# フラクタルの概念に基づく提示情報量制御手法†

小池英樹†† 石井威望†††

計算機から人間への情報伝達媒体は主にディスプレイであるが，その大きさに物理的制約がある以上，問題はいかに着目点とその近傍を表示し，それ以外を隠蔽するかである．また，人間の認知的側面から考えると，その時提示情報量をほぼ一定の量に制御することが重要であると考えられる．本論文では，フラクタルの概念を用いた提示情報量制御手法を提案する．本手法は木として表現できる情報構造に対して適用可能で，その特徴は，(1)着目点とその近傍を着目点の変更に関係なくほぼ一定の数だけ提示できる，(2)この提示量を柔軟に設定できることの2点である．まず，フラクタル次元の概念を図としての完全木から一般木へ拡張し，さらに論理木において各ノードに概念的重みを仮定することで論理木へと拡張した．一般の論理木にフラクタル性を実現するための必要十分条件は，この概念的重みが各分岐点における分岐数$N$と定数$C$，$D$で定義される縮尺率 $r=C\exp(N,-1/D)$ に基づいて決定されることである．次に，論理木のフラクタル性に基づいた提示情報量制御手法を提案し，計算機実験によってその提示量制御能力を示した．次いで，1応用例としてプログラム表示への応用を示した．表示行数をほぼ一定に保ちつつ，着目行近傍を表示することが可能となる．さらに，関連研究との比較によって本手法の有効性を述べるとともに，他分野への応用について考察を行った．

## 1. はじめに

計算機技術の発達は，蓄積可能な情報量の増大をもたらしたが，一方で人間は限られた大きさのディスプレイを通して，これらの情報に接する必要がある．しかし，ディスプレイの大きさに物理的制約がある以上，問題はいかに着目点近傍を表示し，非着目点を表示しないかである．また，人間の認知的側面から見ると，提示情報量過多の場合は，認知に要する負荷が高まり，過少の場合は人間の認知能力を効果的に活用しているとは言えない．したがって，提示情報量を可能な限りこの範囲内に維持し，かつ着目点の移動に伴う提示量の変化を少なくすることが重要である．現在，この提示情報量制御の問題は，プログラム表示，知識ベース表示，Visual Programming，HyperText 等，多くの分野において問題となっており，ユーザ・インタフェース設計における本質的な問題の一つであると考えられる．複雑な情報構造を，何らかの形で抽象化して表示する手法，およびその抽象度を制御できるような手法が必要である．B. B. Mandelbrot が提案したフラクタル[1]は，複雑さを定量化し，これらの数学的取り扱いを可能とした点が意義深く，工学への応用としては画像処理等での研究が盛んである[4),5)]．これらの研究は，画像を複雑な対象と捉え，これに対するフラクタル次元を利用して画像処理を行うものである．本論文は，情報構造に対するフラクタル次元を利用することで，上述した抽象化表示，および抽象度制御を行う手法を提案する．

以下では，まず本手法の基本概念について述べ，次にフラクタル次元の定義を物理的形状を持たない論理木にまで拡張する．さらに，この拡張されたフラクタル次元を用いた提示情報量制御手法を提案し，1例としてプログラム表示への応用を示す．最後に本手法の問題点，関連する他の研究との比較，およびその他の応用について考察を行う．

## 2. 基本概念

**図1**は，フラクタル図形の代表例として頻繁にとり上げられるコッホ曲線である．厳密に言えば，この図は prefractal[2] と呼ばれる近似にすぎず，真の意味でのフラクタル性は無限状態においてのみ実現する．この近似のメカニズムは，ある複雑な対象を観察者側で設定したスケールに従い，抽象化することにより実現される．スケールを大きく設定すると抽象度は高まり，逆に小さく設定すると抽象度は低くなる．

UNIX のディレクトリ構造に代表されるように，計算機で扱われる情報構造は，一般に木，あるいはネットワーク構造を為す．この論理的情報構造を複雑な対象と捉えると，適当なスケールの概念を導入することで，これら計算機で利用される木の抽象化が可能となり，かつ利用者側での抽象度の制御が可能になると考えられる．ただし，ネットワークは便宜上，木に変換することが可能であるので，本論文は木構造のみ扱

† A Fractal-based Method for Information Display Control by HIDEKI KOIKE (Department of Communication and Systems, The University of Electro-Communications) and TAKEMOCHI ISHII (Faculty of Environmental Information, Keio University).
†† 電気通信大学電子情報学科
††† 慶応大学環境情報学部

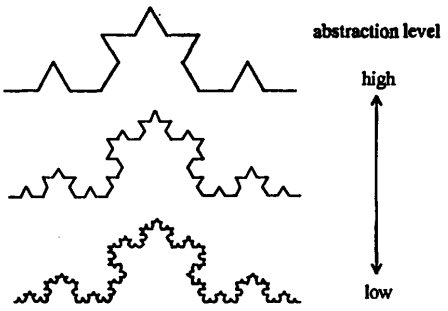


図 1 コッホ曲線
Fig. 1 Koch curve.

うものとする．

上述の抽象化を実現するには，論理木の複雑度，すなわちフラクタル次元を定義する必要がある．しかし，文献 1) 等に述べられているフラクタル次元は，絵として描画された木のフラクタル次元である．以下では，この物理的木に対するフラクタル次元を，論理的木に対するフラクタル次元に拡張する．

## 3. フラクタル

本章ではフラクタルに関する基本的概念と，フラクタル次元の定義について必要事項を簡単にまとめる．

### 3.1 自己相似図形と相似性次元

自己相似図形とは，全体と部分とが厳密に相似な図形のことで，前述のコッホ曲線はその代表的な例である．相似性次元は，ある図形が全体を $1/a$ に縮小した図形 $b$ 個で構成されているとすると，

$$D=\log_a b$$

と定義される．この定義によれば，線分，正方形，立方体の相似性次元は各々 1，2，3 となり，経験的次元と一致する．またコッホ曲線の場合は，

$$\log_3 4=1.26\cdots$$

となり非整数値をとる．この相似性次元は厳密な自己相似図形にしか適用されず，任意の図形の次元は Hausdorff-Besicovitch 次元で定義される．

### 3.2 Hausdorff-Besicovitch 次元

いま空間内の点の集合 $S$ を $d$ 次元球 $h(\delta)=r(d)\delta^d$ で被覆することを考える．この時

$$M_d=\Sigma r(d)\delta^d$$

に対し，$d<D$ で発散し，$d>D$ で 0 となるような $D$ が存在する．この時の $D$ を Hausdorff-Besicovitch 次元と呼び，このような $D$ は与えられた図形に対し唯一存在することが証明されている[2]．自己相似図形においては，相似性次元と Hausdorff-Besicovitch 次元は一致する．

### 3.3 統計的フラクタル

Hausdorff-Besicovitch 次元の欠点は，取り扱いが繁雑なことである．そこで一般によく用いられる方法は，あるスケール $r$ とその $r$ から求められる物理量 $N(r)$ を，縦軸に $\log N(r)$，横軸に $\log r$ をとって log-log プロットする方法である．これによれば，フラクタルは以下に示す右下がりの直線上にプロットされ，直線の傾きからフラクタル次元が得られる．

$$\log N(r)=-D\log r+C_0$$

例として，任意の海岸線の長さを適当な $r$ に従い log-log プロットすると，右下がりの直線となる．このように，正確に自己相似ではないが，広い意味で部分が全体の縮小になっているものを統計的フラクタルと呼ぶ．つまり，log-log プロットが右下がりの直線となることが，フラクタルの本質であるとも言える．

## 4. 木のフラクタル性の概念拡張

### 4.1 完全木のフラクタル性

B. B. Mandelbrot も述べているように，樹形構造はフラクタル性を持つ．図 2 は 2 分木をある反復アルゴリズムによって表示した例であるが，分岐が無限に続く理想的な状態を仮定すれば，この木は厳密な意味での自己相似性を有している．

前出のコッホ曲線の場合，その相似性次元は，あるレベルでの代表長さと次のレベルでの代表長さとの縮尺率(スケール・ファクタ)$r=1/3$ と，得られる図形数 $N=4$ とから，

$$D=-\log_r N=\log_3 4=1.26\cdots$$

と計算された．この定義にならえば，図 2 のような 2 分木の相似性次元は，あるレベルでの枝の長さがその前のレベルでの枝の長さの $r$ 倍になっているとすると

$$D=-\log_r 2$$

で定義される．$N$ 分木の場合も同様にして

$$D=-\log_r N \qquad (1)$$

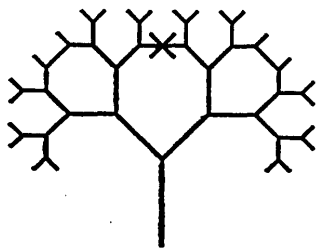

図 2 フラクタルな木
Fig. 2 Fractal tree.

で定義することができる．上式がフラクタルな完全$N$分木（以後完全木と記す）のフラクタル次元の定義である．

### 4.2 一般木のフラクタル性

では，一般木がフラクタルであるための条件は何であろうか．ただし，一般木は完全木のように厳密な自己相似ではないので，ここでは 3.3 節で述べた log-log プロットによる方法を用いることとする．

図2の2分木において，最初の枝の長さを仮に1とすると，深さ$n$のノードから分岐する枝の長さ$\delta$は，

$$\delta = r^n \qquad (2)$$

である．よって，この深さのノードから分岐する枝の長さの総和 $L(\delta)$ は，

$$L(\delta) = (2r)^n$$

となる．(2)式より$n$が

$$n = \ln \delta / \ln r$$

と表されることから，$L(\delta)$ は以下のように表される．

$$L(\delta) = (2r)^n = exp\left(\frac{\ln \delta[\ln 2 + \ln r]}{\ln r}\right) = \delta^{1-D}$$

ただし，

$$D = -\ln 2 / \ln r$$

したがって，その深さにおける枝の総数 $N(\delta)$ は，

$$N(\delta) = 2^n = 2^{\ln \delta / \ln r}$$

であるから，

$$N(\delta) = \delta^{-D}$$

となる．これを横軸に $\log \delta$，縦軸に $\log N(\delta)$ をとったプロットが右下がりの直線となることは明らかで，その傾きは $-D$ であり，結果として log-log プロットで求められる次元は，先に求めた相似性次元と一致する．

さてここで，あるスケール・ファクタ $r_2$ を持つ2分木が，世代 $n$ からスケール・ファクタ $r_3$ を持つ3分木に変化する継ぎ木を仮定する．この時，この木の log-log プロットは**図3**のようになる．いま，2分木のプロット部分と3分木のプロット部分が1直線となるためには，その傾きが等しいこと，つまり，

$$\log_{r_2} 2 = \log_{r_3} 3$$

が必要十分である．これを一般化すれば，ある一般木の各分岐において，分岐数$N$とスケール・ファクタ$r$との間に，常に

$$\log_r N = Constant$$

の関係があるならば，その log-log プロットは右下がりの直線となる．よって，このような一般木は統計的フラクタルであると言える．この時フラクタル次元$D$との関係は以下のようになっている．

$$\log N = -D \log r + C_1$$

変形すると，

$$r = C_2 N^{-\frac{1}{D}} \qquad (3)$$

となる．ただし $C_1$, $C_2$ は定数である．

### 4.3 論理木のフラクタル次元

さて，4.1 節で求めた$N$分木の相似性次元は，枝の長さという物理量を持つ木の相似性次元である．一方，計算機で取り扱う論理的な木の場合，図に描画された木の枝の長さのような，物理的長さというものを持たないため，そのままでは相似性次元の計算は不可能である．しかしこの問題は，木のノードが持つある概念的な重みを仮定することによって概念拡張できる．つまり，論理木の各ノードは，ある概念的な重み（これをフラクタル値と呼ぶ）を持ち，木のレベルが深くなるにつれて，ある縮尺率$r$に従いその値が小さくなっていく，とする（**図4**）．

この概念拡張によって，そのフラクタル値がスケール・ファクタ$r$によって制御される，論理的な完全$N$分木の相似性次元は

$$D = -\log_r N$$

として定義することが可能となる．前出のコッホ曲線は，$D = \log_3 4$ を持つ論理木の具現化の一形態であり，カントール集合は $D = \log_3 2$ を持つ論理木の具現化の一形態であるとも考えられる．

この論理的完全木に対するフラクタル次元の定義

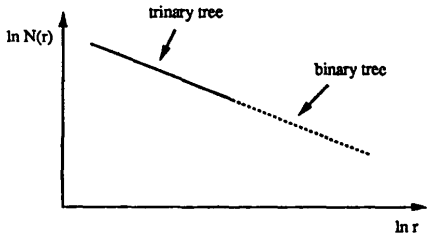


**図3** 2分木および3分木の log-log プロット
Fig. 3 Log-log plot of binary tree and trinary tree.

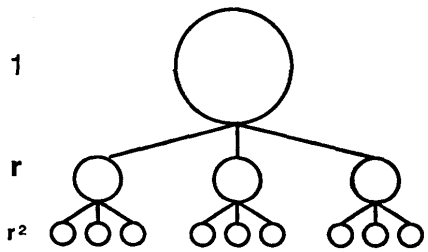


**図4** 論理木のフラクタル性
Fig. 4 Fractal character of logical tree.

は，4.2節と同様にして一般木へも拡張できることは明らかである．

## 5. フラクタル次元に基づく提示情報量制御手法

### 5.1 定式化

本章では，4章で概念拡張した一般の論理木のフラクタル性を用いた，提示情報量制御手法を定式化する．

まず，対象とする情報構造において，現在の着目点をルートとする木を定義する．次にこの木に対し，各ノードにおける分岐数を $N_x$，そこでのスケール・ファクタを $r_x$ とした時，各ノードのフラクタル値 $Fv$ を以下の式に従って決定する．

$$\begin{cases} Fv_{root} = 1 \\ Fv_{child\_of\_x} = r_x Fv_x \end{cases} \quad (4)$$

ただし

$$r_x = CN_x^{-\frac{1}{D}}$$

$D$は適当なフラクタル次元，$C$は $0<C\leq 1$ の定数である．伝播される値は木の下へ行くほど小さくなり，また分岐が多い時ほど子ノードに伝播される値は小さくなる．任意の閾値 $k$ を選び，それ以上の値を持つノードを表示することにすると，閾値に従い異なる表示を得ることができる．簡単化のため $C=1$，$D=1$ とした時の伝播の様子を**図5**に示す．

まず，ルートの値は1である．ルートでの分岐数は2であるから，子ノードの値は(4)式より，$\exp(2, -1/1)=1/2$ となる．以下同様にして各ノードに値が伝播される．そして，図に示すように閾値を1/2に設定すると，合計4個のノードが表示される．また，閾値を1/6以下に設定すれば全ノードが表示されることになる．

図からわかるように，この例において閾値 $k=1/2$ と $k=1/3$ には違いがない．つまり，有効閾値はごく少数で，一見閾値設定の柔軟さが欠如しているように思えるが，これは例が小さな木における1伝播例だからであり，6章の実例で示すように，ある程度の大きさを持った木構造において，個々のノードを各々着目点とした時，統計的に見ると有効閾値の設定は細かく行うことができる．また，図の例では分岐が1の場合，ノードの値が変化しないが，$C$を1未満に設定することで，この問題を回避する．

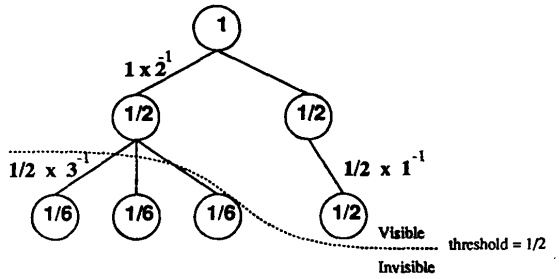


**図5** フラクタル値伝播の概念図
Fig. 5 Calculation of fractal value of each node.

### 5.2 完全木での有効性の検証

1章で述べたように，情報提示において重要なのは，ユーザの着目点が変化しても，全体としての提示量が大きく変動しないことだと考えられる．本節では，本手法の有効性，すなわち提示量一定化の能力について検証する．

前節までの考察から完全木のフラクタル次元は，スケール・ファクタ $r$ と分岐数 $N$ を用いて(1)式で定義された．ゆえに $N$ は $r$ と $D$ によって

$$N=r^{-D}$$

と表される．一方，完全木において深さ $n$ までの総ノード数 $M(n)$ は

$$M(n)=\frac{N^{n+1}-1}{N-1}$$

である．また，深さ $n$ におけるノードのフラクタル値 $Fv$ は

$$Fv=r^n=N^{-\frac{n}{D}}$$

これより

$$-\frac{n}{D}=\log_N Fv$$

$$n=-D\log_N Fv$$

よって

$$M(n)=\frac{N^{1-D\log_N Fv}-1}{N-1}$$

$$=\frac{NFv^{-D}-1}{N-1}$$

$$=\frac{Fv^{-D}-1/N}{1-1/N}$$

となり，$N\to\infty$ に対し $Fv^{-D}$ に収束する．**図6**は，$D$ 一定の基で $k(=Fv)$ を変化させた場合の $N$ と $M(n)$ の関係を，横軸に分岐数 $N$，縦軸に $M(n)$ をとってグラフにしたものである．このグラフから，分岐数が比較的小さいうちに収束値に近づいているのがわかる．つまり，完全木においてある閾値 $k$ 以上のフラクタル値を持つノードの総数 $M(n)$ は，その分岐

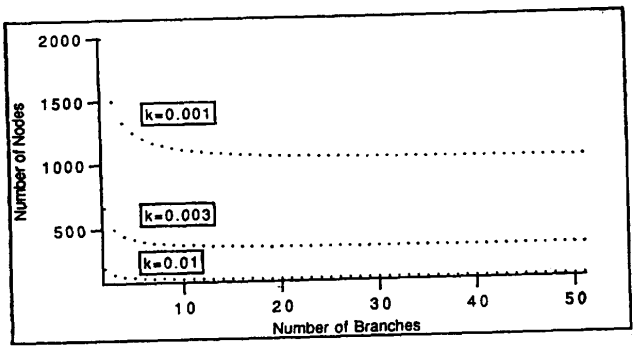


図 6 分岐数$N$とノード数の関係 ($D=1$)
Fig. 6 The relation between the number of branches ($N$) and the number of nodes whose values are greater than threshold ($k$) ($D=1$).

数に関係なくほぼ一定の値となる，と言える．

### 5.3 一般木での有効性

一般木で最悪のケースは，明らかに $N=1$ の完全木の場合である．これを別にすれば，全体としての提示量が分岐数にあまり影響されない（図6）ことから，複数の分岐が混在する一般木においても，その提示量はほぼ一定の値となると考えられる．

また，これはフラクタルの性質を利用することで，以下のようにも説明できる．フラクタルにおいて唯一重要なのはフラクタル次元であり，これがフラクタルの性質，つまり複雑さを決定する．例えば，三陸海岸，ノルウェーの海岸に関係なく，もしそのフラクタル次元が $\log_3 4$ と計測された場合，両者の性質（この場合，任意のスケールとこのスケールで海岸線を計測した回数の関係）は，統計的に等しい．前述の自己相似図形コッホ曲線は，次元 $\log_3 4$ を持つ曲線の特殊な場合であるが，やはり同じ性質を持つ．次元が同一ならば，厳密な自己相似図形（完全木に対応）でも，統計的フラクタル（一般木に対応）でもその性質は同じになることが保証される．

本手法では論理木をフラクタルとして扱えるような拡張を施し，フラクタル図形でのスケールと長さの関係に，閾値とノード数の関係が対応する．したがって，同一次元により生成された以上，完全木，一般木の区別なく，同じ性質を示すと考えられる．

## 6. 適用例：プログラム表示

本章では，5.1 節で定式化した手法のプログラム表示への応用を示す．対象としては，後述する Generalized Fisheye Views との比較のため，文献 6) で使用されたのと同じ，図7に示すCプログラムを用いる．

一般にプログラム開発はスクリーン・エディタで行われる．例えば 23 行表示可能なエディタを用いる場合，図7の 32 行目を中心とすると，表示されるのはその前後各々 11 行ずつである．着目行近傍は詳細に

```
1  #define DIG 40
2  #include <stdio.h>
3  main()
4  {
5      int c, i, x[DIG/4], t[DIG/4], k = DIG/4, noprint = 0;
6      while((c=getchar()) != EOF){
7          if(c >= '0' && c<= '9'){
8              x[0] = 10 * x[0] + (c-'0');
9              for (i=1;i<k;i++){
10                 x[i]=10*x[i]+x[i-1]/10000;
11                 x[i-1] %= 10000;
12             }
13         }else{
14             switch(c){
15                 case '+':
16                     t[0] = t[0] + x[0];
17                     for(i=1;i<k;i++){
18                         t[i]= t[i] + x[i] + t[i-1]/10000;
19                         t[i-1] %= 10000;
20                     }
21                     t[k-1] %= 10000;
22                     break;
23                 case '-':
24                     t[0] = (t[0] + 10000) - x[0];
25                     for(i=1;i<k;i++){
26                         t[i] = (t[i] + 10000) - x[i] - (1 - t[i-1]/10000);
27                         t[i-1] %=10000;
28                     }
29                     t[k-1] %= 10000;
30                     break;
31                 case 'e':
32                     for(i =0;i<k;i++) t[i]= x[i];
33                     break
34                 case 'q':
35                     exit(0);
36                 default:
37                     noprint = 1;
38                     break;
39             }
40             if(!noprint){
41                 for(i=k-1;t[i] <= 0 && i> 0; i--);
42                 printf("%d",t[i]);
43                 if(i > 0) {
44                     for (i --; i>= 0; i--){
45                         printf("%04d",t[i]);
46                     }
47                 }
48                 putchar('\n');
49                 for (i=0; i> k; i++) x[i] = 0;
50             }
51         }
52         noprint = 0;
53     }
54 }
```

図 7 対象とするCプログラム
Fig. 7 Example C program.

```
  6        while((c=getchar()) != EOF){
 13            }else{
 14                switch(c){
 15                    case '+':
 16                        t[0] = t[0] + x[0];
 17                        for(i=1;i<k;i++){
 18                            t[i]= t[i] + x[i] + t[i-1]/10000;
 19                            t[i-i] %= 10000;
 20                        }
 21                        t[k-1] %= 10000;
 22                        break;
 23                    case '-':
 24                        t[0] = (t[0] + 10000) - x[0];
 25                        for(i=1;i<k;i++){
 26                            t[i] = (t[i] + 10000) - x[i] - (1 - t[i-1]/10000);
 27                            t[i-1] %=10000;
 28                        }
 29                        t[k-1] %= 10000;
 30                        break;
 31                    case 'e':
>>32                     for(i =0;i<k;i++) t[i]= x[i];
 33                      break
 34                    case 'q':
 35                        exit(0);
 36                    default:
 37                        noprint = 1;
 38                        break;
 39                }
 40                if(!noprint){
 50                }
```

図 8 32 行目に着目した時の表示 ($k=0.01$)
Fig. 8 Displayed lines when user's focus is on the 32nd line ($k=0.01$).

見ることができる反面，全体の中における着目行の位置を把握するのは難しい．実際のプログラム開発においては，現在編集中の行近傍は詳細に表示されるべきであるが，編集箇所に関連するプログラムのネスト構造を表現する行（例えば，if 文や while 文等）が，明示的に表示されているほうが便利な場合がある．

このプログラムはそのインデントが表すような木として捉えることが可能である．ユーザの着目行が決まれば，この行をルートとした木を定義することができる．次に，式(4)によって各行が持つフラクタル値を決定し，適当な閾値を設定することで各行の表示・非表示を決定できる．例えば$C$，$D$共に1とし，ユーザの焦点が 32 行目にあるとすると，各行のフラクタル値は一意に決定される．この時，各閾値によって表示される行は，**図 8〜11** に示すように変化する．

同様にユーザの焦点が6行目にある時にも，閾値を変更することで異なる表示が得られる．この時の閾値と表示される行数の関係をグラフにしたものが**図 12** である．このグラフからわかるように，異なる行に着目した場合でも閾値と表示行数の関係は同じ曲線上にのる．そして例えば，閾値を 0.01 とすることにより表示行数は約 30 行前後，また閾値を 0.1 とするこ

```
  6        while((c=getchar()) != EOF){
 13            }else{
 14                switch(c){
 15                    case '+':
 16                        t[0] = t[0] + x[0];
 17                        for(i=1;i<k;i++){
 20                        }
 21                        t[k-1] %= 10000;
 22                        break;
 23                    case '-':
 24                        t[0] = (t[0] + 10000) - x[0];
 25                        for(i=1;i<k;i++){
 28                        }
 29                        t[k-1] %= 10000;
 30                        break;
 31                    case 'e':
>>32                     for(i =0;i<k;i++) t[i]= x[i];
 33                      break
 34                    case 'q':
 35                        exit(0);
 36                    default:
 37                        noprint = 1;
 38                        break;
 39                }
 40                if(!noprint){
 50                }
```

図 9 32 行目に着目した時の表示 ($k=0.02$)
Fig. 9 Displayed lines when user's focus is on the 32nd line ($k=0.02$).

```
  6        while((c=getchar()) != EOF){
 13            }else{
 14                switch(c){
 15                    case '+':
 23                    case '-':
 31                    case 'e':
>>32                     for(i =0;i<k;i++) t[i]= x[i];
 33                      break
 34                    case 'q':
 35                        exit(0);
 36                    default:
 37                        noprint = 1;
 38                        break;
 39                }
 40                if(!noprint){
 50                }
```

図 10 32 行目に着目した時の表示 ($k=0.025$)
Fig. 10 Displayed lines when user's focus is on the 32nd line ($k=0.025$).

```
 13            }else{
 14                switch(c){
 15                    case '+':
 23                    case '-':
 31                    case 'e':
>>32                     for(i =0;i<k;i++) t[i]= x[i];
 33                      break
 34                    case 'q':
 35                        exit(0);
 36                    default:
 37                        noprint = 1;
 38                        break;
```

図 11 32 行目に着目した時の表示 ($k=0.05$)
Fig. 11 Displayed lines when user's focus is on the 32nd line ($k=0.05$).

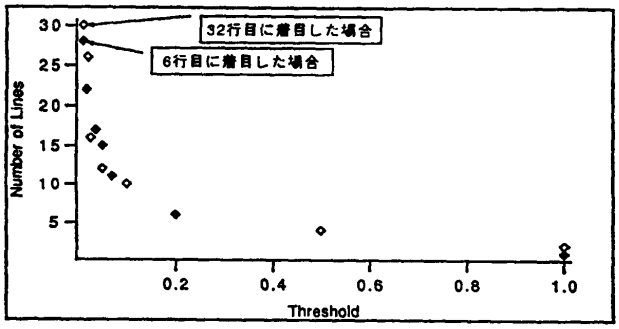


図 12 着目行による閾値と表示行数の関係
Fig. 12 The relation between threshold and the number of displayed lines when user's focus is on the 6th line and the 32nd line.

とにより表示行数は約 10 行前後に制御される．他の行を着目行とした場合をプロットすると，各プロット間の間隙が補間され，結果として有効閾値の数も増加する．

このように本手法を用いることで，表示する総行数をほぼ一定に保ちつつ，着目行近傍を詳細に，かつ周辺部を抽象化して表示することが可能となる．

## 7. 考　　察

### 7.1 本手法の問題点

5章では，完全木における数値解析を基に，フラクタルの性質を利用して，完全木における結果を一般木へと拡張する理想的議論を行った．しかし実際には，多少の誤差が生じざるを得ない．この誤差は，図6のグラフにおいて，分岐数の小さい場合を無視した結果生じるものである．この結果，プログラム表示の例の場合にも，各行を着目点とした場合の閾値と表示行数の関係は，実際にはきれいな1曲線ではなく，ある誤差範囲を持った帯域にプロットされる．この誤差は，木が小さい場合には問題であるが，対象とする情報構造の肥大化に伴い，その程度は相対的に小さくなると考えられる．閾値設定の柔軟性に関しても同様のことが言えて，閾値の離散度が小とみなされるためには，対象とする情報構造がある程度大きな木でなければならない．

本手法は，情報の持つセマンティクスを無視し，シンタクスのみを考慮している．したがって，提示される情報が必ずしもユーザの求めるものである保証はない．例えばプログラム表示例の場合，得られた表示がプログラマの見たい行を提示しているとは限らない．これに関しては，必要に応じてセマンティクスを考慮した，知識工学的側面等からの補助が必要であると思われる．ただし，1章でも述べたように，多くのアプリケーションにおいて，提示情報量制御の必要性が認識されており，その1解決方法として本手法は意義のあるものであると考える．

### 7.2 関連する研究

木構造に着目した提示情報量削減手法としては，まず LISP のプリンタが挙げられる．例えば Common Lisp の場合，*print-level*, *print-length* という二つの大域変数がオブジェクトの表示数を制御する．前者は入れ子になったデータ・オブジェクトを印字する深さを制御し，後者はあるレベルにある要素の印字個数を制御する．詳細な表示例は文献7)に譲るが，この手法はフレキシブルな値の設定が可能である半面，表示する量をシステムが制御することはできない．

より洗練された手法として Generalized Fisheye Views[6] がある．この手法は，ある階層構造において，木の各ノードのルートからの距離 $API$ と，現在着目しているノードからの距離 $D$ を用いて，各ノードが持つ重要度 $DOI$ を以下の式に従い決定する．

$$DOI = API - D$$

そして，ある閾値以上の $DOI$ を持つノードは表示し，それ以外は消去する．

以上の二つの方法は，木の深さを重視している点が共通である．今，深さ $n$ のノードには $n$ という値を与えるものとすると，完全木において閾値 $k$ 以下の値を持つノードの総数 $M$ は，木の分岐数 $N$ と $k$ の関数として以下のように表される．

$$M = \frac{N^{k+1}-1}{N-1}$$

横軸に $N$，縦軸に $k$ 以下の値を持つノードの総数 $M$

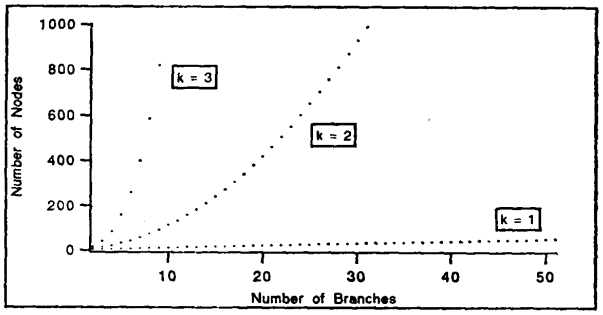


図 13 木の深さを考慮した場合の分岐数 $N$ とノード数の関係
Fig. 13 The relation between the number of branches ($N$) and the number of nodes whose value is less than threshold $k$.

```
 1    #define DIG 40
 2    #include <stdio.h>
 3    main()
 4    {
 5     int c, i, x[DIG/4], t[DIG/4], k = DIG/4, noprint = 0;
 6     while((c=getchar()) != EOF){
 7         if(c >= '0' && c<= '9'){
13          }else{
14              switch(c){
15                  case '+':
33                  case '-':
31                  case 'q':
32                      for(i =0;i<k;i++) t[i]= x[i];
33                      break
34                  case 'q':
36                  default:
39              }
40              if(!noprint){
50              }
51          }
52          noprint = 0;
53      }
54   }
```

図 14 32 行目に着目した時の first order fisheye
Fig. 14 First order fisheye from the 32nd line.

```
 4    {
 6     while((c=getchar()) != EOF){
13          }else{
14              switch(c){
31                  case 'q':
32                      for(i =0;i<k;i++) t[i]= x[i];
```

図 15 32 行目に着目した時の zero order fisheye
Fig. 15 Zero order fisheye from the 32nd line.

をとり，$k$を変化させた時のグラフを図 13 に示す．$k=1$ の時，上式は $M=N+1$ と簡略化され，グラフにおいて右上がりの直線として表される．また，$k>1$ の時には分岐数が増加するに従い$M$は指数関数的に増大する．図 14 と図 15 は，図7の 32 行目に着目した時に，Generalized Fisheye Views を用いた時の閾値による表示の変化である．閾値の変化に伴う表示量の変化が大きい．

結局，これらの手法の問題点は，同一閾値であっても，対象とする木の分岐数が大きければ非常に多くのノードが表示され，分岐数が小さければ僅かのノードしか表示されないこと，つまり，木の形状に依存して表示される量が極端に変化する点が上げられる．さらに，閾値を1大きくしただけで，表示される数は極端に増加するという点において，閾値設定の柔軟性に欠ける．一方，本手法の場合，ある閾値の設定によって提示される量はほぼ一定であると同時に，図6からわかるように，提示したい量に合わせて閾値は自由に設定できるという閾値設定の柔軟性をも持っている．

### 7.3 他への応用可能性の検討

現在，HyperText[8]の特殊例としてのアウトライン・プロセッサが商品化されている．これは各表題とその内容を構成要素とし，必要に応じて内容を隠し，表題のみを表示する機能を持っている．これにより，ユーザは文章全体の流れを表題で把握することができる．しかし，内容の表示・非表示操作はユーザに委ねられており，毎回操作を行うのは面倒である．この文章のアウトラインは木として表現可能であり，本手法の適用が可能である．本手法の適用により，現在の着目点近傍の内容は表示し，それ以外は自動的に閉じることが可能となる．しかも，その時表示されている表題の数はほぼ一定となっている点が重要である．これは一般の HyperText に対しても有効であると考えられる．

Visual Programming[9]における大きな問題の一つとして，図素数増加に伴う図形表現の複雑化がある．この図形表現を木として捉えることが可能であるならば，本手法を適用して着目する図素近傍を詳細に，かつ周辺部を漠然と表示することが可能となる．また，Visual Programming システムはグラフィクスを利用するが，表示図素数増加に伴うシステムの反応速度の低下は避けられない．一方，本手法の持つ提示量制御機能により，(1)人間の認知負荷を一定にできる，と同時に(2)システムの反応性を一定にすることも可能となる．

同様に CG を利用するシステムとして現在，人工現実感システム[10]が注目を集めているが，よりリアルな現実感を得るためには，モデルの記述を詳細に行う必要がある．しかしモデルの記述が詳細になるほど，グラフィクスの反応性は低下し，ユーザの動作に伴うグラフィクスの追従性は悪化する．一般に，3次元 CG におけるグラフィクス・オブジェクトの管理には階層構造を用いることが多いが，本手法をファームウェア化すれば，システムの反応性を一定にしつつ，着目するオブジェクト近傍は詳細に，かつ世界全体は抽象化して表示することが可能となる．

また，本論文では閾値による表示・非表示戦略をとったが，このほかにもフラクタル値をノードの大きさに対応させたグラフ表示や，ノードの輝度に対応させたぼかし表示等も考えられる．

## 8. おわりに

ユーザ・インタフェース設計の立場から，フラクタルの概念を用いた提示情報量制御手法について述べた．本手法は木として表現可能な情報構造に対して適用可能であり，その特徴は(1)閾値の設定で提示情報量をほぼ一定に保てること，(2)この閾値の設定を柔軟に行えることの2点である．

今後の課題としては，多くの実例への適用を通して，本手法を実際のシステムへ組み込んだ際に生じる問題点の検討と，その解決を行う必要があると考えている．

## 参考文献


1) Mandelbrot, B. B.: *The Fractal Geometry of Nature*, W. H. Freeman and Company (1977).
2) Feder, J.: *FRACTALS*, Plenum, New York (1988).
3) 高安秀樹：フラクタル，朝倉書店，東京 (1986).
4) 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, pp. 964-972 (1987).
5) Barnsley, M. F. et al.: Harnessing Chaos for Image Synthesis, *Comput. Gr.*, Vol. 22, No. 4, pp. 131-140 (1988).
6) Furnas, G. W.: Generalized Fisheye Views, *CHI '86*, pp. 16-23, ACM (1986).
7) Steele Jr., G. L.: *Common Lisp: the Language*, Digital Press (1984).
8) Halasz, F. G. et al.: NoteCards in a Nutshell, *CHI+GI 1987*, pp. 45-52, ACM (1987).
9) Shu, N. C.: *Visual Programming*, Van Nostrand Reinhold, New York (1988).
10) Foley, J. D.: Interfaces for Advanced Computing, *Sci. Am.*, Vol 257, No. 4, pp. 126-135 (1987).


(平成3年5月9日受付)
(平成3年12月9日採録)

**小池 英樹**（正会員）

1961年生．1986年東京大学工学部船用機械工学科卒業．1991年同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．同年より電気通信大学電子情報学科助手．計算機ユーザ・インタフェースの研究に従事．特に，ソフトウェア視覚化，ソフトウェア開発環境，フラクタルに興味を持つ．ソフトウェア科学会，人工知能学会，ACM 各会員．

**石井 威望**（正会員）

1930年生．1954年東京大学医学部医学科卒業．1957年同大学工学部機械工学科卒業．同年より1年間通産省．1963年東京大学大学院博士課程修了．工学博士．同年同大学工学部専任講師，1964年同助教授，1973年同教授．1991年より慶応大学環境情報学部教授．システム工学，医用工学の研究に従事．1972年 IEEE 論文賞受賞．