エレガントなフラクタル図形の描き方，その教え方

長　　慎　也†

プログラムは，短く簡潔に，しかも幅広い問題に応用できるように書けることが理想である．このような性質をもったプログラムをエレガントなプログラムと呼ぶことにする．本発表は，既存のプログラムをエレガントなものに書き換える実践例を示す．題材として再帰によるフラクタル図形の描画を選んだ．

既存のフラクタル図形の描画アルゴリズムには，その種類によってさまざま描画方式があり，描き方が統一されていない．具体的には，図の描画位置・向き・長さなどのパラメータをどのように指定するか，画面への描画と再帰呼び出しを行う順序をどうするか，などが図形により異なる．そこで，フラクタル図形を一定の流儀で描ける，すなわち，フラクタル図形をエレガントに描く方式を提供したい．

本発表では，タートルグラフィックスに基づく「親亀・子亀方式」を提案する．本方式を使って，樹木曲線，シェルピンスキーのギャスケット，コッホ曲線を描く実例を示す．

また，本方式が統一された描き方を提供していることを確認するために，プログラミングの初心者に向けた授業において，親亀・子亀方式を用いてフラクタル図形を描画する実習を実施した．

その結果，学習者は 樹木曲線の描き方の基本概念を応用して，シェルピンスキーのギャスケットを自ら描くことができた．また，コッホ曲線についても描き方を工夫させることで描くことができた．さらに，学習者が考えた独自のフラクタル図形を描くこともできた．

Drawing Fractal in an Elegant Way.

Shinya Cho†

1. はじめに

昨年夏のプログラミングシンポジウムにおいて,「エレガントなプログラムとは何か」という議論[1] が交わされた．その中で，簡潔で分かりやすい，一般化されていてより多くのデータ型に適応できる，再帰で書かれている，などの特徴をもったプログラムがエレガントである，という主張があった．また既存のプログラムをエレガントなものに書き換える例として，フラクタル図形の描画をよりエレガントにできないか，という問題が提起された．本発表は，その問題に答えるべく，フラクタル図形をエレガントに描く手法を提案する．

代表的なフラクタル図形としては樹木曲線，コッホ曲線，シェルピンスキーのギャスケット（以下「ギャスケット」），マンデルブロ集合などがある．このうち，マンデルブロ集合以外の図形については，再帰呼び出しを使って簡単に描けることが知られている．フラクタル図形をエレガントに描く具体的な戦略としては次のようなものがある．

(1) マンデルブロ集合を再帰呼び出しで描く手法を見つける

(2) 樹木曲線，ギャスケット，コッホ曲線などの各描画方法の違いを統一し，同じ方法で書けるプログラミング手法を見つける

(1) はまだ見つけるのが難しいと考え，その前段階として，本発表では (2) を試みた．

† 一橋大学
Hitotsubashi University

2. 既存のフラクタルの描画方法

既存のフラクタル図形の描画方式は，その種類によってさまざまであり，描き方が統一されていない．

2.1 描画関数の例

主なフラクタル図形を描画するため手続き（以降，描画関数と呼ぶ）の代表例をしめす．これらの描画関数を次の観点で分類する．

- 引数の与え方
  図の描画位置・向き・長さなどをどのように指定しているか．
- 描画と再帰の順序

描画関数はおおまかに「画面上に実際に線や点を描画する」という作業と「描画関数を再帰的に呼び出す」という作業に分かれるが，それらが描画関数の中でどのような順序で行われているか．

2.1.1 樹木曲線（Java）

まず，Java を使って樹木曲線を描く例を**図 1** に示す．このプログラム，および後述の 2.1.2，2.1.3 で示すプログラムは，あるプログラミングの授業[2] で用いられている資料である．

- 引数の与え方
  描画位置は x0,y0 長さは len 角度は ang
- 描画と再帰の順序
  ★ A において実際の描画を行ってから，★ B および★ C で再帰呼び出しを行う（描画→再帰）

2.1.2 ギャスケット（Java）

ギャスケットを Java で描く例を**図 2** に示す．

- 引数の与え方
  描画位置，長さは x[]，y[]（三角形の頂点） 角度は指定なし（回転しない）
- 描画と再帰の順序
  ★ A で実際の描画を行ってから，★ B 以降で再帰呼び出しを行う（描画→再帰）

2.1.3 コッホ曲線（Java）

コッホ曲線を Java で描く例 を**図 3** に示す．

- 引数の与え方
  長さは length（定数） 角度は angle．描画位置は引数で与えるのではなく，インスタンス変数 xOrig，yOrig で与える．
- 描画と再帰の順序
  まず★ B 以降の再帰呼び出しを行い，もっとも呼び出しの深いレベル，すなわち再帰を打ち切る★ A の時点で実際の描画が行われる．（再帰→描画）

2.1.4 コッホ曲線（ひまわり/ タートルグラフィックス）

タートルグラフィックスを用いたコッホ曲線の描画方法[3] を**図 4** に示す．このプログラムはひまわり[4] で書かれている．

- 引数の与え方
  描画位置と向きは タートルの位置と向き， 長さは「歩数」
- 描画と再帰の順序
  ★ B で再帰呼び出しを行った後，もっとも呼び出しの深いレベル，すなわち再帰を打ち切る★ A の時点で実際の描画が行われる．（再帰→描画）

2.2 既存手法の問題点

- **引数の仕様が図形により違う**
  前節のプログラムは，図形により引数の仕様が大きく異なっており，統一されたプログラムの書き方とは言い難い．
  - 図 1 の 樹木曲線は，長さ，向き，位置を引数で渡している．
  - 図 2 のギャスケットは三角形の位置を配列の引数を用いて渡している．
  - 図 3 の コッホ曲線（Java） は，描画位置をインスタンス変数で渡し，角度を引数，長さを定数で渡している．
  - 図 4 のコッホ曲線（ひまわり）は，描画位置，角度を亀の状態であらわし，長さを引数で渡している．

```
カメ蔵とは亀
線太さ=1
カメ蔵の X=100
カメ蔵を 90 度回す
コッホ曲線（4 で 400 の）
●コッホ曲線（レベルで歩数の）
  もしレベル<1 ならば
    カメ蔵を（歩数）歩進める      //★ A
    戻る
  //★ B
  コッホ曲線（レベル-1，歩数/3）
  カメ蔵を-60 度回す
  コッホ曲線（レベル-1，歩数/3）
  カメ蔵を 120 度回す
  コッホ曲線（レベル-1，歩数/3）
  カメ蔵を-60 度回す
  コッホ曲線（レベル-1，歩数/3）
```

図 4 ひまわりで書かれたコッホ曲線

- **座標計算が必要**
  Java で書かれたプログラムでは，線をひく場所などを計算で求める必要がある．三角関数などを使うのは煩雑であり，数学的な知識が少ない人には負担である．ひまわりのようなタートルグラフィックスを使うと，図形の描画位置および回転角度を亀の状態（位置，向き）に閉じ込めることができ，少し計算の手間が軽減されるが，それでも図形の大きさは，図 4 のように「歩数」などの引数で与えなければならない．そのため，プログラム中に「歩数/3 」などの，移動距離を計算する数式が出現する．これはまだ単純な式であるが，もっと複雑な図形になれば，式も複雑になる恐れがある．
- **「再帰→描画」は，プログラムの振る舞いを把握しにくい．**
  樹木曲線（図 1） や ギャスケット（図 2）のプログラムは「描画→再帰」という順序で描かれる．この順序では，描画関数が呼ばれるごとに，画面に必ず何らかの描画が行われるため，プログラムの振る舞い（今どの辺に図形が描かれているのかなど）が把握しやすい．これに対して コッホ曲線（図 3 , 図 4）は，再帰呼び出

しを打ち切る直前まで一切描画がなされないので，プログラムの振る舞いがわかりにくい．

3. 提案手法

上のような問題点を解決するためには，次のような要件を満たす必要がある．

- 描画関数へ渡す引数を統一する
- 座標計算を極力簡単にする
- 「描画と再帰の順序」を「描画→再帰」に統一する．

このような方針に則った描画方法 「親亀・子亀方式」 を提案する．親亀・子亀方式は，タートルグラフィックスに基づく描画方式であり，つぎのような特徴がある．

- 亀（タートル）は,「前進」命令や「回転」命令などの基本的な命令のほかに，自分自身の複製を生成することが可能である．ある亀が生成した亀を「子亀」，子亀を生成した亀を「親亀」と呼ぶことにする．
- 子亀は，親亀とまったく同じ位置，同じ向きで生成される
- 亀は「大きさ」をもつ．大きさは子亀が生成されるときに指定する．
- 大きさは，亀の移動距離に影響する．たとえば，2 匹の亀 A,B がいて，B は A の 1/2 の大きさであるとする．それぞれに同じ距離を指定して「前進」命令を実行させた場合，B の移動距離は，A の移動距離の 1/2 になる．
- 描画関数は，すべての亀で共有される．

親亀・子亀方式に則って作った描画関数は，次のような手順で構成される．

(1) 親亀が，図形の一部を画面に描く．
(2) 親亀が子亀を作り，適切な大きさ，向き，位置で配置する．
(3) 子亀に対して描画関数を呼び出す．
(4) 子亀がある一定の大きさより小さくなったら，描画関数を終了する．

親亀・子亀方式には次のような利点がある．

- **引数が不要**
  移動距離は亀の大きさに比例して変わるので，同じ描画関数を実行させても，亀の大きさに応じて図形全体が自然に拡大・縮小される．つまり，図形の描画に必要なパラメータ（位置，大きさ，角度）を，亀の状態（位置，大きさ，角度）に完全に閉じ込めている．
  また，再帰を打ち切るタイミングも亀の「大きさ」を利用して判定するため，再帰の呼び出し階層を渡す引数（図 1 や図 3 の `n`，図 2 の `times`，図 4 の「レベル」）も不要になる．
  このため，描画関数には与えるべき引数がなくなる．引数をなくすことで，引数の仕様は確実に統一され，図形による引数の仕様の違いに迷うことがない．
- **計算式が簡潔になる**
  亀の大きさが移動距離に比例するという性質を利用すると，通常のタートルグラフィックスで必要となる「描くべき図形の大きさに応じて移動距離を計算する」という作業が不要になり，プログラム中の式が簡潔になる．

これらの利点は，

- 描画関数へ渡す引数を統一する
- 座標計算を極力簡単にする

という要件を満たす．残りの

- 「描画と再帰の順序」を「描画→再帰」に統一する．

については，描画の手順を工夫して対応する．フラクタル図形を描く手順は一意ではないので,「描画→再帰」となるような描き方をあらかじめうまく選んでおけばよい．

4. 実　　例

親亀・子亀方式を使って，樹木曲線，ギャスケット，コッホ曲線を描く実例を示す．

実装にはオブジェクト指向言語ドリトル[5] を利用した．ドリトルは一般的なタートルグラフィックスの機能を標準の「タートル」オブジェクト☆ を用いて利用できる．また，任意のオブジェクトに対して「作る」というメソッドを実行すると，自分自身を複製して新しいオブジェクトを生成する．複製された新しいオブジェクトは，複製元の状態（位置，向きなど）がコピーされるほか，複製元に対して呼び出せるメソッドをすべて同じように呼び出すことができる．

親亀・子亀方式を実装するにあたり，ドリトルに次のような拡張を行った．

- それぞれのタートルに「大きさ」という状態を追加した．この大きさに比例して移動距離が変化する．画面上に表示される亀のグラフィックスの大きさも変化する．
- タートルオブジェクトに「子亀作る」というメソッドを追加した．これは,「作る」と同様に複製を作成するが，複製されるタートル（子亀）の大きさを引数で与えることができる．この値は，生成元のタートル（親亀）との大きさの比率である．

次に，プログラム例を示す．なお，これらのプログラムを Java アプレット☆☆として公開している．

4.1 樹木曲線

まず，樹木曲線を描くプログラムを考えてみる．

「樹木曲線を描く」メソッドを，次のように定義する．

(1) 親亀が幹を描く

☆ ドリトルはプロトタイプ指向であるので，クラスは存在しない．各機能は最初から用意されているオブジェクトに組み込んで提供する
☆☆ http://eplang.jp/cho/fractal/

(2) 大きさが 1/2 の子亀 A,B を生成する．親亀が描いた幹に対して，それぞれ 60 度右回転，60 度左回転させて配置する．
(3) それぞれの子亀に「樹木曲線を描く」を実行させる．
(4) 亀の大きさが一定以下になったら処理を打ち切る．

図 5 に，親亀と子亀 A,B が描く部分を示す．

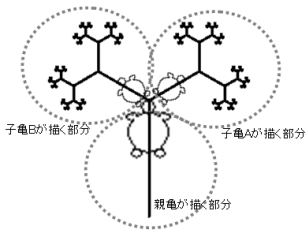

図 5 樹木曲線の描画

ドリトルで実装したプログラムを**図 6** に示す．

```
亀=タートル！作る.
亀！９０ 左回り.
亀：樹木=「
  自分！ 100 歩く.
  「(自分！大きさ?) > 0.05 」! なら 「
    子亀 A = 自分！ (1/2) 子亀作る.
    子亀 A ！60 右回り.
    子亀 A ！樹木.
    子亀 B = 自分！ (1/2) 子亀作る.
    子亀 B ！60 左回り.
    子亀 B ！樹木.
  」 実行.
」.
亀！樹木.
```

図 6 親亀・子亀方式で書かれた樹木曲線

なお，ドリトルの主な構文を**図 1** に示す．

4.2 ギャスケット

「ギャスケットを描く」メソッドは，次のように定義する．
(1) 親亀が三角形を描く．
(2) 各頂点に大きさ 1/2 の子亀 A,B,C を**図 7** のように配置する．
(3) 各子亀に,「ギャスケットを描く」を実行させる．
(4) 亀の大きさが一定以下になったら処理を打ち切る．

プログラムは、**図 8** のようになる．ここでは「描画→再帰」の順序を保つように,「三角形を描く」処理を先に行い，そのあと各頂点に子亀を置いて再帰呼び出しを行っている．

4.3 コッホ曲線

コッホ曲線は， **図 9** のように，線をひかずに頂点を打っていく描き方を採用した．

```
亀＝タートル！作る.
亀：三角形＝「
    自分！100 歩く 120 左回り.
    自分！100 歩く 120 左回り.
    自分！100 歩く 120 左回り.
」.
亀：ギャスケット＝「
  「(自分！大きさ？) ＞ 0.05」! なら「
    自分！三角形.
    子亀 A ＝自分！(1/2) 子亀作る.
    子亀 A ！ ギャスケット.
    自分！100 歩く 120 左回り.
    子亀 B ＝自分！(1/2) 子亀作る.
    子亀 B ！ ギャスケット.
    自分！100 歩く 120 左回り.
    子亀 C ＝自分！(1/2) 子亀作る.
    子亀 C ！ ギャスケット.
    自分！100 歩く 120 左回り.
  」実行.
」.
亀！ギャスケット.
```

図 8 親亀・子亀方式で書かれたギャスケット

この方法に基づいて「コッホ曲線を描く」メソッドを次のように定義する
(1) 親亀が現在位置に点を打つ
(2) **図 10** のように，それぞれ大きさ 1/3 の子亀 A～D を配置する
(3) それぞれの子亀に「コッホ曲線を描く」を命令する．
(4) 亀の大きさが一定以下になったら処理を打ち切る．

プログラムは**図 11** のようになる．

```
亀= タートル！作る.
亀：コッホ図形=「
  「(自分! 大きさ?) > 0.004 」! なら 「
    自分！ ペンなし 点を打つ.
    子亀 A ＝自分！ (1/3) 子亀作る.
    子亀 A！コッホ図形.
    自分！ 100 歩く 60 左回り.
    子亀 B ＝自分！ (1/3) 子亀作る.
    子亀 B！コッホ図形.
    自分！ 100 歩く 120 右回り.
    子亀 C ＝自分！ (1/3) 子亀作る.
    子亀 C！コッホ図形.
    自分！ 100 歩く 60 左回り.
    子亀 D ＝自分！ (1/3) 子亀作る.
    子亀 D！コッホ図形.
  」 実行.
」.
亀! コッホ図形.
```

図 11 親亀・子亀方式で書かれたコッホ曲線

5. 授業での実践

プログラミングの入門授業において，親亀・子亀方式を

表 1 ドリトルの主な構文

構文要素	文法	備考
インスタンス変数参照	**オブジェクト**：**変数名**	
メソッド定義	**オブジェクト**：**メソッド名**＝「**メソッドの実装**」.	実際は，インスタンス変数への手続きの代入
メソッド呼び出し	**オブジェクト**！**引数*** **メソッド名**	**引数**は数値定数または（）で囲まれた式
メソッド呼び出し（カスケード）	**オブジェクト**！**引数*** **メソッド名** **引数*** **メソッド名** …	メソッドの返却値（多くの場合**オブジェクト**自身）を用いて別のメソッド呼び出しを連続して行う
条件判断	「**条件**」！なら「**命令 1**」そうでなければ「**命令 2**」実行.	if (**条件**) { **命令 1** } else { **命令 2** } ’そうでなければ’ 以降は省略可能

使ってフラクタル図形を描く実践を行った．実践を通して，親亀・子亀方式がプログラミングの初心者でも理解できるほど簡潔で，多くのフラクタル図形に簡単に適用できることを示す．

5.1 授業概要

- **授業名** 獨協大学 コンピュータ入門 b
- **時期** 2008 年 9 月 29 日〜12 月 22 日（予定）
- **受講者数** 46 人
- **学年・学部** 1 年 経営学科
- **プログラミング経験** プログラミング経験は，ほとんどない．

この授業は，プログラミングの基本的な概念を習得することを目的としている．演習に使うプログラミング言語はドリトルである．その中で，タートルグラフィックスによる描画の延長としてフラクタル図形の描画を演習に取り入れた．

5.2 授業経過

授業内容を**表 2** に示す．第 2 回から第 5 回までは，フラクタル図形の描画に必要な予備知識を，授業に織り交ぜて説明した．

- 第 2 回では，タートルグラフィックスの基本を教えた．
- 第 3 回では，手続き抽象を教えた．「三角」「四角」「旗」などの命令を作らせ，亀の移動・回転を組み合わせて任意の位置と向きで図形が描けることを示した．
- 第 4 回では，子亀の生成について教えた．子亀の大きさをかえることで拡大縮小ができることを教えた．
- 第 5 回では，条件判断の構文を教えた．再帰を打ち切るための条件を書く際に必要になる．

実際にフラクタル図形の描画の演習を行ったのは第 6 回から第 8 回である．

第 6 回では，再帰の概念を教えた．まず，処理の順番を容易に追跡できる末尾再帰（再帰呼び出しが，手続きの最後に 1 回だけ発生する）を教えた．**図 12** に示した図形は，**図 13** で示されるプログラムによって描かれたものである．これを便宜上樹木曲線（末尾再帰版）と呼ぶ．

第 7 回では，樹木曲線☆の説明を行った．まず，樹木曲線を描くプログラム（ 図 6 ）を示した．次に，その動作を理解させるため，描き方をまず**図 14** のようなワークシートを用いて，紙の上で実際に描かせた．

その後，ギャスケット☆☆を描く演習を出題した．この演習では**図 15** で示したワークシートを示し，「図全体と相似になっている部分を見つけよ」という指示を行い，具体的な描画手順を各自考えさせ日本語で記入させた．

それだけでは正しい描画手順を導出できた学習者がいな

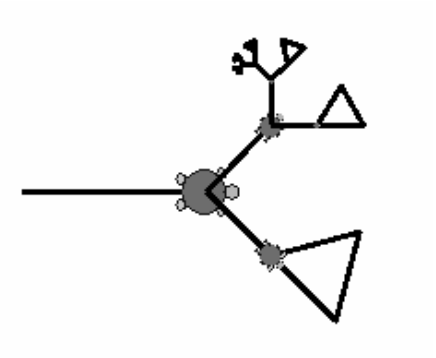

図 12 樹木曲線（末尾再帰版）

```
かめた=タートル！作る。
かめた：旗＝「
    自分！ １００ 歩く。
    自分！ １００ 歩く １２０ 左回り。
    自分！ １００ 歩く １２０ 左回り。
    自分！ １００ 歩く １２０ 左回り。
」 。
かめた：木＝「
  「(自分！大きさ？) ＞０．０１」！なら 「
    自分！100 歩く。
    子亀 A ＝ 自分！ 0.5 子亀作る。
    子亀 A ！ ４５ 右回り 旗。
    子亀 B ＝ 自分！ 0.5 子亀作る。
    子亀 B ！ ４５ 左回り 木。
  」実行。
」。
かめた！木。
```

図 13 樹木曲線（末尾再帰版）を描くプログラム

☆ 授業では単に「木」という名前で呼んだ

☆☆ 授業では「複合三角形」という名前で呼んだ

表 2 授業内容

回	内容
1	ガイダンス
2	オブジェクトの概念，タートルグラフィックス基礎
3	手続き抽象（メソッドの作成）
4	子亀の生成
5	条件判断
6	フラクタル図形（末尾再帰）
7	フラクタル図形（樹木曲線，ギャスケット）
8	フラクタル図形（コッホ曲線，C 曲線）
9-12	作品制作など

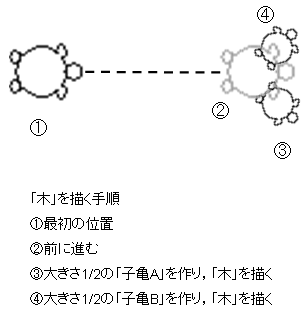

図 14 樹木曲線の描画手順書（兼ワークシート）

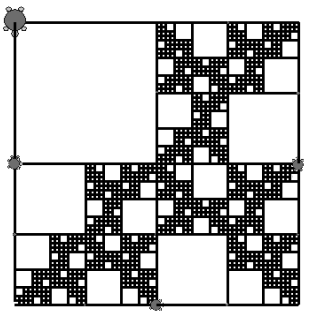

図 16 作品例

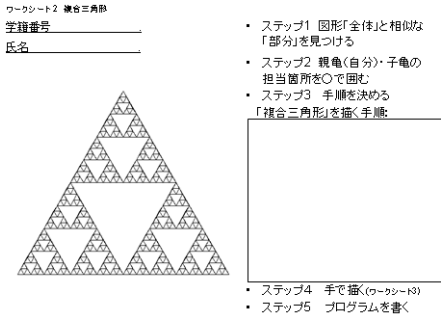

図 15 ギャスケットのワークシート

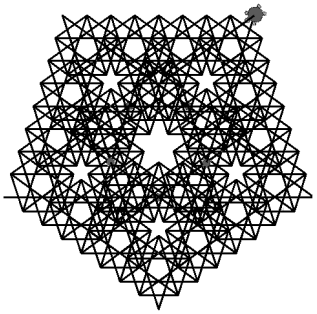

図 17 作品例

かったので，出題後約 15 分後にヒントとして「樹木曲線と違って，子亀を作る場所は一箇所にまとまっているわけではなく，親亀を移動させながら別々の場所に配置する必要がある」という旨のヒントを与えると，それからさらに 15 分後，プログラムを完成させた学習者が現れ始めた．

また，応用課題として，フラクタル図形を自由に描かせた．作品例を**図 16**，**図 17** に示す．

第 8 回では，コッホ曲線☆の描画を行った．描画のための手順を日本語で記述した描画手順書（**図 18** ）と，コッホ曲線を手で描くためワークシート（**図 19** ）を示し，まず手で描く演習をおこない，その後プログラムを作成させた．演習開始 10 分程度で，プログラムを完成させた学習者が現れ始めた．

また，時間がある人のための任意の課題で，C 曲線☆☆（**図 20** ） の描画を出題した．**図 21** のような図による説明のみで，描画手順は示さなかったが，3 人の学習者が正し

☆ 授業では「三角曲線」という名前で呼んだ

☆☆ 授業では「直角曲線」という名前で呼んだ

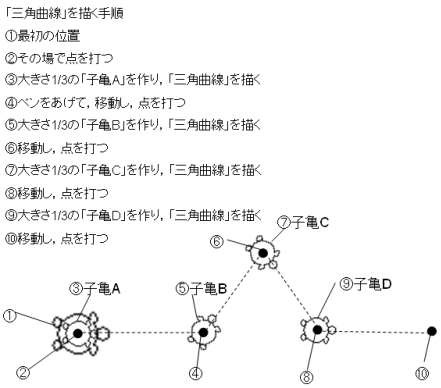

図 18　コッホ曲線の描画手順書

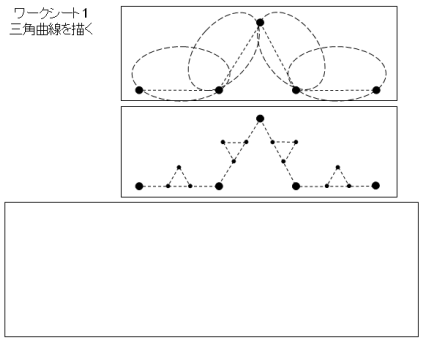

図 19　コッホ曲線のワークシート

い図形を描くプログラムを完成させた．

図 20　C 曲線

5.3　実践結果

授業での実践を通して，次のことが明らかになった．

- 親亀・子亀方式による 樹木曲線の描画手順とプログラムとを提示された学習者は，それを応用して， ギャスケットの描画手順を考え出し，プログラムを書くことができた．
- コッホ曲線の描画手順を日本語によって提示された学習者は，それをもとにプログラムを書くことができた．
- 一部の学習者は，コッホ曲線の描画手順を応用して， C 曲線の描画手順を考え出し，プログラムを書くことができた．

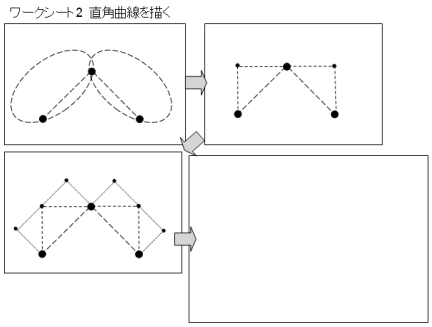

図 21　C 曲線のワークシート

これらのことから，学習者は樹木曲線と ギャスケットのプログラムを同じようなプログラムであると認識していたといえる．また，コッホ曲線のプログラムについては，日本語による描画手順が提示されたものの，描画手順には「ある程度のところで再帰を打ち切る」「亀をこの位置にこのような角度で置く」などの具体的な指示はなかった．それでも学習者は樹木曲線や ギャスケットで習得したプログラミングの手法を応用してプログラムを完成させることができた．つまり，コッホ曲線のプログラムも樹木曲線や ギャスケットと類似したプログラムであると認識したといえる．

また，C 曲線， 図 16 や 図 17 などの作品を自分で作成することができ，習得した描画手法を様々な種類のフラクタル図形の描画へと簡単に応用することもできた．

よって，親亀・子亀方式は，再帰呼び出しを使って描ける様々なフラクタル図形に対して一定の描き方を与える，エレガントな描画方法といえるだろう．

6.　議　　論

前節の実践において，

「親亀・子亀方式による樹木曲線の描き方を教えただけで，学習者はギャスケットや コッホ曲線も次々と描いてくれた」

となるのが理想であるが，ギャスケットや コッホ曲線を描く際には補足説明が必要であった．このことは，親亀・子亀方式に則った ギャスケットや コッホ曲線のプログラムには不自然な，エレガントでない部分もあることを示している．

ここでは，ギャスケットや コッホ曲線を描くための他のプログラムを示し，どのプログラムがエレガントであるかを議論したい．

6.1 「描画」と「再帰」を交互に繰り返す ギャスケット

授業において提出されたギャスケットのプログラムのうち，半数以上が**図 22** のようなプログラムであった．このプログラムは，図 8 と違い，描画関数の最初に「三角形を描く」という動作を行っていない．しかし，実際に描かれる図形は同じである．なぜなら，親亀が子亀を配置する過程で移動する軌跡が，三角形を描いたのと同じ軌跡になっているからである．

ワークシート（図 15）において各学習者が書いた日本語による描画手順には「三角形を描く」という動作を最初に行っているものはほとんどなく，いきなり「子亀を作り，ギャスケットを描く」という手順から始めていた．つまり「描画→再帰」という順番を厳密に守らなくても，再帰呼び出しの振る舞いを理解する上での支障はあまりなかったということができる．

```
亀＝タートル！作る.
亀：ギャスケット＝「
   「(自分！大きさ？) ＞ 0.05」！なら「
      子亀 A ＝自分！(1/2) 子亀作る.
      子亀 A ！ ギャスケット.
      自分！100 歩く 120 左回り.
      子亀 B ＝自分！(1/2) 子亀作る.
      子亀 B ！ ギャスケット.
      自分！100 歩く 120 左回り.
      子亀 C ＝自分！(1/2) 子亀作る.
      子亀 C ！ ギャスケット.
      自分！100 歩く 120 左回り.
   」実行.
」.
亀！ギャスケット.
```

図 22 ギャスケットを描く別のプログラム

6.2 線で描くコッホ曲線

点を使って「描画→再帰」の順序で描く方式の 図 11 と異なり,「再帰→描画」の順序でコッホ曲線を描くプログラムは**図 23** のようになる．この方式では，図形を線で描いているという点で，樹木曲線や ギャスケットと同じ描き方ということができる反面,「そうでなければ」という新しい構文を導入する必要がある点や，ワークシートを用いて手で描かせる演習がやりにくい（再帰呼び出しを打ち切る直前まで筆を動かせない）などの欠点がある．どちらの方法がよいのかは議論の分かれるところである．

```
亀＝ タートル！作る.
亀：コッホ図形＝「
   「(自分！ 大きさ?) ＞ 0.01 」！ なら 「
      自分！ペンなし.
      子亀 A ＝自分！ (1/3)  子亀作る.
      子亀 A！コッホ図形.
      自分！ １００ 歩く ６０ 左回り.
      子亀 B ＝自分！ (1/3)  子亀作る.
      子亀 B！コッホ図形.
      自分！100 歩く １２０ 右回り.
      子亀 C ＝自分！ (1/3)  子亀作る.
      子亀 C！コッホ図形.
      自分！ 100 あるく ６０ 左回り.
      子亀 D ＝自分！ (1/3)  子亀作る.
      子亀 D！コッホ図形.
   」 そうでなければ「自分！１００ 歩く」実行.
」.
亀！ コッホ図形.
```

図 23 コッホ曲線を描く別のプログラム

7. ま と め

フラクタル図形を，その種類によらず一定の流儀で描くことができるエレガントな描画方法「親亀・子亀方式」を提案した．プログラミングの初心者に対して行った演習では，学習者が親亀・子亀方式を多様な図形に適用できたことを示した．今後は，マンデルブロ集合など，さらに多くのフラクタル図形にも本手法が適用できるか，あるいはフラクタル図形以外の再帰を使って解く問題についても，エレガントなやりかたがないかどうか，などを考察していく．

参 考 文 献

1) 伊地知宏：エレガントなプログラムを求む. 情報処理学会 夏のプログラミングシンポジウム 2008.
2) 伊地知宏：計算機プログラミング I, http://lecture.ecc.u-tokyo.ac.jp/cichiji/cp-05/.
3) 日本語プログラミング言語なでしこ プログラム掲示板 サンプル：コッホ曲線 http://www.himanavi.net/cgi/nade-bbs/.
4) 酒徳峰章：ひまわり-日本語プログラミング言語, http://kujirahand.com/himawari/.
5) 兼宗進：プログラミング言語「ドリトル」, http://dolittle.eplang.jp/.

```
public void tree(Graphics g, int n, double x0, double y0, double len, double ang) {
    if (n <= 0) { return; }
    double x = len * Math.cos(radian * ang) + x0;
            // 枝の終点の x 座標を計算
    double y = len * Math.sin(radian * ang) + y0;
            // 枝の終点の y 座標を計算
    g.drawLine((int) x0, (int) (height - y0), (int) x, (int) (height - y)); // ★ A
            // 始点終点が与えられた枝の描画
    tree(g, n - 1, x, y, len * scale, ang - angle); // ★ B
            // 右側の枝の描画 (再帰呼び出し)
    tree(g, n - 1, x, y, len * scale, ang + angle); // ★ C
            // 左側の枝の描画 (再帰呼び出し)
}
```

図 1 Java で書かれた樹木曲線

```
 public void drawTriangle(Graphics g, int times, int x[], int y[]) {
     if (times < 1) { return; }    // 繰り返しが 1 未満なら終了
     int x1 [], y1 [], x2 [], y2 [];
     /* 各辺の中点を結んだ三角形の描画 */
     x1 = new int[]{(x[0] + x[1]) / 2, (x[1] + x[2]) / 2, (x[2] + x[0]) / 2};
     y1 = new int[]{(y[0] + y[1]) / 2, (y[1] + y[2]) / 2, (y[2] + y[0]) / 2};
     g.drawPolygon(x1, y1, 3); // ★ A
     /* 真ん中以外の三角形に対して繰り返し三角形を描画 ★ B */
     x2 = new int[]{x1[0], x[1], x1[1]};
     y2 = new int[]{y1[0], y[1], y1[1]};
     drawTriangle(g, times - 1, x2, y2);    // 三角形描画の再帰呼び出し
     x2 = new int[]{x1[2], x1[1], x[2]};
     y2 = new int[]{y1[2], y1[1], y[2]};
     drawTriangle(g, times - 1, x2, y2);    // 三角形描画の再帰呼び出し
     x2 = new int[]{x[0], x1[0], x1[2]};
     y2 = new int[]{y[0], y1[0], y1[2]};
     drawTriangle(g, times - 1, x2, y2);    // 三角形描画の再帰呼び出し
 }
```

図 2 Java で書かれたギャスケット

```
public void drawKoch(Graphics g, int n, double angle) {

     double x, y, angleR;

     if (n <= 0) {                                    // もう分割されない状態 (★ A)
         angleR = Math.PI / 180 * angle;              // 回転角度のラジアン変換
         x = length * Math.cos(angleR) + xOrig;       // 次の x 座標値
         y = length * Math.sin(angleR) + yOrig;       // 次の y 座標値
         g.drawLine((int) xOrig, (int) (200 - yOrig), (int) x, (int) (200 - y));
                                                      // 直線の描画
         xOrig = x;                                   // 次の点を基点に
         yOrig = y;                                   // 次の点を基点に
         return;
     }
     // ★ B
     drawKoch(g, n - 1, angle);         // コッホ図形描画の再帰呼び出し
     drawKoch(g, n - 1, angle + 60);    // コッホ図形描画の再帰呼び出し (60 度方向)
     drawKoch(g, n - 1, angle - 60);    // コッホ図形描画の再帰呼び出し (-60 度方向)
     drawKoch(g, n - 1, angle);         // コッホ図形描画の再帰呼び出し

 }
```

図 3 Java で書かれたコッホ曲線

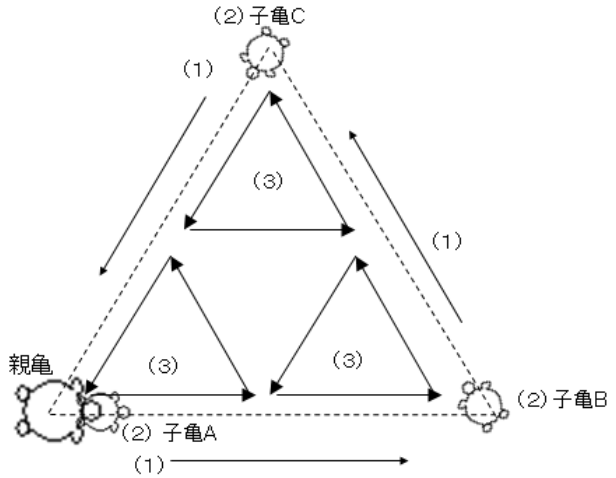

図 7　ギャスケットの描画

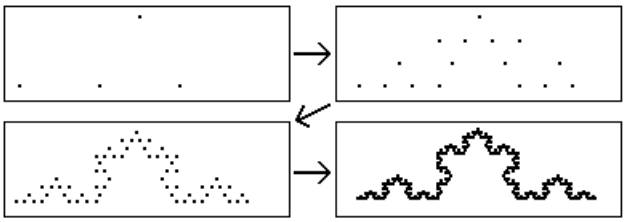

図 9　頂点を打つことによるコッホ曲線の描画方法

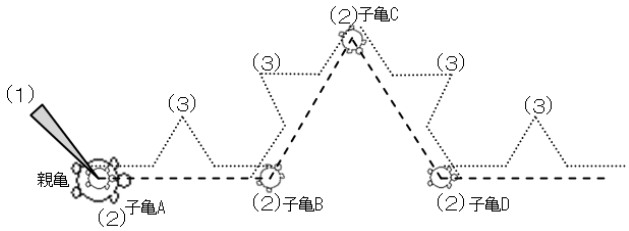

図 10　コッホ曲線の描画