境界としてのソフトウェア・アート

久保田 晃弘
kihiro Kubota

コンピュータ・アートの再定義

デジタル・コンピュータが登場してから，すでに半世紀以上の年月が過ぎた．今日では，都市生活の隅々にまでコンピュータが浸透し，科学者や技術者のみならず，芸術的な創作活動をするアーティストやデザイナーにとっても，その存在や機能と無関係でいることは，ほとんど不可能である．

コンピュータの登場に引き続いて，1960 年代の初頭に誕生した，プログラム言語を主たる手段として芸術作品を生成する「コンピュータ・アート」も，今では「ソフトウェア・アート」と総称される，より広い創作領域によって再定義されるまでに成長した．ソフトウェアを芸術表現のための（独自の方法やスキルを必要とする）ユニークでダイナミックなメディアとして用いるソフトウェア・アートをメディア芸術の世界で最初に定義したのは，2001 年にベルリンで開催された，アートとデジタル文化のための「トランスメディアーレ」フェスティヴァルであった．

そこではソフトウェア・アートが以下のように定義されている［1］

One definition suggested for Software Art is that it encompassesprojects in which self-written algorithmic computer software - stand-alone programs or script-based applications - is not merely afunctional tool but in itself an artistic creation.

この定義の第一のポイントは「self-written algorithmic computer software 」という一節にある．「アルゴリズムを自分で書く」というその行為は，今回の展覧会における，60 年代のアルゴリズミック・アートのパイオニアたちのアプローチと直接的につながっている．さらに最後の「artistic creation」という宣言によって，ソフトウェアが（科学的でなく）芸術的であることの意味を再検討する必要性を提示する．

ソフトウェア・アートの学際性

60 年代と今日のコンピュータ環境の最も大きな違いは，インターネットの有無だろう．ソフトウェアの非物質性，デジタル・データの複製可能性という特徴を生かして，インターネット上でソフトウェア・アートのポータルサイト「runme.org」［2］を構築したのが，自らもソフトウェア・アーティストとして活動するロシアのアレクセイ・シュルギンである．

runme.org のトップページに掲載されている，ソフトウエア・アートのカテゴリーリストは次の通りである．

algorithmic appreciation（2）
> non-code-related（1）
> pseudo-quines（0）

appropriation and plagiarism（5）
> stealing（0）

artificial intelligence（10）

artistic tool（37）
> audiovisual（28）
> narrative（4）
> useless（1）

bots and agents（15）

browser art（15）

code art（17）
> code poetry（7）
> minimal code（1）
> obfuscation（3）
> programming languages（4）
> quines（1）

conceptual software（30）
> without hardware - formal instruction（3）

data transformation（26）
> data collage（8）
> multimedia（5）
> sonification（3）
> visualization（3）

digital aesthetics r&d（9）
> disfunctionality（3）
> low tech（5）

digital folk and artisanship（15）
> ascii art（2）
> audio-visual（1）
> gimmicks（5）
> screen savers（1）

existing software manipulations（7）
> artistic re-packaging（1）
> cracks and patches（0）
> instructions（1）
> software plugins（2）

games（10）
> deconstruction and modification（6）
> public games（2）

generative art（43）
> algorithmic audio（10）
> algorithmic desgn（5）
> algorithmic image（16）
> algorithmic multimedia（8）

hardware transformation（6）

installation-based（6）

institutional critique（7）

performance-based（9）

political and activist software（23）
> cease-and-desist-ware（5）
> illicit software（2）
> rhetorical software（1）
> software resistance（12）
> useful activist software（2）

social software（2）

software cultures - links（13）

system dysfunctionality（6）
> denial of service（3）
> virus - security（3）

text - software art related（47）
> aesthetics of software art（7）
> cultural critique of software（13）
> history of software art（12）
> weblog（1）

text manipulation（30）
> text editors（4）

［注］（ ）内の数字は、2006 年 9 月 20 日現在の、それぞれのカテゴリに含まれる作品数である。

詩や美学といった伝統的なテーマから，人工知能やエージェントといったコンピュータ科学，あるいはゲームのようなポップカルチャーまで，ここで示されているカテゴリーリストの広範さは，今日のソフトウェア・アートの学際性を如実に反映している．さらに同じ runme.org トップページ右側のキーワード・クラウドを子細に見れば，ソフトウェア・アートとそれをとりまくコミュニティが，芸術家と科学者 / 技術者に代表される異分野の触発や協同を喚起する，創造的な場となっていることが良くわかる．

アーティストとプログラミング

コンピュータ・アートが誕生してから半世紀近く経たのち，ある種のリバイバル的な要素も含みながらも，ソフトウェア・アートのような新たな形で，再びアルゴリズムやコードにアーティストの注目が集まった背景として，今から 10 年程前，90 年代半ばにおこったプログラミング革命，より正確にはアーティストのためのプログラミング革命があったことを忘れるわけにはいかない．

それまでのプログラミングは基本的に，コンピュータ科学者や工学者，技術者の文化に根差していた．プログラムは正確に動かなければならない，プログラムは構造（モジュール）化されていなければならない，プログラムは効率良くなければならない，プログラムは簡潔でなければならない…．こんなある種の完璧主義に根差した価値観がプログラムの美のベースにあり，それをいかにエレガントに実現するかが，すなわち「アート・オブ・プログラミング」の実践であった．そうした文化を反映して，従来のプログラミングの教科書の多くは，数の並べ替えや文字列処理，数値計算やデータ構造といった例題から始まる［3］．しかし視覚的，空間的にものごとを考え，直感的，身体的にコミュニケートするアーティストやデザイナーに，そうした抽象的・論理的な例題は向いていない．

アーティストがまず最初に興味を持つのは，ディスプレイ上に点を打ち，線を引き，面を塗ることであり，モニタスピーカから，クリック，サイン波，ノイズなどの音を出すことである．だからアーティストやデザイナーのためのプログラミング（教育）は，まず画面上に線をひくことや，スピーカで音を出すことから出発する．プログラミングの結果が視覚や聴覚に即座にフィードバックされ，それを受けて即座にプログラムを修正できるようになっていなければならない．

最初の内はプログラムをシンプルにしたり，エレガントにすることをあまり考える必要はない．どんなに愚直で非効率的な書き方でも，とにかくプログラムを動かすことが重要だ．一旦プログラムが走りさえすれば，あとは点や線を動かしたり，音を変化させたり，マウスやキーボードでインタラクトさせる過程で，構造化や抽象化のテクニックを必要に応じて少しずつ身につけていけば良い．アーティストのためのプログラミングは，点や線を「描く（造形）」「動かす（運動）」「インタラクトする（反応）」という 3 つのステップによる，ヒューリスティックなプロセスに根差している．

プログラミング環境とコミュニティ

こうしたプログラミング教育を行なった先駆者が，80 年代初頭にプログラミング言語 LOGO を開発したシーモア・パパートや，90 年代に DBN（Design byNumbers）を実装した米 MIT のジョン前田である［4］．そこから生まれた Processing［5］というオープンなプログラミング環境は，インターネットを介し ，世界各地のアーティストやデザイン教育の現場に拡がった．プロセッシングでは，個々のプログラムのことをスケッチ（ブック）と呼ぶが，これは Processing がアルゴリズミックな作品をつくるためのドローイングやスケッチ，すなわち（あらゆる創作にとって必要不可欠な）プロトタイピングに特化したツールであることの表れである．

今日では Processing 以外にも，フリーでオープンなプログラミング環境が，いくつも公開されている［6］［7］［8］．同様に，それらを活用したアルゴリズミックな作品をネット上で公開しているアーティストも多い［9］［10］．

オープンなプログラミング環境には，そのユーザやディベロッパを核とした，オープンなコミュニティがインターネット上に形成される．そこではアプリケーションのみならず，ソースコードやドキュメントが自由に交換され，知識や文化が広く伝達，共有されている．最初に紹介した runme.org のような，アルゴリズム・アートに関連したポータルサイトや，数々の Blog サイトの存在も重要だ［11］［12］［13］［14］．

インターネットを介したオープンな情報共有や共同学習の場は，学会や論文の壁に守られていたアカデミズムとストリートの境界を融解し，今では誰もがその気にさえなれば，プログラミングに必要な環境や知識を自分の手で得ることができる．画像ファイルやサウンドファイル同様に，プログラム・コードも自由にサンプリング／リミックスすることが可能であり，ヘルプファイルやサンプルファイル，オンライン上の作品（のソースコード）を活用することで，発見的な，あるいは事例と類推による学習や制作を自在に行うことができる．そこにはある種の無駄や試行錯誤が入り込まざるを得ないが，そうした一見冗長なものの中にこそ，新たな表現の可能性が潜在している．プログラミングやソフトウェアの世界にも，DJ カルチャー［15］が深く浸透しつつある．

素材としてのプログラミング言語

アルゴリズムを自ら書くソフトウェア・アーティストにとって，プログラミング言語は，木材や金属，絵具やキャンバスのような，表現のための「素材」である．アートは，最終的には素材や手法に依拠しない，普遍的な何ものかを獲得しなければならないし，素材そのものに囚われ過ぎることはフェティシズムの罠に堕ちてしまう危険性はあるが，それでも自分が選択した素材の特徴を知り，それを自在に操作できるようにすることは，表現することの基本のひとつであることに変りはない．逆にいえば，それぞれの素材には，その素材に適した形式や使用法（過程）があり，その選択や組み合わせの適切さに対する洞察が，優れた表現を生み出すためには必要不可欠である．

汎用のアプリケーションを使用せず，直接プログラムを書くことで表現を生み出すことは，自らが選択した素材を自らが直接操作する，ということである．その背景には，直接操作によるディテールの制御や，高速かつ大量の処理速度といったプラクティカルな利点のみならず，自分が使用するツールや環境を自分で構築するという，リベラルな DYI 精神がある．企業によって開発される汎用のアプリケーションは，ハードウェアの進歩と一体化した商業的利潤追求のためのヴァージョンアップにより過剰に肥大化し，その結果汎用アプリケーションの枠内で表現できることは，その豪華な見かけとは裏腹に，日に日に狭められつつある．

プログラミングという素材の直接操作によって，コンピュータが持つ本当の潜在力や可能性を発掘し，そこから自分ならではの表現を見い出すことこそが，コンピュータやソフトウェアを芸術的なメディアとして最大限に活用するための，今なお最も有効なアプローチである．

ライヴ・コーディング

近年「ライヴ・コーディング」あるいは「オン・ザ・フライ・プログラミング」「ジャスト・イン・タイム・プログラミング」と呼ばれる手法によるパフォーマンスが，一部のソフトウェア・アーティストの手で行なわれ始めた．ライヴ・コーディングとは，事前に準備したプログラムを実行させるだけでなく，その場で並行してプログラムを書いたり修正することで，表現を生成するアルゴリズムをそのものをリアルタイムに生成・操作するパフォーマンス手法である．

ライヴ・コーディングにおいては，プログラミング言語の素材としての特徴が明確かつ切実に表れる．例えば，LISP や Scheme あるいは Haskel のような関数型言語は，C++ や Java

のような命令型言語に比べて，プログラムを格段に短かく書くことができる．こうした生産的な言語を用いれば，プログラム・コードを直接操作することが，ボタンやスライダといったGUI以上に，操作性が良く自由度の高いインターフェイスに成り得る．それは単なるパフォーマンスの方法論を越えて，今日の商用アプリケーションを支配してきたGUIパラダイムの再検討と，その本質的な見直しにもつながっていく．

ライヴ・コーディングに関する代表的なオンライン・コミュニティがTOPLAP［16］である．Autoshop やAuto-Illustratorといったジェネラティヴなグラフィック・ツールの開発者でもあるエイドリアン・ワード［17］，klipp av［18］というラップトップAVユニットで活動するニック・コリンズとフレドリック・オロフソン，fluxus［19］というSchemeベースのライヴ・コーディング用グラフィックス言語を開発するデイヴ・グリフィス［20］といった，コーディングの強者たちが立ち上げたこのサイトには，ライヴ・コーディングをめぐるさまざまなインフォメーションやドキュメント，ディスカッションがアップロードされている．

ライヴ・コーディングを行うためのプログラミング言語にもさまざまなものがある．既出のfluxusだけでなくChucK［21］やimpromptu［22］のように，ライヴ・コーディングのために新たに開発されたものを始めとして，汎用言語のLISPやPerl，LOGOなどからSuperCollider3［23］のJITLib（Just-In-TimeLibrary）まで，さまざまな言語が使用可能である．基本的には，ソースコードを逐次解釈しながら実行できるインタープリタ言語を用いるのが良い．とはいえ前述のように，ライヴ・コーディングの可能性は，プログラミング言語のデザイン・パラダイムや実装の良し悪しに大きく依存する．

芸術と科学の境界

素材としてのプログラミング言語の可能性を，人間のスキルや判断によって最大限に活用しようとするライヴ・コーディングは，アルゴリズミックな表現におけるプログラム・コードそのものの意味や形式に，もう一度光を当て直す．アーティストによるプログラミングは，往々にしてコードから生成されるアーティスティックな表現のみに注意が向き，プログラムコード自体は，映像やサウンドを生み出すための影の存在になってしまいがちだ．だが本当にそれで良いのだろうか．コードは表現を生み出すための裏方であり，その中身の詳細やコードそのものの意味や形式は問われなくとも良いのだろうか．

米スタンフォード大学のコンピュータプログラミング芸術名誉教授のドナルド・クヌースは「科学はコンピュータに説明できるくらい私たちが良く理解していることであり，芸術はそれ以外のすべて」だと語る［24］．だとすれば，コンピュータ科学の対象であるコードによって記述できないことこそが芸術であり，コンピュータ・アートとは，芸術という語り得ぬものに沈黙せず，逆に饒舌になることだといわなければならない．

確かに芸術と科学を一点に収束させるのは不可能だ．しかしだからこそ，その境界に着目しなければならない．科学と芸術が共に変化し続ける限り，その境界も常に流動的であり続ける．最初に示したように，学際的なソフトウェア・アートにおいて，形式的なプログラム・コードとそれを実行することから生み出されるアーティスティックな表現は，科学と芸術の境界を顕在化する，1対1に結びついた双子の兄弟である．アーティストのためのコンピュータ環境がインターネットを介して世界各地に普及しつつある今，コードの実行結果としての造形や構成のみならず，コードそのものの美を支えてきた抽象性や汎用性，効率や整合性から生まれる美的感情を再考することが必要だ．そのためにもライヴ・コーディングのようなマージナルなプログラミングを通じて，アート・オブ・プログラミングとコンピュータ・アートの境界の在処を問い，そこに積極的に介入していくことが，境界としてのソフトウェア・アートの，ひいては芸術と科学の未来を考えるための，重要なテーマのひとつになるだろう．道はまだ始まったばかりである．

参考文献

［1］transmediale.01, http://www.transmediale.de/01/en/conf_artistic.htm
［2］runme.org - say it with software art!, http://runme.org/ （シュルギンはオルガ・ゴリウノーヴァらと，ソフトウェア・アート・フェスティヴァルのReadme, http://readme.runme.orgも主宰している．）
［3］アラン・ビアマン『やさしいコンピュータ科学』アスキー，1993.
［4］ジョン前田『Design By Numbers―デジタル・メディアのデザイン技法』ソフトバンク，2001.
［5］Processing 1.0（BETA）, http://processing.org/
［6］DrawBot, http://drawbot.com/
［7］NodeBox, http://www.nodebox.net/
［8］Context Free, http://www.contextfreeart.org/
［9］Gallery of Computation generative artifacts, http://www.complexification.net/
［10］FLIGHT404, http://flight404.com/
［11］Code & form » Computational aesthetics,http://workshop.evolutionzone.com/
［12］Generator.x: Software and generative strategies in art anddesign, http://www.generatorx.no/
［13］artificial.dk - your resource on net art, software art, and othercomputer based art forms, http://www.artificial.dk/
［14］CodeTree: Watch Your Code Branch Out and Grow,http://www.codetree.org/
［15］ウルフ・ポーシャルト『DJカルチャー ポップカルチャーの思想史』，三元社，2004.
［16］Main Page - Toplap, http://www.toplap.org
［17］SIGNWAVE UK - Home, http://www.signwave.co.uk/
［18］KLIPP AV, http://www.klippav.org/
［19］fluxus, http://www.pawfal.org/Software/fluxus/
［20］dave's page of art and programming, http://www.pawfal.org/dave/
［21］ChucK => Strongly-timed, On-the-fly Audio Programming Language, http://chuck.cs.princeton.edu/
［22］impromptu, http://impromptu.moso.com.au/
［23］SuperCollider - Hub, http://supercollider.sourceforge.net/
［24］ドナルド・クヌース『コンピュータ科学者がめったに語らないこと』エスアイビー・アクセス，2003.