

# OPTiにおける極限集合描画アルゴリズム

和田昌昭

奈良女子大学理学部情報科学科

## 1 最近のプログラミング事情

この「トポロジーとコンピュータ」には様々な方面の方が参加しているので、最初に、プログラムを書いたことはないが興味を持っている、あるいは、これからプログラミングを始めてみたいという人のために、最近のプログラミング事情について少し書いてみよう。

### 1.1 プログラミング言語

まず、数あるプログラミング言語の中でどの言語を使えばよいのか、ということであるが、数学に関してプログラムを作るという目的であれば、C言語またはC++言語を用いるのが最も一般的であろう。ちなみにOPTiのソースコードはC++言語で書かれている。

C言語は、科学技術分野で現在最も広く使われている言語である。プログラミング言語としての歴史が長く、プログラムライブラリ等の蓄積があることがC言語の利点である。また書籍も豊富で勉強しやすい。

C++言語は1990年頃に新たに作られた言語で、C言語をベースに、オブジェクト指向プログラミングが行えるような拡張が行われている。言語仕様はC言語に対して上位互換となっており、C言語のプログラムはそのままC++言語のプログラムとしても実行できる。

### 1.2 プログラミングの実際

プログラミングを始めようとする、教科書を読むだけではもちろんダメで、コンピュータ自体やプログラムを開発するためのパッケージを入手する必要がある。初心者にとってハードルが高いのは、実はプログラミングの内容ではなく、プログラミングを行うことができる環境を用意することだと思う。その意味で、たいていのC言語の教科書の最初に出てくる“Hello World!”と表示するだけの味気ないプログラムを実際に実行できるようになることが最初の大きな難関と言える。

プログラミング自体については、条件分岐や繰り返し（C 言語ではそれぞれ if 文と for 文）を使ったプログラムはすぐに書けるようになる。C 言語の「関数」は一連の手続きをまとめたもので、数学における関数とはかなり違ったものであるが、その関数を使いこなせるようになるのが次のハードルと言えるだろうか。特に、ポインタの概念を理解することが重要で、ポインタを縦横に用いた関数の再帰呼び出しが自由に行えるようになると、数学的にはかなり高度なプログラムを書くことができる。

C++ 言語においては、これらに加えてオブジェクト指向プログラミングが行えるようになってきているが、このオブジェクト指向という考え方は、それなくしてはプログラムが書けないというようなものではない。が、プログラムの再利用や、大規模プログラムのソースコードの管理を考えると、オブジェクト指向でプログラムを書いておくと便利という性質のものである。

### 1.3 オペレーティングシステム

現在一般的に使われているほとんどのコンピュータは Windows マシン、Macintosh、Unix マシンに分類することができる。これらはコンピュータに搭載されているオペレーティングシステム（OS）による分類で、コンピュータ自体の製造メーカーはさまざまである。NEC、富士通、IBM、Sony 等々、Apple を除くほとんどのコンピュータメーカーが OS として Microsoft 社の Windows（Windows2000、WindowsNT、WindowsXP 等の総称）を採用しており、市場においても Windows マシンが大部分（9割？）を占めている。

Apple 社は、独自の OS である Mac OS を搭載した Macintosh シリーズのコンピュータを販売しているが、デザイン、出版、教育といった一部の分野で根強い人気がある。大学や研究所等では Macintosh が用いられている場合が多く、筆者も普段 Macintosh を使っている。従って OPTi も Macintosh 用のアプリケーションソフトになっている。

Unix は 1970 年代に作られて以来現在に至るまで使い続けられている OS で、中には商用のものもあるが、一般的にはソースコードが公開され、多くの人がボランティアベースで改良を続けている。Windows や Mac OS といったパソコン用 OS と異なり、初めからマルチユーザー用 OS として開発されてきたために、ネットワーク管理にすぐれ、サーバーコンピュータ用の OS として採用されていることが多い。BSD や System V、あるいは商用 Unix である Solaris といった変種があり、最近では Linux と呼ばれる Unix の一種がパソコンユーザを中心に広く用いられている。パソコン上で Unix を使う場合には、初めから Unix がインストールされているということは少なく、Windows マシンに後から CD 等により Unix をインストールして用いることになる。

Apple 社は、Mac OS 9 までは独自 OS として開発してきたが、Mac OS X（テント）と呼ばれる現在最新のバージョンを開発するにあたって、OS のカーネル部分

(メモリ管理やスケジュール管理を行う OS の最も核の部分) を Mach カーネルと呼ばれる BSD Unix 系のカーネルに置き換えるという大きな変更を行った。その結果、今は Mac OS 9 の使用者と Mac OS X の使用者が入り交じった過渡期にあたり、その両方を考慮せざるを得ない状況になっている。

Mac OS 9 用に作られたアプリケーションを Classic アプリケーション、Mac OS X 用に作られたものを Carbon アプリケーションと呼ぶが、その双方が Mac OS 9、Mac OS X の両方で動くような工夫がなされている。Apple 社ではこれら以外にも Cocoa アプリケーションと呼ばれる Mac OS X 専用のアプリケーション開発もサポートしている。ちなみに OPTi 3.30 は Classic アプリケーションであるが、近い将来 Carbon アプリケーションに変更する予定である。

表 1: Mac OS 9 から Mac OS X への移行

	Classic アプリ	Carbon アプリ (CarbonLib 使用)	Cocoa アプリ
Mac OS 9			×
Mac OS X	(Classic 環境)		

## 1.4 プログラム開発環境

プログラム開発は、OS によって異なる。C++ 言語 (あるいは C 言語) を用いるとして、Windows 上でプログラム開発を行うのであれば、使用するプログラム開発環境は Microsoft 社の Visual C++ パッケージが事実上の標準となっている。

最近では、アプリケーションプログラムを開発する際に、プログラムを一から全部書くということはほとんど無くなった。どうするのかと言うと、メーカー等があらかじめ C++ のクラスライブラリという形でアプリケーションのひな形を用意してくれており、それに自分が必要とする機能を追加する部分だけプログラムを書き足してアプリケーションを作るのである。Windows 上には、MFC (Microsoft Foundation Classes) と呼ばれるクラスライブラリが用意されており、それを用いるのが便利である。

Macintosh 上でプログラム開発を行うのであれば、以前は Symantec 社や Apple 社自身が作ったプログラム開発環境を用いることもあったが、現在使用できるのは、事実上 Metrowerks 社の CodeWarrior というパッケージに限られている。パッケージ自体は完成度が高く、また Metrowerks 社のサポート体制も整っていて、非常に使いやすい。ただし、学生が使用するには価格がかなり高いのが難点である。CodeWarrior を用いてアプリケーションプログラムを開発するには、同じく Metrowerks 社が用意している PowerPlant と呼ばれるクラスライブラリを用いるのが便利である。

Unix 環境でのアプリケーションプログラム開発については、筆者は詳しくないので概略しか分からないが、たとえば GNU のツール類を用いて X Window と OSF/Motif パッケージを使ってアプリケーションを作るといのがかなり一般的だと思う。

Unix に関しては、さまざまなレベルの開発環境が整っており、ネットワークを通じて無料で使用可能となっている。また、ソースコードも公開されていることが多いので、必要に応じて自分で変更を加えることなども可能である。ただし、これらのツール類が個人によって開発・管理されているために、どのようなツールが使用可能かの情報が得にくいことや、ツールによってサポートの質に大きなばらつきがあることが欠点であろうか。

表 2: プログラム開発環境

OS	Windows	Mac OS	Unix
グラフィック	(Win32 API)	QuickDraw	X Window
プログラム開発ソフト	Visual C++	CodeWarrior	GNU ツール等
フレームワーク	MFC	PowerPlant	OSF/Motif 等

## 1.5 グラフィクス

コンピュータを用いて美しいグラフィクスを表示することは、コンピュータプログラミングの醍醐味の一つと言ってよいと思うが、現状ではそのためのプログラミング環境が整備されているとは言いがたい。

C および C++ の特徴の一つとして、オペレーティングシステムやハードウェアに依存する部分は言語仕様とはせず、それぞれに応じたライブラリを用意するようになっている。特に、グラフィックについて言語レベルでは一切規定していないので、プログラムを作成する環境ごとに書き方が違ってしまっている。

Windows では Win32 API と呼ばれるシステム関数呼び出しのセット中にグラフィクスを扱う一連の関数が用意されており、それを用いてディスプレイ上に図形を表示する。Mac OS の場合は、OS の一部として QuickDraw と呼ばれるコンポーネントを持っており、それが図形描画を受け持っている。Unix においては、MIT で開発された X Window というグラフィックライブラリが図形表示のための標準となっている。これら異なる OS 用のグラフィックライブラリ間には、概念的には共通部分が多いにもかかわらず、コードレベルでは全く異なっており、グラフィックを用いるプログラムを作成する場合、異なる OS で同じプログラムが動くようにしようとすると、グラフィック部分のコードは全面的に書き直さないといけないことになる。

これでは全く不便だし，プログラム開発効率も悪いので，最近は OpenGL というグラフィックライブラリを中心とした，グラフィックプログラムの標準化が進められている．近い将来，OS に依存しない形でグラフィックを扱うプログラムを書くことができるようになるかも知れない．

## 2 極限集合描画アルゴリズム

さて，極限集合の描画アルゴリズムに話を移そう．OPTi が扱うのは，数学的には，一点穴空きトーラス群の擬等角変形というものであるが，ここでは詳しい定義等は述べない．興味ある読者には [1, 2, 3] を参照してもらうことにしよう．

以下の理解のために必要なのは，一点穴空きトーラス群の極限集合は球面上の Jordan 曲線であって，その内部にカスプと呼ばれる点が稠密に分布している，という事実である．

### 2.1 極限集合の記号表現

一点穴空きトーラスは， $(2, 2, 2, \infty)$  型オービフォールドの指数 2 の分岐被覆である．従って，一点穴空きトーラス群の極限集合は， $(2, 2, 2, \infty)$  オービフォールド群の極限集合と同じものであって，OPTi においては，取扱いのしやすさから， $(2, 2, 2, \infty)$  オービフォールド群を用いて内部計算を行っている．

Poincaré 円板  $D$  内で，境界  $S_\infty^1$  上に頂点を持つ 3 角形  $T$  を考える．3 角形  $T$  内部の一点から 3 辺に下ろした垂線の足を  $p, q, r$  とし，これら 3 点に関する双曲的な  $\pi$ -回転をそれぞれ  $P, Q, R$  とすると， $P, Q, R$  は (一つの) Fuchs 型  $(2, 2, 2, \infty)$  オービフォールド群を生成する (図 1) Fuchs 型の群においては無限円周  $S_\infty^1$  全体が極限集合になっていることに注意しておく．このときカスプとは，3 角形  $T$  の頂点に  $P, Q, R$  を有限回作用させて得られる点のことである．

$S_\infty^1$  を 3 角形  $T$  の頂点のところを 3 つの円弧に分割し， $P, Q, R$  の側にある部分をそれぞれ  $I_P, I_Q, I_R$  とする．このとき，以下のようにして， $S_\infty^1$  上の任意の点を  $P, Q, R$  の無限列として「表現」することができる．

$S_\infty^1$  内の任意の点  $x$  に対して，点列  $\{x_i\}$  と  $P, Q, R$  からなる記号列  $\{S_i\}$  を次で定義する．

$$\begin{cases} x_0 = x, \\ x_i \in I_{S_i}, \\ x_{i+1} = S_i(x_i). \end{cases}$$

各点  $x \in S_\infty^1$  に対して得られる記号列

$$S_0 S_1 S_2 \dots$$

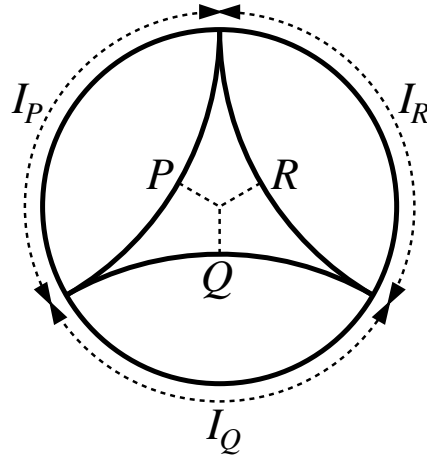


図 1:  $P, Q, R$  は  $(2, 2, 2, \infty)$  オービフォルド群を生成する

は条件  $S_i \neq S_{i+1} (\forall i)$  を満たしていて, いわば  $(2, 2, 2, \infty)$  オービフォルドの基本群

$$\langle P, Q, R \mid P^2 = Q^2 = R^2 = 1 \rangle$$

における無限語とも言えるものである. この記号列を  $x \in S_\infty^1$  の記号表現と呼ぶことにする.

この対応はカスプ点に対してちょうど 2 つの記号表現が対応することを除けば 1 対 1 である. カスプに関しては, たとえば  $x_n \in I_P \cap I_Q$  とすると次の 2 つの記号表現が対応する.

$$\begin{aligned} S_0 \dots S_{n-1} P R Q P R Q \dots \\ S_0 \dots S_{n-1} Q R P Q R P \dots \end{aligned}$$

この状況は, 実数の小数展開に似ている. 全ての実数に対して無限小数が対応するが, 有限小数 (カスプに対応) に対しては末尾が 0 の繰返し又は末尾が 9 の繰返しの 2 通りの小数展開が可能である.

さて, 上の対応の逆対応を次のように定義することができる. まず対応

$$P : I_P \longrightarrow I_Q \cup I_R$$

は拡大写像であることに注意する. すなわち

$$|P(x) - P(y)| \leq |x - y| \quad (\forall x, y \in I_P).$$

定義より

$$x = S_{n-1} \dots S_1 S_0(x)$$

であるから

$$x = S_0 S_1 \dots S_{n-1}(x_n), \quad x_n \in I_{S_n}$$

であるが、右辺の各  $S_i$  が縮小写像であることから、 $n$  が増えるに従って

$$S_0 S_1 \dots S_{n-1}(I_{S_n})$$

の長さは減少し、極限では 0 になる（数学的に厳密な説明ではない。）こうして記号表現

$$S_0 S_1 \dots S_n \dots$$

に対して一点  $x$  が定まる。

$S_\infty^1$  上の点  $x$  とその記号表現を同一視して

$$x = S_0 S_1 \dots S_n \dots$$

と書くことにすると、 $S = P, Q, R$  に対して

$$S(x) = S S_0 S_1 \dots S_n \dots$$

である。

## 2.2 極限集合の描画

前節では、Fuchs 群の場合に極限集合上の各点とその記号表現の対応を説明したが、実は、この対応は擬 Fuchs 群の場合でも成り立つ。このことを利用して、擬 Fuchs 型  $(2, 2, 2, \infty)$  オービフォールド群の極限集合を効率良く描くことができる。

コンピュータでは無限記号列を扱うことはできないので、有限列で近似することになる。すなわち、極限集合である Jordan 曲線を直接描くことはできないので、その中に稠密に分布するカスプを結んだ折れ線で近似して描くことになる。

描画のアルゴリズムは、次のようにすればよい。たとえば記号列

$$S_0 \dots S_{n-1} P$$

に対応する区間を処理するためには、その区間が記号列

$$S_0 \dots S_{n-1} P Q$$

$$S_0 \dots S_{n-1} P R$$

に対応する部分区間の和集合になっていることに注意すれば、それらの区間を別々に処理すればよいので、再帰呼び出しがうまく使えるわけである。処理すべき区間の両端点の距離が十分に小さい場合には（たとえば画面上のピクセルのサイズの 10 分の 1 以下）処理を打ち切りその 2 点を直線で結んでしまう。

以上が極限集合の描画の基本的な考え方であるが、OPTi においては、Poincaré 円板ではなく上半平面モデルからスタートするために、アルゴリズムに若干の修正が必要である。また、再帰呼び出し処理の打ち切り条件に関しても、ピクセルサイズとの比較のみではなく、表示されているウィンドウ内部か外部かといった条件についても細かい調整がされているが、それらについてはここでは省略する。

## 参考文献

- [1] H. Akiyoshi, M. Sakuma, M. Wada and Y. Yamashita, *Punctured torus groups and two-parabolic groups*, 数理解析研究所講究録 1065 “Analysis and Geometry of Hyperbolic Spaces” (1998), 61–73.
- [2] H. Akiyoshi, M. Sakuma, M. Wada and Y. Yamashita, *Ford domains of punctured torus groups and two-bridge knot groups*, 数理解析研究所講究録 1163 “双曲空間とその関連分野 II” (2000), 67–77.
- [3] M. Wada, OPTi における離散性判定アルゴリズム, プレプリント.