修　士　論　文

着物模様のためのデザイン支援ツールの提案

指導教官　　宮田一乗　教授

北陸先端科学技術大学院大学
知識科学研究科知識システム基礎学専攻

450020　鎌田 洋輔

審査委員：　宮田　一乗　教授（主査）
永井　由佳里　助教授
金井　秀明　助教授
佐藤　賢二　助教授

2006 年 2 月

Design Supporting Tool for Japanese Traditional Kimono Patterns

Yosuke Kamada

School of Knowledge Science,
Japan Advanced Institute of Science and Technology
March 2006

Keywords: Ornament Patterns, Japanese Kimono, generate algorithm, interactive design.

Japanese traditional kimonos have beautiful expressions about patterns and colors. It has been limited to design kimono patterns in craftsmen, because the user is required to have a proficient design sense for designing kimono patterns. Kimono patterns are used to make clothes or bags, and they have revived recently. This thesis extracts implicit rules of arrangement from kimono design, and proposes a design support system for novice users based on those rules.

This thesis focuses on plant patterns in *Kaga Yuzen*, in the means of difficulty covering all various kimono patterns. Furthermore, this paper only treats arrangements of kimono patterns.

To extract the arrangement rules from *Kaga Yuzen*, ten samples are randomly selected from the portfolio of *Kaga Yuzen*. This system is implemented in two common rules, "Flow" and "Hierarchy (layer structure)", which are extracted from ten samples.

The design support system consists of two elements; pattern generation algorithm and user interface. L-System is the algorithm for modeling real plants. As the kimono's plant patterns differ from the real plants, this paper improves the L-System to express ornamental plants. This thesis also proposes a user interface that makes it possible to design kimono pattern easily by considering the rule of *flow* and *hierarchy.*

概要

日本の伝統的な着物模様には，その独特の柄，色彩表現に魅了されるものが多々ある．しかし，その表現には高度なデザインセンスが必要であり，熟練した着物職人でしか描けないのが現状である．近年，着物模様は，洋服や鞄の柄に用いられており，模様の価値そのものが見直されてきている．以上のような背景から，本研究では，着物模様の暗黙的な規則性をシステムとして形式知化し，誰にでも簡単に模様を生成できるデザイン支援システムの実現を目指す．

着物模様には，様々な種類があり，そのすべてを網羅することは非常に困難であるため，ある程度の制約を設けることが必要であった．本研究では，対象とする着物を「加賀友禅」とし，その中でも植物模様とテーマ限定することにした．また，模様の配置のみに注目し，色彩の組み合わせは考慮しないことにした．

加賀友禅の模様から規則性を抽出するために，実際の職人から加賀友禅集をお借りした．そこから無作為に抜粋した 10 個のサンプル模様から加賀友禅模様の分析を行なった．加賀友禅模様の有する共通の規則性に，「流れ」と「階層性(レイヤー構造)」があることが分かった．ここで流れとは，一定の規則を持った模様の配置を指し，階層性とは，個々の模様に対して前景・背景などの区別があることを指す．この二つの規則を考慮して，デザイン支援システムを構築した．

デザイン支援システムは，大別して，模様の生成アルゴリズムおよび，ユーザインターフェイスの２つで構成される．植物模様の生成アルゴリズムには，L - System のアルゴリズムを利用した．L - System は実在する植物を CG で表現するために考案されたものであり，これを応用することで，模様らしい植物表現を実現することが可能となった．ユーザインターフェイスとしては，ユーザが複雑な操作をしなくても，模様における「流れ」・「階層性」を考慮して模様生成ができる，直感的なインターフェイスを考案した．

以上の構成要素を実装することで，着物模様を誰にでも簡単に作りだせるデザイン支援ツールを実現できた．

目　次

図　目　次

表　目　次

第　1　章
序論

本章では，はじめに研究の背景と目的および，アプローチについて述べ，つづいて研究対象の着物模様を概観する．

1.1　研究の背景と目的

日本の伝統的な着物模様には，その独特の柄や色彩表現に魅了されるものが多々あり，洋服や鞄の柄に用いられるなど，模様の価値そのものが見直されてきている．しかし，その表現には高度なデザインセンスを必要とし，熟練の着物職人のみが描くことができるのが現状である．このような着物模様をデザインセンスの乏しい人が描く場合，着物模様における樹木や花などの模様の配置や，構成要素である各種模様の表現など，様々なことを考慮しなければならないため，多大な労力と時間を要する．本研究では，そのような労力を省き，着物模様の構成などに関する知識をパターン化し，模様デザインを支援するシステムの実現を目的とする．

1.2　研究のアプローチ

本研究では，着物模様の暗黙的な規則性を抽出して形式知化（パターン化）し，着物模様のデザインを支援するシステムの実現を目指した．このシステムの実現に際して，以下に述べる研究のアプローチをとった．

1）実際の着物模様のサンプル集から暗黙的な規則性を抽出する．

2）樹木の模様を手で描くような手間のかかる作業を減らすため，樹木の模様

を自動生成するアルゴリズムを開発する．

3）細かなデザイン作業を省いた，簡単な操作で模様を生成するユーザインターフェイスを開発する．

1.3　研究の対象

研究対象の着物模様は「加賀友禅」の留袖に描かれた模様に限定する．留袖では，図 1.1 に示すように，着物の下の部分にのみ模様が描かれる．着物模様の対象を加賀友禅の留袖に限定しても，すべての題材を網羅することは非常に困難であるため，植物を題材としたものに限定する．

また，本研究では模様の配置のみに注目し，色彩の組み合わせは考慮しない．色彩を考慮しない理由として，着物の独特の色彩をディスプレイで忠実に表現することが非常に困難であることや，多種多彩な色彩の組み合わせから共通の規則性を抽出することが困難であることなどが挙げられる．

図 1.1　留袖の友禅模様

1.4　関連研究

デザイン支援ツールの関連研究として，以下に述べる 2 つの項目について調査を行なった.

1)模様生成のためのアルゴリズムの研究

CG 分野における模様生成アルゴリズムの研究例として，装飾模様の自動生成法[1]が挙げられる．この研究は，単純な生成規則から均一な装飾模様を自動生成する手法を用いている．このため，模様生成にユーザの意思が介入できず，着物模様固有の複雑な模様をデザインすることは困難である．また，本研究の表現対象である植物のグラフィック表現の研究例として，L-System[2]が挙げられる．これは，植物の複雑な形状を単純な生成規則の組み合わせで表現したものである．しかし，リアルな植物表現の生成を目指したものであるため，着物模様における独特の枝や幹の誇張した表現までは考慮していない.

2)模様生成のためのユーザインターフェイスの研究

デザインツールにおけるユーザインターフェイスの代表例に，Adobe 社の Illustrator が挙げられる．このツールには，デザインセンスのある人にとって自由な表現が可能な機能が十分に備わっている．しかし，模様をどのように配置すればよいかをアドバイスするような，デザインを支援する機能は備わっていない．そのため，デザインセンスの乏しい人にとっては，樹木の模様を描くだけでも操作が非常に困難であり，使い勝手のよいものとは言い切れない．また，植物をインタラクティブに操作し簡単に生成できる研究 [3]も報告されている．この研究では，リアルな植物を生成するための支援ツールの開発が目的であり，２次元の着物模様を生成するためのものではない.

本研究では，1)で述べた L-System を利用して，着物模様特有の植物表現を可能とする樹木アルゴリズムの実現を目指す．また，着物模様の暗黙的な規則性を考慮しながら簡単に模様生成が行える，デザイン支援を目的としたユーザインターフェイスの開発を目指す.

第　2　章

配置規則の抽出

本章では，実際の留袖模様の画集（サンプル模様集）から，共通する規則性を抽出する 2 つの手法について述べる．なお，本章で述べるサンプル模様集とは，図 8.1～10 に示す 10 枚の画像を指す．

2.1　抽出手法 1

対象とする着物模様に共通の規則性を抽出する手法として，模様全体の特徴を把握する比較法を提案する．比較法として，1）模様の中央付近の流れを表現した「中心線」，2）上部の流れを表現した「上領域線」，3）下部の流れを表現した「下領域線」の，領域ごとの比較法を提案する．

2.1.1　　中心線比較

図 2.1 は，サンプル模様集から模様の中心線を抽出した図を示す．

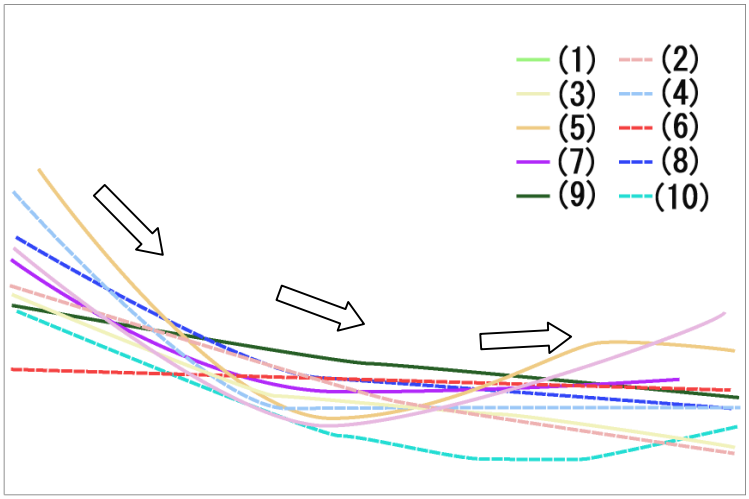


図 2.1　中心線比較

着物の各模様は，図 2.1 中の矢印で示すように，左上から，なだらかに右下方へ向かう流れを描いており，共通した流れが存在すると考えられる．流れの具体例を，図 2.2 に示す．

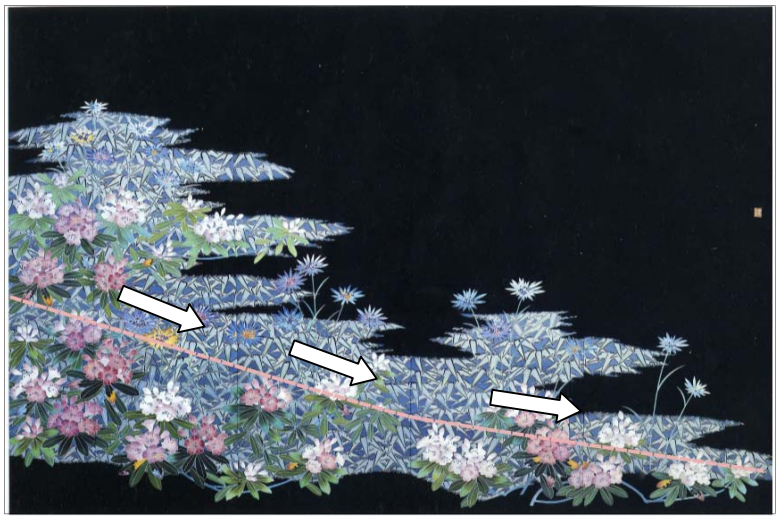


図 2.2　図 8.2 の中心線の流れ

2.1.2　　上領域線比較

図 2.3 は，サンプル模様集における模様の上領域線を抽出した図を示す．

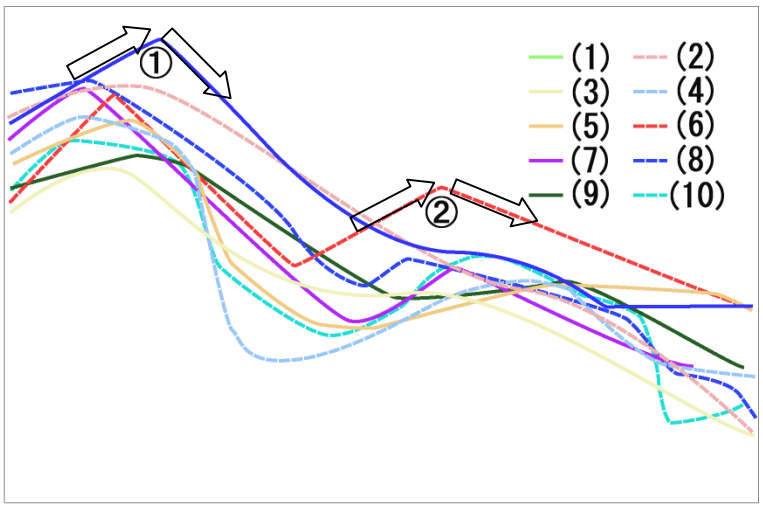


図 2.3　上領域比較

図 2.3 から，各模様には，2つの大きな特徴点を有した共通した流れがあることが分かる．ひ

とつは，模様の左側にある山型の部分(図中①)である．もうひとつは，模様の右側にある山型の部分(図中②)である．模様の流れは，①から徐々に右下になだらかに下り，②へと続いている．流れの具体例を，図 2.4 に示す．

図 2.4　図 8.3 の上領域線の流れ

2.1.3　　下領域線比較

図 2.5 は，サンプル模様集における模様の下領域線を抽出した図を示す．

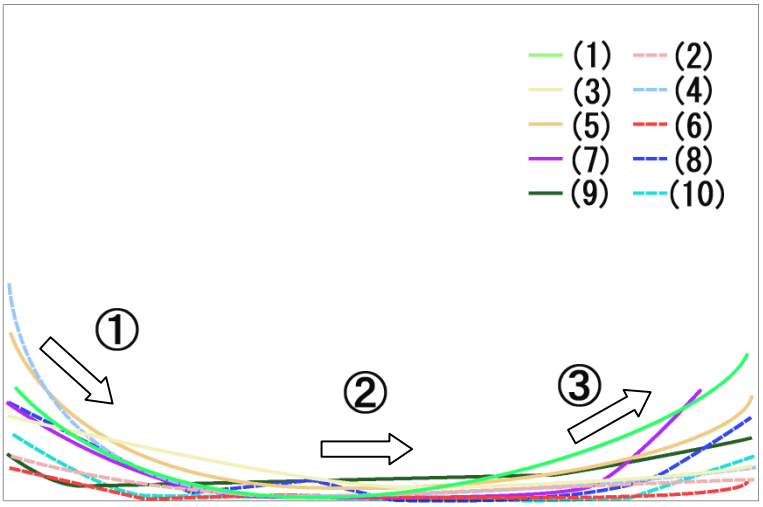


図 2.5　下領域比較

各模様には，図 2.5 に示すように，模様の左部分からの急な下り（図中①），中央部における平行した流れ（図中②），右部分での上り（図中③）の共通した流れが見受

けられる．流れの具体例を図 2.6 に示す．

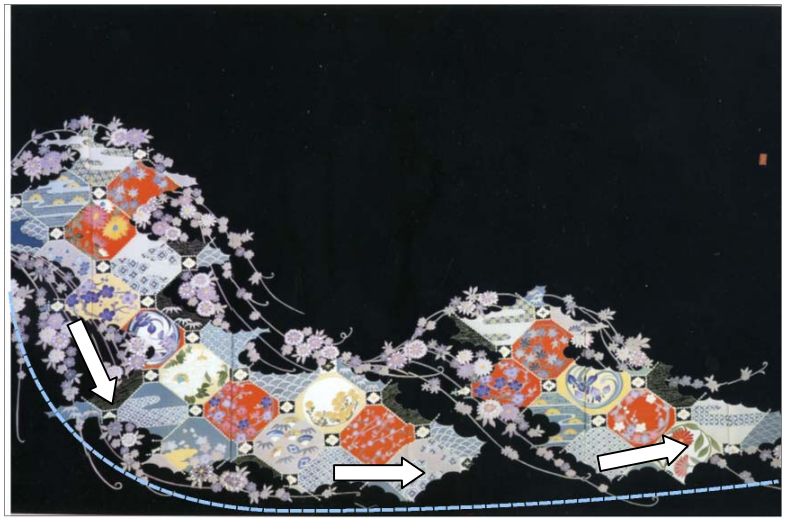


図 2.6　図 8.4 の下領域線の流れ

2.1.4　抽出手法 1 のまとめ

以上の結果から，着物模様には「流れ」に対する共通性があることがわかり，「流れ」は着物模様における重要なデザイン要素であるといえる．

以上で述べた流れは，各模様の「全体の流れ」であり，模様の中にある樹木の模様，花の模様，葉の模様などの「個別の流れ」を分析したものではない．そこで，さらに詳しく，各模様の「個別の流れ」を分析することにする．以下，2.1 節で述べた「流れ」を「全体の流れ」とし，2.2 節で後述する流れを「個別の流れ」と定義する．

2.2　抽出手法 2

「個別の流れ」を分析するにあたり，模様の配置構造を分析する上で重要であると考えられる「階層性（レイヤー構造）」に関しても注目する．本節で述べる「個別の流れ」とは，各樹木の模様の配置における規則性のことを指し，「階層性」とは，各模様の配置における前景・背景などの前後関係のことを指す．

2.2.1　図 8.1 からの抽出

・ 個別の流れ

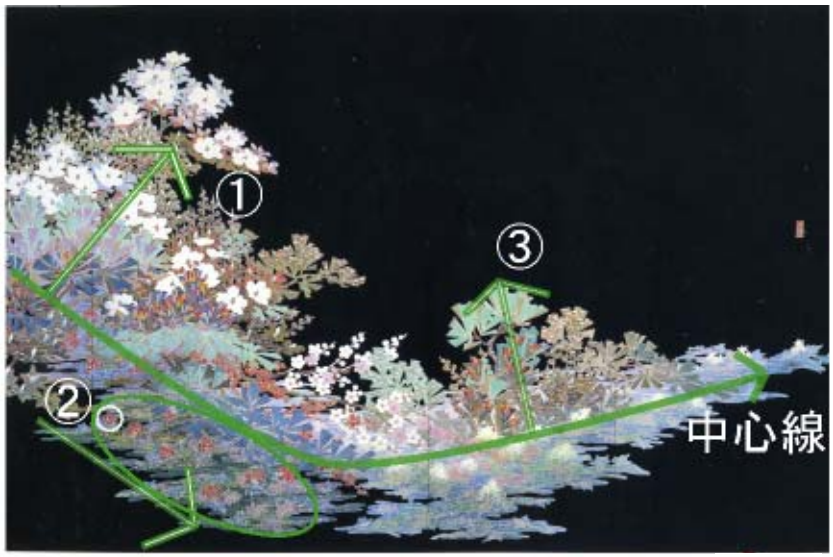


図 2.7　図 8.1 の個別の流れ

図 2.7 の①～③の部分における，個別の流れの特徴を表 2.1 に示す．これらの特徴から，中心線を基準として，樹木の模様の配置が決定されているといえる．

表 2.1　図 8.1 の個別の流れの特徴

位置	特徴
①	中心線から垂直方向に樹木の模様を配置
②	中心線に沿って白円で示した赤い花の模様を配置
③	中心線から垂直方向に樹木の模様を配置

・ 階層性

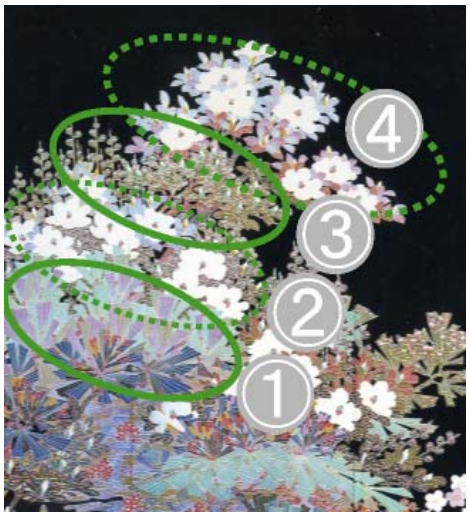


表 2.2 図 8.1 の階層性の特徴

位置	特徴
①	葉の模様を最前面に配置
②	白い花の模様を①の背後に配置
③	葉の模様を②の背後に配置
④	白い花の模様を③の背後に配置

図 2.8　図 8.1 の階層性

図 2.8 の①～③の部分における，階層性の特徴を表 2.2 に示す．これらの特徴から，図 8.1 には 4 層の階層があるといえる．

2.2.2　　図 8.2 からの抽出

・ 個別の流れ

図 2.9　図 8.2 の個別の流れ

図 2.9 に示すように，この模様には中心線 1 で表された全体の流れ以外に，中心線 2 で示す花の模様の個別の流れが存在する．表 2.3 に，中心線 2 についての個別の流れの特徴を示す．これらの特徴から，花の模様は中心線 2 を軸とした配置であることが分かる．

表 2.3　図 8.2 の個別の流れの特徴

位置	特徴
①	中心線 2 に沿って花の模様を上から下に配置
②	中心線 2 に沿って花の模様を左から右に配置

・ **階層性**

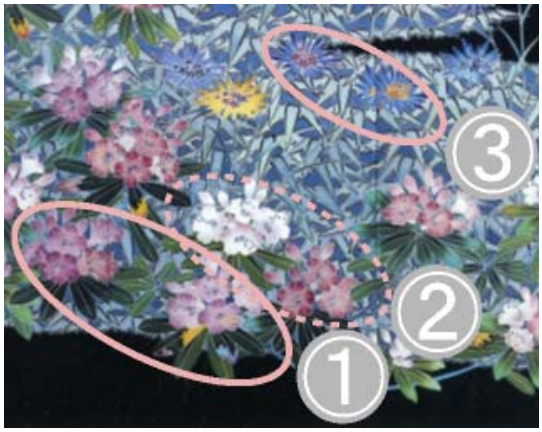


図 2.10　図 8.2 の階層性

表 2.4 図 8.2 の階層性の特徴

位置	特徴
①	花の模様を最前面に配置
②	花の模様を①の背後に配置
③	青い花の模様を②の背後に配置

図 2.10 の①～③において，表 2.4 に示した階層性があることが分かった．これらの特徴から，図 8.2 は 3 層の階層があるといえる．

2.2.3　図 8.3 からの抽出

・ 個別の流れ

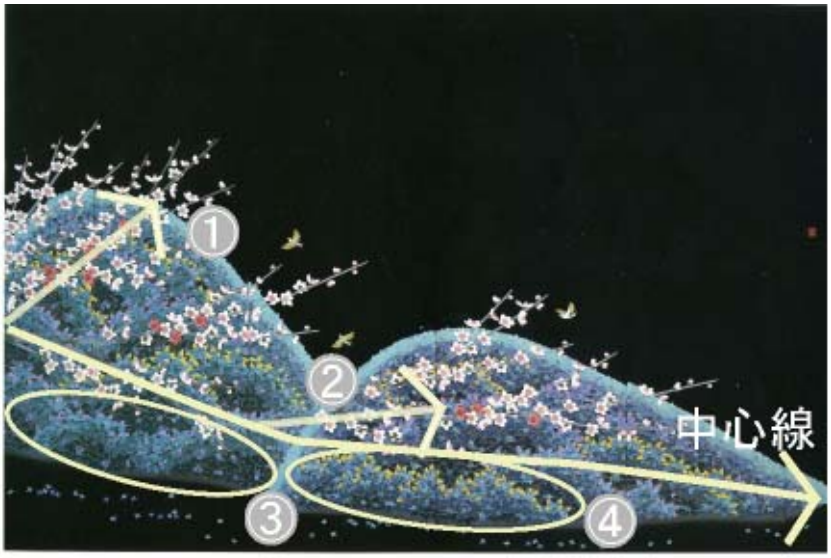


図 2.11　図 8.3 の個別の流れ

図 2.11 の①～④において，表 2.5 で示した特徴があることが分かった．これらの特徴から，図 8.3 は中心線を軸とした模様配置である．

表 2.5　図 8.3 の個別の流れの特徴

位置	特徴
①	樹木の模様を，中心線を軸として右上に放射状に配置
②	樹木の模様を，中心線を軸として右上に放射状に配置
③	青色の葉の模様を，中心線に沿って左から右に並んで配置
④	黄色の葉の模様を，中心線に沿って左から右に並んで配置

・ 階層性

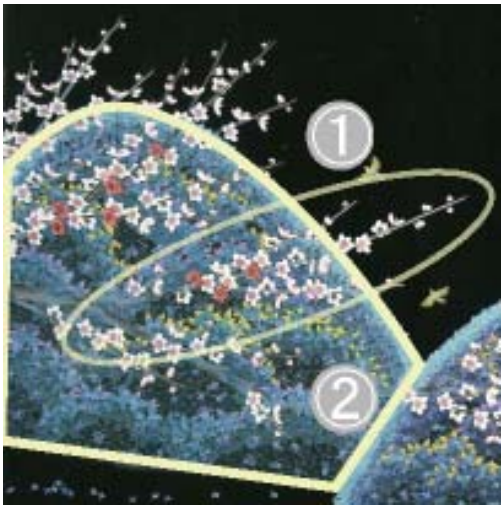


図 2.12　図 8.3 の階層性

表 2.6 図 8.3 の階層性の特徴

位置	特徴
①	樹木の模様を最前面に配置
②	山形の模様を①の背後に配置

図 2.12 の①，②において，表 2.6 に示す階層性があることが分かった．これらの特徴から，図 8.3 は，樹木の模様と山の模様の 2 層の階層があるといえる．

2.2.4　図 8.4 からの抽出

・ 個別の流れ

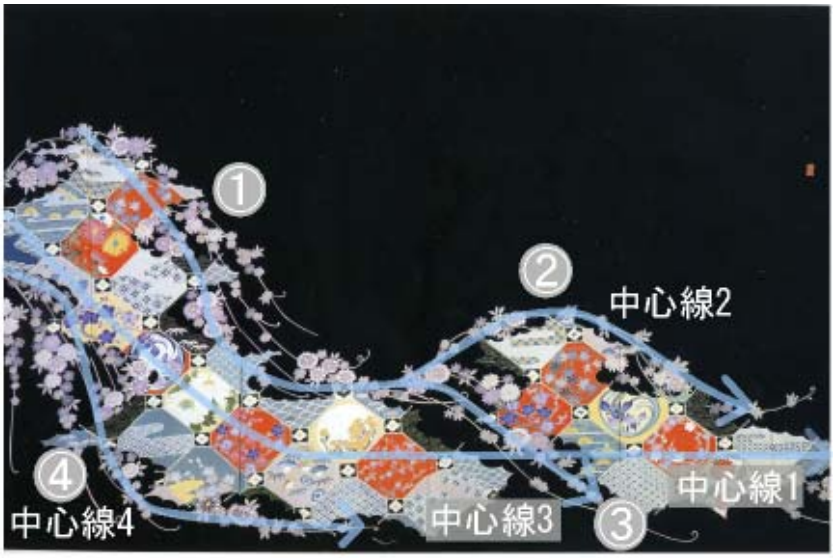


図 2.13　図 8.4 の個別の流れ

図 2.13 に示すように，この模様には中心線 1 とは別の 3 本の中心線 2,3,4 があることがわかる．各中心線は左上から右下に向かっており，表 2.7 で示す特徴を持つ．これらの特徴から，図 8.4 の模様は 4 本の個別の流れを軸にして配置していることが分かる．

表 2.7 図 8.4 の個別の流れの特徴

位置	特徴
①	中心線 2 に沿って，花の模様を配置
②	中心線 2 に沿って，花の模様を配置
③	中心線 3 に沿って，花の模様を配置
④	中心線 4 に沿って，花の模様を配置

・ 階層性

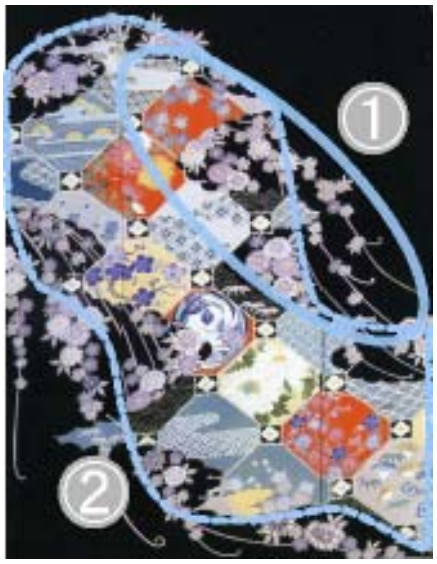


図 2.14　図 8.4 の階層性

表 2.8 図 8.4 の階層性の特徴

位置	特徴
①	花の模様を最前面に配置
②	背景の連続模様を①の背後に配置

図 2.14 の①，②において，表 2.8 に示す階層性の特徴があることが分かった．これらの特徴から，図 8.4 は 2 層の階層があるといえる．

2.2.5　　図 8.5 からの抽出

・ 個別の流れ

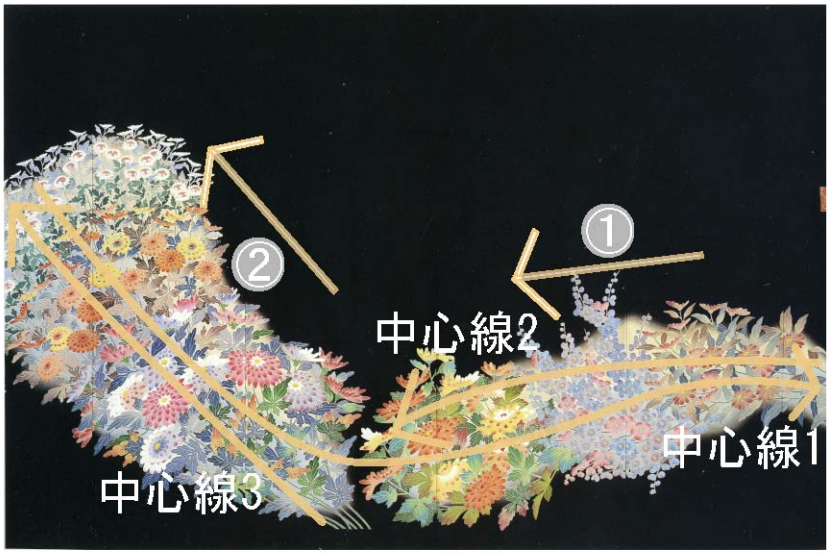


図 2.15　　図 8.5 の個別の流れ

図 2.15 に示すように，この模様には中心線 1 に沿って，中心線 2,3 で表す 2 本の個別の流れが存在する．表 2.9 に，各中心線の個別の流れの特徴を示す．これらの特徴から，図 8.5 の模様は，2 本の軸に沿った模様配置であることが分かる

表 2.9 図 8.5 の個別の流れの特徴

位置	特徴
①	中心線 2 に沿って，樹木の模様を配置
②	中心線 3 に沿って，樹木の模様を配置

・ 階層性

表 2.10 図 8.5 の階層性の特徴

位置	特徴
①	花の模様を最前面に配置
②	葉の模様を①の背後に配置
③	花の模様を②の背後に配置
④	葉の模様を③の背後に配置

図 2.16　図 8.5 の階層性

図 2.16 の①～④において，表 2.10 に示す階層性があることが分かった．これらの特徴から，図 8.5 は 4 層の階層があるといえる．

2.2.6　図 8.6 からの抽出

・ 個別の流れ

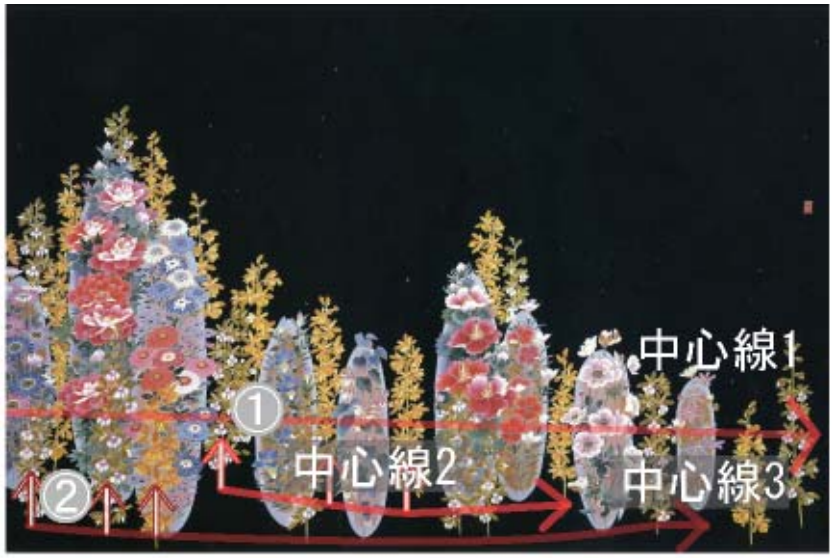


図 2.17　図 8.6 の個別の流れ

図 2.17 に示すように，この模様には中心線 1 以外に，中心線 2,3 で表す 2 本の個別の流れが存在する．表 2.11 に，各中心線の個別の流れの特徴を示す．これらの特徴から，2 本の個別の流れを軸として模様を配置していることが分かる．

表 2.11 図 8.6 の個別の流れの特徴

位置	特徴
①	中心線 2 に沿って，花の模様を配置
②	中心線 2 に沿って，花の模様を配置

・ 階層性

図 2.18　図 8.6 の階層性

表 2.12 図 8.6 の階層性の特徴

位置	特徴
①	樹木の模様を最前面に配置
②	樹木の模様を①の背後に配置
③	樹木の模様を②の背後に配置

図 2.18 の①～③において，表 2.12 に示す階層性があることが分かった．これらの特徴から，図 8.6 は 3 層の階層があるといえる．

2.2.7　　図 8.7 からの抽出

・ 個別の流れ

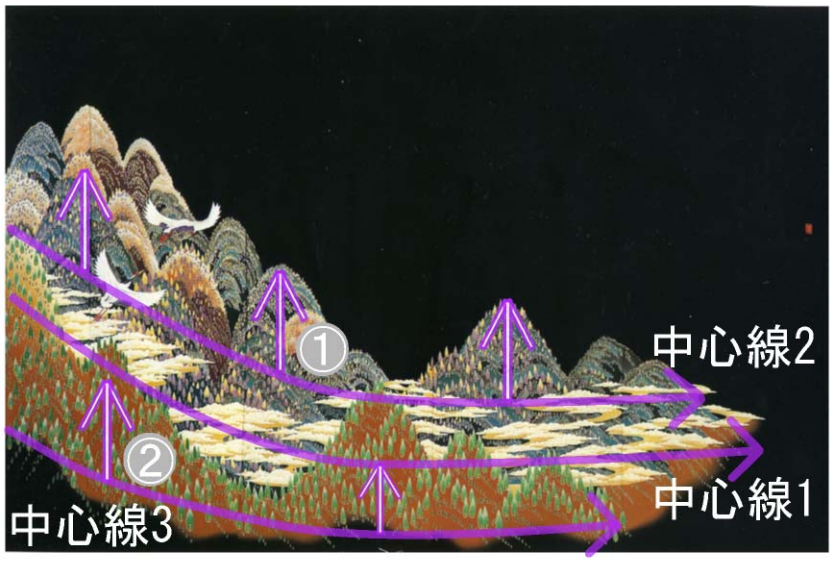


図 2.19　図 8.7 の個別の流れ

図 2.19 に示すように，この模様には，中心線 1 以外に中心線 2,3 で表す 2 つの大きな個別の流れが存在する．表 2.13 に，各中心線の個別の流れの特徴を示す．これらの特徴により，この模様は 2 本の個別の流れを軸として模様を配置していることが分かる．

表 2.13 図 8.7 の個別の流れの特徴

位置	特徴
①	山の模様を中心線 2 に沿って配置
②	緑の木のある山の模様を，中心線 3 に沿って配置

・ 階層性

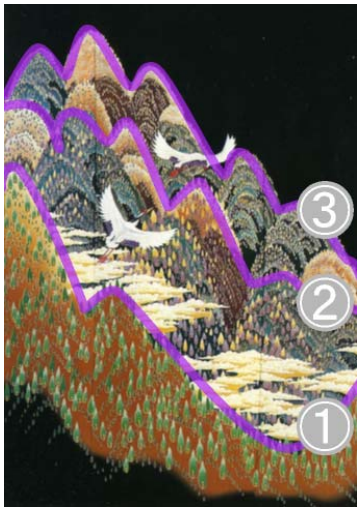


図 2.20　図 8.7 の階層性

表 2.14 図 8.7 の階層性の特徴

位置	特徴
①	緑の木のある山の模様を最前面に配置
②	青色，茶色の山の模様を①の背後に配置
③	青色，茶色の山の模様を②の背後に配置

図 2.20 の①～③において，表 2.14 に示す階層性の特徴があることが分かった．これらの特徴から，図 8.7 は 3 層の階層があるといえる．

2.2.8　図 8.8 からの抽出

・ 個別の流れ

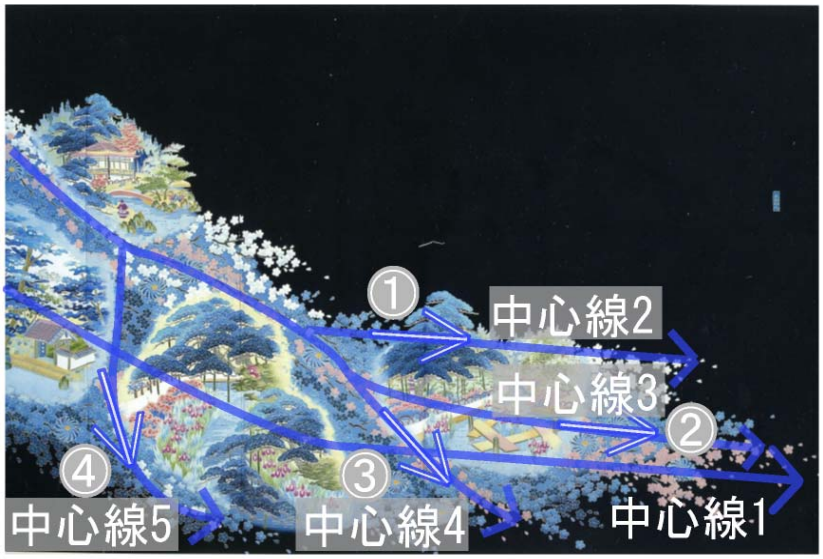


図 2.21 図 8.8 の個別の流れ

図 2.21 に示すように，この模様には，中心線 1 以外に中心線 2, 3, 4, 5 で表す 4 本の個別の流れが存在する．表 2.15 に，各中心線の個別の流れの特徴を示す．これらの特徴により，この模様は 4 本の個別の流れを軸として模様を配置していることが分かる．

表 2.15 図 8.8 の個別の流れの特徴

位置	特徴
①	木の枝を中心線 2 に沿って伸ばす
②	木の枝を中心線 3 に沿って伸ばす
③	木の枝を中心線 4 に沿って伸ばす
④	木の枝を中心線 5 に沿って伸ばす

・ 階層性

図 2.22 図 8.8 の階層性

表 2.16 図 8.8 の階層性の特徴

位置	特徴
①	庭園模様を配置
②	花・葉の模様を①の背後に配置

図 2.22 の①，②において，表 2.16 に示す階層性の特徴があることが分かった．これらの特徴から，図 8.8 は 2 層の階層があるといえる．

2.2.9　図 8.9 からの抽出

・ 個別の流れ

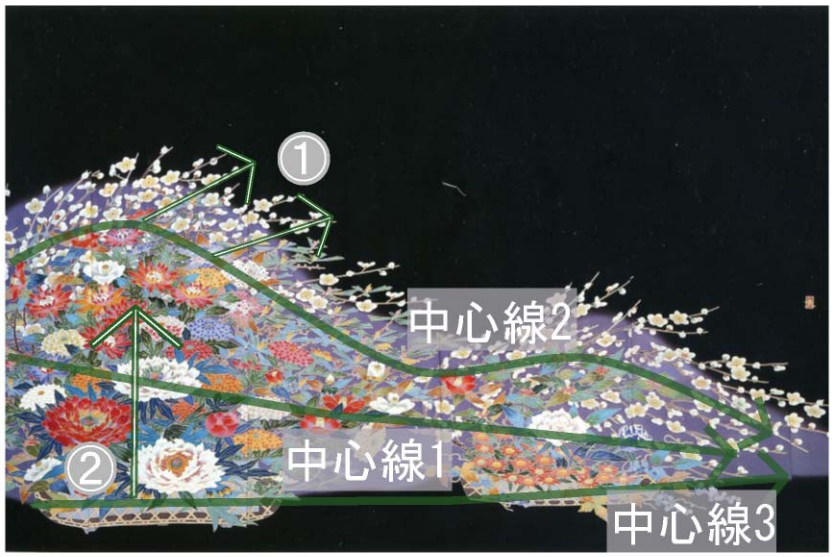


図 2.23　図 8.9 の個別の流れ

図 2.23 に示すように，この模様には，中心線 1 以外に中心線 2，3 で表す 2 本の大きな個別の流れが存在する．表 2.17 に，各中心線の個別の流れの特徴を示す．これらの特徴から，この模様は 2 つの流れを軸として模様を配置していることが分かる．

表 2.17 図 8.9 の個別の流れの特徴

位置	特徴
①	樹木の模様を，中心線 2 に沿って右上に向かって放射状に伸ばす
②	樹木の模様を，中心線 3 に対して垂直な方向へ伸ばす

・ 階層性

図 2.24 図 8.9 の階層性

表 2.18 図 8.9 の階層性の特徴

位置	特徴
①	大きな花を配置
②	①より小さな花を，①の背後に配置
③	②より小さな花を，②の背後に配置

図 2.24 の①～③において，表 2.18 に示す階層性の特徴があることが分かった．これらの特徴から，図 8.9 は 3 層の階層があるといえる．

2.2.10 図 8.10 からの抽出

・ 個別の流れ

図 2.25 図 8.10 の個別の流れ

図 2.25 に示すように，この模様には，中心線 1 以外に中心線 2,3 で表す 2 本の個別の流れが存在する．表 2.19 に，各中心線の個別の流れの特徴を示す．これらの特徴から，この模様は 2 つの流れを軸として模様を配置していることが分かる．

表 2.19 図 8.10 の個別の流れの特徴

位置	特徴
①	樹木の模様を，中心線 2 に対して垂直に伸ばす
②	樹木の模様を，中心線 3 に対して上方向に伸ばす

・ **階層性**

図 2.26　図 8.10 の階層性

表 2.20 図 8.10 の階層性の特徴

位置	特徴
①	花の模様を配置
②	①の背後に花の模様を配置
③	②の背後に花の模様を配置

図 2.26 の①～③において，表 2.20 に示すような階層性の特徴があることが分かった．これらの特徴から，図 8.10 は 3 層の階層があるといえる．

他の共通の規則性

2.1 節および，2.2 節で述べた模様の「個別の流れ」，「階層」の他に，以下の共通する特徴がある．

[Rule 2.3.1] すべての花・枝には，上下の区別がある．

図 2.27 植物模様のサンプル画像

図 2.27 に示すような模様では，上下の区別はない．しかし，図 8.1 のような植物模様では，下から上へ植物が伸び，それぞれの花の模様もすべて上を向いている．このように，着物模様には上下の区別があるといえる．

[Rule 2.3.2] 実際の植物をそのまま模写しているわけではない．
実際の植物をそのまま模写するのではなく，花の大きさを変化させたり，枝の伸び方に強弱を付けたりなど，ある程度の抑揚を植物につけている．

2.4　職人へのインタビュー

着物模様を作成する際の重要な事項に関して，職人のヒアリングを行った．以下にそのインタビュー結果の要約を示す．

- 模様を書く前に，全体の流れをある程度決定する．
- 花の種類は，一つに限定せず，多種の花模様を描くようにしている．
  例：主に描く模様は（牡丹・菊・桔梗・紅葉・椿・松・竹・梅など）である．
- 昔と今では，描き方が違う場合がある．
  例：菊の花びら 1 枚一枚に抑揚をつけて現代風にするなど

・ 書き始めはメインの部分から書き始める
・ メインの花を一番大きく，周りの花をそれよりも小さく描く（模様の大きさに強弱をつける）
・ 昔ながらの模様の手法は現在でも変わらない
例：（唐草模様・波模様・宝文様など）

2.5 まとめ

職人のヒアリングの結果で印象に残ったのが，模様の「流れ」が非常に大事であるということだった．2.1，2.2 章で述べた模様の「全体の流れ」，「個別の流れ」は，このことからも重要であると考える．また，階層性に関しても，サンプル模様集のすべての模様に共通するルールであることが分かった．これにより，「流れ」と「階層性」を考慮して模様を生成することが重要と考える．

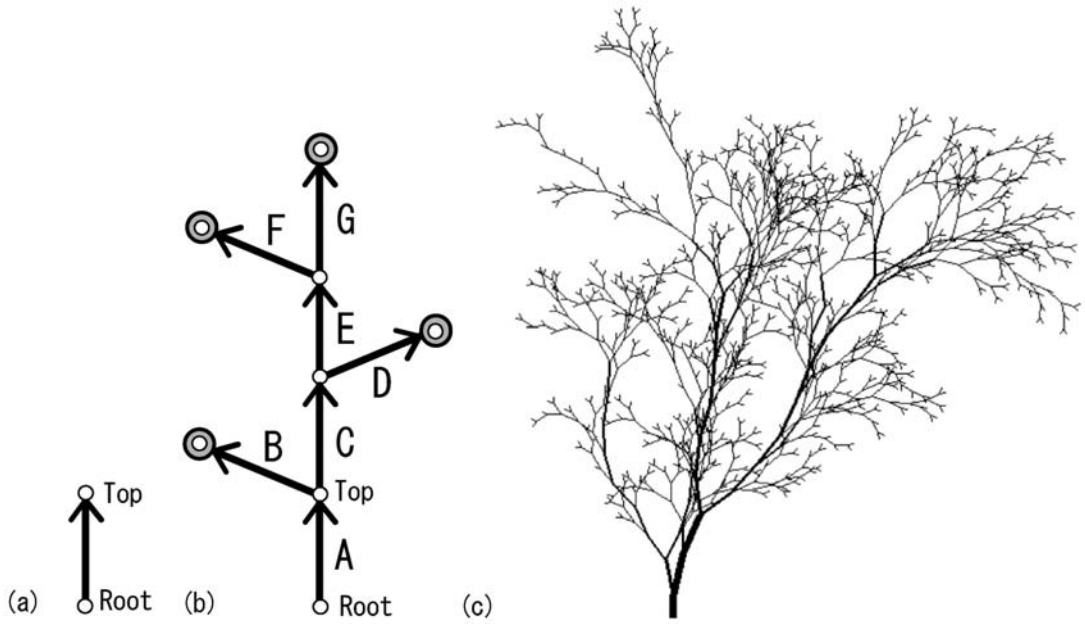
以上の結論を基に，着物模様のデザイン支援システムの構築を行なう．本論文で構築したデザイン支援システムは，以下に示す 3 つの作業手順を経るユーザインターフェイスを有する．ユーザインターフェイスに関しては 4 章で後述する．

1. 模様の上領域・下領域線を描く（全体の流れを決定）
2. 個々の植物・花・葉の配置する中心線を描く（個別の流れを決定）
3. 階層性（レイヤー構造）を考え，2. で配置した植物模様の前後関係を決定する

第　3　章
模様生成アルゴリズム

3.1　L－System の応用

L-System とは，図 3.1(a)のように，Root 地点から，Top 地点に向けて，線分を描画するという単純なステップを再帰的に繰り返すことで，複雑な図形を描画するアルゴリズムを指す．図 3.1(b)では図 3.1(a)の線分が A→B→C→D→E→F→G の順で描画される線分の集合に変換することで樹木を描画している．このステップを再帰的に繰り返すことで，図 3.1(c)に示すような樹形図を描くことができる．本研究では，このアルゴリズムを応用して着物模様における樹木の模様を生成する．



(a)基本ステップ　(b)生成ステップ　(c)生成された樹形図

図 3.1 L-System

3.1.1　樹木の模様の種類

L-System を用いて樹木の模様を生成するにあたり，サンプル模様集に描かれている樹木の模様を参照して，図 3.2 に示すような 8 種類の生成規則を定義した．以降，それぞれの樹木の模様を Tree1～Tree8 と表記する．

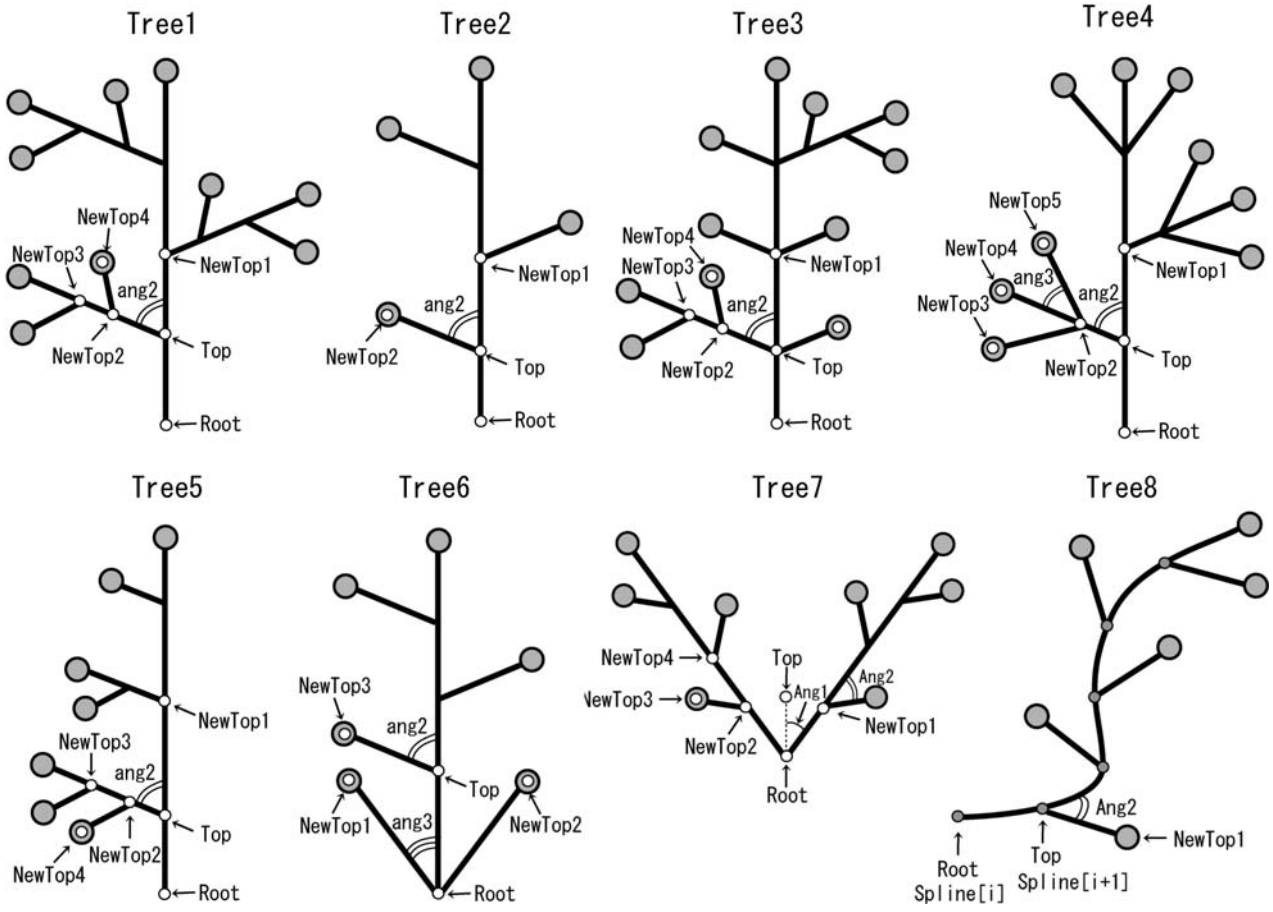


図 3.2 樹木の模様の種類

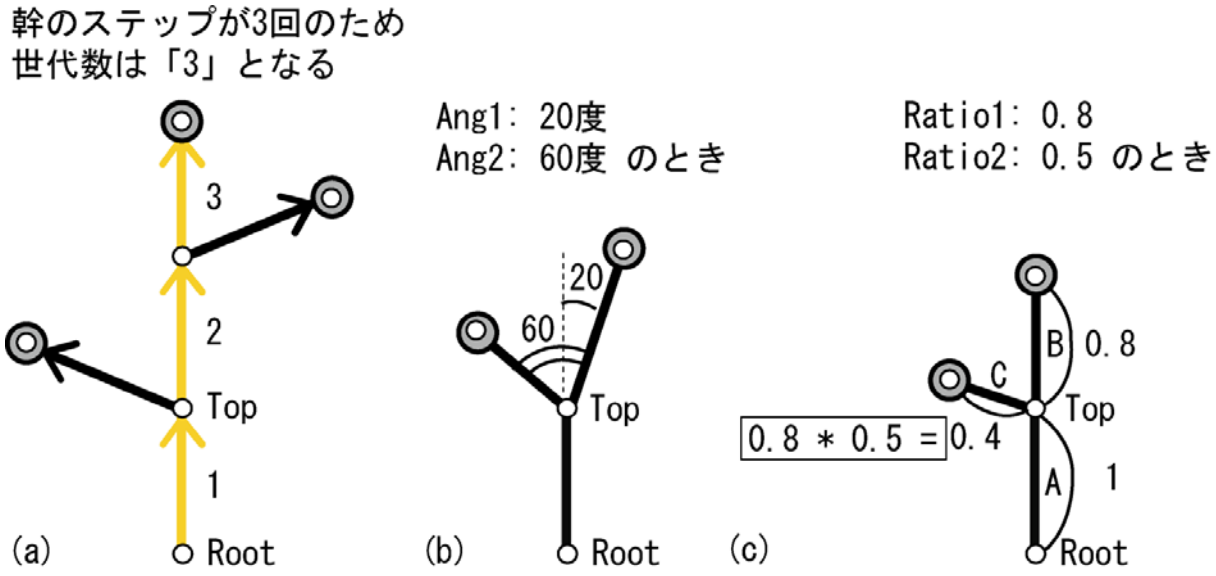
3.1.2　用語の説明

以降 3.1.4～3.1.11 において，それぞれの樹木の生成アルゴリズム内で示している用語を，表 3.1 のように定義することにする．

表 3.1　用語の説明

用語	説明
世代	樹木の幹のステップ数
Root	開始地点(スクリーン座標)
Top ， NewTop1 ， NewTop2 NewTop3，NewTop4	スクリーン座標系での樹木の模様の頂点
Ang1	樹木の幹の水直方向に対する角度
Ang2	樹木の幹と枝の間の角度
Ratio1	樹木の幹の倍率
Ratio2	樹木の枝の倍率
Vector(a, b)	ベクトル $\vec{ab}$ （a, b はスクリーン座標）
Rotate(ab, Ang1)	ベクトル $\vec{ab}$ を Ang1 の角度に回転させる

ここで，表 3.1 内の「世代」「Ang1・Ang2」「Ratio1・Ratio2」のパラメータの意味を，図 3.3 に示す.



(a)世代数　　(b)Ang1・Ang2　　(c)Ratio1・Ratio2

図 3.3 パラメータの説明図

世代数は, 樹木の幹のステップ数であり, 図 3.1(a)は世代数が 3 のときを示す. 図 3.1(b)は，Ang1 が 20 度，Ang2 が 60 度のときの枝分かれの様子を示す．図 3.2(c)は，樹木の枝や幹の倍率の適用例を示す．Ratio1 は，幹 A が成長し幹 B になる場合の幹の

成長率を指定するパラメータであり，幹 B の長さは幹 A の長さに Ratio1 を掛けたもので求められる．同様に，Ratio2 は幹 B が分岐して枝 C になる場合の成長率を指定するパラメータであり，枝 C の長さは，幹 B の長さに Ratio2 を掛けたもので求められる．図 3.2(c)の例では，幹 A の長さを 1 とした場合，幹 B は 0.8 の長さに，枝 C は 0.4 の長さとなる．

3.1.3　条件判定

Tree1～8 のすべての生成規則に，条件文①に示すような，樹木の模様の成長を制限するための条件がある．これは，上領域線の上部分に樹木の頂点がきた場合，その時点で *false* の値を関数の呼び出し側に返し，樹木の生成を止めるための条件である．

条件文①「if(Top ＞ 上領域線)　return false;」

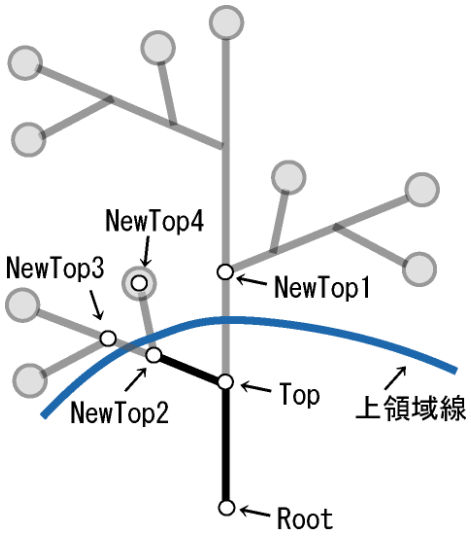


図 3.4　上領域線による制約条件

図 3.4 に示すような上領域線の場合，Top の位置は上領域線より下にあるために，Top の頂点までラインが描画される．一方，NewTop1 は上領域線よりも上にあるために，条件文①が適用され，それより上部の幹は生成しない．同様に，枝の部分では，NewTop2 までは描画されるが，NewTop3，NewTop4 以降の枝の頂点は描画しない．以上のように，条件文①は木の成長を上領域線までに制限する．

3.1.4 Tree1

Tree1 には，以下に述べる 2 つの関数があり，TreeMain 関数で樹木の模様の幹の部分を，TreeSub 関数で枝の部分の頂点座標をそれぞれ求める．

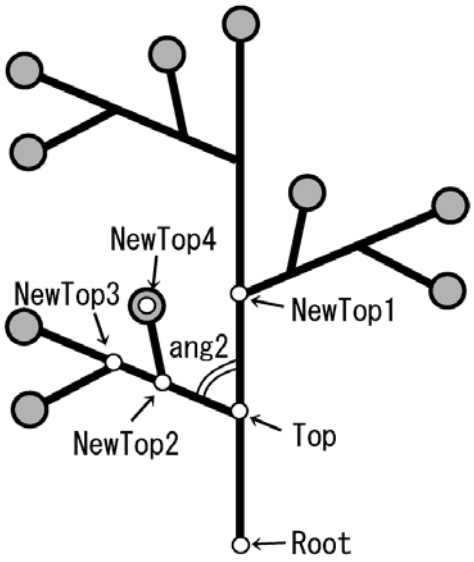


図 3.5 Tree1

・ TreeMain 関数

```
TreeMain(世代, Root, Top)
{
    //① 領域判定
    if(Top > 上領域線) return false;

    if(世代 > 0){

        //② 幹の新しい頂点を求める
        NewTop1
         = Rotate(Vector(Root,Top) * Ratio1, Ang1) + Top;

        //③ 枝の頂点を求める
        if(世代 == 偶数){
            NewTop2
             = Rotate(Vector(Top,NewTop1)* Ratio2, -Ang2);//左
        }
        else{
            NewTop2
             = Rotate(Vector(Top,NewTop1)* Ratio2,  Ang2);//右
        }

        //④ 再帰呼び出し
        TreeMain(世代-1, Top, NewTop1);
        TreeSub(3, Top, NewTop2);
    }
    else//⑤ 世代が0で終了
        return false;
}
```

表 3.2 Tree1 における TreeMain 関数

TreeMain 関数	
場所	説明
①	Top 座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
②	Root・Top の頂点座標と，Ratio1 の値から NewTop1 を求める
③	世代数の偶数・奇数の別により，枝を左右に配置するように NewTop2 を求める
④	・Top と NewTop1 を，「TreeMain」関数の引数にすることで，木の幹の頂点座標を順に求める． ・Top と NewTop2 を「TreeSub」関数の引数にすることで，木の枝の頂点座標を順に求める．
⑤	世代数が 0 になった時点で樹木の模様生成を終了する

・ TreeSub 関数

```
TreeSub(世代, Top, NewTop2)
{
        //⑥ 領域判定
        if(NewTop2 > 上領域線) return false;

        if(世代 > 0){

                //⑦ 幹の新しい頂点を求める
                NewTop3
                 = Rotate(Vector(Top,NewTop2)* Ratio1, Ang1) + NewTop2;

                //⑧ 枝の頂点を求める
                if(世代 == 偶数){
                   NewTop4
                    = Rotate(Vector(NewTop2,NewTop3)* Ratio2, -Ang2);//左
                }
                else{
                   NewTop4
                    = Rotate(Vector(NewTop2,NewTop3)* Ratio2,  Ang2);//右
                }

                //⑨ 再帰呼び出し
                TreeSub(世代-1, NewTop2, NewTop3);
        }
        else//⑩ 世代が0で終了
          return false;
}
```

表 3.3 Tree1 における TreeSub 関数

TreeSub 関数	
場所	説明
⑥	NewTop2 の座標が上領域線の下にある場合のみプログラムを続けるように条件判定を行う
⑦	Top と NewTop2 の頂点座標と，Ratio1 の値から NewTop3 を求める
⑧	世代数の偶数・奇数の別により，枝を左右に配置するように NewTop4 を求める
⑨	NewTop2，NewTop3 を「TreeSub」関数の引数にすることで，木の枝の頂点座標を順に求める．
⑩	世代数が 0 になった時点で樹木の模様生成を終了する

このように，Tree1 では，Main 関数において，樹木の模様の幹の部分を，Sub 関数において，枝の部分の頂点座標をそれぞれ求めている．

3.1.5 Tree2

Tree2 は，以下に述べる 1 つの関数のみであり TreeMain 関数で，樹木の模様の幹の部分と，枝の部分の両方の頂点座標を求める．

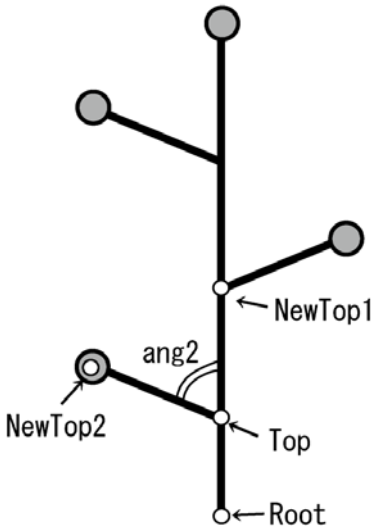


図 3.6 Tree2

・ TreeMain 関数

```
TreeMain(世代, Root, Top)
{
    //① 領域判定
    if(Top > 上領域線) return false;

    if(世代 > 0){

        //② 幹の新しい頂点を求める
        NewTop1
         = Rotate(Vector(Root,Top) * Ratio1, Ang1) + Top;

        //③ 枝の頂点を求める
        if(世代 == 偶数){
            NewTop2
             = Rotate(Vector(Top,NewTop1)* Ratio2, -Ang2);//左
        }
        else{
            NewTop2
             = Rotate(Vector(Top,NewTop1)* Ratio2,  Ang2);//右
        }

        //④ 再帰呼び出し
        TreeMain(世代-1, Top, NewTop1);
    }
    else//⑤ 世代が0で終了
        return false;
}
```

表 3.4 Tree2 における TreeMain 関数

TreeMain 関数	
場所	説明
①	Top 座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
②	Root・Top の頂点座標と，Ratio1 の値から NewTop1 を求める
③	世代数の偶数・奇数の別により，枝を左右に配置するように NewTop2 を求める
④	・Top と NewTop1 を，「TreeMain」関数の引数にすることで，木の幹の頂点座標を順に求める． ・NewTop2 は枝が単純なため，関数に引き渡さない．
⑤	世代数が 0 になった時点で樹木の模様生成を終了する

3.1.6 Tree3

Tree3 は,以下に述べる 2 つの関数があり,TreeMain 関数で樹木の模様の幹の部分を,TreeSub 関数で枝の部分の頂点座標をそれぞれ求める．

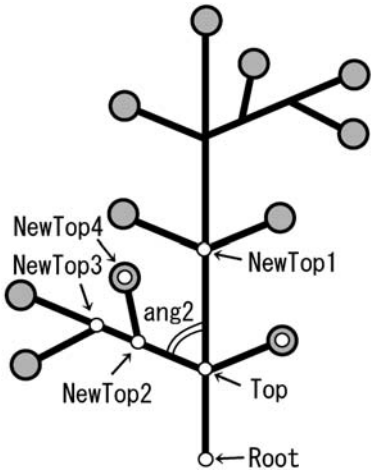


図 3.7 Tree3

・ TreeMain 関数

```
TreeMain(世代, Root, Top)
{
    //① 領域判定
    if(Top > 上領域線) return false;

    if(世代 > 0){

        //② 幹の新しい頂点を求める
        NewTop1
         = Rotate(Vector(Root,Top) * Ratio1, Ang1) + Top;

        //③ 枝の頂点を求める
        NewTop2
             = Rotate(Vector(Top,NewTop1)* Ratio2, -Ang2);//左
        NewTop5
             = Rotate(Vector(Top,NewTop1)* Ratio2, Ang2); //右

        //④ 再帰呼び出し
        if(世代%3 == 0){
            TreeMain(世代-1, Top, NewTop1);
            TreeSub(3, Top, NewTop2);//左
        }
        else if(世代%3 == 1){
            TreeMain(世代-1, Top, NewTop1);
            TreeSub(3, Top, NewTop5);//右
        }
        else
            TreeMain(世代-1, Top, NewTop1);
    }
    else//⑤ 世代が0で終了
        return false;
}
```

表 3.5 Tree3 における TreeMain 関数

TreeMain 関数	
場所	説明
①	Top 座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
②	Root・Top の頂点座標と，Ratio1 の値から NewTop1 を求める
③	枝を左右に配置するように NewTop2，NewTop5 を求める
④	3 つの枝の付き方を，世代数を 3 で割った数の余りの条件を利用して，余り 0，1，2 によって，①右が単純な枝，左に複雑な枝　②左が単純な枝，右に複雑な枝　③左右両方とも単純な枝　の 3 通りの枝の配置をおこなう
⑤	世代数が 0 になった時点で樹木の模様生成を終了する

・ TreeSub 関数

```
TreeSub(世代, Top, NewTop2)
{
        //⑥ 領域判定
        if(NewTop2 > 上領域線) return false;

        if(世代 > 0){

                //⑦ 幹の新しい頂点を求める
                NewTop3
                 = Rotate(Vector(Top,NewTop2)* Ratio1, Ang1) + NewTop2;

                //⑧ 枝の頂点を求める
                if(世代 == 偶数){
                   NewTop4
                    = Rotate(Vector(NewTop2,NewTop3)* Ratio2, -Ang2);
                }
                else{
                   NewTop4
                    = Rotate(Vector(NewTop2,NewTop3)* Ratio2,  Ang2);
                }

                //⑨ 再帰呼び出し
                TreeSub(世代-1, NewTop2, NewTop3);
        }
        else//⑩  世代が0で終了
             return false;
}
```

表 3.6 Tree3 における TreeSub 関数

TreeSub 関数	
場所	説明
⑥	NewTop2 の座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
⑦	Top と NewTop2 の頂点座標と，Ratio1 の値から NewTop3 を求める
⑧	世代数の偶数・奇数の別により，枝を左右に配置するように NewTop4 を求める
⑨	NewTop2，NewTop3 を「TreeSub」関数の引数にすることで，木の枝の頂点座標を順に求める.
⑩	世代数が 0 になった時点で樹木の模様生成を終了する

Tree3 では，枝の付き方に大きな特徴がある．3 つのパターンを用意し，それを場合分けして枝の付き方を変化させることで，新たな樹木の模様を表現する.

3.1.7 Tree4

Tree4 は,以下に述べる 2 つの関数があり,TreeMain 関数で樹木の模様の幹の部分を,TreeSub 関数で枝の部分の頂点座標をそれぞれ求める

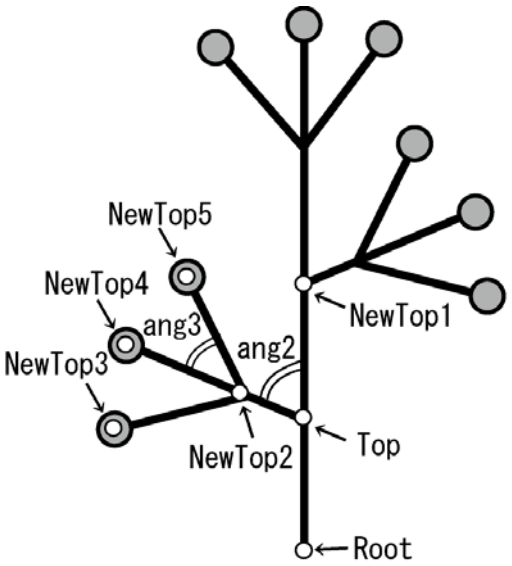


図 3.8 Tree4

・ TreeMain 関数

```
TreeMain(世代, Root, Top)
{
    //① 領域判定
    if(Top > 上領域線) return false;

    if(世代 > 0){

        //② 幹の新しい頂点を求める
        NewTop1
         = Rotate(Vector(Root,Top) * Ratio1, Ang1) + Top;

        //③ 枝の頂点を求める
        if(世代 == 偶数){
            NewTop2
             = Rotate(Vector(Top,NewTop1)* Ratio2, -Ang2);//左
        }
        else{
            NewTop2
             = Rotate(Vector(Top,NewTop1)* Ratio2,  Ang2);//右
        }

        //④ 再帰呼び出し
        TreeMain(世代-1, Top, NewTop1);
        TreeSub(1, Top, NewTop2);
    }
    else//⑤ 世代が0で終了
        return false;
}
```

表 3.7 Tree4 における TreeMain 関数

TreeMain 関数	
場所	説明
①	Top 座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
②	Root・Top の頂点座標と，Ratio1 の値から NewTop1 を求める
③	世代数の偶数・奇数の別により，枝を左右に配置するように NewTop2 を求める
④	・Top と NewTop1 を，「TreeMain」関数の引数にすることで，木の幹の頂点座標を順に求めることができる． ・Top と NewTop2 を「TreeSub」関数の引数にすることで，木の枝の頂点座標を順に求める．
⑤	世代数が 0 になった時点でプログラムを終了させる

・ TreeSub 関数

```
TreeSub(世代, Top, NewTop2)
{
        //⑥ 領域判定
        if(NewTop2 > 上領域線) return false;

        if(世代 > 0){

                //⑦ 幹の新しい頂点を求める
                NewTop4
                 = Rotate(Vector(Top,NewTop2)* Ratio1, Ang1) + NewTop2;

                //⑧ 枝の頂点を求める
                NewTop3
                 = Rotate(Vector(NewTop2,NewTop4)* Ratio2, -Ang3);//左
                NewTop5
                 = Rotate(Vector(NewTop2,NewTop4)* Ratio2,  Ang3);//右

                //⑨ 再帰呼び出し
                TreeSub(世代-1, NewTop2, NewTop3);
        }
        else//⑩ 世代が0で終了
             return false;
}
```

表 3.8 Tree4 における TreeSub 関数

TreeSub 関数	
場所	説明
⑥	NewTop2 の座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
⑦	Top と NewTop2 の頂点座標と，Ratio1 の値から NewTop4 を求める
⑧	枝を左右に配置するように，NewTop2，NewTop4 の値と，Ratio2 の値から左の枝(NewTop3)と右の枝(NewTop5)を求める
⑨	NewTop2，NewTop3 を「TreeSub」関数の引数にすることで，木の枝の頂点座標を順に求める．
⑩	世代数が 0 になった時点で樹木の模様生成を終了する

Tree4 では，枝の形に大きな特徴がある．樹木の幹となる部分は，Tree1 の場合と同一であるが，枝の部分では，枝分岐のパラメータ Ang3 を追加し，枝を三叉の状態にしている．

3.1.8　Tree5

Tree5 は,以下に述べる 2 つの関数があり,TreeMain 関数で樹木の模様の幹の部分を,TreeSub 関数で枝の部分の頂点座標をそれぞれ求める

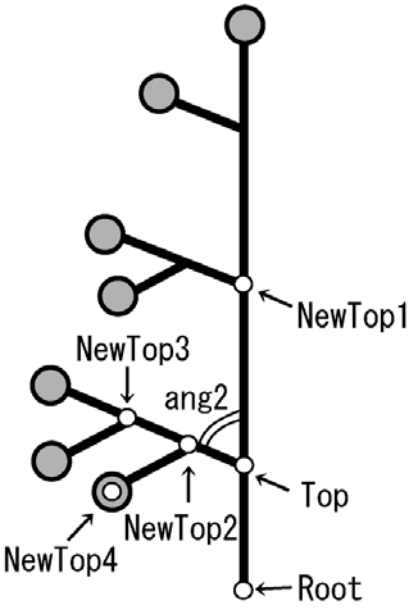


図 3.9 Tree5

・ TreeMain 関数

```
TreeMain(世代, Root, Top)
{
    //① 領域判定
    if(Top > 上領域線) return false;

    if(世代 > 0){

        //② 幹の新しい頂点を求める
        NewTop1
         = Rotate(Vector(Root,Top) * Ratio1, Ang1) + Top;

        //③ 枝の頂点を求める
        NewTop2
         = Rotate(Vector(Top,NewTop1)* Ratio2, -Ang2);//左

        //④ 再帰呼び出し
        TreeMain(世代-1, Top, NewTop1);
        TreeSub(世代-1, Top, NewTop2);

    }
    else//⑤ 世代が0で終了
        return false;
}
```

表 3.9 Tree5 における TreeMain 関数

TreeMain 関数	
場所	説明
①	Top 座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
②	Root・Top の頂点座標と，Ratio1 の値から NewTop1 を求める
③	枝を左にのみ配置するように NewTop2 を求める
④	・Top と NewTop1 を，「TreeMain」関数の引数にすることで，木の幹の頂点座標を順に求める． ・Top と NewTop2 を「TreeSub」関数の引数にすることで，木の枝の頂点座標を順に求める．
⑤	世代数が 0 になった時点で樹木の模様生成を終了する

・ TreeSub 関数

```
TreeSub(世代, Top, NewTop2)
{
        //⑥ 領域判定
        if(NewTop2 > 上領域線) return false;

        if(世代 > 0){

                //⑦ 幹の新しい頂点を求める
                NewTop3
                 = Rotate(Vector(Top,NewTop2)* Ratio1, Ang1) + NewTop2;

                //⑧ 枝の頂点を求める
                NewTop4
                 = Rotate(Vector(NewTop2,NewTop3)* Ratio2, -Ang2);//左

                //⑨ 再帰呼び出し
                TreeSub(世代-1, NewTop2, NewTop3);
        }
        else//⑩ 世代が0で終了
             return false;
}
```

表 3.10 Tree5 における TreeSub 関数

TreeSub 関数	
場所	説明
⑥	NewTop2 の座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
⑦	Top と NewTop2 の頂点座標と，Ratio1 の値から NewTop3 を求める
⑧	枝を左に配置するように，NewTop2，NewTop3 の値と，Ratio2 の値から NewTop4 を求める
⑨	NewTop2，NewTop3 を「TreeSub」関数の引数にすることで，木の枝の頂点座標を順に求める．
⑩	世代数が 0 になった時点で樹木の模様生成を終了する

Tree5 では，すべての枝を左側のみに付けることが大きな特徴である．したがって，偶数・奇数などの条件は不要となり，枝の付き方は単純な構造になっている．

3.1.9 Tree6

Tree6 は，以下に述べる 1 つの関数のみであり TreeMain 関数で，樹木の模様の幹の部分と，枝の部分の両方の頂点座標を求める．

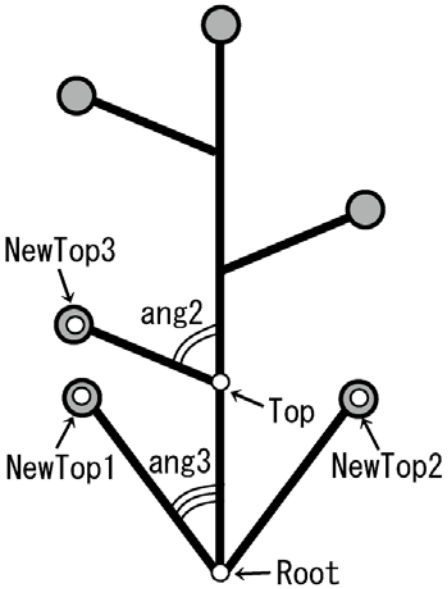


図 3.10 Tree6

・ TreeMain 関数

```
TreeMain(世代, Root, Top)
{
    //① 領域判定
    if(Top > 上領域線) return false;

    if(世代 > 0){

        if(世代 == 初期値){
            //② 枝の頂点を求める
            NewTop1
             = Rotate(Vector(Root,Top)* Ratio3, -Ang3);//左

            //③ 枝の頂点を求める
            NewTop2
             = Rotate(Vector(Root,Top)* Ratio3,  Ang3);//右
        }

        //④ 幹の新しい頂点を求める
        NewTop4
         = Rotate(Vector(Root,Top)* Ratio1, Ang1) + Top;

        //⑤ 枝の頂点を求める
        if(世代 == 偶数){
            NewTop3
             = Rotate(Vector(Top,NewTop4)* Ratio2, -Ang2);//左
        }
        else{
            NewTop3
             = Rotate(Vector(Top,NewTop4)* Ratio2,  Ang2);//右
        }

        //⑥ 再帰呼び出し
        TreeMain(世代-1, Top, NewTop4);

    }
    else//⑦ 世代が0で終了
        return false;
}
```

表 3.11 Tree6 における TreeMain 関数

TreeMain 関数	
場所	説明
①	Top 座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
②	世代の数が初期値の場合，Root・Top の頂点座標と，Ratio3 の値から NewTop1 を求める
③	世代の数が初期値の場合，Root・Top の頂点座標と，Ratio3 の値から NewTop2 を求める
④	Root・Top の頂点座標と，Ratio1 の値から NewTop4 を求める
⑤	世代数の偶数・奇数の別により，枝を左右に配置するように NewTop2 を求める
⑥	・Top と NewTop4 を，「TreeMain」関数の引数にすることで，木の幹の頂点座標を順に求める． ・木の枝が単純なので，TreeSub 関数は呼び出さない
⑦	世代数が 0 になった時点で樹木の模様生成を終了する

Tree6 では，根元の部分において，“世代数が初期値の場合”という条件式を適用することで，2 つの頂点 NewTop1，NewTop2 を求めていることが特徴である．なお，Ratio3 は根元にある 2 つの枝の長さを変動させるために定義したパラメータである．

3.1.10　Tree7

Tree7 は，以下に述べる 2 つの関数があり，TreeMain 関数で分岐の頂点を，TreeSub 関数で 2 本の樹木の幹と枝の頂点座標をそれぞれ求める．

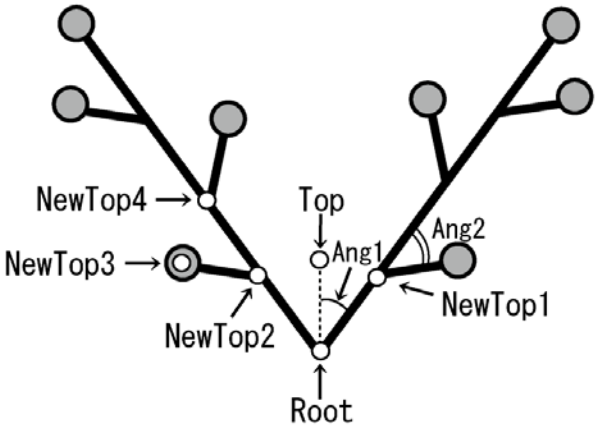


図 3.11 Tree7

・ TreeMain 関数

```
TreeMain(世代, Root, Top)
{

    //① 領域判定
    if(Top > 上領域線) return false;

    if(世代>0){

        if(世代 == 初期値){
            //② 幹の頂点を求める
            NewTop1
             = Rotate(Vector(Root,Top)* Ratio1,  Ang1);//右

            //③ 幹の頂点を求める
            NewTop2
            = Rotate(Vector(Root,Top)* Ratio1, -Ang1);//左
        }

        //④ 再帰呼び出し
        TreeSub(世代-1, Root, NewTop1);
        TreeSub(世代-1, Root, NewTop2);

    }
    else//⑤ 世代が0で終了
        return false;
}
```

表 3.12 Tree7 における TreeMain 関数

TreeMain 関数	
場所	説明
①	Top 座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
②	Root・Top の頂点座標と，Ratio1 の値から NewTop1 を求める
③	Root・Top の頂点座標と，Ratio1 の値から NewTop2 を求める
④	・「TreeMain」関数は呼び出さない. ・Root と NewTop1,2 を「TreeSub」関数の引数にすることで，2 本の木の幹の頂点座標を順に求める.
⑤	世代数が 0 になった時点で樹木の模様生成を終了する

・ TreeSub 関数

```
TreeSub(世代, Root, NewTop2)
{
        //⑥ 領域判定
        if(NewTop2 > 上領域線) return false;

        if(世代 > 0){

                //⑦ 幹の新しい頂点を求める
                NewTop4
                 = Rotate(Vector(Top,NewTop2)* Ratio1, Ang1) + NewTop2;

                //⑧ 枝の頂点を求める
                if(世代 == 偶数){
                   NewTop3
                    = Rotate(Vector(NewTop2,NewTop4)* Ratio2, -Ang2);//左
                }
                else{
                   NewTop3
                    = Rotate(Vector(NewTop2,NewTop4)* Ratio2,  Ang2);//右
                }

                //⑨ 再帰呼び出し
                TreeSub(世代-1, NewTop2, NewTop4);
        }
        else//⑩ 世代が0で終了
            return false;
}
```

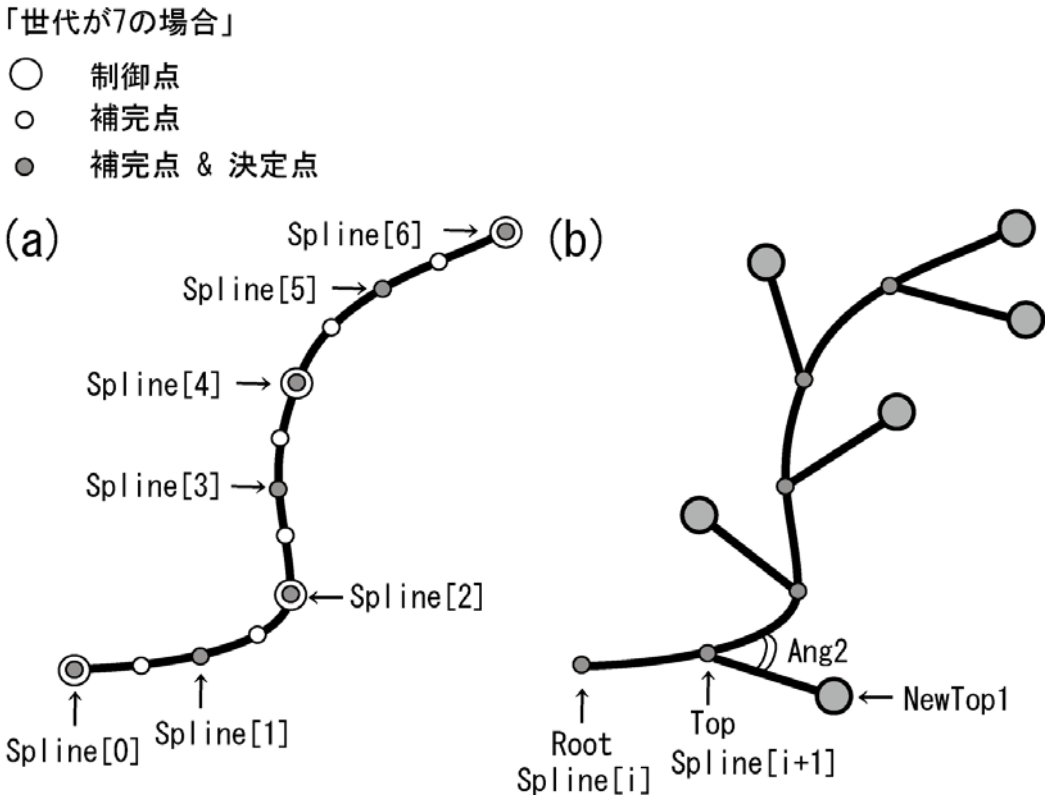
表 3.13 Tree7 における TreeSub 関数

TreeSub 関数	
場所	説明
⑥	NewTop2 の座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
⑦	Top と NewTop2 の頂点座標と，Ratio1 の値から NewTop4 を求める
⑧	世代数の偶数・奇数の別により，枝を左右に配置するように NewTop3 を求める
⑨	NewTop2, NewTop4 を「TreeSub」関数の引数にすることで，木の枝の頂点座標を順に求める.
⑩	世代数が 0 になった時点で樹木の模様生成を終了する

Tree7 は，根元の部分で 2 つの樹木に分かれることが大きな特徴である．初期値である Top の座標は，2 本の樹木の生える方向を決定するためのみに用いられる．求めた 2 本の樹木は Tree2 と同じものである.

3.1.11　Tree8

Tree8 は，以下に述べる 1 つの関数のみであり TreeMain 関数で，樹木の模様の幹の部分と，枝の部分の両方の頂点座標を求める．



(a)基礎ラインの生成　　(b)Tree8

図 3.12　Tree8

Tree8 は，他の樹木の模様と違い初期設定が異なるため，再帰アルゴリズムを用いていない．Tree8 を生成するには，まず基礎のラインを指定する必要がある．Tree8 はこの基礎ラインに沿って生成される．このラインは，3.2 章で後述するスプライン曲線を用いて生成する．スプライン曲線は，4 つの制御点を指定することにより，その間の点を補完して求めることができる．図 3.12(a)に，4 つの制御点と，その間の補完点を示している．求めた補完点は，すべて利用するわけではなく，世代数によって利用する個数が変わる．図 3.12 の例は，世代数が 7，7 つの補完点のときを示している．以降，それぞれの補完点を Spline[0]～Spline[6]と表記する．図 3.12(b)に示すように，7 つの補完点を指定後，各補完点から枝を伸ばして，樹木の模様を生成する．

・ TreeMain 関数

```
TreeMain(世代)
{
    // ①世代の数だけ繰り返す
    for(int i=0; i<世代-1; i++){

        Root =   Spline[i];
        Top =    Spline[i+1];

        //② 領域判定
        if(Top > 上領域線) return false;

        if(i == 偶数){
            //③ 幹の頂点を求める
            NewTop1
             = Rotate(Vector(Root,Top)* Ratio2,   -Ang1);//左
        }
        else{
            //④ 幹の頂点を求める
            NewTop1
             = Rotate(Vector(Root,Top)* Ratio2,   Ang1);//右
        }
    }

    //⑤ 世代が0で終了
    return false;
}
```

表 3.14 Tree8 における TreeMain 関数

TreeMain 関数	
場所	説明
①	Root と Top に，Spline[i]，Spline[i+1]の座標を代入する処理を，世代の数だけ繰り返す
②	Top 座標が上領域線の下にある場合のみプログラムを続けるように条件判定をおこなう
③，④	世代数の偶数・奇数の別により，枝を左右に配置するように NewTop1 を求める
⑤	世代数が 0 になった時点で樹木の模様生成を終了する

Tree8 の大きな特徴は，Spline 補完で求めた補完点を利用して，樹木を生成することである．Spline[0]と Spline[1]の値から順に木の幹の頂点座標を求め，枝の頂点は Tree1 などと同様に世代の数で左右に配置する．

3.2　曲線補完

L-system で自動生成した木の模様に対して，3 次ベジェ曲線と 3 次スプライン補完を利用することで，さらに着物模様らしい表現に近づけた．本節では，ベジェ曲線とスプライン補完法について触れた後に，これらの曲線表現を利用した樹木の模様の表現法について述べる．

3.2.1　3 次ベジェ曲線

3 次ベジェ曲線は図 3.13 に示すような 4 個の制御点で定義される曲線であり，t をパラメータにして式(1)で表される．パラメータ t は各補完点の位置を制御する．

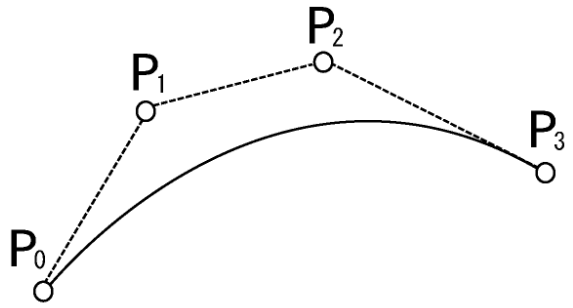


図 3.13 3 次ベジェ曲線

$$P(t) = P_0 B_0^3(t) + P_1 B_1^3(t) + P_2 B_2^3(t) + P_3 B_3^3(t) \qquad (1)$$

ここで重み関数 $B_i^3(t)$ は 3 次バーンスタイン関数と呼ばれ，式(2)で表される．

$$\begin{aligned} B_0^3(t) &= (1-t)^3 \\ B_1^3(t) &= 3t(1-t)^2 \\ B_2^3(t) &= 3t^2(1-t) \\ B_3^3(t) &= t^3 \end{aligned} \quad (0 \le t \le 1) \qquad (2)$$

たとえば，式(1)のパラメータ t に 0.25，0.5，0.75 の 3 つの値を代入した場合の各補完点は，図 3.14 に示す位置の点となる．t を 0.0 から順に 0.1 の刻み幅で 1 になる

まで連続して代入することにより，曲線上の各点を求めることができる．

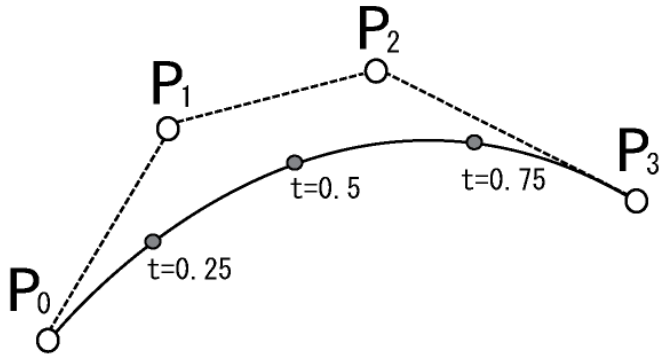


図 3.14 3 次ベジェ曲線の補完点の例

3.2.2　3 次スプライン補完法

3 次スプライン補完とは,与えられた複数の点の隣り合う 2 点間を 3 次関数で補完し，点同士の接続を滑らかにしたものである．

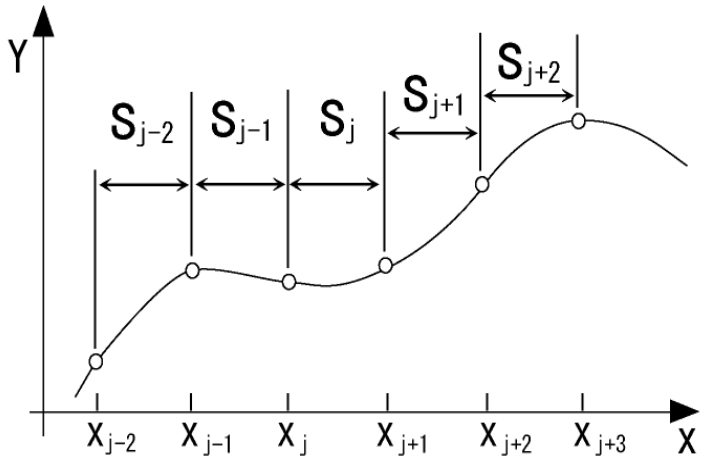


図 3.15　スプライン補完

補完に使用する 3 次関数は次の式で表される．

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \quad (3)$$

図 3.13 に示すように，補完するデータを$(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$とし，区間$[x_j, x_{j+1}]$で補完する関数を$S_j(x)$とする．

ここで，式(3)において，各区間が滑らかに接続するa_j, b_j, c_j, d_jを求めればよい．なお，関数$S_j(x)$を一意に決定するために，式(4)の５つの前提条件を与える．

$$
\begin{aligned}
&\textbf{前提条件１：}\ S_j(x_j) = y_j \\
&\textbf{前提条件２：}\ S_j(x_{j+1}) = S_{j+1}(x_{j+1}) = y_{j+1} \\
&\textbf{前提条件３：}\ S_j'(x_{j+1}) = S_{j+1}'(x_{j+1}) \\
&\textbf{前提条件４：}\ S_j''(x_{j+1}) = S_{j+1}''(x_{j+1}) \\
&\textbf{前提条件５：}\ S_0''(0) = S_{n-1}''(x_n) = 0
\end{aligned}
\qquad (4)
$$

以上の定義式から求めた$S_j(x)$を連結したものを，3 次スプライン曲線とする．

3.2.3　樹木の模様生成

ベジェ曲線とスプライン曲線を使用する理由として，頂点間の補完点を求めることで，滑らかな樹木模様が表現できることが挙げられる．樹木の幹や枝の模様は，図 3.16(a)に示すように各頂点間に矩形を配置して描画するため，頂点間の距離が遠いと得られる形状の角が目立つ．そこで，スプライン補完で補完点を求め，図 3.16(b)に示すように滑らかな形状を描画する．

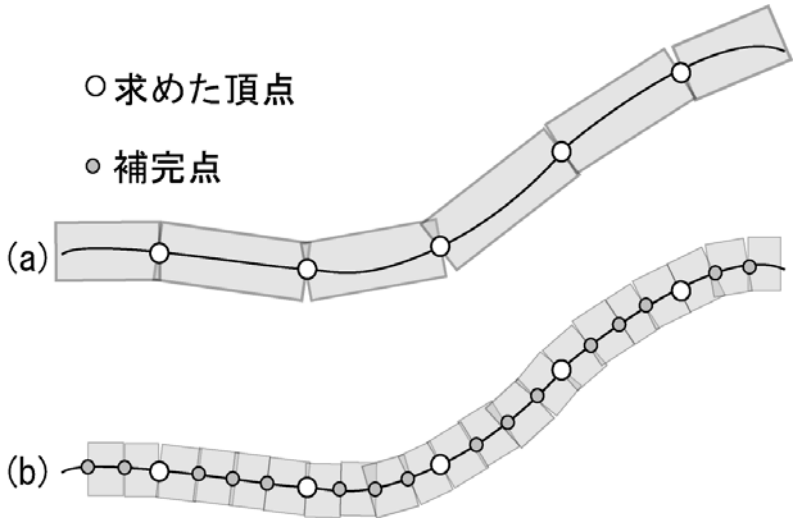
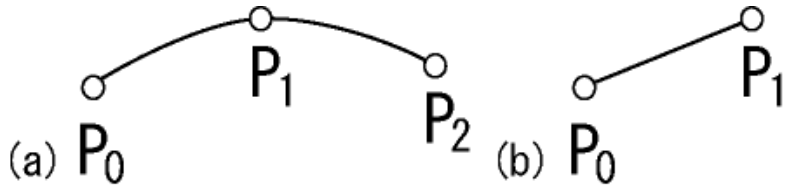


図 3.16　ライン描画

曲線近似の種類は，以下に示すような場合分けをした．

2 点を結ぶ場合　　：3 次ベジェ曲線
3 点以上を結ぶ場合　　：スプライン曲線

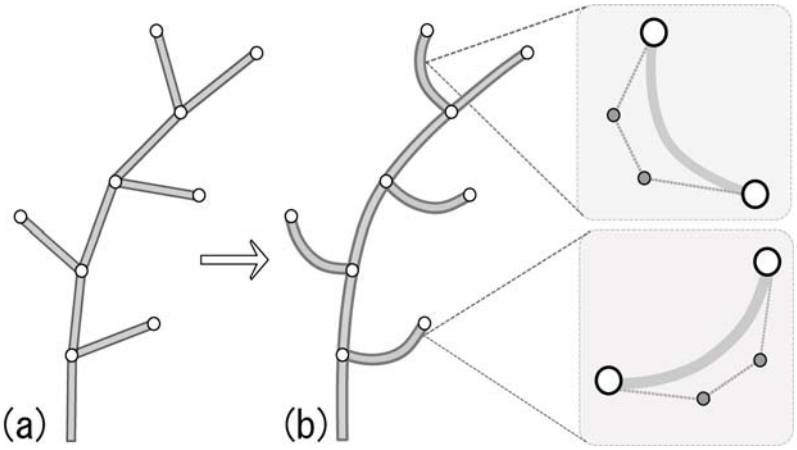
この場合分けの理由として，スプライン曲線では，図 3.17(a)に示したように，3 点以上を補完する場合は，それぞれの点を通る曲線を求めることができるが，2 点の場合は，直線として補完される．このため，2 点の場合は，ベジェ曲線を使うことにした．ベジェ曲線には，4 点のうち，端の 2 点のみを補完曲線が通り，その他の 2 点を通らない特徴がある．また，通らない 2 点を動かすことにより，端の 2 点を通る様々な曲線を生成することができる．したがって，2 点間を補完する場合には，新たに 2 点の制御点を求めて，表現力の高いベジェ曲線を用いて補完する．



(a)3 点の場合　　(b)2 点の場合

図 3.17　スプライン補完

図 3.18 に曲線補完の効果を示す．図 3.18(a) は頂点間を直線で結んだ場合を，図 3.18(b)は，曲線補完を施した場合を表している．

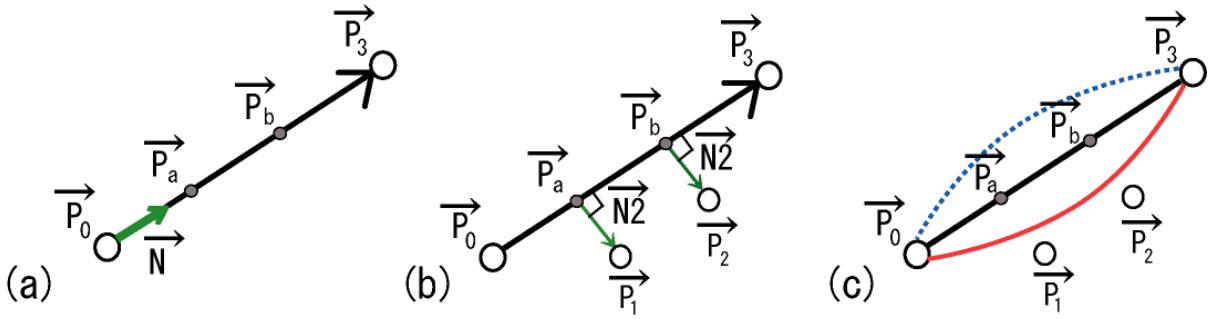


(a)補完前　　(b)補完後

図 3.18　補完木

図 3.18(a)の場合，木の幹と枝の形状が共に角ばった印象を与え，実際の模様とは

かけ離れている．一方，図 3.18(b)に示すように曲線補完を施すと，より樹木の模様らしい表現になる．なお，ベジェ曲線を表現するには，以下に示すような新たな 2 点を求める必要がある．



(a)Pa，Pb，N の求め方　　(b)P1，P2 の求め方　　(c)求めたベジェ曲線

図 3.19　ベジェ曲線の制御点の求め方

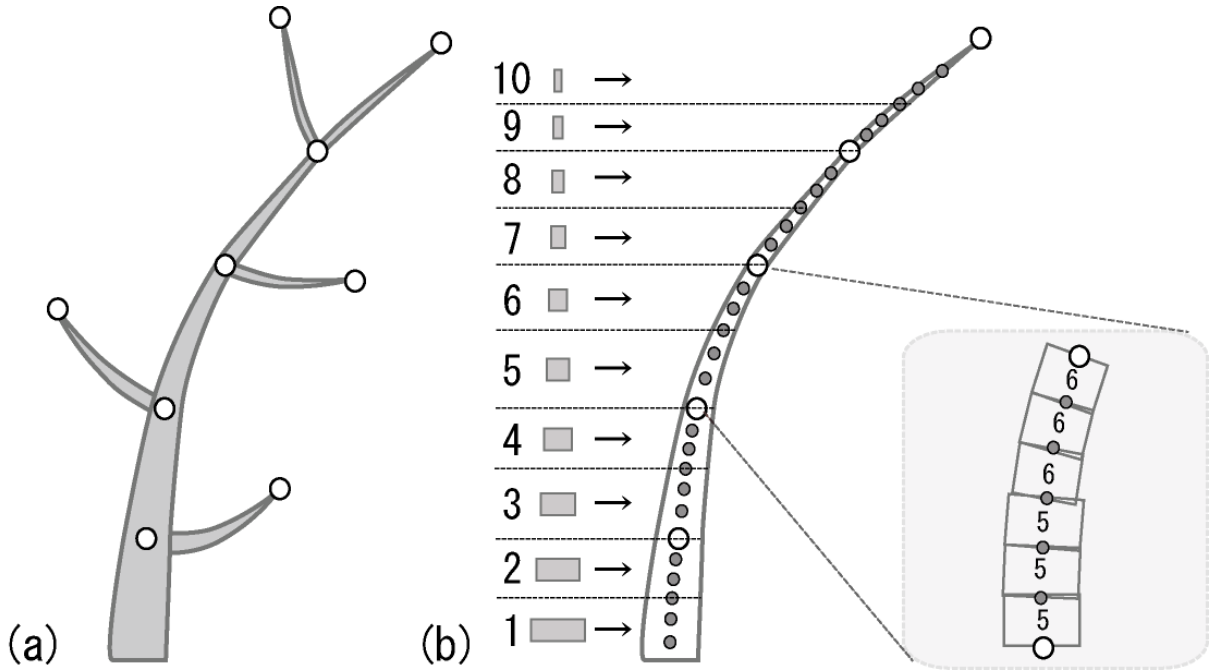
各点を，$\vec{P0}$，$\vec{P1}$，$\vec{P2}$，$\vec{P3}$，$\vec{Pa}$，$\vec{Pb}$ でベクトル表記する．枝の生成で求めた 2 点を $\vec{P0}$，$\vec{P3}$ とし，求めたい新たな 2 点の制御点を $\vec{P1}$，$\vec{P2}$ とする．まず，図 3.19(a)の $\overrightarrow{P0P3}$ を 3 等分したときの内分点である $\vec{Pa}$，$\vec{Pb}$ を，式(5)で求める．次に式(6)で示したように，ベクトル $\overrightarrow{P0P3}$ を正規ベクトルを $\vec{N}$ とする．次に，$\vec{N}$ を時計周りに 90 度回転させたベクトル $\vec{N2}$ を求め，$\vec{P1}$，$\vec{P2}$ を式(7)を用いて算出する．求めた新たな制御点 $\vec{P1}$，$\vec{P2}$ を用いて，最終的に，図 3.19(c)にの赤ラインで示したベジェ曲線が得られる．なお，図 3.19(c)で示した赤のベジェ曲線は式(7)の右の枝の場合であり，左の枝の場合は図 3.19(c)の青ラインの配置となる．ただし，枝の左右の別は，幹の伸長方向に対する向きとする．

$$\begin{aligned}\vec{P_a} &= \frac{2}{3}\vec{P_0} + \frac{1}{3}\vec{P_3} \\ \vec{P_b} &= \frac{1}{3}\vec{P_0} + \frac{2}{3}\vec{P_3}\end{aligned} \quad (5)$$

$$\begin{aligned}\vec{N} &= Normalize(\overrightarrow{P_0P_3}) \\ \vec{N2} &= Rotate(\vec{N}, 90)\end{aligned} \quad (6)$$

$$\overrightarrow{P_1} = \overrightarrow{P_a} \pm 6.0 * \overrightarrow{N2}$$
$$\overrightarrow{P_2} = \overrightarrow{P_b} \pm 6.0 * \overrightarrow{N2} \quad (7) \quad ((+)\text{枝が右} \quad (-)\text{枝が左})$$

曲線による模様表現の質は向上できたが，木の幹や枝の長さが均一であるため，さらなる改善が必要であると考えた．そこで，枝や幹の太さを段階的に細くして表現することにした．



(a)枝と幹の太さの表現　　(b)太さの表現方法

図 3.20　枝と幹の表現

図 3.20 (a)に示すように，木の幹の太さを根元から遠ざかるにつれて徐々に細くする．具体的には，図 3.20(b)で示すような処理を行い，さらに樹木の模様らしい表現を実現した．各処理を以下に述べる．

① 求めた補完点を 10 個の領域に分割する．

② 縦の長さは同じで，横の幅を変えた矩形を 10 個用意する．

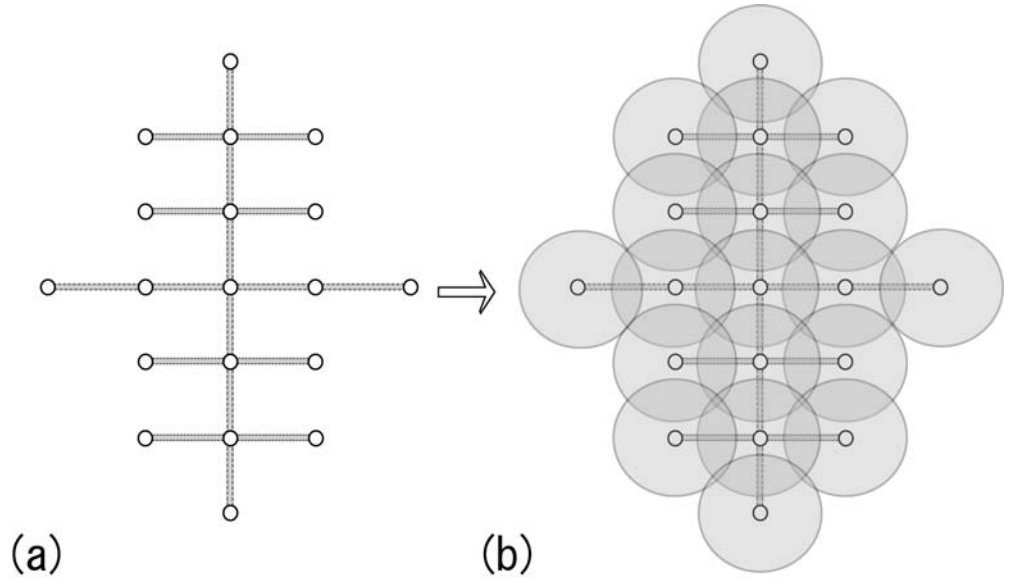
③ 図 3.20(b)で示した「1」の領域に一番大きな矩形を配置し，各領域を水平方向に一定の縮小率でスケーリングし，徐々に小さな矩形を配置する．枝も同様の処理を施す．

3.3　背景模様生成

本節では，背景の模様生成に用いる合成手法について述べる．

3.3.1　　マスク画像生成

マスク画像生成には，3.1 節で述べた L-System を利用する．まず，図 3.21(a)に示すような木を生成し，生成形状の各頂点に対して，図 3.21(b)に示すようにあらかじめ与えられた画像を描画しマスク画像とする．図 3.21(b)の例では円を描画しているが他の図形でも可である．ここで，木の枝や幹などは，説明のために表示したものであり，実際には表示しない．

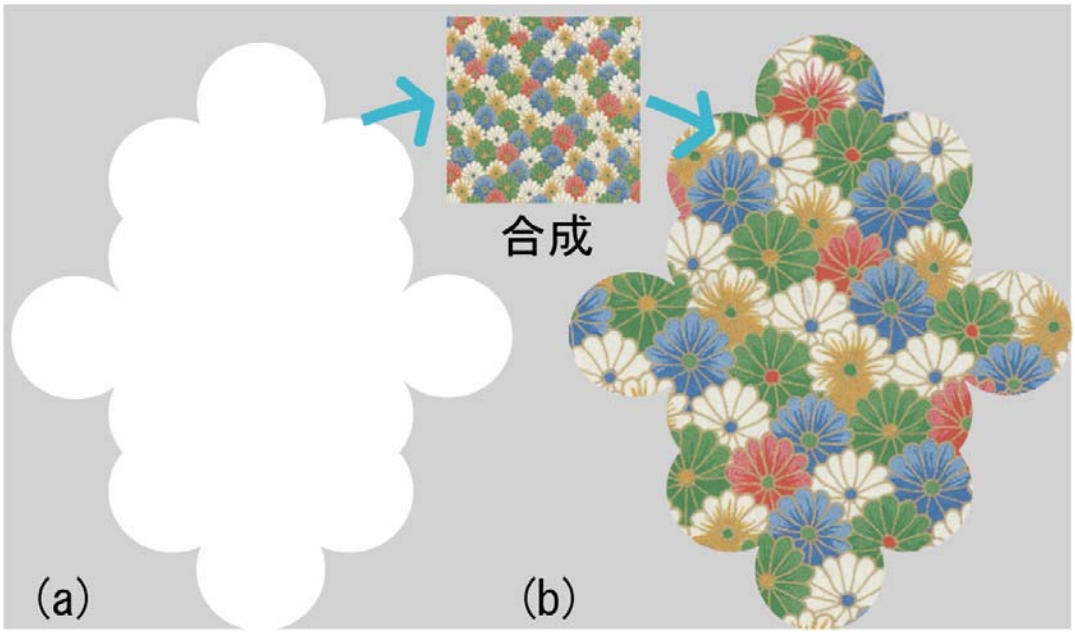


(a)基礎に使用する木　　(b)円を描画した画像

図 3.21　マスク画像の生成

3.3.2　　マスク合成

図 3.22(a)に示すマスク画像に対して，好みの画像を合成することで図 3.22(b)に示す合成結果を得る．ここで，図 3.22(a)の白領域部分にのみ，模様を合成する．生成する木やマスク画像を変更することで，様々な形の合成模様を生成できる．



(a)白円で描画したマスク画像　　(b)マスク合成結果

図 3.22　マスク合成の例

第　4　章

ユーザインターフェイス

本章では，着物模様のデザイン支援ツールのユーザインターフェイスに関して述べる．

4.1　ツールの全体デザイン

本研究で実装したユーザインターフェイスは，図 4.1 に示すような模様をデザインする作業エリアと，図 4.2 に示すツールボックスで構成されている．ツールボックスは，レイヤー背景用とそれ以外のレイヤー用の 2 種類を用意する．以降，図 4.2(a)のツールボックスを「ツールボックス 1～7]に分割，図 4.2(b)のツールボックスを「ツールボックス 7～9」に分割して述べる．

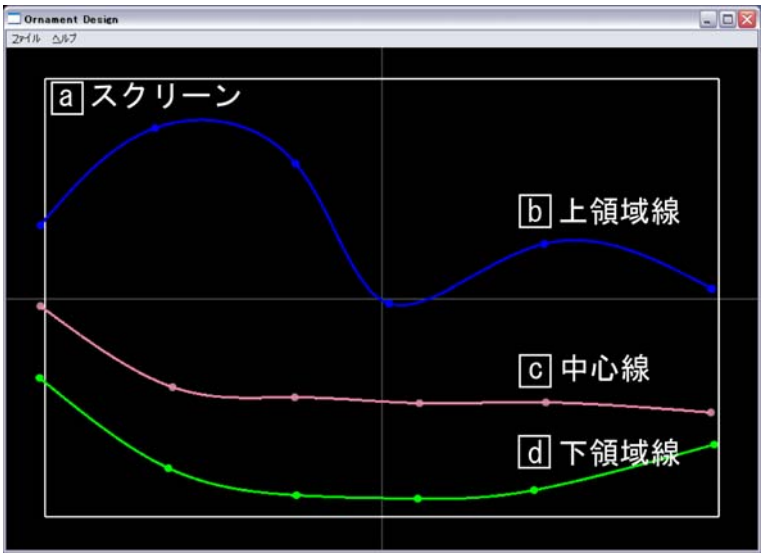
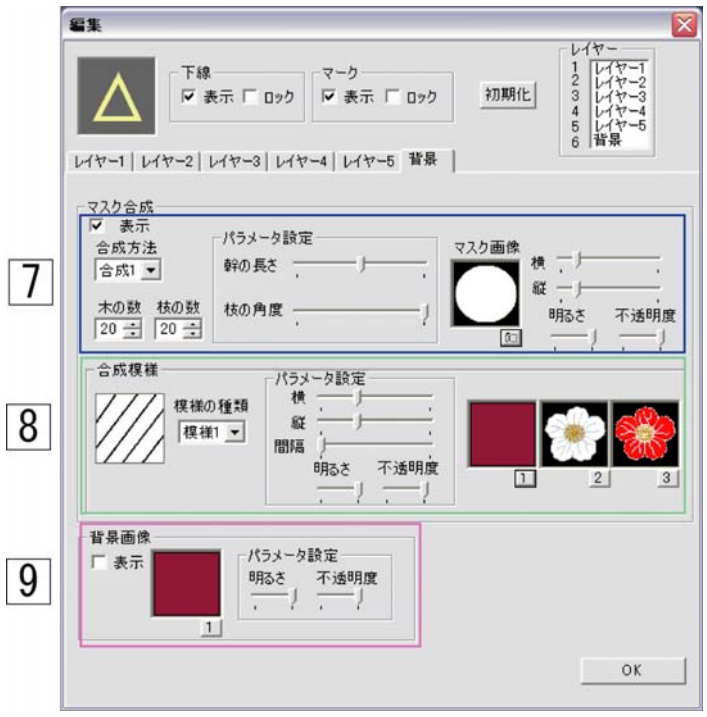


図 4.1　作業エリア

(a) ツールボックス[1-6](レイヤー1～5 用)



(b) ツールボックス[7-9](レイヤー背景用)

図 4.2 ツールボックス

4.2　作業エリア

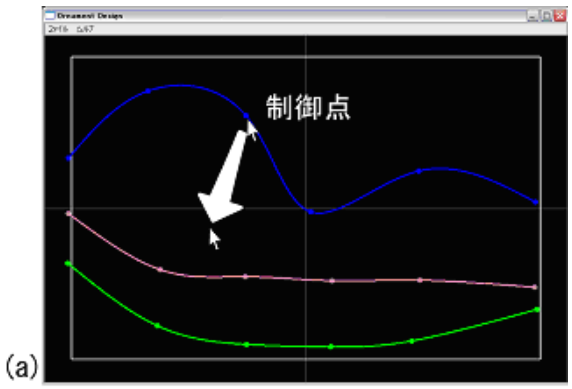
本節では，図 4.1 に示した作業エリアの構成要素について述べる．

4.2.1　　スクリーン

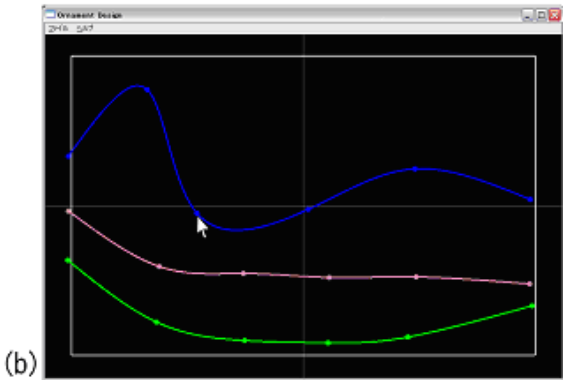
図 4.1 の[a]で示した四角の枠の部分が，模様を生成するためのスクリーンとなる．ユーザはこの枠内に模様を生成しなければならない．

4.2.2　　上領域・下領域・中心線の操作

図 4.1 の[b]～[d]の各線は，3.2 節で述べた 3 次スプライン曲線で表現する．曲線の制御点は，複雑な形状にも対応できるように 6 個に定めた．図 4.3(a)のように，ユーザは制御点をマウスでドラッグして，自由に曲線の形状を変更する．図 4.3(b)に曲線の操作の後を示す．



(a)移動前



(b)移動後

図 4.3 曲線の操作の様子

4.3　ツールボックス 1

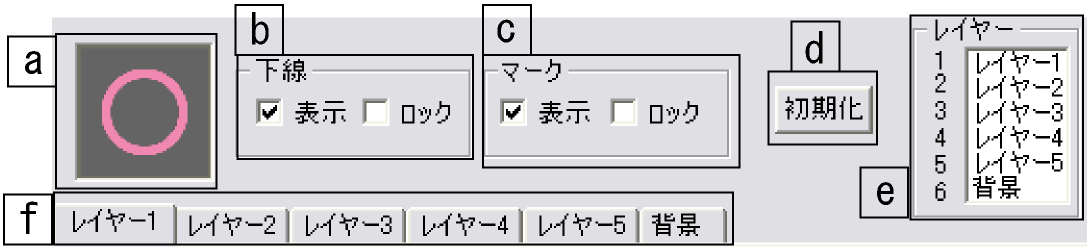


図 4.4　ツールボックス 1

4.3.1　　レイヤーマーク

図 4.4-[a]にレイヤーを区別するためのマークを表示する．図 4.5 に示すような各記号を生成した樹木の模様の根元に表示し，レイヤーの判別を容易にする．レイヤー1の操作画面を図 4.6 に示す．

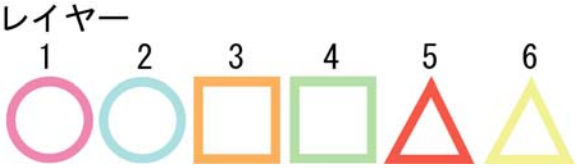


図 4.5　レイヤー別のマーク

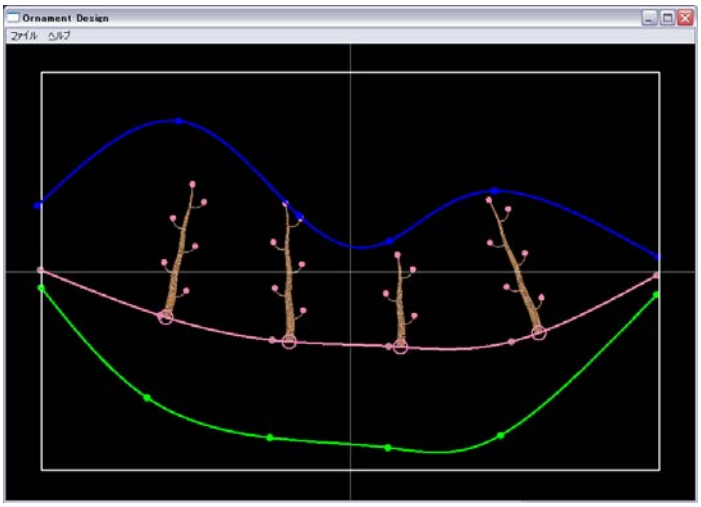


図 4.6　レイヤー 1 の画面

4.3.2　　下線

図 4.4-[b]では，下線の表示・移動を制御する．

・　「表示」: チェックすることで，図 4.6 に示すように，下線（上領域・中心・下領

域線）や，スクリーンの中心点を縦横に通るグリッド線を表示する．図 4.7 に「表示」のチェックを外した場合のスクリーンの様子を示す．

図 4.7　下線の消去

・　「ロック」: チェックすることで，スクリーン内の各制御点を固定し，操作不可とする．これは，ユーザの意図に反して，制御点が移動することを避けるために設けた．

4.3.3　マーク

図 4.4-[c] では，マークの表示・移動を制御する．

・　「表示」: チェックすることで，レイヤーマークを表示する．
・　「ロック」: チェックすることで，レイヤーマークを固定する．

4.3.4　初期化

図 4.4-[d] では，スクリーンの状態を制御する．

・　「初期化ボタン」: 図 4.8 に示すように樹木の模様をすべて消去し，下線を初期値へ戻す．

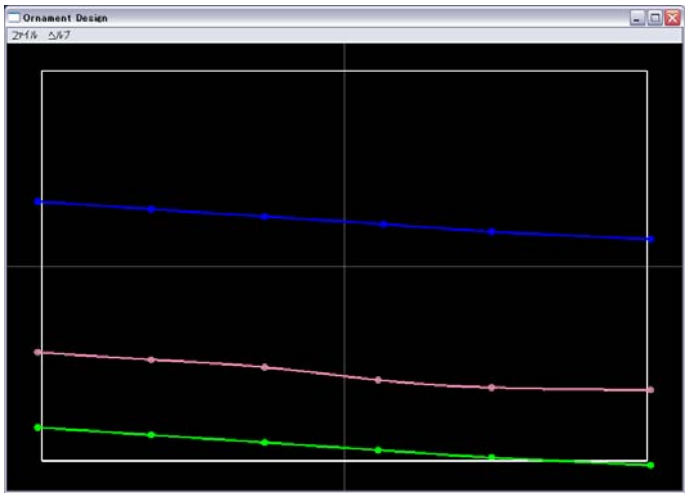
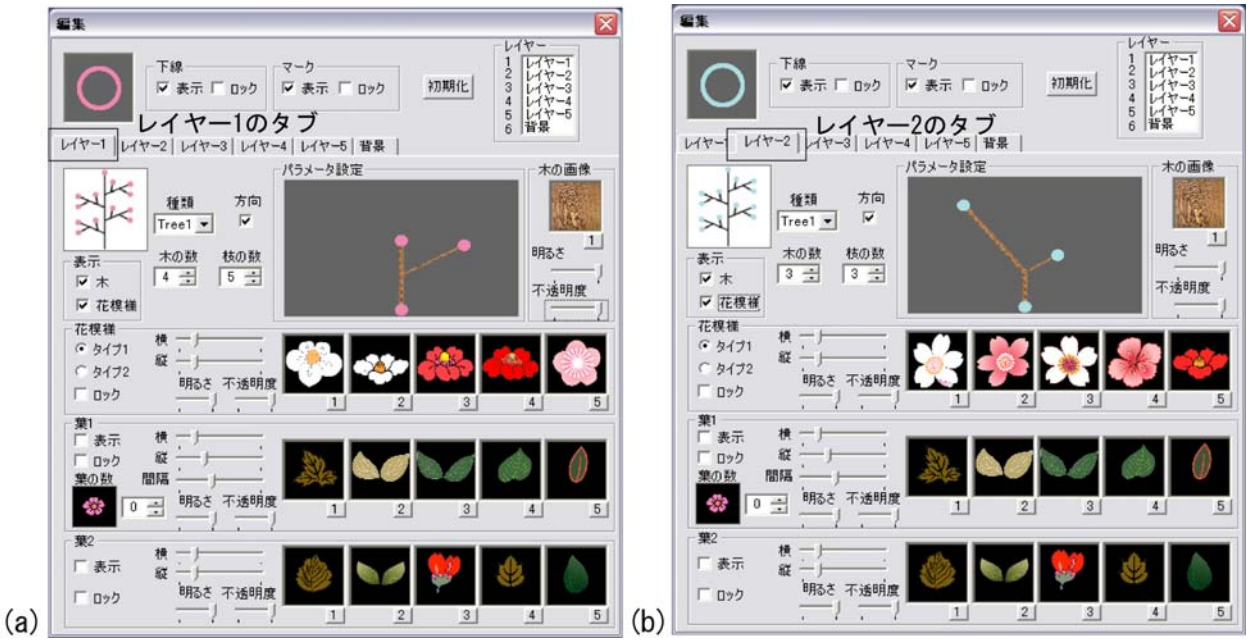


図 4.8　初期化画面

4.3.5　レイヤー画面

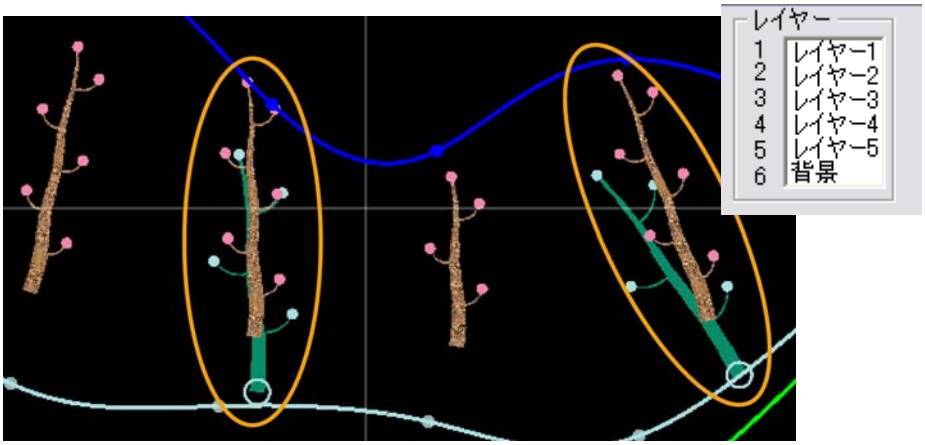
図 4.4-[e][f] では，レイヤー別データ・レイヤーの前後関係を制御する．図 4.9 に示したタブ機能により，レイヤー別にデータを管理して模様を生成する．例えば，レイヤー1 では樹木の種類を Tree1 とし，レイヤー2 では Tree2 に指定できる．



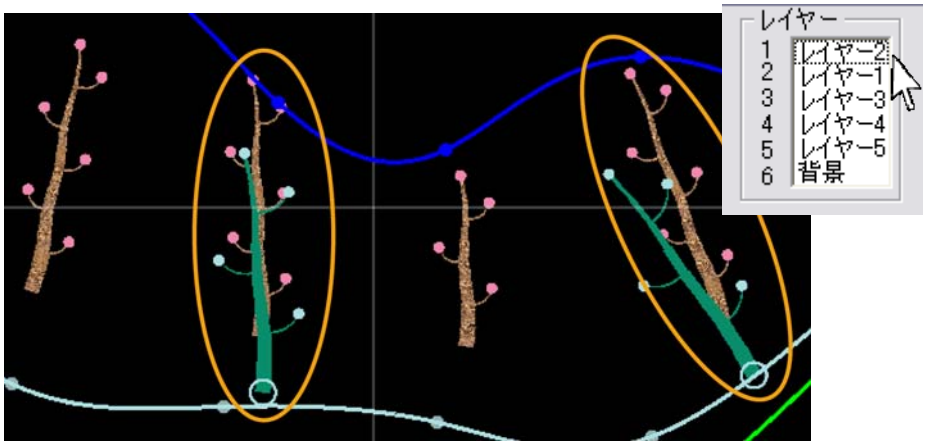
(a) レイヤー1　　(b) レイヤー2

図 4.9　ツールボックス

また，レイヤー機能により，模様の前後関係を変更することも可能である．図 4.10 では，レイヤー１の 4 本の茶色の樹木の模様と，レイヤー2 の 2 本の緑色の樹木が描画されている．図 4.10(a)では，レイヤーの順番がレイヤー1，レイヤー2 の順になっており，楕円で示した部分では，レイヤー１の樹木を前面に描画する．この状態から，図 4.10(b)のように，レイヤーの順番を入れ替えると，図 4.10(b)の楕円で示した部分のように，樹木の前後を入れ替えて描画する．



(a) レイヤー1 が手前



(b) レイヤー２が手前

図 4.10 レイヤー構造の表示例

4.4　ツールボックス 2

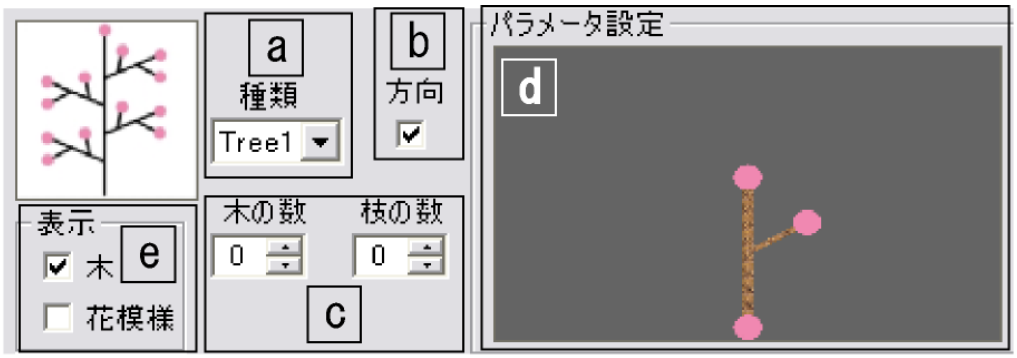


図 4.11　ツールボックス 2

4.4.1　　樹木の模様の種類

図 4.11-[a]のプルダウンメニューにより，図 4.12 に示すような生成する木の種類を選択する．

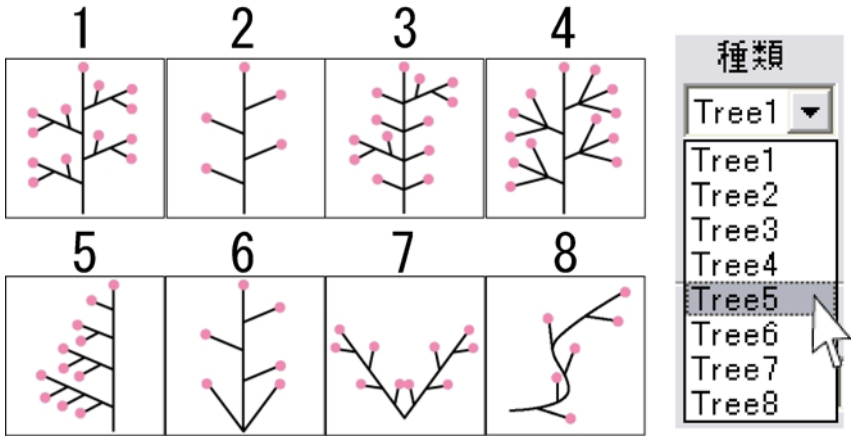


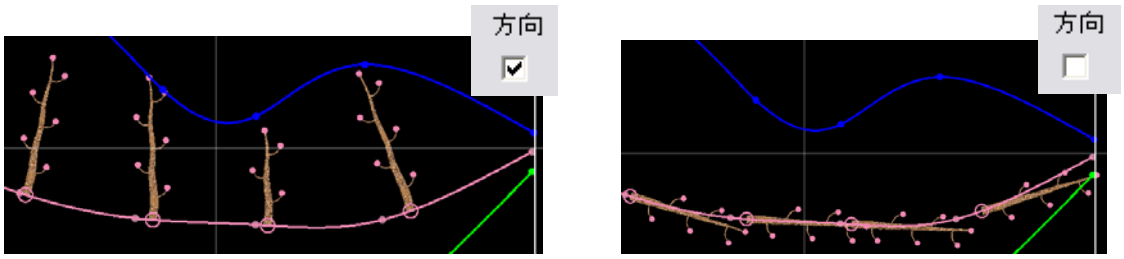
図 4.12　樹木模様の選択の様子

4.4.2　　樹木の生成方向

図 4.11-[b]のチェックボックスは，樹木の生え方を変更する．

- チェックされている場合：木が中心線から垂直に生える
- チェックされていない場合：木が中心線に平行に生える

その様子を図 4.13 に示す.



(a)垂直　　(b)水平

図 4.13　樹木の生成方向の指定

4.4.3　樹木の設定

図 4.11-[c]は，生成する木の数と枝の数を指定する．ここで，木の数は 0～10 に，枝の数は 0～20 に制限する.樹木は生成条件により上領域よりも下に生成されるため，図 4.14 に示すように，①の木では枝の数が 5 個あるのに対して，②の木では上領域の制限を受けて枝の数が 5 個まで表示されず，枝の数が 3 つまでとなっている.

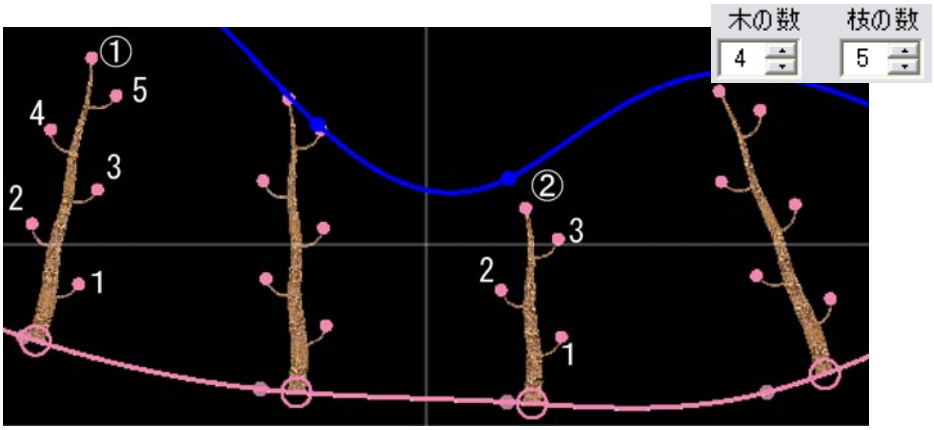
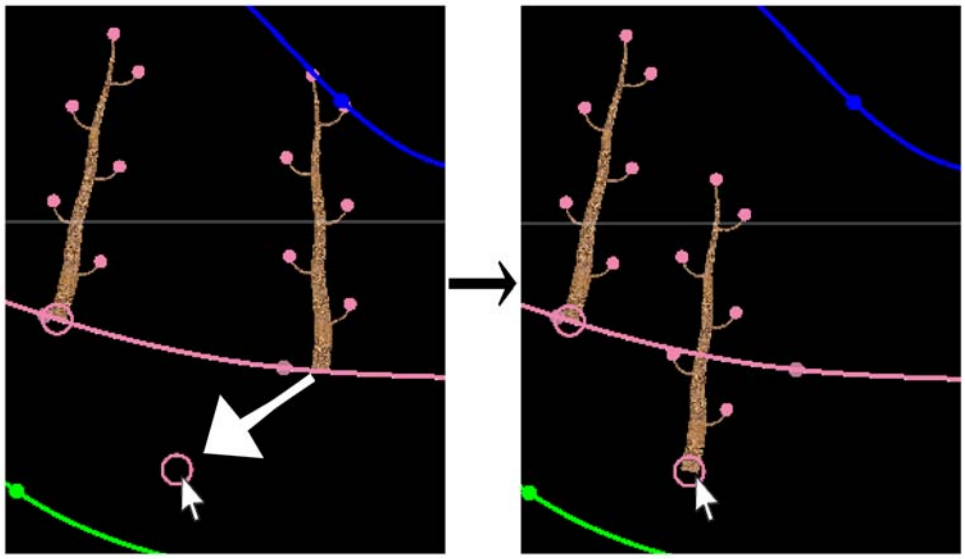


図 4.14 木・枝の数の指定

また，生成された木模様は，図 4.15 に示すようにレイヤーマークをドラッグすることにより，自由な場所へ移動することができる.



(a)移動前　　(b)移動後

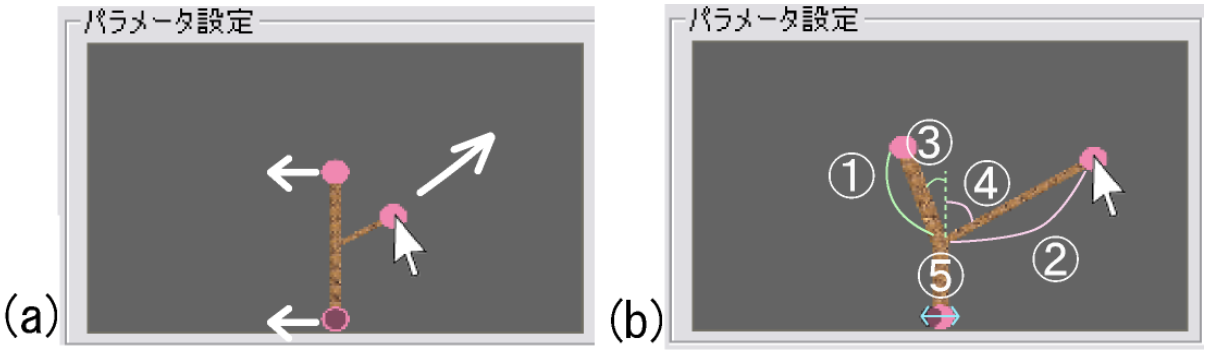
図 4.15　樹木の模様の移動の様子

4.4.4　樹木のパラメータ設定

図 4.11-[d]のパラメータ設定では,樹木の生成パラメータを変更することができる.パラメータは，図 4.16 に示す簡易モデル木を直接操作することで設定する．設定する樹木のパラメータには，図 4.16(b)の図中の記号に対応した，以下の 5 つの要素がある.

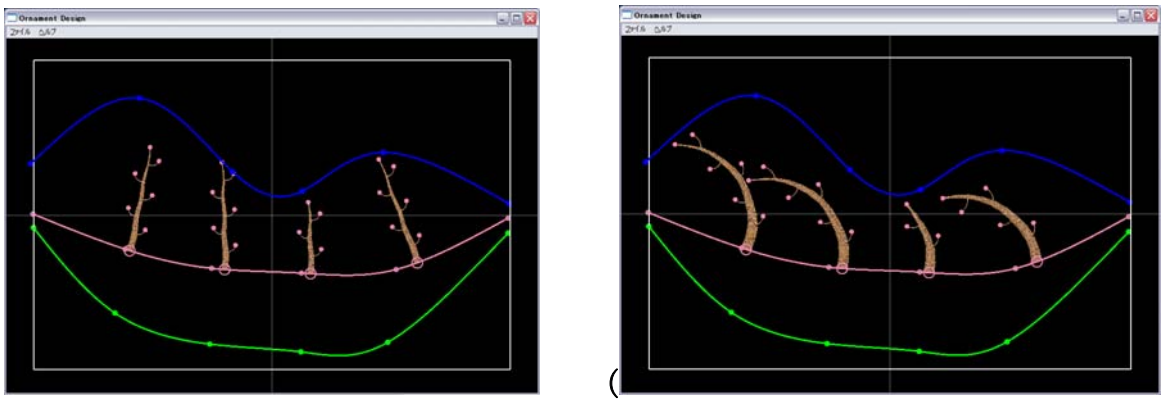
①木の幹の長さ
②木の枝の長さ
③木の幹の角度
④木の枝の角度
⑤木の太さ

図 4.16(a)に示すように，簡易モデル木の円で示した制御点の移動に伴い，木模様は様々な形状に変化する．図 4.17 に，変化した木模様を示す.



(a)変更前　　(b)変更後

図 4.16　パラメータ設定



(a)変更前　　(b)変更後

図 4.17　パラメータ変更

図 4.16 の②の枝の長さを長くしたため，図 4.17(a)と比較して図 4.17(b)の枝は長くなっている．さらに，③の幹の角度を変更し⑤の幹の太さを変更したため，すべての樹木が左に曲がり，幹の太さが太くなっている．

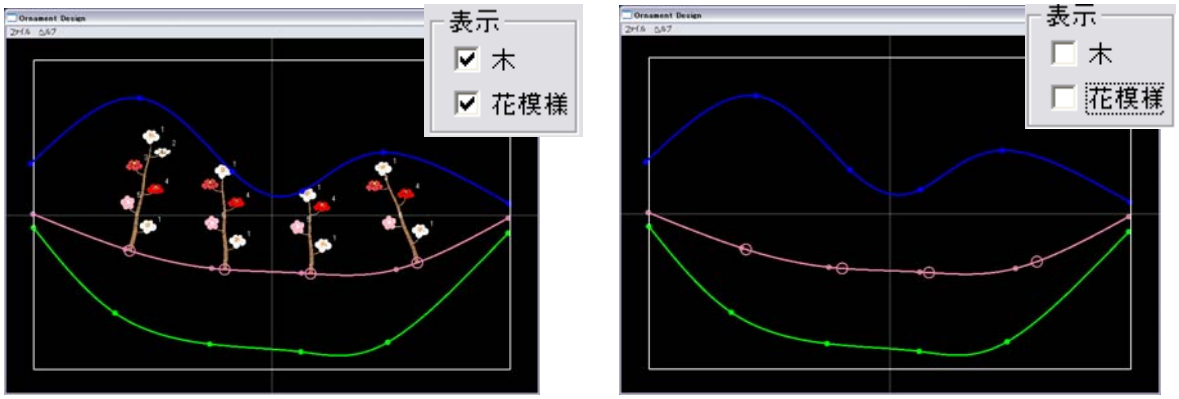
4.4.5　表示

図 4.11-[e]は，木の表示，画像の表示を決定する．

・「木」　　：チェックされている場合，生成された木が表示される

・「花模様」　　：チェックされている場合，木模様の頂点に画像が表示される

図 4.18 に，その操作画面の様子を示す．



(a)木・花模様にチェック　　(b)チェックなし

図 4.18 表示のチェックの有無

4.5　ツールボックス 3

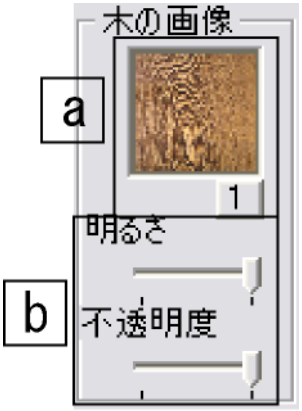


図 4.19 ツールボックス 3

4.5.1　木の画像の選択

図 4.19-[a]では，樹木の模様の幹と枝の画像の設定をする．ここで選択した画像は，図 4.20 に示すように，それぞれの樹木の幹や枝の模様に割り当てられる．また，ユーザは，画像の右下のボタンを押すことにより，画像ファイルを選択することができ，様々な画像に変更することができる．その操作画面の様子を図 4.21 に示す．

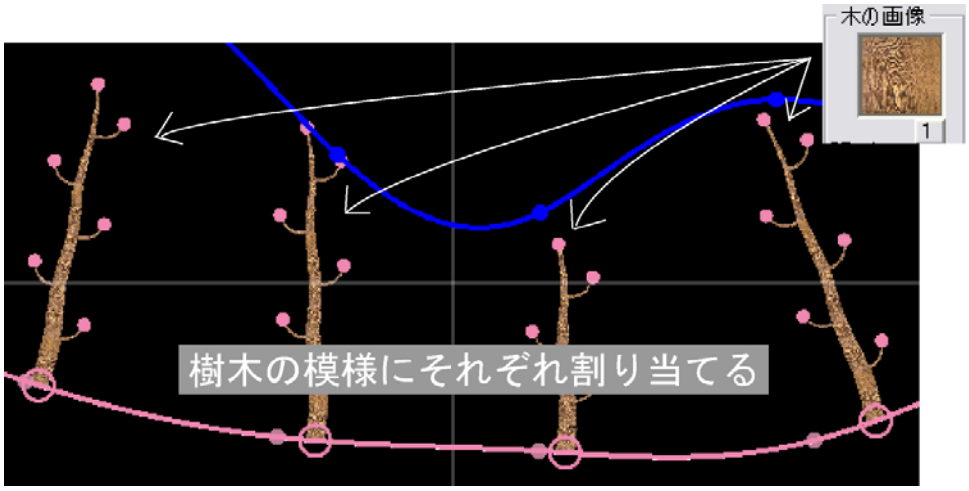


図 4.20　樹木の画像の割り当て

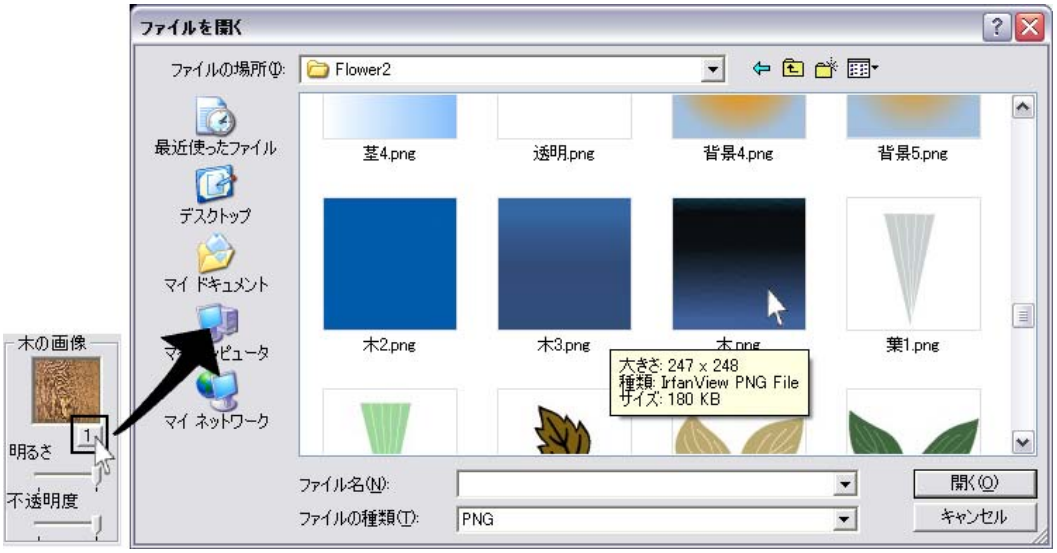
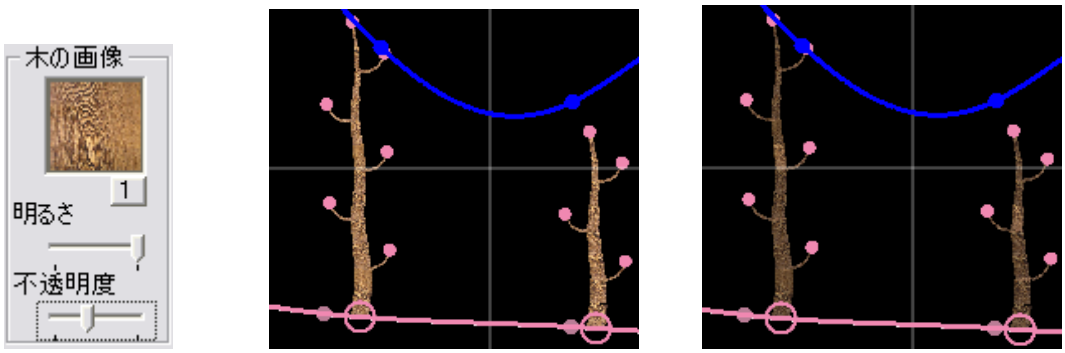


図 4.21　木の画像の選択

4.5.2　　木の画像のパラメータ設定

図 4.19-[b]では，樹木の画像の明るさや不透明度を設定する

図 4.22(a)のように，不透明度の値を下げることで，図 4.22(b)の状態から図 4.22(c)の状態へと樹木模様の透明度が変化する．



(a)透明度の変更　　(b)変更前　　(c)変更後

図 4.22　不透明度の設定

4.6　ツールボックス 4

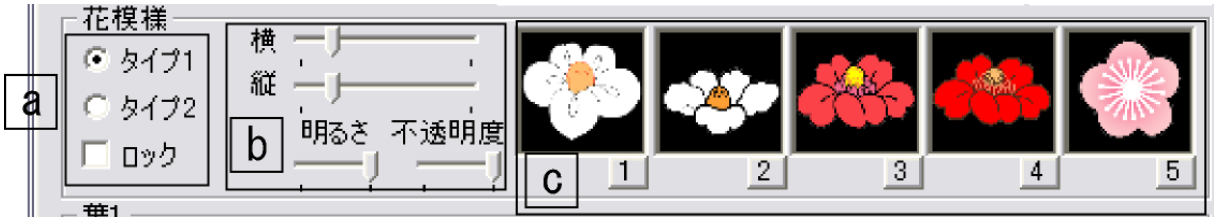


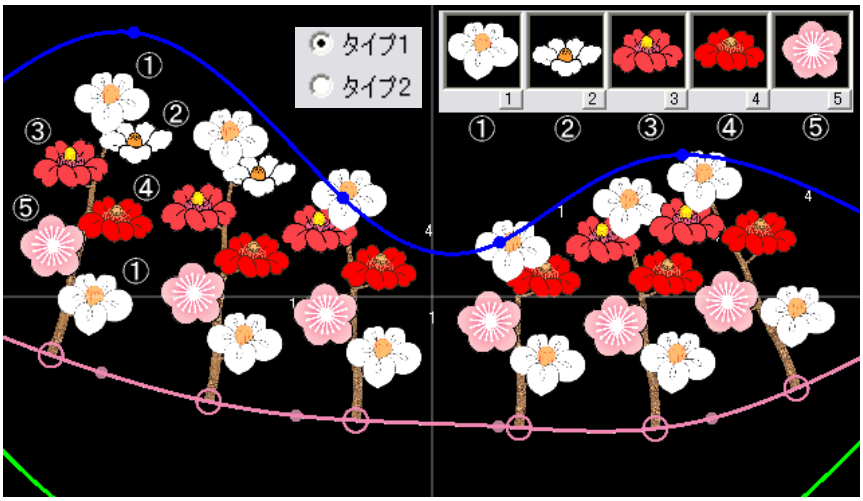
図 4.23　ツールボックス 4

4.6.1　　花模様－タイプ・ロック

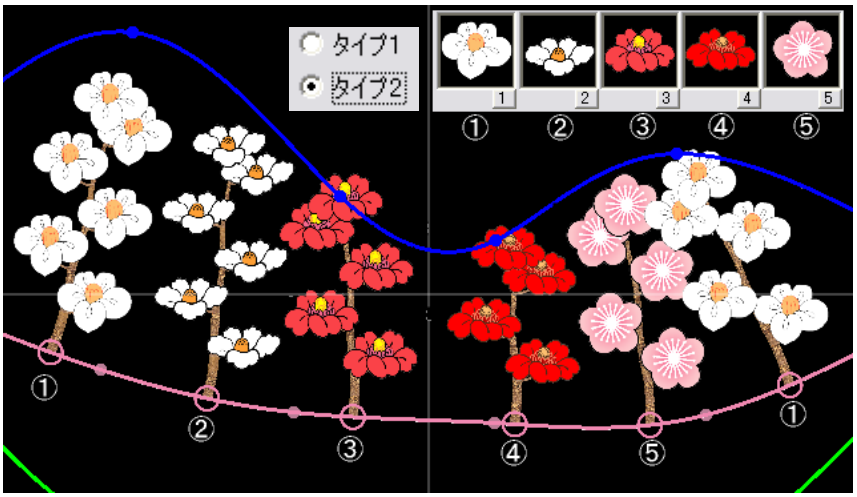
図 4.23-[a]の「タイプ」では，樹木の模様に花の画像を付加する場合の，以下の 2 通りの表示法を指定する．

タイプ 1　：それぞれの樹木の模様において，花の画像を貼り付ける．
タイプ 2　：樹木の模様ごとに花の画像の割当てを変えて，貼り付ける．

図 4.24 に，その操作画面の様子を示す．



(a) タイプ 1



(b) タイプ 2

図 4.24 花の画像の貼り付けの違い

図 4.24(a)では，すべての木が同じ順に花の画像を表示している．ここで，花の画像は 5 つまでしか指定できないため，図 4.24(a)のように，花の画像を表示する枝が 6 つ以上の場合は，6 つ目以降は 5 の剰余＋ 1 にあたる画像を割り当てる．

図 4.24(b)では，すべての木が同じ種類の花の画像を表示している．ここでも，花の画像は 5 つまでしか指定できないため，木の数が 6～10 本の場合は，6 本目以降は 5 の剰余+1 にあたる画像を割り当てる．

図 4.23-[a]の「ロック」では，花の画像の移動を制御する．

「ロック」

・ チェックされている場合　　：花の画像は移動しなくなる

・ チェックされていない場合　：花の画像は移動できる

4.6.2　花模様のパラメータ設定

図 4.23-[b]では，図 4.25 に示すように花の画像のパラメータの「横」および「縦」の大きさを変更できる．また，画像の明るさや不透明度の設定も行なうことができる．

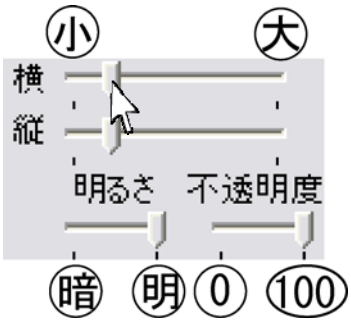
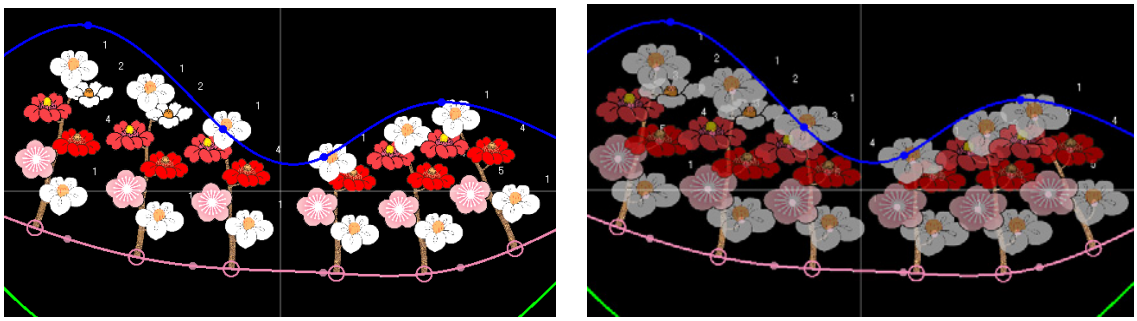


図 4.25　花の画像のパラメータ



(a)変更前　　(b)変更後

図 4.26　花の画像のパラメータ変更の例

図 4.26(a)の状態から，花画像の横幅を大きくし不透明度を下げると，図 4.26(b)に示すように，横に大きく透明な花画像を得る．

4.6.3　花の画像選択

図 4.23-[c]では，花の画像を設定する．図 4.27 に示すように，各画像の右下のボタンを押すことで，好みの花の画像のファイルを選択する．

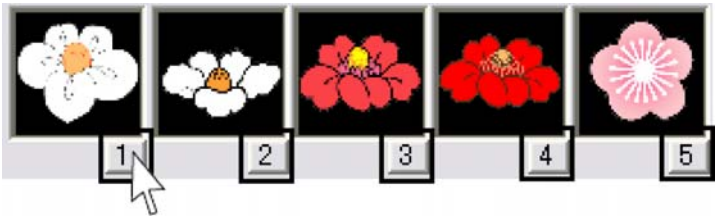


図 4.27　花の画像リスト

4.7　ツールボックス 5

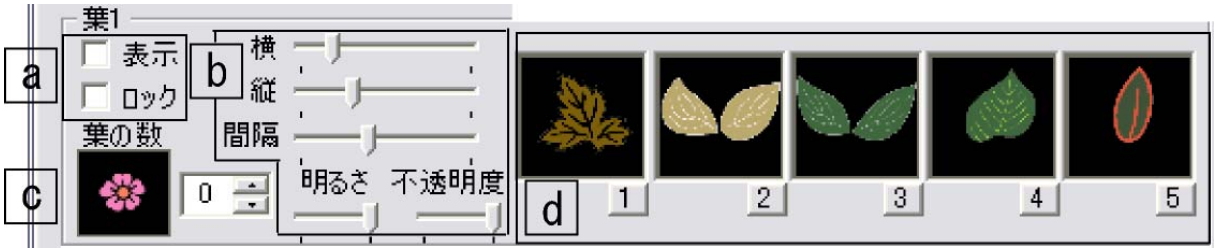
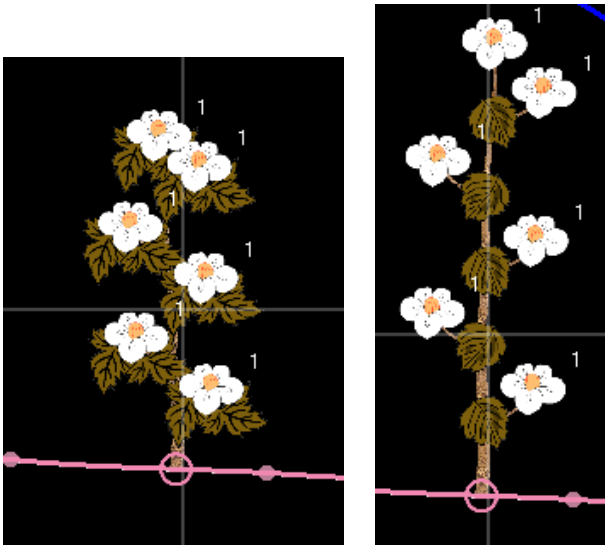


図 4.28　ツールボックス 5

4.7.1　　葉の画像の設定

樹木の模様に，葉の画像を割り当てる場合，2 つの方法がある．それぞれの割り当て方法を葉 1 の割り当て法，葉 2 の割り当て法と定義する．



(a)葉 1　　(b)葉 2

図 4.29　葉の画像の設定

・ 葉 1 の割り当て法

図 4.29(a)に示すように，それぞれ樹木の花の模様の周りに葉の画像を配置する．

・ 葉 2 の割り当て法

図 4.29(b)に示すように，樹木の枝の付け根の部分にそれぞれの葉の画像を配置する．

4.7.2　葉 1－表示・ロック

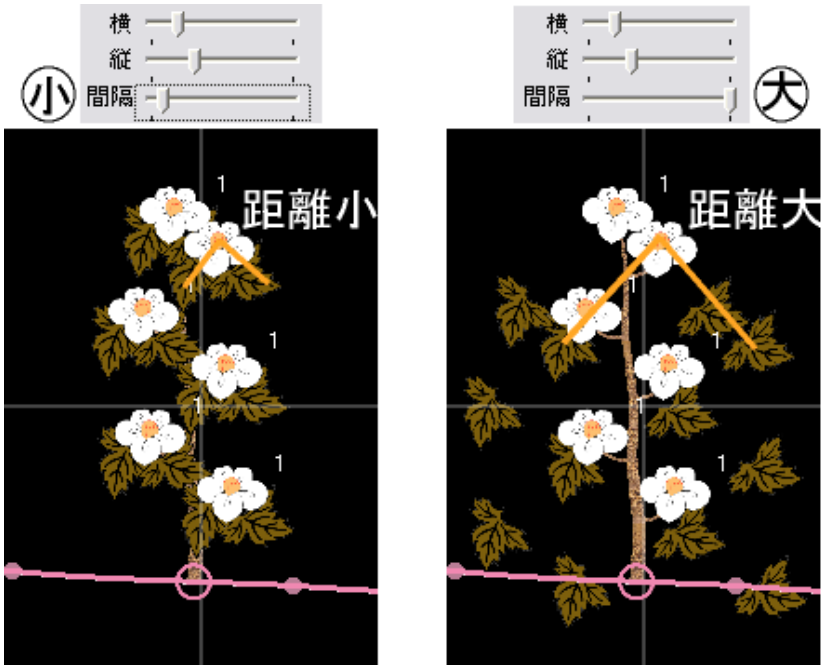
図 4. 28-[a] では，葉 1 の表示・移動を制御する．

・ 「表示」: チェックすることで，葉 1 を表示する．

・ 「ロック」: チェックすることで，葉 1 を固定する．

4.7.3　葉 1－パラメータ設定

図 4. 28-[b]では, 葉 1 の画像の横および縦の大きさと, 花の画像までの距離を示す「間隔」のパラメータを制御する．また，葉 1 の画像の明るさと透明度も設定する．本項では，「間隔」パラメータに関してのみ述べる．



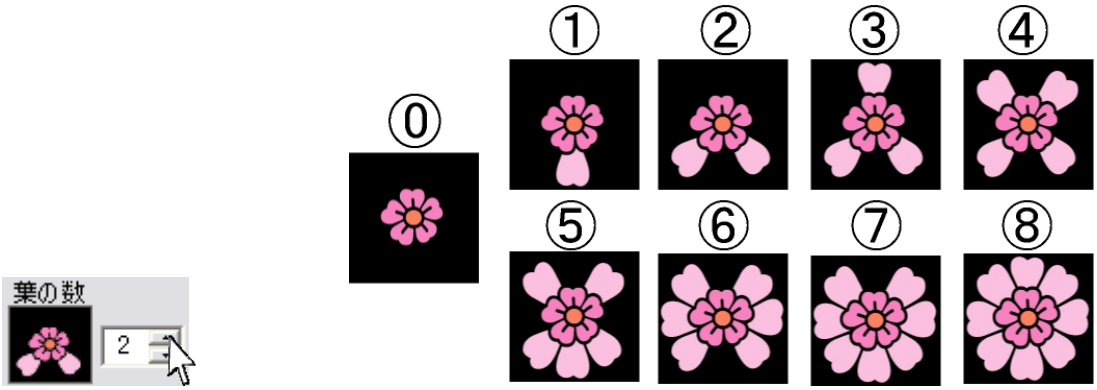
(a) 距離小　　(b) 距離大

図 4. 30 葉 1-パラメータの設定例

図 4.30(a)では，間隔パラメータを小さくしたため，花の模様と葉 1 画像との距離が縮まっている．逆に，図 4.30(b)では，間隔のパラメータを大きくしたため，花の模様と葉 1 の画像との距離が開いている．

4.7.4　葉 1－葉の数

図 4.28-[c]では，葉の周りに配置する葉の数を指定する．図 4.31(a)のカウンタをクリックすることで 0～8 枚までの葉の数を指定する．それぞれの葉の枚数に対応した葉の配置を，図 4.31(b)に示す．



(a)葉の数　　(b)葉の種類

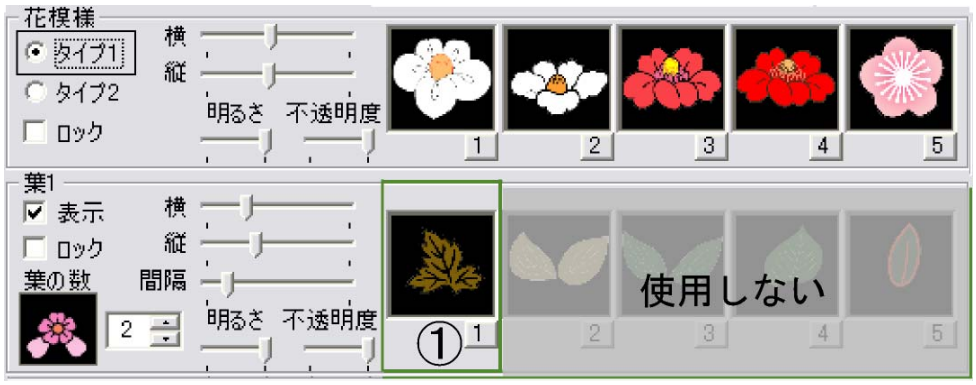
図 4.31　葉 1-葉の数と配置法

4.7.5　葉 1－画像

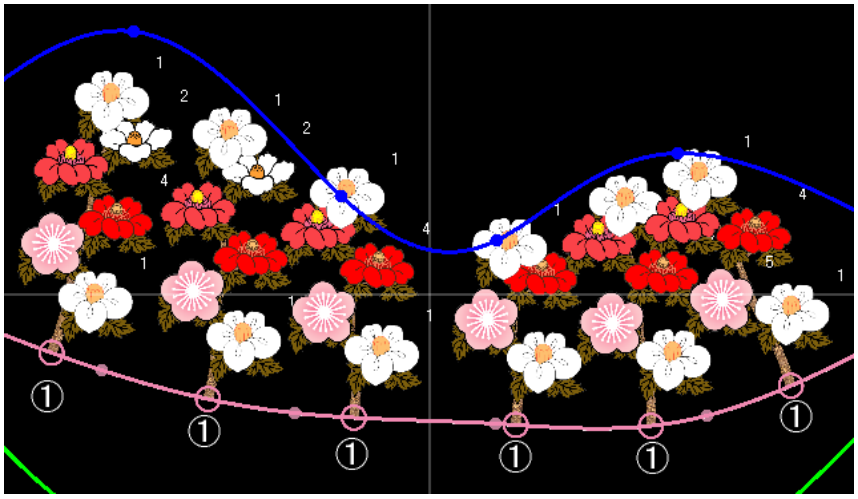
図 4.28-[d]では，葉の画像の割り当てを指定する．割り当ての方法は，4.6.1 項で述べた花模様のタイプに関連付けられている．すなわち，葉 1 の画像の配置は，花の模様のツールボックスで選択したタイプに依存する．以下に，タイプ 1 の場合と，タイプ 2 の場合のときの葉 1 の画像の割り当てについて述べる．

・葉 1 －　タイプ 1

タイプ 1 では，葉 1 の画像は，①の部分のみしか使用しない．樹木への葉の割り当ては，図 4.32(b)に示すように，すべて同一の①の画像を表示する．葉 1 の画像は，右下のボタンを押して画像ファイルを選択して指定する．



(a)画像リスト

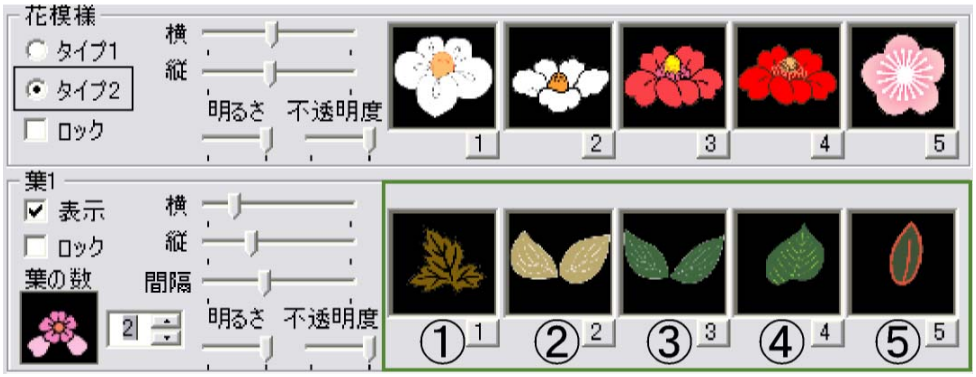


(b)結果画像

図 4.32　葉 1-タイプ 1

・葉 1 -　タイプ 2

タイプ 2 では，図 4.33(a)で示すように，葉 1 の画像は，①～⑤のすべてを使用する．樹木へは，図 4.33(b)に示すように，木の数によって順に割り当てられる．樹木の数が，6 本目以降は 5 の剰余+1 にあたる葉 1 の画像を割り当てる．葉の画像の選択法は同様の操作を行う．



(a)画像リスト

(b)結果画像

図 4.33　葉 1-タイプ 2

4.8　ツールボックス 6

図 4.34 ツールボックス 6

4.8.1　葉 2－表示・ロック

図 4.34-[a] では，葉 2 の表示や移動を制御する．

・　「表示」: チェックすることで，葉 2 を表示する．

・　「ロック」: チェックすることで，葉 2 を固定する．

4.8.2　葉 2－パラメータ設定

図 4.34-[b]では，葉 2 の画像の横および縦の大きさを設定する．

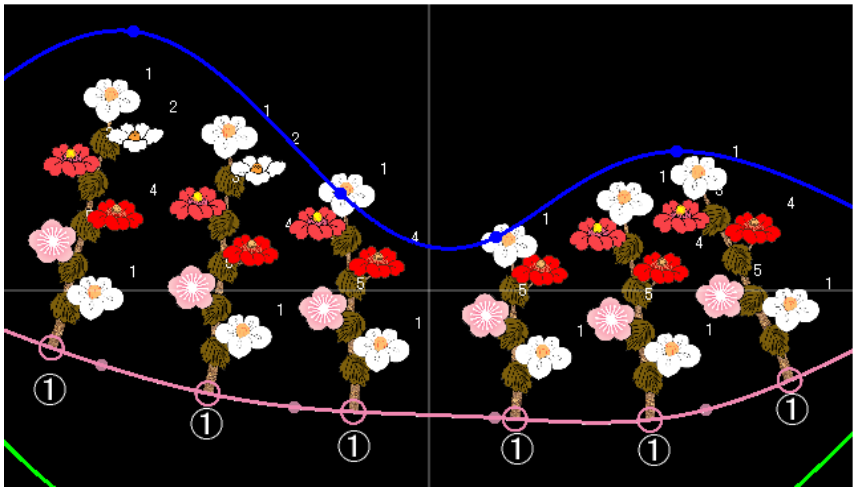
4.8.3　葉 2－画像

図 4.34-[c]では，葉 2 の画像の割り当てを指定する．割り当ての方法は，4.6.1 で述べた花模様のタイプに関連付けられている．すなわち，葉 2 の画像の配置は，花の模様のツールボックスで選択したタイプに依存する．

・　葉 2 －　タイプ 1

タイプ 1 では，葉 1 の場合と同様に，図 4.35(a)で示した①の画像しか使用しない．樹木への葉画像の割り当ては，図 4.35(b)に示すように，樹木の数に依存せず，すべて同一の①の画像を使用する．葉の画像の選択法は同様の操作を行う．

(a)画像リスト



(b)結果画像

図 4.35　葉 2-タイプ 1

・　葉 2 －　タイプ 2

タイプ 2 では，葉 1 の場合と同様に，葉の画像には，図 4.36(a)に示したの①～⑤のすべての画像を使用する．樹木への画像の割り当ては，図 4.36(b)に示すように，木の数によって順に割り当てる．樹木の数が 6 本目以降は 5 の剰余+1 にあたる画像を割り当てる．葉の画像の選択法は同様の操作を行う．

(a)画像リスト

(b)結果画像

図 4.36 葉 2-タイプ 2

4.9 ツールボックス 7

ここからは，背景模様の生成についての操作である．

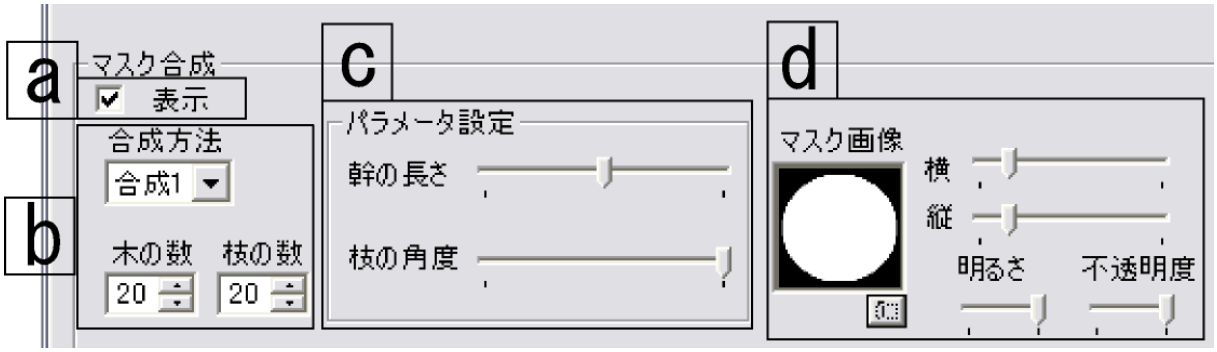


図 4.37 ツールボックス 7

4.9.1 マスク画像－表示

図 4.37-[a]では，マスク画像の表示を制御する．

・ 「表示」：チェックすることで，マスク画像を表示する．

4.9.2　マスク生成のための樹木生成

図 4.37-[b]では，マスク画像の生成規則に対するパラメータを設定する．マスク画像生成のための木の種類と制約条件から,「合成 1～4」の 4 通りの生成方法を定義する．以下，レイヤー背景の中心線を図 4.38 に示すように設定し，図 4.37-[b]にある「木の数」を 10 個と設定して説明する..

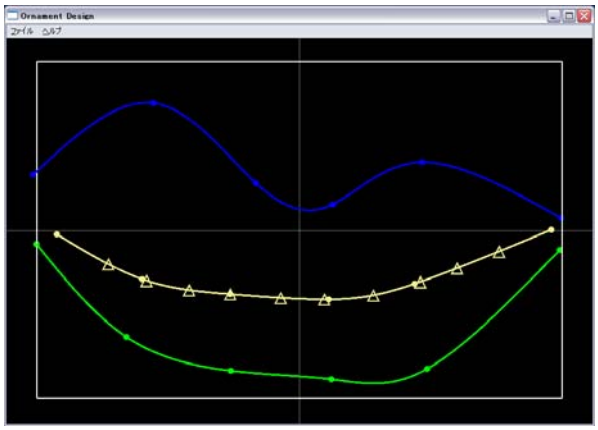


図 4.38　背景レイヤーの初期設定

・ 合成 1

図 4.39 に示すような単純木を生成し，各頂点に図 4.37-[d]で指定したマスク画像を描画する．ここで，生成する単純木は，上・下領域線の制約を受ける．そして，ツールボックス 8 で生成する模様を割り当てて，図 4.39 に示す合成結果を得る．

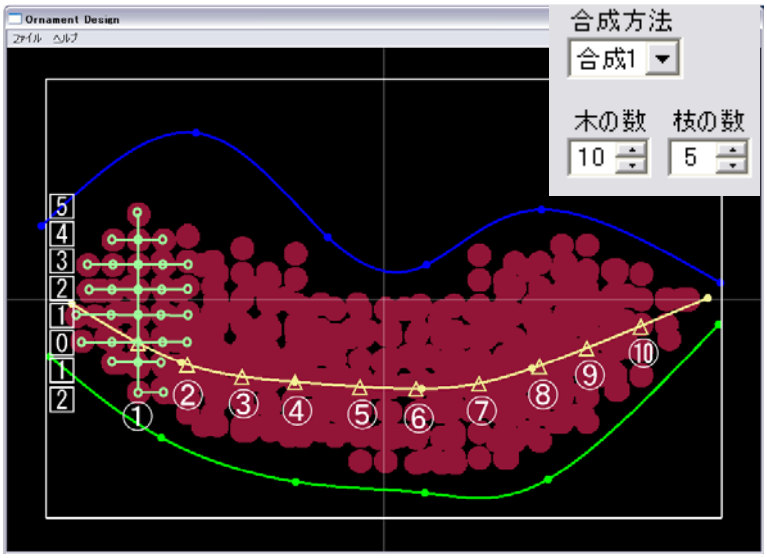


図 4.39　合成 1

図 4.39 では，木の数を 10 に，枝の数を 5 に設定しているが，①の木では，レイヤーマークから下は下領域線によって制約を受け，2 段階しか伸びていない.

・ 合成 2

合成 1 と同様の単純木を生成するが，樹木の生成に図 4.40 内の赤の円で示した制約を設ける.

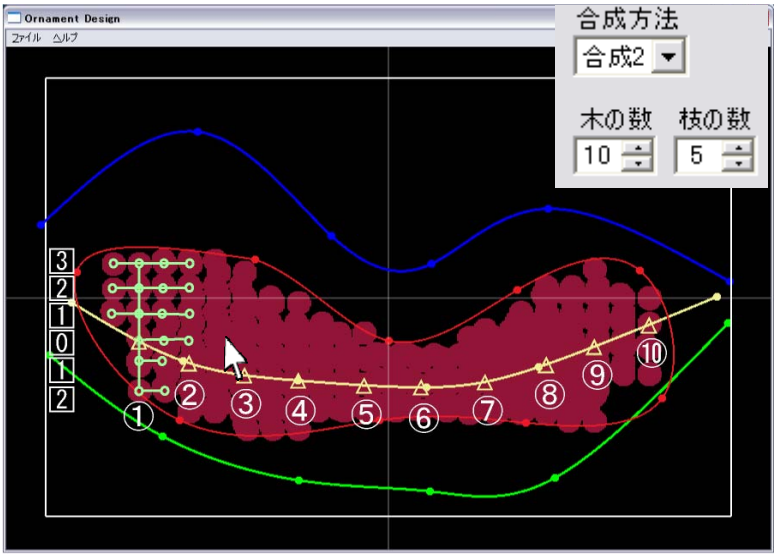


図 4.40 合成 2

. この制約のために，図の①に示した樹木の枝の数が，上方向に 3 段階，下方向に 2 段階しか伸びていない.

・ 合成 3

合成 3 は，合成 1, 2 とは枝の付き方が違う樹木を生成し，合成 1 と同じように合成する. 図 4.41 に示すように. 合成 1 に比べ, 隙間が目立つマスク画像が生成されている。また，合成 1 と同様に上領域，下領域線の制約を受けている.

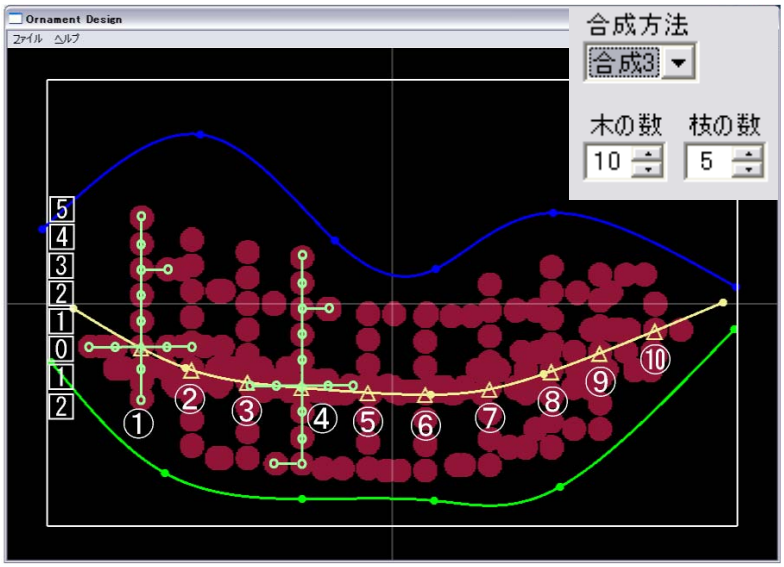


図 4.41　合成 3

・ 合成 4

合成 4 では，合成 3 と同様の単純木を使い，図 4.42 内の赤の円で囲まれた部分にのみマスク画像を生成する制約を設ける..

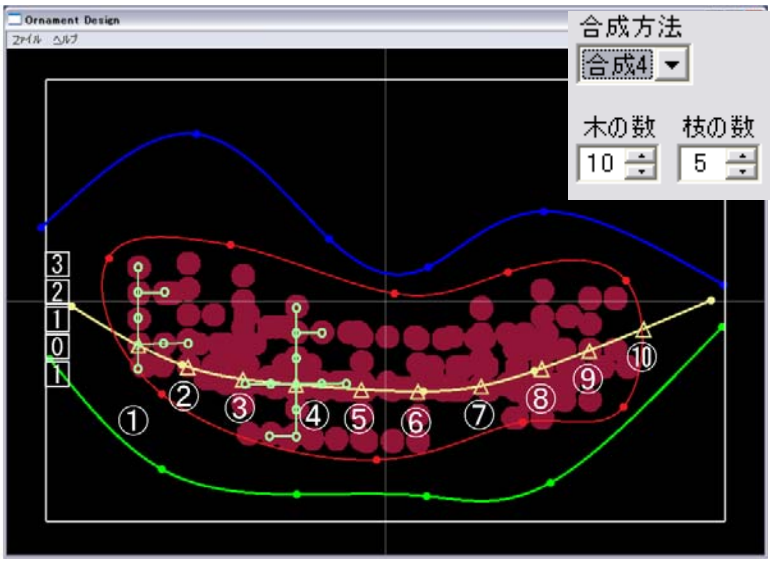


図 4.42　合成 4

4.9.3　マスク画像ための樹木のパラメータ設定

図 4.37-[c]では，マスク画像生成のための樹木の長さと枝の角度を指定する．合成 1 の場合を例に挙げて述べる．図 4.44(a)に示す合成結果に対して，図 4.43 の「幹の長

さ」を短くすると，図 4.44(b)のようになる．また，「枝の角度」を 0 度にすると，図 4.44(c)のようになる．

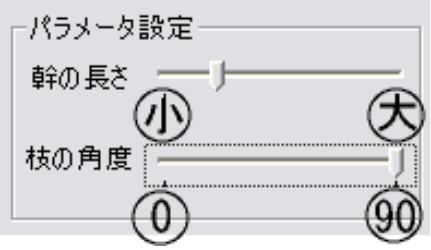
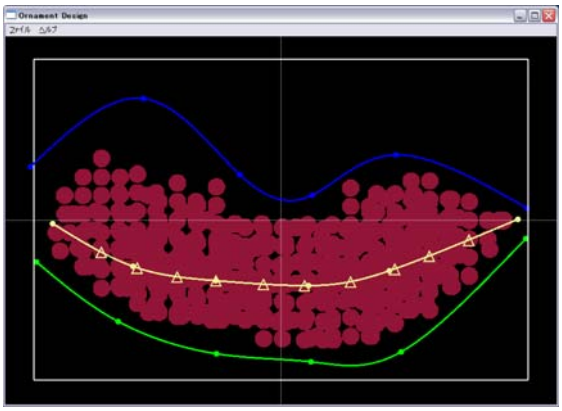
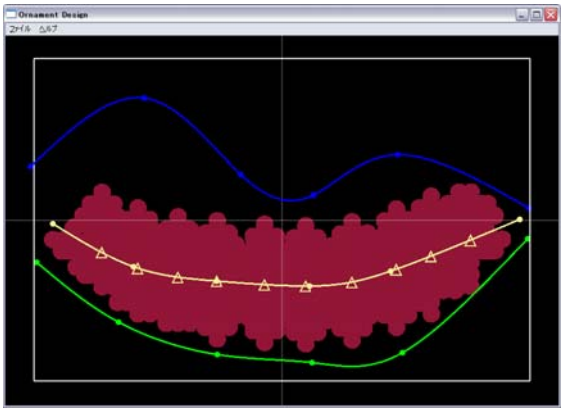


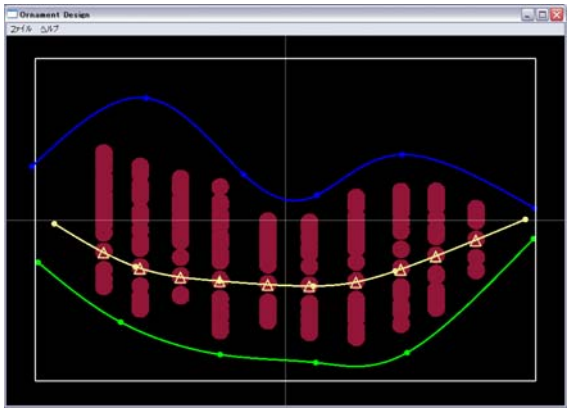
図 4.43　パラメータ設定



(a)変更前



(a)幹の長さを短く変更



(c)枝の角度を 0 度に変更

図 4.44　パラメータ変更の例

4.9.4　　マスク画像

図 4.37-[d]では，マスク画像の選択と，マスク画像の縦・横の長さや，明るさおよび，

透明度を設定する．マスク画像は，画像の右下のボタンを押し，画像ファイルを選択して指定する．図 4.46(a)の合成結果に対して，図 4.45 に示すマスク画像に変更した結果を図 4.46(b)に示す．

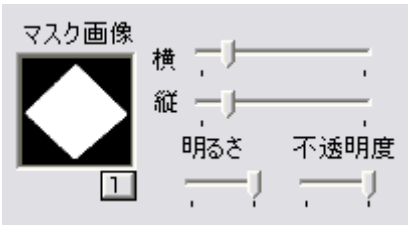
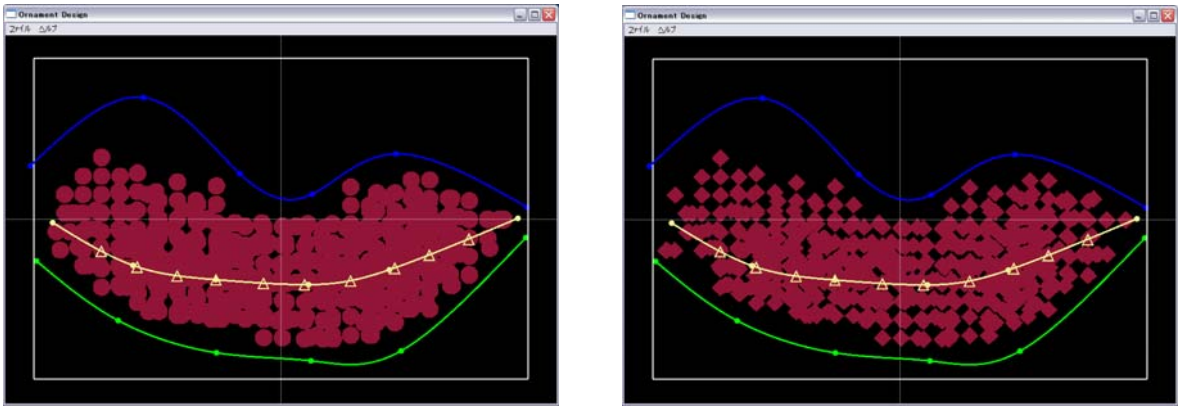


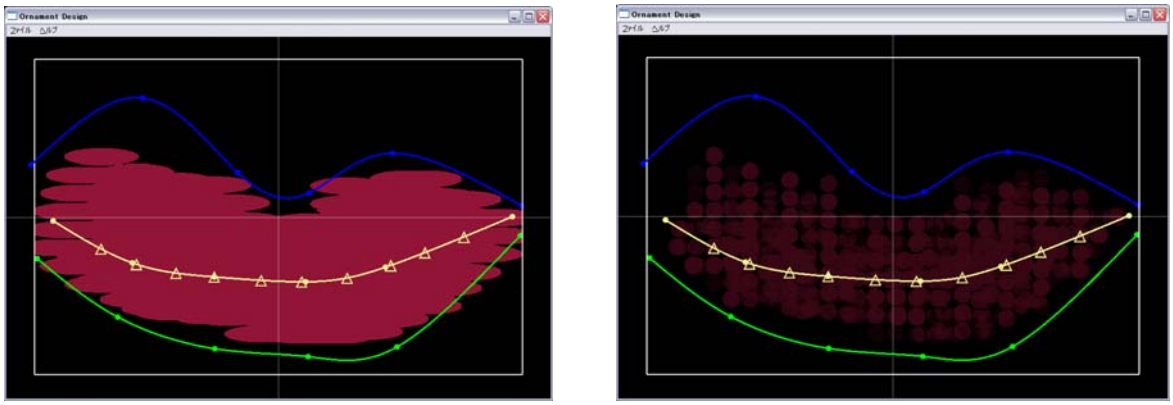
図 4.45　マスク画像



(a)変更前　　(b)変更後

図 4.46　マスク画像の変更

次に，図 4.46(a)の合成結果に対して，図 4.47(a)はマスク画像の横幅を大きくした場合，図 4.47(b)は不透明度の値を下げた場合を示す．



(a)横サイズ変更　　(b)不透明度変更

図 4.47　パラメータの変更例

4.10 ツールボックス 8

ここで述べる模様 1～5 は，生成したマスク画像に合成する模様画像である．

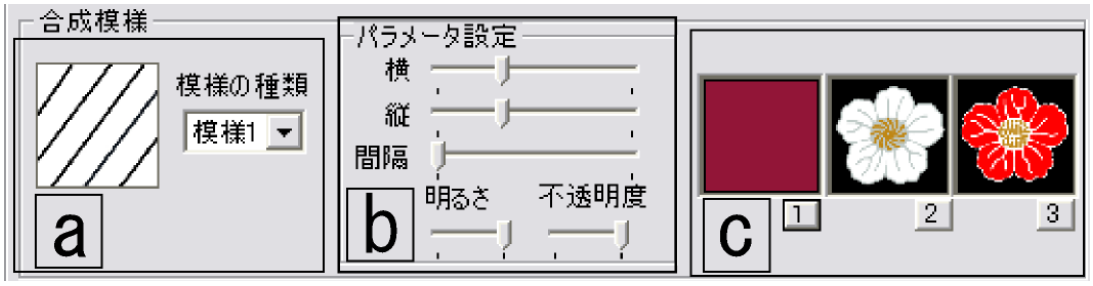


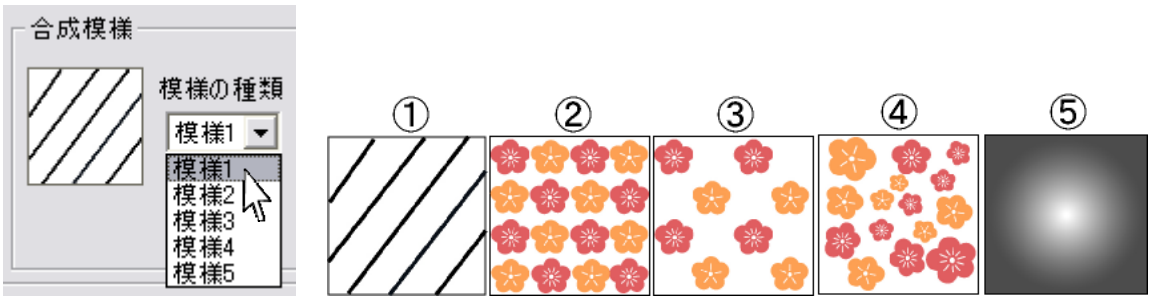
図 4.48　ツールボックス 8

4.10.1　模様の種類

図 4.48-[a]では，マスク合成に使用する模様の種類を設定する．模様には，以下の 5 種類がある．

模様①：　ベタ塗り
模様②：　並べて配列
模様③：　間隔を空けて配列
模様④：　ランダムに配列
模様⑤：　特殊なベタ塗り

模様は，図 4.49(a)に示すようにプルダウンメニューで指定する．この際，図 4.49(a)の左上の画像は，4.49(b)に示した画像に置き換わる．



(a)選択の様子　　(b)模様の種類

図 4.49　模様の選択

4.10.2　合成画像

図 4.48-[c]では，マスク画像に対して合成する画像を選択する．ここで，マスク合成に使用する画像は，模様の種類によって異なる．図 4.50 に示すように，模様①と⑤に対しては，一番左の画像のみを使用し，模様②③④に対しては，すべての画像を使用する．このように，画像の使用数は模様の種類に依存する．以降に模様 1～5 について述べる．

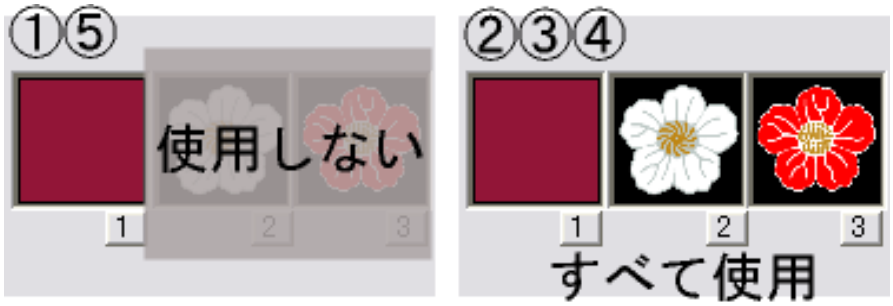


図 4.50　画像の使用数

・　**模様 1**

模様 1 では，生成したマスク画像に対して，図 4.50 に示す最初の画像のみを合成する．図 4.51 に合成結果を示す．

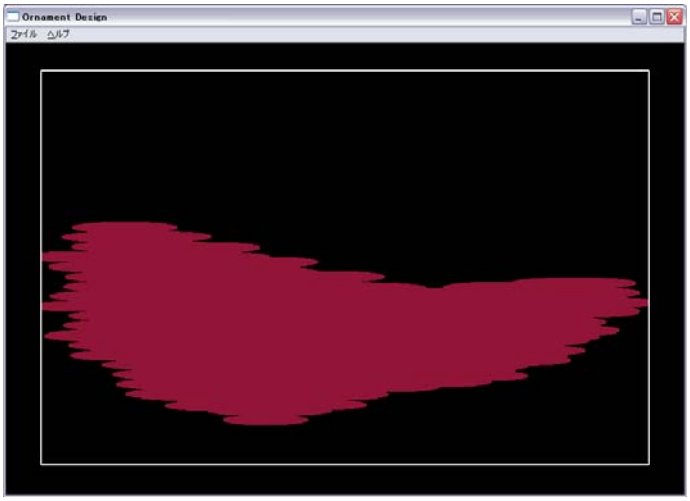


図 4.51　模様 1 の合成結果

・ **模様 2**

模様 2 では，生成したマスク画像に対して，図 4.50 に示す 1 の画像を背景に，残りの 2 つの画像を連続的に並べて配列する．図 4.52 に合成結果を示す

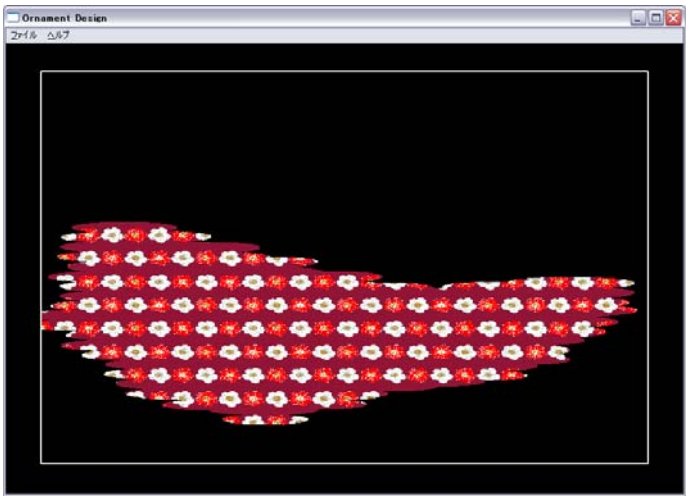


図 4.52 模様 2 の合成結果

・ **模様 3**

模様 3 は，模様 2 と基本的に同じであるが，隣り合う左右の模様を同じ模様に配置する．図 4.53 に合成結果を示す

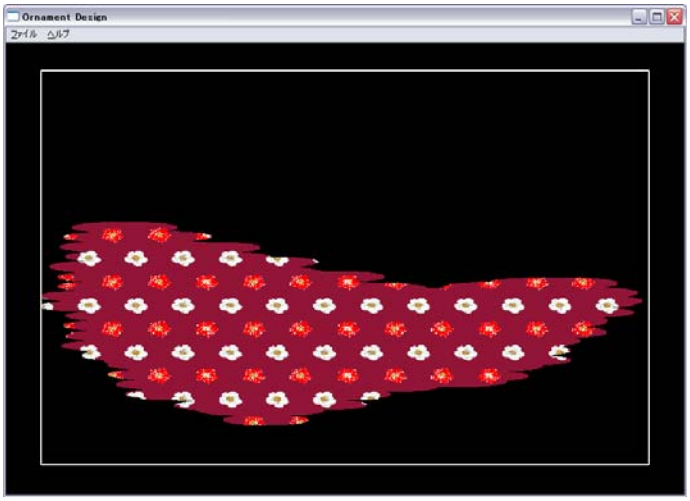


図 4.53 模様 3 の合成結果

・ **模様 4**

模様 4 では，生成したマスク画像に対して，図 4.50 に示す 1 の画像を背景に，残りの 2 つの画像の位置・大きさ・角度をランダムに設定して配置して合成する．図 4.54 に合成結果を示す

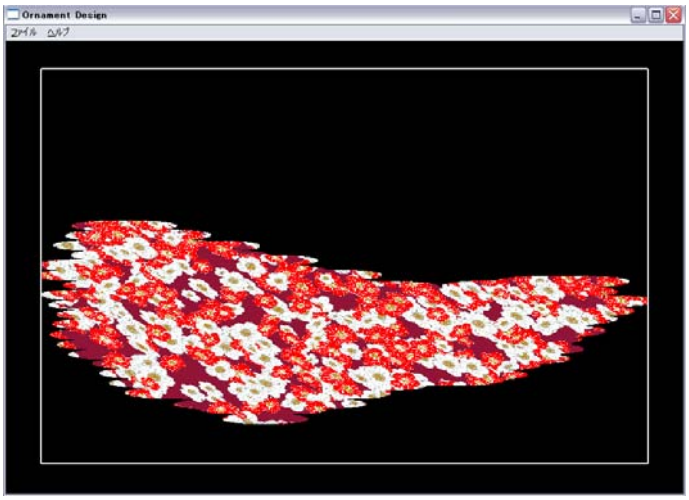
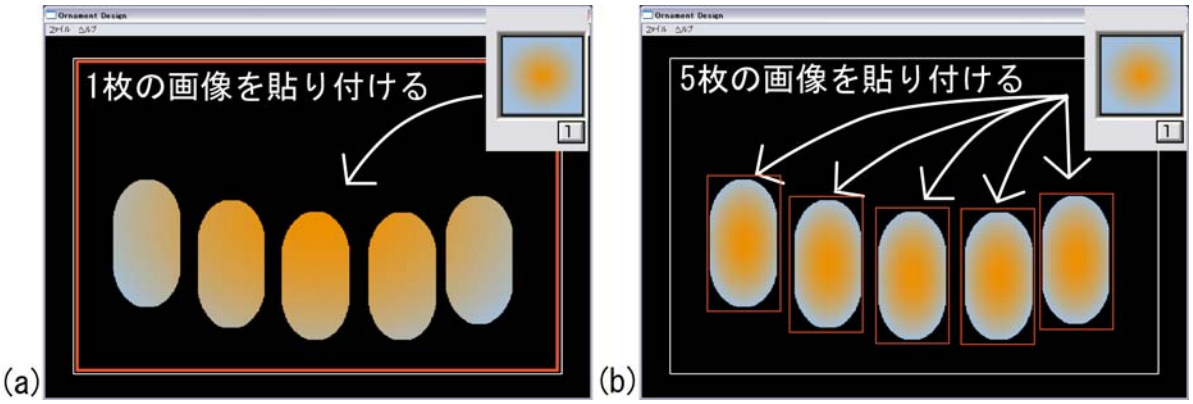


図 4.54 模様 4 の合成結果

・ **模様 5**

模様 5 では，生成したマスク画像に対して，図 4.50 に示す一番左の画像のみを合成する．模様 5 の特徴は，画像の貼り付け方にある．合成 1 では，図 4.55(a)に示すように，生成されたマスク画像に対して，画像を赤枠の範囲内に貼り付ける．それに対して合成 5 では，図 4.55(b)に示すように，生成されたマスク画像に対して、5 つの赤枠の範囲ごとに貼り付ける．



(a)模様 1　　(b)模様 5

図 4.55　模様 1 と模様 5 の比較

4.10.3　模様のパラメータ設定

図 4.48-[b]は，模様に使用する 3 つの画像の横・縦・間隔・明るさ・不透明度の各パラメータを設定する．各パラメータには，模様の種類によって，設定の可否の制約がある．

模様 1 :「明るさ・不透明度」のパラメータ設定が可能
模様 2,3,4 :「縦・横・間隔・明るさ・不透明度」のパラメータ設定が可能
模様 5:　「横・縦・明るさ・不透明度」のパラメータ設定が可能

図 4.56 は，それぞれのパラメータの値の設定を示している．ここで，図 4.57(a)の模様に対して，間隔のパラメータを広くした場合の合成結果を図 4.57(b)に示す．

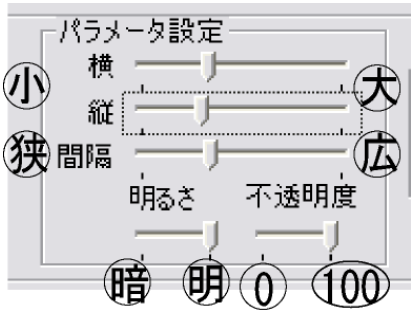
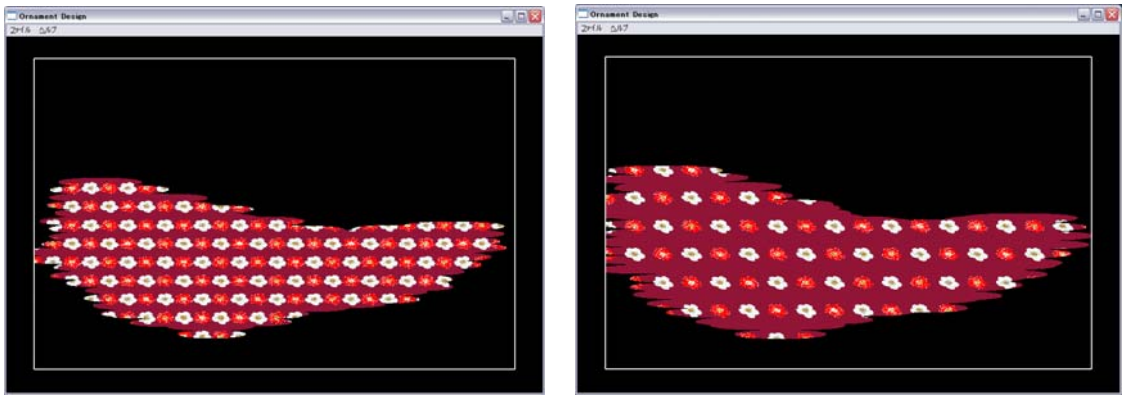


図 4.56　パラメータ設定



(a)変更前　　(b)変更後

図 4.57　パラメータ変更の例

4.11 ツールボックス 9

ツールボックス 9 は，背景画像を指定する．

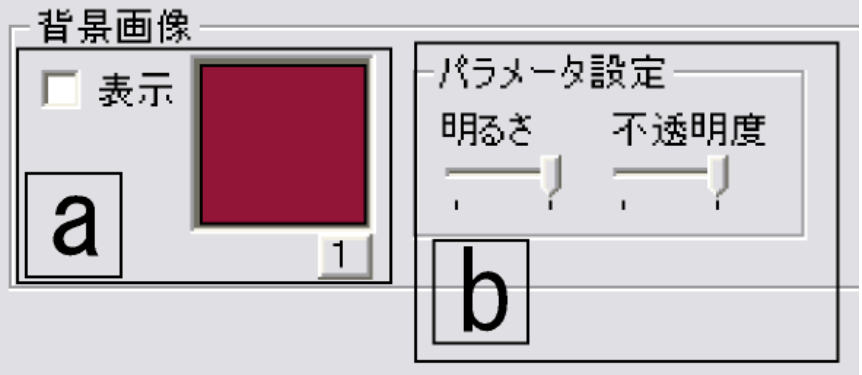


図 4.58　ツールボックス 9

4.11.1　背景画像の表示

図 4.58-[a]では，背景画像の表示を指定する．

・　「表示」：チェックすることで，背景画像を表示する．

また，画像の右下のボタンを押すことで，背景画像に用いるファイルを選択する．図 4.58-[a]で指定した画像を背景画像とした結果を図 4.59 に示す．

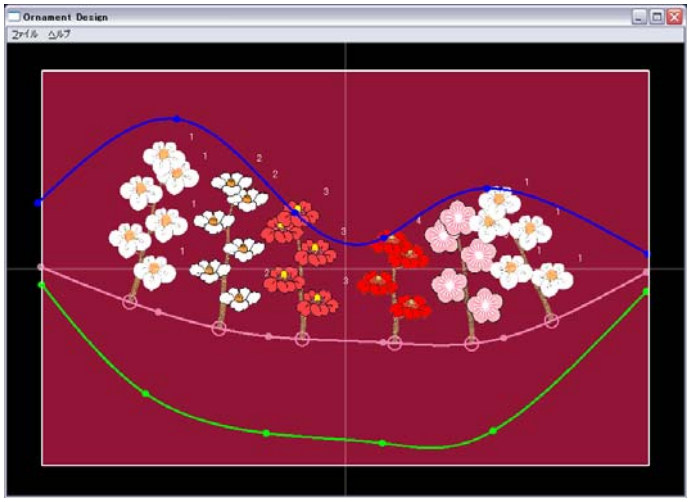


図 4.59　背景画像の表示例

4.11.2　パラメータ設定

図 4.58-[b]は，これまでと同様に，明るさ・不透明度のパラメータを指定する．図 4.60 は図 4.59 の明るさを暗くしたものである．

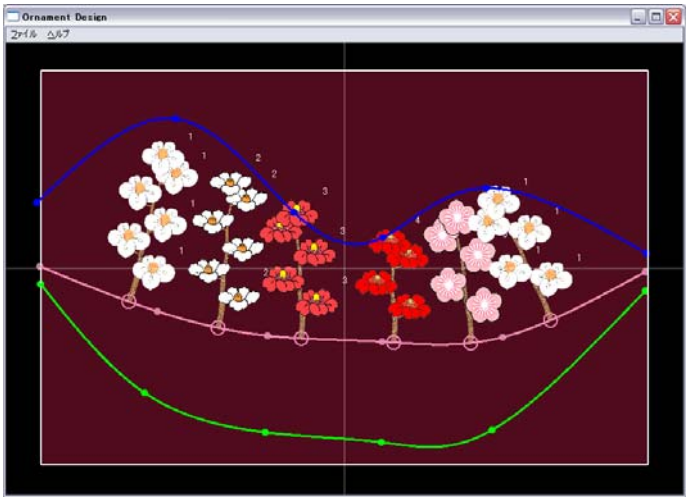


図 4.60　背景画像の明るさの変更例

4.12 データ管理機能

本節では，模様生成後に必要となるデータ管理について述べる．スクリーン画面の左上にある「ファイル」を選択すると，図 4.61 に示すような a～e の 5 つの項目を表示する．以下に各項目について述べる．

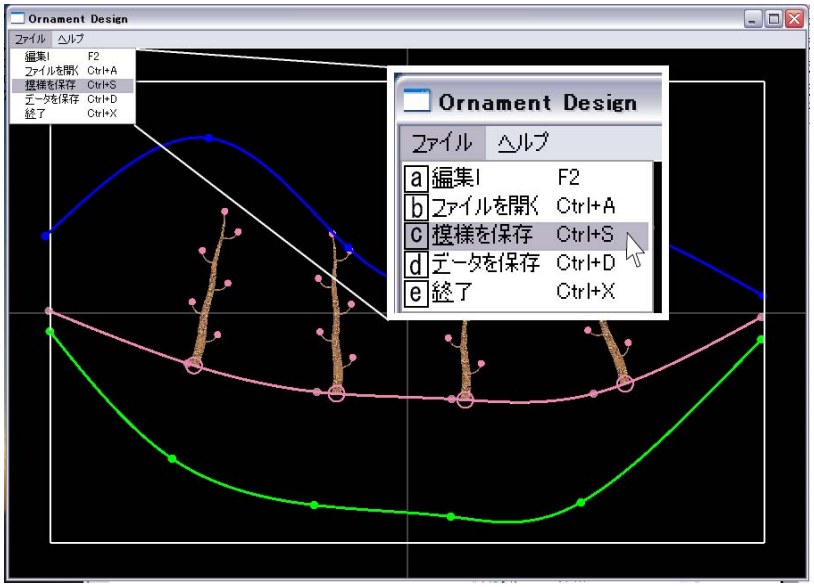


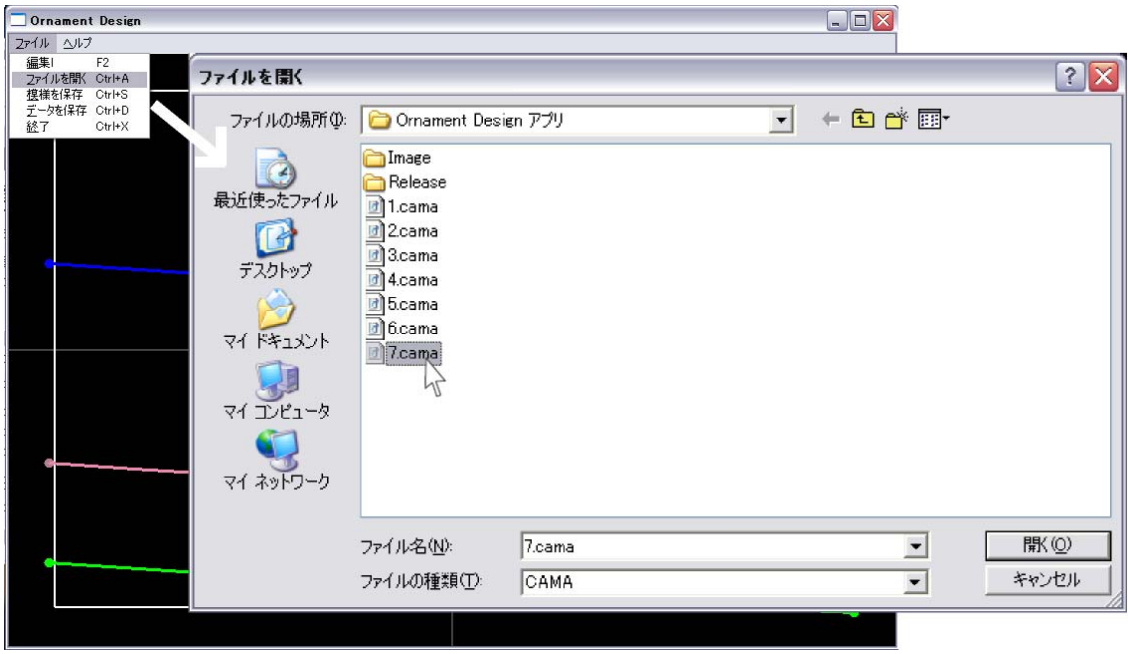
図 4.61　データ管理機能の画面

4.12.1　編集

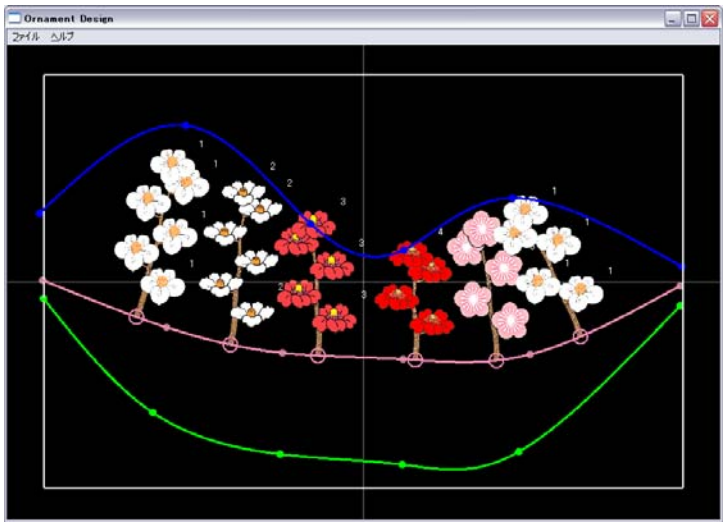
図 4.61-[a]の「編集」を選択することで，図 4.2 で示したツールボックスが表示される．本システムでは，ツールボックスは自動的に表示されるように初期設定しているが，ツールボックスを閉じることも出来るため，閉じツールボックスを再表示させるための項目である．

4.12.2　ファイルを開く

図 4.61-[b]では，保存した模様のデータを呼び出す．図 4.62(a)は，初期画面において「ファイルを開く」を選択した場合を示す．この例では「7.cama」ファイルを選択し，図 4.62(b)に示す模様を表示する．拡張子「.cama」のファイルは，模様のデータを保存するためファイルである．このファイルフォーマットを次項で述べる．



(a)ファイル選択



(b)表示結果

図 4.62 ファイルを開く

4.12.3　ファイルのフォーマット

拡張子「.cama」のファイルは，以下に示す模様生成のデータを保存する．

「ｃama ファイルフォーマット」

- 木模様の根元の頂点データ
- 木模様の枝・幹・角度などのデータ
- 花模様・葉 1・葉 2 のファイル参照先データ
- 花模様・葉 1・葉 2 の大きさなどのデータ
- 上領域線・中心線・下領域線の頂点データ
- 背景模様のデータ

が保存されている．

この外部ファイルにより，生成した模様をいつでも呼び出すことができる．

4.12.4　模様を保存

図 4.61-[c]では，生成した模様を画像データとして保存する．スクリーン内に表示されている領域を「bmp」「png」「jpg」の 3 つの拡張子で保存することができる．

4.12.5　データ保存

図 4.61-[d]では，作業中の模様データを c ama ファイルとして保存する．

4.12.6　終了

図 4.61-[e]では，ツールを終了する．

4.13 配置調整

ユーザは，花の模様・葉 1 ・葉 2 の模様を個別に移動・回転・拡大・縮小することで，模様の最終調整をする．

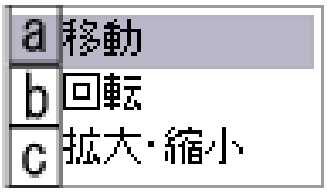


図 4.63　配置調整のポップアップメニュー

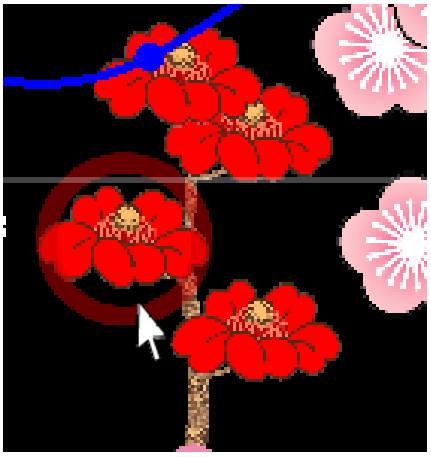
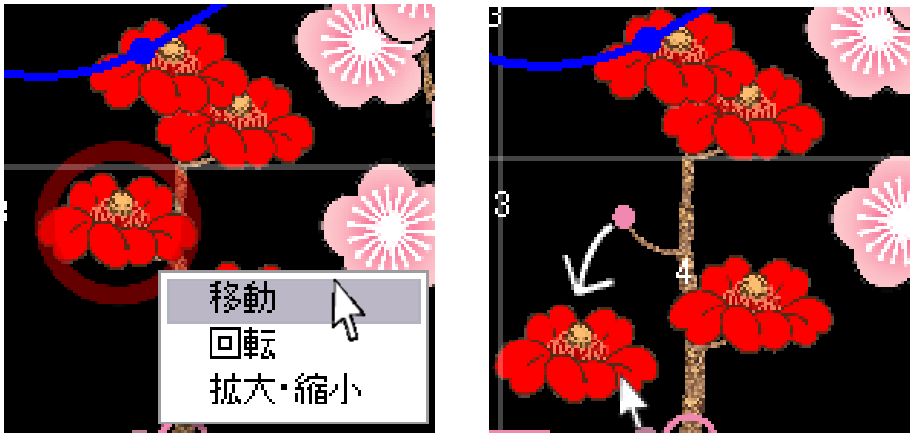


図 4.64　花模様の選択の様子

編集作業における共通操作は，図 4.64 に示すような，マウスクリックによる編集対象の選択である．クリックされた模様の周りに赤円を表示し，さらに右クリックをすると，図 4.63 に示す 3 項目をポップアップ表示する．以下にその 3 項目について説明する．

4.13.1　移動

図 4.65 に示すように，表示された項目の「移動」を選択した後，マウスのドラッグ操作で花模様を移動する．

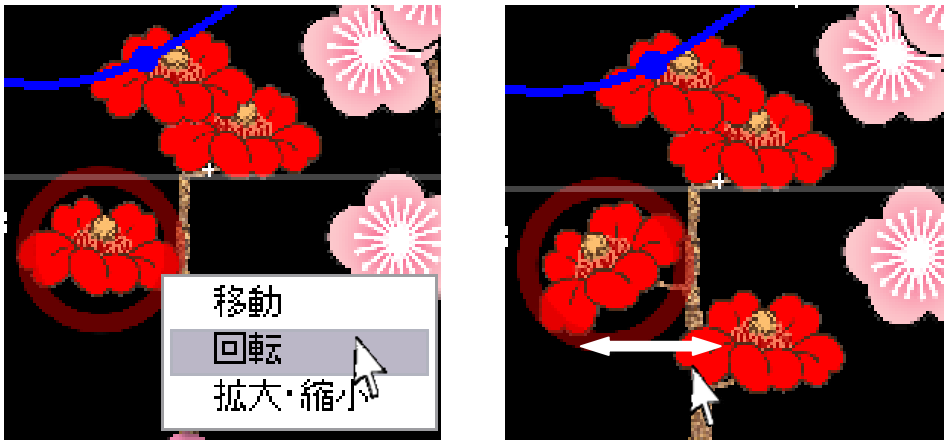


(a)移動を選択　　(b)移動操作

図 4.65　移動操作の様子

4.13.2　回転

図 4.66 に示すように，表示された項目の「回転」を選択した後，マウスのドラッグ操作により，花の模様を回転する．

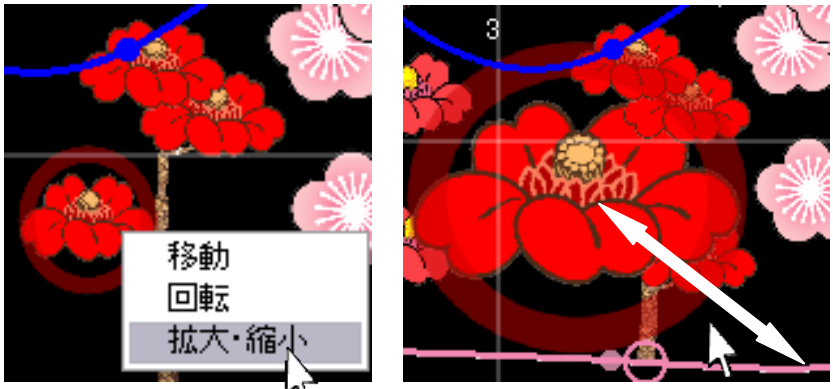


(a)回転を選択　　(b)回転操作

図 4.66　回転操作の様子

4.13.3　拡大・縮小

図 4.67 に示すように，表示された項目の「拡大・縮小」を選択した後，マウスを花模様の中心に近づけて花の模様を縮小，遠ざけて拡大する．



(a)拡大・縮小を選択　　(b)拡大操作

図 4.67　拡大・縮小操作の様子

第　5　章

生成プロセス

本章では，構築したシステムを用いて模様をデザインするプロセスについて述べる．

5.1　　生成プロセスの概要

模様生成は，図 5.1 に示す 4 つのプロセスを経る．完成した模様は，最終的にイメージ画像として保存するか，模様の配置をデータとして保存する．

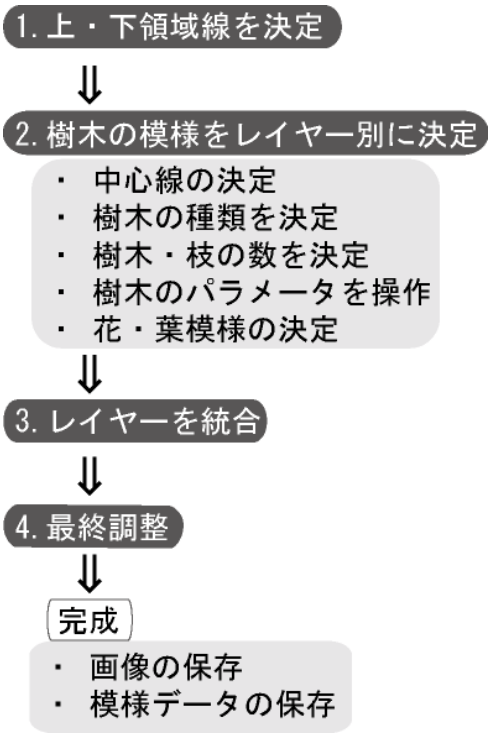


図 5.1　生成プロセスのフロー

以降の節では，各プロセスについて詳しく述べる．

5.2　上・下領域線の決定

最初のプロセスでは，図 5.2 に示すように，①の上領域線，②の下領域線を配置する．

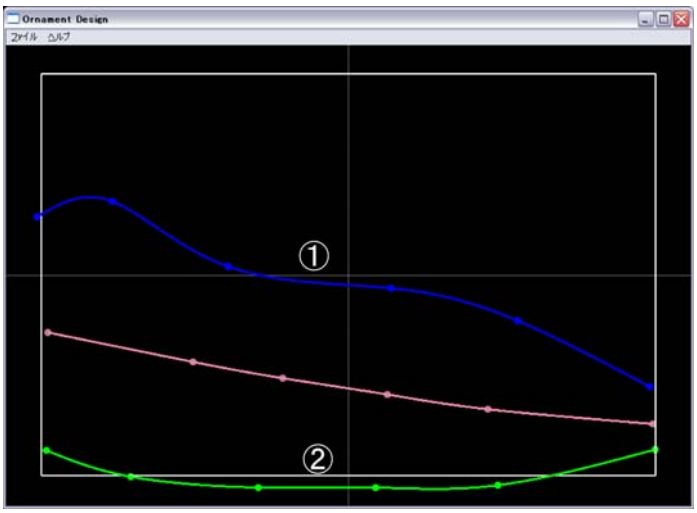


図 5.2　上・下領域線の決定

5.3　樹木の模様をレイヤー別に決定

本節で述べるデザインプロセスは，レイヤーごとに行なう．

5.3.1　中心線の決定

図 5.3 に示すようにレイヤーの中心線を決定する．

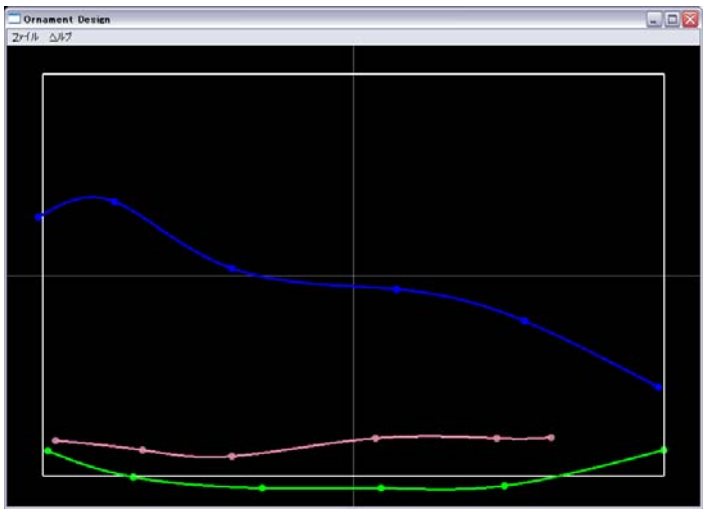


図 5.3　中心線の決定

5.3.2　　樹木の模様の設定

図 5.4 に示すように，①で樹木の模様の「種類」を設定し（この例では Tree6 を選択），②で「木の数」を指定する（この例では 4 に設定）．指定後のスクリーンを図 5.5 に示す．

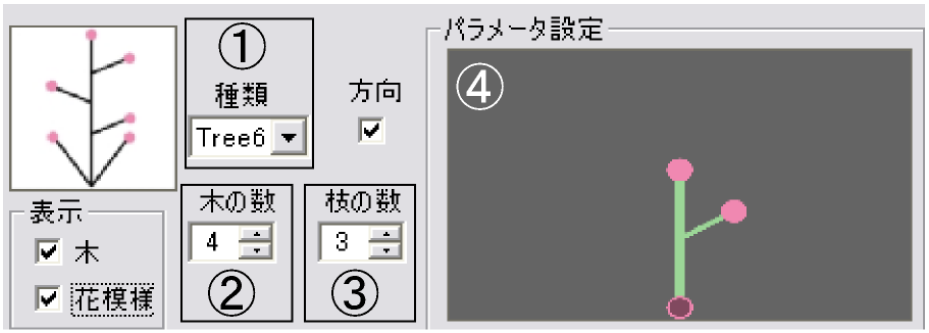


図 5.4 操作画面

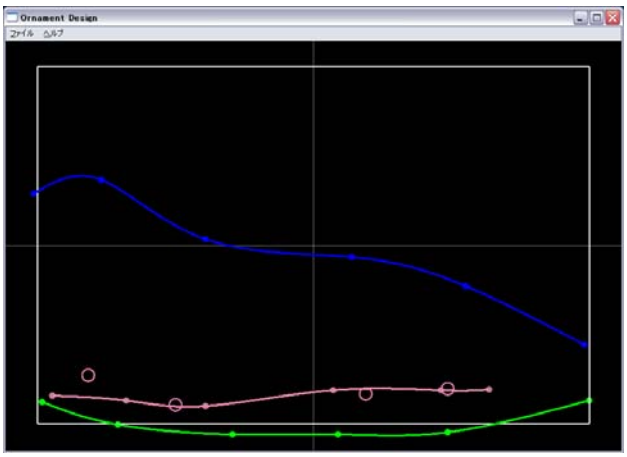
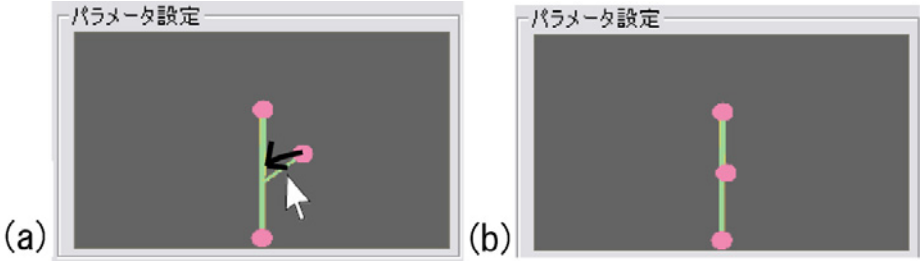


図 5.5　樹木の数の決定

樹木の数の指定後，図 5.4③で「枝の数」を指定する（この例では 3 に設定）．その後，図 5.4④において，図 5.6(a)に示すように，木の枝の角度や長さなどを調整する．指定後のスクリーンを図 5.7 に示す．



(a)設定前　　(b)設定後

図 5.6　パラメータ設定

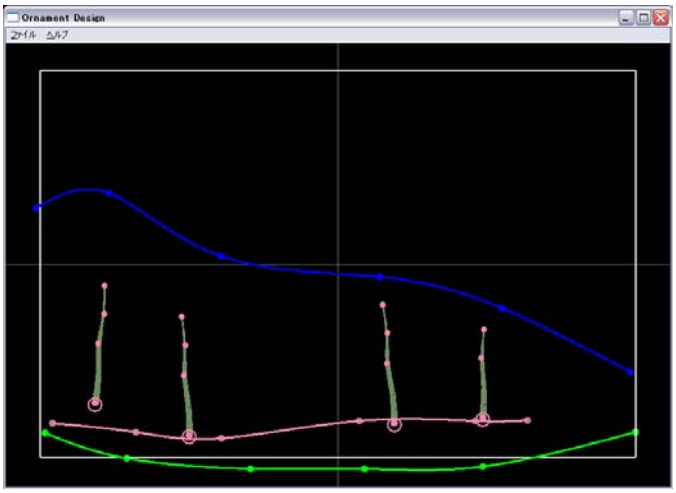


図 5.7　パラメータ調整後の画面

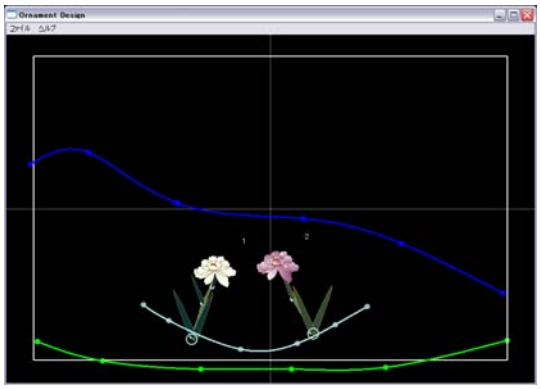
5.3.3　　花・葉模様の決定

樹木の模様の設定後，花や葉画像を指定する．ここでは，花の画像の設定以外に，花や葉の大きさなどの調整も必要となる．

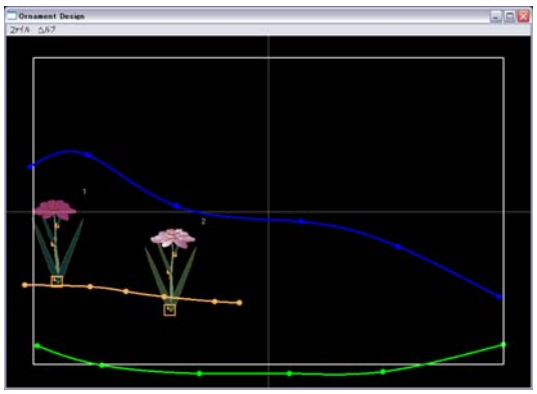
図 5.8　花・葉模様の決定

5.4　　レイヤーを統合

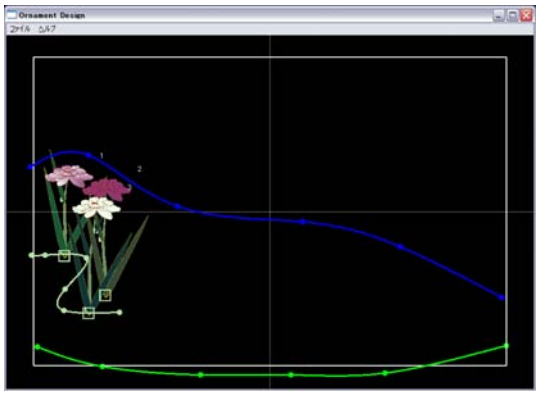
5.3 節の作業を各レイヤーに対して行なう．図 5.9(a)～(e)に各レイヤーの結果画像を示す．最終的に，図 5.10 に示すように完成した 6 つのレイヤーを同時に表示させ統合する．



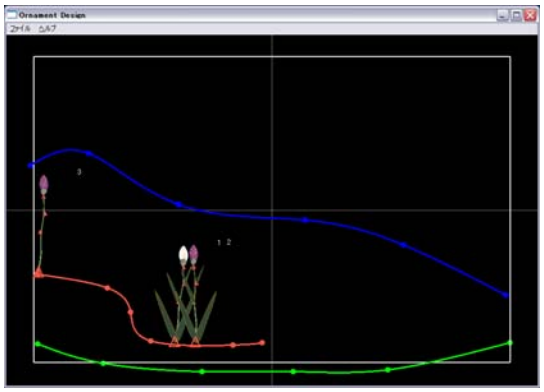
(a) レイヤー2



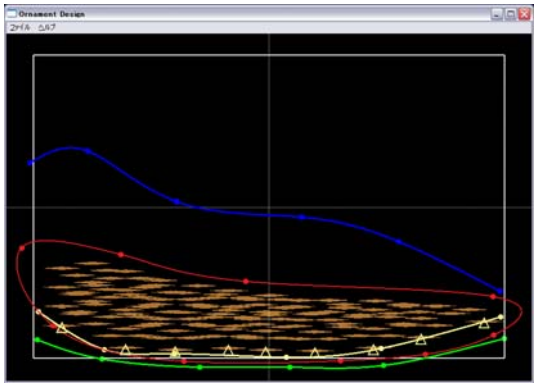
(b) レイヤー3



(c) レイヤー4



(d) レイヤー5



(e) レイヤー背景

図 5.9　レイヤーごとの画像

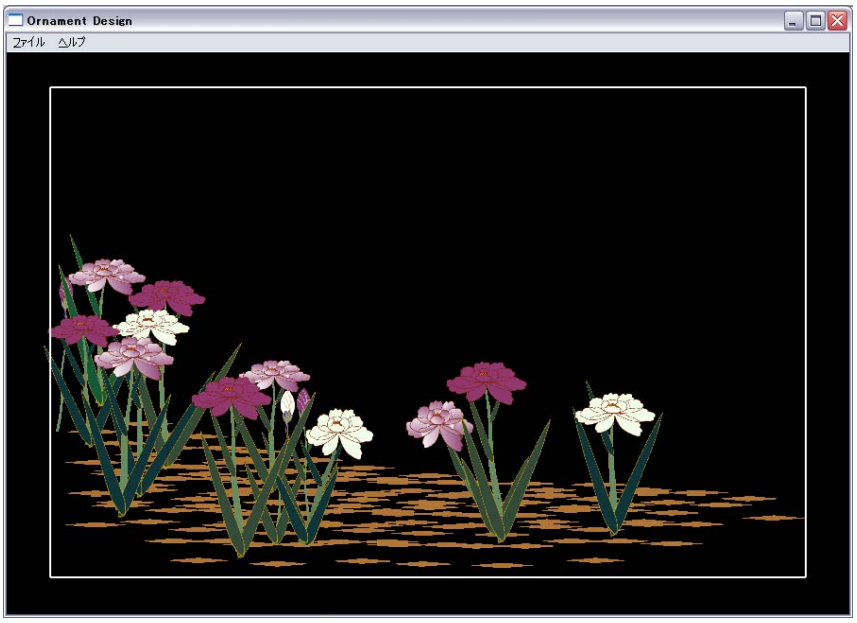
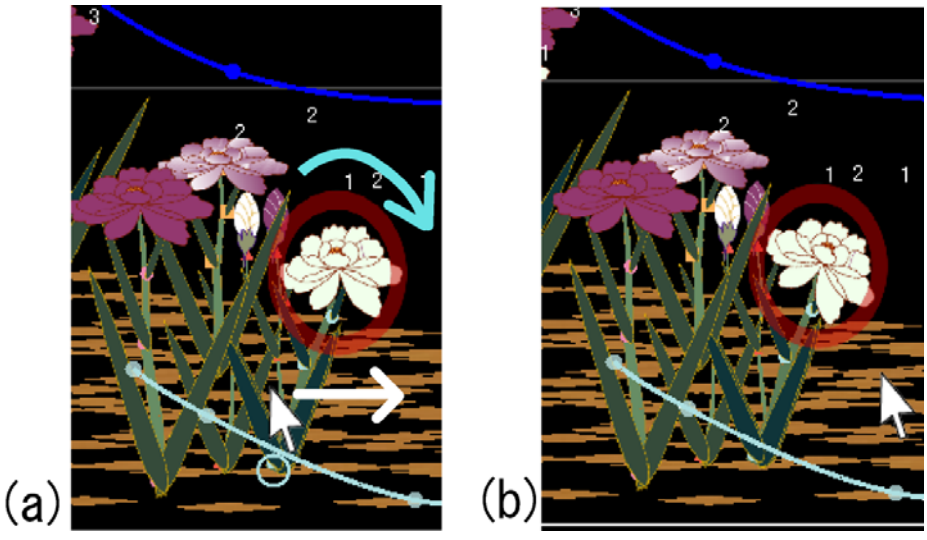


図 5.10　統合画像

5.5　最終調整

最終調整では，花・葉模様などの配置・角度・大きさなどを個別に調整する．図 5.11 では，円で囲まれた花模様を回転して調整している．



(a) 調整前　　(b) 調整後

図 5.11　最終調整の様子

5.6　画像を保存

完成したスクリーン上の模様は，図 5.12 に示すように画像として保存できる．また，同時に模様の配置データを保存し，いつでも編集できる．

図 5.12　保存画像

第　6　章

結果

5 章で述べた生成プロセスを経て，以下に示す 8 つの模様を生成した．6.1 に結果模様を，6.2 では，それぞれの模様について説明する．

6.1　結果模様

図 6.1　結果模様 1

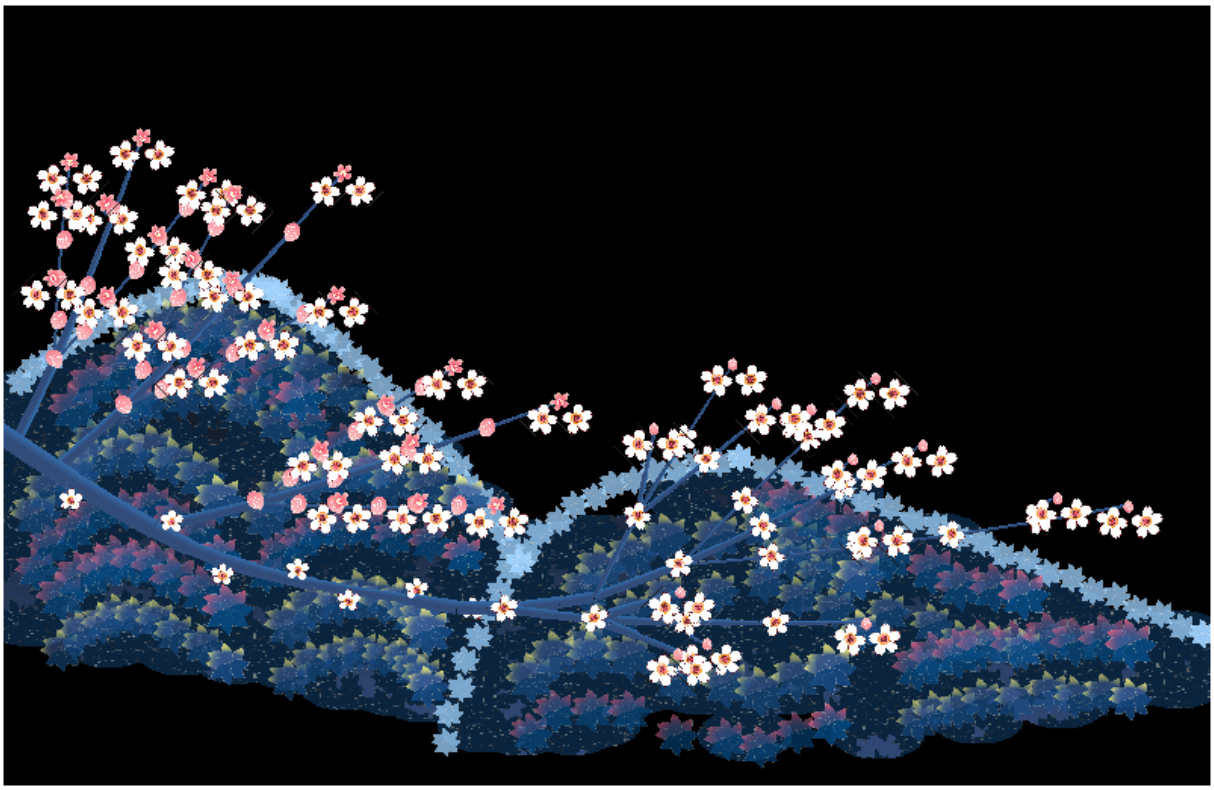


図 6.2　結果模様 2

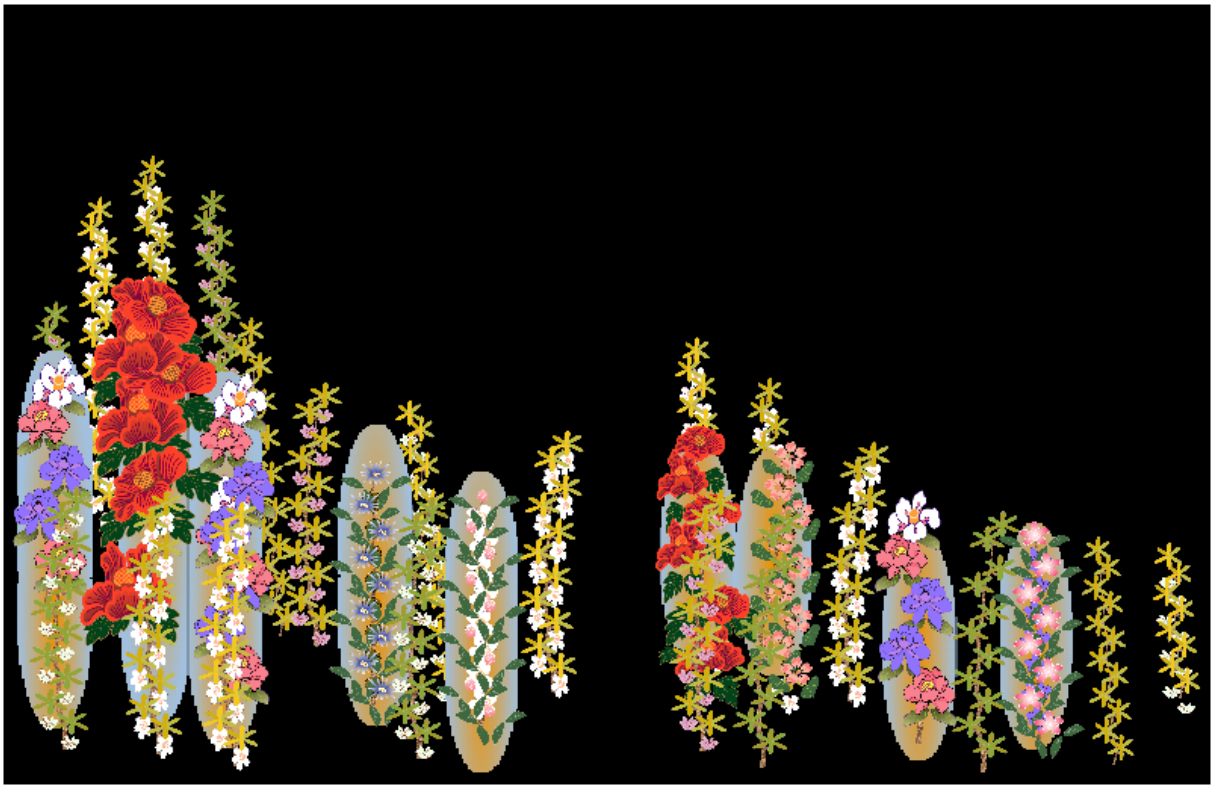


図 6.3　結果模様 3

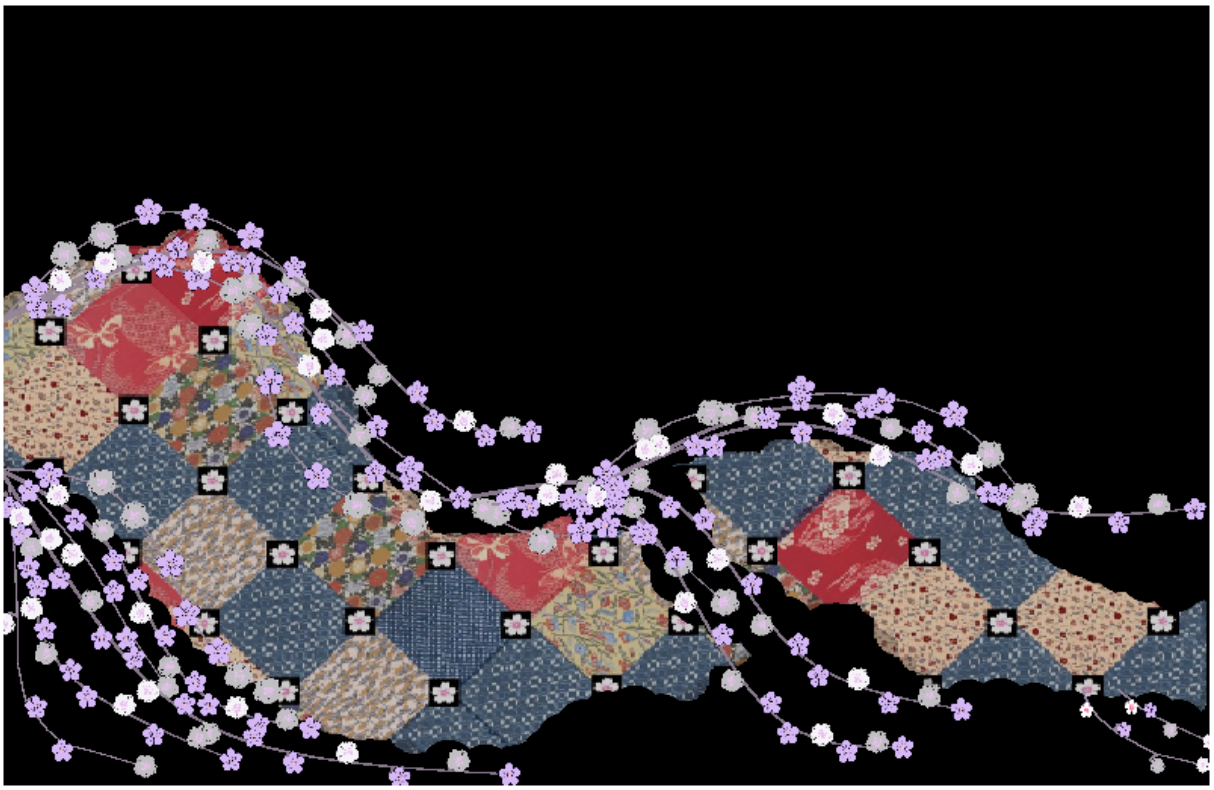


図 6.4　結果模様 4

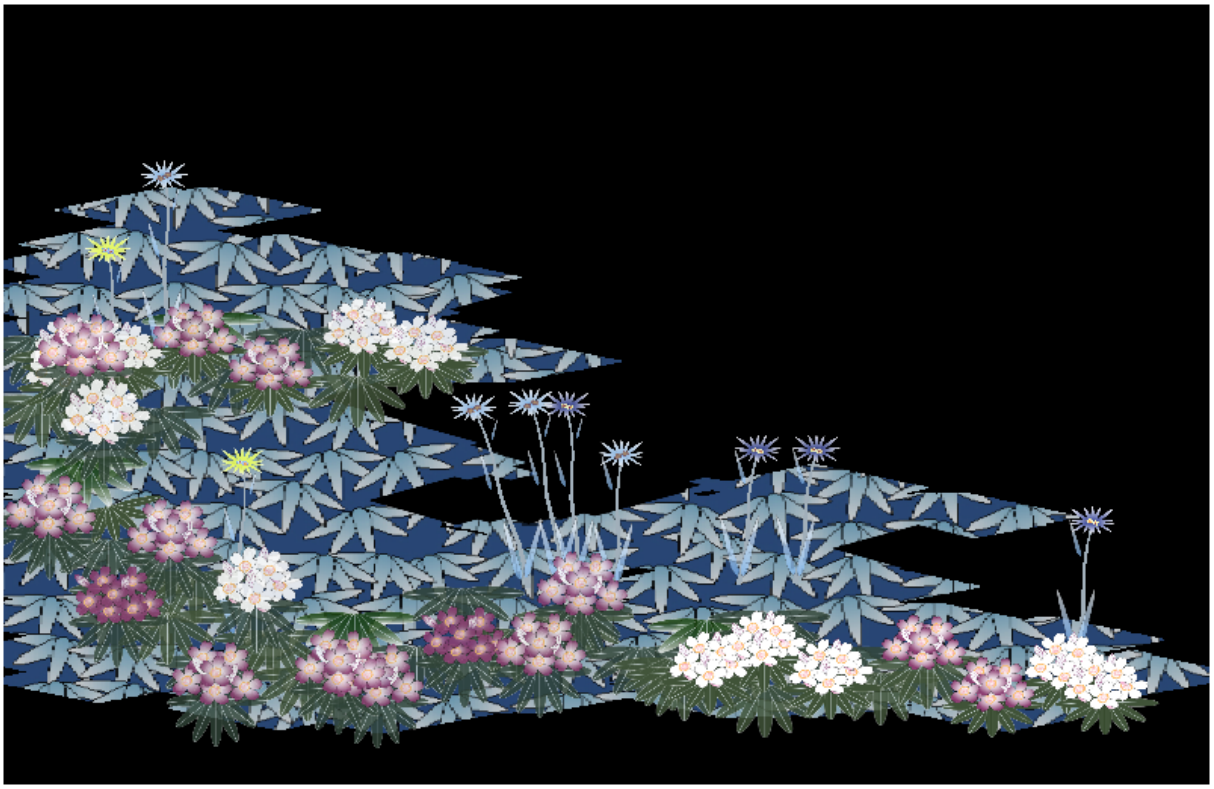


図 6.5　結果模様 5

図 6.6　結果模様 6

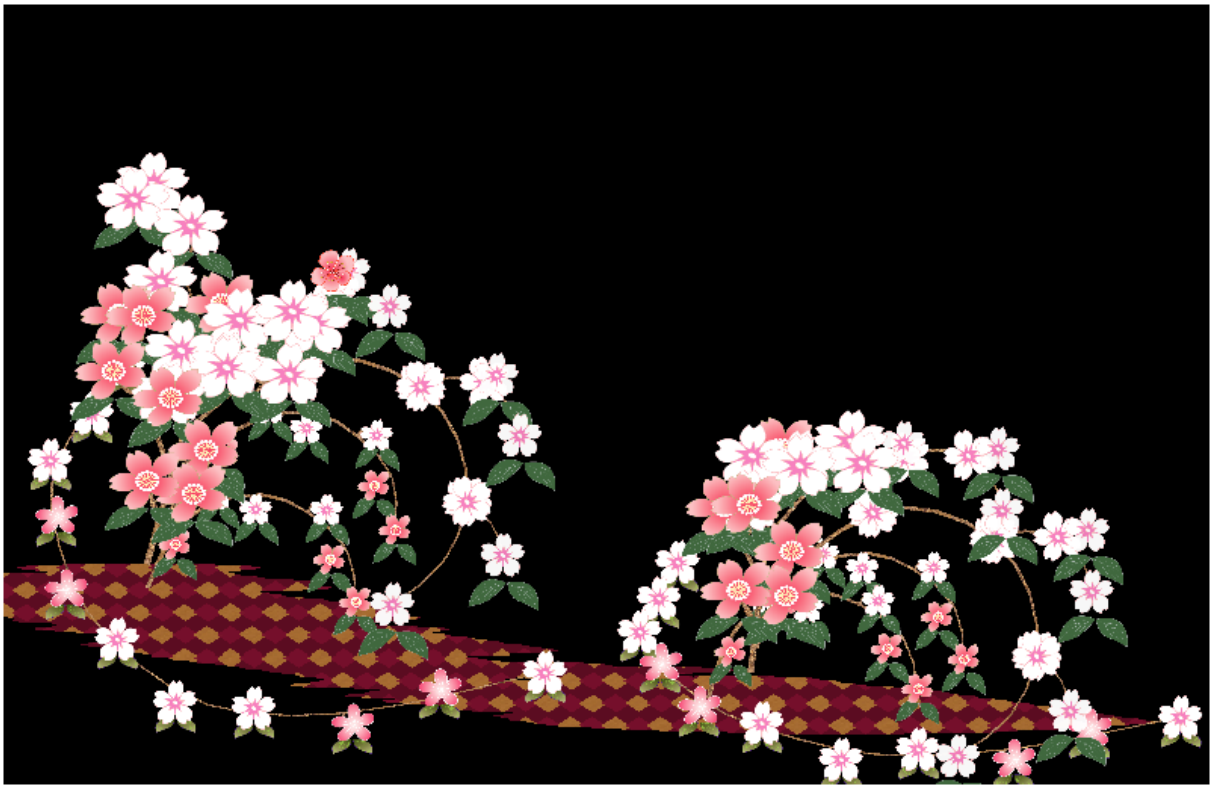


図 6.7　結果模様 7

図 6.8　結果模様 8

6.2　模様説明

6.2.1　　模様 1

この模様は，図 8.10 をモデルにして作成した．木模様には，3 章で述べた Tree6 のみを用いている．それぞれの木模様を，レイヤーを使い前後に配置して表現することで，奥行きのある模様を生成している．背景の模様には合成模様 1 を使い合成している．

6.2.2　　模様 2

この模様は，図 8.3 をモデルにして作成した．3 章で述べた Tree4 を用いて，前面の木模様を描いている．背景の山模様は，Tree8 を用いて，山の輪郭と山の内部の模様を生成している．

6.2.3　模様 3

この模様は，図 8.6 をモデルにして作成した．3 章で述べた Tree2 を使いすべての木模様を描いている．背景画像には，合成模様のタイプとして合成 5 を用いて，木模様の背後に描画している背景模様を描いている．木模様を前面と後面の二つに大きく分割し，その間に背景模様を描いていることが大きな特徴といえる．これは，レイヤー構造を考慮したことで実現できた模様といえる．

6.2.4　模様 4

この模様は，図 8.4 をモデルにして作成した．3 章で述べた Tree8 とスプライン補完を用いて，前面に描画している蔓のような植物模様を表現している．背景模様の画像は，予め用意しておいたものであり，それを合成したものである．

6.2.5　模様 5

この模様は，図 8.2 をモデルにして作成した．3 章で述べた Tree8 を使用し前面の模様を描いている．前面の模様は，複数のレイヤーに分けて生成しており，前後関係がある．背景の画像は，予め用意した画像を合成したものである．

6.2.6　模様 6

この模様は，オリジナルのデザインであり，加賀友禅模様から得られた基本的な流れを考慮しながら描いている．模様には，3 章で述べた Tree3 を用いて前面の木模様を描いている．また下部分にある葉模様は，Tree8 を用いて流れのある模様となっている．独自の模様ではあるが，着物模様らしい表現が実現できている．

6.2.7　模様 7

この模様も，オリジナルのデザインである．模様には，3 章で述べた Tree5 を主に使用している．Tree5 を木模様の中心におき，その他の蔓の部分を Tree8 で表現している．背景模様には，合成模様 2 を用いている．二つの画像を使い，背景の格子模様を表現している．

6.2.8　模様 8

この模様も，オリジナルのデザインである．模様には，3 章で述べた Tree1 を用いている．Tree1 を主な木模様として，下の葉模様には Tree8 を用いて表現している．背景模様には，合成模様 2 を用い，二つの画像を使用して生成している．

第　7　章

まとめ

ここでは，デザイン支援ツールに関してのまとめと，課題と展望について述べる．

7.1　研究のまとめ

本研究では，実際の加賀友禅の模様から共通する規則性を形式化し，それを考慮した誰にでも簡単に着物模様を生成できるデザイン支援ツールの実現を目指した．着物模様の分析の結果，「流れ」と「階層性」があることが分かり，この知見を基にシステムの構築を行なった．

システムの構築には，着物を生成するための生成アルゴリズムと，ユーザが簡単に模様生成できるユーザインターフェイスを実装した．模様生成には，植物模様を生成するための L-System の応用アルゴリズムと，背景画像生成のためのマスク合成処理などを実装した．また，ユーザが流れや階層性を簡単に考慮して模様生成できるようなユーザインターフェイスの実現を目指した．

結果として，6 章で述べた 8 つの模様を生成することができた．模様は，実際の加賀友禅を参考にしたものと，オリジナルでデザインしたものがある．実際の模様と比べてみると，花・葉などの色彩の違いはあるものの，非常に似たような印象を与える模様が生成できたと考える．また，オリジナルの模様に対しても，着物模様に極めて近いものが生成できた．

7.2　課題

7.2.1　色彩表現に関する課題

本研究では，模様の配置のみに注目し，色彩の調和などは考慮しなかった．そのた

め 6 章の結果模様を見ても分かるように，着物職人の求める質に及ばない部分が残されていることが分かる．その原因として，ユーザがあらかじめ用意した模様の画像を直接表示するだけでは，表現力が乏しいことなどが挙げられる．解決方法としては，表示した花・葉画像に対して模様らしい表現になるようなフィルタを開発することなどが考えられる．表現に用いるフィルタの具体例としては，花・葉画像に対してグラデーションを付加したり，ぼかし処理を施すものなどが挙げられる．

7.2.2　インターフェイスに関する課題

6 章で示した模様を完成するまでに，それぞれ 40～60 分程度の時間を要した．効率的なデザイン作業のためには，さらに短時間で模様をデザインするインターフェイスの開発が必要であるといえる．時間を要した主な理由として，以下の４つの理由が挙げられる．

①レイヤーの数に制限がある
②木模様の数に制限がある
③模様の配置を個別に行なわなければならない
④花・葉画像の適用に制約がある

①と②は，それぞれ，レイヤーの数，木模様の数を増加させるだけで対処可能である．③への対処法としては，いくつかの木模様をグループ化して配置できるようにすることなどが挙げられる．④への対処法としては，スクリーンに表示されている樹木の模様に対して花模様の画像を直接ドラッグ&ドロップして貼り付けができるような機能を付加することで解決できると考える．

7.3　展望

7.2 節で述べた課題を解決することで，実用的な模様生成ツールが実現できると考える．そして，実際に着物模様をデザインする職人に利用して頂き，その模様の配置をデータとして保存すれば，デザインセンスがないユーザでも，そのデータを出発点として，自分の意図した模様を生成することも可能となる．すなわち，プロのデザイン知を事例データベース化することで，着物デザインのコーパス構築も可能である．また，今回は留袖模様のみを対象としたが，晴着の模様，鞄，帽子などの模様などに対象を増やすことで，デザインの楽しさを人々に広めることにも貢献できると考える．

付録

サンプル画像

加賀友禅の模様集[2]から，無作為に 10 個の模様を選択した．以下にその 10 個の模様を示す．

図 8.1 サンプル画像 1

図 8.2 サンプル画像 2

図 8.3 サンプル画像 3

図 8.4 サンプル画像 4

図 8.5 サンプル画像 5

図 8.6 サンプル画像 6

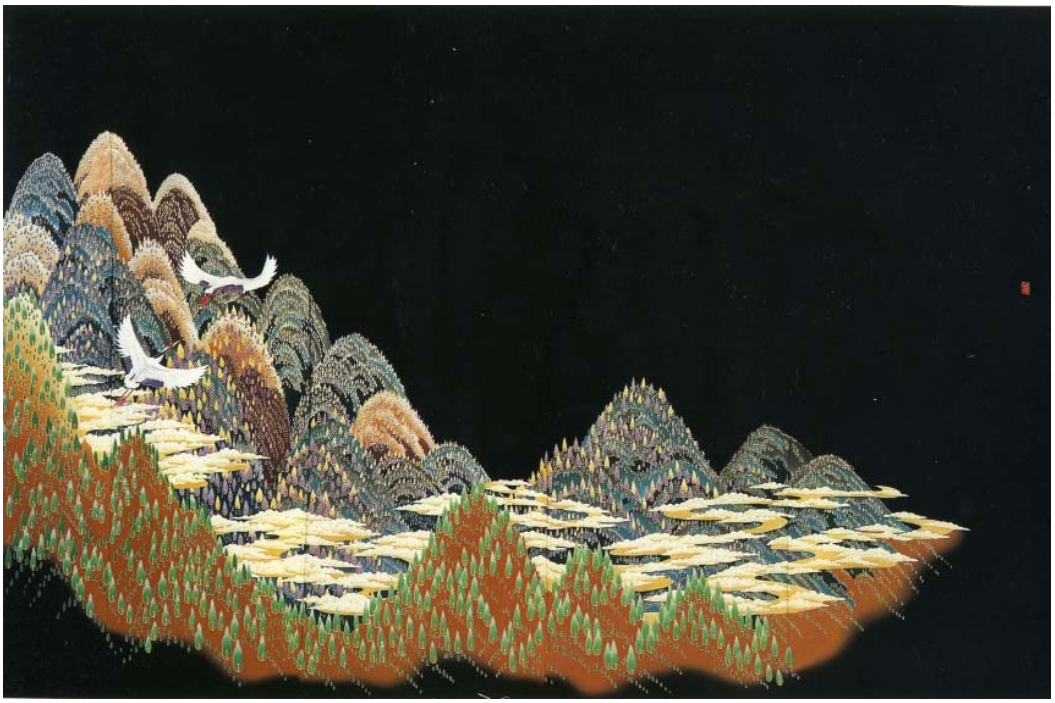


図 8.7 サンプル画像 7

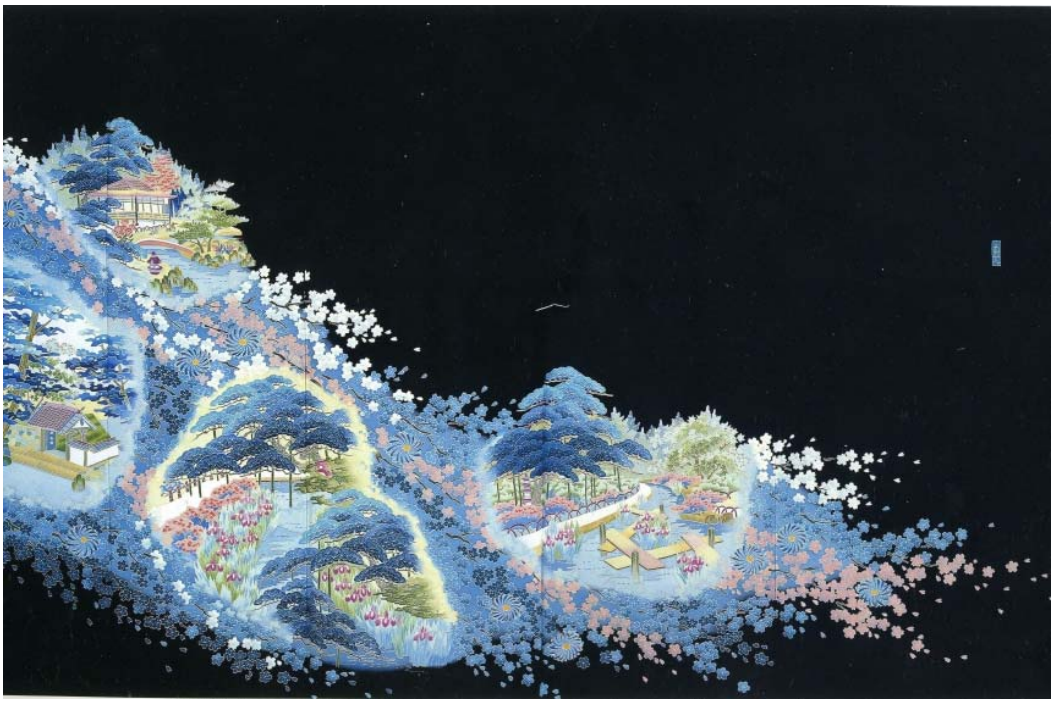


図 8.8 サンプル画像 8

図 8.9 サンプル画像 9

図 8.10 サンプル画像 10

謝辞

本研究をするにあたり，様々なアドバイスを頂いた宮田一乘先生には，非常に感謝しております．宮田研究室のこの 2 年間において，すばらしい体験をすることができました．本研究以外においても，紙相撲「ﾄﾝﾄﾝ」の製作は，ものづくりに対する難しさ，楽しさなどを学ぶことができ，さらに，アメリカのロサンゼルスでの展示など，普通では体験できない経験をすることができました．研究室のメンバーもこの一年で非常に増え，来期においても，多くの学生が志望しているようなので，この宮田研究室が，この後も，活躍していくことを願っています．

また，本研究において，実際の着物職人のの森田耕三先生にも，大変お世話になりました．森田先生には，着物模様の複雑さ，すばらしさを教えて頂き，そのアドバイスは非常にシステム構築において役立ちました．

副テーマにおいては，着物模様からの暗黙的な規則性の抽出において，野口尚孝先生に，様々なアドバイスを頂きました．また，去年の 9 月からは，野口先生からの引継ぎを快く承諾してくださった，永井由香里先生にも，副テーマにおいてのご指導を頂き本当に感謝しています．

その他，宮田研究室のメンバーは，非常に雰囲気のよいすばらしいメンバーだと思います．そういった良い雰囲気をこれからも宮田研のカラーとして，保ってほしいと思っています．

参 考 文 献

[1] Michael T. Wong, “Computer-Generated Floral Ornament”, Proc. of SIGGRAPH, pp.423-434, 1998.

[2] 加賀友禅 現代作家集 第 12 巻 フジアート出版.

[3] PRUSINKIEWICZ, P. and LINDENMAYER, A. 1990. “The Algorithmic Beauty of Plants”. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[4] Takashi Ijiri, Takeo Igarashi, “Floral Diagram and Inflorescences” Proc. of SIGGRAPH, pp.720-726, 2005.

[5] Philip J.Schneider, David H.Eberly, “Geometric Tools for Computer Graphics”, Morgan Kaufmann Publichers.

[6] Nelson S.-H. Chu and C.-L. Tai, “MoXi: Real-Time Ink Dispersion in Absorbent Paper”, Proc. of SIGGRAPH, pp.504-511, 2005.

[7] Theodore Kim, Ming Lin, “Visual Simulation of Ice Crystal Growth”, Proc. of SIGGRAPH, pp.86-97, 2003.

[8] Boudon, F. and Prusinkiewicz, P. and Federl, P. and Godin, C. and Karwowski, R., “Interactive design of bonsai tree models”, Computer Graphics Forum, Proc. of Eurographics, pp.591-599.

[9] DEUSSEN O. and LINTERMANN, B. 1999. “Interactive Modeling of Plants”. IEEE Computer Graphics and Applications, 19, 1, pp.56-65.

[10] DEUSSEN, O. and LINTERMANN, B. 1997. “A Modeling Method and User Interface for Creating Plants”. Proc. of Graphics Interface 97, pp.189-197.

[11] Joanna L. Power, A.J. Bernheim Brush, Przemyslaw Prusinkiewiczt, David H. Salesin “Interactive arrangement of botanical L-system models”, Proc. of SIGGRAPH, pp.175-182, 1999.

[12] Katsuhiko Onishi, Shoichi Hasuike, Yoshifumi Kitamura, Fumio Kishino “A Study for Interactive Modeling Trees by using Growth Simulation”.

[13] PRUSINKIEWICZ, P., HAMMEL, M., HANAN, J., and MĚCH, R. 1996. “L-systems: From the Theory to Visual Models of Plants”. Proc. of the 2 CSIRO Symposium on Computational Challenges in Life Sciences.

[14] PRUSINKIEWICZ, P., JAMES, M., and MĚCH, R. 1994. “Synthetic Topiary”. Proc. of ACM SIGGRAPH 94, ACM, 351-358.

[15] さくらほりきり“日本の模様　伝統の色と形”

[16] 藤原久勝　“着物模様事典”