Vol.2009-CE-99 No.3
2009/5/23

情報処理学会研究報告
IPSJ SIG Technical Report

文科系大学生を対象とした再帰プログラミングの学習

長　　慎　也†

プログラミングにおいて重要で，かつ理解が難しい概念として再帰呼び出しがある．本発表では，文科系の大学生を対象に，フラクタル図形の描画を題材とした再帰呼び出しの学習を行う授業の実践結果を報告する．簡潔で，多くのフラクタル図形に適用可能な描画方法を用いることで，学習者は再帰呼び出しの振る舞いを理解したり，自分でオリジナルの図形を作成したりすることができた．

Learning Recursive Programs for non-CS students

Shinya Cho†

This paper proposes a programming method of drawing fractals for novice students to learn concepts of recursive programming. Using the method, students can draw various fractals with little modifications. This method is applied to a classroom of introductory programming course. Students could easily understand the behavior of the programs and could create their own fractals.

1. はじめに

プログラミングの初心者にとって理解が難しい概念として，再帰呼び出しがある．これを効果的に教える題材としてフラクタル図形の描画が利用されている．フラクタル図形は，視覚的に実行結果がわかり，きれいな見た目をもち，種類も豊富であることから，学習者にとっても興味の持てる題材と言える．

2. フラクタル図形の既存の描画方法

文科系学生を対象としたプログラミングの入門授業において，フラクタル図形を用いて再帰呼び出しの概念を習得させる試みはすでに行われている[1)] が，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という問題点がある[2)]．このため，学習者が戸惑ったり，難しいと感じたりする場合が多い．

代表的なフラクタル図形である樹木曲線，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」）を描画するため Java プログラム（以降，描画関数と呼ぶ）の例をそれぞれ，**図 1**，**図 2**，**図 3** に示す．

これらの描画関数には次のような問題点がある．

- **計算式が複雑**
  線をひく場所の座標などを，四則演算や三角関数を利用して求める必要がある．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．
- **引数の仕様が統一されていない**
  各描画関数は，図形の種類によって引数の仕様に違いがある．
  - 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．

  これらの違いにより，学習者がプログラムを理解しにくくなったり，自分で新しいフラクタル図形を創作するときに，どの描画方式を使えばいいかわからなくなったりする可能性がある．
- **描画手順が統一されていない**.
  樹木曲線（図 1）は，★ 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．また，ギャスケット（図 2）のプログラムは★ 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．これら 2 つのプログラムは，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が把握しやすい．一方 コッホ曲線（図 3）は，★ 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．この順序では，再帰呼び出

† 一橋大学
Hitotsubashi University

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を１つ１つ追っていくことができず，理解しにくい．

3. 提 案 手 法

上のような問題点を解決する，フラクタル図形の描画方式として「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．

4. 実　　装

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[3] 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．

☆ 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

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．

4.1 樹 木 曲 線

「樹木曲線を描く」メソッドは，次のように定義する．

(1) 親亀が幹を描く
(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 4 に，親亀と子亀 A,B が描く部分を示す．

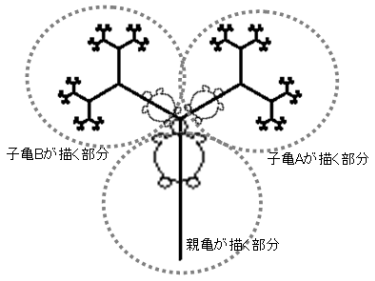


図 4　樹木曲線の描画

ドリトルで実装したプログラムを**図 5** 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 5　親亀・子亀方式で書かれた樹木曲線

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 6** 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 7** 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．

```
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 7 親亀・子亀方式で書かれたギャスケット

4.3 コッホ曲線

コッホ曲線は， **図 8** 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．

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 9** 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 10** のようになる．

4.4 親亀・子亀方式の利点

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`）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．

```
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 10 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
- **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

表 1 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

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 1** 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学習した．「歩く」や「右回り」の命令の組み合わせで絵を描く演習を行った．
- 第 3 回では，メソッドの概念を学習した．「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．例として，**図 11** のようなプログラムを提示した．これは，**図 12** のような図を表示する．

```
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！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 旗。 // ★ A
」。
かめた！木。
```

図 11 子亀を用いた，図形を描くプログラム

- 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．まず，**図 13** に示した図形を描くプログラム（**図 14**）を学習者に示した．これは図 12 のプログラムの ★ A の「旗」の部分を「木」に変更するだけで得られる．ただしこのプログラムを実行すると，図形は正しく描かれるが，再帰呼び出しを終了する条件がないため，永久にオブジェクトが生成される．これを防ぐために，**図 15** のようなプログラムにする必要があることを示した．

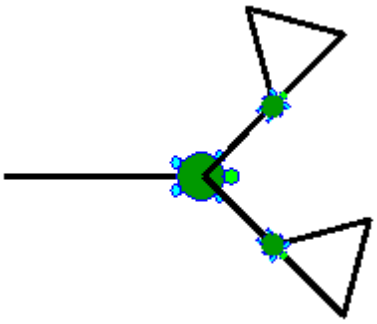


図 12 子亀を用いた図形の描画例

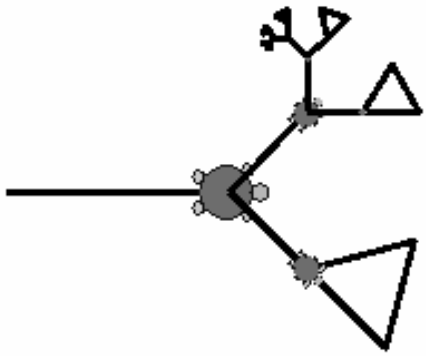


図 13 樹木曲線（末尾再帰版）

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 5 ）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 16** 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 17** 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

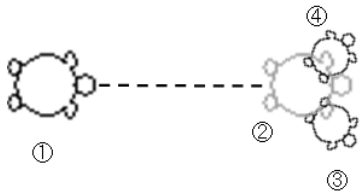
☆ 授業では単に「木」という名前で呼んだ
☆☆ 授業では「複合三角形」という名前で呼んだ

```
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！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。
」。
かめた！木。
```

図 14 樹木曲線（末尾再帰版）を描くプログラム（終了条件なし）

```
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 15 樹木曲線（末尾再帰版）を描くプログラム



「木」を描く手順
①最初の位置
②前に進む
③大きさ1/2の「子亀A」を作り，「木」を描く
④大きさ1/2の「子亀B」を作り，「木」を描く

図 16 樹木曲線の描画手順書（兼ワークシート）

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．

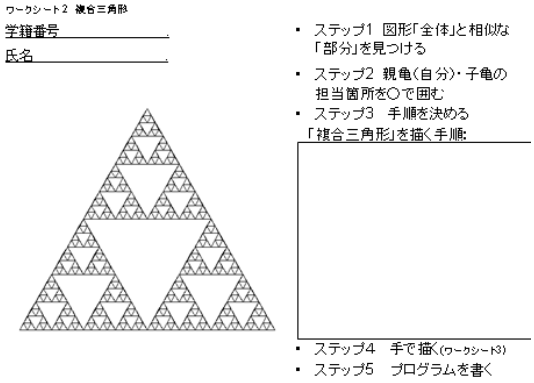


図 17 ギャスケットのワークシート

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 18** に示す．

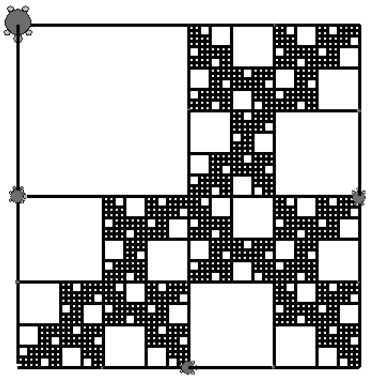


図 18 作品例

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 19** ）と，コッホ曲線を手で描くためワークシート（**図 20** ）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．

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．

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

☆ 授業では「三角曲線」という名前で呼んだ

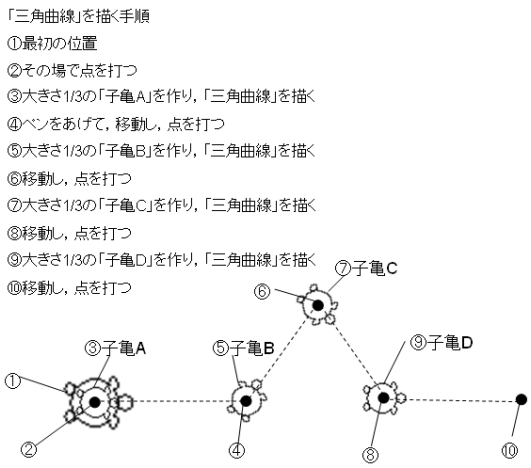


図 19 コッホ曲線の描画手順書

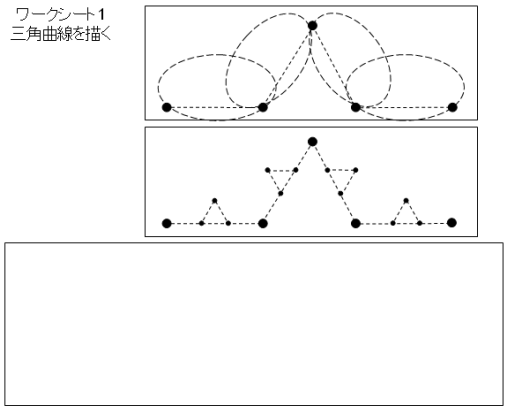


図 20 コッホ曲線のワークシート

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，学習者は独自の作品を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．

6. 考　　察

親亀・子亀方式を取り入れた実践においては，学習者がフラクタル図形の構造を理解したり，フラクタル図形をプログラムを用いて描いたりすることに関して一定の結果が得られた．「子亀」 という具体物を使って呼び出し階層の可視化を行ったことや，再帰を続けるか打ち切るかを「亀の大きさ」を使って判定したことにより，フラクタル図形の描画方法を直観的に説明できた．

反面，再帰呼び出しの概念を理解させるのには必ずしも十分ではなかったといえる．常に子亀を作り，それに対してメソッドを呼び出す方式を教えたため，学習者は「他のオブジェクトの他のメソッドを呼び出す」という感覚でプログラムを理解しており，「自分自身を呼び出してる」という意識は希薄であったといえる．また，「亀の大きさ」を使って再帰呼び出しの打ち切りを判定したため，再帰呼び出しの深さを意識させるには不十分であった．

一般的な再帰呼び出しでは，引数の値が重要な役割を担っている．引数を変えながら自分自身を呼び出すことによって，違う結果を得ることができるし，引数の値が特定の条件になったら再帰呼び出しを打ち切ることもできる．しかし，親亀・子亀方式ではあえて引数の概念を隠蔽しており，再帰呼び出しの導入として用いるのには適しているが，再帰呼び出しの本質を理解させることは難しい，本方式で学習をさせた後，引数を使った再帰呼び出しを教える段階にいかに繋げていくかが課題となる．

また，親亀・子亀方式を用いたプログラムにおいては，図形が一瞬にして描画されるため，再帰の振舞いを手続き的に理解するのが難しかった．ドリトルは，タイマー機能などを用いて，亀をゆっくり歩かせることも可能であるので，再帰呼び出しをアニメーションのように見せて，振舞いに対する理解を深めることも検討したい．

7. ま と め

再帰呼び出しの理解を深めるためのプログラミング手法として，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，本方式で教えた知識を利用して，より高度な再帰呼び出しの概念の理解へとつなげるための教材を開発していく．

参 考 文 献

1) 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/.
2) 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.
3) 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

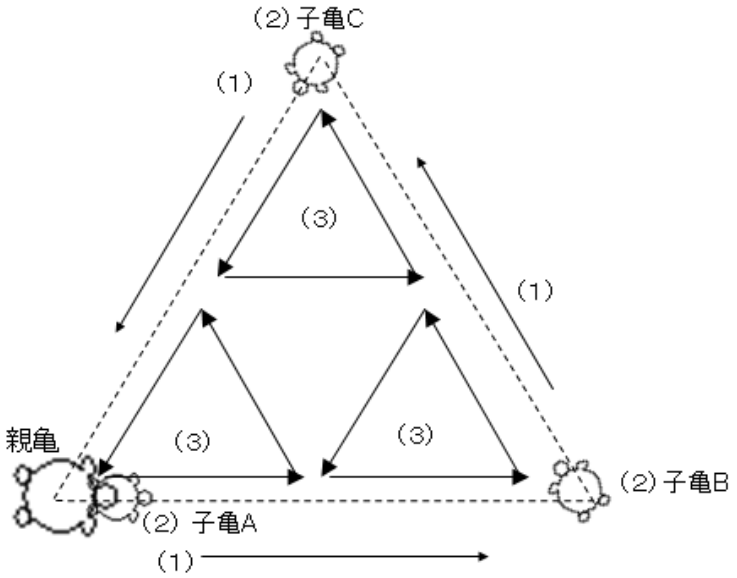


図 6 ギャスケットの描画

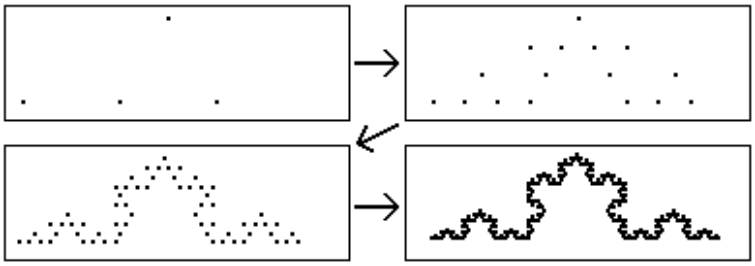


図 8 頂点を打つことによるコッホ曲線の描画方法

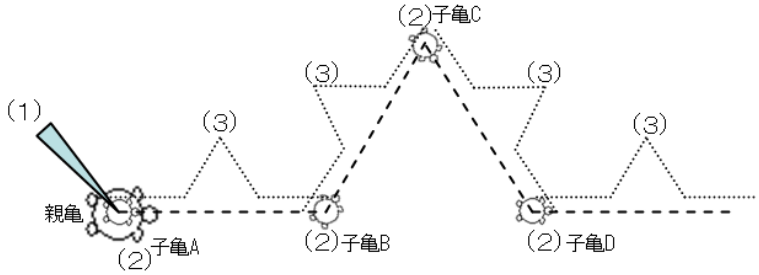


図 9 コッホ曲線の描画