フラクタルの概念に基づく提示情報量制御手法 Fractal View の Lisp プリンタへの応用

小池 英樹†

計算機ユーザは物理的に限られた大きさのディスプレイを通じて膨大な情報と対話しなければならない．この問題の1解決法として，著者らは以前フラクタルの概念に基づく提示情報量制御手法を提案した．本手法は対象とする情報構造の形に関係なくほぼ一定の情報量を提示することができる．本論文はこの提示量制御手法の Lisp プリンタへの応用について述べる．従来の一般的な Lisp プリンタは，S式の木構造において木の深さと各深さでの兄弟の数によって提示量を制御しているが，(1)対象S式の構造によって表示量が著しく異なる；(2)閾値の増減による提示量の変化が激しい；(3)2つの閾値の調節が困難；という問題点がある．一方，著者らの提案した手法を利用すると，非常に簡単なアルゴリズムによってこれら問題点を解決することができる．我々は，本手法を利用した Lisp プリンタを実現し，一般のプリンタとの比較によって，その有効性を示す．さらに本手法の特徴の1つである，ユーザの着目点近傍の詳細と周辺部の概略の統合表示を利用した，着目点指向プリンタへと拡張し，その表示例をも示す．

An Application of a Fractal-based Method for Information Display Control to a Lisp Printer

Hideki Koike†

Computer users must interact with large amounts of information through video displays which are physically limited in size. To minimize this problem, the authors had proposed a fractal-based method for information display control, which can keep the total amount of displayed information nearly constant without relation to the structure of information. This paper described an application of this method to the Lisp printer. In general, Lisp printers control the display of S-expressions by focusing on the depth of the S-expression and the number of siblings at each depth. However, with this method: (1) the total amount changes considerably corresponding to the target S-expression; (2) it also changes considerably when each threshold is incremented or decremented; (3) it is hard to hundle two thresholds. We implemented a Lisp printer which used our method, and showed its ability through the comparative experiment to the normal printer. We, moreover, extended this printer to a focus-oriented printer which used another feature of our method, the integration of details and contexts.

1. はじめに

情報システムは社会のインフラストラクチャとして重要な役割を果たしている．そして計算機ハード／ソフト両面での発達は，膨大な情報量の操作を可能としている．しかしながら，計算機から人間への情報の伝達は主に計算機ディスプレイで行われる．したがって，計算機ユーザはその大きさに物理的な制約のあるディスプレイを通じて，この膨大な量の情報と対話しなければならない．

このいわゆる“small screen problem[8]”は，多くのアプリケーション・ソフトウェアにおいて大きな問題となっている．これに対する解決法としては：(1)大画面を使う方法；(2)必要な情報だけを表示する手法；がある．(1)で物理的大画面を使用するものとしては MIT の DataLand[4] が有名である．DataLand はプロジェクトを用いて壁一面を出力領域とし，各種情報の表示を行った．頭部搭載型ディスプレイを用いた仮想現実感研究[5]は，ユーザの全周囲の仮想的な大画面を使用できる．しかしこの両者ともに，表示する要素がさらに増加した時には，同じ問題に直面することになる．古典的なものとしては，画面スクロールがある．実際には画面に収まりきらない対象を，物理的に制限された画面上で上下（あるいは左右）に移動させる技術である．しかし，本論文で示すようなプログ

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, University of Electro-Communications

ラム表示など，ある種のアプリケーションにとっては，ユーザの着目点と同時に周辺の重要な部分が表示されることが望ましい．

(2)としては例えば,Generalized Fisheye Views[6]がある．対象の構文情報を利用し，ユーザの着目点近傍は詳細に表示しつつ周辺部は重要な目印だけを表示することができる．ただし，この手法の問題は提示情報量を制御することができなかった点である．

これに対し著者らはフラクタル[12]の概念を，木として表現できる情報構造に応用した提示情報量制御手法を提案し[11]，これを Fractal View[10] と呼んだ．フラクタルの概念を形を持たない情報構造に適用することで，焦点近傍の詳細と周辺部の概略を統合した表示を可能とすると同時に，対象とする情報構造の形の相違による表示量の差を少なくすることに成功している．

本論文は Fractal View を応用した Lisp プリンタについて述べている．本論文の目的は，文献 11) で定式化した手法を利用した実際のシステムを実現し，その有効性を示すことである．必ずしも本格的な Lisp プリンタを作成することは意図しなかったが，実現されたプリンタは十分実用に耐え得るものであり，幾つかの点において既存のプリンタより明らかに優れている．次章ではまず Fractal View について説明する．3章では既存の Lisp プリンタの問題点を述べ，我々の提案する手法の Lisp プリンタへの応用について述べる．4章では，実際に作成したプリンタの出力例を示し，一般のプリンタとの比較を行う．5章ではこれを拡張して実現した着目点指向プリンタについて述べる．6章では本手法の問題点等に関する考察を行う．7章は結論である．

2. Fractal View

周知のとおりフラクタルとは B. Mandelbrot の造語で，広い意味での自己相似なものを指す．例えばフラクタル図形の代表としてしばしば参照されるコッホ曲線は，厳密な意味での自己相似性を有する．一方，海岸線，木，雲などの自然物の形は必ずしも正確に自己相似ではないが，統計的な意味において自己相似性を有し，やはりフラクタルであることがわかっている[12]．数学的にはフラクタルは図形の複雑さの尺度であるが，物理ではさまざまな物理現象のモデル化[16]，工学では画像解析[9]やコンピュータ・グラフィックスを利用した画像生成[2]に利用されている．

著者らの提案した提示情報量制御手法 Fractal View は，フラクタルの概念を情報構造に応用したものである．コッホ曲線に代表されるフラクタル図形において，その“真の”フラクタル性は無限状態においてのみ実現する．言い換えると，真のコッホ曲線とは無限回の再帰呼びだしで描画されるものである．この複雑な図形は，必要に応じた縮尺（スケール）によって近似され利用されている．Fractal View はこのスケール変更による複雑な対象の近似メカニズムに着目し，情報構造を“複雑な対象”と捉えることでこれを抽象化表示する手法である．

具体的には以下のとおりである．論理的な木の各節点 x に，ある概念的な重み Fv_x を仮定する．次に着目節点を新しいルートとして以下の式にしたがって値の伝播を行う．

$$\begin{cases} Fv_{root} = 1 \\ Fv_{child\_of\_x} = r_x \times Fv_x \end{cases} \quad (1)$$

ただし，

$$r_x = CN_x^{-1/D}$$

ここで N_x は節点 x での分岐数，D は適当なフラクタル次元を表す定数，C は $0<C\leq 1$ の定数である．**図1**は値の伝播例を示している．この状態である閾値を設定し，その閾値以上の値を持っている節点は表示し，それ以外は消去する．本手法の特徴は：

- 焦点近傍の詳細と周辺部の概略表示ができること；
- 閾値以上の値を持つ節点数は木の形に関係なくほぼ同程度となること；
- 閾値は統計的な意味において連続的に設定できること；

の3点である．定数 C, D の意味など詳細に関しては文献 11) を参照されたい．

文献 11) では，プログラムの段下げが構成する木構造に対して本手法を適用した場合の例について議論を行った．次章では実際に本手法を応用して実現した Lisp プリンタについて述べる．

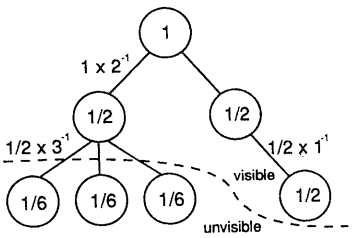

図 1 フラクタル値の伝播
Fig. 1 The propagation of fractal values.

3. Lisp プリンタへの応用

3.1 一般のプリンタとその問題点

Lisp ではS式を出力する際に細部の省略表示，いわゆる holophrasting[18] がよく用いられる．この省略表示の目的は：(1)出力が画面全体から溢れないこと；(2)不必要な情報を省略して着目点に集中しやすくすること；である．

この省略表示には，S式の木構造に着目した方法が用いられる．例えば (a (b c) d) というS式は図2のような木として表現できる．このとき木の深さと，同じ深さでの兄弟節点の数に着目する方法である．Common Lisp[15] の場合，*print-level*, *print-length* という2つの大域変数がS式の表示量を制御する．前者は入れ子になったオブジェクトを印字する深さ，つまり木の深さを制御し，後者はある深さにある要素の印字個数，つまり各々の深さで表示する要素の数を制御する．このS式の深さと長さに着目した表示制御は，一般の Lisp プリンタで広く採用されている．

しかし，この制御手法には次に述べる幾つかの問題点がある．以下の例では，一方はネストが深く2分木ととらえられるS式，

((((1 2)(3 4))((5 6)(7 8)))(((1 2)(3 4))((5 6)(7 8))))

他方は，ネストが浅い単純なS式，

(1 2 3 4 5)

を考え，前者はシンボル nested，後者はシンボル simple にそれぞれ束縛されているものとする．

第1の問題点は，対象S式の構造によって提示量が大きく異なることである．いま初期状態で *print-level* を4，*print-length* を2に設定すると，各々の出力は以下のようになる．ただし，“>” は Common Lisp のプロンプトである．

```
>nested
((((1 2)(3 4))((5 6)(7 8)))(((1 2)(3 4))((5 6)(7 8))))
> simple
(1 2 …)
```

ネストの深いリストの表示では全要素が表示されるにもかかわらず，単純なリストを出力する場合にはわずか2個の要素しか表示されない．

この問題は *print-length* を5に設定することで解決できるように思われる．これは局所的には成功するが，大局的には他の副作用を生じる．例えば，5分木として表現できる深さ3のS式を考えると，これを表示しようとすると合計125個のアトムが表示されることになる．つまり，*print-length* を大きくすると，S式での表示量はその分岐数にしたがい指数関数的に増大する．

第2の問題点は，提示量が各閾値の増減によって，極端に変わってしまうことである．例えば nested を出力する場合，*print-level* を4のままで *print-length* を2から1に変更すると，

```
> nested
((((1 …) …) …) …)
```

と，表示は1要素となってしまう．同様に *print-length* を2に固定し，*print-level* を4から3に変更すると，

```
> nested
(((# #) (# #))((# #) (# #)))
```

と，アトムの表示数は0になる．

第3の問題点はユーザが操作すべき閾値が2つ存在することである．ユーザは適当な表示を得るために，これら2つの閾値を調節しなければならない．そして対象とするリストが変わる度にユーザは2つの閾値を変更しなければならない．しかも問題点の1で述べたように，この局所的に調節された閾値は，他のS式では表示量が過多，あるいは過小だったりする．

まとめると，一般の Lisp プリンタの問題点は：

1. 同一閾値条件で異なるS式での表示量差が大きい；
2. 閾値を1段階増減したときの表示量変化が大きい；
3. 2つの閾値の調節が難しい；

ことである．

3.2 Fractal View プリンタの実現

S式は木として捉えられるので，S式の省略表示に Fractal View を応用することができる．**図2**において，ルートを焦点として式1にしたがい各節点の重みを計算する．C, D は1とする．まずルートの重みは1である．次にルートは3つの枝別れを持つので，各子節点の重みは，

$$3^{-1/1}\times 1=\frac{1}{3}$$

と決定できる．さらに2番目の子供は2つの枝別れを持つので，その子供の重みは，

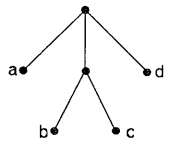

図2 S式 (a (b c) d) を表す木
Fig. 2 A tree which represents (a (b c) d).

$$2^{-1/1}\times\frac{1}{3}=\frac{1}{6}$$

と決まる．そして出力時には，ある節点の持つ値が閾値未満の場合，アトムは '?'，コンスは '%' として省略表示する．**図3**に Fractal View を応用したプリンタのアルゴリズムを示す．これからわかるように，計算時間は対象とするオブジェクト全体の量ではなく，表示する量に依存するため，大きなS式の場合でも十分短い時間で表示できる．

図2の例では閾値が3つ (1, 1/3, 1/6) しかなく，閾

```
procedure FRACTAL_PRINT (x : S式; scale : real);
begin
    if scale < threshold
      if x がアトム
        ?を表示
      else
        %を表示
    else
      if x がアトム
        x を表示
      else begin
        nelem := x の長さ
        newscale := scale × nelem^-1/1
        for x の各要素 i do
            FRACTAL_PRINT(i, newscale)
      end;
end;
```

図 3 Fractal View プリンタのアルゴリズム
Fig. 3 An algorithm of the Fractal View Printer.

```
(define-machine fib
  (registers n val continue)
  (controller
     (assign continue fib-done)
   fib-loop
     (branch (< (fetch n) 2) immediate-answer)
     (save continue)
     (assign continue afterfib-n-1)
     (save n)
     (assign n (- (fetch n) 1))
     (goto fib-loop)
   afterfib-n-1
     (restore n)
     (restore continue)
     (assign n (- (fetch n) 2))
     (save continue)
     (assign continue afterfib-n-2)
     (save val)
     (goto fib-loop)
   afterfib-n-2
     (assign n (fetch val))
     (restore val)
     (restore continue)
     (assign val
           (+ (fetch val)(fetch n)))
     (goto (fetch continue))
   immediate-answer
     (assign val (fetch n))
     (goto (fetch continue))
   fib-done))
```

図 4 分岐の多いS式
Fig. 4 S-expression which has many branches.

```
(define (+terms L1 L2)
  (cond ((empty-termlist? L1) L2)
        ((empty-termlist? L2) L1)
        (else
         (let ((t1 (first-term L1))
               (t2 (first-term L2)))
           (cond ((> (order t1) (order t2))
                  (adjoin-term
                   t1
                   (+terms (rest-terms L1) L2)))
                 ((< (order t1) (order t2))
                  (adjoin-term
                   t2
                   (+terms L1 (rest-terms L2))))
                 (else
                  (adjoin-term
                   (make-term (order t1)
                              (add (coeff t1)
                                   (coeff t2)))
                   (+terms (rest-terms L1)
                           (rest-terms L2)))))))))
```

図 5 ネストの深いS式
Fig. 5 S-expression which is deeply nested.

```
1 ]=> (set! print-length 27)

;Value: 27

1 ]=> (set! print-level 3)

;Value: 3

1 ]=> (normal-pp branched-list)          ; (A)

(define-machine
 fib
 (registers n val continue)
 (controller   % fib-loop
  %            % %
  %            % %
  %            % %
  afterfib-n-1 % %
  %            % %
  %            % afterfib-n-2
  %            % %
  %            % immediate-answer
  %            % fib-done))
;No value

1 ]=> (normal-pp nested-list)            ; (B)

(define (+terms l1 l2) (cond % % %))
;No value

1 ]=> (set! print-level 4)

;Value: 3

1 ]=> (normal-pp nested-list)            ; (C)

(define
 (+terms l1 l2)
 (cond (% l2) (% l1) (else %)))
;No value

1 ]=> (normal-pp branched-list)          ; (D)

(define-machine
 fib
 (registers n val continue)
 (controller
  (assign continue fib-done)
  fib-loop
  (branch % immediate-answer)
  (save continue)
  (assign continue afterfib-n-1)
  (save n)
  (assign n %)
  (goto fib-loop)
  afterfib-n-1
  (restore n)
  (restore continue)
  (assign n %)
  (save continue)
  (assign continue afterfib-n-2)
  (save val)
  (goto fib-loop)
  afterfib-n-2
  (assign n %)
  (restore val)
  (restore continue)
  (assign val %)
  (goto %)
  immediate-answer
  (assign val %)
  (goto %)
  fib-done))
;No value
```

図 6 一般のプリンタでの表示例
Fig. 6 Session with the normal Lisp Printer.

値設定の柔軟性に欠けているようにみえるが，より大きなS式や，いろいろな形の木を想定すると，統計的には閾値は多くなる．そして最大の特徴は対象がいかなるS式でもある閾値以上の値は統計的にほぼ同程度となる点[11]である．

Fractal View を利用したプリンタは，現在 MIT Scheme 7.2α[7]，Common Lisp 上にそれぞれ実現されている．実際には，S式は図3を利用したフィルタで細部省略された後，プリティプリンタによって出力される．次章以降の例では Scheme 上に実現されたプリンタを用い，同様に実現した一般的な省略手法を利用したプリンタとの比較を行う．

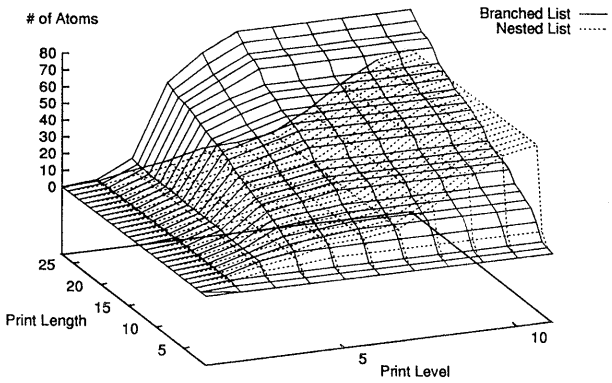

図 7 一般のプリンタを使用した場合，枝分かれの多いリスト (BranchedList) とネストの深いリスト (NestedList) における，print-length, print-level, 表示アトム数の関係
Fig. 7 The relation between print-length, print-level, and the number of displayed atoms in the branched list and the nested list when using normal Lisp Printer.

4. 実 行 例

出力の対象としては，文献 1) から**図4**と**図5**に示す2つのリストを用いる．前者は分岐の多いリストの代表で，シンボル branched-list に束縛され，後者はネストの深いリストの代表で，シンボル nested-list に束縛されている．

まず**図6**は一般のプリンタによるセッション例である．図中，"1]=>" は Scheme のプロンプトである．normal-pp がプリンタで，exp を出力するには (normal-pp exp) として呼び出される．表示は大域変数 print-length と print-level によって制御され，初期値はそれぞれ 27 と 3 である☆．この状態で各S式を表示すると図6 (A)，(B)のようになる．branched-list に比べ nested-list の表示量が少ない．そこで print-level を1だけ増して4にすると，nested-list の表示は多少増加するが(C)，最初の branched-list に比較してまだ表示量は少ない．ところがこの閾値変更により，branched-list はほとんどの要素が表示されてしまう(D)．一般のプリンタの場合，この中間表示はできないことに注意されたい．また print-length の減少は，本来対等であるべき各ラベルに差をつけることになり好ましくない．**図7**は branched-list, nested-list それぞれにおける print-level と print-length と表示アトム数の関係を示したグラフである☆☆．nested-list は print-level 一定の時，print-length の小さい範囲で敏感に反応する一方，branched-list は print-length 一定の時，print-level の小さい範囲で増加する特徴を持つ．

一方，**図8**は Fractal View プリンタによるセッション例である．fractal-pp がプリンタであり，表示は変数 fractal-threshold で制御される．branched-list の初期表示 (A) を一般のプリンタの場合 (図6 (A)) に近づけるため，閾値 fractal-threshold は 0.003 に設定されている．この時，ネストの深いリストは図8 (B)のように表示される．一般のプリンタに比べてその表示数は明らかに多く，両リストでの表示量の差が小さいことがわかる．

さらに Fractal View プリンタの場合は，提示量を徐々に変化させることが可能である．閾値を 0.0017 に変更することで，図8 (C)，(D) のように nested-list, branched-list ともに微妙な表示の増加ができる．もちろん，閾値をさらに小さくして，branched-list の表示量を増加させることは簡単である．**図9**は fractal-threshold と表示アトム数の関係を示したグラフである．fractal-threshold が 0.03 程度までは両者での表示量差はほとんどない．その後，両者の差が大きくなりだすが，一般のプリンタに比べると差の広がり方が小さい．

☆ branched-list はフィボナッチ数列を計算するための抽象機械を表現したもの[11]で，同じレベルに出現する fib-loop, afterfib-n-1, afterfib-n-2, immediate-answer, fib-done は各々ラベルの役割を果たしている．このラベルを同等に扱うために各初期値はこのように設定された．

☆☆ 実際には各計測点間は連続的に増加するのではなく階段状となる．

5. 着目点指向プリンタ

前章で示した Lisp プリンタの発展として，着目点指向プリンタ[18]が考えられる．Lisp のプログラミングで大きなS式を扱う場合，着目点近傍を詳細に表示し，他を概略表示するのが便利である．前章の例では，焦点を常にルートにおいていたが，これを移動できるようにすることで，Fractal View のもう1つの特徴である，着目点近傍と周辺部の統合表示が利用できる．

我々は Fractal View プリンタを拡張し，着目点指向プリンタを試作した．このプリンタはS式を引数として受けると，内部的に実際に木構造を作る．次のS式を表示するまでこの木構造は維持される．そして，焦点が移動すると式1に基づき各節点へ値の伝播を行う．現在，print, up, down, next, previous の4コマンドがあり，それぞれS式の表示，着目点の親への移動，長男への移動，次の兄弟への移動，前の兄弟への移動を行う．

図 11 は着目点指向プリンタの実行例である．表示例としてはやはり文献1) から**図 10** に示すプログラムを使用した．最初焦点はルートにおかれている．この状態で print コマンドを入力すると，入れ子になった関数 sqrt の詳細が省略された表示が得られる(A)．関数名と引数並び (sqrt x) と，関数 sqrt の本体 (sqrt-iter 1 x) は表示されるが，3つの内部関数は，その存在を示すにとどまり，詳細は省略表示されている．

次に down で define に焦点を移動した後，2つの next で焦点を内部関数 good-enough? に移動し表示する．今度は着目している関数が詳細に表示され，他の関数定義の部分は省略される(B)．さらに next コマンドで次の関数 improve の定義に移動すると，この関数を詳細に表示することができる(C)．

このように Fractal View を利用すると着目点近傍を詳細に表示し，周辺部はその概略だけを表示することができる．しかも，焦点が移動した場合の表示量変化を少なくすることができる．こうした表示は一般のプリンタでの省略手法では実現が困難であろう．

6. 考　　察

本手法を Lisp プリンタに適用した時の最大の問題点は，関数やマクロ定義を表示するときにその関数名やマクロ名の表示を保証しないことである．Common Lisp の場合には，*print-length* を2以上に設定しておけば，defun に続く関数名は常に表示される．一方，本手法の場合，関数名と同じレベルにある要素が多いと，これが省略されてしまうことがある．

ただし，これは従来のプリンタの優位性を支持するものではない．なぜならば，S式によっては *print-length* を2以上に設定することで，表示領域から溢れてしまうことが考えられるからである．これを許すのであれば，Fractal View プリンタにおいて図3を拡張し，関数名等は常に表示するようにすることもできる．

本論文の主目的は実用的な Lisp プリンタの実現法を論じることではなく，Fractal View の提示量制御

```
1 ]=> (set! fractal-threshold 0.003)

;Value: .01

1 ]=> (fractal-pp branched-list)                    ; (A)

(define-machine
 fib
 (registers n val continue)
 (controller   (? ? ?) fib-loop
  (? % ?)      (? ?)   (? ? ?)
  (? ?)        (? ? %) (? ?)
  afterfib-n-1 (? ?)   (? ?)
  (? ? %)      (? ?)   (? ? ?)
  (? ?)        (? ?)   afterfib-n-2
  (? ? %)      (? ?)   (? ?)
  (? ? %)      (? %)   immediate-answer
  (? ? %)      (? %)   fib-done))
;No value

1 ]=> (fractal-pp nested-list)                      ; (B)

(define
 (+terms l1 l2)
 (cond
  ((empty-termlist? l1) l2)
  ((empty-termlist? l2) l1)
  (else (let ((? %) (? %)) (? % % %)))))
;No value

1 ]=> (set! fractal-threshold 0.0017)

;Value: .003

1 ]=> (fractal-pp nested-list)                      ; (C)

(define
 (+terms l1 l2)
 (cond
  ((empty-termlist? l1) l2)
  ((empty-termlist? l2) l1)
  (else
   (let
    ((t1 (? ?)) (t2 (? ?)))
    (cond (% %) (% %) (? %))))))
;No value

1 ]=> (fractal-pp branched-list)                    ; (D)

(define-machine
 fib
 (registers n val continue)
 (controller   (? ? ?)          fib-loop
  (? % ?)      (save continue) (? ? ?)
  (save n)     (? ? %)         (goto fib-loop)
  afterfib-n-1 (restore n)     (restore continue)
  (? ? %)      (save continue) (? ? ?)
  (save val)   (goto fib-loop) afterfib-n-2
  (? ? %)      (restore val)   (restore continue)
  (? ? %)      (goto (? ?))    immediate-answer
  (? ? %)      (goto (? ?))    fib-done))
;No value
```

図 8 Fractal View プリンタでの表示例
Fig. 8 Session with the Fractal View Printer.

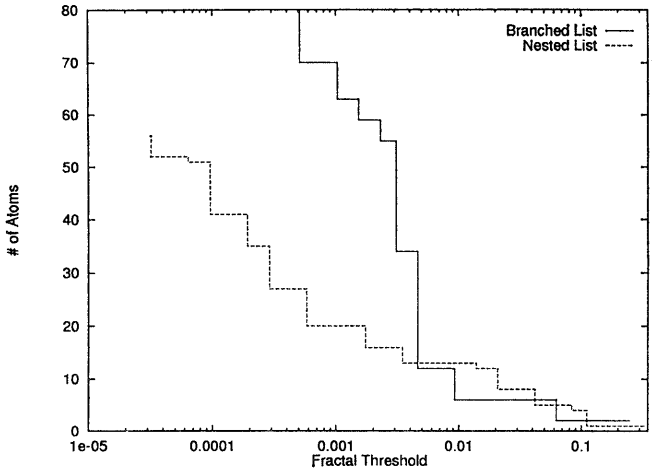

図 9 Fractal View プリンタを使用した場合，枝分かれの多いリスト (BranchedList) とネストの深いリスト (NestedList) における，fractal-threshold と表示アトム数の関係
Fig. 9 The relation between fractal threshold and the number of displayed atoms in the branched list and the nested list when using Fractal View Printer.

```
(define (sqrt x)
  (define (good-enough? guess x)
    (< (abs (- (square guess) x)) .001))
  (define (improve guess x)
    (average guess (/ x guess)))
  (define (sqrt-iter guess x)
    (if (good-enough? guess x)
        guess
        (sqrt-iter (improve guess x) x)))
  (sqrt-iter 1 x)))
```

図 10 Scheme プログラム
Fig. 10 An example Scheme program.

```
1 ]=> (print)                                    ;(A)

(define
 (sqrt x)
 (define (? ? ?) (? % ?))
 (define (? ? ?) (? ? %))
 (define (? ? ?) (? % ? %))
 (sqrt-iter 1 x))
;No value

1 ]=> (begin(down)(next)(next)(print))           ;(B)

(define
 (? ? ?)
 (define (good-enough? guess x) (< (? %) .001))
 (? % %)
 (? % %)
 (? ? ?))
;No value

1 ]=> (begin(next)(print))                       ;(C)

(define
 (? ? ?)
 (? % %)
 (define
  (improve guess x)
  (average guess (? ? ?)))
 (? % %)
 (? ? ?))
;No value
```

図 11 着目点指向 Fractal View プリンタでの表示例
Fig. 11 Session with the focus-oriented Fractal View Printer.

能力を Lisp プリンタによって示すことであった．表示オブジェクト数，対象S式のタイプ（関数・マクロ定義，算術式，データなど），ユーザの焦点・意図すべてを満足したプリンタの実現には，知識工学の支援が必要であろう．ただし，この場合そのアルゴリズムはより複雑になり，計算時間もかかることが予想される．一方，Fractal View プリンタは非常に簡単なアルゴリズムで実現されており，一般の Lisp プリンタと同じくらい実用的である．そして，表示量の制御，表示量の柔軟な増減，着目点近傍の表示という点においてこれに勝っている．

一般に Lisp プログラムは小さな関数の集合として実現されるため，1つの関数が非常に大きくなることは少ない．Common Lisp では，let，do 内の局所変数や，cond での条件分岐が多いとき以外は，分岐数もあまり多くはならない．一方，本論文で取り上げた Scheme の場合には Pascal と同じように関数の中に関数を定義することができる．したがって，Common Lisp と比べ各関数は大きく，かつ分岐数の多い S 式となる傾向がある．この場合 Fractal View プリンタがより有効である．また，本論文では Lisp を例にしたが，Cや Pascal といった他の構造化言語でも，その構文木を編集時に対話的に取得できるようにすれば，本手法を応用した構文エディタ[3),13)]を実現できる．

7. おわりに

フラクタルの概念を応用した提示情報量制御手法 Fractal View の Lisp プリンタへの応用を示した．Fractal View の持つ提示量制御能力により，対象とするS式の形の違いによる提示量の差を少なくすることができた．そして，その表示量を柔軟に変更することができた．さらに，提示量制御と同時に着目点近傍の詳細と周辺部の概略表示を利用した着目点指向エディタを示した．今後は対話的なプログラム編集環境に本手法を実装することを考えている．

参考文献

1) Abelson, H., Sussman, G. J. and Sussman, J.: *Structure and Interpretation of Computer*

Programs, MIT Press (1985).
2) Barnsley, M. F., Jacquin, A., Malassenet, F., Reuter, L. and Sloan, A. D.: Harnessing Chaos for Image Synthesis, *Comput. Gr.*, Vol. 22, No. 4, pp. 131-140 (1988).
3) Barstow, D. R.: A Display-Oriented Editor for Interlisp, Barstow, D. R., Shrobe, H. E. and Sandewall, E. eds., *Interactive Programming Environments*, McGraw-Hill (1984).
4) Bolt, R. A.: *The Human Interface*, Lifetime Learning Publications (1984).
5) Feiner, S. and Beshers, C.: Worlds within Worlds Metaphors for Exploring n-Dimensional Virtual Worlds, *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST '90)*, pp. 76-83, ACM Press (1990).
6) Furnas, G. W.: Generalized Fisheye Views, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '86)*, pp. 16-23, ACM Press (1986).
7) Hanson, C.: *MIT Scheme Reference Manual* (1991).
8) Henderson, D. A., Jr. and Card, S. K.: Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-based Graphical User Interface, *ACM Trans. Graphics*, Vol. 5, No. 3, pp. 211-243 (1986).
9) 金子 博: フラクタル特徴とテクスチャ解析, 信学論, Vol. 70, No. 5 (1987).
10) 小池英樹: Generalized Fractal Views, 日本ソフトウェア科学会第8回大会予稿集, pp. 169-172 (1991).
11) 小池英樹, 石井威望: フラクタルの概念に基づく提示情報量制御手法, 情報処理学会論文誌, Vol. 33, No. 2, pp. 101-109 (1992).
12) Mandelbrot, B. B.: *The Fractal Geometry of Nature*, W. H. Freeman and Company, New York (1982).
13) Mikelsons, M.: Prettyprinting in an Interactive Programming Environment, *ACM SIGPLAN Notices*, Vol. 16, No. 6, pp. 108-116 (1981).
14) Peitgen, H.-O., Jürgens, H. and Saupe, D. eds.: *Fractals for the Classroom: Part One Introduction to Fractals and Chaos*, Springer-Verlag, New York (1992).
15) Steele, G. L., Jr.: *Common Lisp the Language, 2nd edition*, Digital Press (1990).
16) 高安秀樹: フラクタル, 朝倉書店 (1986).
17) Vicsek, T.: *Fractal Growth Phenomena*, World Scientific, London (1989).
18) 和田英一: Lisp ウィンドウ (またはSウィンドウ), 第30回冬のプログラミングシンポジウム (1989).

(平成5年2月25日受付)
(平成6年5月12日採録)

小池 英樹 (正会員)

1961年生. 1991年東京大学大学院工学系研究科情報工学専攻博士課程修了. 工学博士. 同年電気通信大学電子情報学科助手. 1994年同大学大学院情報システム学研究科助教授. 同年よりカリフォルニア大学バークレー校客員研究員. 情報視覚化の研究に従事. 特に3次元グラフィックスを利用したソフトウェア開発環境, フラクタル・カオス等の情報視覚化への応用に興味を持つ. 1991年日本ソフトウェア科学会高橋奨励賞受賞. 日本ソフトウェア科学会, 人工知能学会, IEEE/CS, ACM 各会員.