〔公　開〕

TR－C－0135

樹木画像を入力とする３次元樹木形状のフラクタルモデルの自動推定方法（遺伝的アルゴリズムによる方法）

桑原　教彰　Noriaki KUYAHARA
志和　新一　Shinichi SHIWA
岸野　文郎　Fumio KISHINO

１９９６　３．５

ＡＴＲ通信システム研究所

樹木画像を入力とする３次元樹木形状のフラクタルモデルの自動推定方法（遺伝的アルゴリズムによる方法）

桑原教彰，　志和新一，　岸野文郎
ＡＴＲ通信システム研究所

あらまし

自然景観の仮想空間表示には，多種多様な樹木形状が必要であり，そのような，シーンに相応しい形状を対話的に作成することは大変な労力を要する．そこで筆者らは，樹木形状の正面図と側面図を入力として，意図する樹木形状を生成するフラクタル形状を，適当に定めた初期値から最適傾斜法を用いて，自動的に推定する方法を提案した．本報告では，フラクタル形状の自動推定に遺伝的アルゴリズム（ＧＡ）を用いた場合の結果を，既に提案している最適傾斜法を用いた場合の結果と比較し，最適傾斜法を用いた場合のほうが良好な結果を得られることを示す．

１．　はじめに

筆者らが研究を進めている人工現実感の応用形態である臨場感通信[1]では，３次元ＣＧで合成された画像を２眼式立体表示することにより生成した仮想空間を，遠隔地に離れた利用者が共有することで，利用者があたかも一堂に会しているかのような雰囲気を作り出すことを目的としている．筆者らのシステムは，３次元ＣＧと立体表示装置を用いて仮想空間を生成することで，利用者の視点位置に応じたリアルタイムでの仮想空間表示及び，仮想空間を構成する仮想物体の利用者の手による直接操作を実現している．筆者らはよりリアリティの高い仮想空間を実現するために，自然の景観を仮想空間として表示することを検討している．

近年，リアリティの高い仮想空間表示の手段として，仮想空間の背景表示に実写画像などを用いるなどの手法が提案されている[2-6]．しかし，例えば風などによる草木などの揺らぎの表現には，形状モデルに基づいたオブジェクトの変形が必要であり，３次元ＣＧによる表示に頼るほかない[7,8]．そこで筆者らは，樹木の多数存在するシーンを，３次元ＣＧを用いて実時間で運動視表示を可能にする，高速３次元樹木画像生成法を提案した[9,10]．これは，３次元樹木形状データをフラクタルを用いてモデル化し，形状粗さに応じて樹木形状データを生成し，利用者近くに表示される樹木には高精細な形状データを，利用者の遠方の樹木には粗い形状データを用いて表示を行なう．これにより，シーン全体のデータ量を大幅に削減することができる．そして，形状粗さに応じた樹木形状データに対して，樹木画像をテクスチャマップすることで，形状粗さの変更を利用者に感じさせることのない，３次元樹木の立体表示が可能になった．このとき，テクスチャマップに用いる樹木画像は，フラクタルモデルに基づいて生成される詳細形状をレンダリング処理し，表示面に投影して得たものを使用した．上記手法では，表現したい樹木形状を近似するフラクタルモデルのパラメタは，試行錯誤的に決定した．しかし，自然景観の仮想空間表示を行なう場合，多種多様な樹木形状が必要となり，そのような，シーンに相応しい形状を生成するフラクタルモデルを，試行錯誤的に作成することは大変な労力を要する．そこで筆者らは，シーンに必要な３次元樹木形状の正面図と側面図を入力して，意図する３次元樹木

形状を生成するフラクタルモデルを，自動的に求める手法を考案した．すなわち，３次元樹木形状の正面図と側面図に適合するような３次元樹木形状を生成するよう，フラクタルモデルのパラメタを最適傾斜法により自動的に推定する[11]．このようにして推定したフラクタルモデルを，高速３次元樹木画像生成法[9,10]に用いることで，仮想空間中に設計者の意図するシーンを，実時間運動視表示することができる．

図１に，筆者らの自然景観を対象とする仮想空間表示システムの概要を示す．まず，（１）景観を構成する樹木形状のフラクタルモデルを実写樹木画像より生成する[11]．これを用いて（２）形状粗さに応じて簡略化された樹木形状のデータベースを作成する．また，（３）位置センサーから得られる利用者の位置情報と樹木の位置から距離を求めて，樹木形状の距離に応じた形状粗さを計算する．その結果を用いて（４）形状粗さに応じた簡略化形状を生成し，それに樹木画像をテクスチャマップして表示する[9,10]．これにより，図２に示すような，リアリティの高い仮想空間表示が可能になる．

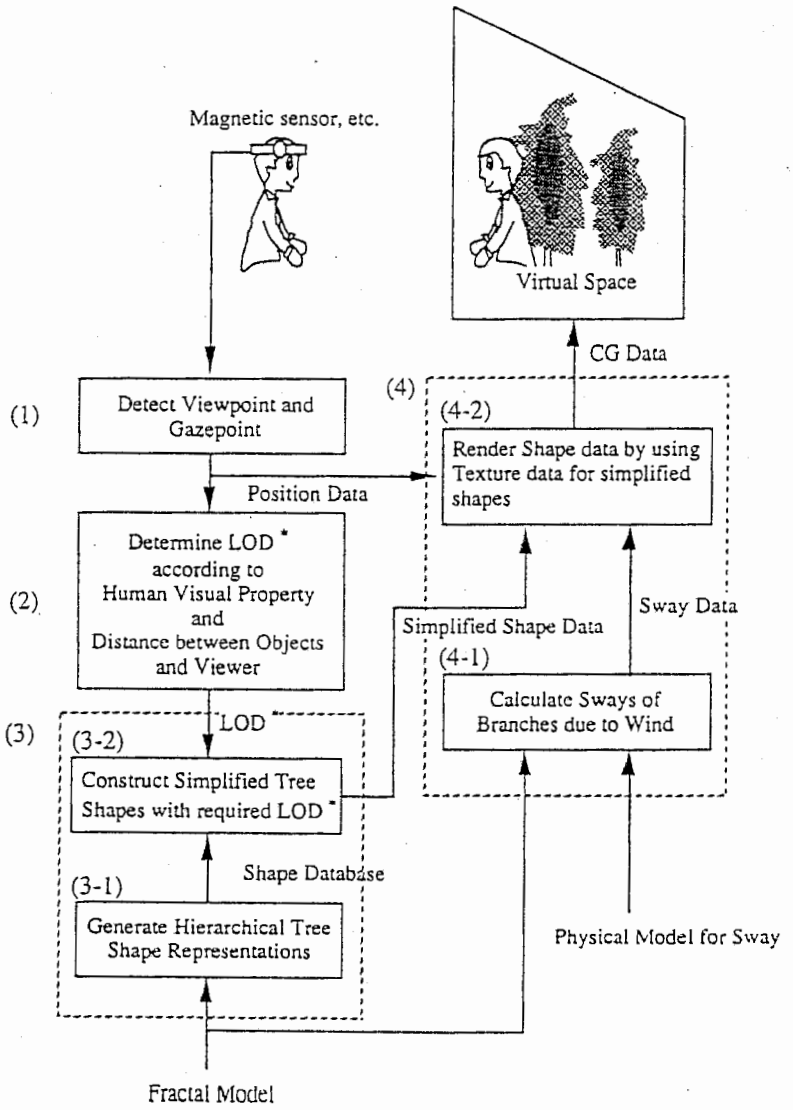


図１　自然景観の仮想空間表示システム

図２　リアリティの高い仮想空間

本報告では，フラクタル形状の自動推定にＧＡを用いた場合の結果を，既に提案している最適傾斜法を用いた場合の結果と比較し，最適傾斜法を用いた場合のほうが良好な結果を得られることを示す．以下に本報告の構成を説明する．まず，２章では，３次元樹木画像の正面図と側面図に適合するフラクタル形状を，ＧＡを用いて自動的に推定する方法を簡単に説明する．３章では，ＧＡを用いて自動的に推定されたフラクタル形状と，最適傾斜法を用いて自動的に推定されたフラクタル形状を比較する．最後に４章で，まとめを述べる．

2．　樹木画像を入力とする3次元樹木形状のフラクタルモデルの自動推定方法

2．1　フラクタルモデルの自動推定方法の概要

3次元樹木形状を得るために，従来の成長モデルに基づく樹木形状生成手法12-15)では，成長のパラメタを設定してシミュレーションを行ない，その結果を確認し，更にパラメタを調整するといった，対話的な作業が必要である．しかし，目に留まった絵画や写真中の樹木形状が欲しいときには，その画像を入力として，意図する3次元樹木形状を生成するパラメタが，自動的に推定できることが望ましい．

これまでに，提案されている3次元形状の再構成手法[16-20]は，再構成される形状を凸と仮定している．また，シルエットに現われない凹面に対応した手法[21]では，得られる形状は基本的に，連続シルエットを結んだ曲面であり，樹木のように，シルエット内部に微細で入り組んだ構造を持つ形状は扱えない．そこで，3次元樹木形状は統計的にフラクタル性を有することから，そのような複雑な内部の構造を再現するために，フラクタルモデルを用いた．筆者らの提案した手法[11]は，既知の3次元形状に適合するフラクタルモデルを推定する形状再構成手法[22-25]とは異なり，3次元樹木形状の正面図と側面図を入力として，図3に示すように，フラクタルモデルのパラメタを調節して，3次元樹木形状の正面図と側面図にフラクタル形状を合わせ込み，仮想空間の3次元ＣＧ表示に使用するフラクタルモデルを得る．

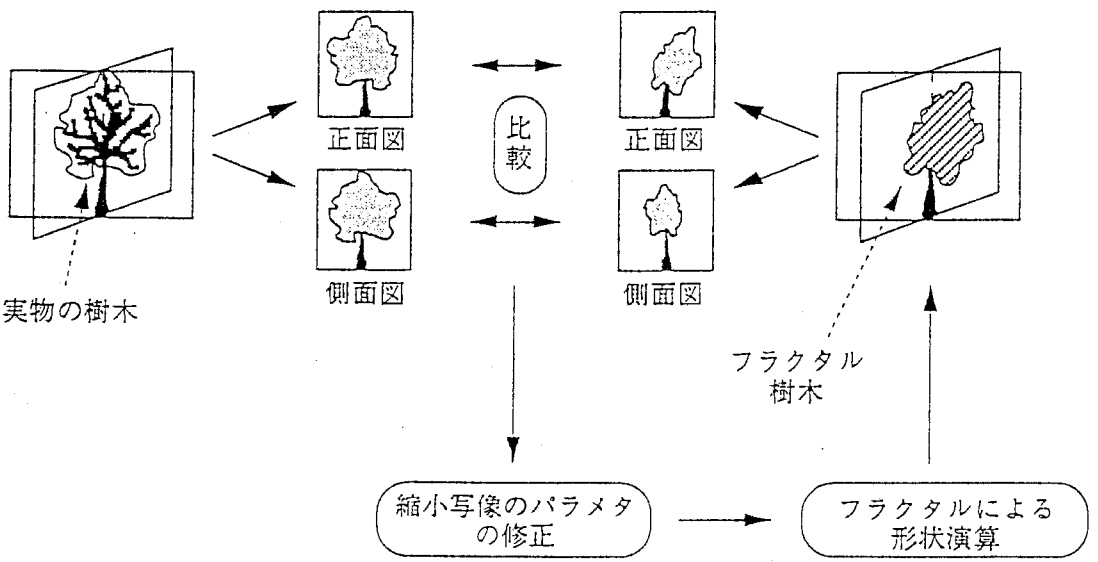


図3　実写樹木画像を入力とする3次元樹木形状のフラクタルモデルの推定方法の概要

このときのフラクタルモデルは，図４に示すパラメタの組 (yn,sn, θ zn, θ yn) で表現できると仮定した．ynは枝AnBnが発生する位置，snは幹OSに対する枝AnBnの長さ比，θ zn, θ ynはそれぞれ，枝AnBnの延びる方向のｚ軸周り，ｙ軸周りの回転量を表す．このパラメタの組は，幹OSの直下の全ての枝について計算される．このとき，３次元樹木形状Ｔを定義する縮小写像ω 1～ω N (N≧1) は３次元アフィン変換である．樹木形状Ｔは，式 (1) に示すように，幹OSを初期値にしてω 1～ω Nを再帰的に適用した結果の和集合として得られる．

$$T=\lim_{k\to\infty}\bigcup_{i=1}^{k}\bigcup_{(j1,\cdots,jk)=(1,\cdots,1)}^{(n,\cdots,n)}\omega_{j1}\cdots\omega_{jk}(OS)\cup OS \quad (1)$$

ω n (1≦n≦N) の具体的な表現を式 (2) に示す．

$$\omega\ n=\mathrm{Trans}\ (0,yn,0)\cdot\mathrm{Roty}\ (\theta\ yn)\cdot\mathrm{Rotz}\ (-\theta\ zn)\cdot\mathrm{Scale}\ (sn) \quad (2)$$

ただし，

Trans (x,y,z)	：	平行移動
Rotz (θ)	：	ｚ軸周りの回転
Roty (θ)	：	ｙ軸周りの回転
Scale (s)	：	拡大縮小

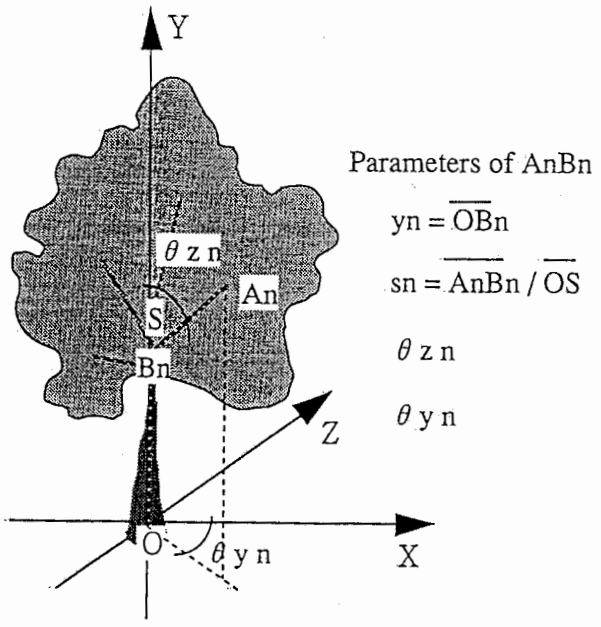


図４　３次元樹木形状のフラクタルモデル

パラメタ調節の際の，入力画像とフラクタル形状の適合度の評価尺度としては，式 (3) に示す，各シルエット上での正規化された相互相関[26] の和を用いることにする．式 (1) では，fi はフラクタル形状の投影画像，gi はシルエット画像，ζi (i は fi を含む領域を示す（i= 正面図，側面図）．式 (1) は，$gi = c \cdot fi$ のとき最大になる．

$$H(\omega 1,...,\omega 5) = \sum_i \frac{\iint_{\zeta i} fi(x,y)gi(x+u,y+v)dxdy}{\sqrt{\iint_{\zeta i} fi^2(x,y)dxdy \iint_{\zeta i} gi^2(x+u,y+v)dxdy}} \quad (3)$$

本手法ではフラクタルモデルは５個の縮小写像 ω n (n=1...5) よりなり，それぞれ４個のパラメタを有する[11]．よって式 (1) は４×５個のパラメタを持つ非線形な式であり，既に筆者らは，最適傾斜法を用いて，これを最大にする縮小写像の組 { ω n : n=1,…,5} を求める手法を提案している[11]．今回は，この部分にＧＡを用いた．

２．２　ＧＡを用いたパラメタの推定方法

ＧＡ[27,28] とは，人間をはじめとする生物の進化過程を模擬したもので，生物の適応性に基づくものである．すなわち，自然淘汰システムの，選択，交差，突然変異を各世代において実行することで，関数などを用いてきれいに記述できない最適化問題を解くことなどを目的として研究されている．具体的には，まず最初に，（ａ）最適化する変数Ｘ１～Ｘｎを染色体に割り当て，適当な個体数で集団を形成する．つぎに，（ｂ）各個体（染色体）の表す解の性質を評価し，その結果を適応度に反映させる．そして，（ｃ）適応度が高いものほど高い確立で交配プールに選択し，交差と突然変異を一定の確率で発生させ世代交代を促す．

上記のアルゴリズムを，縮小写像のパラメタの推定に用いるため，まず，縮小写像のパラメタ {y0, s0, θ z0, θ y0, …, y5, s5, θ z5, θ y5} の２０個を，染色体に割り当てる．各パラメタは 128 で正規化し，７ビットを割り当てた．これを用いて，以下の手順でパラメタの推定を行なった．

（１）　第１世代の集団として，縮小写像のパラメタの初期値をランダムに発生させ，個体数1000の集団を形成する．

（２）　各個体に割り当てられたパラメタを用いて，式（４．４）によって，各シルエット上での生成されたフラクタル形状との相互相関の和を計算する．

（３）　計算結果の値の大きいものから順に集団をソートし，最初の個体から順に高い選択確率を与えて，交配プールにセットする．

（４）　交配プールから，２つの個体を確率的に選択し，交差，突然変異を実行し，次の世代の個体を生成する．これを，新しい世代の集団の個体数が1000になるまで実行する．突然変異の確率は，0.001とした．

（５）　世代数が５０になるまで，上記（１）～（４）を繰り返す．

3．　ＧＡを用いたパラメタの推定方法

図５は，ＧＡを用いたフラクタルモデルの推定方法を用いたとき，各世代で得られた相互相関の最大値を，世代ごとにプロットした図である．このときの入力シルエット画像には，表１に示すフラクタルモデルにより生成された正面図と側面図を用いた．これは，表１のフラクタルモデルがシンメトリーな形状を生成することから，推定された形状との比較が容易に行なえることによる．図５から，世代が進むにつれて相互相関の値は改善されていく様子がわかる．そして48世代目に最大値である約0.661を得た．一方，適傾斜法により推定されたフラクタルモデルでは，約0.756が得られている．

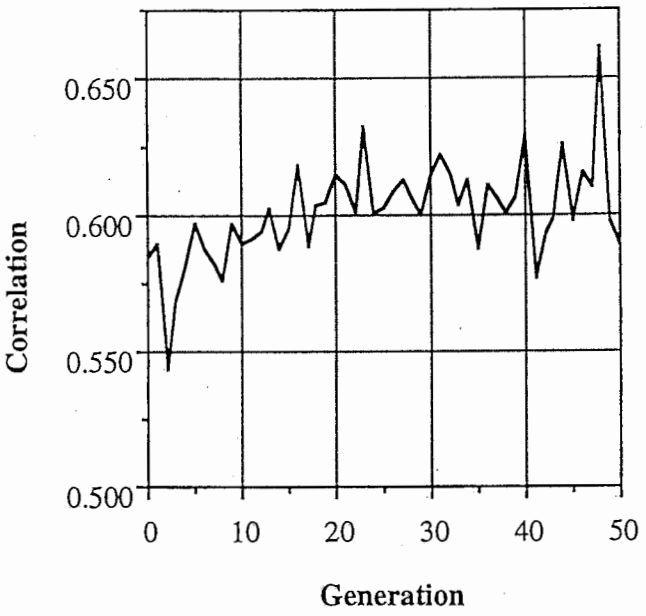


図５　ＧＡを用いたフラクタルモデルの推定での各世代で得られた相互相関の最大値

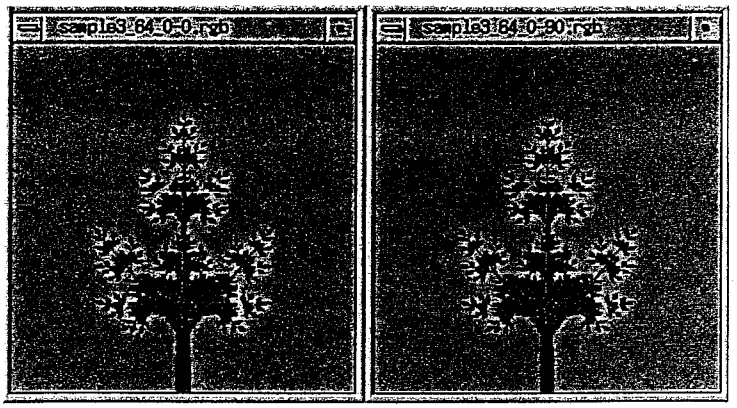
表１　入力画像を生成するのに用いたフラクタルモデルの縮小写像のパラメタ

sn	yn	θ zn	θ yn
0.50	0.40	0.00	0.00
0.50	0.40	45.0	0.00
0.50	0.40	45.0	90.0
0.50	0.40	45.0	180.0
0.50	0.40	45.0	270.0

表２には，ＧＡにより推定されたフラクタルモデルと，最適傾斜法により推定されたフラクタルモデルを示す．そして，図６には，入力シルエット画像と，それらフラクタルモデルを用いて生成された形状の正面図と側面図を示す．表２，図６から，シルエットとの相互相関の低い，ＧＡにより推定されたフラクタルモデルから生成される形状は，最適傾斜法により推定されたフラクタルモデルに比べて，元の形状をあまりよく反映していないことが分かる．

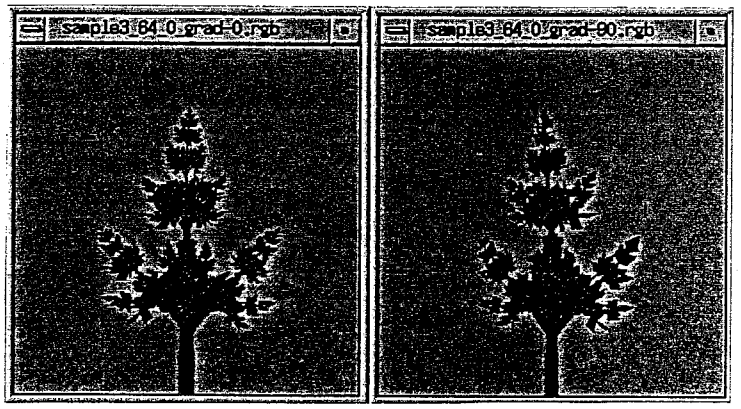
表２　推定されたフラクタルモデルの縮小写像のパラメタ

	sn	yn	θ zn	θ yn
Genetic Algorithm	0.409	0.562	36.563	261.562
	0.480	0.394	71.719	27.422
	0.423	0.344	5.625	158.203
	0.522	0.300	37.969	222.187
	0.423	0.262	46.406	336.094
Optimal Gradient Method	0.546	0.373	3.202	291.578
	0.468	0.505	42.056	3.229
	0.447	0.467	42.215	92.449
	0.491	0.460	41.746	182.266
	0.445	0.470	38.311	272.716



(a) Input Images

(b) Genetic Algorithm



(c) Optimal Gradient Method

図６　推定されたフラクタルモデルによる３ＤＣＧ表示例

4．　まとめ

本報告では，3次元樹木形状の正面図と側面図画像を入力として，それによく適合する3次元樹木形状を生成するフラクタルモデルを，ＧＡを用いて自動的に推定を行なった．そして，その結果を，筆者らが以前提案した，最適傾斜法などによりフラクタルモデルを自動的に推定を行なった結果と比較した．

その結果，筆者らが以前提案した手法では約0.756の相互相関が得られているサンプルに対し，約0.661の相互相関しか得られなかった．また，生成された画像例（図6）から，主観的にも入力画像と大きく異なることが分かる．これと比較して，筆者らが以前提案した，最適傾斜法などを用いるフラクタルモデルの自動推定は，妥当な推定結果が得られているといえる．

参考文献

1) F.Kishino, J.Oya, H.Takemura and N.Terashima : "Virtual Space Teleconferencing System - Real Time Detection and Reproduction of 3D Human Images", HCI'93 Proceedings, (1993) .

2) P.Metzger : "Adding reality to the virtual", VRAIS'93, pp.7-13 (1993) .

3) P.Milgram : "Mixed reality : Research issues in merging real and virtual worlds", ATR Workshop on Virtual Space Teleconferencing, 1993, (1993) .

4) P.Milgram, S.Zhai and D.Drascic : "Applications of augmented reality for human-robot communication", Proc.Int.Conf.Intelligent Robots and Systems, pp.26-30 (1993) .

5) S.Feiner, B.Macintyre and P.Seligmann : "Knowledge based augmented reality", C.ACM, 36, 7, pp.53-62 (1993) .

6) 片山昭宏, 田中宏一郎, 押野隆弘, 田村秀行 : "多視点画像データの補間処理による視点追従型立体画像表示", 3次元画像コンファレンス'94, pp.7-12 (1994) .

7) M.Shinya and A.Fournier : "Stochastic Motion - Motion Under the Influence of Wind", EUROGRAPHICS'92, 11, 3, pp.C119-C128 (1992) .

8) 千葉則茂, 河野充, 佐藤義人, 村岡一信, 斎藤伸自 : "風による樹木の揺らぎ画像生成法の検討", 画像電子学会誌, 22, 5, pp.475-483 (1993) .

9) 桑原教彰, 鉄谷信二, 森井精啓, 岸野文郎 : "視点からの距離による樹木データの階層化", 第24回画像工学コンファレンス論文集, pp.265-268 (1993) .

10) 桑原教彰, 鉄谷信二, 志和新一, 岸野文郎 : "フラクタルを用いた階層的な樹木形状表現による3次元樹木画像の高速生成方法", 電子情報通信学会論文誌 D-II, J78-D-II, 7, pp.1091-1104 (1995) .

11) 桑原教彰, 志和新一, 新井民夫 : "樹木画像を入力とする3次元樹木形状のフラクタルモデルの自動推定方法", 画像電子学会誌, 24, 5, pp.541-549 (1995) .

12) E.Oppenheimer : "Real Time Design and Animation of Fractal Plants and Trees", Computer Graphics, 20, 4, pp.55-64 (1986) .

13) P.Prusinkiewics, A.Lindermayer and J.Hanan : "Development Models of Herbaceous Plants for Computer Imagery Purpose", Computer Graphics, 22, 4, pp.141-150 (1988) .

14) 金丸直義, 斎藤伸自, 千葉則茂, 高橋清明 : "向日性による樹木の自然な枝振りのＣＧシミュレーション", 電子情報通信学会論文誌 D-II, J75-D-II, 1, pp.76-85 (1992) .

15) 千葉則茂, 大川俊一, 村岡一信, 三浦守 : "ＣＧのための樹木の成長モデルー架空の「植物ホルモン」による自然な樹形の生成ー", 電子情報通信学会論文誌 D-II, J76-DII, 8, pp.1722-1733 (1993) .

16) G.Lafue : "Recognition of three-dimensional objects from orthographic views", ACM/SIGGRAPH, pp.103-108 (1976) .

17) H.Sakurai and D.C.Gossard : "Solid model input through orthographic views", ACM/SIGGRAPH, pp.243-252 (1983) .

18) P.Giblin and R.Weiss : "Reconstruction surface from profiles", 1st ICCV, pp.136-144 (1987) .

19) R.Cipolla and A.Black : "The dynamic analysis of apparent contours", 3rd ICCV, pp.616-623 (1990) .

20) 安居院猛, 小峯賢治, 長橋宏：“シルエット画像を用いた鉢植え植物の３次元モデリング”, 1995年電子情報通信学会総合大会講演論文集, 情報・システム 2, pp.380 (1995).

21) J.Yu.ZHENG, 岸野文郎：“連続シルエットを用いた３次元モデルの復元とその未知領域の検出”, 電子情報通信学会論文誌 D-II, J-76-D-II, 6, pp.1114-1122 (1993).

22) 横矢直和：“フラクタルによる３次元複雑形状の解析とその応用”, 信学技報, PRU86-23, (1986).

23) 荒川賢一, エリック　クロトコフ：“フラクタル幾何を適用した自然地形のモデリング”, 電子情報通信学会論文誌 D-II, J-76-D-II, 12, pp.2564-2577 (1993).

24) A.E.Jacquin：“Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations”, IEEE Trans. Image Process., 1, 1, pp.18-30 (1992).

25) T.Mitsa J.Quian and J.R.Galvin：“3-D Modeling of lung Morphogenesis using Fractals”, SPIE, 1898, pp.540-548 (1993).

26) 長尾真（監訳）：“デジタル画像処理”, 近代科学社, (1978).

27) J. H. Holland：“Adaptation in Natural and Artificial System”, The University of Michigan Press, (1975).

28) D. E. Goldberg：“Genetic Algorithms in Search”, Addison Wesley, (1989).

付録

樹木画像生成ＰＧ　プログラム説明書　１部

樹木画像生成ＰＧ

プログラム説明書

目次

1　要求仕様書

本ソフトウエアは、屋外をモデルにした仮想空間を開発する際に、実写景観ＶＴＲ画像から仮想空間を構成する為に必要な３次元ＣＧオブジェクトなどを作成する作業を支援し、その開発が容易に行えることを目的とする。
仮想空間に自然な景観画像を表示可能にする為の３次元樹木画像の高速生成を目的とした樹木形状データを Genetic Algorithm によって獲得する。

1. 出力

   (a) 評価関数 setCpD の各世代毎の最大値。

   (b) 樹木形状データ。

   (c) 実行中のログ。

2　システム仕様書

2.1　機能概要

2.1.1　処理ブロック

1. オブジェクトデータを初期化する。
2. パラメータを初期化する。
3. 樹木の初期画像を表示する。
4. ＧＡ処理を行う。
   (a) 結果出力用のファイルをオープンする。
   (b) 初期集団を生成する。
   (c) param ファイルに設定した世代数分、以下の処理を繰り返す。
      - 交差による複製を行う。
      - 突然変異を行う。
      - 適合度を計算する。
   (d) 最適な遺伝子のデータを取得する。
   (e) 終了処理を行う。
   (f) 結果出力用のファイルをクローズする。
5. 樹木形状データを出力する。
6. システムを終了する。

2.1.2 処理説明

1. オブジェクトデータの初期化
   ｘ、ｙ、ｚ座標、スケール、角度の初期値をセットする。

2. パラメータの初期化

3. 樹木の初期画像の表示
   (a) 背景を黒色で塗りつぶす。
   (b) 樹木画像とパラメータを表示する。

4. ＧＡ処理
   “GA Test” と表示し、ＧＡによる幹、枝の移動を開始する。

5. ＧＡよる幹、枝の移動

(a) 結果出力用のファイルをオープンする。

(b) 初期集団を生成する。

- “item(1～4).dat”ファイルを読み込み、座標の最大値と最小値をセットする。
- “param”ファイルを読み込み、世代数、個体数等の変数の値をセットする。
- “param”ファイルから、突然変異率と交差率を取得する。
- 個体の生成に必要な遺伝子の領域を確保する。
  ランダム関数により適当なビット列を決定し、個体を生成する。
  その個体について適合度を計算する。

(c) “param”ファイルに設定した世代数分、以下の処理を繰り返す。

- “param”ファイルから取得した交差率に従い、交差による複製を行う。
  複製できない遺伝子は絶滅する。
- “param”ファイルから取得した突然変異率に従い、突然変異を行う。
- 複製後、または突然変異後の遺伝子について適合度を計算する。
  - 遺伝子のビット列を読み出し、座標、半径、角度に値をセットする。
  - 座標、半径、角度がそれぞれの最大値、最小値の範囲内にあるかチェックする。
  - 幹と枝全ての座標、半径、角度が範囲内に存在する場合、評価関数 setCpD() を実行してその値を取得する。
  - 世代が同じ間、各個体の setCpD() の値を比較して最大になる時の値、個体 No.、遺伝子を更新し、その世代における最適値を取得する。
  - 世代が変わった時、前世代までの最適値と現世代の最適値を比較して最大となる時の値、個体 No.、遺伝子、世代を更新し、全世代を通しての最適値を取得する。
  - 全ての個体の値が求まったら、適合度の最も高い (setCpD の値が大きい) 時のデータを取得し、範囲をチェックする。
  - 適合度の高いものほど選択され易くなるようにソートする。

(d) 全世代を通して適合度の最も高い遺伝子からデータを取得し、範囲をチェックする。

(e) 各処理に使用したバッファをクリアする。

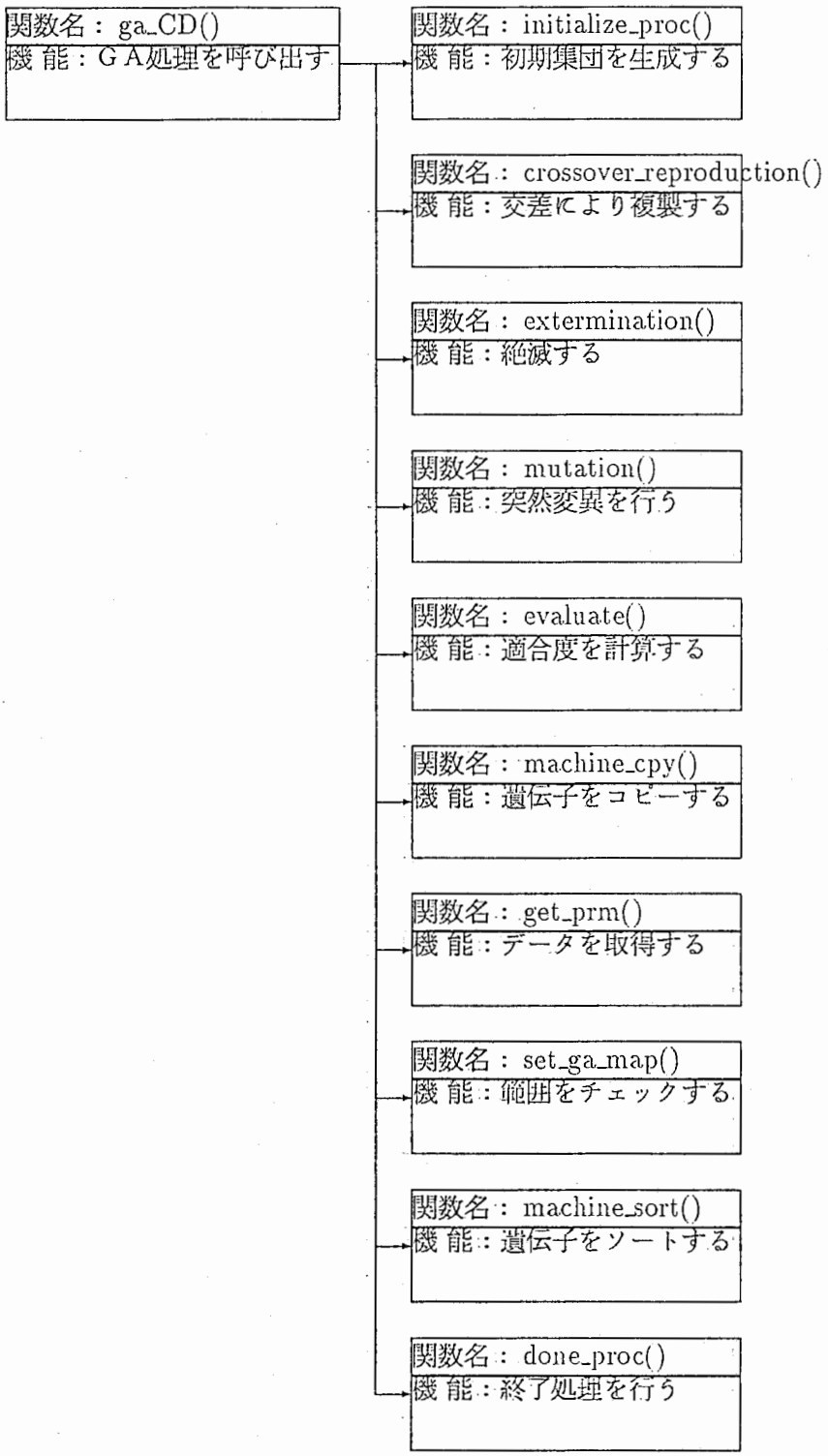
(f) 結果出力用のファイルをクローズする。

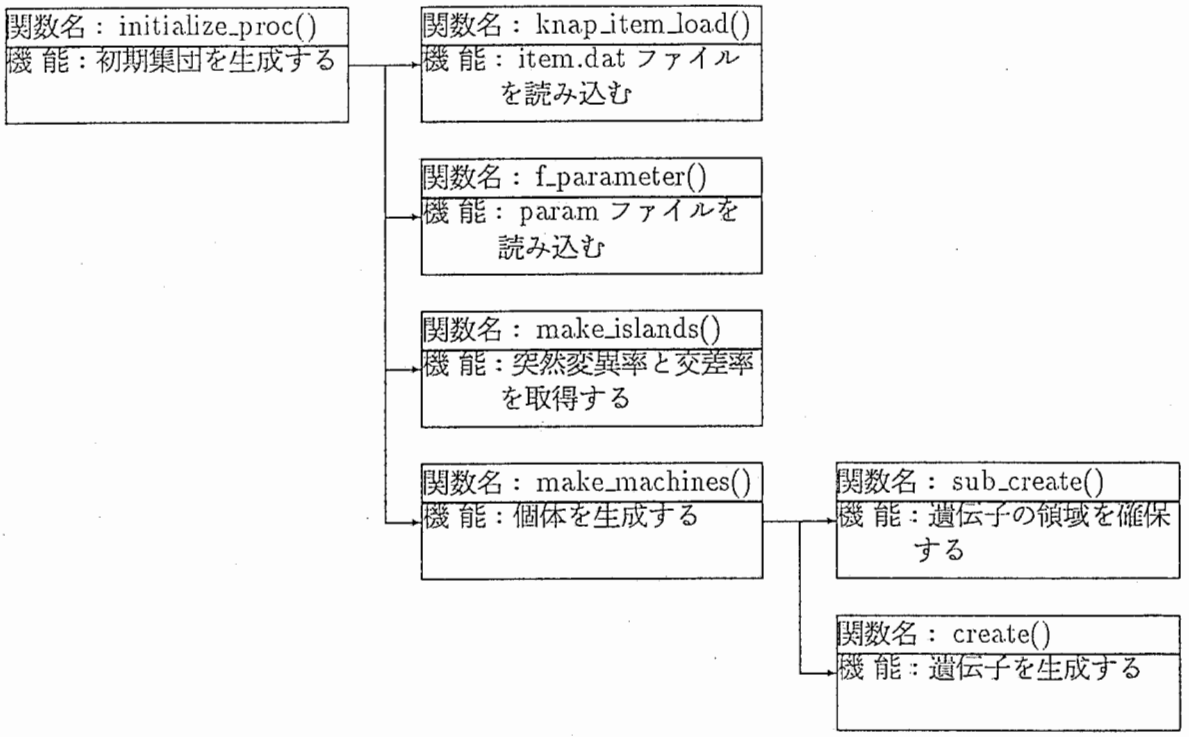
6. 樹木形状データの出力
   “GA Test End” と表示し、樹木形状データをファイルに出力する。

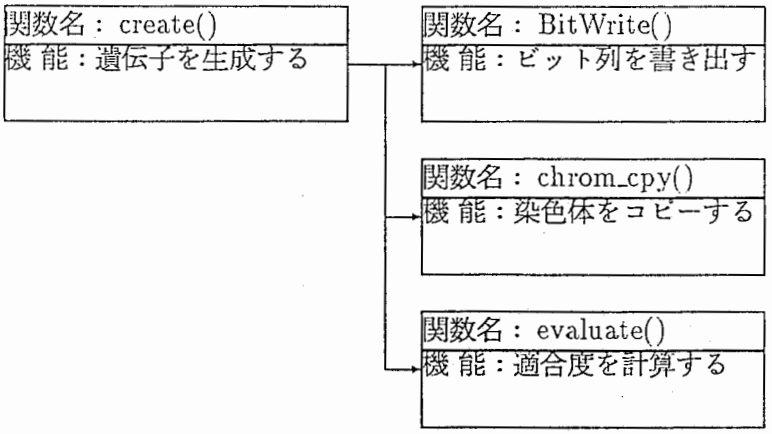
7. システムの終了
   使用した領域をクリアして、システムを終了する。

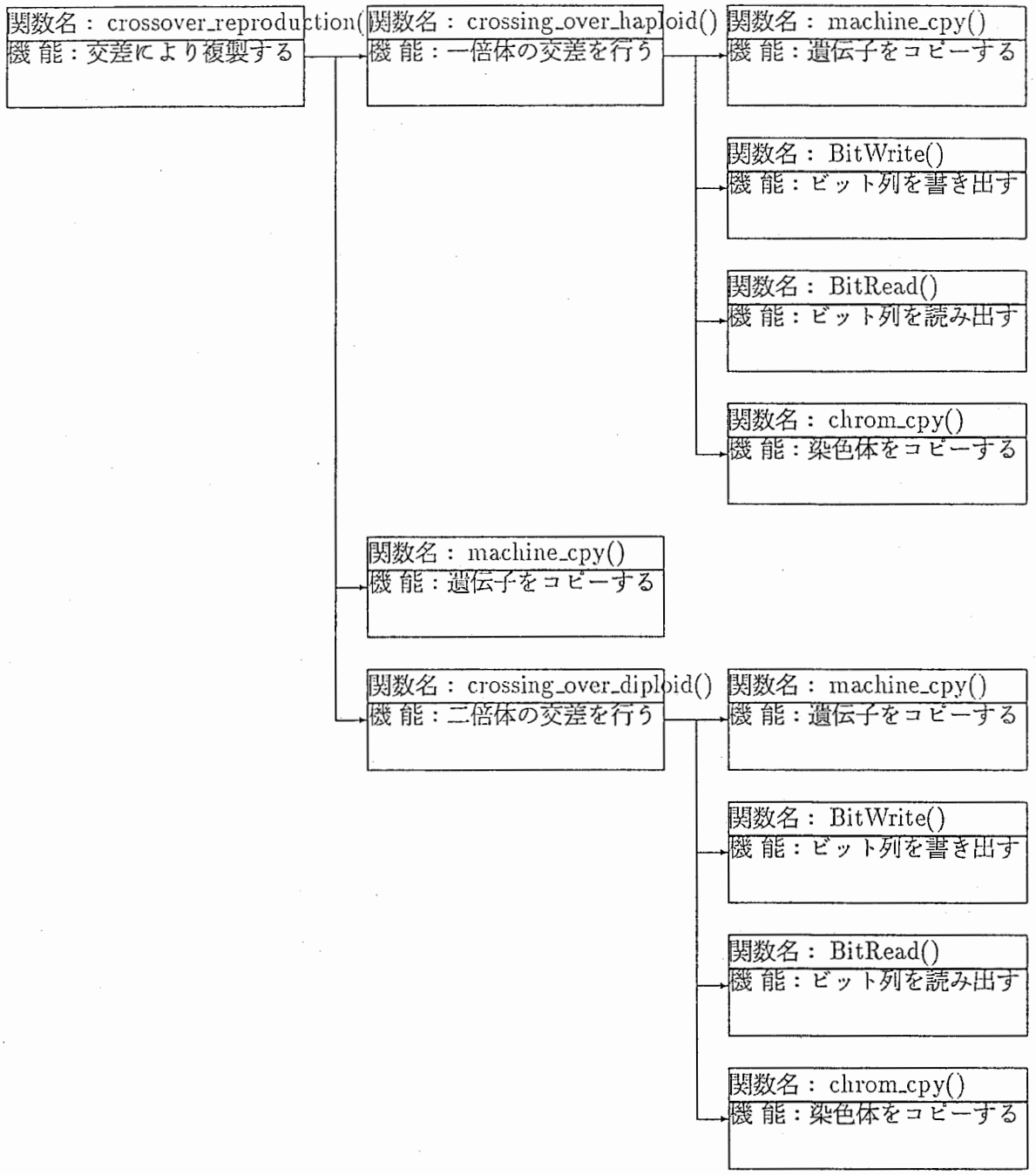
3　プログラム仕様書

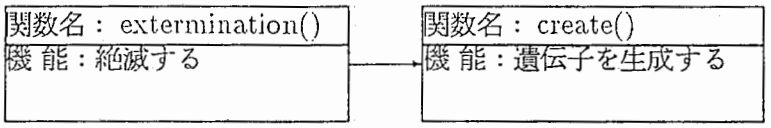
3.1　プログラム構成

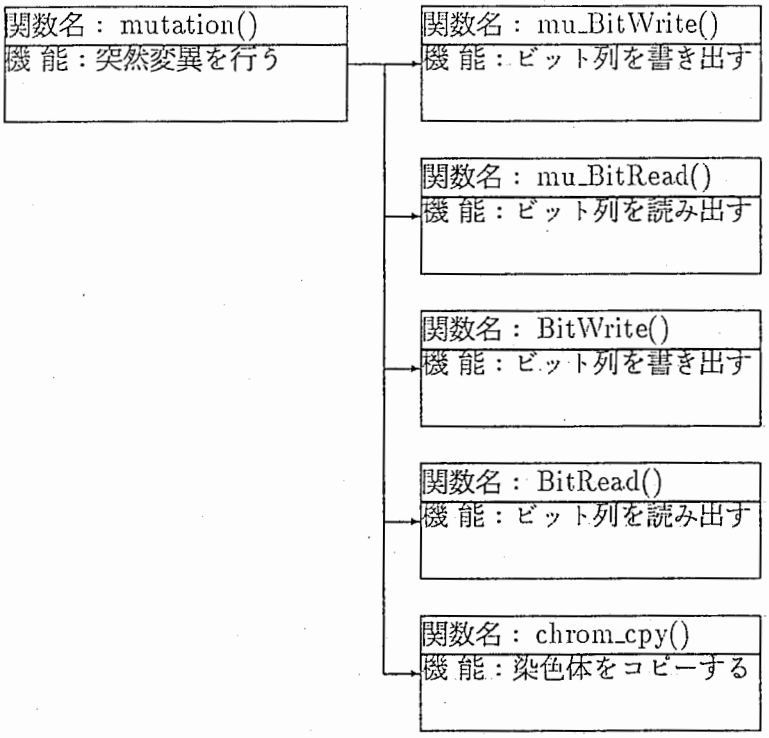


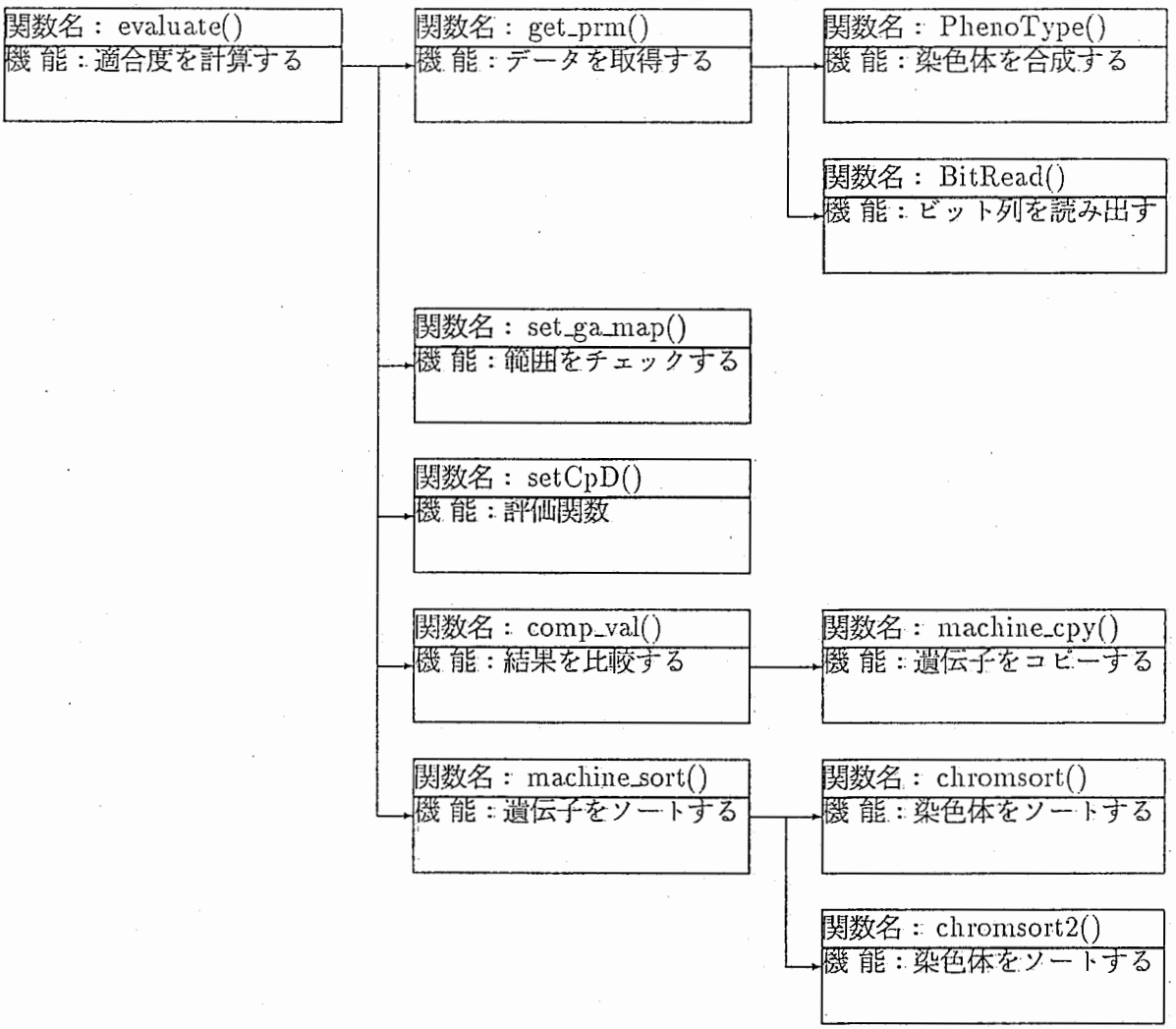

関数名： initialize_proc()
機 能：初期集団を生成する
関数名： knap_item_load()
機 能： item.dat ファイル
を読み込む
関数名： f_parameter()
機 能： param ファイルを
読み込む
関数名： make_islands()
機 能：突然変異率と交差率
を取得する
関数名： make_machines()
機 能：個体を生成する
関数名： sub_create()
機 能：遺伝子の領域を確保
する
関数名： create()
機 能：遺伝子を生成する

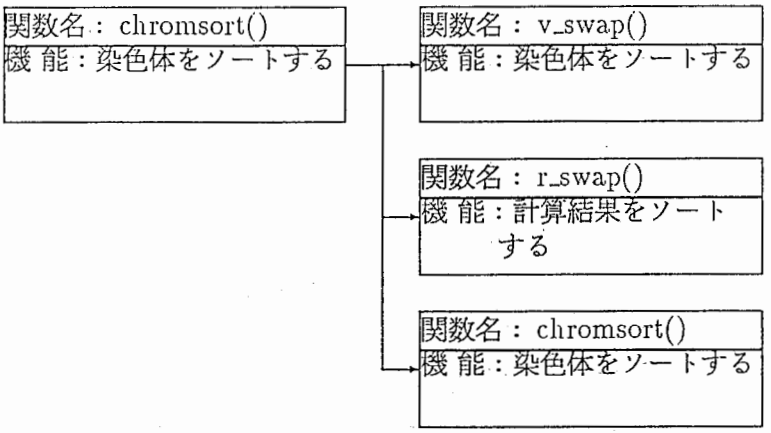

関数名： create()
機 能：遺伝子を生成する
関数名： BitWrite()
機 能：ビット列を書き出す
関数名： chrom_cpy()
機 能：染色体をコピーする
関数名： evaluate()
機 能：適合度を計算する

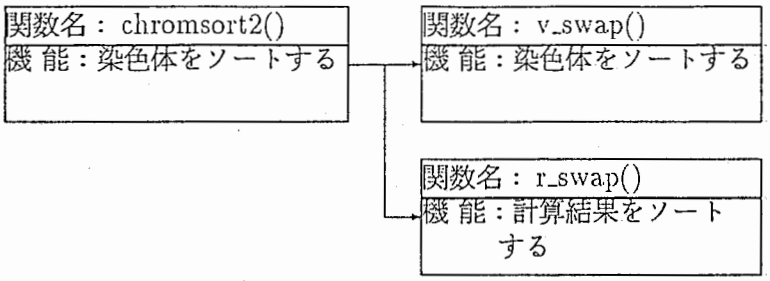

関数名： crossover_reproduction(
機 能：交差により複製する
関数名： crossing_over_haploid()
機 能：一倍体の交差を行う
関数名： machine_cpy()
機 能：遺伝子をコピーする
関数名： BitWrite()
機 能：ビット列を書き出す
関数名： BitRead()
機 能：ビット列を読み出す
関数名： chrom_cpy()
機 能：染色体をコピーする
関数名： machine_cpy()
機 能：遺伝子をコピーする
関数名： crossing_over_diploid()
機 能：二倍体の交差を行う
関数名： machine_cpy()
機 能：遺伝子をコピーする
関数名： BitWrite()
機 能：ビット列を書き出す
関数名： BitRead()
機 能：ビット列を読み出す
関数名： chrom_cpy()
機 能：染色体をコピーする

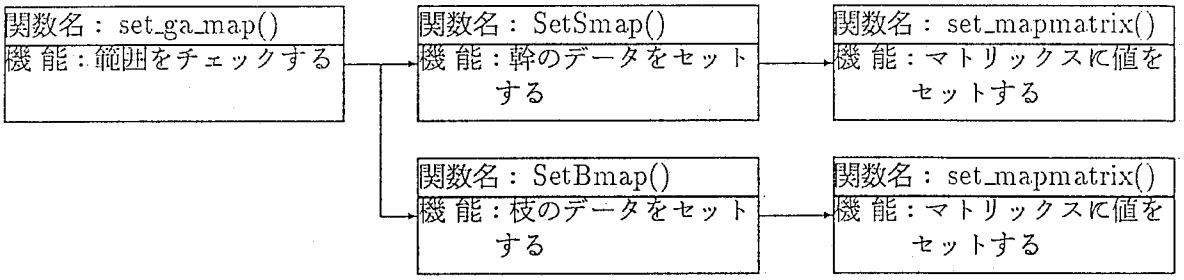

関数名： extermination()
機 能：絶滅する
関数名： create()
機 能：遺伝子を生成する

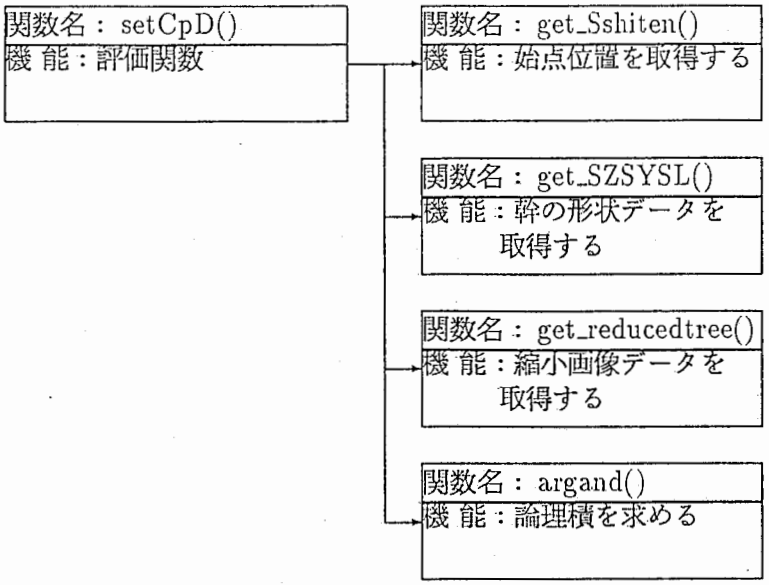

関数名： mutation()
機 能：突然変異を行う
関数名： mu_BitWrite()
機 能：ビット列を書き出す
関数名： mu_BitRead()
機 能：ビット列を読み出す
関数名： BitWrite()
機 能：ビット列を書き出す
関数名： BitRead()
機 能：ビット列を読み出す
関数名： chrom_cpy()
機 能：染色体をコピーする


関数名： evaluate()
機 能：適合度を計算する
関数名： get_prm()
機 能：データを取得する
関数名： PhenoType()
機 能：染色体を合成する
関数名： BitRead()
機 能：ビット列を読み出す
関数名： set_ga_map()
機 能：範囲をチェックする
関数名： setCpD()
機 能：評価関数
関数名： comp_val()
機 能：結果を比較する
関数名： machine_cpy()
機 能：遺伝子をコピーする
関数名： machine_sort()
機 能：遺伝子をソートする
関数名： chromsort()
機 能：染色体をソートする
関数名： chromsort2()
機 能：染色体をソートする


関数名： chromsort()
機 能：染色体をソートする
関数名： v_swap()
機 能：染色体をソートする
関数名： r_swap()
機 能：計算結果をソート
する
関数名： chromsort()
機 能：染色体をソートする


関数名： chromsort2()
機 能：染色体をソートする
関数名： v_swap()
機 能：染色体をソートする
関数名： r_swap()
機 能：計算結果をソート
する


関数名： set_ga_map()
機 能：範囲をチェックする
関数名： SetSmap()
機 能：幹のデータをセットする
関数名： set_mapmatrix()
機 能：マトリックスに値をセットする
関数名： SetBmap()
機 能：枝のデータをセットする
関数名： set_mapmatrix()
機 能：マトリックスに値をセットする


関数名： setCpD()
機 能：評価関数
関数名： get_Sshiten()
機 能：始点位置を取得する
関数名： get_SZSYSL()
機 能：幹の形状データを
取得する
関数名： get_reducedtree()
機 能：縮小画像データを
取得する
関数名： argand()
機 能：論理積を求める

4 取り扱い説明書

4.1 起動方法

1. ciris24 にログインし、次のディレクトリに移動する。
   %cd /tmp_mnt/home/nagamune/project5/ACTIVE/LOG.Var

2. スクリーンセーバをオフにする。

3. testcase に対して experiment.GA.sh を実行し、結果を確認する。

   %experiment.GA.sh testcase
   %cd ../DATA.var1
   %../rgb_gen sample3_64_0_ga.dat 64
   %mkreg sample3_64_0_ga-0.rgb
   %mkreg sample3_64_0_ga-90.rgb
   %cmpedge 1 sample3_64_0
   %lpr -Pimagen sample3_64_0-0.cmpedge sample3_64_0-90.cmpedge

4. var1name, var2name, var3name に対して experiment.GA.sh を実行し、結果を確認する。var2name, var3name の場合は、以下の var1 の文字列をそれぞれ、var2, var3 に変更して実行する。

   %experiment.GA.sh var1name
   %cd ../DATA.var1
   %mkrgb.sh gadat
   %mkreg.sh gargb
   %cmpedge.sh cmpfile
   %corel.sh cmpfile ¿ var1.corel
   %lpr -Pimagen var1.corel

5　修正・追加点

以下の機能を修正及び追加した。

1. 全世代について処理を終えた後、最も適合度の高いデータを取得する。
   <参照> ga_CD(),　comp_val()

2. 適合度の計算において、全個体の中で最も適合度の高いデータを取得する。
   <参照> evaluate(),　comp_val()

3. 計算結果を比較する際、最大になる時の値だけでなくその時の個体 No. と遺伝子も取得する。
   また世代が変わる時は、別の領域へそれぞれ移して保存する。
   <参照> comp_val()

4. 個体の生成において、計算結果の比較で使用する保存用の領域も確保しておく。
   <参照> make_machines()

5. 終了処理で、保存用の領域をクリアする。
   <参照> done_proc()

6　プログラムリスト

```
/*************************************************************************/
/**                                                                     **/
/**     ifsga.c    : tree_exe source file, genetic algorithm.           **/
/**                                                                     **/
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <gl.h>
#include <device.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>

#include "knap.h"
#include "deftree.h"
#include "calc.h"
#include "ifscm.h"
#include "ifsin.h"
#include "ifsadj.h"
#include "ifscl.h"
#include "keyboard.h"

/* ********************************************** Functions ***** */
static int      set_ga_map(float prm[20]);
static int      setCpD(float *CpD, int *rg[4]);
static void     SetSmap(float, float, float, float);
static void     SetBmap(int,float,float,float,float);

/* ********************************************** Grobals ***** */
float           CpD;
float           prm[PRM_NUM];
float           value[POP_NUM];
float           val;
float           max_value;
float           gen_max_value;
int             gen;
int             gen_max;
int             max_pnum;
int             gen_max_pnum;
FILE            *fopen(), *fpp1;

/* ****************************************** Program start. ***** */
 a_CD(int *rg[4])
{
    MACHINE     *machine;
    int         generation;
    int         island;
    int         j;

    gen           = 0;
    gen_max       = 0;
    max_pnum      = 0;
    gen_max_pnum  = 0;
    max_value     = 0.0;
    gen_max_value = 0.0;

    fpp1 = fopen("kekka", "w");                         /* ファイルオープン */
    fprintf(fpp1, "<generation>\n\n");

    initialize_proc(rg);                                /* 初期集団の生成   */

    for(generation=1; generation<=GENERATION_MAX; generation++){
      for(island=0; island<ISLAND_NUM; island++){
                                                        /* 交差による複製   */
        if(crossover_reproduction(island, mac[island]) == -1){
```

```
          extermination(island, generation, mac[island], rg);     /* 絶滅 */
        }
        mutation(island, mac[island]);                  /* 突然変異          */
        evaluate(island, generation, mac[island], rg);/* 適合度の計算      */
      }
    }
    fprintf(fpp1, "********************\n");
    fprintf(fpp1, "<%d> %f (%d)\n", gen_max, gen_max_value, gen_max_pnum);

    machine_cpy(mac[island-1], gmax_mac[0]);            /* 遺伝子のコピー    */
    get_prm(island-1, mac[island-1], 0);                /* データ取得        */
    set_ga_map(prm);                                    /* 範囲のチェック    */
    machine_sort(island, mac[island-1]);                /* 遺伝子のソート    */

    done_proc();                                        /*  終了処理         */
    fclose(fpp1);                                       /* ファイルクローズ  */
    return;
}

/***************************
)     交差による複製
***************************/
int crossover_reproduction(island, machine)
int     island;
MACHINE *machine;
{
    int         i, j;
    int         parent1, parent2;
    float       Total_V = 0.0;
    float       tval = 0.0;
    float       popval[POP_NUM];
    MACHINE     *new_mac = tmp_mac[island];
    double      *repro = (double *)calloc(POPULATION[island], sizeof(double));
    double      rand_repro1, rand_repro2;
    char        *new_chrom = (char *)calloc(BIT_LENGTH/8+(BIT_LENGTH%8!=0), sizeof(ch
ar));

    if((repro == NULL) || (new_chrom == NULL)){
      fprintf(stderr, "Can't allocate memory\n");
      exit(-1);
    }

    for(i=0, j=1; i<POPULATION[island]; i++, j++){
      popval[i] = (float)j/(float)POPULATION[island];
      Total_V += popval[i];
    }

    if(Total_V == 0.0){
      free(repro);
      free(new_chrom);
      return -1;
    }

    repro[0] = 0.0;
    for(i=1; i<POPULATION[island]; i++){
      repro[i] = repro[i-1] + (double)popval[i-1]/(double)Total_V;
    }
    repro[i-1] = 1.0;

    for(i=0; i<POPULATION[island]/(3-CHROMTYPE); i++){
      do{
        rand_repro1 = rnd();
        rand_repro2 = rnd();

        for(parent1=0; parent1<POPULATION[island]; parent1++){
          if(repro[parent1] > rand_repro1)
```

```
            break;
          }
          for(parent2=0; parent2<POPULATION[island]; parent2++){
            if(repro[parent2] > rand_repro2)
              break;
          }
        } while((AUTOGAMY == FALSE) && (parent1 == parent2));

        if(CHROMTYPE == HAPLOID){                                    /* 一倍体 */
          if(SEX == TRUE){                                           /* 有性   */
            crossing_over_haploid(island, parent1-1, parent2-1,
                                  i, i+POPULATION[island]/2, machine, new_mac);
          }else{                                                     /* 無性   */
            machine_cpy(&new_mac[i], &machine[parent1-1]);
            machine_cpy(&new_mac[i+POPULATION[island]/2], &machine[parent2-1]);
          }
        }else{                                                       /* 二倍体 */
          if(SEX == TRUE){                                           /* 有性   */
            crossing_over_diploid(island, parent1-1, parent2-1,
                                  i, machine, new_mac, new_chrom);
          }else{                                                     /* 無性   */
            machine_cpy(&new_mac[i], rnd() > 0.5 ? &machine[parent1-1] :
                        &machine[parent2-1]);
          }
        }
      }
      for(i=0; i<POPULATION[island]; i++)
        machine_cpy(&machine[i], &new_mac[i]);              /* 遺伝子のコピー */

      free(repro);
      free(new_chrom);

      return 0;
}

/*****************************
        一倍体の交差
*****************************/
void crossing_over_haploid(island, parent1, parent2, descendant1, descendant2,
                           machine, new_mac)
int     island;
int     parent1, parent2, descendant1, descendant2;
MACHINE *machine, *new_mac;
{
      int     i=1, j;
                                                          /* 遺伝子のコピー      */
      machine_cpy(&new_mac[descendant1], &machine[parent1]);
      machine_cpy(&new_mac[descendant2], &machine[parent2]);

      for(j=0; j<BIT_LENGTH; j++){
        if(rnd() < CROSSOVER[island]){
          i = 1 - i;
          COUNT_CR++;
        }
                                                          /* ビット列の書き出し */
        BitWrite(new_mac[descendant1].chrom1, j,
                 BitRead(machine[(i ? parent1 : parent2)].chrom1, j));
        BitWrite(new_mac[descendant2].chrom1, j,
                 BitRead(machine[(i ? parent2 : parent1)].chrom1, j));
      }
                                                          /* 染色体のコピー      */
      chrom_cpy(new_mac[descendant1].chrom2, new_mac[descendant1].chrom1);
      chrom_cpy(new_mac[descendant2].chrom2, new_mac[descendant2].chrom1);
}

/*****************************
```

```
        二倍体の交差
****************************/
void crossing_over_diploid(island, parent1, parent2, descendant, machine,
                           new_mac, new_chrom)
int     island;
int     parent1, parent2, descendant;
MACHINE *machine, *new_mac;
char    *new_chrom;
{
    int     i=1, j;
    int     cross = 0;
    int     which_chrom;
                                                    /* 遺伝子のコピー      */
    machine_cpy(&new_mac[(rnd() > 0.5) ? parent1 : parent2],
                &machine[(rnd() > 0.5) ? parent1 : parent2]);

    which_chrom = (rnd() > 0.5);
    for(j=0; j<BIT_LENGTH; j++){
      if(rnd() < CROSSOVER[island]){
        i = 1 - i;
        cross++;
      }

      if(which_chrom)                               /* ビット列の書き出し */
        BitWrite(new_chrom, j, BitRead((i ? machine[parent1].chrom1 :
                                        machine[parent1].chrom2), j));
      else
        BitWrite(new_chrom, j, BitRead((i ? machine[parent1].chrom2 :
                                        machine[parent1].chrom1), j));
    }
                                                    /* 染色体のコピー      */
    chrom_cpy(new_mac[descendant].chrom1, new_chrom);

    i = 1;
    which_chrom = (rnd() > 0.5);
    for(j=0; j<BIT_LENGTH; j++){
      if(rnd() < CROSSOVER[island]){
        i = 1 - i;
        cross++;
      }

      if(which_chrom)
        BitWrite(new_chrom, j, BitRead((i ? machine[parent2].chrom1 :
                                        machine[parent2].chrom2), j));
      else
        BitWrite(new_chrom, j, BitRead((i ? machine[parent2].chrom2 :
                                        machine[parent2].chrom1), j));
    }
    chrom_cpy(new_mac[descendant].chrom2, new_chrom);
    COUNT_CR += (int)(cross/2.0 + 0.5);
}

/****************************
        適合度の計算
****************************/
void evaluate(island, generation, machine, rg)
int     island;
int     generation;
MACHINE *machine;
int     *rg[4];
{
    int    pnum;
    int    i, flag=0;
    int    j;
    char   *chrom=(char *)calloc(BIT_LENGTH/8+(BIT_LENGTH%8!=0),sizeof(char));
    static int  k=1;
```

```
    if(chrom == NULL){
      fprintf(stderr, "Can't allocate memory\n");
      exit(-1);
    }

    if(FLUCT){                                          /* 環境変動有      */
      if(generation%PERIOD == 0) k = 1 - k;
    }

    for(pnum=0; pnum<POPULATION[island];){
      get_prm(island, machine, pnum);                   /* データ取得      */
      flag = set_ga_map(prm);                           /* 範囲のチェック  */

      if(flag == 1){
        break;
      }else{                                            /* 評価関数        */
        if((setCpD(&value[pnum], rg)) == 1) value[pnum] = 0.0;
        val = value[pnum];
      }
      if(flag != 1)                                     /* 結果の比較      */
        comp_val(island, generation, val, pnum, machine);

      flag = 0;
      pnum++;
    }
    get_prm(island, machine, max_pnum);
    set_ga_map(prm);

    machine_sort(island, machine);                      /* 遺伝子のソート  */
    free(chrom);
}

/*****************************
        データ取得
*****************************/
void   get_prm(island, machine, pnum)
int     island;
MACHINE *machine;
int     pnum;
{
    int   i, j, k=0;
    int   start_y, end_x, end_y, end_z;
    char  *chrom=(char *)calloc(BIT_LENGTH/8+(BIT_LENGTH%8!=0),sizeof(char));

    for(i=0; i<PRM_NUM;){
      if(pnum > POPULATION[island]) break;
                                                        /* 染色体の合成    */
      PhenoType(chrom, machine[pnum].chrom1, machine[pnum].chrom2);

      start_y = 0;
      for(j=0; j<START_Y_GENE; j++){
        start_y += BitRead(chrom, j+i/4*28) * START_Y[j];
      }
      machine[pnum].prm[i] = start_y;
      if(i == 0) prm[i] = (float)start_y/128.0*0.6+0.1;
      else       prm[i] = (float)start_y/128.0*0.6+0.1;
      i++;

      end_x = 0;
      for(j=START_Y_GENE; j<(START_Y_GENE+END_X_GENE); j++){
        end_x += BitRead(chrom, j+i/4*28) * END_X[j-START_Y_GENE];
      }
      machine[pnum].prm[i] = end_x;
      if(i == 1) prm[i] = (float)end_x/128.0*0.3+0.3;
      else       prm[i] = (float)end_x/128.0*0.8+0.2;
```

```
      i++;

      end_y = 0;
      for(j=(START_Y_GENE+END_X_GENE); j<(START_Y_GENE+END_X_GENE+END_Y_GENE);j++){
        end_y += BitRead(chrom, j+i/4*28) * END_Y[j-(START_Y_GENE+END_X_GENE)];
      }
      machine[pnum].prm[i] = end_y;
      prm[i] = (float)end_y/128.0*90.0;
      i++;

      end_z = 0;
      for(j=(START_Y_GENE+END_X_GENE+END_Y_GENE);
          j<(START_Y_GENE+END_X_GENE+END_Y_GENE+END_Z_GENE);j++){
        end_z += BitRead(chrom, j+i/4*28) * END_Z[j-(START_Y_GENE+END_X_GENE+END_Y_GE
NE)];
      }
      machine[pnum].prm[i] = end_z;
      if(i == 3) prm[i] = (float)end_z/128.0*360.0;
      else       prm[i] = (float)end_z/128.0*90.0+90.0*k++;
      i++;
    }
}

/*****************************
        結果の比較
*****************************/
void comp_val(island, generation, val, pnum, machine)
int     island;
int     generation;
float   val;
int     pnum;
MACHINE *machine;
{
    int     j;

    if(gen == generation){
      if(val >= max_value){
        max_value = val;
        max_pnum  = pnum;
        machine_cpy(pmax_mac[0], &machine[max_pnum]);   /* 遺伝子のコピー */
      }
    }else{
      fprintf(fpp1, "<%d> %f (%d)\n", gen, max_value, max_pnum);

      if(max_value >= gen_max_value){
        gen_max_value = max_value;
        gen_max_pnum  = max_pnum;
        gen_max       = gen;
        machine_cpy(gmax_mac[0], pmax_mac[0]);
      }
      max_value = 0.0;
      gen = generation;

      if(val >= max_value){
        max_value = val;
        max_pnum  = pnum;
      }
    }
}

/*****************************
      遺伝子のソート
*****************************/
void machine_sort(island, machine)
int     island;
MACHINE *machine;
```

```
{
    int       i, mac_sum1, mac_sum2, j, k;
    float     wvalue[POP_NUM*2];
    char      **chrom = (char **)calloc(POPULATION[island]*2, sizeof(char *));

    if(chrom == NULL){
      fprintf(stderr, "Can't allocate memory\n");
      exit(-1);
    }
    mac_sum1 = mac_sum2 = 0;

    for(i=0; i<POPULATION[island]; i++){
      if(value[i] != 0.0){
        chrom[mac_sum1] = machine[i].chrom1;
        wvalue[mac_sum1] = value[i];
        mac_sum1++;
      }
    }

    for(i=0; i<POPULATION[island]; i++){
      if(value[i] != 0.0){
        chrom[mac_sum1+mac_sum2] = machine[i].chrom2;
        wvalue[mac_sum1+mac_sum2] = value[i];
        mac_sum2++;
      }
    }

    if(mac_sum1+mac_sum2 == 0){
      Class = 0;
      free(chrom);
      return;
    }
    chromsort(wvalue, chrom, 0, mac_sum1+mac_sum2-1); /* 染色体のソート    */
    chromsort2(wvalue, chrom, 0, mac_sum1+mac_sum2-1);

    for(i=0, j=0; i<mac_sum1+mac_sum2; i++){
      value[j] = wvalue[i];

      machine[j].chrom1 = chrom[i++];
      machine[j].chrom2 = chrom[i];
      j++;
    }

    Class = 1;
    for(i=1; i<mac_sum1+mac_sum2; i++){
      if(memcmp(chrom[i], chrom[i-1], BIT_LENGTH/8+(BIT_LENGTH%8!=0)) != 0)
        Class++;
    }
    free(chrom);
}

/*****************************
       染色体のソート
*****************************/
void chromsort2(r, v, left, right)
char  **v;
int   left, right;
float r[POP_NUM];
{
    int     i, j;

    for(i=left, j=right; i<=(left+right)/2; i++, j--){
      r_swap(r, i, j);                              /* 計算結果のソート  */
      v_swap(v, i, j);                              /* 染色体のソート    */
    }
}
```

```
/*****************************
        染色体のソート
*****************************/
void chromsort(r, v, left, right)
char  **v;
int   left, right;
float r[POP_NUM];
{
    int     i, last;

    if(left >= right) return;

    v_swap(v, left, (left+right)/2);                    /* 染色体のソート    */
    r_swap(r, left, (left+right)/2);                    /* 計算結果のソート  */
    last = left;

    for(i=left+1; i<=right; i++){
      if(r[i] >= r[left]){
        r_swap(r, ++last, i);
        v_swap(v, last, i);
      }
    }
    v_swap(v, left, last);
    r_swap(r, left, last);

    chromsort(r, v, left, last-1);                      /* 染色体のソート    */
    chromsort(r, v, last+1 , right);
}

/*****************************
        染色体のソート
*****************************/
v_swap(v, i, j)
char **v;
int  i,j;
{
    char     *temp;

    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}

/*****************************
      計算結果のソート
*****************************/
r_swap(r, i, j)
float r[POP_NUM];
int   i,j;
{
    float     temp;

    temp = r[i];
    r[i] = r[j];
    r[j] = temp;
}

/*****************************
        突然変異
*****************************/
void mutation(island, machine)
int     island;
MACHINE *machine;
{
    int     i, j;
```

```
    for(i=0; i<POPULATION[island]; i++){
      if(META_MU == ON){
        for(j=0; j<BIT_LENGTH; j++){

          if(rnd() < 1.0/machine[i].mu){            /* ビット列の書き出し */
            mu_BitWrite(machine[i].mu, j, 1-mu_BitRead(machine[i].mu, j));
          }
        }
      }
      for(j=0; j<BIT_LENGTH; j++){
        if(rnd() < 1.0/machine[i].mu){
          BitWrite(machine[i].chrom1, j, 1-BitRead(machine[i].chrom1, j));
          COUNT_MU++;
        }
      }
      if(CHROMTYPE == DIPLOID){
        for(j=0; j<BIT_LENGTH; j++){
          if(rnd() < 1.0/machine[i].mu){
            BitWrite(machine[i].chrom2, j, 1-BitRead(machine[i].chrom2, j));
            COUNT_MU++;
          }
        }
      }else                                          /* 染色体のコピー     */
        chrom_cpy(machine[i].chrom2, machine[i].chrom1);
    }
}

/*****************************
       初期集団の生成
*****************************/
void initialize_proc(rg)
int *rg[4];
{
    FILE     *fp;

    knap_item_load();                        /* "item.dat"ファイルの読み込み */
    fp = f_parameter();                      /* "param"ファイルの読み込み    */
    make_islands(fp);                        /* 突然変異率と交差率の取得     */
    make_machines(rg);                       /* 個体の生成                   */
    fclose(fp);
}

 *****************************
          終了処理
*****************************/
void done_proc()
{
    int     i;

    for(i=0; i<ISLAND_NUM; i++){
      free(mac[i][0].chrom1);
      free(mac[i][0].chrom2);
      free(mac[i]);
      free(tmp_mac[i]);
      free(pmax_mac[i]);
      free(gmax_mac[i]);
    }
    free(mac);
    free(tmp_mac);
    free(pmax_mac);
    free(gmax_mac);
    free(POPULATION);
    free(MUTATION);
    free(CROSSOVER);
}
```

```
/*****************************
"item.dat"ファイルの読み込み
*****************************/
void knap_item_load()
{
    FILE     *fp1, *fp2, *fp3, *fp4;
    int      i;

    if(((fp1 = fopen("item1.dat", "r")) == NULL) ||
       ((fp2 = fopen("item2.dat", "r")) == NULL) ||
       ((fp3 = fopen("item3.dat", "r")) == NULL) ||
       ((fp4 = fopen("item4.dat", "r")) == NULL)){
      fprintf(stderr, "can't open item.dat\n");
      exit(-1);
    }
    fscanf(fp1, "Gene = %d\n", &START_Y_GENE);
    fscanf(fp2, "Gene = %d\n", &END_X_GENE);
    fscanf(fp3, "Gene = %d\n", &END_Y_GENE);
    fscanf(fp4, "Gene = %d\n", &END_Z_GENE);

    START_Y = (int *)calloc(START_Y_GENE, sizeof(int));
    END_X   = (int *)calloc(END_X_GENE,   sizeof(int));
    END_Y   = (int *)calloc(END_Y_GENE,   sizeof(int));
    END_Z   = (int *)calloc(END_Z_GENE,   sizeof(int));

    fscanf(fp1, "\nSTART_Y-ITEM:\n");
    fscanf(fp2, "\nEND_X-ITEM:\n");
    fscanf(fp3, "\nEND_Y-ITEM:\n");
    fscanf(fp4, "\nEND_Z-ITEM:\n");

    for(i=0; i<START_Y_GENE; i++) fscanf(fp1, "%d", &START_Y[i]);
    for(i=0; i<END_X_GENE;   i++) fscanf(fp2, "%d", &END_X[i]);
    for(i=0; i<END_Y_GENE;   i++) fscanf(fp3, "%d", &END_Y[i]);
    for(i=0; i<END_Z_GENE;   i++) fscanf(fp4, "%d", &END_Z[i]);

    fscanf(fp1, "\nStart_y_maximum = %d\n", &START_Y_MAXIMUM);
    fscanf(fp1, "\nStart_y_minimum = %d\n", &START_Y_MINIMUM);
    fscanf(fp2, "\nEnd_x_maximum = %d\n",   &END_X_MAXIMUM);
    fscanf(fp2, "\nEnd_x_minimum = %d\n",   &END_X_MINIMUM);
    fscanf(fp3, "\nEnd_y_maximum = %d\n",   &END_Y_MAXIMUM);
    fscanf(fp3, "\nEnd_y_minimum = %d\n",   &END_Y_MINIMUM);
    fscanf(fp4, "\nEnd_z_maximum = %d\n",   &END_Z_MAXIMUM);
    fscanf(fp4, "\nEnd_z_minimum = %d\n",   &END_Z_MINIMUM);

    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
}

/*****************************
  "param"ファイルの読み込み
*****************************/
FILE * f_parameter()
{
    int      i, pop;
    FILE     *fp;

    if((fp = fopen("param", "r")) == NULL){
      fprintf(stderr, "Can't open this file");
      exit(-1);
    }
    fscanf(fp, "Max generation =%d\n", &GENERATION_MAX);   /* 世代数       */
    fscanf(fp, "Island num = %d\n", &ISLAND_NUM);          /* 島の数       */
```

```
    if(ISLAND_NUM != 1)
      fscanf(fp, "Migration rate = %le\n", &MIGRATION);     /* 移住率       */

    fscanf(fp, "Population of island = %d\n", &pop);        /* 個体数       */
    if(pop % 2){
      fprintf(stderr, "Bad number this Population\n");
      exit(-1);
    }
    fscanf(fp, "Sexual = %d\n", &SEX);                      /* 有性or無性   */

    if(SEX) fscanf(fp, "Autogamy = %d\n", &AUTOGAMY);       /* 自家受精     */
    else    AUTOGAMY = ON;

    fscanf(fp, "ChromType = %d\n", &CHROMTYPE);             /* 染色体数     */
    fscanf(fp, "Meta mutation = %d\n", &META_MU);           /* メタ突然変異*/
    fscanf(fp, "Generation step = %d\n", &GENERATION_STEP);/* 何世代毎     */
    fscanf(fp, "Fluctuation = %d\n", &FLUCT);               /* 環境変動     */

    if(FLUCT){
      fscanf(fp, "High-weight limit =%d\n", &HIGH_W);
      fscanf(fp, "Low-weight limit =%d\n", &LOW_W);
      fscanf(fp, "Period =%d\n", &PERIOD);
    }

    POPULATION = (int *)calloc(ISLAND_NUM, sizeof(int));
    for(i=0; i<ISLAND_NUM; i++) POPULATION[i] = pop;

    return fp;
}

/*****************************
  突然変異率と交差率の取得
*****************************/
void make_islands(fp)
FILE *fp;
{
    int     i, is;

    MUTATION  = (double *)calloc(ISLAND_NUM, sizeof(double));/* 突然変異率 */
    CROSSOVER = (double *)calloc(ISLAND_NUM, sizeof(double));/* 交差率     */

    for(i=0; i<ISLAND_NUM; i++){
      if(fp == NULL){
        fprintf(stderr, "No.%2d island's mutation rate = ", i+1);

        if(SEX == TRUE){                                    /* 有性       */
          fprintf(stderr, "No.%2d island's crossing-over rate = ", i+1);
          scanf("%le", &CROSSOVER[i]);
        }
      }else{
        fscanf(fp, "No.%d island's mutation rate =%lf\n", &is, &MUTATION[i]);
        if(SEX == ON)                                       /* 有性       */
          fscanf(fp, "No.%d island's crossing-over rate =%lf\n",
                 &is, &CROSSOVER[i]);
      }
    }
}

/*****************************
         個体の生成
*****************************/
void make_machines(rg)
int *rg[4];
{
    int     i, j;
```

```
    mac      = (MACHINE **)calloc(ISLAND_NUM, sizeof(MACHINE *));
    tmp_mac  = (MACHINE **)calloc(ISLAND_NUM, sizeof(MACHINE *));
    pmax_mac = (MACHINE **)calloc(ISLAND_NUM, sizeof(MACHINE *));
    gmax_mac = (MACHINE **)calloc(ISLAND_NUM, sizeof(MACHINE *));

    for(i=0; i<ISLAND_NUM; i++){
      mac[i]      = (MACHINE *)calloc(POPULATION[i], sizeof(MACHINE));
      tmp_mac[i]  = (MACHINE *)calloc(POPULATION[i], sizeof(MACHINE));
      pmax_mac[i] = (MACHINE *)calloc(POPULATION[i], sizeof(MACHINE));
      gmax_mac[i] = (MACHINE *)calloc(POPULATION[i], sizeof(MACHINE));

      if(mac[i] == NULL || tmp_mac[i] == NULL ||
         pmax_mac[i] == NULL || gmax_mac[i] == NULL){
        fprintf(stderr, "Can't allocate memory\n");
        exit(-1);
      }
      sub_create(mac[i],      POPULATION[i]);          /* 遺伝子の領域確保 */
      sub_create(tmp_mac[i],  POPULATION[i]);
      sub_create(pmax_mac[i], POPULATION[i]);
      sub_create(gmax_mac[i], POPULATION[i]);
      create(i, 0, mac[i], rg);                        /* 遺伝子の生成     */
    }
}

/*****************************
        遺伝子の生成
*****************************/
void create(island, generation, machine, rg)
int     island;
int     generation;
MACHINE *machine;
int     *rg[4];
{
    int     i, j;

    srand(time(0));

    for(i=0; i<POPULATION[island]; i++){
      for(j=0; j<BIT_LENGTH; j++){                     /* ビット列の書き出し */
        BitWrite(machine[i].chrom1, j, (rnd()<0.5));
      }
      if(CHROMTYPE == DIPLOID)                         /* 二倍体           */
        for(j=0; j<BIT_LENGTH; j++)
          BitWrite(machine[i].chrom2, j, (rnd()<0.5));
                                                       /* 一倍体           */
      else                                             /* 染色体のコピー   */
        chrom_cpy(machine[i].chrom2, machine[i].chrom1);

      machine[i].mu = (unsigned long)(1.0/MUTATION[island]);
    }
    evaluate(island, generation, machine, rg);         /* 適合度の計算     */
}

/*****************************
      遺伝子の領域確保
*****************************/
void sub_create(machine, num)
MACHINE *machine;
int     num;
{
    int     i;
    int     c_length = BIT_LENGTH/8+(BIT_LENGTH%8!=0);

    machine[0].chrom1 = (char *)calloc(c_length * num, sizeof(char));
    machine[0].chrom2 = (char *)calloc(c_length * num, sizeof(char));
```

```
    if(machine[0].chrom1 == NULL || machine[0].chrom2 == NULL){
      fprintf(stderr, "Can't allocate memory\n");
      exit(-1);
    }

    for(i=1; i<num; i++){
      machine[i].chrom1 = machine[0].chrom1 + c_length * i;
      machine[i].chrom2 = machine[0].chrom2 + c_length * i;
    }
}

/****************************
          絶滅
****************************/
void extermination(island, generation, machine, rg)
int     island;
int     generation;
MACHINE *machine;
int     *rg[4];
{
    puts("Exterminate !!");
    create(island, generation, machine, rg);          /* 遺伝子の生成      */
}

/****************************
     ビット列の書き出し
****************************/
int BitWrite(chrom, pos, val)
char *chrom;
int  pos, val;
{
    char     mask, buff;

    mask = (char) (val << (pos % 8));
    buff = (char) (chrom[pos / 8] & ~(1 << (pos % 8)));

    chrom[pos / 8] = (char) (buff | mask);

    return val;
}

/****************************
     ビット列の読み出し
****************************/
int BitRead(chrom, pos)
char *chrom;
int  pos;
{
    int     val;

    val = (chrom[pos / 8] >> (pos % 8)) & 1;
    return val;
}

/****************************
      遺伝子のコピー
****************************/
void machine_cpy(m1, m2)
MACHINE *m1, *m2;
{
    int     i;

    m1->mu      = m2->mu;
    for(i=0; i<PRM_NUM; i++) m1->prm[i] = m2->prm[i];
```

```
    memcpy(m1->chrom1, m2->chrom1, BIT_LENGTH / 8 + (BIT_LENGTH % 8 != 0));
    memcpy(m1->chrom2, m2->chrom2, BIT_LENGTH / 8 + (BIT_LENGTH % 8 != 0));
}

/*****************************
       染色体のコピー
*****************************/
void chrom_cpy(c1, c2)
char *c1, *c2;
{
    memcpy(c1, c2, BIT_LENGTH/8+(BIT_LENGTH%8!=0));
}

/*****************************
       染色体の合成
*****************************/
void PhenoType(dest, c1, c2)
char *dest, *c1, *c2;
{
    int     i;

    for(i=0; i<BIT_LENGTH/8+(BIT_LENGTH%8!=0); i++)
      dest[i] = c1[i] | c2[i];
}

/*****************************
       範囲のチェック
*****************************/
int set_ga_map(float prm[20])
{
    int     i, ref1=0;

    SetSmap(prm[0], prm[1], prm[2], prm[3]);          /* 幹データセット    */

    for(i=1; i<5; i++){                               /* 枝データセット    */
      SetBmap(i-1, prm[4*i], prm[4*i+1], prm[4*i+2], prm[4*i+3]);
    }
    return(ref1);
}

/*****************************
       幹データセット
*****************************/
void SetSmap(float SL, float SP, float SZ, float SY)
{
    float     SYrad, SZrad;

    stem[0].LengthRatio = SL;
    stem[0].StartLength = SP;
    stem[0].Rotation    = SY;
    stem[0].StemAngle   = SZ;

    SYrad = SY * radtodeg;
    SZrad = SZ * radtodeg;
                                          /* マトリックスに値をセットする */
    set_mapmatrix(&stem[0].map, SL,
        cosf(SZrad), cosf(SYrad), sinf(SZrad), sinf(SYrad), SP);
}

/*****************************
       枝データセット
*****************************/
void SetBmap(int bno,float BL,float BP,float BZ,float BY)
{
    float     BYrad, BZrad;
```

```
    branch[0][bno].Position    = BP;
    branch[0][bno].LengthRatio = BL;
    branch[0][bno].BranchAngle = BZ;
    branch[0][bno].Rotation    = BY;

    BYrad = BY * radtodeg;
    BZrad = BZ * radtodeg;
                                        /* マトリックスに値をセットする */
    set_mapmatrix(&branch[0][bno].map, BL,
                  cosf(BZrad), cosf(BYrad), sinf(BZrad), sinf(BYrad), BP);
}

/*****************************
          評価関数
*****************************/
static int setCpD(float *CpD, int *rg[4])
{
    float     Cang[2], Dang[2], SS, SZ, SY, SL, BZ, BY, BL, Gang[2];
    float     ratio;
    int       eno, gazo, i;
)
    /* Added by Kuwahara */
    float       Tang[2], gam = 0.5;

    SS = get_Sshiten();                         /* 始点位置を取得する        */
    get_SZSYSL(&SZ, &SY, &SL);                  /* 幹の形状データを取得する */
/*write_treedatafile2( stdout );*/

    for(gazo=0; gazo<2; gazo++){                /* 縮小画像データを取得する */
      get_reducedtree(rg[0], 0.0, 0.0, 0.0, 1.0, gazo, (!gazo ? 1 : 0) );
      argand(tree_region[gazo], rg[0], rg[2], N);       /* 論理積を求める */

      Cang[gazo] = (float)numregion(rg[2], N*N);
      Gang[gazo] = (float)numregion(rg[0], N*N);
      Tang[gazo] = (float)numregion(tree_region[gazo], N*N);
    }
    ratio = ( Gang[0] + Gang[1] ) / ( Tang[0] + Tang[1] );

    if( ratio == 0.0 )
        *CpD = 0.0;
    else
        *CpD = Cang[0]/sqrt(Tang[0]*Gang[0])+Cang[1]/sqrt(Tang[1]*Gang[1]);
    if( (ratio > 2.00) || (ratio < 0.50) ){
      return( 1 );
    }else{
      return( 0 );
    }
}
```