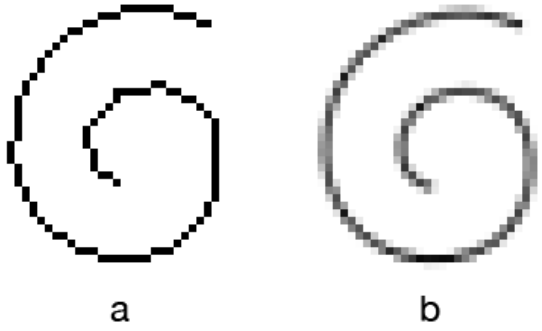


図 9 a:ジャギーの残る線 b:アンチエイリアス処理をした線

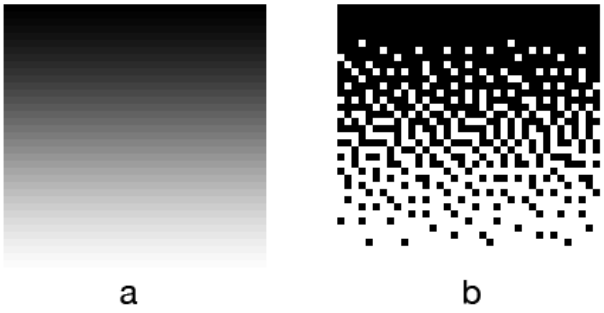


図 10 a:グラデーション b:ディザ処理によるグラデーション

ると、常に編集をした結果を見ながら描画のできるペイントツールが使われるようになった。例えばマルチペイントと呼ばれるソフトウェア (図 **8**) は 1980 年代後半から 1990 年代前半にかけて、16 色ゲームのグラフィック制作に幅広く活用された。その描画方法は現在の Photoshop のようなものではなく、パレットに作った 16 色の色を画面上にピクセルとして並べていくような作業が必要だった。

この作業で特徴的な工程に “ジャギ取り” と呼ばれるものがある。現在出回っているほぼ全てのペイントソフトウェアではドットを並べた際にうまれる “ジャギー” と呼ばれるギザギザを “アンチエイリアシング” という処理を加える事で自動的になめらかに補正をかけている (図 **8**)。しかし、16 色で表現をしている時代、色数や処理速度が限られていることからアンチエイリアスの自動処理は一般的ではなく、ギザギザの隙間に自身で作った中間の色を配置することで、ジャギーを軽減していた。

また、色数の上限が 16 色であることは、滑らかな

a b

図 11 a:フルカラー画像 b:ディザ処理により減色された画像

色変化によるグラデーション表現を難しい物にしていた。そのため、グラデーションの表現は色の変化によるものではなく、ディザ処理と呼ばれる図 10b のようなハーフトーンパターンを敷き詰める方法で表現をすることが一般的であった。同様に、エアブラシのような描画跡がグラデーション表現になるものも、このディザ処理によってその多くが表現された。その後、家庭用パーソナルコンピューターが 256 色まで表示が可能になると、フルカラーで表現ができる業務用のマシンで描画をし、後から 256 色への減色処理を施す手法がとられた。この原色処理は、アルゴリズムによって結果が大きく異なったため、より再現度の高いソフトウェアを皆が求めていた。その中でも OPTPiX [17] は 256 色への変換後も遠目ではフルカラーのものと見分けがつかないほどの再現度を持っていたため、当時さまざまなゲームメーカーが OPTPiX を活用していた (図 11)。

このように、技術の変化は描画用ソフトウェアやその使用方法にまで影響を与え、表現者はその時々のキャンバスに最適な手法を模索し、エンボディメントを獲得することで自身の身体の一部としてそれらを扱ってきた。

プログラミングの手法においても、スケッチをするようにより感覚的に表現をするために様々な挑戦がなされてきた。Casey Reas と Benjamin Fly はグラフィック表現に特化したプログラミング環境 “Processing”[18] を開発した (図 12)。これは、それまでのプログラミング環境で画面上に一本の線を表現するためにまるで呪文のような命令文を羅列しなければならなかったのに対し、line という命令を一行記述するだけで画面上に線を描画できる、容易に映像を扱うことのできる環境であった。また、教育用のプログラミング環境として JAVA と呼ばれる広く使われている言語をベースとして作られているため、より複雑なソフトウェア開発をしたい場合、シームレスに次の段階の勉強ができるようになっていた。このスケッチをするようにコーディングをするという思想は、Processing の保存用デフォルトファイル名が Sketch であることからも伺える。

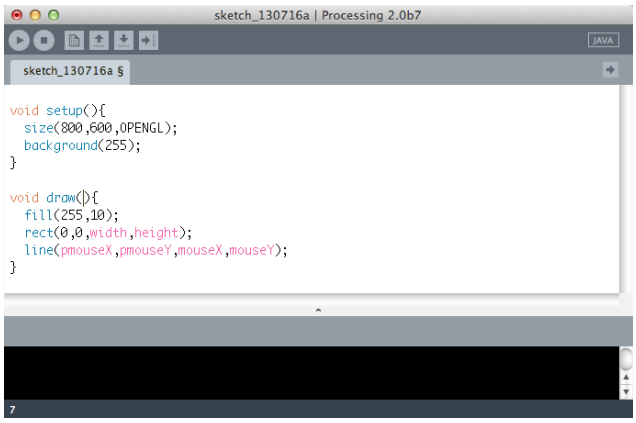


図 12 Processing

図 13 MAX/MSP によるビジュアルプログラミング

[17] http://www.webtech.co.jp/optpix/
[18] http://www.processing.org/

ビジュアルプログラミング環境と呼ばれる方法では、計算のためのユニットを二次元平面上に配置し、それらを線で繋ぐことで処理を行う。例えばMAX/MSPという音楽編集ソフトウェアでは、MIDIと呼ばれる信号でデジタル楽器から取り込まれた音の信号を数値として扱い、ミキシングやパンの変化、スピーカーへの出力などを線でつなぐことで自在に扱うことができる(図**13**)。また、Apple社のQuartz-ComposerやDerivative社のTouchDesigner等は、インタラクティブ性を持ったヴィジュアルをコーディング以外の方法で構築する道具として世界中の多くの場所で活用されている。特にTouchDesignerは、動画のエフェクトや3Dグラフィックスとの連携等に特化しており、映像を常に再生しながら編集するVJ(VideoJockey)[19]の用途に多く使われている。これは、ヴィジュアルプログラミング環境が、楽器を身体の一部として演奏するように、感覚に合わせたリアルタイムな編集ができるレベルでエンボディメントを獲得しているものと言える。

このように、コンピューターはハードウェアの進歩とともに、その使用目的が計算だけではなくなり、インタラクティブな動作や美術表現にまで至ったことで、表現活動のための道具としても認識されはじめている。

表現をするためのソフトウェアを道具と考えるのであれば、プログラミング言語はその道具を作るための道具と考えることができる。道具を作るための道具は“二次的道具”、もしくは“二次道具”と呼ばれている。これは文化人類学において、かつて道具の利用が人間に固有の特徴と考えられていたものが、鳥などが枝を使い巣を作る等して道具を扱うことから、二次的道具を使うことを人間と物との関係と定義付けたことに由来する。例えばPhotoshopのような編集用アプリケーションを道具と定義した場合、それを作るために使用されたプログラミング言語は道具を作るための道具であることから二次的道具として定義することができる。

デザインをする時、最初必ずエスキース帳などに簡単にスケッチをする。これは頭の中にあるイメージを可能な限り素早く目の前に実現させ、イメージを確認したり共有したりするためだ。また、素材の方からより適切な表現を導き出すために、紙や粘土を使って面や量を現実世界に配置してみる作業も、また一種のスケッチといえるだろう。しかしながら、インタラクティブに動くイメージを紙や粘土を使って確認や共有をするのは非常に難しい。そのため、紙や粘土でスケッチをするように、全体の動きを素早くプログラミングできる事はイメージをふくらませるためにも非常に重要な工程となる。

ペイントソフトウェアのような一次的道具がビットを素材として整形することを目的としていると考えるのであれば、デザイナーはより感覚的に素材を扱うために、独自の道具を用意する必要があるだろう。コンピューターを用いた表現には、プログラミングを二次的道具として独自の一次的道具を作り、ビットという素材を自在に扱うことが要求されている。

3. 研究の流れ

本章では、本研究に至るまでの間に開発した代表的なソフトウェアを挙げ、本研究に至るまでの流れをまとめる。

3.1 Sequential Graphics

2007年、描画時の臨場感を再現するペイントソフ“SequentialGraphics”(図**14**)を開発した。このソフトウェアは、常にループする時間軸を持つキャンバスに線を描画することで、描画時のタッチの速さや強さを記録し、再現することができる。表現された絵は、画家の手の動きを感じさせるものになり、描画時の臨場感を伝えることができる。これにより、静止画と動画の特徴を兼ね備えた「動く静止画」の実現を目指した。

[19]画像や映像を素材として、DJのように音楽に合わせてリアルタイムに映像を編集する。VisualJockeyとも呼ばれる。

図 14 SequentialGraphics での描画例

従来のペイントソフトは、主として現実世界のメディアである紙や、絵の具の隆起、水分量低下による擦れなどの質感をシミュレートし、再現しようとしている。技術の進歩に依る再現力は飛躍的に向上したが、物理世界の再現を目指している以上、物理世界のメディアの表現力を超えることはない。コンピューターにはコンピューターに適した表現手法が存在すると考える。

ペイントソフトウェアには新しい表現手法を求めた様々なアプローチが存在する。Ryokai らによるによる I/O Brush[6) 7)] では、カメラを備えた筆型デバイスからの入力画像をテクスチャとして使うことによって、カメラとペイントソフ トの融合を実現した。Cassinelli らによる Khronos Projector [8)] では、時空間を自分の手で操るデバイスを実現した、これらの研究はコンピューターによる新しい表現手法を開拓したものとして評価できるが、従来の絵画技法に精通したプロの画家が新しい表現を発展させる土台として考えられるようなものではない。それは、画家が絵を描く際の心理的作用に着目していなかったからではないかと考える。また、Maeda による TimePaint[9)] は、時間軸を持つペ イントソフトであり、描画後にフレームを掴んで動かすことでフレーム間の関係性を奥行きを持って見 せることができる。同様のソフトウェアとして、時間軸を持つ線をキャンバスに描画できる Levin による Yellowtail [10)] や、Gysin による Chronodraw[11)] があるが、どちらもペイントソフトの形式を借りたソフトウェア・アート作品であり、コンピューター上の動的な表現を模索した先駆的な作品である。それに対して本ソフトウェアは、画家が表現活動の基盤として使える新しいメディアを目指したものであり、その点が大きく異なる。SequentialGraphics では、これらの前例に対し、線を書く際の手の動きの情報を保存することによって描画時の臨場感を再現することを目指した。

図 15 時間経過によるストロークの変化

キャンバスは約 1.5 秒の動画のような時間軸を保持し、常にループして再生されている。図 **15** に一本の線が描かれる様子を示す。左下の視点から右上の終点へと、およそ 0.5 秒間手が動き、f で線が完成している。書き始めは弱く、途中から強く筆圧も変化している。ループされている時間軸が一周して戻ると、キャンバスはまた a から f までを再生しなおす。この時間変化を伴う線を積層することによって、描画時の勢いをそのままキャンバスに記録することを試みた。

SequentialGraphics は独立行政法人情報処理推進機構 (IPA) の未踏ソフトウェア創造事業の支援のもと 2007 年に完成し (図 **16**)、2008 年にはワールド・ビジネス・サテライトのトレンドたまごで紹介された。

3.2 Material Reader

2010 年、一冊の電子書籍”地球マテリアルブック”の開発を通じて、その電子書籍プラットフォーム”MaterialReader PublishingPlatform ”（電子書籍リーダである”MaterialReader”とその記述フォーマットであ

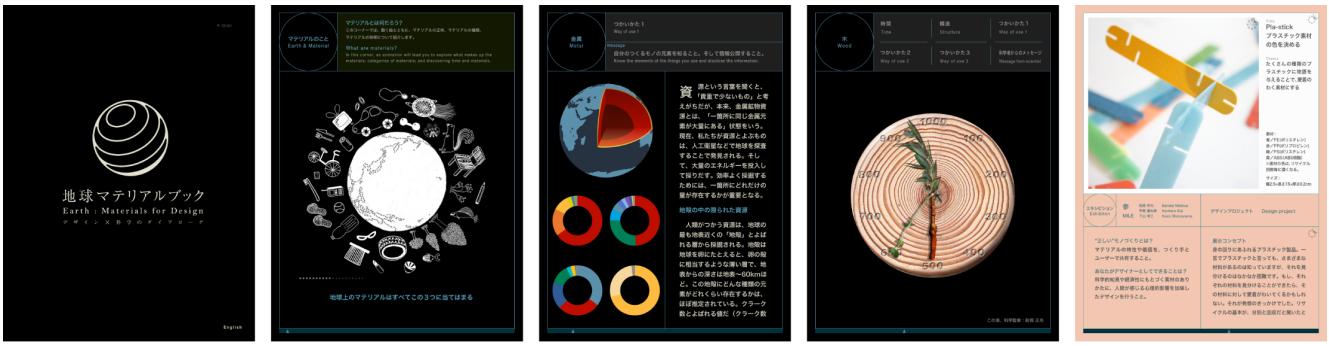


図 17 地球マテリアルブック

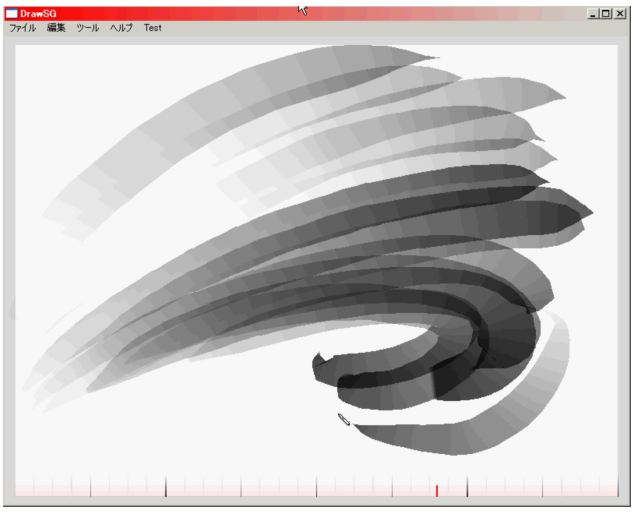


図 16 完成したソフトウェア

る”MOML”の組）の開発を行った。デザイナーの考える理想の電子書籍を実現するために、ラピッドプロトタイピングの思想を取り入れ、プロジェクト当初から多くのプロトタイプを開発し、そこから得られた知見をシステムへとフィードバックした。MaterialReader プロジェクトでは、その開発課程からデザイナーがいかにシステムを Material（素材）として扱うことが重要であるかをその開発課程から導き出すことができた。

一般的にデザイナーとエンジニアの協業には、業務分担の明快化という問題がある。業務の発注内容を明解にするために、仕様書によって言葉で機能を指定する。仮にデザイナーが何らかの機能を思いついたとしても、通常のプロセスでは、それを一旦言葉で表現し、仕様書にしてからエンジニアに伝える必要がある。この方法では、うまく言語化できない機能や実物をみることではじめてわかる機能を反映させることはできない。プロトタイプ思考では、エンジニアは素早く作れるプロトタイプを繰り返し制作し、デザイナーは実物を見ながらアイデアを膨らませ、両者は実物を元にコミュニケーションし、仕様を固める。これにより、実物を見た時の感覚を反映させた製品を実現することができる。

2010 年、日本科学未来館は「デザイン科学　地球マテリアル会議」[20] という展覧会を開催した。この展覧会の終了後、成果をまとめるカタログとして「地球マテリアルブック」を制作する事となった (図 **17**)。当時はちょうど電子書籍が普及をはじめた時期であり、カタログを電子書籍で作ることが適切と考えられた。この地球マテリアルブックという電子書籍コンテンツの開発と同時並行で、その電子書籍のリーダーやフォーマットを実装した。デザイナーとエンジニアが共同で開発を行う場合、デザイナーの要望を実現するためにエンジニアに負荷がかかることが多い。しかし作業内容の適切なトレードオフを行うことで、デザイナーの要求を最大限に受け入れながらも、エンジニアの負荷を一定程度に抑えることができる。本システムでは、コンテンツの記述に XML を採用し、デザイナーに直接書いてもらうこととした。エンジニアには記述言語を実装する手間が発生し、デザイナーには XML を書く手間が発

[20] http://www.miraikan.jst.go.jp/spevent/earthloungevol7/

生するが、結果としてデザイナー側にコンテンツ記述における試行錯誤を任せることができるようになる。結果として高速なプロトタイピング環境が生まれ、双方の手間を抑えつつ質を上げる環境が構築された。

デザイナーにとって、デザインのための道具は重要な意味を持つ。デザイナーは元来、道具を使うだけではなく、道具を調整したり、新たな道具をつくり上げることを通して、新しい表現力を獲得してきた。現行の与えられた環境でデザインする限り、デザイナーはその決められた領域においてしか創造性を発揮することができない。MaterialReaderプロジェクトでは、電子書籍のフォーマットやリーダーをゼロからつくりなおすことにした。それにより、デザイナーにとって重要な、新しい道具を作ったり、調整したりすることによって新しい表現力を獲得することを目標とした。

MaterialReaderは最終的にスクリプトを読み込ませることで書籍として表示するプラットフォームを構築したが、初期段階ではプログラムに直接コンテンツを書き込むかたちでの開発をしていた。しかし、コンテンツと同時に開発をしていることもあり、コンテンツの内容が変化するたびにプログラムの構造から書き換えるといった効率の悪いプロトタイピングを繰り返していた。これは目的の造形物を作るために、常に粘土ですべてを作り直している状況に近く、目的の形状が変わるたびに何度も形を崩して作り直しているような状況だ。しかしながら、ほぼすべてのアイデアは、画像を配置したり、動画を配置したり、と基本となるルールを共有しているため、それらの共通する部分のみを先に作っておき、違う部分のみ書き換えていけるのではないかと考えた。そこで、プロトタイピングをしながらも、基本となる構造だけは静的なものとするため、書籍を記述するための言語MOML(Material Object Markup Language)の設計を行うことになった。この言語はXML をベースにしており、ホームページを作成するためのHTML と同じような記述により電子書籍の制作を可能とした。

図 18 東京 23 区の人口密度をグラフ化したもの

本研究では、デザイナーとエンジニアが共同作業を行うための手法について大きく取り上げているため、コラボレーション手法に関する資料 を主に参照した。2.1 の章でも触れているが、IDEO の提唱するデザイン思考や、Kelly による『発想する会社!』[12]、棚橋による『デザイン思考の仕事術』[13] では作業プロセスが実例と共に具体的に述べられいることから、プロジェクトの進行をする際に取り入れた。インタフェースの実装には、Normanによる『The Psychology of Everyday Things』[14] で説かれ ているフィードバックに関する記述や、AR 技術を電子書籍に取り入れたDoらによるARBookCreator[15] 等を参考にしている。地球マテリアルブックは2011年に完成し、日本科学未来館からAppStoreを通して無料で配信された。その後、MaterialReaderは電子書籍プラットフォームとして“5by8”や“日本ワインガイド”といった様々な書籍のためのプラットフォームとして活用されている。

3.3 draffic Visualizer

2012年、株式会社電通とゼンリンデータコムとともにNTT ドコモ社のオートGPS を通じて取得した位置情報を可視化するdraffic Visualizerの開発を行った。開発開始当時、可視化は主に記録された位置を点として描画したり、履歴を線で繋ぐなどの手

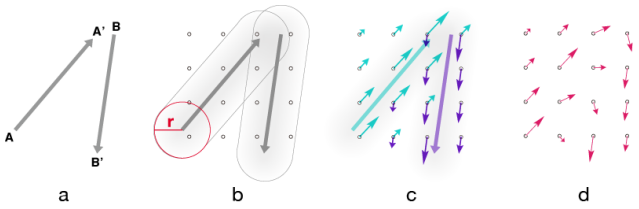


図 19 ベクトル場による動線の記録

図 20 移動ベクトルがベクトル場に与える影響

図 21 ベクトル場を元にしたパーティクル表現

法がとられていた。しかしながら、時間情報を保持した各経路の位置情報は個人情報にあたり、可視化は個人情報保護法の観点から問題が生じるため、ビジュアルを一般に公開することが出来ずに居た。そのため、公開資料は各駅の利用比率や、エリアに対して人が出入りした人数などの集計情報を、グラフなどにまとめたものがそのほとんどであった。そのため、開発は位置情報を個人情報を保持しないかたちでビジュアライズする手法の模索を目的に行なわれ、最終的に受け取った緯度・経度の情報を匿名化しながら可視化するソフトウェアを構築した。

図 18 は制作したビジュアライザを使い、2013年 2 月 11 日から 2 月 17 日の一週間、東京 23 区内で人々が移動した経路を元に人口密度の変移を可視化したものだ。数値で各所の人数を表示するのに比べ、東京全体にどの程度の比率で人口が分散しているのかを感覚的に捉えることが可能だ。

先も触れた通り、各個人の位置情報を点として捉え線でつなぐことは個人情報の開示に値する。本企画ではこれを解消するためにビジュアライズする対象を逆転し、個人ではなくフィールド側にベクトル[21]情報を蓄積させる方法を提案した。このアイデアは、交通量調査の結果が個人情報にあたらないことから着想を得ている。もし交通量調査員が地球上で 50m おきにメッシュ状に配置（以後これを“ベクトル場”と呼ぶ）でき、それぞれの調査員が交通量をベクトルで記録、加算していった場合、個人情報を持たずに人の移動した形跡のみ表示できるのではないかと考えた。

図 19a-d はベクトル場の記録手法を段階で示したものだ。特定の時間にフィールド上の点 A から点 A′ と点 B から点 B′ に対して人が移動した場合図 19a のような状況が生まれる（以後これを“移動ベクトル”と呼ぶ）。この移動ベクトルをベクトル場に記録する時、それぞれの移動がベクトル場に与える範囲を図 20a の r を半径とした円 o とし、円の中央に近いほどベクトル場に強い影響を与えるものとする。この時、ベクトル場の特定の点 P に与えるベクトル p に A から A′ に移動するベクトル a が与える影響は図 20b のような計算となる。その結果、図 19c のように、ベクトル場はそれぞれの移動ベクトルから影響を受け、この蓄積されたベクトルを加算することで図 19d のようなベクトル場が形成される。この情報を元に、パーティクル[22]を配置し、ベクトル場からの情報によって流動させることで、個人情報を完全に破棄した状態で人の動きの表現を目指した。

draffic visualizerでは最終的に、図 21 のように地図上に人の流れが表示される。画面内の明るい部分

[21]空間上で大きさと向きを持った量

[22]大量の点によって表現される粒子

は人口密度が高く、赤い部分は人の動きがより大きい場所となる。画面の中央には山手線がリング上の帯を作り、各主要駅から千葉や埼玉、横浜方面に流れが分散しているのが見て取れる。考案したアイデアや計算式自体は極めて単純なものであったが、それらをルールとして人の動きに適用し、繰り返し調整しながらイメージしたビジュアルに近づける作業は、絵筆を持って一枚の絵を書いているのと近い感覚で行なわれていた。

3.4 ソフトウェア開発

先に紹介した 3 種のソフトウェアに共通する事象として、既存のソフトウェアを道具としたコンテンツの制作を行なわず、可能な限り最小単位に近いビットを扱って制作をしている点が挙げられる。ソフトウェア開発において、既存のパッケージや用意された雛形などを使用せずにゼロから開発することを一般的に “スクラッチ開発” と呼び、これらの制作手法はこれにあたる。

例えば 3.2MaterialReader では独自の書籍フォーマット MOML を制作している。2010 年当時は ePub(Electronic PUBlication)[23] が標準規格として定まりつつあり、多くの電子書籍リーダーはこの企画をそのフォーマットとして取り入れた。既存のフォーマットを使う場合、エンジニアは最初から道具を手にしており、その労力は最小限にまで抑えられる。また、標準規格は環境に縛られることもなく、安価なデバイスから iPad のようなリッチなデバイスまで様々なハードウェアで再生可能になるため、構築済みコンテンツの価値が著しく失われるリスクも低くなり、非常にメリットが大きい。しかしながら、汎用性の高いフォーマットはより多くの環境に対応するためにその表現の幅が最小限にまで削られている。そのため、ePub の仕様に従って電子書籍を作ることは、多くのデバイスで見てもらえる反面、インタラクティブ性をもたせたり、なめらかなアニメーションをしたりといった視覚的に自由な表現を行うことに向かない。これらの理由から、より自由な発想を行うために記述フォーマットからの制作を行う必要があった。

図 22 Processing によるスケッチ

また、3.1SequentialGraphics においてはスクラッチ開発をする意味がより明確であった。既存のペイントソフトウェアが 2D ペイントを紙の再現を基本的な思想として掲げる中、時間軸を持ち、描画したストロークが繰り返し再生されるキャンバスを既存のソフトウェアで生み出す事は極めて難しく、プログラミングを用いてスクラッチ開発をする必要があったためだ。

これは 3.3draffic においても同様であった。draffic の場合この傾向はさらに強く、BigData[24] と呼ばれる今までには存在しなかった膨大な情報を扱うために、汎用手法の確立していない分野での創作活動を行い、新しいヴィジュアルを生み出した。

このような開発手法をデザインに取り込む場合、自身の手で素材であるビットに触れ、常にフィードバックを得ながら作業を進める事が非常に重要となる。例えばデザイナーが物理的な素材を扱う際、自身の手によって素材に触れ、変化し続ける形状からイメージを膨らませる。2.2 章で挙げたプロトタイプ思考がこれにあたり、繰り返しモノをつくり、検証を繰り返すことで、そこから改良のアイデアや新しい視点を導き出す手法を指す。

SequentialGraphics の開発では、当初 Processing を使用した実験を行った。初期のプログラムは “画面

[23] 国際電子出版フォーラム (International Digital Publishing Forum, IDPF) が定めた電子書籍ファイルフォーマット

[24] 通常の手法では処理する事が困難なほど膨大で複雑な情報

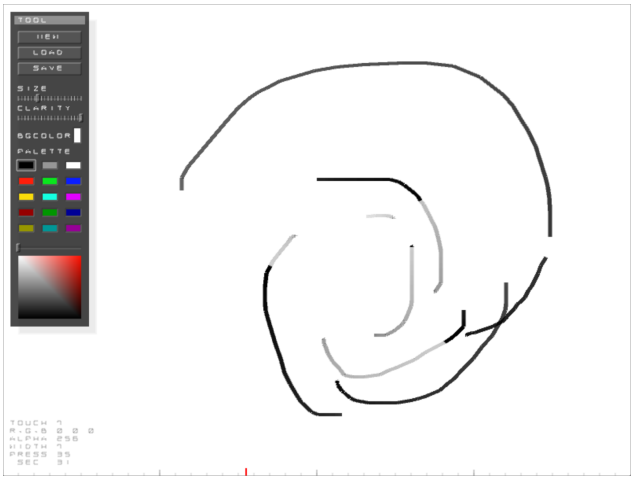


図 23 Processing によるプロトタイプ

上に線を描く” という非常に単純などうさを実現するための、ほんの数行で終わるスケッチのようなものだった (図 **22**)。しかし、当時のコードには単純なミスがあり、画面が更新されるたびに白く塗りつぶされてしまったため、図 **22**a のようにマウスから短い線が出るだけのものだった。しかし、フレーム毎に画面を更新する色を半透明にしたところ、図 **22**b のようにマウスの軌跡に引かれる線が徐々に消えてゆくようになった。予定外の動作ではあったものの、マウスの動きについてくる姿が面白かったため、その軌跡をすべて記録したところ、軌跡が折り重なることで、より複雑な動きをすることに気がついた。この時の感覚は時間軸を伴うことから、紙と鉛筆を前に思考を巡らしたところで生み出されるものではなかった。

その後、この軌跡が残る面白さをより強く感じるようにするために、絵を描くためのソフトウェアとして再度 Processing でのプロトタイプの開発を行った (図 **23**)。このプロトタイプでは、線の太さや透明度、色といった絵を描くために必要な基本的なパラメータを変化させるためにツールボックスを用意し、実際に絵を書いてみることにした。最終的に、これらのプロトタイピングでの知見を元に C++言語によってソフトウェアを構築した。

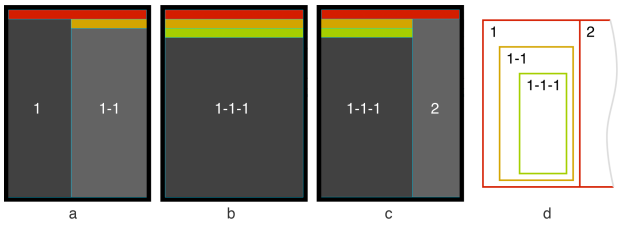


図 24 Processing によるページ遷移プロトタイプ

プロトタイピングは MaterialReader の構築時も数多く行っている。当時本の紙がめくれるようなページ遷移を真似た電子書籍が一般的であった中、我々は電子書籍でしかできないページ遷移を検討していた。その中の一つに入れ子状になったページをスライドすることで、コンテンツが引っかかるように進んでいく案が出た。このプロジェクトでは、最終的に iPad での実装をすることが決定していたため、プロトタイプ段階から iPad での実装が検討されたが、iPad で使用されている言語である Objective-C [25] は C 言語をベースとして作られているため、C 言語と同様にベースとなる部分をしっかりと設計する必要があり、ライトなプロトタイプ作成には向かないため、最初は Processing を使い実験をすることとなった。

図 **24** は実際に Processing で作られたプロトタイプだが、全体が図 **24**d のような入れ子構造になっている。図 **24**a では最初のページから少しスライドした状況で、入れ子状態になった 2 番めのページが右側から入ってきている。入れ子の状態は画面上部に色の重なりで表現されており、そのままスライドしていくことで図 **24**b のように入れ子の最下層のページまでが表示される。最下層の状況でさらにスライドすると、図 **24**c のように今度は一番上の層の 2 ページ目が現れる。

結果的にこのプロトタイプは、図 **25** のように MaterialReader に実装が行なわれた。プロトタイプでは入れ子構造として実装されたアイデアは、その後繰り返し実験が行なわれる中で、「コンテンツが変化す

[25] C 言語をベースにオブジェクト指向機能を持たせた上位互換言語、MacOSX の公式開発言語として使用されている

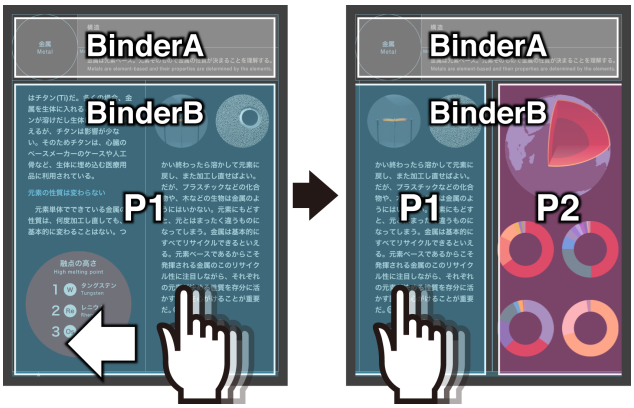


図 25 MaterialReaderに実装されたページ遷移

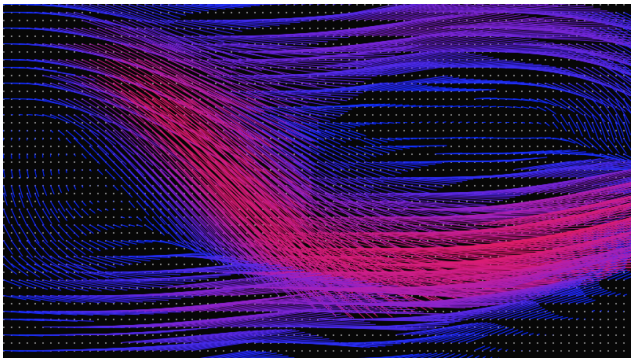


図 26 Processingによるベクトル場のプロトタイプ

る部分のみめくれる」という部分が重要な要素であると判断し、バインダーの連携という形で実現した。図 25 では BinderA が見出しとなっており、BinderB の部分を指でスライドすると、ページ下部のみがめくれていき、見出しの内容が変化する時のみ、BinderA が一緒にスライドする。この他にも MaterialReader では様々なプロトタイプによる実験が繰り返されたが、実際に動くスケッチを目の前で描きながら発想をすることで、より具体的なイメージをプロジェクト内で共有出来、最初は思いつかないアイデアを具現化するに至っている。

drafficでは開発期間が短く、プロトタイピングに許される時間が短かったため、最終的なイメージだけではなく、実際にそれを実現するための計算方法まで考えた上でプロトタイピングに望んだ。

最初に行った実験では、図 26 のように大量の点を配置したベクトル場の上でマウスを動かし、図 19

図 27 ビジュアライザへのベクトル場の実装

で発案した手順を各点に対して行うことで、想像していたビジュアルが実際に生まれるかを確認した。このプロトタイピングには Processingを使い、二次元平面上でその実験を行った。図 19 では各点から線としてベクトルが表示され、その大小を青から赤までのグラデーションで表現されている。頭の中で生み出されたビジュアルと実際の計算の生み出すビジュアルは必ずしも一致せず、初期のプロトタイプでは値が発散し、ベクトルの表示が画面外にまで及んでしまう等の問題が浮き彫りとなった。

その後、計算式の修正を数回にわたって行い、図 27 のように、実際の 3 次元フィールド上にベクトル場を形成することで、実際のデータではどのように見えるのかを確かめる作業を行った。drafficではベクトル表示だけではなく、人口密度グラフや比率グラフなど様々な実装を実験的におこなったが、数値の計算という目に見えない方法によるビジュアル表現において、短い時間でもプロトタイプによってそのアイデアが実現可能か否かを確認することは、自身の書いた筆跡を確認する行為と等しく、次の一手を思案するのに大いに役立った。

3 本のソフトウェアにはそれぞれ開発行程にプロトタイピングという名の試行錯誤の段階が存在することがわかる。このプロセスは紙に鉛筆で繰り返しスケッチをする手法と似ている。現在数多くのプログラミング言語が使用されているが、それぞれの言語

には扱う処理に得手不得手が存在する。よって、その先にあるビットをどのように扱いたいかによって使用する道具を最初に選ばなくてはならない。例えばC言語は高級言語と呼ばれ、非常に高速な処理を記述することができるが、記述するためにメモリ管理など様々な事柄を意識しなければならない。問題が起きるとすぐにプログラムが停止してしまうことから細かい設計が必要となるため、プロトタイピングなどには向かない。これに対し、Ruby や ActionScript 等のスクリプト言語[26]はメモリの管理が自動的に行なわれるため細かく意識する必要がなく、問題がおきても致命的なもので無い限りはその動作を続ける。そのため、後々の設計などを気にすることなく記述でき、プロトタイピングなどの用途に向いている。Reas はその著書[1]で”木工がオーク（樫）、バルサ、松など、いろいろな木材の特性を知っているように、ソフトウェアに精通している人は、種々のプログラミング言語の特性を知っている”と表現している。

物理世界のスケッチでは、鉛筆の先から生み出される形状が常にイメージ通りのものにはならない。予定外の結果をバグと言うのであれば、いわば常にバグが発生し続ける状況で結論を導き出さなければならない。しかし、まだ完全にはイメージが固まっていない状態で、非常に多くの回数試行錯誤を繰り返しながら良いものを選び出すことは、数多くの可能性を元に取捨選択をする機会を得ることに繋がる。一般的なソフトウェア開発において、この試行錯誤の時間は限られている。しかしながら、プログラムを使ったデザインの提案を続ける中で、この予測不可能性との対峙の機会を得ることがソフトウェア開発において非常に重要であるといえる。

[26]ソフトウェアの動作内容を台本のように記述し制御することが可能な簡易的なプログラミング言語

図 28 0 と 1 の羅列によって絵を表現する 2D グラフィクス

4. AirStroke の開発

本章では先に述べた基本思想と研究背景をもとに、3 次元ペイントソフトウェア “AirStroke” の開発を行い、その過程を記す。まずソフトウェア概要で本ソフトウェアの特徴を紹介する。その後、Study1 で表現のための手法の検討、Study2 でその手法を用いた表現技法の検討、Study3 でそれらを直感的に使えるようにする環境の構築の経緯を紹介する。

4.1 ソフトウェア概要

AirStroke はコンピューター上の仮想の 3 次元空間に、エアブラシで絵の具をまき散らすような直感的な表現を行うためにオリジナルのペイントソフトウェアとしてゼロから開発を行った。

2.3 章でも記したが 2D グラフィクスでは、ディスプレイに画像を表示するためには図 **28** のように帯状の 0 と 1 を面として並べて絵を表現している。Photoshop や Painter といった多くの 2D ペイントソフトウェアは、マウスカーソルの軌跡をもとに、この 0 と 1 を並べる処理をしており、その際表現に使用される最小単位がピクセルと呼ばれている。各種 2D ペイントソフトウェアでは、このピクセルを筆や鉛筆といった現実世界で扱う描画道具を仮想的に再現することで、より直感的に表現ができる環境を構築しており、これをラスタ型グラフィクスと呼ぶ。それに対し、Illustrator のように点と線を元に面を表示するものをベクタ型グラフィクスと呼び、主に書籍やポスターといった印刷物に用いられている。

現在映画やゲームなどで広く使用されている 3DCG では、主に図 **29** のように仮想空間に配置し

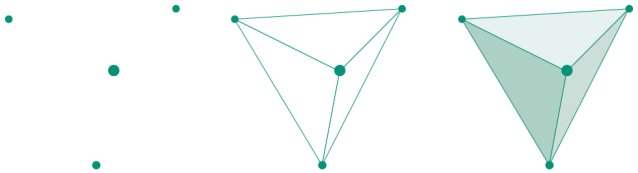


図 29 頂点をつないで面を構成する 3D グラフィクス

た頂点を線でつなぎ、面を構成して光をあてることで物体を表現するベクタ型グラフィックに似た手法が主流となっている。しかし、この方法で自由に表現をするためには、空間上の点をピンセットでつまむようにして細かく移動させながら思い描くビジュアルに近づける必要があり、決して直感的とはいえない。そのため、AirStroke では、大量の点を空間上に配置することで、無重力空間にエアブラシで顔料を撒くような、直感的な表現を目指し、様々な手法を検討した。

現在様々な 3D ソフトがペイント機能を実装している。しかし、それらの多くは 3D 空間上のポリゴンの表面に 2 次元的な描画を実現したものだ。例えば Adams らによる研究『Interactive 3D painting on point-sampled objects』[16] では、3D のモデルの表面に仮想の筆と絵の具をシミュレーションすることで、現実世界の物体に筆で色を塗るような感覚でテクスチャの作成を可能としている。同様に Igarashi らの研究『Adaptive unwrapping for interactive texture painting』[17] では、3D モデル上にインタラクティブにペイントを行うための環境を効率化し、よりに快適に動作するためのアルゴリズムの研究を行っている。さらに、Ehmann らの研究『A touch-enabled system for multi-resolution modeling and 3D painting』[18] ではフォースフィードバック機能を有したデバイスを用い、より感覚的なペイントを可能としたシステムを構築している。また、美術表現のためのものではないが、Owada らによる『Volume Painter』では [19]、立体物の内部に対してペイントを行う事で、内部が詰まったモデルを作成し、切断等を行った際に断面の観察を可能としている。無料の 3D モデリングソフトとして有名な Blender[27] においても同様に、3D モデルの表面にペイント可能な機能を提供している。最も近い研究として、Disney Research の Schmid らによる『Over Coat』[20] の研究が挙げられる。Schmid らの開発したソフトウェアは、土台となる 3D モデルの上にペイントをする形式ではあるものの、モデルの表面を複数の透明なキャンバスでコーティングし、毛や霧のような素材での描画を実現している。これらの研究に対し、AirStroke では土台となる 3D モデルを用いず、大量の点を用いて密度を持った量を表現する事を目指している。

空間上に大量のドットを配置して量を表現する方法として Voxel が挙げられる。Voxel では、点ではなく小さな立方体をレゴブロックのように並べることで量にしているが、主に MRI データの可視化や地質データの可視化など、データビジュアライゼーションの手法として用いられることが多く、美術表現に用いられることは少ない。Voxel はゲーム分野でも活用されており、例えばオンライン上の共有空間で積み木のように立方体を扱えるようにした mincraft [28] が挙げられる。しかし、Voxel は空間上に描画表現をするためのツールとして開発されたものではなく、その最小単位の粒度も絵画技法のためのものとしては荒い。また、細かな立方体が集まっているため、最終的な見た目として中身の詰まった量を表現するのに適している。それに対し、AirStroke では半透明の雲や霧のようなものを空中に散布するイメージを実現するために、より細かく大量な点が描画される必要がある。

以上の理由から、AirStroke では既存のソフトウェアや技法を使用せず、最小単位である三次元空間上のピクセルの概念と独自の表現手法を検討し、ソフトウェア開発をすることとなった。

---

[27] http://blender.jp/
[28] https://minecraft.net/

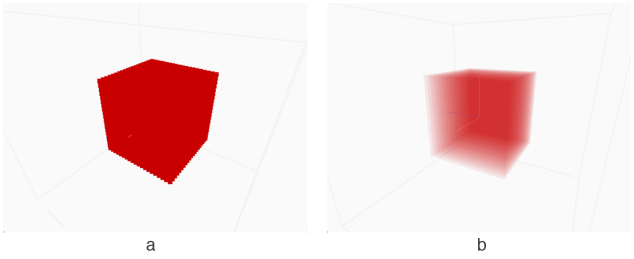


図 30 a: 不透明ピクセルの立方体 b: 半透明ピクセルの立方体

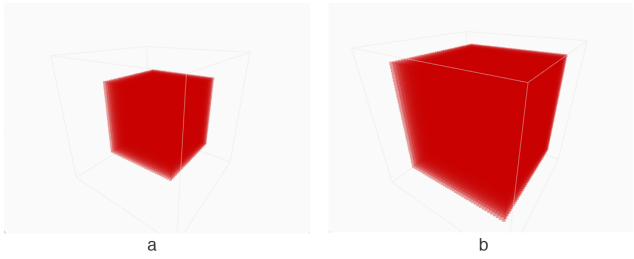


図 31 a: 30^3 の立方体 b: 40^3 の立方体

4.2 Study1:表現手法の検討

空間への描画手法を検討するにあたり、まずは大量のドットを表示することが現実的であるか否かを判断する必要があったため、プロトタイプの制作を行うことにした。

4.2.1 Processing によるプロトタイピング

まずは基本思想の章で挙げた Processing を使って空間にピクセルを配置するサンプルを作成した。

図 **30**a のサンプルは各辺に 20 枚ずつ、20*20*20(以後 20^3 と表記) 枚で合計 8000 枚の正方形の板立方体状に並べ、常にカメラの方向を向けることでドットとして表現したものだ。さらに、雲や霧のような表現をするためには半透明のドットが大量に並ぶ状況を作る必要があるため、各ドットの透明度を上げた図 **30**b のようなサンプルを作成した。これらの実験により、大量の点を画面上に配置することで量として認識できることを確認できた。また、半透明のドットを使用することで、中央が濃く周囲にいくほど色が薄いゼリーのような質感が生まれることが判明した。しかしながら、2D のアイコンですら一辺 32 ピクセルを使用しているのに対し、20^3 ドットの立方体では自由な絵画表現のための環境と呼ぶには解像度が低すぎる。そのため、まずはどの程度の解像度までであれば描画が可能かの実験を行うことにした。

一辺に並ぶドットの数を 10 ずつ増やして実験を繰り返した結果、図 **31**a では 30^3 ドットで 30〜31FPS [29] でていたものが、40^3 ドット並べた図 **31**b では 13FPS

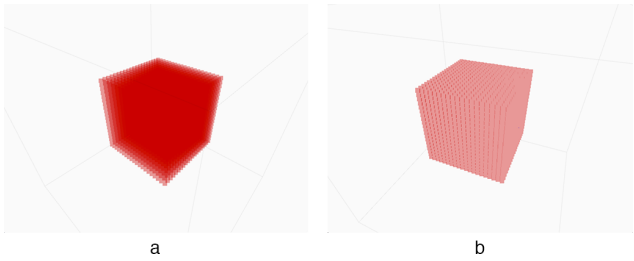


図 32 Z バッファによる透過問題

まで落ち、50^3 ドットでは 1FPS と、ほとんど動かなくなった。2D ペイントソフトウェアでは多くの場合、線などが描画された場合に更新された領域のみを再描画することで、描画の際の処理負荷を軽減している。これは 3D ゲームやアニメーションでは不可能だが、AirStroke の場合、描画内容自体に変更がない限り表示内容が変わらないため、更新があった場合のみ再描画する手法をとることも可能だ。しかし、描画時のズームインやズームアウト、視点の変更などを、空間把握のために変化させながら作業できる環境が望ましいため、1 秒間に最低 30 回の再描画が出来る環境が必要であると考えた。

今回の実験では、30^3 ドットを超えた時点で 30FPS を切ったことから、大量のドットによって量を表現することは難しいと判断できる。しかし、処理速度の問題はその言語自体の持つ特性や、処理の記述方法によって大きく変わるため、次の段階ではより高速な処理を行うための最適化を行うこととした。

速度以外の問題として、透過したドットが正常に描画されないという問題が発生した。図 **32**a では中央から周囲にかけて色が薄くなっているのに対し、同じものを裏から見た場合の図 **32**b では全体が均

[29] Frame Per Sec : 一秒間に描画する回数を指す

```
fill(255,0,0);
rect(0,0,10,10);
```

図 33 Processing で赤い正方形を描画するコード

一な色になってしまっている。これは、視線の手前にある物体で隠される物体や面を検出して描画しないようにする軽量化のための “z バッファ” と呼ばれる描画方法による影響で、手前にある半透明オブジェクトが描画されてから奥にオブジェクトが描画された場合、奥のオブジェクトが描画されない。オブジェクトの描画はメモリ上に並ぶデータの順番で行われるため、この問題を解決するためには描画される前にメモリ上のデータの順番をカメラからの距離によって並べ直さなければならない。しかし、この処理を行うためには全ての表示ドットに対してカメラからの距離による並べ替えを行い、カメラから遠いドットから順番に描画を行う必要がある。この問題においても、表示ドット量の問題と同様に、より高速な環境でのプロトタイピングで対策をとることとした。

4.2.2 C 言語によるプロトタイピング

基本設計で述べた通り、Processing はプロトタイピングを簡単にできるように環境が用意されている。

例えば、図 **33** は Processing で画面上に正方形を描画する部分のコード、図 **34** は C 言語を使い、高速に正方形を描画する部分のコードだが、正方形を 1 つ描画するだけでもその分量や複雑さに大きな違いがあることがわかる。C 言語は高速に動作する反面、処理に細かな記述が必要となるため簡単なことを行うプロトタイピングには向かない。しかしながら、Processing で用意された枠を超えた処理をしたい場合、C 言語と同様に複雑な記述が必要となってしまい、その恩恵を得ることができなくなってしまう。今回のように作るものとその問題点が明確になっており、さらに処理速度がそのネックとなっている場合、C 言語をプロトタイピングに使用するべきと考え、実装をおこなった。

```
float vertices[] = {
    -1.0, -1.0, 0.0,
    -1.0,  1.0, 0.0,
    1.0,  1.0, 0.0,
    1.0,  -1.0, 0.0,
};
float colors[] ={
    1.0, 0.0, 0.0, 1.0,
    1.0, 0.0, 0.0, 1.0,
    1.0, 0.0, 0.0, 1.0,
    1.0, 0.0, 0.0, 1.0,
};
GLuint vbo_vertices = 0;
GLuint vbo_colors = 0;

glGenBuffers(1, &vbo_vertices);
glGenBuffers(1, &vbo_colors);

glBindBuffer(GL_ARRAY_BUFFER, vbo_vertices);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
             vertices, GL_DYNAMIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, vbo_colors);
glBufferData(GL_ARRAY_BUFFER, sizeof(colors),
             colors, GL_DYNAMIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);

glBindBuffer(GL_ARRAY_BUFFER, vbo_vertices);
glVertexPointer(3, GL_FLOAT, 0, 0);
glBindBuffer(GL_ARRAY_BUFFER, vbo_colors);
glColorPointer(4,GL_FLOAT, 0, 0);

glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glDrawArrays(GL_QUADS, 0, 4);
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
```

図 34 C 言語で高速に赤い正方形を描画するコード

本プロトタイプ (図 **34**) では VBO[30] と呼ばれる高速化の手法を使用している。リアルタイムの 3D レンダリング [31] においてボトルネックになる処理として、メインメモリから GPU [32] メモリへのデータ転送が挙げられる。通常の描画方法では 30FPS で描画している最中に 100^3 個のドットの情報を順次転送した場合、秒間 $100^3 \times 30 = 30,000,000$ 回の転送が必要となってしまう。これに対し VBO を使用した場合、描画前に一度全てのドット情報を転送してから描画を行うため、秒間 30 回の転送で済む。Processing でも VBO の使用は可能ではあるが、先に述べた通り複雑な記述が必要となってしまうため、より高速な言語環境である C 言語でのプロトタイピングに移行することとした。

図 **35** は図 **33** の Processing による描画と図 **34** の C

[30] Vertex Buffer Object
[31] 3D 空間上の頂点や色、テクスチャ等の情報をもとに、2D 画面上に描画する処理
[32] Graphics Processing Unit、CPU とは別に画像処理を担当するユニット

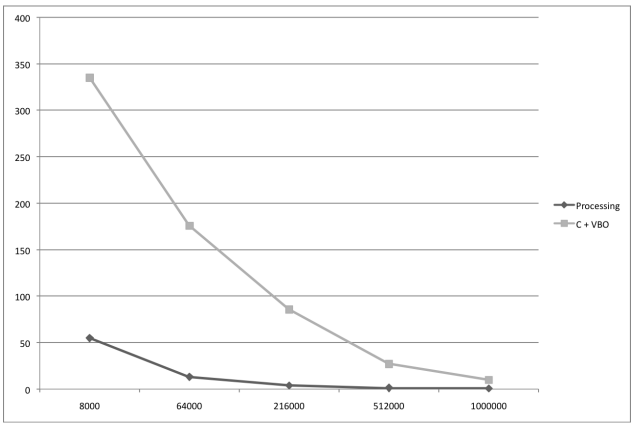


図 35 Processing と C+VBO の描画スピード比較

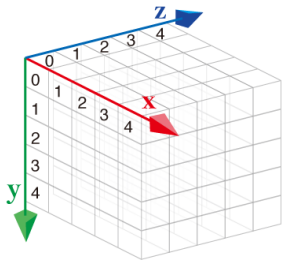


図 36 3D キャンバスのピクセル構造

言語と VBO による描画の速度を総ドット数（横軸）に対して保持できる描画速度（縦軸）で表している。Processing を使用したサンプルでは $20^3 = 8000$ ドットの段階で 30FPS を下回っているのに対し、C 言語のサンプルでは $80^3 = 512000$ ドットまで 30FPS を保持しており、プロトタイピングに使用する環境の変化が描画速度に影響を与えていることがわかる。

描画速度の改善に伴い表示できるドット数は増えたものの、自由な表現を行うための密度としてはまだ十分とはいえない。そこで、実際に保持しているデータに対し、一定量まで間引いた表示することで、使用する環境のスペックに合わせて表示量の調整ができるのではないかと考えた。

図 **36** は一辺が 5 ピクセルによって構成されたキューブ状のデータ構造を表している。2.2 章の図 **2** で示した通り、2D グラフィクスにおけるピクセルのデータは 32bit 4bytes の中に 1byte ごとに 0〜255 までの数値で R,G,B,A の色情報を格納し、これを一列

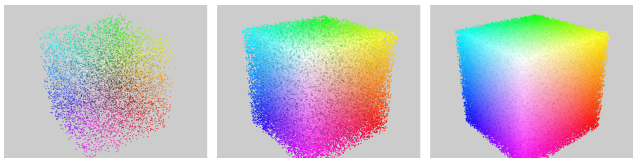


図 37 表示ドットの密度による見え方の違い

に並べた状態を 1 枚の画像データとして保持している。本プロトタイプではこのデータを図 **36** のようにキューブ状に並べてメモリ上に保持しており（以後これを 3D キャンバスと呼ぶ)、描画をする際にこの 3D キャンバスから必要な位置の色を参照して表示することとした。

図 **37** の 3 枚のサンプルは 200^3 の 3D キャンバスから情報を間引いて表示をしたもので、左から順に 10%、30%、50%のドットを表示をしたものだ。見え方自体はカメラの距離によっても変動するが、サンプルでは立方体のキャンバスが画面内に収まることを基準として密度を検討している。結果、30%の密度ではドットが足りず、ノイズが乗っているように見えてしまっているため、最低 50%、100^3 個の表示が必要であることがわかった。描画される対象物がどの程度細密であるかによって、必要な密度も変動するが、プロトタイプ段階においてまずは 100^3 のドットを表示を目標とした。

4.2.3 C++言語によるプロトタイピング

C++言語は、C 言語をオブジェクト指向[33]で使えるように拡張した言語で、クラスと呼ばれるデータ構造により複雑な動作をするプログラムの構築に使用される。現在販売されているソフトウェアにはこの C++言語を使用しているものが多く、大規模な開発を行うためにはあらかじめ設計書を書いた上で、実装ユニットごとにチームで分担して作業を行うことが多い。C 言語をベースとしているため速度は非常に高速ではあるが、C 言語以上に複雑な記述が必要なため、そのままの形でプロトタイピングに使用

[33] データ構造と振る舞いを持つものを全て物体 (オブジェクト) として捉える概念

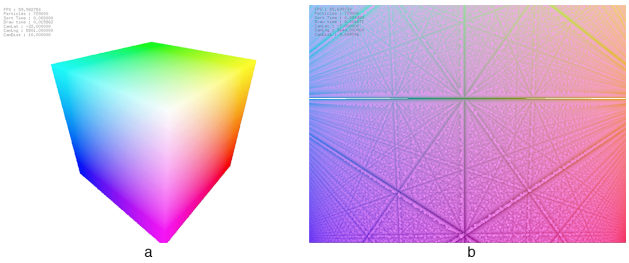


図 38 頂点のみを描画することによる高速化

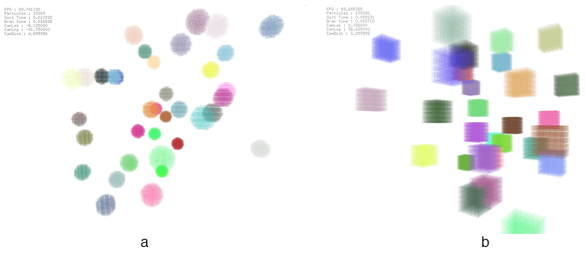


図 39 ドットの集合体による幾何形体の描画

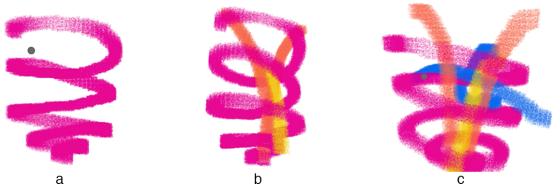


図 40 幾何学形体の連続描画によるペイント

されることは少ない。この問題に対し、Liebermanらは C 言語の速度を保持したまま Processing における記述の容易さを取り込んだ OpenFrameworks[34] というライブラリを開発した。本プロトタイプではこの OpenFrameworks を使用し、プロトタイピングを行いながら最終的なソフトウェアの基本構造を設計した。

まず最初に速度の問題を解消するため、今まで板で表現していたドットを頂点情報のみで描画する手法を試した。図 **38**a では 90^3 ドットの描画を行っているが、このサンプルでは 60FPS の速度が出ている。これまで 1 ドットに対して 4 つの頂点を指定し、板を描画したものを常にカメラの方向に向ける処理を行っていたが、1 ドットの表示に必要な頂点の数が 4 分の 1 になったことと、板の向きを再計算する負荷がなくなったことによって、速度の問題を解消することができた。また、密度が上がることにより、前プロトタイプとは違った硬質な質感を持った量が表現できている。図 **38**b は図 **38**b のカメラの位置を近づけたものだが、この立方体が大量の点で構成されていることがわかる。この段階において、大量の点の配置による量の表現と、それを表現のための要素として使用することが可能であることが明確となったため、実際に描画を行うための準備に取りかかった。

2D ペイントソフトウェアにおけるブラシツールは一般的にカーソルの軌跡に円のような幾何形体を大量に並べることでブラシを再現する。AirStroke では空間上に軌跡を残し、全方向からそれが一定の太さを持った線に見える必要があるため、球体や立方体を並べることで表現することとした。そのため、まずは空間上の任意の場所に透明度を持ったドットの集合体により幾何形体を表示するところから準備を行った。図 **39**a のサンプルでは指定された座標から一定の半径内に指定された色のドットを配置する処理をランダムな位置に対して複数回実行したもので、図 **39**b でも同様に立方体をランダムに描画している。この実装によって、各種ツールの基本となる自由な位置へのオブジェクト描画の準備が整った。

図 **40**a は三次元空間上でのカーソルの軌跡に対し、先に用意した幾何学形体を連続して描画し、3D キャンバス上に螺旋を描いたものだ。ここでの 3 次元空間上でのカーソル位置指定には LeapMotion[35] センサーを使用している。図 **40**b では描画された螺旋形体の中を通すようにオレンジと黄色で線を描き、空間上での前後関係がどのように見えるかを確認した。ここでは当初の想定通り、手前の赤い色が中央を通る黄色とオレンジの線を透過し、雲で描いているような状況を作り上げていることがわかる。さらに図 **40**c では、青い線をほかの線に重なるように描き、見え方を確認しているが、この段階では既に描かれ

[34] http://www.openframeworks.cc/

[35] https://www.leapmotion.com/

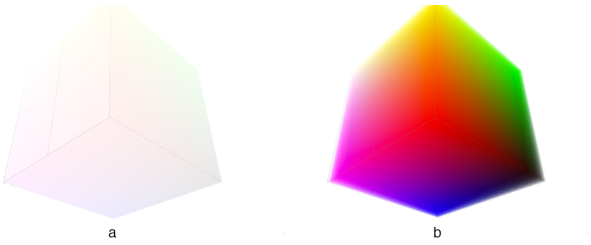


図 41 z-sort による透過問題の解消

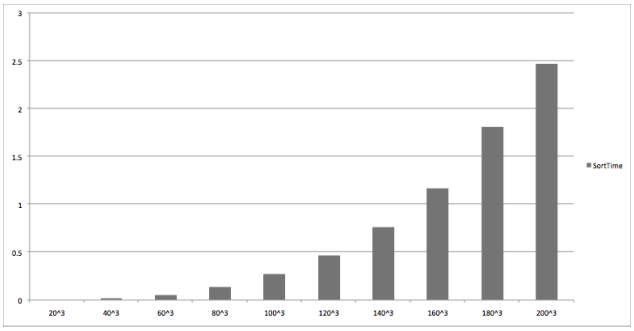


図 42 z-sort にかかる時間

ている座標への描画は単純に情報が上書きされるため、交差部分では青の要素が強くなっていることがわかる。

Processing によるプロトタイプで問題となっていた z-buffer による透過が正常に描画できない問題 (図 **32**) に対し、本プロトタイプでは z-sort と呼ばれる描画順番の並べ替え処理の実装を行い、問題の解決を図った。図 **41**a は Processing でのプロトタイプと同様にある一定方向から見た場合の透過処理において、一番手前のピクセルのみが描画され、奥にあるドットが描画されていない状況が発生している。これに対し、図 **32**b は、カメラの位置が変動した際に全てのドットのカメラからの距離に応じて、描画の順番のソート [36] を行っている。

Processing でのプロトタイピングにおいて、表示のソートには一定の処理時間がかかると記したが、本プロトタイピングにおいて実際にソートにかかる時間を計測したところ、図 **42** の通り 100^3 ドットで 0.264 秒、200^3 ドットで 2.467 秒の時間がかかっている。100^3 ドットで描画とソートの処理を同時に行うと、描画を行わなくても $1/2.467 = 0.405fps$ となってしまい、描画中にカーソルが 2.4 秒に 1 度しか動かない状況となり現実的ではない。そのため、ソートに関しては別のスレッド [37] でで行い、計算が終わり次第新しい並び順と差し替える方法をとった。これにより、100^3 ドットの表示をしている場合、カメラの位置が変わった後表示が一瞬くずれるものの、作業を継続する中で約 0.2 秒後には正常な表示に切り替わる状況となった。

[36] ランダムに並んだ値を昇順や降順など一定の規則によって並べ替える処理

図 43 a:ペンツール　b:エアブラシツール

以上のプロトタイピングにより、大量のドットを空間に配置することが表現のための手法として有効であることがわかり、キャンバスの基本となる要素が揃ったことからペンやエアブラシ等の各種ツールの実装に移行した。

4.3 Study2:表現技法の検討

本章では Study1 で検討した表現手法をもとに、実装した代表的なツールとその制作過程を紹介する。技法の研究段階では、本実装として C++言語と OSX のネイティブ言語 [38] である Objective-C[39] を使い、配布可能なソフトウェアとして実装を行った。

4.3.1 基本ツールの実装

ツールの実装では、まず基本となるペンツールとエアブラシツールの実装を行った。図 **43**a のペンツールでの描画は Study1 での幾何形体を描画する手法を用い、球体をカーソルの軌跡に添って連続して描

[37] CPU で複数の処理を同時に行う際の最小単位
[38] コンピューター本来の言語である機械語
[39] C 言語をベースにオブジェクト指向機能を持たせた上位互換言語、OSX の公式開発言語

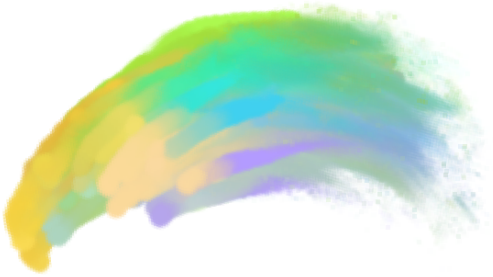
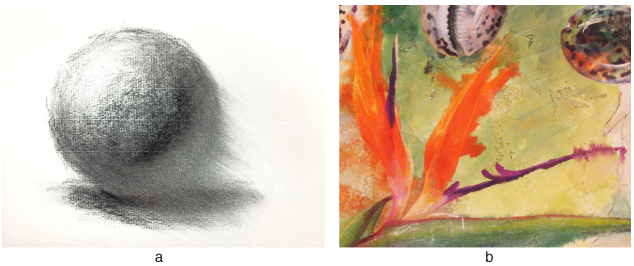


図 44 水彩ペンによる描画



a　　b

図 45 a:木炭によるぼかし　b:絵の具によるにじみ

画している。特定の座標から一定の半径に存在するドットを全て同じ透明度で描画しているため、外周にドットによる段差が発生しているのが分かる。これに対し図 **43**b のエアブラシツールによる描画では、球体の描画座標から外周にいくに従って透明度が高くなることで、外周に発生する段差を軽減し、密度の高い雲で描かれたような質感を実現している。

また、エアブラシの拡張として水彩ブラシの実装を行った。これは、描画色としてブラシに指定されている色を、描画座標に既に描かれている色と常に混色をしながら描くことにより、水彩のように色が混ざる状況を 3D キャンバス上で生み出している。図 **44** は実装した水彩ブラシで描画をした例だが、描画色がキャンバスの色と混ざり、途中から描画色が変化しているのがわかる。水彩ブラシのアルゴリズムについては、須崎らによる著書 [21] を参考にし、２次元用の計算式を３次元用に拡張することで実装を行った。須崎らはその著書の中で “ブラシなどのペイント部分の評価というのは、非常に感覚的なもの” と記しており、実際水彩ツールにおいては “どの程度混ぜるのか” といったパラメータについては可変としておき、使ってみて調整を行うという手法をとった。

4.3.2　にじみ・ぼかし

鉛筆や絵の具などの画材で描画をする際、多くの場合にじみやぼかしといった手法が使用される。これらの手法は故意に使用されることもあれば、画材の特性上偶然発生することもあり、前者として使用する場合、全体のバランスを整えたり、色や質感をより複雑なものにするという効果がある。例えば図 **45**a は木炭で球体を描画している過程でぼかしを使用している。球体の描写の過程では、球の左側のように様々な明度の調子 [40] に対して右下が全体的に暗くなるため、ガーゼ等で右下全体を擦ることで調子を整えている。また同時に、擦ることで紙の凸面だけではなく凹面にまで木炭の粉が擦り込まれることでグレーの彩度に変化が生まれ、より多くの質感が生まれる。図 **45**b はアクリル絵の具による描画だが、描画したモチーフをあえて水でにじませることで鮮やかな花の色を強調している。また、モチーフの一部を絵の中の世界の一部として溶け込ませ、目立つ部分のみを描画することで見え方に心地よい抑揚が生まれる。これらの技法は、一見して過去に描画したものを潰す行為にも見える。しかしながら、その経過で積層された木炭や絵の具の粒子は、新たな層の隙間から覗いたり、透過したりすることで、より複雑な色あいを生む。

2D グラフィクスの分野では、コンピューター上で水彩表現を行うための様々な手法が研究されている。例えば Curtis らによる『Computer-Generated Waterclor』[22] では描画するキャンバスの凹凸までもシミュレーションすることで、写真から一見本物の水彩画と判別できない画像を生成している。しかしながら、これらの研究は主に水彩の見た目を再現することにフォーカスしており、先に述べた描画過程に使用することによる表現の広がりに注視したもの

[40] デッサンにおけるグレーの濃淡や色の階調を指す

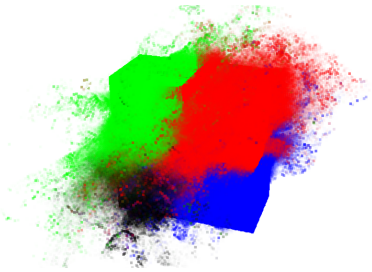


図 46 指先ツールによる編集

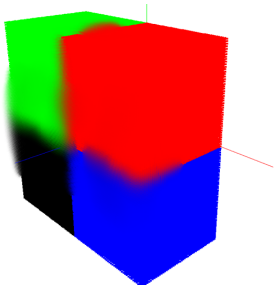


図 47 にじみツールによる編集

ではない。

また、にじみやぼかしといった表現は既存の 3D モデリングの技法においても不可能ではない。しかし、テクスチャを貼った板を細かく動かしながら調整するような現在の手法では、木炭や絵の具を指先で擦りその度合いを確認し、また擦るといった繰り返しの中で試行錯誤するような描写には向いていない。AirStroke ではこれらの技法を三次元空間上での表現に取り入れるために、新たなツールの開発を行った。

図 **46** では 4 色の隣接する立方体を指先ツールによって編集した結果を示している。これは、カーソルの座標から一定の半径の三次元ピクセルに対し、現在のカーソルの色とピクセルの色の平均をとるという非常にシンプルなアルゴリズムで動作している。カーソル位置のピクセルカラーを srcCol、求めたい位置のピクセルカラーを destCol とした場合、具体的には $srcCol \div 2 + destCol \div 2 = destCol$ としてこれを描画座標から一定の半径内のピクセルに対して適用している。図 **47** のにじみツールでは同様に、カーソル座標から一定の半径内のピクセルに対し、

図 48 指先ツールによる花の描画サンプル

範囲内全てのピクセルの平均値を適用している。

図 **48** のサンプルはここまでに実装したエアブラシツールと指先ツール、にじみツールを使用して花の描画を行ったものだ。ここでは非常に簡単な技法を使用しており、エアブラシで色空間に色を球体状に配置し、それを指先ツールで花の形になるように延ばしている。また、下部のドットの荒く見えた部分に対してにじみツールを使用し、空間になじませるようにしている。

4.3.3 描画の抑揚

ここまでの実装により、図 **48** のサンプルにあるような、エアブラシでの描画とそれを擦ったりにじませたりといった編集により、当初の目的であった雲や霧のような素材で空中に描くという目的は達成された。しかしながら、実装したツールで何か細かく絵を描こうとした際、一定のサイズのブラシでは細かい描写や絵筆の持つような抑揚や、そこからうまれる緊張感といったものを生むことが非常に難しいことを感じ、ブラシサイズに変化を生む方法を検討した。

C++言語によるプロトタイピングの章でも挙げたが、三次元空間状の位置の指定には LeapMotion と呼ばれるデバイスを指定している。しかし、このデバイスにはマウスやペンタブレット等と違い、入力の

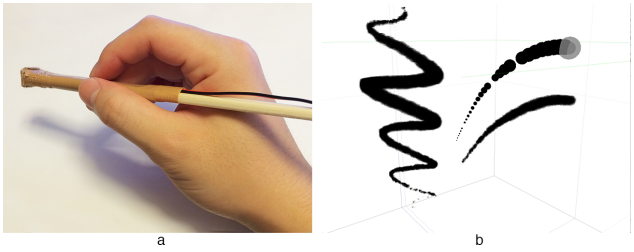


図 49 a:入力デバイスのプロトタイプ　b:ブラシサイズの変化

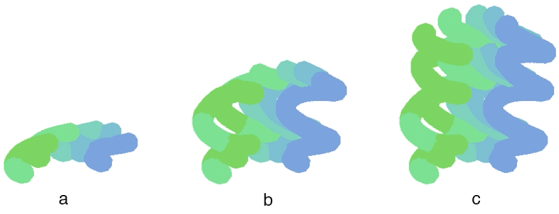


図 50 5 本の指による描画

トリガーとなるセンサーが存在しない。そのため、まずは図 **49**a のように割り箸に圧力センサーをガムテープで括り付けたようなラピッドプロトタイプを作成し、その値を描画ブラシのサイズとして割り当てた。尚、センサーのアナログ値は Arduino[41] を使用して数値に変換し、シリアルポート [42] 経由で入力している。その結果、図 **49**b のように三次元空間状に抑揚のついたブラシの軌跡が表示された。

4.3.4　５本指でのペイント

先の描画の抑揚の検討において、描画する線の太さを調節するために新しいデバイスの検討をしたが、使用している LeapMotion センサーは複数本の指による空間座標の指定ができるため、5 本のすべての指による同時描画がどのような効果を生むのかの実験を行った。

図 **50** は 5 本の指にそれぞれ異なる色を割り当て、画面上に描画を行ったものだ。手を開いた状態の 5 本指の位置にそれぞれのカーソルの先端が割り当てられ、そのまま螺旋状に線を描く過程が図 **50** の a から c までとして記録されている。この実装により、空中で手を動かす行為を描画のための行為として扱うことが可能となった。しかし、実際にこの方法で絵を描こうとした場合、描画対象に対して目的の場所に適切な量の描画が必要となり、5 本の指全てに意識を通わせて思った通りの描画をすることは極めて難しい事が判明した。最初は 5 本の指で描いていても、途中から 1 本の指で描きはじめ、場合によってはセンサーが意図しない指を描画のための指と認識してしまうことから、認識するカーソルの数は設定画面から変更できる仕様とした。

以上の実装以外にも、消しゴムツールやカラーピッカー、画面の回転や移動といった基本ツールに関しては、特に特徴的な実装をしているものではないため、ここでの説明は割愛する。本章の実装により実際に絵を書くための要素がひと通り揃った。

4.4　Study3:表現環境の構築

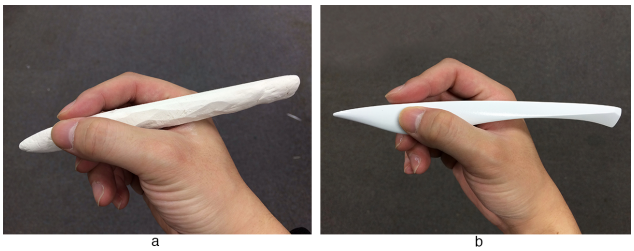
本章では前章までに実装した様々なツールに対し、入力のためのデバイスの検討や UI (User Interface)[43] の実装を行うことで、最終的に配布可能なアプリケーションとして構築することを試みた。本章で紹介する各種インタフェースは、前章と同様に実際に絵を描きながら実装を進め、問題点や新機能を試行錯誤の中から見つけ出すことを目指している。

4.4.1　入力デバイス

4.3.3 描画の抑揚の章において、描画を行う線に対する抑揚をつけることがラピッドプロトタイプによって有効であることを確認した。しかし、割り箸を使ったデバイスは握り心地が悪いだけでなく、センサーからの認識率も悪く、実際に描画に使うには非常にストレスが多い。そのため、快適な描画環境の構築をめざし、より使いやすいデバイスの設計を行った。

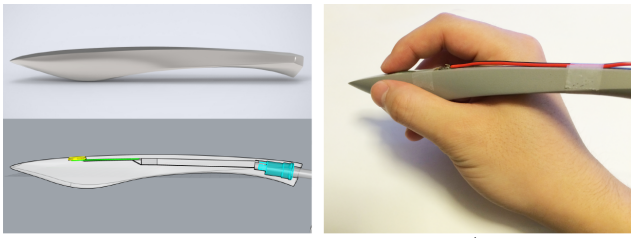
3 次元空間で使用するデバイスは今までにあまり例がなく、規定の形状が存在しないことから、まずは図 **51**a のように粘土によるプロトタイピングで様々

[41] http://arduino.cc/
[42] PC 登場当初から用意されていた通信用インターフェイス
[43] アイコンやボタンといった操作のための機構



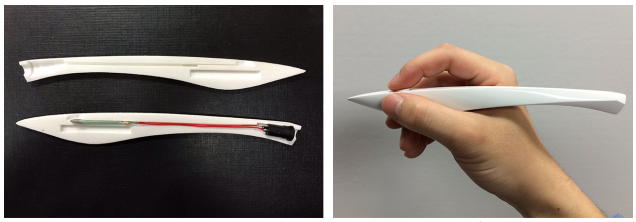
a b

図 51 粘土によるプロトタイピング



a b

図 52 a:粘土をもとにモデリング　b:センサーの位置を調整



a b

図 53 a:センサーを入れた状態　b:完成したデバイス

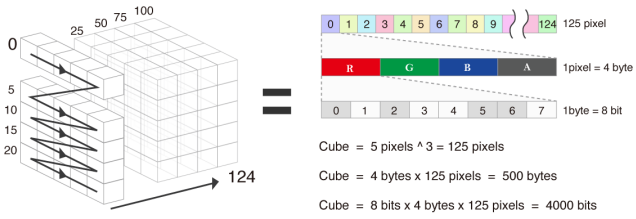


図 54 キャンバスのメモリ構造

な形状を作成した。デバイスをデザインするにあたり、先に挙げた握り心地と、センサーからの認識率の向上にフォーカスして形状を決定した。図 **51**b は数ある形状の中から、より認識率が良く持ちやすい形状の粘土を磨き、軽く表面に塗装をして最終的なグリップ感を確認したものだ。最終的にペンの形状に落ち着いたが、まずは握りやすくするためにグリップ部分を太くし、センサーからの認識率を上げるために平坦に削られた先端部分が握った場所よりも前にせり出している。LeapMotion センサーはデバイスの下部から認識をするため、出来る限り先端部分が前に突出している状況を作り出すために、握り位置を自然に持った際に一般的なペンの場合よりも後ろになるような形状とした。また、ペンの後部からケーブルが垂れた場合も手の上でバランスがとれるように、握り部分に重りを入れる事ができるよう考慮している。

最終的な形状が固まった段階で、ボタンやコネクタなどセンサーを入れる事を想定したモデリングを行い、3D プリントによる出力を行った (図 **52**a)。この段階では、ボタンの位置が確定していないため、図 **52**b のようにセンサーを上面にはり、持った際の指の位置がセンサーの上に来るか否かを確認した。また、ケーブルを繋げて実際に描画をすることで、ペンの前方にどの程度の重りを入れるべきかなどの検討を繰り返し行い、最終的なモデルを作成した。

図 **53**a は最終的なモデルを出力し、中にコネクタとセンサーを収めたものだ。先方に平らな圧力センサーを配置し、赤いケーブルで末尾のコネクタと接続しているのがわかる。表面処理を施すことによって手触りの良さが生まれるよう試みた結果、最終的に図 **53**b のようなペンデバイスが完成した。

4.4.2　画像フォーマットの作成

2.2 章の図 **2** で示した通り、2D グラフィクスにおけるピクセルのデータは 32bit 4bytes の 1byte ごとに RGBA の色情報を格納し、一列に並べた状態を 1 枚の画像データとして保持している。C 言語によるプロトタイピングでも紹介したが、AirStroke でも同様に、3D キャンバスのデータを図 **54** のようにメモリ空間上に一本の帯状のデータ構造として保持している。データは XYZ の直交線を軸とするデカルト座標系で構成され、各ピクセルの空間上の座標は基点となるメモリ位置からの距離により決定する。図 **54** では x:0,y:0,z:0 をメモリ位置 (以後 index と呼ぶ) の 0 番目とし、5 番目まで行くと Y 座標が 1 段進み

x:0,y:1,z:0 へと繋がり index は 5 となる。X が 4 番目、Y が 4 段目までいくと、今度は Z 軸に対して 1 段移動し、x:0,y:0,z:1 といった具合に立方体のピクセルデータが一本の線上に展開されてメモリ上に配置される。この時、座標 (x,y,z) から index を求めたい場合、$index = z * (width * height) + y * width + x$ の式で求められる。実データのサイズはキャンバス作成時に決定され、メモリ上に領域が確保される。

ペイントソフトウェアの環境において、最も重要な機能として描いた絵の保存が挙げられる。描画中、画像のデータは全てメモリ空間上に配置され、アプリケーションの終了とともに全てのデータはクリアされる。そのため、ほぼ全てのソフトウェアが保存のためのフォーマットを持ち、フォーマットをそろえることで様々なソフトウェアで同じ画像を開くことができるようになる。代表的なフォーマットとして、BMP、JPEG、PNG などが挙げられるが、それぞれのフォーマットには特徴があり、用途に応じて使い分けられている。例えば BMP は全てのピクセルデータをそのままの状態でファイルとして保存するため、保存による劣化が全く発生しないが、ファイルサイズは非常に大きくなる。逆に JPEG の場合、ファイルサイズは圧縮により大幅に削減できるものの、圧縮時に画像が劣化するため、オリジナルの保存等には向かない。PNG は JPEG よりも圧縮率は悪いものの、画像を劣化させないまま保存し、透明度情報も付与できる。現在のところ、3 次元上に並んだピクセル情報を保存するためのフォーマットに汎用的なものは存在せず、簡素なライブラリも出回っていないため、まずは独自のフォーマットで情報を保存し、必要に応じて汎用フォーマットに対応することにした。

図 **55** は AirStroke で描画したデータを保存するために作成したファイルフォーマットで、BitmapData のフォーマットを参考としている。特徴として、透過情報を必要とするために画素あたりのビットをア

CubeMapHeader		16byte	
	cfType	2byte	For recognize the file type
	cfSize	8byte	File data size
	cfOffset	4byte	Offset from file head to data head
	backup	2byte	
CubeMapInfoHeader		24bytes	
	ciSize	2byte	InfoSize
	ciWidth	4byte	Cube width
	ciHeight	4byte	Cube height
	ciDepth	4byte	Cube depth
	ciCompression	2byte	0:None 1:RLE
	ciSizeImage	8byte	Image data size
Data	data	nByte	

図 55 3D キャンバスのデータフォーマット

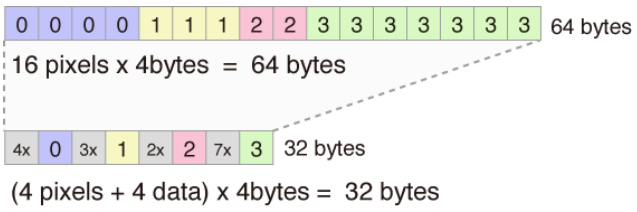


図 56 RLE によるキャンバスのデータの圧縮

ルファ付き 32bit で固定していることや、InfoHeader に奥行きのサイズを示す cfDepth を記録していることが挙げられる。また、当初、3D キャンバスの全ての情報をそのままの形で保存するつもりでいたが、800^3 ピクセルのキャンバスであってもそのサイズは $800^3 dots \times 4bytes = 2048,000,000 bytes$ となり、1 ファイルで約 2GB のディスク容量を使用してしまう。そこで、シンプルで可逆性があることから圧縮フォーマットに RLE(Run Length Encoding) を使用することとした。

RLE 圧縮はメモリ上で同じ値が複数回並んだ際、最初のピクセルの値と繰り返された回数を保存することでデータ量の削減を図る手法で、この手法を用いることで描かれた量に応じた容量で無劣化での保存が可能となる。図 **56** は 4 色で塗られた 16 ピクセルの画像を RLE で圧縮した場合を図解したものだ。保存データでは 4 ピクセル 0 を、3 ピクセル 1 を、2 ピクセル 2 を、7 ピクセル 3 を、と行った具合に繰り返されるピクセル数と色の情報をセットで保存する。この例では元データが 64bytes あるものが保存データでは 32bytes と 50%の圧縮がかかったこと

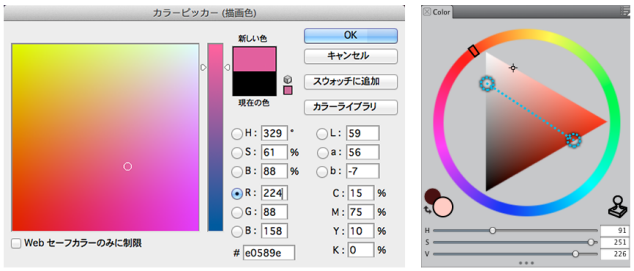


図 57 a:Photoshop のカラーピッカー　b:Painter のカラーピッカー

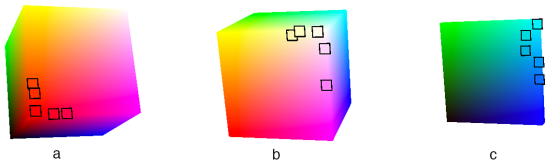


図 58 キューブ状のカラーピッカー

がわかる。RLE には一定数以上同じデータが連続しない場合、逆にデータ量が膨らんでしまうという問題点があるが、連続しないデータが続く場合は連続するデータまでの間の圧縮を行わない PackBits と呼ばれる手法を用いている。

4.4.3　カラーピッカーの実装

色の選択のためのインタフェースとして、一般的なペイントソフトには図 **57** のようなカラーピッカーが用意されている。多くの場合、色相や彩度を決めるスライダーと、色を選択するための色面で構成されており、色を選択するためにはまずウインドウの表示をし、スライダーで色相を調整してから色面で色を選択し、最後に OK ボタンを押すことで色が決定する。しかし AirStroke では、カーソルが基本的に 3 次元で扱われるため、平面上のインタフェースが扱いづらく、さらに 5 本指でのペイントの章で紹介したように、5 本それぞれの指で違う色をピックできることが望ましいため、三次元空間上で 5 つの色を同時に拾うことの出来るインタフェースを目指した。図 **58** は実装したキューブ状のピッカーから 5 本の指で同時に 5 つの色を指定する経過を示している。立方体の 8 つの頂点にはそれぞれ色が割り当て

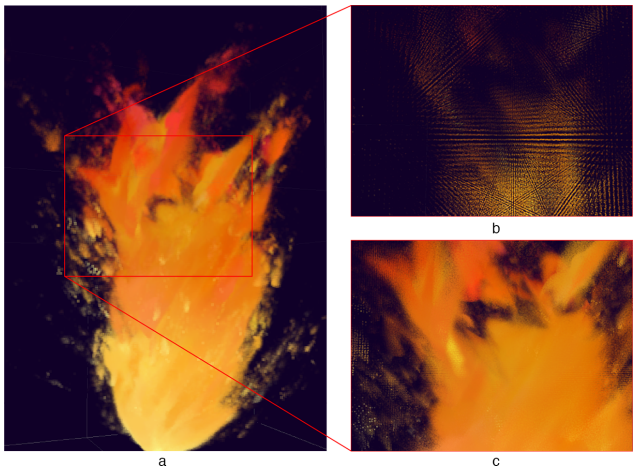


図 59 ドット表示密度の調整

られており、空間上で指を動かすことにより、指の位置から色を拾い上げるようになっている。図 **58**a では指を手前左下に動かすことで左下にある赤エリアから、図 **58**b では右上に指を動かすことで白に近い明るいエリアから、そのまま奥に指を動かすことで水色のエリアから色を選択している。このように、3 次元空間の手の動きをそのまま立体的な色空間に当てはめることで、これまで 1 次元スライダーと 2 次元色面に分離していたカラーピッカーを 3 次元にまとめることが可能となった。

これまでも、カラーピッカーの 3 次元化には様々な研究がなされており、例えば Takatsuka らによる『Three Dimensional Colour Pickers』[23] では、立方体や円錐など様々な形状のカラーピッカーを提案している。本研究では、3 次元ペイントにおけるユーザビリティの観点から 3 次元化しているためテーマは根本的に異なるが、色のマッピングにおける知見として参考とした。

4.4.4　ドットの表示密度

4.2.2 章の C 言語によるプロトタイプの高速化の検討において、画面上へのドット表示を実際にメモリ上に存在するデータから間引いて行うことで軽量化を図った。しかし、実際に描画をする上で表示を間引く割合が一定である場合、カメラとの距離が近

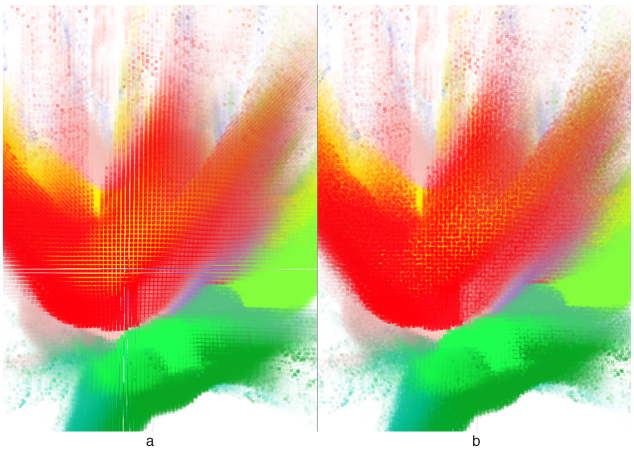


図 60 a:そのままでの描画　b:ノイズを乗せた描画

づくことで表示密度が落ちてしまう。鉛筆や絵の具によって物理的に描画をする場合においても、細かく描画をしたり、全体像を確認したりするために、視点を離したり近づけたりを繰り返しながら作業を進める。表示密度が固定されている場合、このような基本となる視点移動が困難になってしまうため、、モチーフを描画する際のユーザビリティにも影響がでることから、カメラの距離に応じて表示密度を可変とするための実装を行った。

図 **59**a は実装したツールを使用して描画を行ったサンプルだが、図 **59**a の表示密度のまま対象にカメラが寄ることで図 **59**b のようにスカスカになってしまう。逆に図 **59**a の時点で表示密度を上げてしまうと、処理速度の問題意外にも、多くのドットが表示の際に潰れてしまい、処理に無駄が生まれてしまう。そこで、カメラの位置によって表示密度の再計算を行い、画面内に入る部分のみを描画することで、図 **59**c のように処理の負荷を最小限に押さえつつ、一定の見え方を保持しながらズームイン/ズームアウトを可能とした。

また、処理速度を保持するために表示密度を落とした際、その間隔が一定となることで、図 **60**a のように隙間に“モアレ”が生まれてしまう。このモアレは後ろにあるオブジェクトを透過してしまうため、ドットの密度に偏りがあるように見えてしまい、さ

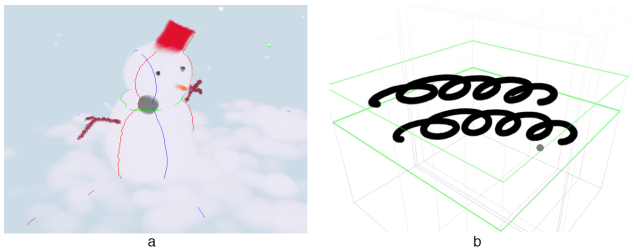


図 61 a:ガイドの表示　b:特定軸のロック

らにはカメラの移動によってモアレの位置が移動するため、画面がチラつき描画に影響を及ぼす原因にもなる。そのため、図 **60**b のように全てのドットに対してランダムなノイズを乗せて表示することで、チラつきを抑え、エアブラシで吹いたような均等な分布を実現している。

4.4.5　ガイド機能

実際にモチーフを用意し描画を進めると、3 次元空間におけるカーソル位置の認識に悩まされた。仮想空間のオブジェクトをどのようにして把握できるようにするかは、Virtual Reality の分野において様々な取り組みが行われたきた。例えば、３次元空間上の仮想 2 次元のキャンバスに対して、触覚的にペイントを行う手法として Baxter らの研究 [24] では仮想の絵筆を 3D 空間上でシミュレートし、描画時のフィードバックをデバイスを通して伝える方法を用いている。また、Ishii らによる『Tangible Bits』[25] では、HCI(Human Computer Inteface) という言葉を使い、様々な方法によって仮想世界に存在するビットに触れる方法について検討している。3 次元空間の認知手法として代表的なものに HMD(Head Mount Display) を使用した手法が挙げられる。最初の HMD デバイスは 1968 年に Ivan Sutherland によって開発され [26]、以後その有効性や用途について様々な研究が行われてきた。2013 年には Oculus Rift[44] と呼ばれる超小型 HMD デバイスが市販され、AirStroke においてもデバイスへの対応を行った。

しかし、多くのユーザーはこれらの特殊なデバイ

[44] http://www.oculusvr.com/

スを持ち合わせていない。本研究ではデザインの視点から、現在のコンピューティングインフラにおいて新しい表現手法を見つけ出す事を目的としており、最終的にユーザーの元へアプリケーションを届けるためには現在普及しているディスプレイ上での表示を基本として制作することが望ましいと考え、ガイド等の表示による解決策を模索した。

最も問題となったのは、カーソルの三次元的な位置の認識で、目的の場所に描いたつもりでも予想よりも手前や奥にズレる事が多い。特に同一色の量を持った塊を描く際、表面に陰影がつかないためにそれが 3 次元的にどのような形状をしているのかを認識する事が非常に困難であった。そこで、図 **61**a のように、XYZ の各軸からレーザー光線を当てているようなガイドを表示することで、カーソルが描画対象のどこに接しているのかを認識しやすくした。図 **61**a では雪だるまの表面上に赤と青と緑の線が描かれることによって、白く潰れてしまっている球体がどのような形状をしているのかを認識することが出来る。

次に問題となったのが、最初に空間上にアタリ[45]をつけることの難しさで、これは最初からイメージを三次元的に意識をする事の難しさが原因の一つだ。この問題に対して、図 **61**b のような軸のロック機能を実装した。これは特定、もしくは複数の軸に対して有効にすることで、その軸に対するカーソルの移動を制限するもので、図 **61**b では Y 軸に対してロックをかけることで、X と Z 軸のみが有効となり、ロック時のカーソル位置の平面上にのみ描画することが可能となる。これにより、まずは各軸に対して二次元的な下書きをし、そこにモチーフを描く事が可能となった。

これらのガイド機能は空間を認識するのに完全なものではないが、実際の描画において必要となった最低限の機能であることから、今後デバイスが普及するまでの間の補助的な機能として有効であるといえる。

[45] 全体をとらえる際の目安となる印

図 62 桜

図 63 蛸

4.5 描画サンプル

本章では開発と平行して描画したサンプルと、複数のユーザーに実際にソフトウェアを使用して描いてもらったサンプルのいくつかを紹介する。

図 **62** は著者が実装を行いながら描画したもので、幹の部分の描画では細かな前後関係の把握が必要となったためガイドの実装を行い、花の部分ではよりダイナミックな流れを表現するためににじみツールの実装を行った。図 **63** はユーザー作品で、ツールの説明を行った直後に描いたものだ。上下左右だけではなく、奥に向かって描画が出来る旨を説明したと

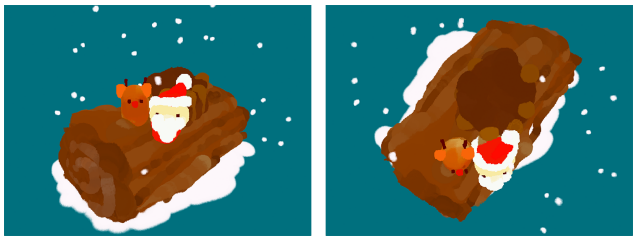


図 64 ビュッシュ・ド・ノエル

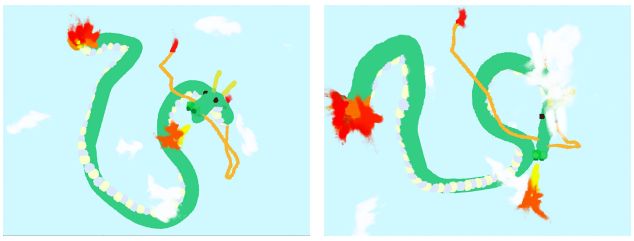


図 65 龍

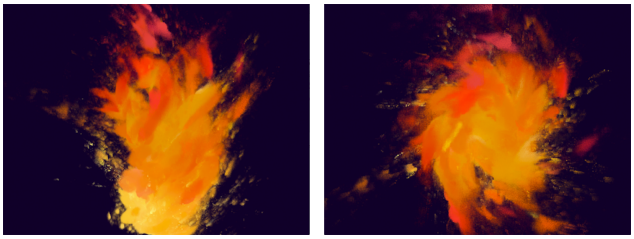


図 66 炎

ころ、空間に広がる蛸の足が生まれた。ツールに慣れないせいか、目鼻部分でカーソル位置が定まらず苦戦した跡がみえる。図 **64** も同一のユーザーの作品だが、ツールに慣れる約 1 時間ほどの間に 2 枚程度の試し書きを経て本作品を描いた。図 **63** と比較すると描く際のカーソル位置の把握が緻密になり、サンタの帽子や目鼻の描写などの描写にツールへの慣れが見受けられる。図 **65** は描画キャンバスの立方体空間を埋めるような大胆な動きをしている龍と周りを取り巻く雲で構成されている。図 **66** は開発初期に描かれた作品で、ブラシのサイズが一定であったために細かい部分は消しゴムで消しながら描画された。その後、この問題点からブラシの抑揚の開発を行い、入力用ペンデバイスの制作に繋がった。

これらの作品はそれぞれ異なるタイミングで描かれているが、各段階で必要に応じた実装が行われ、4.3,4.4 章で紹介した各要素につながっている。このように、より身体性を伴うツールの開発を行うためには、開発と同時にそれを使用し、必要に応じた試行錯誤が重要となると言える。

5. 考察

本章では、研究を通して重要となった点を『ゼロベースでの制作』『素材からの発想』『道具の身体か』の 3 項に絞り、具体例をもとに考察としてまとめる。

5.1 ゼロベースでの制作

3. 章で挙げた 3 本のソフトウェア、及び本研究において開発を行った AirStroke は、それぞれコンピューターの概念を深く理解し、プログラミング言語を用いてゼロからの制作することで、既存の道具の概念に縛られない自由な発想を行った。

3.4 章でスクラッチ開発について触れたが、スクラッチ開発に対し、さらに既存のリソースを一切使用しない手法をフルスクラッチと呼ぶ。2.3 章で紹介した通り、元来ソフトウェアはその多くがフルスクラッチで構築されていた。しかし、近年用いられているソフトウェア開発技法である “オブジェクト指向 (Object Oriented)” は、フルスクラッチとは逆の思想を持っている。これは扱われるデータやその機能を物理世界のモノ (Object) になぞらえて扱うことからそう呼ばれ、プログラマーは抽象化された機能をレゴブロックのように組み合わせることでソフトウェアを構築する。オブジェクト化された機能は何度でも再利用でき、入力の値を変えることでその挙動が変化する。現在のソフトウェア開発はその大半がこのオブジェクト指向を用いている。例えば Windows や OSX のアプリケーション開発では、それぞれの OS 提供元である Microsoft や Apple から SDK [46] と呼ばれる開発のためのオブジェクト郡が配布される。プログラマーは図形を描画したり音を出したりするために SDK に描画命令を送るだけで良い。2.3 章で挙げたように、フルスクラッチで画面上にピクセルを並べて線を引くことは非常に手間のかかる作業となる。SDK はこの作業を抽象化し “画面に線を引く” という命令を受けることで画面上

[46] Software Development Kit

に指定の位置にアンチエイリアスのかかった線を描画する。ビジネスとしてソフトウェアを開発する場合、このような既に準備されたパーツを使用せずにゼロからフルスクラッチで開発することは、よほどの理由がない限り行わない。

紹介した3種類のソフトウェア、及びAirStrokeはこのSDKを使用し作られている。SDKは非常に基礎的な機能の提供をしていることから、その応用範囲は非常に広く、ビットを原材料と考えるのであれば、SDKの利用は直接原材料を扱っているのに限りなく近いといえる。現在SDKで提供されていない機能を簡単に使うことのできる、“ライブラリ (Library)”や、そのライブラリの制御を容易にする“ソフトウェアフレームワーク (Software Framewok)”、さらにフレームワークをGUI[47]などにより視覚的にもわかりやすく使用できるにした“IDE(Integrated Development Environment) ”と呼ばれるパッケージ類を、個人や企業が有償または無償で公開している。このように開発環境は抽象化されればされるほど、低コストで多くのことを実現することができる。

ソフトウェアエンジニアリングにおいて、既にある機能と同じものを再実装することを“車輪の再発明”と呼び、一般的に避けるべき行為としている。結果として、可能な限り簡単に利用できる出来合いのパッケージを組み合わせ、目的を実現しようとするエンジニアが多く、似たような機能やビジュアルを持つソフトウェアが量産されることで多様性が失われるようになった。例えばWebサービスなどが良い例だ。ほぼ全てのサービスがApacheと呼ばれるサーバー[48]ソフトウェアにSQLと呼ばれるデータベース[49]を接続し、PHPやPerl,Ruby等のスクリプト言語[50]環境の上でHTMLを生成し、インタラクティブな部分はJavaScript[51]で記述する。さらに効率的にするために、jQueryとよばれるライブラリによってビジュアルパーツ (アニメーションメニューや画像ギャラリーなど）を簡単に準備できる。ここまで土台が同じ土俵でデザイナーが競合相手との差を産もうとした場合、ユーザーインタフェースやアニメーションの細かな違い程度しか生むことができず、多様性が生まれない。

このように思想として資産の再利用が前提で、既存の枠から外れたアイデアを実現しようとした場合、自身でより土台に近い部分から制作すべきところが視野に入らず、アイデア創出の幅を大きく狭めてしまうことになる。

先も述べた通り、AirStrokeの開発にはSDKやライブラリ、フレームワークといった抽象化されたパッケージを使用している。コンピューターの世界では0と1を自分で並べない限り本当の意味のゼロベースにはならないだろう。しかし、必要でさえあれば、SDKやライブラリ、フレームワークを使用せずに直接0と1を並べることを視野にいれるべきだ。最も重要なのは、フルスクラッチ開発であることではなく、使える資産を活用しながらも発想をゼロベースで行い、既存の枠にとらわれない制作をすることである。

5.2 素材からの発想

本稿で紹介したそれぞれのソフトウェアにおけるプロトタイピングでは、紙や粘土、木材等を扱うのと同様に、試行錯誤を繰り返す中で0と1という素材に直接触れながらアイデアを練り、頭ではなく手で考えることで、素材の特性に合う表現手法を見つけ出した。

3.4章では過去に行ったプロトタイピングについて紹介しているが、AirStrokeにおいても4.2章Study1において3段階のプロトタイピングを行っている。

[47] Graphical User Interface
[48] クライアントからの要求に対して何らかの処理を行い応答するプログラム
[49] 大量のデータを集めて管理し、検索等の要求に従い情報を出し入れするプログラム
[50] ソフトウェアの動作を台本のように記述できる容易なプログラム言語
[51] インターネットブラウザの上で動作する簡易スクリプト言語

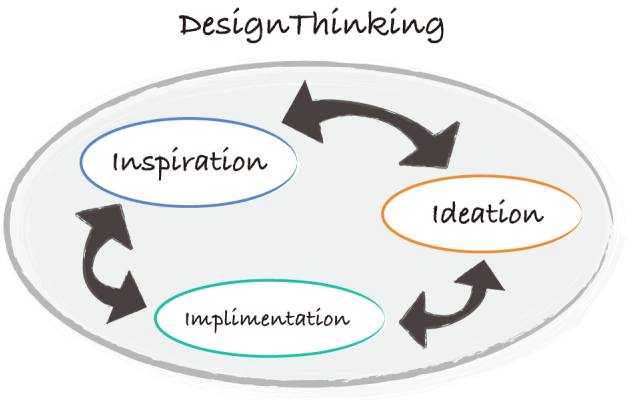


図 67 デザイン思考におけるプロセス

このプロセスでは当初の仮定である“大量の点によって空間に絵が描けるのではないか”というアイデアに対し、現在持つ技術力とテクノロジーで実現可能な限界点を見定めている。4.2.1Processing による最初のプロトタイピング段階ではキューブ状に板を並べることで、仮定であった点の集合による描画が可能か否か、また、スムーズに表示できるドット量の限界による理想と現実の差の確認をした。次に行った 4.2.2C 言語によるプロトタイピング段階では、より難易度の高いプログラミング環境による高速化や、描画方法の最適化により、技術面からのアプローチによるイメージの実現をめざし、4.2.3C++言語を使用した最終段階では技術の限界点を元に、表示内容を限定することにより描画するドット数を間引くなどして、イメージを現実に引き寄せる試みをしている。

このプロセスは、粘土を使い立体物を制作する際のそれとよく似ている。粘土の特性を理解しないうちは、頭に描いた形状を作っても強度が足りずに歪んだり崩れたりしてしまう。試行錯誤を繰り返す中で、粘土の含む水分量などによって形状を保持することができる限界点を次第に理解し、イメージした形状を少し変化させて条件を満たすことで、無理のない美しい形状を生み出す事が可能となる。実際に 4.3 章において行った表現技法の検討プロセスも、描画を繰り返しながら必要となる機能を導きだし、実装を行っている。

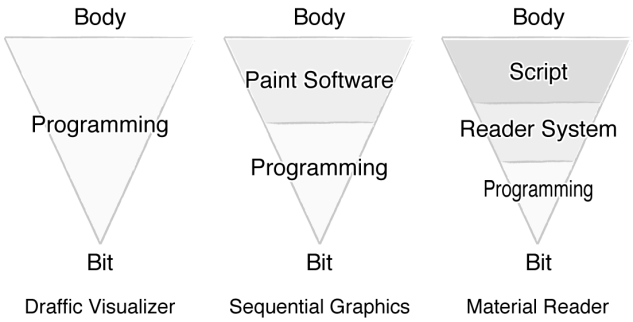


図 68 各ソフトウェアで表現に用いた道具とその技術的抽象度

図 **67** は Brown らによるデザイン思考を実践する際のプロセスを示している。この手法は、常に利用者を中心に考え、現地調査やブレインストーミング、プロトタイピングといった手法を繰り返し用いて、プロダクトを利用するユーザの経験を一連の物語として構築するものである。このような手法を Donald A. Norman はその著書『Living with Complexity』[27] においてエクスペリエンスをデザインすると記している. プロトタイピングはこのプロセスを行う上で非常に重要であるとされている。

コンピューターを活用した提案にデザイナーが関わる際、多くの場合デザイナーがプログラムコードを触る事は少ない。しかし、発想後にまずはラピッドプロトタイプを作り、実際に動くものを手にしながら試行錯誤を繰り返す行為は、紙や粘土、木材といった素材から最も適した表現手法を見つけ出す行為と等しい。著者はエンジニアであると同時にデザイナーでもあるが、デザイナーが自らの手で素材を触り、試行錯誤を繰り返すとき、より良い結果を生み出すと考えている。

5.3 道具の身体化

コンピューターを使ったデザインを行うためには、その構成要素であるビットを自由に扱う必要がある。2.3 章においてプログラミング言語をビットを素材として加工するための二次的道具として定義した。図 **68** は 3. 章で挙げた 3 種類の制作事例について、道具の抽象度についてまとめたものだ。図の上部が

身体、下部をビットとして表し、身体がビットを素材として扱い表現活動を行うために扱った道具とその抽象度を示している。各道具が Body に近いほど技術的な抽象度が高くなり、Bit に近いほど 0 と 1 を意識した手法を用いた表現が必要があることを示している。

例えば 3.1SequentialGraphics ではプログラミング言語を二次的道具として扱い、一次的道具のペイントソフトウェアを開発することで、絵を描くという身体性を伴った表現環境を構築している。これにより、“勢いの描画” という表現を実現した。

3.2MaterialReader はデザイナーとエンジニアの協業を前提とした表現環境の構築を試みている。デザイナーは Script(MOML) を一次的道具として扱い、エンジニアが二次的道具である ReaderSystem を三次的道具であるプログラミング言語によって構築している。この抽象化構造により、デザイナーはスクリプトのみを扱うことで、Bit からのフィードバックを得ることが出来、エンジニアは ReaderSystem を加工することで、表現の幅を広げ続けることが可能となる。さらに、デザイナー自身にスクリプトを記述させることで、製作中のコンテンツに対していかに身体の一部として捉えられる状況を作るかという提案をしている。

それらに対し、3.3drafficVisualizer では、頭のなかにイメージしたビジュアルを実現するために、プログラミング言語以外の道具を用いずに直接計算式を記述することで表現を行っている。この手法は 0 と 1 を直接素材として扱う行為に近く、感覚を通した表現を一度頭のなかでプログラムに変換しながら構築する。プログラミング言語を道具として身体化をするのに相応の修練が必要となるが、道具から一切の規制を受けず、自由な表現が可能であった。

これらの試みは全て、エンボディメントを伴い素材であるビットを扱う方法の模索ということができる。これは 2.3 節章で身体性として記した “エンボディメント (Embodiment)” を伴った開発と言える。最小単位であるビットを感性を元にした表現に使用するためには、身体の一部として捉える必要がある。そのための道具である編集用ソフトウェアや、プログラミング言語は、コンピューターを感性的に扱うためのインタフェースということができるだろう。

AirStroke では、プロトタイピングによって生まれた表現手法から、それを扱うために適した技法を見つけ出し、アプリケーションという形で道具として構築した。これは先に挙げた抽象度では Sequential-Graphics と同様にペイントソフトウェアを道具として制作したものだ。

4.5 章の描画サンプルからも分かる通り、ユーザーが思い通りに描画を行う環境を実現している。これらの経過から、プロトタイピングによって見つけ出した表現手法は、より感覚的に扱うための身体化された道具をデザインすることで、制作物を自分自身の一部としてとらえ表現を行うことが重要であると言える。

6. まとめ

本稿ではデザインの視点から、コンピューターの構成要素であるビットという素材を最大限活用するために重要となる要素を過去の作例から導きだし、3 次元ペイントソフトウェア “AirStroke” の開発を行った。開発の過程では 3 度にわたるプロトタイピングを元に、頭で思い描くイメージと現在の技術で実現可能な境界を見定め、既存のソフトウェアに依存しない技術的解決と発想の転換により、大量のドットを使用して空間に絵を書くための手法を確立した。その後、新たな手法に適した表現の方法を実際に描画を行う中から導きだし、3 次元空間へのにじみやぼかし、抑揚といった技法としてまとめた。同時に、これらの技法を道具として扱えるようにするため、ユーザーインタフェースや入力デバイスといった描画のための環境構築を行い、AirStroke というソフト

ウェアにまとめた。AirStroke を使用した描画サンプルでは、著者だけではなく様々なユーザーによって描かれた作例を挙げ、開発したソフトウェアが他者においても有用であることを示した。考察においては、開発のプロセスにおいて重要となった事項を『ゼロベースでの制作』『素材からの発想』『道具の身体化』としてまとめ、コンピューターを活用したデザインにおける要点として言語化した。

今後デザインには、コンピューター環境を最大限に活用した表現や提案が求められるようになる。デザイナーは既存の道具のみではなく、それらを構成する要素が何であるかを理解し、プログラミング言語という二次的道具を通して既存のソフトウェアによって規定された表現力の限界を超える必要がある。ものづくりは本来自身の手の中にある素材をこね回し、試行錯誤をする中で素材に最も適した手法と技法をもって表現へと繋げていくものであり、特性を知り自身の身体の延長として配置できて初めて作品が生まれる。ソフトウェアや言語は今後も次々と生まれ、それに伴い表現の幅も飛躍的に広がるだろう。そのとき、本研究が既存の枠にとらわれない、自由な発想を生み出す手がかりになれればと願う。

参 考 文 献

1) Reas, C.: *Form+Code in Design, Art, and Architecture (Design Briefs)*, Princeton Architectural Press (2010). （翻訳：FORM+CODE -デザイン／アート／建築における、かたちとコード，早川書房，2010）.
2) 上杉 繁堀内 智貴：道具の身体化現象の評価方法に関する研究ー指示棒接触による錯視の時間変化の検討ー，*Human Interface Symposium 2010*, pp. 1039–1042 (2010).
3) Brown, T.: *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*, HarperBusiness (2009).（翻訳：ティム・ブラウン，デザイン思考が世界を変える — イノベーションを導く新しい考え方，早川書房，2010）.
4) 奥出直人：デザイン思考の道具箱ーイノベーションを生む会社のつくり方，早川書房 (2007).
5) History, I. U. C.: Hollerith 1890 Census Tabulator. http://www.columbia.edu/cu/computinghistory/census-tabulator.html
6) Ryokai, K., Marti, S. and Ishii, H.: I/O Brush: Drawing with Everyday Objects as Ink, *CHI '04*, pp. 303–310 (2004).
7) Ryokai, K., Marti, S. and Ishii, H.: Designing the world as your palette, *CHI'05 extended abstracts on Human factors in computing systems*, ACM, pp. 1037–1049 (2005).
8) Cassinelli, A. and Ishikawa, M.: Khronos Projector, *SIGGRAPH '05: ACM SIGGRAPH 2005 Emerging technologies*, p. 10 (2005).
9) Maeda, J.: Time Paint (1994).
10) Levin, G.: Yellowtail (2000). http://www.flong.com/projects/yellowtail/
11) Gysin, A.: Chronodraw (2001). http://www.ertdfgcvb.ch/p1/chronodraw/
12) Kelley, T. and Littman, J.: *The Art of Innovation: Lessons in Creativity from IDEO, America's Leading Design Firm*, Doubleday (2001).（翻訳：トム・ケリー，発想する会社! ー世界最高のデザイン・ファーム IDEO に学ぶイノベーションの技法，早川書房，2002）.
13) 棚橋弘季：ひらめきを計画的に生み出す デザイン思考の仕事術，日本実業出版社 (2009).
14) Norman, D. A.: *The Psychology of Everyday Things*, Basic Books (1988).
15) Do, T. and Lee, J.: Creating 3D E-books with ARBookCreator, *Proceedings of the International Conference on Advances in Computer Enterntainment Technology*, ACM, pp. 429–430 (2009).
16) Adams, B., Wicke, M., Dutré, P., Gross, M., Pauly, M. and Teschner, M.: Interactive 3D painting on point-sampled objects, *Proceedings of the First Eurographics conference on Point-Based Graphics*, Eurographics Association, pp. 57–66 (2004).
17) Igarashi, T. and Cosgrove, D.: Adaptive unwrapping for interactive texture painting, *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, pp. 209–216 (2001).
18) Ehmann, S. A., Gregory, A. D. and Lin, M. C.: A touch-enabled system for multi-resolution modeling and 3D painting ‡, *The Journal of Visualization and Computer Animation*, Vol. 12, No. 3, pp. 145–157 (2001).
19) Owada, S., Harada, T., Holzer, P. and Igarashi, T.: Volume painter: Geometry-guided volume modeling by sketching on the cross-section, *Proceedings of the Fifth Eurographics conference on Sketch-Based Interfaces and Modeling*, Eurographics Association, pp. 9–16 (2008).
20) Schmid, J., Senn, M., Gross, M. and Sumner, R.: OverCoat: an implicit canvas for 3D painting, *To appear in ACM TOG*, Vol. 30, p. 4 (2011).
21) 荻野 友隆須崎 亮太郎：フルスクラッチによるグラフィックスプログラミング入門，秀和システム (2004).
22) Curtis, C. J., Anderson, S. E., Seims, J. E., Fleischer, K. W. and Salesin, D. H.: Computer-generated watercolor, *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., pp. 421–430 (1997).
23) Takatsuka, Y. W. M.: Three Dimensional Colour Pickers (2005).
24) Baxter, B., Scheib, V., Lin, M. C. and Manocha, D.: DAB: interactive haptic painting with 3D virtual brushes, *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, pp. 461–468 (2001).
25) Ishii, H. and Ullmer, B.: Tangible bits: towards seamless interfaces between people, bits and atoms, *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, ACM, pp. 234–241 (1997).
26) Sutherland, I. E.: A head-mounted three dimensional display, *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, ACM, pp. 757–764 (1968).
27) Norman, D. A.: *Living with Complexity*, MIT Press (2011).