

11-開-03

データベース構築促進及び技術開発に関する報告書

地形データを効率よくデータベース化する技術開発

平成12年3月

財団法人 データベース振興センター

委託先 株式会社セタ



この事業は、競輪の補助金を受けて実施したものである。

11-開-03

データベース構築促進及び技術開発に関する報告書
地形データを効率よくデータベース化する技術開発

平成12年3月

財団法人 データベース振興センター

委託先 株式会社セタ

序

データベースは、わが国の情報化の進展上、重要な役割を果たすものと期待されている。今後、データベースの普及により、わが国において健全な高度情報化社会の形成が期待される。さらに海外に対して提供可能なデータベースの整備は、国際的な情報化への貢献および自由な情報流通の確保の観点からも必要である。現在わが国で流通しているデータベースの中でわが国独自のものは約半数であるが、わが国データベースサービスひいてはバランスある情報産業の健全な発展を図るためには、今後もわが国独自のデータベースの構築およびデータベース関連技術の研究開発を強力に促進し、データベースの拡充を図る必要がある。

このような要請に応えるため、(財)データベース振興センターでは日本自転車振興会から機械工業振興資金の交付を受けて、データベースの構築および技術開発について民間企業、団体等に対して委託事業を実施している。委託事業の内容は、社会的、経済的、国際的に重要で、また地域および産業の発展の促進に寄与すると考えられているデータベースの構築とデータベース作成の効率化、流通の促進、利用の円滑化・容易化などに関係したソフトウェア技術・ハードウェア技術である。

本事業の推進に当って、当財団に学識経験者の方々に構成されるデータベース構築・技術開発促進委員会(委員長 東海大学教授 上條史彦氏)を設置している。

この「地形データを効率よくデータベース化する技術開発」は、平成11年度のデータベースの構築促進および技術開発促進事業として実施した課題の一つで、当財団が(株)セタに対して委託実施したものである。この成果が、データベースに興味をお持ちの方々や諸分野の皆様方のお役に立てば幸いである。

なお、平成11年度データベースの構築促進および技術開発促進事業で実施した課題は次表のとおりである。

平成12年 3 月

財団法人 データベース振興センター

平成11年度 データベース構築・技術開発促進事業委託課題一覧

区 分	No.	課 題 名	企 業 名
一 般	1	認定サービスにおけるデータベースの有効利用に関する調査研究	(株) シネジャーナルプロダクション
	2	ゲームソフトにおける知的財産権管理流通ビジネスに関する調査研究	京都リサーチパーク (株)
	3	地形データを効率よくデータベース化する技術開発	(株) セタ
	4	戦前期雑誌記事索引100万件の第一期データベース構築	(株) 皓星社
	5	POSデータに対するデータマイニング手法群の比較研究と実用化技術の開発	(株) 日経リサーチ
	6	北九州産業技術発展の歴史記録の収集とデータベース化	(株) 日鉄技術情報センター
地域振興	7	写真データベース構築	(株) 琉球新報社
	8	バリアフリー施設データベースのプロトタイプ作成	(株) 札幌ネクシス
	9	新居浜市機械産業における加工技術ノウハウ継承データベース構築	(株) 四国インターネット
	10	インターネットによるラジオニュース公開型データベース構築	(株) 山梨放送

1.背景と目的	1
2.実施体制	5
2.1.実施体制	5
2.2.実施スケジュール	5
2.3.開発環境および使用ソフトウェア	5
3.開発方針	6
4.基礎技術	8
4.1.ポリゴンデータ構造	8
4.1.1.位相構造保持の必要性	8
4.1.2.ハーフエッジデータ構造	9
4.1.3.ハーフエッジデータ構造で扱う要素	13
4.1.4.オイラーオペレータ	17
4.2.一般ポリゴンの三角形分割	22
4.2.1.望ましい三角形分割	22
4.2.2. <i>Delaunary</i> 三角形分割	22
4.2.3.三角形分割の方法	23
5.ライブラリ使用方法	26
5.1.等高線→等高線ポリゴン	26
5.2.植生ポリゴン+等高線ポリゴン→地形ポリゴン	35
5.3.植生ポリゴン+標高メッシュ→地形ポリゴン	46
6.ポリゴン形式の評価	51
7.まとめ	56

1. 背景と目的

近年のコンピュータグラフィックス関連の技術進歩により、物体の立体的な表示が容易に行えるようになりつつある。立体表示技術を応用することで、より多くの人にわかりやすく様々な情報を伝えられるようになると期待できる。

立体表示技術の適用が期待される分野のひとつに地形情報が考えられる。従来地形情報は平面地図によって表現されているが、立体表示技術を適用することで、標高を同時に表現することができる。これにより以下のような効果が期待できる。

- 地域防災システムや救急看護システムにおける避難、誘導経路の迅速、確実な検討。
- 気候情報と立体地形情報を組み合わせることによる、営農情報のわかりやすい提供。

コンピュータで何らかの情報を処理するためには、コンピュータ内部でどのように情報を表現するかが問題になる。このコンピュータ内部での情報の表現方法をデータ構造と呼ぶ。互いに異なるデータ構造では、同じ情報を表現しても、必要となるメモリ容量や処理時間に大きな差が生じることがある。

立体地形情報を表現するデータ構造としては、メッシュ形式とポリゴン形式の二つが考えられる。以下に両データ構造の概要と特徴をまとめる。

● メッシュ形式

立体地形情報を表現する一般的なデータ形式。国土地理院が発行する標高数値地図にも採用されている。

地形を縦、横の等間隔の境界線(グリッド)で小さな升目(セル)に分割し、各セルの代表点の高度や植生を記録する。領域を網目(メッシュ)のように分割することからメッシュ形式と呼ばれる(図 1-1)。

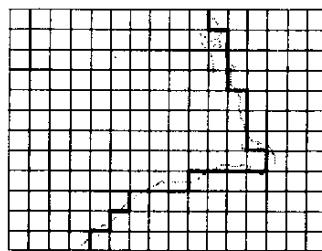


図 1-1:メッシュ形式

- 長所

単純な構造であるためデータの意味が分かりやすくプログラミングが簡単になる。

- 短所

セルに対して情報を格納するためセルよりも細かい構造を表現できない。

行政区境界のように本質的に線状の情報や、河川や道路のように縮尺によっては線として表現することが適切な情報の表現が困難。

適切な表現精度を持ったセルの大きさの決定が困難。

例えば単純な形状の海岸線と複雑な形の池が近接する場合では、海岸線に合わせてセルの大きさを決めると池の表現精度が確保できなくなり、池に合わせてセルの大きさを決めると海岸線では必要以上にセルが小さくデータ容量が大きくなる(図 1-2)。

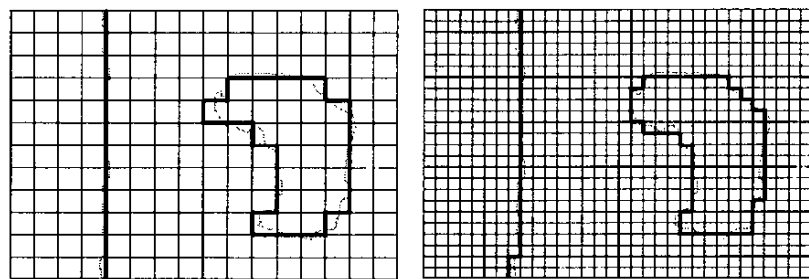


図 1-2: メッシュ形式の最適セルサイズ決定が難しい例

- ポリゴン形式

コンピュータグラフィックスの分野で立体を表現する一般的なデータ構造。Web ページ上での立体表現に用いられる VRML 言語は基本的にはポリゴン形式でデータを表現している。

物体の表面を多角形(ポリゴン)で近似して表現する。

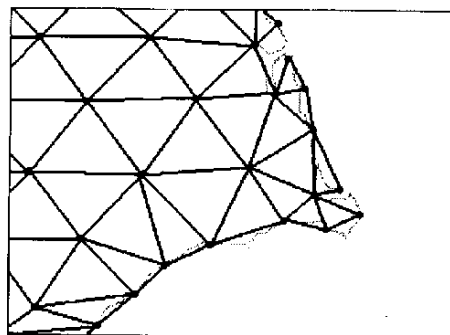


図 1-3:ポリゴン形式

- 長所

自由度が高いため表現力が高く、異なる表現精度のものを同時に扱うことができる(図 1-4)。

多角形の縁に情報を持たせることで、線状の構造も正確に表現することができる。

最近の表示用ハードウェアにはポリゴンを高速に描画する機能が備わっている場合が多く、安価に高品位の表示を行うことができる。

- 短所

データ構造としては複雑なので、プログラミングやデータ作成が比較的困難。

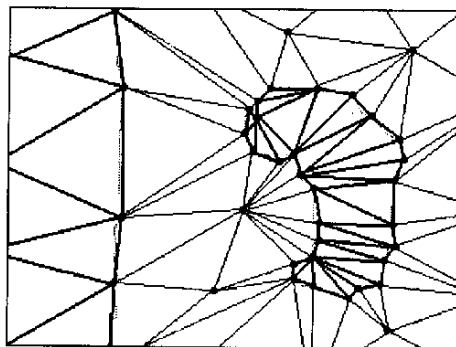


図 1-4: 図 1-2のポリゴン形式による表現

両データ構造の特徴を比較すると、従来一般的に使用されてきたメッシュ形式よりもポリゴン形式のほうが立体地形情報のデータ構造としては望ましいと考えられる。

特にポリゴン形式では必要な表現精度をメッシュ形式に比べ少ないメモリ容量で実現できることに注目すると、以下の展開なども期待できる。

- 地形データをデータ通信によって配信する。
- 組み込み機器のようにメモリ資源が限られた機器で地形情報を扱う。

一方、地形情報を効率よくポリゴン形式に変換する確立された方法は広く知られていないため、現状では多量の地形情報をポリゴン形式に変換し活用することは困難である。ポリゴン形式の活用には、効率的なデータ作成方法の確立が望まれる。

本開発は以下を目的に実施した。

- 立体地形情報のポリゴン形式による表現を広く利用可能にするために、地形図に記載された地形情報をポリゴン形式に効率よく変換するための基礎技術を開発する。
- 立体地形情報のポリゴン形式による表現と、従来のメッシュ形式による表現との表現能力、必要メモリ容量の比較検討を行い、ポリゴン形式の利点を検証する。

2. 実施体制

2.1. 実施体制

本開発の実施体制は以下の通りとする。

- 社長 : 富士本 淳
- 取締役 : 高尾 賢次
- 取締役 : 野中 誠之
- 経理部 : 井関 秀雄
- 開発部 : 菅原 徹
- 開発部 : 渋谷 充宏
- 補助員 : 駒形 和重
- 補助員 : 金子 俊行

2.2. 実施スケジュール

本開発の実施スケジュールを図 2-1に示す。

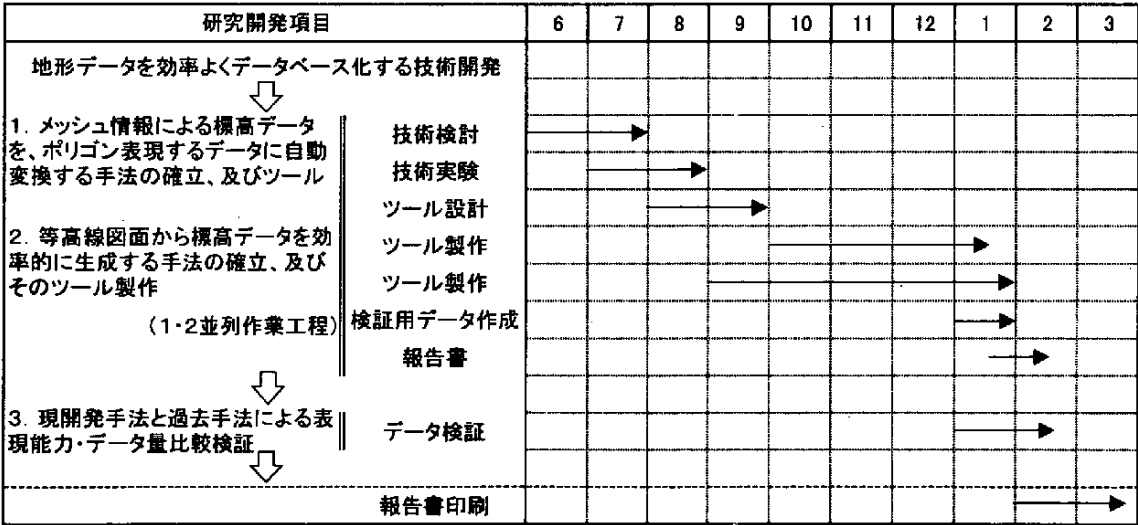


図 2-1:実施スケジュール

2.3. 開発環境および使用ソフトウェア

WinNT4.0 SP4 VC6.0SP3

3. 開発方針

この節では、地形図に記載された地形情報のポリゴン形式への変換の方針を検討する。

地形情報は、平面情報と標高情報の大きく二つに分けて考えることができる。それぞれの特徴を以下にまとめる。

- 平面情報

- ◇ 従来の画像データ化に関する様々な技術を適用することができるため、データ化は比較的容易である。
- ◇ 地形データの作成用途によって記載すべき内容が変化するため、用途に応じて独自に作成する必要がある。

- 標高情報

- ◇ 地形図からデータ化するための広く知られた方法はなく、データ化は比較的困難である。
- ◇ 地形データの作成用途によらないため、対象領域が一定であれば同一データを利用することができる。そのため、一般的に利用可能なライブラリを用意しておけば、多くの場合は独自に作成する必要はなくなる。

これらの特徴から、平面情報と標高情報を地形図から分離して入力し、それぞれを独自にデータ化し、最終的に合成する手法が効果的であると考えられる。

地形情報を平面と標高情報に分離する手法が効果的であるためには、標高データの入手が容易でなくてはならない。

一般的な用途には、国土地理院が発行する数値地図が利用できる。この数値地図は、日本全土を約 50m 間隔メッシュに分割したときの各メッシュの標高が電子データとして記録されている。

しかし、より細かい精度の標高データが必要な場合などには独自に標高データを作成する効果的な手段が必要になると思われる。

地形図における最も一般的な標高情報の表現は等高線表記であると考えられるので、等高線情報から標高データの生成方法を開発することで、独自の標高データ作成が効率的に行えるようになると思う。

開発の目的は基礎技術の開発であるため、作成対象はアプリケーションプログラムではなく、一般に利用可能なライブラリプログラムとする。

本開発の開発方針を以下にまとめる。

- 立体地形データの作成方針として、平面情報と標高情報を地形図から分離して入力し、それぞれを独自にデータ化し、最終的に合成する手法を採用する。
- 一般的用途の標高データとしては、国土地理院発行の数値地図の使用を想定する。
- 数値地図では標高データとして不十分な場合に対応するために、等高線データから標高データを作成する手法を開発する。

4. 基礎技術

4.1. ポリゴンデータ構造

この節では本開発で利用したポリゴンデータ構造について概説する。

4.1.1. 位相構造保持の必要性

コンピュータによる計算では数値の精度が有限であるために、数値計算の結果に誤差が生じる場合がある。図形同士の交差や重なりの有無を調べる計算の途中に数値誤差が発生すると、交差状態などが誤って判断される場合がある。数学的・論理的な正当性のみを考慮し、このような誤判定が起こりうることを考慮せずに構成されたアルゴリズムでは、誤った結果を出力したり、場合によっては途中で処理が続けられなくなりアルゴリズムが破綻してしまうこともある。

誤った結果を出力する例を図 4-1(a)に示す。これは四角形 AECF を三角形分割する問題である。ただし、四角形の内部に点 B があるものとする。図 4-1(a)の分解は点 A, B, C が一直線上にあるため、 $\triangle ABC$ の面積はゼロにはなるが数学的には正しい分割とみなすことができる。しかしこの三角形分割に対して、任意の点に対応する三角形を探索するアルゴリズムを適用すると誤った結果を出力する場合がある。具体的には、数値誤差により線分 AB と線分 AC が厳密に重なり合わず、隙間があるように判定され、点 D に対応する三角形が存在しないと判定される場合がある(図 4-1(a)下)。

上記のような数値誤差による幾何処理の不具合を回避するための方法のひとつに、位相構造を優先させる方法がある。これは、計算誤差が生じる環境で計量的性質を厳密に求めることは困難だが、組み合わせ論的に決定される位相情報は厳密に求めることが比較的容易なことに注目し、要求される位相的な条件を満たした出力が常に得られるようにアルゴリズムを設計する方法である。上記の三角形分割の例では「内部の辺はふたつの三角形に共有される」という位相的性質を出力に要求することで、点の探索アルゴリズムに適した出力を得ることができる(図 4-1(b))。

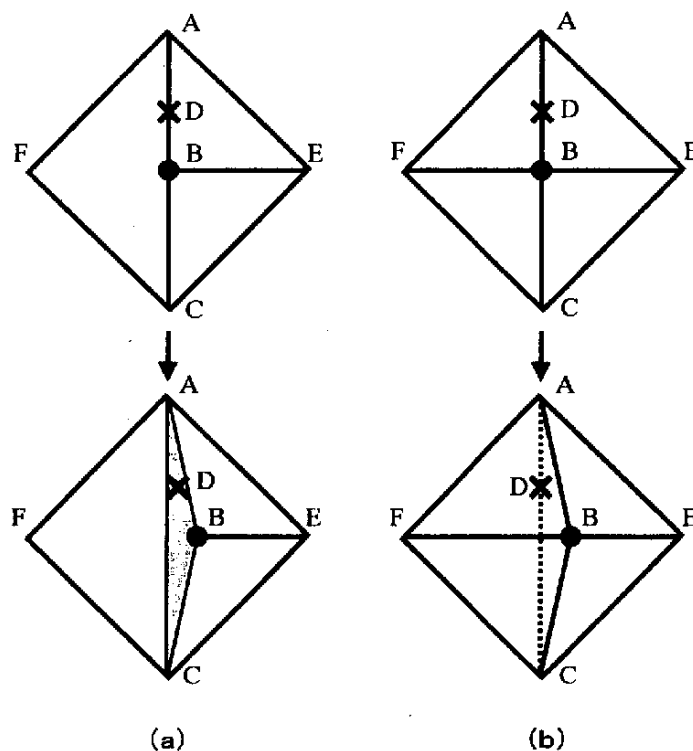


図 4-1: 四角形における三角形分割の図

4.1.2. ハーフエッジデータ構造

位相構造の保持を考慮したポリゴンデータ構造に、ハーフエッジデータ構造が提案されている。本開発ではこのデータ構造を採用する。

(1) 基本アイディア

現在の実用的なCADに用いられているデータ構造の基本的な考え方は、1974年にB. G. バームガード(B.G.Baumgart)によって提案されたウイングドエッジデータ構造という名で知られている(図 4-2)。

このデータ構造は、単純な面分のみによって構成された多面体を対象とし、その表と裏を区別し、稜線を中心として隣接する頂点、稜線、面分の接続関係をポイントなどによって陽に記述している。このデータ構造と立体生成のための方法であるオイラーオペレータを用いることにより、クラインの壺のような非現実的な位相を持つ立体が発生しないことが保証される。また、形状要素が連続的に接続しているという性質が、コンピュータ処理に適した形式で表現されているので、能率的な形状処理を行うことができる。

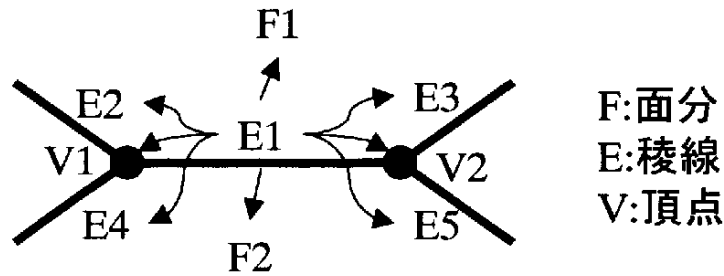


図 4-2:ウイングドエッジデータ構造

(2) ハーフエッジデータ構造

ウイングドエッジデータ構造では、探索のさいに、頻繁に判断処理が必要とされる。そこで、探索のさいの判断処理を不必要とし、より能率的な処理を可能とするデータ構造に、現在のCADにおいて広く使われているハーフエッジデータ構造がある(図 4-3)。

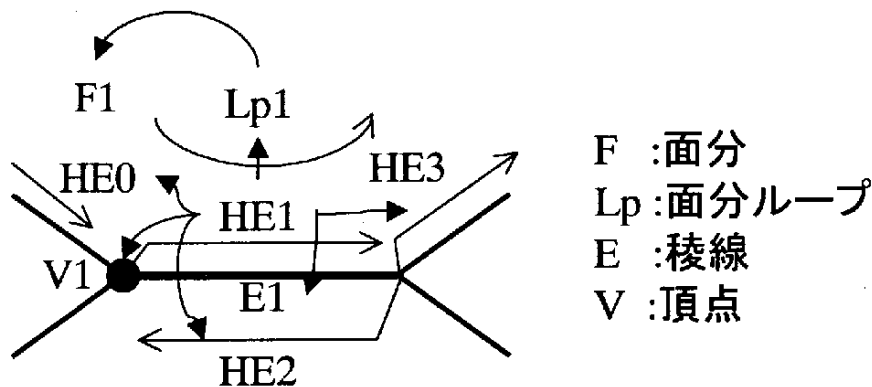


図 4-3:ハーフエッジデータ構造

ハーフエッジデータ構造では、稜線をふたつに分割したハーフエッジと呼ばれる幾何要素を導入し、ハーフエッジを中心にデータ構造を構築する。ウイングドエッジデータ構造では稜線に対応する面分と頂点がそれぞれふたつずつ存在し、頻繁な判定処理が要求されたが、稜線を分割しハーフエッジとすることで、対応する面分と頂点を一意に決めることができる。ハーフエッジ構造は頻繁な判定処理を行わずに、位相構造の効率的な取り扱いを可能にするデータ構造である。

図 4-3に示すハーフエッジHE1は以下の要素へのポインタを持っている。

- ・ 対応する頂点V1
- ・ HE1が所属するループLp1
- ・ 同じ稜線に対応するハーフエッジHE2

- 同一面分ループ(後述)に属する手前のハーフエッジ HE0 と次のハーフエッジ HE3
- 対応する稜線 E1

また面分ループ Lp1 は Lp1 が境界を定義している面分 F1 へのポインタを持っている。

面分ループ(Face Loop)は多重連結な面分(図 4-4)の記述を可能とする方法として、ハーフエッジ構造に対し、新たな形状要素として導入されている。図 4-4 の面分はいくつかの頂点と稜線からなる回路によって、境界が定められている。この境界を示している Lp1、Lp2、Lp3 を面分に対する面分ループと呼ぶ。

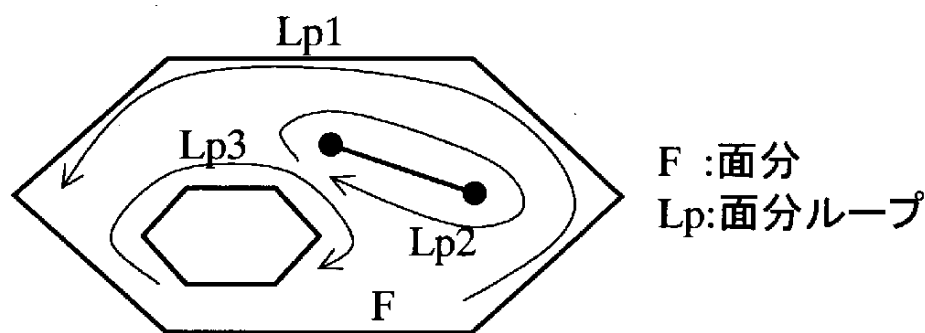


図 4-4: 面分ループ

これらのポインタを順にたどることで、探索を能率的に行うことができる。

たとえば、ある頂点に接続した全ての稜線を巡回するには、頂点が示すハーフエッジから稜線を得て、さらにそのハーフエッジから対になっているハーフエッジを得る。そのハーフエッジから次のハーフエッジを得て、そこから稜線を得ることを続けていけば、ある頂点に接続したすべての稜線を巡回することができる。

また、ある面分を構成する全ての頂点を巡回するには、面分を構成している面分ループを得て、さらに面分ループを構成しているハーフエッジを得て、そのハーフエッジより頂点を得ることで、面分を構成している全ての頂点を巡回することができる。

(3) オイラーオペレータの概念

多くのポインタがお互いを指すデータ構造では、ポインタのつなぎ替え操作が複雑でプログラミング上のミスがおこりやすい。プログラミングミスを減らすために、ハーフエッジデータ構造に対して行う必要最低限の単純な操作をサブルーチンとして提供し、それらのサブルーチンを組み合わせて使用することで、複雑な操作を実現することが提案されている。

この方針を採る場合、どのような操作を行うサブルーチンを用意すれば必要十分であるかを詳細に検討する必要がある。ハーフエッジデータ構造に対して、オイラー

オペレータと呼ばれる一連の操作を適切な順序で適用することで、任意の多面体が生成できることが示されている。

オイラーオペレータは、18世紀の数学者オイラー (Leonhard Euler) によって発見され、後にポアンカレ (H. Poincaré) により一般化された、多面体の頂点、稜線 (線分)、面分の数の間に存在する関係式によって特徴付けられる。本開発で用いるハーフエッジデータ構造は、点と線分と面分よりなる図形で貫通穴の無い構造の多面体である。この図形に対して成立するオイラーの関係式は、 v を頂点数、 e を稜線数、 f を面分数としたとき、以下で表される。

$$v - e + f = 2$$

右辺の2は、オイラー指数と呼ばれる。

オイラーオペレータは、オイラー指数を一定に保つように設計されている。

4.1.3. ハーフエッジデータ構造で扱う要素

この節ではハーフエッジデータ構造で扱うデータ要素と、要素間の接続情報を示す。

(1) シェル (Shell)

シェルは1つの多面体を表す。多面体とは互いに連結な面の集合のことである。

シェルが直接管理する情報は以下のとおりである。

- ①自分が所有する頂点集合
- ②自分が所有する面分集合
- ③自分が所有する稜線集合

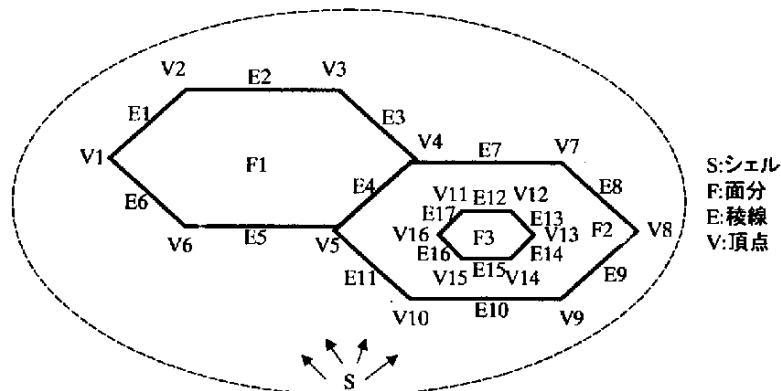


図 4-5:シェルからの情報の取得

(2) 面分 (Face)

面分とは1つ以上の頂点列で囲まれる図形で、点と線分を境界とする面のことである。また、面分に穴が空いている場合もあるが、これは面分ループを複数持つことによって対応している。

面分が直接管理する情報は以下のとおりである。

- ①自分を所有しているシェル
- ②自分が所有するループ集合

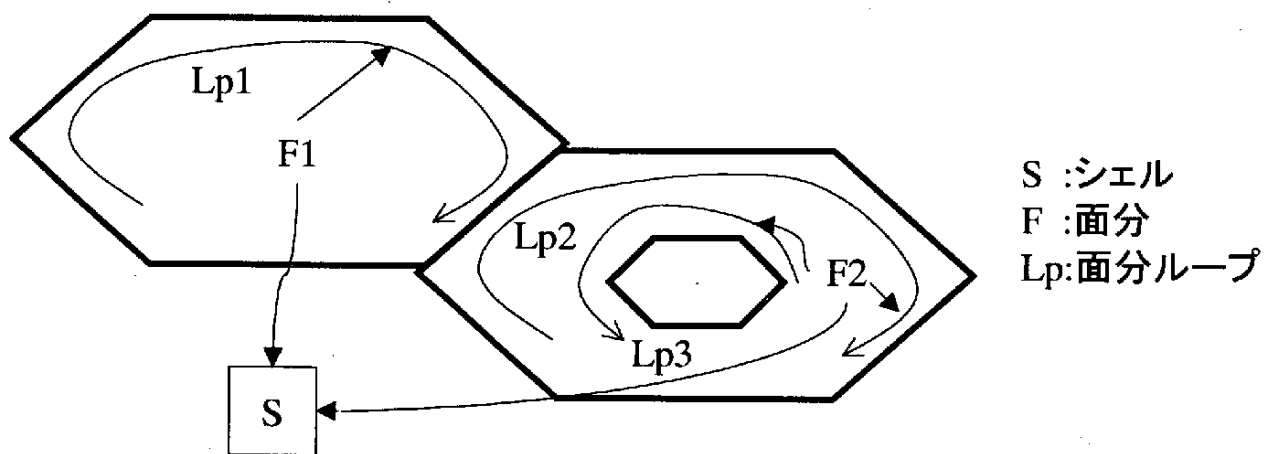


図 4-6: 面分からの情報の取得

(3) ループ (Loop)

ループとは面分の境界を構成する頂点の回路のことである。

ループが直接管理する情報は以下のとおりである。

- ①自分を所有している面分
- ②自分が所有するハーフエッジ集合

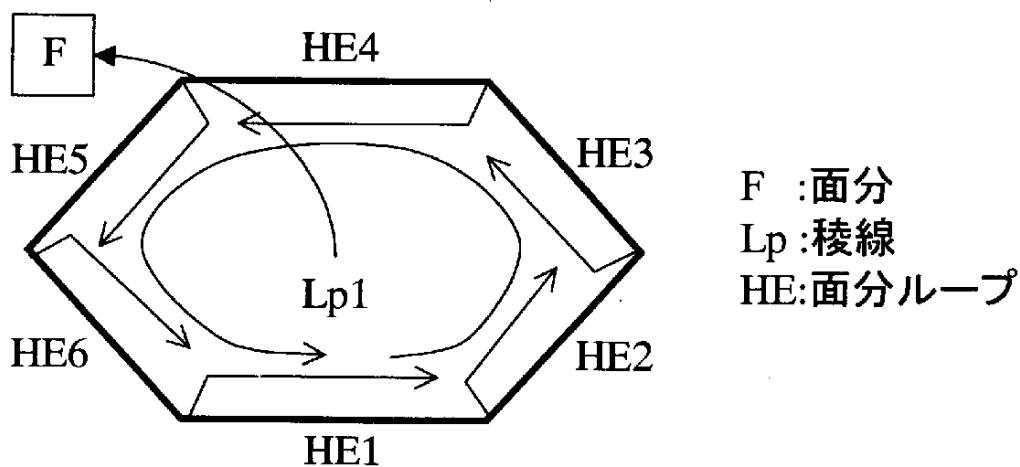


図 4-7: ループからの情報の取得

(4) 稜線 (E d g e)

稜線とは面分を構成している線分のことである。2つの対のハーフエッジによって、稜線を表現することができる。

稜線が直接管理する情報は以下のとおりである。

- ① 自分に対応するふたつのハーフエッジ

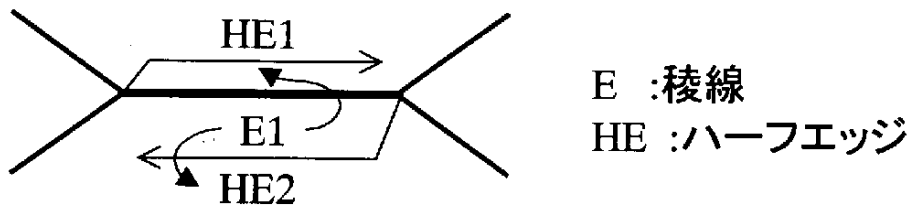


図 4-8: 稜線からの情報の取得

(5) ハーフエッジ (H a l f E d g e)

ハーフエッジとは稜線を2つに分けたものである。ループ上で隣り合うハーフエッジと双方向リンクリストを構成構成しており、ループ上の頂点の巡回や、頂点に接続する稜線の巡回を効率よく行うことができる。

ハーフエッジがが直接管理する情報は以下のとおりである。

- ① 自分と対のハーフエッジ
- ② 自分に対応する頂点
- ③ 自分に対応するループ
- ④ 自分に対応する稜線
- ⑤ 自分に対応する面分
- ⑥ 面分ループ中心に次のハーフエッジ
- ⑦ 面分ループ中心に前のハーフエッジ
- ⑧ (頂点ループ中心に次のハーフエッジ)
- ⑨ (頂点ループ中心に前のハーフエッジ)

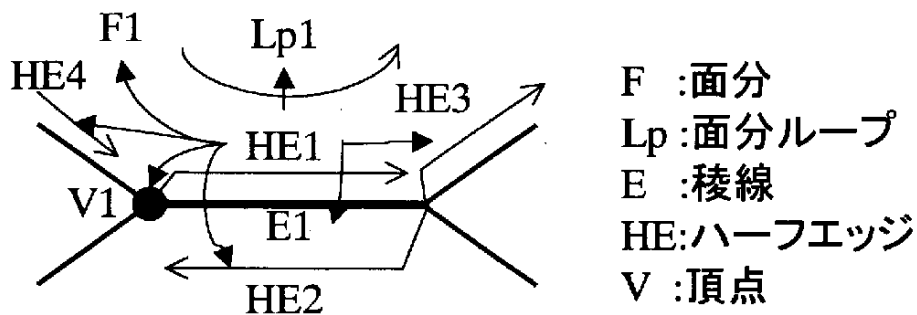


図 4-9: ハーフエッジからの情報の取得

(6) 頂点 (Vertex)

頂点とは面分を構成している点のことである。

頂点が直接管理する情報は以下のとおりである。

- ①自分を所有しているシェル
- ②自分に対応する任意の1つのハーフエッジ

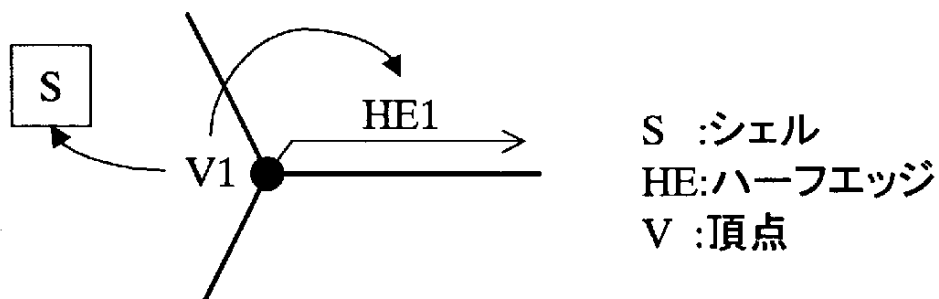


図 4-10: 頂点からの情報の取得

4.1.4. オイラーオペレータ

この節では、ハーフエッジデータ構造を操作するオイラーオペレータを示す。

(1) 核図発生オペレータ

(A) csvfl (CreateShellVertexFaceLoop)

csvflは、形状生成の一番初めに実行されるオペレータで、1つの頂点、1つの面分、1つのループよりなる退縮したシェルを発生する(図 4-11)。

(B) dsvfl (DeleteShellVertexFaceLoop)

dsvflは、csvflとは全く逆の操作、すなわち1つの頂点、1つの面分、1つのループよりなる退縮したシェルがあるとき、それらをすべて除去し無の状態にする(図 4-11)。

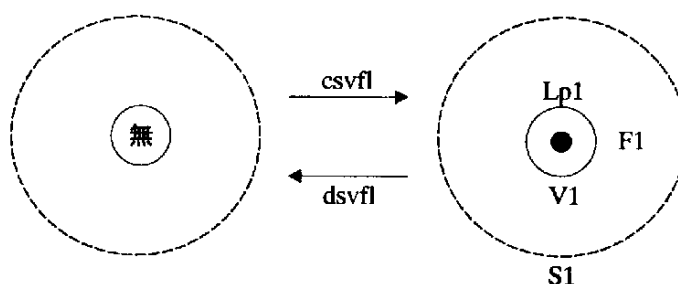


図 4-11: csvflとdsvflの操作

(2) 局所処理オペレータ

局所位相の操作を行うオペレータは以下に示す(A)(B)(C)(D)(E)(F)である。

(A) cev (CreateEdgeVertex)

cevは、存在する頂点を2つに分割してその間に稜線を発生させ、その結果頂点と稜線が1つずつ増加する(図 4-12(a))。cevは、csvflを実行直後の独立点にも適用されるし(図 4-12(b))、また図 4-12(c)の場合のように、処理の途中で一時的に作られるストラット稜線の発生にも適用される。

図 4-12(a)では頂点V1をオペレータに渡すことにより、V2と、稜線E1ができる。

図 4-12(b)(c)では頂点V1、V2をオペレータに渡すことにより、稜線E1ができる。

(B) dev (DeleteEdgeVertex)

devは図 4-12に示すcevの逆操作を行う。すなわちdevは、2つの異なる頂点を結ぶ稜線が与えられたとき、その稜線を除去し、頂点を1つに併合する。

図 4-12(a)(b)(c)では頂点V2をオペレータに渡すことにより、頂点V2と稜線E1がきえる。

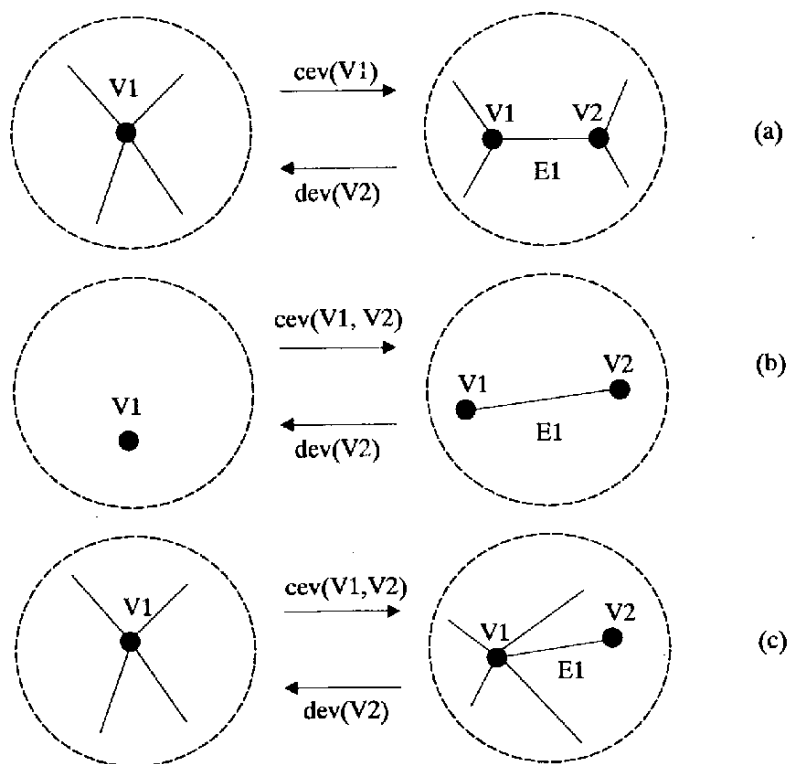


図 4-12: cevとdevの操作

(C) cefl (CreateEdgeFaceLoop)

ceflは面分の2つの頂点間に稜線を渡すことにより2つの面分に分割し、その結果面分と稜線が1つずつ増加し、またループも1つ増加する(図 4-13(a))。ceflはcsvflを実行直後の独立頂点にも適用されるし(図 4-13(b))、また、図 4-13(c)の場合にも適用される。

図 4-13(a)ではハーフエッジHE1, HE2をオペレータに渡すことにより、稜線E1と面分F1とループLp1ができる。

図 4-13(b)(c)ではハーフエッジHE1をオペレータに渡すことにより、稜線E1と面分F1とループLp1ができる。

(D) defl (DeleteEdgeFaceLoop)

deflは図 4-13のceflの逆操作、すなわち2つの異なる面分を共有する稜線が与えられると、その稜線を除去し、面分を1つに併合する。

図 4-13(a)(b)(c)では稜線E1をオペレータに渡すことにより、稜線E1と面分F1とループLp1がきえる。

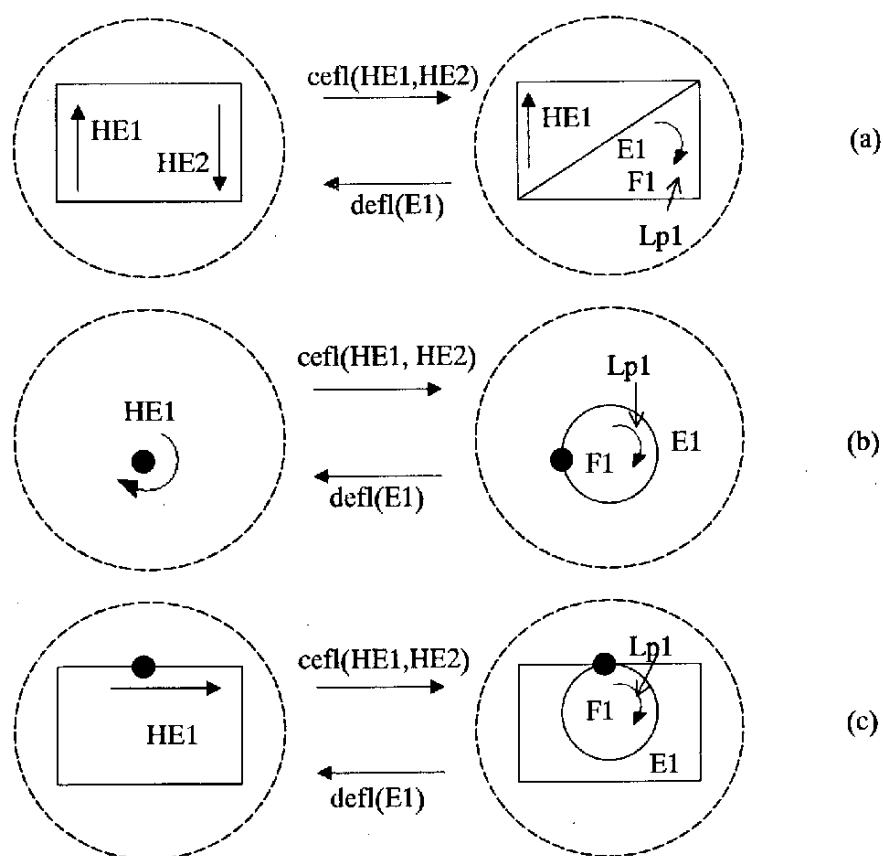


図 4-13: ceflとdeflの操作

(E) decl (DeleteEdge_CreateLoop)

declはループにおいて2回現れる稜線を除去し、ループを2つに分割する(図 4-14(a))。declは図 4-14(b)(c)に示す特別な場合も適用される。

図 4-14(a)(b)(c)では稜線E1、頂点V1をオペレータに渡すことにより、稜線E1が消え、ループLp2ができる。

(F) cedl (CRateEdge_DeleteLoop)

cedlは図 4-14のdeclの逆操作、すなわち2つのループの頂点間に稜線を渡して、1つのループに併合する。

図 4-14(a)(b)(c)ではハーフエッジHF1, HF2をオペレータに渡すことにより、稜線E1ができ、ループLp2がきえる。

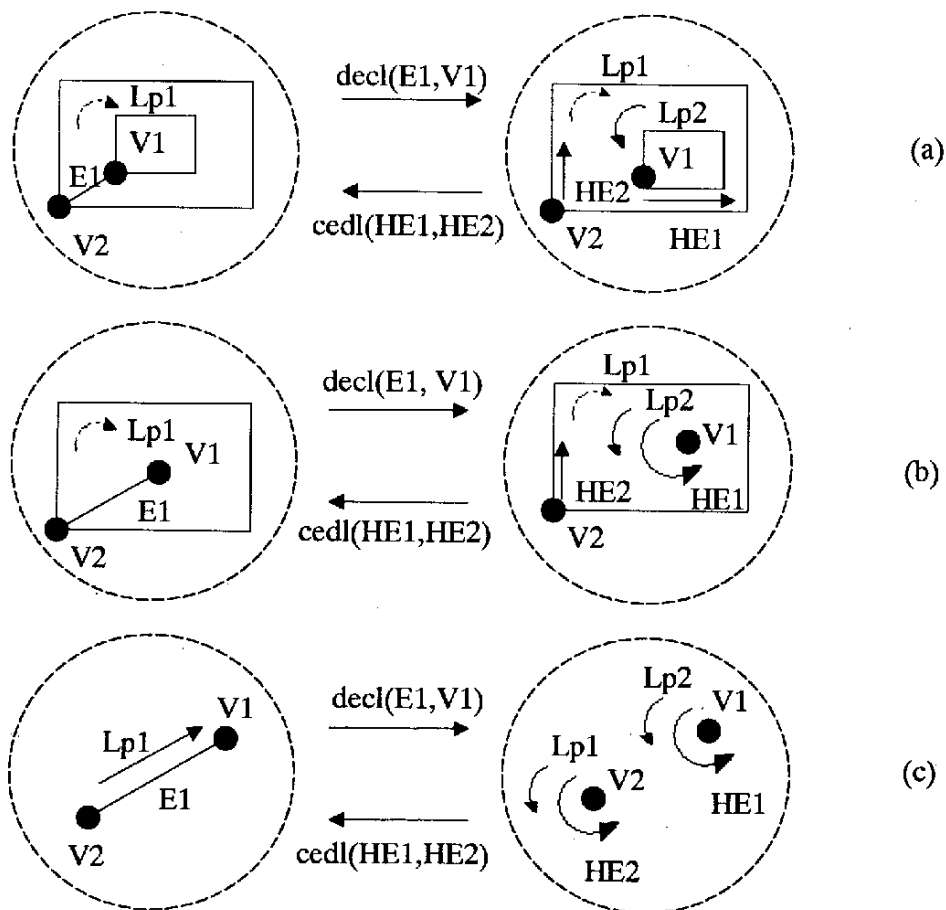


図 4-14: declとcedlの操作

(3) 大局処理オペレータ

前述のオペレータが局所の位相操作に限定されているのに対し、次に述べる操作は、シェルを2つに分けたり、またシェルに貫通する穴(ジーナス)をあけるなどの大局的な位相操作を行う。今回の開発にジーナスはないので用いないが、参考までに示しておく。

(A) dfds (DeleteFace_DevideShell)

dfdsは2つの面分f1、f2が与えられたとき、f1にf2のループを加えて多重連結な面分とし、f2を除去する(図 4-15(a))。ここに図 4-15(a)の右図において、中央は穴になっている。この操作により、f1、f2が同一のシェルに属する場合には、ジーナスが1つだけ増加する。f1、f2が異なるシェルに属する場合には、dfds、2つのシェルを1つに結合する働きをする。図 4-15(b)は、dfdsにより輪環面が作られる場合を示している。

図 4-15では面分F1、F2をオペレータに渡すことにより、面分F2がきえ、ループLp1ができる。

(B) cfjs (CreateFace_JoinShell)

cfjsはdfdsの逆操作で、普通には物体を貫通する穴を埋め、ジーナスを1つだけ減少させる。

図 4-15ではループLp1をオペレータに渡すことにより、面分F2ができ、ループLp1がきえる。

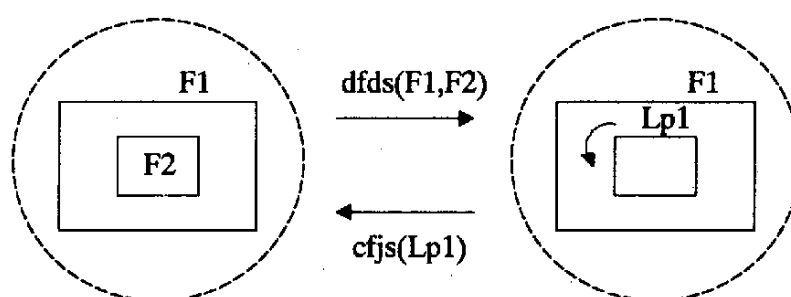


図 4-15: dfdsとcfjsの操作

4.2.一般ポリゴンの三角形分割

多角形を三角形分割する方法はいくつか知られている。本開発で採用したDelaunary 三角形分割について概説する。

4.2.1.望ましい三角形分割

ポリゴンデータを生成する場合には、平面データ内で多角形として表現されている領域を分割し、三角形の集合として扱えると都合が良いことが多い。しかし多角形の三角形への分割は一般に一意に定まらないため、考えられる分割の中から望ましい分割を選択する必要がある。

曲面の三角形メッシュによる近似の理論によれば、鋭角を含む三角形分割は、標高の変化率が大きくなり、近似精度が悪くなることが知られている。

よい三角形分割と悪い三角形分割の例を図 4-16に示す。

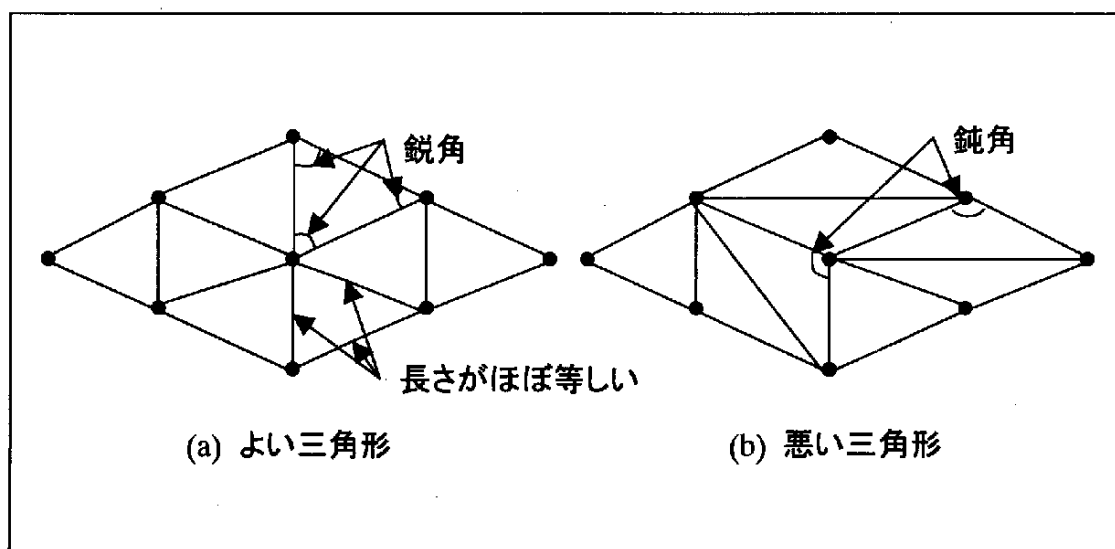


図 4-16: よい三角形と悪い三角形

4.2.2.Delaunary 三角形分割

望ましい三角形分割を正確に定式化したものに、Delaunary 三角形分割がある。これは分割結果三角形の内角の最小角度が最大となるような分割方式のひとつである。この分割は、曲面の近似としても望ましい性質を持っていることが知られている。

4.2.3. 三角形分割の方法

まず、元図形領域を形成する頂点群を一つの面分ループとして多角形を作成する。これを面分ループ作成とする。

次に、その面分ループの連続する3頂点のうち、正しく三角形を構成できるものを任意に選び、三角形に分割して面分ループから切り離してゆく。この操作により、多角形の頂点数が三角形分割により1つずつ減少していき、最終的には頂点数3の三角形となる。また、分割した三角形の中に頂点がある場合、頂点を中心にさらに三角形に分割する。

この時点で多角形は三角形に正しく分割されているが、図 4-17 (b) のように望ましくない三角形が多く含まれている。この分割を改良して **Delaunary 三角形分割** にするために、フリッピングと呼ばれる方法を適用する。

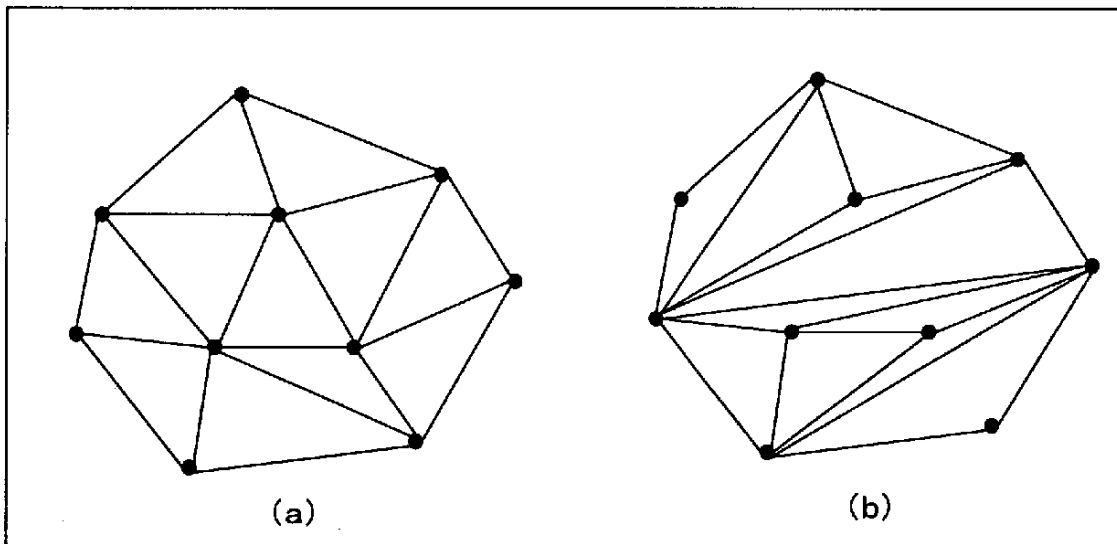


図 4-17: 三角形メッシュ

フリッピングとは、稜線を入れ替える手続きを指す。三角形分割時に作成された稜線に注目し、それを含む2つの三角形で形成される対面分(四辺形)の角度を判定する(図 4-18(a))。また、その注目稜線と交差する四辺形対角線を仮定し、その対角線を含む新対面分の角度も同様に判定する(図 4-18(b))。フリッピングによって最小角度が大となる場合を有効なフリッピングと呼ぶ。

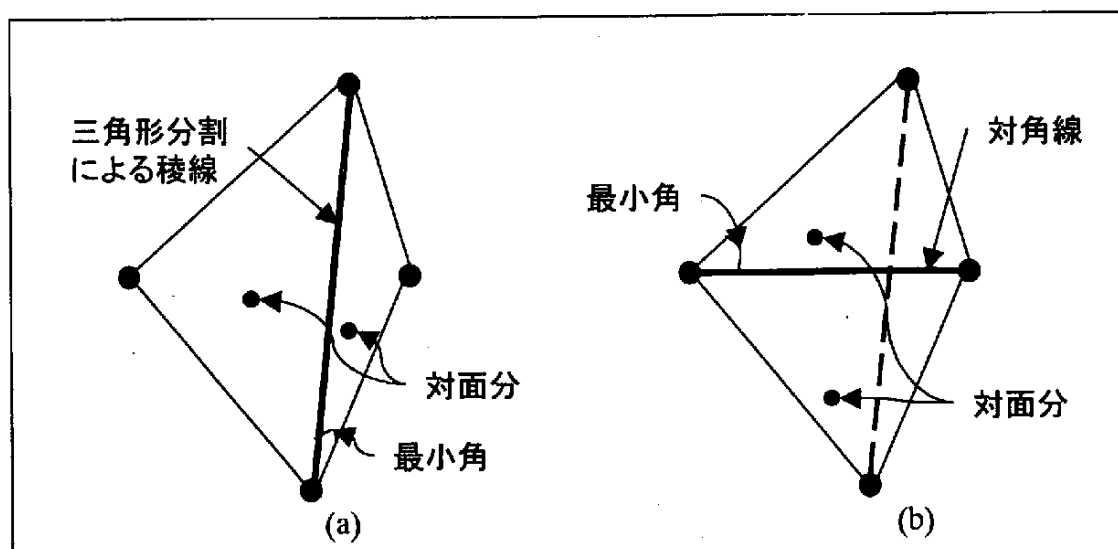


図 4-18: フリップング

任意の三角形分割に対して、有効なフリップングを次々に適用してゆくと、有効なフリップングが存在しない三角形分割に到達すること、さらに有効なフリップングが存在しない三角形分割は、**Delaunary** 三角形分割になることが示されている。この性質を利用し、有効なフリップングを繰り返すことで **Delaunary** 三角形分割を得る。

例えば、図 4-18(b) に有効なフリップングを適用することで図 4-18(a) の分割が得られる。この過程を図 4-19 に示す。

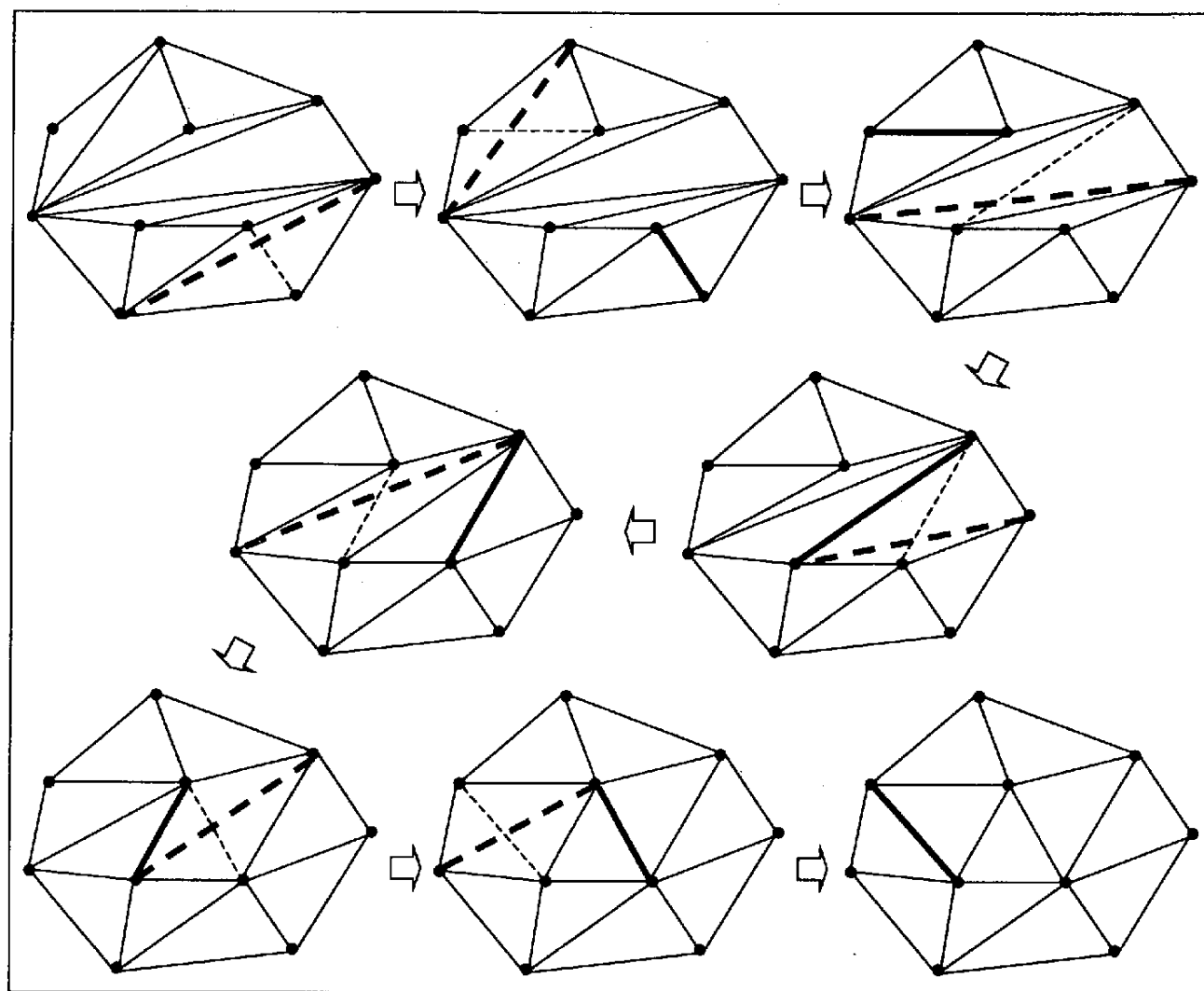


図 4-19: フリップングによる分割改良過程

5. ライブラリ使用方法

5.1. 等高線→等高線ポリゴン

この節では、等高線データから等高線ポリゴンを生成するライブラリプログラムの使用方法を示す。

(1) 等高線データの入力方法

ポリゴン化する等高線の範囲を決定するために、等高線図面上に閉曲線を設定する。この閉曲線を *B* と呼ぶ。

B の向きは、進行方向右側がポリゴン化対象範囲になるように決める。

B 内部の等高線の向きは、進行方向右側の標高が高くなるように決める。

B と *B* 内部の等高線を折れ線化する。つまり各曲線上の適当な位置に頂点を作成し、頂点同士を結ぶ線分によって曲線を近似する。生成される折れ線は以下の条件を満たさなくてはならない。

- すべての線分は、端点を除いて他の線分と交差してはならない。
- *B* と等高線の交点は、単一の頂点でなくてはならない(等高線同士は交差しない)。

図 5-1 に正しい等高線データ例を、図 5-2 に不正な等高線データ例を示す。

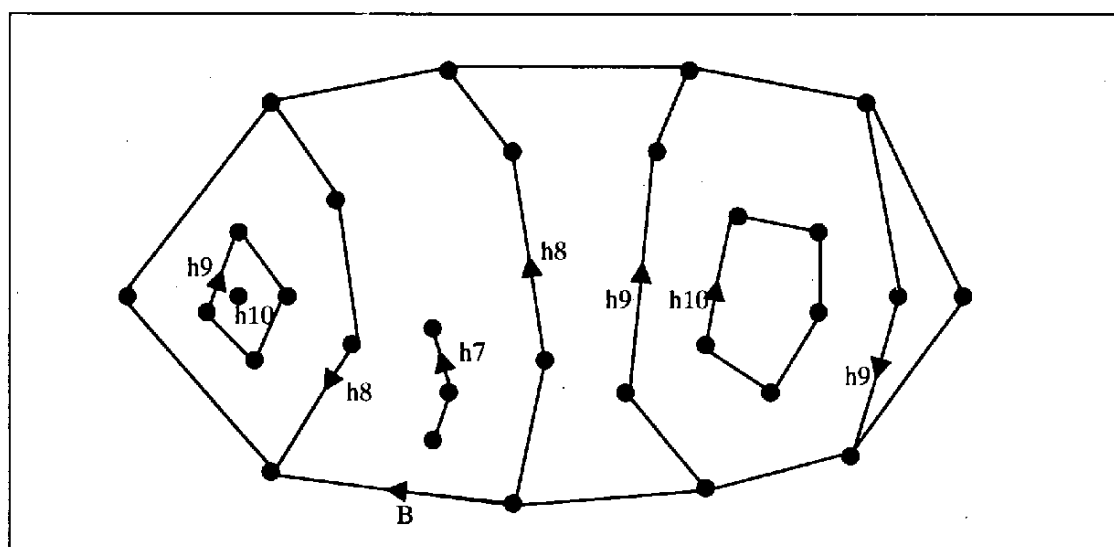


図 5-1: 正しい等高線データ例

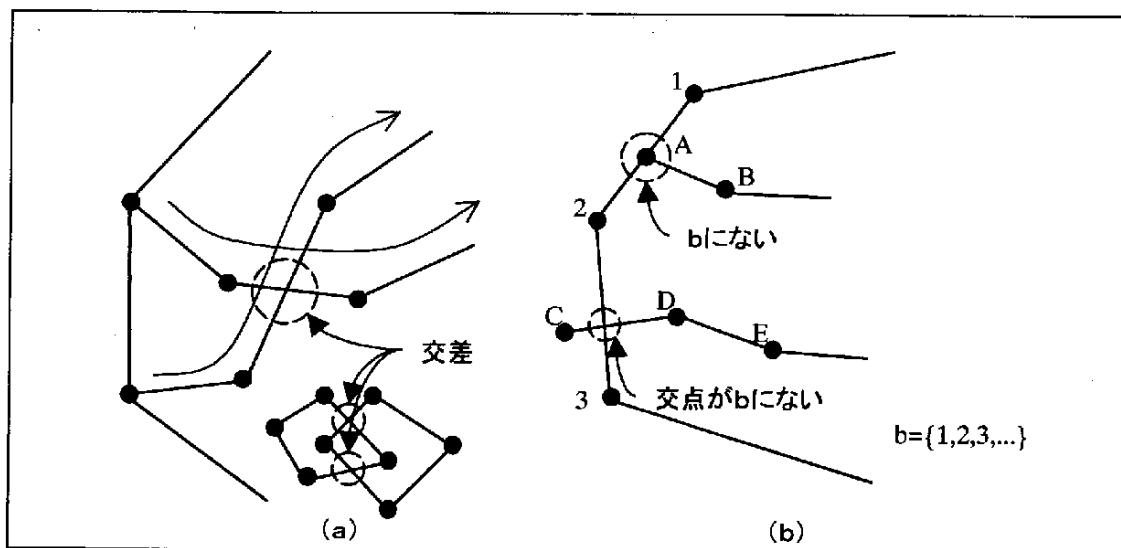


図 5-2:不正な等高線データ例

サブルーチンへの受け渡しは、頂点と、折れ線化した B と等高線は図 5-3に示す `ContourLines` クラスのインスタンスに登録し、サブルーチンへの引数とすることで行う。

```

/*****
// 地形データベース
// 折れ線による等高線
*****/
#ifndef __DB_CONTOURLINES__
#define __DB_CONTOURLINES__

#include <vector>
#include <memory>
using namespace std;

/*****
// 折れ線による等高線
*****/
class ContourLines {
public:
    //***** クラス *****
    //===== 点 =====
    class Point {
    private:
        friend ContourLines;
        Point( ContourLines* pOwner, float fX, float fY );
        // コピー代入禁止
    private:
        Point( const Point& );
        Point& operator=( const Point& );
    public:

```

```

//      位置
float    GetX()    const;
float    GetY()    const;

void      SetX( float fX );
void      SetY( float fY );

//      この Point を所有する ContourLines
ContourLines    *GetContourLinesP();
const ContourLines    *GetContourLinesP()    const;

//詳細略
};

//===== 折れ線 =====
class    Lines    {
//詳細略

private:
    friend    ContourLines;
    Lines( ContourLines *pOwner );

    Lines( ContourLines *pOwner, const vector<int>& anPointIndex );

    Lines( ContourLines *pOwner, const int *pFirst, const int *pLast );

//      コピー代入禁止
private:
    Lines( const Lines& );
    Lines&    operator=( const Lines& );

public:
//      この Point を所有する ContourLines
ContourLines    *GetContourLinesP();
const ContourLines    *GetContourLinesP()    const;

//      閉じているか?
bool    IsClose()const;
void    SetClose( bool bClose );

//      高度
float    GetHigh()    const;
bool    IsValidHigh()    const;

void    SetHigh( float fHigh );
void    SetInvalidHigh();

//      頂点参照
int    GetPointNo( int nIndex )    const;

const Point*    GetPointP( int nIndex )    const;

//      頂点数
int    GetPointNum()    const;

//      有効なデータか?
bool    IsValid()const;

//詳細略
};

private:
//***** メンバ変数 *****
//詳細略

```

```

//***** メンバ関数 *****
//===== 基本操作 =====
public:
    //      構築子
    ContourLines();

    //      コピー代入禁止
private:
    ContourLines( const ContourLines& );
    ContourLines& operator=( const ContourLines& );

//===== 操作 =====
public:
    //      クリア
    void    Clear();

    //---- 頂点 ----
    //      作成      座標(fX,fY)の頂点を作る      Ret:頂点番号(<0:作成失敗)
    int      CreatePoint( float fX, float fY );

    //      インデックス参照
    Point      *GetPointP( int nIndex );
    const Point      *GetPointP( int nIndex ) const;

    //      要素数
    int      GetPointNum() const;

    //---- 折れ線 ----
    //      作成      頂点インデックスの配列イテレータ(pPointIndexFirst,pPointIndexLast)
    //                  からなる折れ線を作成する
    //                  Ret:折れ線番号(<0:作成失敗)
    int      CreateLines(
        bool bClose,
        const int *pPointIndexFirst, const int *pPointIndexLast
    );
    int      CreateLines(
        bool bClose,
        const int *pPointIndexFirst, const int *pPointIndexLast,
        float fHigh
    );

    //      インデックス参照
    Lines      *GetLinesP( int nIndex );
    const Lines      *GetLinesP( int nIndex ) const;

    //      要素数
    int      GetLinesNum() const;

    //詳細略
};

#endif//__DB_CONTOURLINES__

```

図 5-3:ContourLines ヘッダファイル

(2) 等高線ポリゴンの出力方法

以下の特徴を持った等高線ポリゴンが出力される。

- 辺を互いに共有する三角形ポリゴン
- 閉折れ線 B 内部の任意の点に対応する三角形は必ずひとつに決まる(穴は空いていない、重なるポリゴンは存在しない)。
- 等高線を構成するすべての線分に対応する辺が存在する。辺の標高は対応する等高線辺の標高に等しい。

等高線ポリゴンは図 5-4に示す TriangleData クラスのインスタンスとして出力される。

```
/*******
//      地形データベース
//      三角形ポリゴン
/*******
#ifndef      __DB_TRIANGLEDATA__
#define      __DB_TRIANGLEDATA__

#include      <vector>
using namespace      std;

/*******
//      三角形ポリゴンデータ
/*******
class TriangleData      {

                                /******* メンバ変数      *****

private:
    //      詳細略

                                /******* メンバ関数      *****

    //=====      基本操作      =====

public:
    //      構築子
    TriangleData();

    //      消失子
    ~TriangleData();

    //=====      操作      =====

public:
    //      頂点登録 Ret:頂点番号
    int      RegisterVertex( float fX, float fY, float fZ );

    //      頂点番号 nNo 参照
    float      GetVertexX( int nNo )      const;
    float      GetVertexY( int nNo )      const;
    float      GetVertexZ( int nNo )      const;

    //      頂点数
    int      GetVertexSize()      const;
```

```

//      三角形登録      頂点番号 nVtx0,nVtx1,nVtx2を結ぶ属性 nAtr の三角形を登録する
//      Ret:三角形番号
int      RegisterTriangle( int nVtx0, int nVtx1, int nVtx2, int nAtr );

//      三角形番号 nTriNo 参照
//      第 nVtx 頂点番号 0 <= nVtx < 3
int      GetTriangleVertexNo( int nTriNo, int nVtx ) const;

int      GetTriangleAttribute( int nTriNo ) const;

//      三角形数
int      GetTriangleNum()          const;

//      詳細略
};

#endif//__DB_TRIANGLEDATA__

```

図 5-4:TriangleData ヘッドファイル

(3) サンプルプログラムと出力例

図 5-5に示した等高線データをポリゴン化するサンプルプログラムを図 5-7に示す。サンプルプログラムのビルドには `cntpol.lib` が必要である。

サンプルプログラムを実行した結果生成される等高線ポリゴンを、図 5-6に示す。

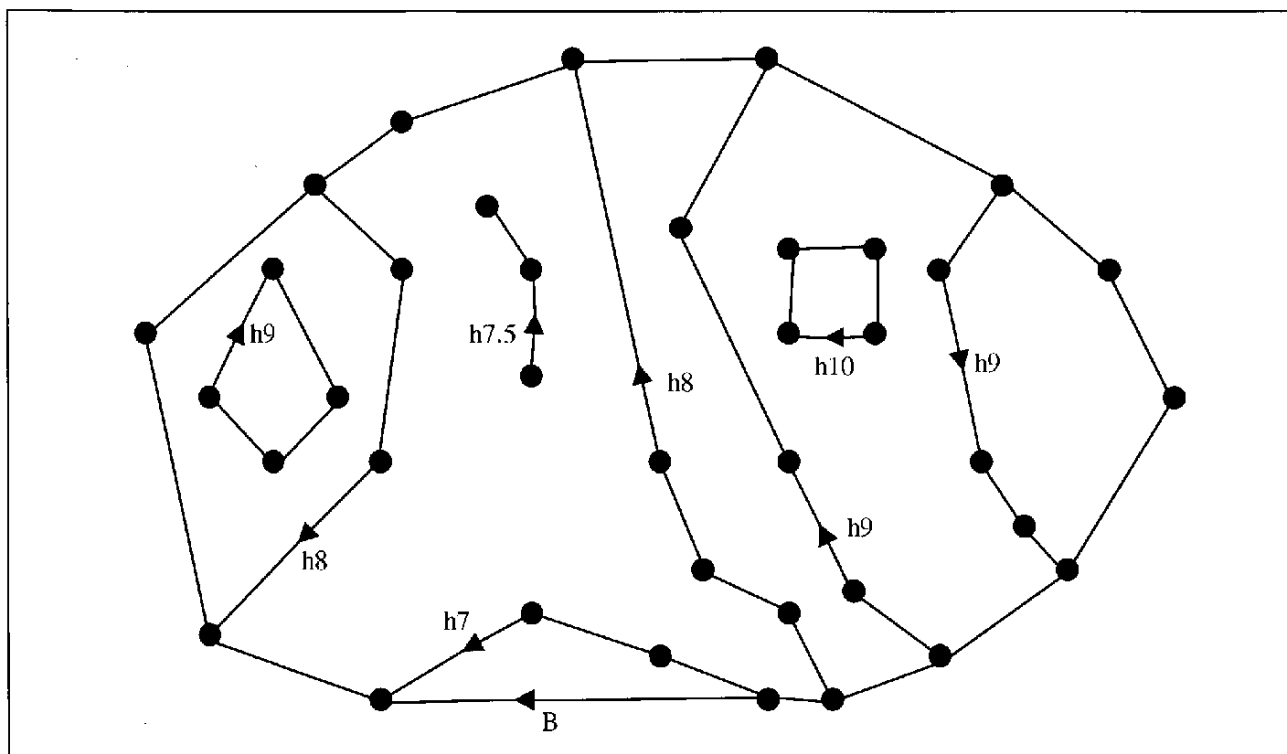


図 5-5: ポリゴン化する等高線データ

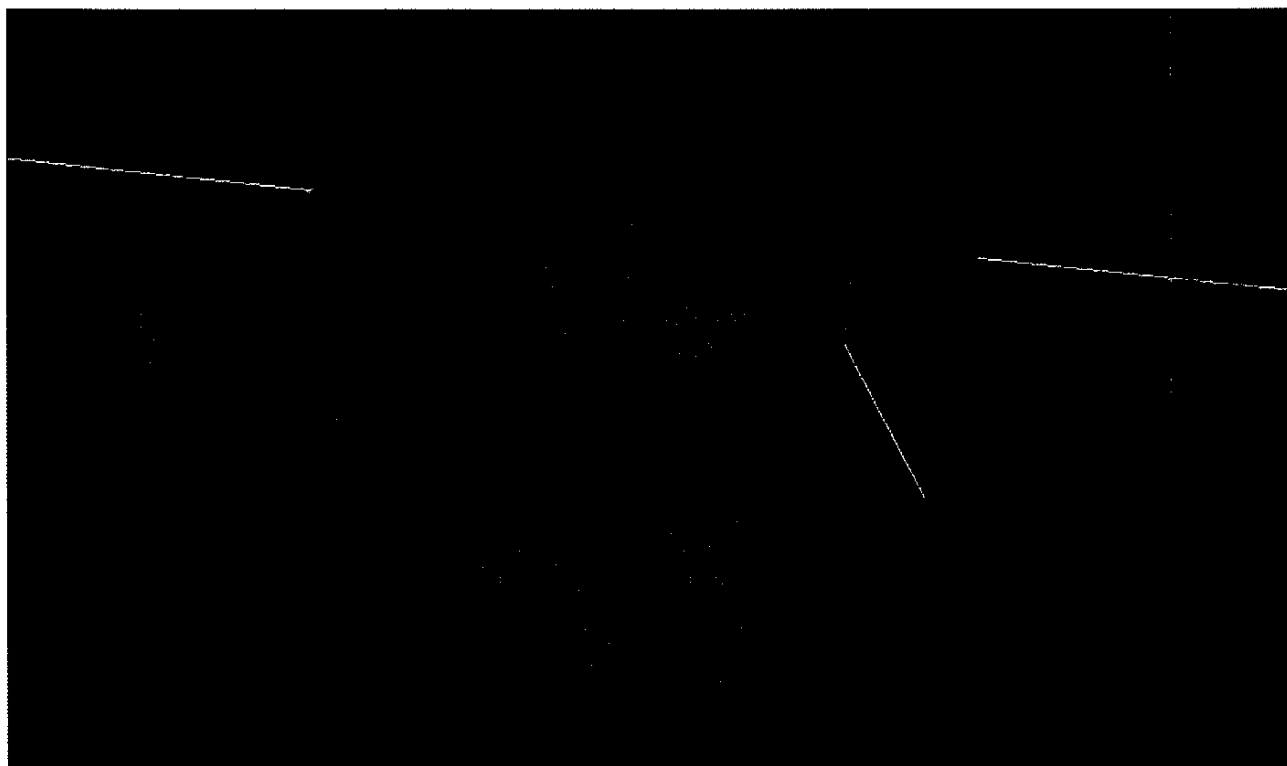


図 5-6: 生成された等高線ポリゴン

```

//*****
//  地形データベース
//  等高線から等高線ポリゴン生成サンプル
//*****
#include      "Sample1.h"
#include      "ContourToHPol.h"
#include      "ContourLines.h"
#include      "TriangleData.h"

#include      <cassert>

namespace    {

//=====
//  [pFirst,pLast)の要素 i を anIndex[i]で置き換えたものを *pan に作る
//=====
void  makeNoS(
    std::vector<int> *pan,
    const int *pFirst, const int *pLast, const std::vector<int>& anIndex
)
{
    assert( pan!=NULL );
    pan->clear();
    for ( ; pFirst!=pLast ; ++pFirst ) {
        pan->push_back( anIndex[*pFirst] );
    }
}

}; //      end of  namespace

//*****
//  等高線から等高線ポリゴン生成サンプル
//      Ret:エラーコード
//*****
int  sample1( TriangleData* pTri )
{
    assert( pTri!=NULL );

    ContourLines    cnt;

    //      頂点を登録する
    static  float    s_aafPos[][2] = {
        { 2.5, 16 }, { 3.5, 17.5 },
        { 6, 20 }, { 10.5, 19 },
        { 11.5, 16.5 }, { 11.5, 10.5 },
        { 11.5, 9.5 }, { 11, 7.5 },
        { 9.5, 5.5 }, { 7, 4 },
        { 5, 5 }, { 3.5, 6.7 },
        { 1.5, 10.5 }, { 1.5, 13.5 },
        { 5, 18 }, { 7, 19 },
        { 8, 18 }, { 7, 17 },
        { 8, 16.5 }, { 5, 16 },
        { 8, 12 }, { 10, 11.5 },
        { 10.5, 10 }, { 10.5, 14 },
        { 11, 12 }, { 4, 11.5 },
        { 8, 10 }, { 10, 9 },
        { 4.5, 10 }, { 6, 10 },
        { 6, 8.5 }, { 4.5, 8.5 },
        { 9, 6 }, { 8, 7 },
        { 5, 7.5 }, { 4, 14.5 },
        { 5, 14 }, { 6.5, 14 },
    };

    int  i;
    std::vector<int>  anVtxIdx;
    //      anVtxIdx[i] = s_aafPos[i]に対応する cnt 内の頂点番号
    for ( i=0 ; i < sizeof(s_aafPos)/sizeof(*s_aafPos) ; ++i ) {
        int  nIndex = cnt.CreatePoint( s_aafPos[i][0], s_aafPos[i][1] );
        if ( nIndex < 0 ) return(1);
    }
}

```

```

        anVtxIdx.push_back( nIndex );
    }

    // 折れ線を登録する
    int nBorderIdx;
    {
        static int s_anIdx[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        nBorderIdx = cnt.CreateLines( true, anNo.begin(), anNo.end() );
        if ( nBorderIdx < 0 ) return(1);
    }

    {
        static int s_anIdx[] = { 14, 15, 16, 17 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( cnt.CreateLines( true, anNo.begin(), anNo.end(), 9 ) < 0 ) return(1);
    }

    {
        static int s_anIdx[] = { 3, 18, 19, 1 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( cnt.CreateLines( false, anNo.begin(), anNo.end(), 8 ) < 0 ) return(1);
    }

    {
        static int s_anIdx[] = { 13, 20, 21, 22, 6 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( cnt.CreateLines( false, anNo.begin(), anNo.end(), 8 ) < 0 ) return(1);
    }

    {
        static int s_anIdx[] = { 4, 23, 24, 5 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( cnt.CreateLines( false, anNo.begin(), anNo.end(), 7 ) < 0 ) return(1);
    }

    {
        static int s_anIdx[] = { 12, 25, 26, 27, 7 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( cnt.CreateLines( false, anNo.begin(), anNo.end(), 9 ) < 0 ) return(1);
    }

    {
        static int s_anIdx[] = { 28, 29, 30, 31 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( cnt.CreateLines( true, anNo.begin(), anNo.end(), 10 ) < 0 ) return(1);
    }

    {
        static int s_anIdx[] = { 8, 32, 33, 34, 11 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( cnt.CreateLines( false, anNo.begin(), anNo.end(), 9 ) < 0 ) return(1);
    }

    {
        static int s_anIdx[] = { 35, 36, 37 };
        std::vector<int> anNo;
        makeNoS(&anNo, s_anIdx, s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( cnt.CreateLines( false, anNo.begin(), anNo.end(), 7.5 ) < 0 )
            return(1);
    }

    // 等高線ポリゴン生成
    return( ContourToHPol( pTri, cnt, nBorderIdx ) );
}

```

図 5-7: 等高線データをポリゴン化するサンプルプログラム

5.2. 植生ポリゴン + 等高線ポリゴン → 地形ポリゴン

この節では、標高データである等高線ポリゴンと、平面地形データである植生ポリゴンを合成するライブラリプログラムの使用方法を示す。

(1) 植生ポリゴンの入力方法

ポリゴン化する植生図面の範囲を決定するために、植生図面上に閉曲線を設定する。

地形図の対象領域内で使用されている植生をすべてリストアップし、すべての植生に互いに異なる非負の整数値を割り当てておく。この整数値を「属性」と呼ぶ。

地形図上の属性の境界に線をひく。植生境界線は他の境界線に接続するまで延長しておく。

境界線を折れ線化する。つまり各曲線上の適当な位置に頂点を作成し、頂点同士を結ぶ線分によって曲線を近似する。生成される折れ線は以下の条件を満たさなくてはならない。

- すべての線分は、端点を除いて他の線分と交差してはならない。
- 各境界線の接続点は、単一の頂点でなくてはならない。

境界線で区切られた領域を「面分」と呼ぶ。面分を構成する境界線を「ループ」と呼ぶ。ループの向きは、進行方向右側が面分になるように決める。面分に含まれるループの数は、面分に空いた穴の数-1 になる。

図 5-8に正しい植生ポリゴン例を、図 5-9に不正な植生ポリゴン例を示す。

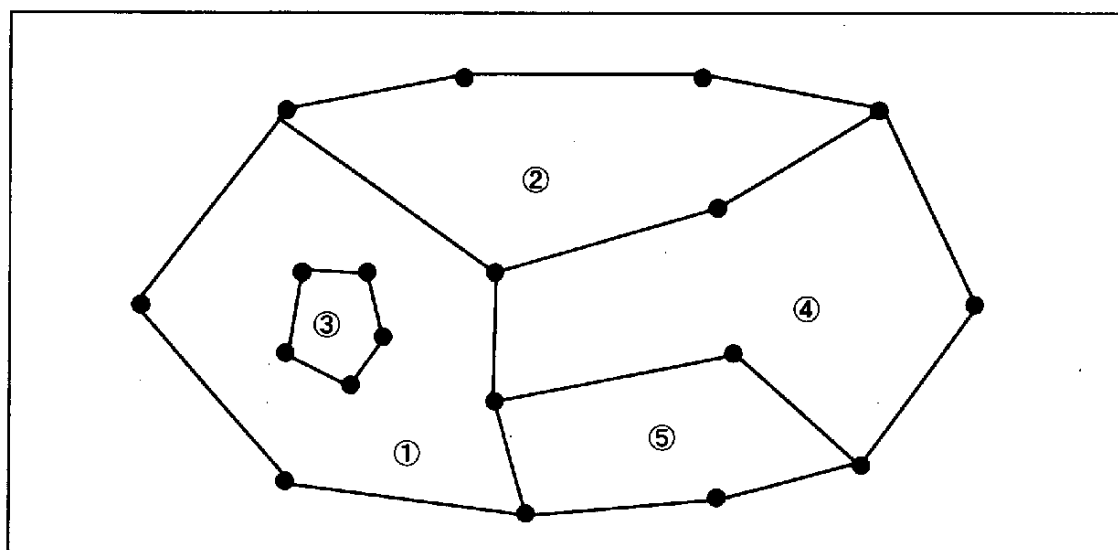


図 5-8: 正しい植生ポリゴン例

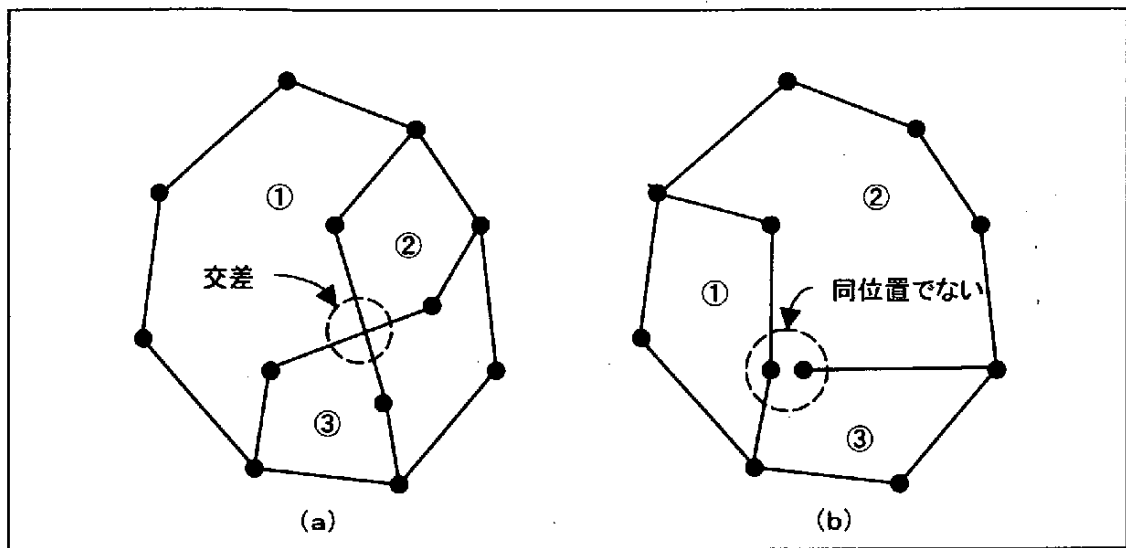


図 5-9: 不正な植生ポリゴン例

サブルーチンへの受け渡しは、頂点と、面分を図 5-10 に示す PolygonData クラスのインスタンスに登録し、サブルーチンへの引数とすることで行う。

```

/*****
// 地形データベース
// 植生データ
*****/
#ifndef __DB_POLYGONDATA_H__
#define __DB_POLYGONDATA_H__
#include <vector>
#include <memory>
using namespace std;

/*****
// 植生データ
*****/
class PolygonData {
public:
    /***** クラス *****/
    //===== 頂点 =====
    class Point {
    private:
        //詳細略
    public:
        /***** メンバ変数 *****/
        //===== 基本操作 =====
        /***** メンバ関数 *****/
        // 構築子
        Point( float fX, float fY, float fHigh );
        // コピー代入 禁止

```

```

private:
    Point( const Point& );
    Point& operator=( const Point& );

    //===== 操作 =====
public:
    //      座標
    //      (fX,fY)座標、高度 fHigh で設定
    void      SetX( float fX );
    void      SetY( float fY );
    void      SetHigh( float fHigh );

    //      (x,y)座標、高度を得る
    const float      GetX() const;
    const float      GetY() const;
    const float      GetHigh() const;

    //詳細略
};

//===== 面分 =====
class      Face      {

    //***** メンバ変数 *****

private:
    //詳細略

    //***** メンバ関数 *****
    //===== 基本操作 =====
public:
    //      構築子
    //      アトリビュート番号を nNo にする
    explicit Face( int nNo );

    //      コピー代入 禁止
private:
    Face( const Face& );
    Face& operator=( const Face& );

    //===== 操作 =====
public:
    //---- 面分のアトリビュート番号 ----
    //      アトリビュート番号を nNo に設定      nNo < 0 の場合は無効な面分になる
    void      SetAttributeNo( int nNo );

    //      アトリビュート番号を得る
    int      GetAttributeNo() const;

    //---- 面分ループ ----
    //      面分に面分ループ vLp を追加 Ret: 面分ループ番号
    //      vLp は空であってはならない。
    int RegisterLoop( const vector<int>& vLp );

    //      インデックス参照
    //      nNo 番目の面分ループを得る
    //      nNo は 0 <= nNo < 面分ループ配列の要素数
    const vector<int>* GetLoopP( int nNo ) const;

    //      面分ループ配列の要素数
    int GetLoopNum() const;

    //詳細略
};

//***** メンバ変数 *****

private:
    //詳細略

```

```

//***** メンバ関数 *****
//===== 基本操作 =====
public:
    //      構築子
    PolygonData();

    //      コピー代入 禁止
private:
    PolygonData( const PolygonData& );
    PolygonData& operator=( const PolygonData& );

//===== 操作 =====
public:
    //      クリア
    void Clear();

    //---- 頂点 ----
    //      作成      (fX,fY)座標、高度 fHigh の頂点を作成 Ret:頂点番号
    //                  作成した頂点を参照する場合は GetPointP()を使用
    int      CreatePoint( float fX, float fY, float fHigh );

    //      インデックス参照
    //                  頂点配列の nIndex 番目の頂点を得る
    //                  nIndex は 0 <= nIndex < 頂点配列の要素数
    const Point *GetPointP( int nIndex ) const;
    Point *GetPointP( int nIndex );

    //      頂点配列の要素数
    int      GetPointNum() const;

    //---- 面分 ----
    //      作成      アトリビュート番号 nNo、面分ループが空の面分を作成 Ret:面分番号
    int      CreateFace( int nNo );

    //      インデックス参照
    //                  面分配列から nIndex 番目の面分を得る
    //                  nIndex は 0 <= nIndex < 面分配列の要素数
    const Face* GetFaceP( int nIndex ) const;
    Face* GetFaceP( int nIndex );

    //      面分配列の要素数
    int      GetFaceNum() const;

    //      頂点がすべて使用されているか
    bool      IsPointUsed() const;

    //      未使用の頂点を anVtx に入れる
    void      GetUnusedPoint( vector<int>& anVtx ) const;

    //詳細略
};

#endif//__DB_POLYGONDATA_H__

```

図 5-10:PolygonData ヘッドファイル

(2) 等高線ポリゴンの入力方法

以下の条件を満たす三角形ポリゴンメッシュを等高線ポリゴンとして入力する。

- 辺を互いに共有する三角形ポリゴン
- 植生ポリゴンが存在する任意の点に対応する三角形は必ずひとつに決まる(穴は空いていない、重なるポリゴンは存在しない)。

図 5-11に正しい等高線ポリゴン例を、図 5-12に不正な等高線ポリゴン例を示す。

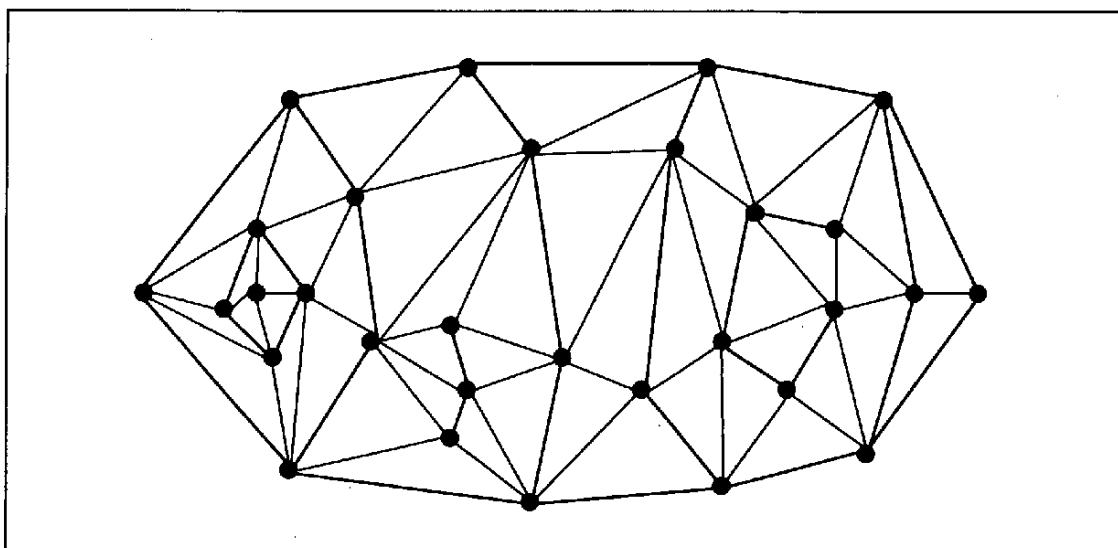


図 5-11: 正しい等高線ポリゴン例

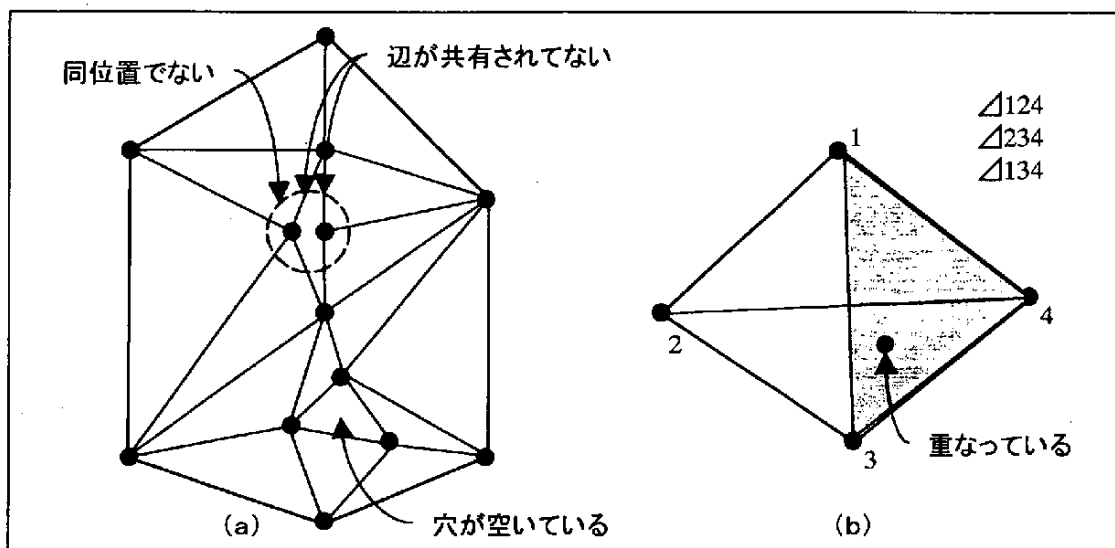


図 5-12: 不正な等高線ポリゴン例

サブルーチンへの受け渡しは、頂点と三角形ポリゴンを構成する頂点番号列を、図 5-4に示した **TriangleData** クラスのインスタンスに登録し、サブルーチンへの引数とすることで行う。

5.1で示した等高線データから等高線ポリゴンデータを生成するライブラリプログラムの出力は上の条件を満たす **TriangleData** インスタンスを出力するため、そのまま入力値として利用することができる。

(3) 地形ポリゴンの出力形式

以下の特徴を持った地形ポリゴンが出力される。

- 辺を互いに共有する三角形ポリゴン
- 植生ポリゴンが存在する任意の点に対応する三角形ポリゴンは必ずひとつに決まる(穴は空いていない、重なるポリゴンは存在しない)。
- 三角形ポリゴンの属性は、対応する植生ポリゴンの属性に等しい。
- 三角形ポリゴンの標高は、対応する等高線ポリゴンの標高に指定誤差範囲内で等しい。

等高線ポリゴンは図 5-4に示した **TriangleData** クラスのインスタンスとして出力される。

(4) サンプルプログラムと出力例

図 5-13に示した植生データと、図 5-6に示した等高線ポリゴンを合成するサンプルプログラムを図 5-15に示す。サンプルプログラムのビルドには **cmbpol.lib** が必要である。

サンプルプログラムを実行した結果生成される地形ポリゴンを、図 5-14に示す。

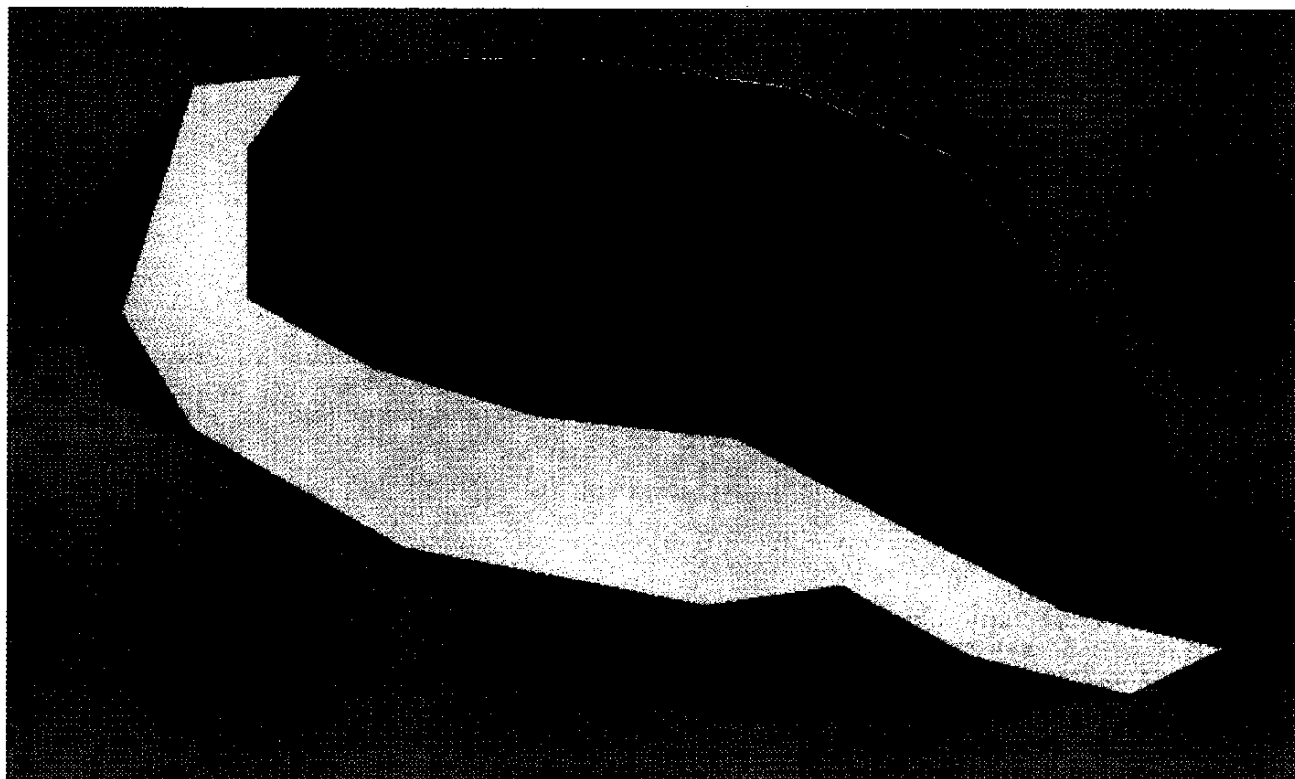


図 5-13： 植生ポリゴン

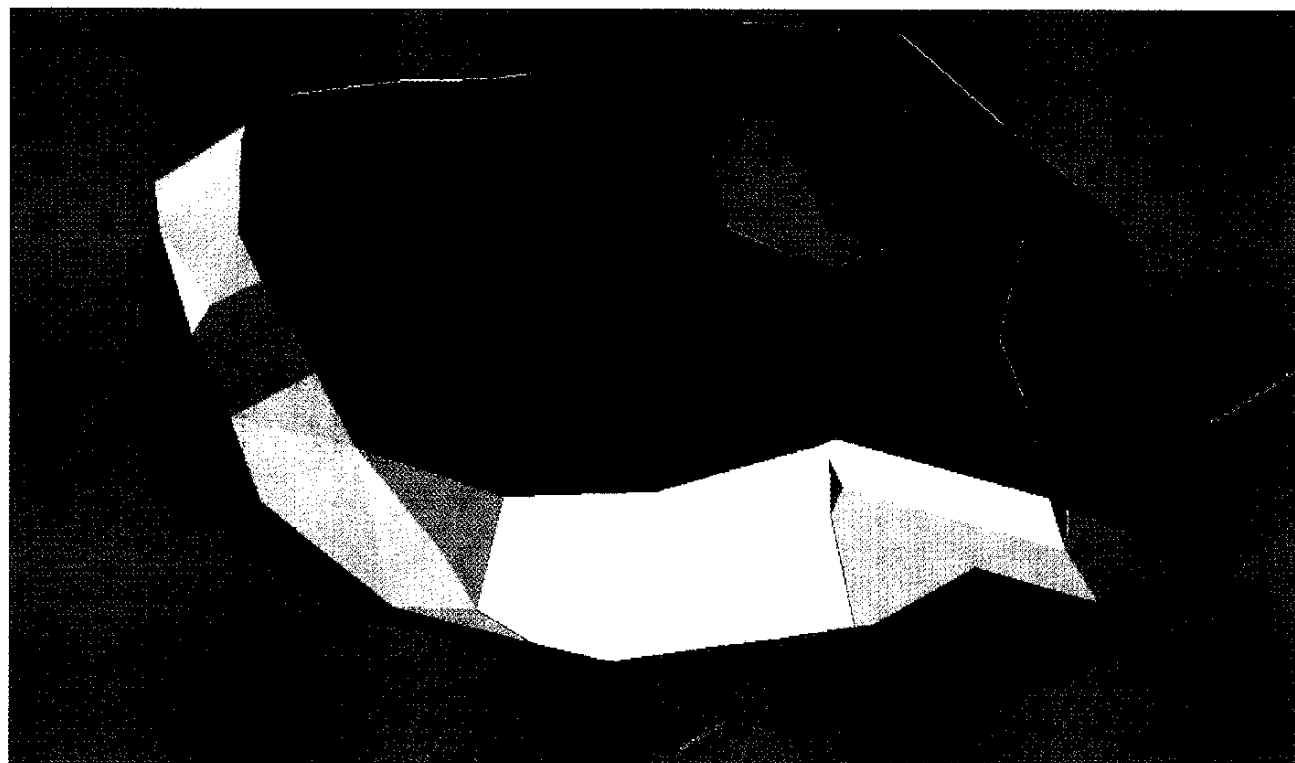


図 5-14： 合成された地形ポリゴン

```

//*****
//  地形データベース
//  植生と等高線ポリゴン合成 サンプル
//*****
#include      "Sample2.h"
#include      "CombinePolygon.h"
#include      "Sample1.h"
#include      "TriangleData.h"
#include      "PolygonData.h"

#include      <cassert>

namespace    {

//=====
//  [pFirst,pLast)の要素 i を anIndex[i]で置き換えたものを *pan に作る
//=====
void  makeNoS(
    std::vector<int> *pan,
    const int *pFirst, const int *pLast, const std::vector<int>& anIndex
)
{
    assert( pan!=NULL );
    pan->clear();
    for ( ; pFirst!=pLast ; ++pFirst ) {
        pan->push_back( anIndex[*pFirst] );
    }
}

}; //      end of  namespace


//*****
//  植生ポリゴン作成 サンプル
//      Ret:エラーコード
//*****
int  makeSampleF( PolygonData *pPol )
{
    pPol->Clear();

    static  float  //      頂点を登録する
               s_aafPos[][2] = {
    { 6.5, 19.5 }, { 10, 18.5 },
    { 11, 16 }, { 11, 10.5 },
    { 11, 9.5 }, { 10.5, 8 },
    { 9, 6 }, { 8, 5 },
    { 7, 4.5 }, { 5.5, 5.5 },
    { 4, 7 }, { 3.5, 8 },
    { 3, 9 }, { 2, 10.5 },
    { 2, 13 }, { 3, 15.5 },
    { 4, 17 }, { 5.5, 18.5 },
    { 10, 17 }, { 10.8, 15 },
    { 10.7, 12 }, { 9.5, 9 },
    { 8.5, 8 }, { 8.5, 6.5 },
    { 7, 18 }, { 9, 16 },
    { 9, 14 }, { 8.5, 12 },
    { 7.5, 10 }, { 7.5, 8 },
    { 7.5, 6 }, { 6, 17 },
    { 7.5, 16 }, { 8, 14 },
    { 7, 12 }, { 6.5, 10 },
    { 6, 9.5 }, { 5, 9 },
    { 4, 7.5 }, { 5, 8 },
    { 5.5, 9 }, { 6.5, 9 },
    { 6.5, 8 }, { 7, 7 },
    { 6.5, 6 }, { 5, 16.5 },
    { 6, 16.5 }, { 7, 15 },
    { 7, 14 }, { 6.5, 13 },
    { 5, 12.5 }, { 4, 13.5 },
    
```

```

{      3.5,    13.5  },      {      3.5,    15    },
{      4,     16    },      {      4,     12.5  },
{      5,     12    },      {      5.5,    10.5  },
{      5,     9.5   },      {      4,     9.5   },
{      3,     10.5  },      {      3,     12    }
};

int i;
std::vector<int> anVtxIdx; //anVtxIdx[i] = s_aafPos[i]に対応する cnt 内の頂点番号
for ( i=0 ; i < sizeof(s_aafPos)/sizeof(*s_aafPos) ; ++i ) {
    int nIndex = pPol->CreatePoint( s_aafPos[i][0], s_aafPos[i][1], 0 );
    if ( nIndex < 0 ) return(1); // 失敗
    anVtxIdx.push_back( nIndex );
}

// 面を登録する

{
    int nFace = pPol->CreateFace( 1 );
    if ( nFace < 0 ) return(1); // 失敗
    PolygonData::Face *pFace = pPol->GetFaceP(nFace);
    {
        static int s_anIdx[] = { 31,32,33,34,35,36,37,12,13,14,15,16 };
        std::vector<int> anNo;
        makeNoS(
            &anNo, s_anIdx,
            s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
    }
    {
        static int s_anIdx[] = { 54,53,52,51,50,49,48,47,46,45 };
        std::vector<int> anNo;
        makeNoS(
            &anNo, s_anIdx,
            s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
    }
    {
        static int s_anIdx[] = { 61,60,59,58,57,56,55 };
        std::vector<int> anNo;
        makeNoS(
            &anNo, s_anIdx,
            s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
    }
}

{
    int nFace = pPol->CreateFace( 2 );
    if ( nFace < 0 ) return(1); // 失敗
    PolygonData::Face *pFace = pPol->GetFaceP(nFace);
    {
        static int s_anIdx[] = { 45,46,47,48,49,50,51,52,53,54 };
        std::vector<int> anNo;
        makeNoS(
            &anNo, s_anIdx,
            s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
    }
}

{
    int nFace = pPol->CreateFace( 3 );
    if ( nFace < 0 ) return(1); // 失敗
    PolygonData::Face *pFace = pPol->GetFaceP(nFace);
    {
        static int s_anIdx[] = { 55,56,57,58,59,60,61 };
        std::vector<int> anNo;
        makeNoS(
            &anNo, s_anIdx,
            s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
        if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
    }
}

```

```

    }
    {
        int nFace = pPol->CreateFace( 4 );
        if ( nFace < 0 ) return(1); // 失敗
        PolygonData::Face *pFace = pPol->GetFaceP(nFace);
        {
            static int s_anIdx[] = {
                17,24,25,26,27,28,29,30,8,9,44,43,42,41,40,39,
                38,10,11,12,37,36,35,34,33,32,31,16
            };
            std::vector<int> anNo;
            makeNoS(
                &anNo, s_anIdx,
                s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
            if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
        }
    }
    {
        int nFace = pPol->CreateFace( 5 );
        if ( nFace < 0 ) return(1); // 失敗
        PolygonData::Face *pFace = pPol->GetFaceP(nFace);
        {
            static int s_anIdx[] = { 10,38,39,40,41,42,43,44,9 };
            std::vector<int> anNo;
            makeNoS(
                &anNo, s_anIdx,
                s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
            if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
        }
    }
    {
        int nFace = pPol->CreateFace( 6 );
        if ( nFace < 0 ) return(1); // 失敗
        PolygonData::Face *pFace = pPol->GetFaceP(nFace);
        {
            static int s_anIdx[] = {
                0,18,19,20,21,22,23,7,8,30,29,28,27,26,25,24,17
            };
            std::vector<int> anNo;
            makeNoS(
                &anNo, s_anIdx,
                s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
            if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
        }
    }
    {
        int nFace = pPol->CreateFace( 7 );
        if ( nFace < 0 ) return(1); // 失敗
        PolygonData::Face *pFace = pPol->GetFaceP(nFace);
        {
            static int s_anIdx[] = { 0,1,2,3,4,5,6,7,23,22,21,20,19,18
            };
            std::vector<int> anNo;
            makeNoS(
                &anNo, s_anIdx,
                s_anIdx+sizeof(s_anIdx)/sizeof(*s_anIdx), anVtxIdx );
            if ( pFace->RegisterLoop( anNo ) < 0 ) return(1);
        }
    }
    return( 0 ); // 成功
}

```

```

//*****
// 植生と等高線ポリゴンを許容高度誤差 fHighEps 以内で合成するサンプル
// Ret:エラーコード
//*****
int sample2( TriangleData* pTri, float fHighEps )
{

```

```

//      植生ポリゴン作成
PolygonData    polF;
if ( makeSampleF(&polF)!=0 )    return(1);    //      失敗

//      等高線ポリゴン作成
TriangleData    triGround;
if ( sample1( &triGround )!=0 )    return(1);    //      失敗

//      合成
return( CombinePolygon( pTri, polF, triGround, fHighEps ) );
}

```

図 5-15: 植生,等高線ポリゴン合成サンプルプログラム

5.3.植生ポリゴン+標高メッシュ→地形ポリゴン

(1) 植生ポリゴンの入力方法

5.2と同様の方法で、植生ポリゴンを入力する。

(2) 標高メッシュの入力方法

地平面を X,Y 座標軸方向に等間隔に分割して得られた各矩形(セル)の平均高度を、二次元配列で表現したものを標高メッシュとして入力する。

二次元配列を図 5-16に示す MeshData クラスのインスタンスに登録する。

```
//*****
//    地形データベース
//    メッシュ標高
//*****
#ifndef __DB_MESHDATA__
#define __DB_MESHDATA__

#include <vector>
using namespace std;

//*****
//    標高メッシュデータ
//*****
class MeshData
{
    //***** メンバ変数 *****
private:
    //    詳細略

    //***** メンバ関数 *****
    //===== 基本操作 =====
public:
    //    構築子
    MeshData();

    //    消失子
    ~MeshData();

    //===== 操作 =====
public:
    //    始点    添え字(0,0)のセルの中心に対応する座標
    void    SetOrigin( float fOrgX, float fOrgY );
    float    GetOrigionX()    const;
    float    GetOrigionY()    const;

    //    メッシュ間隔
    //    メッシュ間隔設定    Ret:エラーコード
    int    SetInterval( float fltvX, float fltvY );

    float    GetIntervalX()    const;
```

```

float    GetIntervalY()    const;

//      セル添え字の有効範囲 0<=nX<GetSizeX() && 0<=nY<GetSizeY()の(nX,nY)が有効
//      セル添え字の有効範囲設定
//      拡大の場合は新しく作られるセルの標高は fInitHigh になる
void    SetSize( int nSizeX, int nSizeY, float fInitHigh =0 );

int      GetSizeX()        const;
int      GetSizeY()        const;

//      セル中心点の座標
void    GetCellCenterCoord(
                                float *pfX, float *pfY, float *pfZ,
                                int nX, int nY, float fInvalidHigh =0
                                )    const;

//      有効な添え字か?
bool    IsValidIndex( int nX, int nY )    const;

//      標高      添え字(nX,nY)のセルの標高
//      標高設定 Ret:エラーコード
int      SetHigh( int nX, int nY, float fHigh );

//      標高取得 正当な添え字でない場合は fInvalidHigh が返る
float    GetHigh( int nX, int nY, float fInvalidHigh =0 )    const;

//詳細略
};

#endif//__DB_MESHDATA__

```

図 5-16: MeshData ヘッダファイル

(3) 地形ポリゴンの出力形式

5.2と同様の形式の地形ポリゴンが出力される。

(4) サンプルプログラムと出力例

図 5-13に示した植生データと、図 5-17に示した標高メッシュデータをポリゴン化するサンプルプログラムを図 5-19に示す。サンプルプログラムのビルドには `cmbmesh.lib` が必要である。

サンプルプログラムを実行した結果合成される地形ポリゴンを、図 5-18に示す。

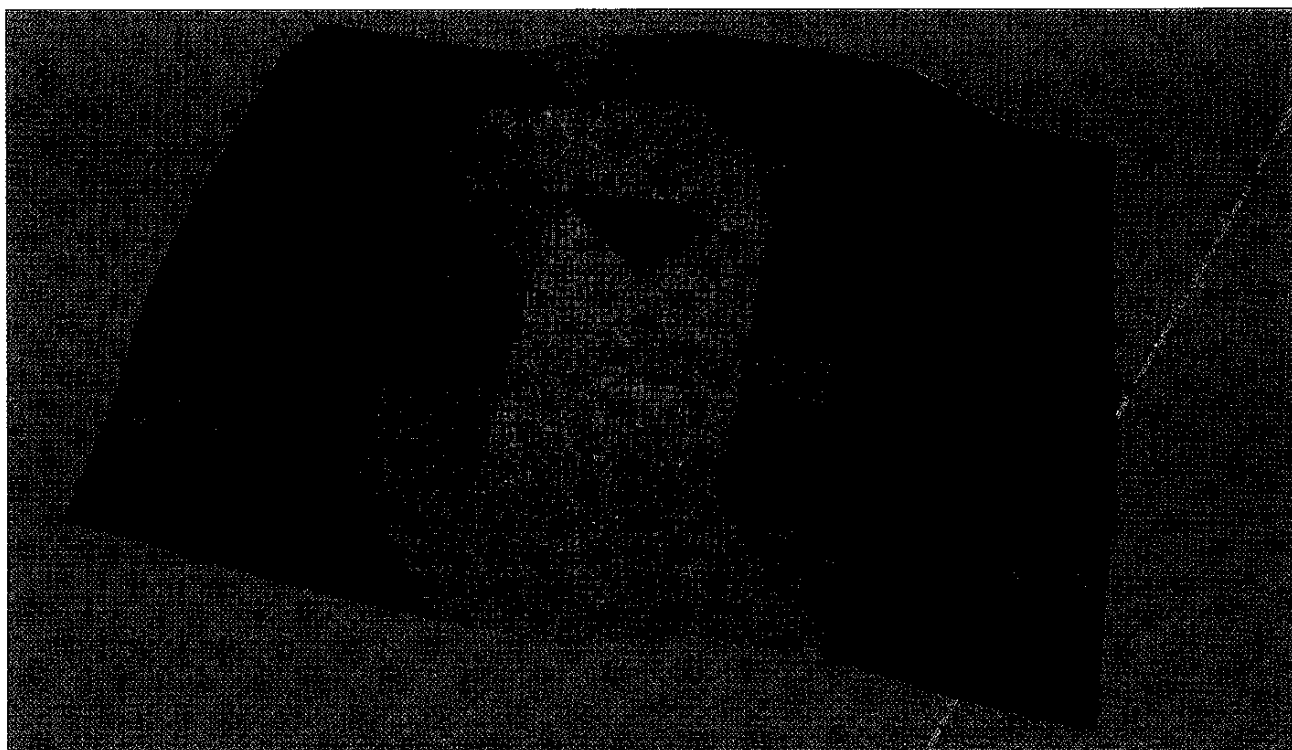


図 5-17： 標高メッシュ



図 5-18: 合成された地形ポリゴン

```

//*****
//    地形データベース
//    植生と標高メッシュ合成サンプル
//*****
#include    "Sample3.h"
#include    "CombineMesh.h"
#include    "PolygonData.h"
#include    "TriangleData.h"
#include    "MeshData.h"
#include    "Sample2.h"

//*****
//    植生と標高メッシュを許容高度誤差fHighEps 以内に合成するサンプル
//    Ret: エラーコード
//*****
int
sample3( TriangleData* pTri, float fHighEps )
{
    //            植生ポリゴン作成
    PolygonData    polF;
    if ( makeSampleF(&polF)!=0 )        return(1);        //            失敗

    //            標高メッシュ作成
    MeshData        mesh;
    {
        mesh.SetSize( 7, 9 );
        mesh.SetInterval( 2.2, 2.2 );
        mesh.SetOrigin( 0,3 );

        mesh.SetHigh( 0,0, 7 );    mesh.SetHigh( 1,0, 7.3 );    mesh.SetHigh( 2,0, 7.5 );
        mesh.SetHigh( 3,0, 8 );    mesh.SetHigh( 4,0, 8.3 );    mesh.SetHigh( 5,0, 8 );
        mesh.SetHigh( 6,0, 7.8 );

        mesh.SetHigh( 0,1, 7.2 );    mesh.SetHigh( 1,1, 8 );    mesh.SetHigh( 2,1, 8 );
    }
}

```

```

mesh.SetHigh( 3,1, 8.14 ); mesh.SetHigh( 4,1, 8.5 ); mesh.SetHigh( 5,1, 8 );
mesh.SetHigh( 6,1, 8 );

mesh.SetHigh( 0,2, 8.3 ); mesh.SetHigh( 1,2, 9 ); mesh.SetHigh( 2,2, 9.02 );
mesh.SetHigh( 3,2, 8.87 ); mesh.SetHigh( 4,2, 9 ); mesh.SetHigh( 5,2, 9 );
mesh.SetHigh( 6,2, 8.7 );

mesh.SetHigh( 0,3, 8.5 ); mesh.SetHigh( 1,3, 9 ); mesh.SetHigh( 2,3, 9.88 );
mesh.SetHigh( 3,3, 10 ); mesh.SetHigh( 4,3, 9 ); mesh.SetHigh( 5,3, 9 );
mesh.SetHigh( 6,3, 8.5 );

mesh.SetHigh( 0,4, 8.5 ); mesh.SetHigh( 1,4, 8.9 ); mesh.SetHigh( 2,4, 8.3 );
mesh.SetHigh( 3,4, 8.88 ); mesh.SetHigh( 4,4, 8.5 ); mesh.SetHigh( 5,4, 8 );
mesh.SetHigh( 6,4, 7 );

mesh.SetHigh( 0,5, 8 ); mesh.SetHigh( 1,5, 8.23 ); mesh.SetHigh( 2,5, 7.97 );
mesh.SetHigh( 3,5, 7.79 ); mesh.SetHigh( 4,5, 7.66 ); mesh.SetHigh( 5,5, 7.43 );
mesh.SetHigh( 6,5, 6.8 );

mesh.SetHigh( 0,6, 7 ); mesh.SetHigh( 1,6, 7.5 ); mesh.SetHigh( 2,6, 7.55 );
mesh.SetHigh( 3,6, 7.74 ); mesh.SetHigh( 4,6, 7.59 ); mesh.SetHigh( 5,6, 7.26 );
mesh.SetHigh( 6,6, 7 );

mesh.SetHigh( 0,7, 7 ); mesh.SetHigh( 1,7, 7.5 ); mesh.SetHigh( 2,7, 8 );
mesh.SetHigh( 3,7, 8.75 ); mesh.SetHigh( 4,7, 8.33 ); mesh.SetHigh( 5,7, 8.14 );
mesh.SetHigh( 6,7, 7.5 );

mesh.SetHigh( 0,8, 7 ); mesh.SetHigh( 1,8, 8 ); mesh.SetHigh( 2,8, 8.3 );
mesh.SetHigh( 3,8, 8.67 ); mesh.SetHigh( 4,8, 8.71 ); mesh.SetHigh( 5,8, 8.14 );
mesh.SetHigh( 6,8, 8 );

mesh.SetHigh( 0,9, 7 ); mesh.SetHigh( 1,9, 7.5 ); mesh.SetHigh( 2,9, 8 );
mesh.SetHigh( 3,9, 8.5 ); mesh.SetHigh( 4,9, 8.5 ); mesh.SetHigh( 5,9, 8 );
mesh.SetHigh( 6,9, 7.6 );
}

// 合成
return( CombineMesh( pTri, polF, mesh, fHighEps ) );
}

```

図 5-19: 植生,標高メッシュ合成サンプルプログラム

6. ポリゴン形式の評価

メッシュ形式と比較したポリゴン形式の利点を検証する。

(1) 手段

同一の地形情報をポリゴン形式とメッシュ形式で表現し、メモリ容量と表現能力の観点から両者の比較を行う。

地形情報は比較的高低差があり、複雑な植生形状を持っている湖周辺とする。

ポリゴン形式は本開発のライブラリを用いて作成を行う。

メッシュ形式は以下の基準で選んだメッシュ間隔で作成を行う。

「粗」メッシュ地形データ

メモリ容量がポリゴン形式と同等になるメッシュ間隔。

「中」メッシュ地形データ

比較参考のための「粗」と「細」の間のメッシュ間隔。

「細」メッシュ地形データ

ポリゴン形式で表現できていた要素を、ほぼ表現できる最大のメッシュ間隔。道路や川を周囲の植生情報と区別できるメッシュ間隔を選ぶ。

(2) 結果

作成したポリゴン地形データと、メッシュ地形データの立体図を図6-2～図6-5に示す。各メッシュ地形データのメッシュ間隔を表6-1に示す。

表 6-1: データとメッシュ間隔

データ名	メッシュ間隔
「粗」メッシュ地形データ	50mメッシュ間隔
「中」メッシュ地形データ	25mメッシュ間隔
「細」メッシュ地形データ	12.5mメッシュ間隔

各地形データのメモリ容量とメモリ容量の比を表6-2と図6-1に示す。

メモリ容量算出の計算方法は以下の通り。

・ポリゴン形式のメモリ容量

頂点数 × 1頂点のメモリ容量 + ポリゴン数 × 1ポリゴンのメモリ容量 = メモリ容量

・メッシュ形式のメモリ容量

メッシュ数 × 1メッシュのメモリ容量 = メモリ容量

表 6-2: 形式による要求メモリ容量

データ形式		メモリ容量				
ポリゴン		頂点		ポリゴン		合計[byte]
		数	メモリ容量[byte]	数	メモリ容量[byte]	
		721	12	1352	8	19468
メッシュ	間隔[m]	メッシュ		ポリゴン		ポリゴンとの比
		数	メモリ容量[byte]	数	メモリ容量[byte]	
	12.5			39872	6	239232
	25.0			9836	6	59016
	50.0			2402	6	14412

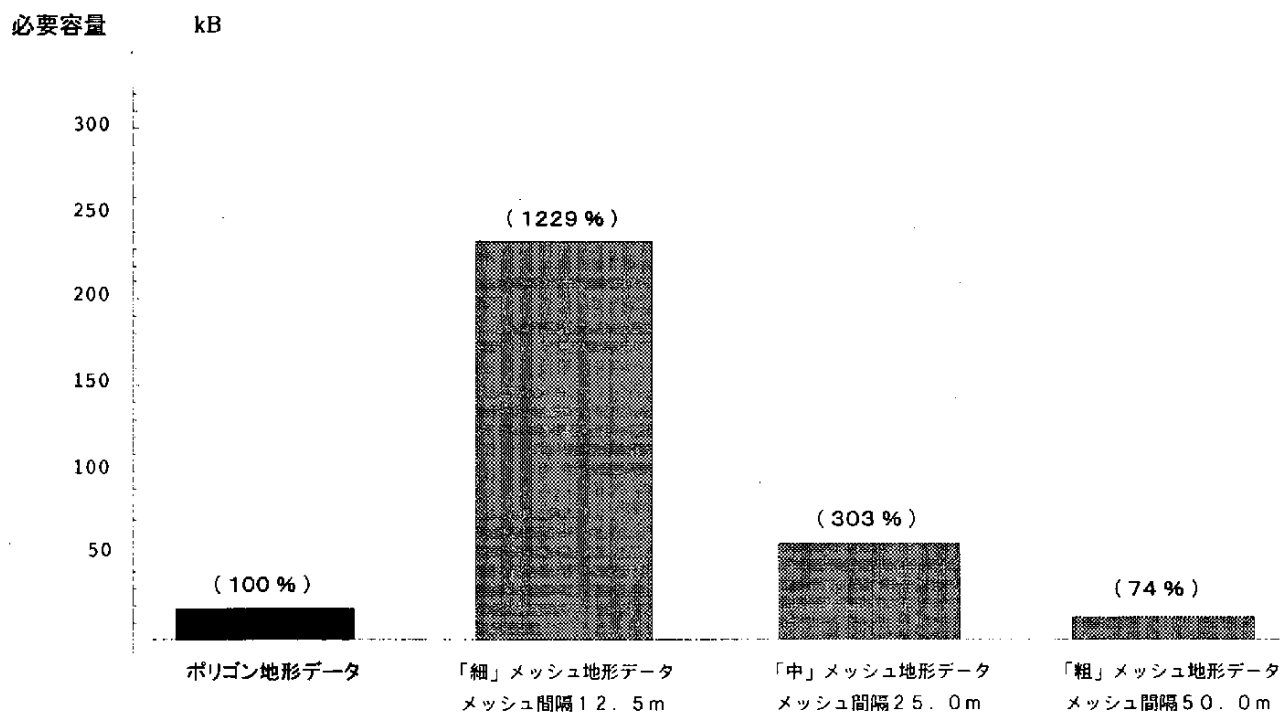


図 6-1: 形式による要求メモリ容量

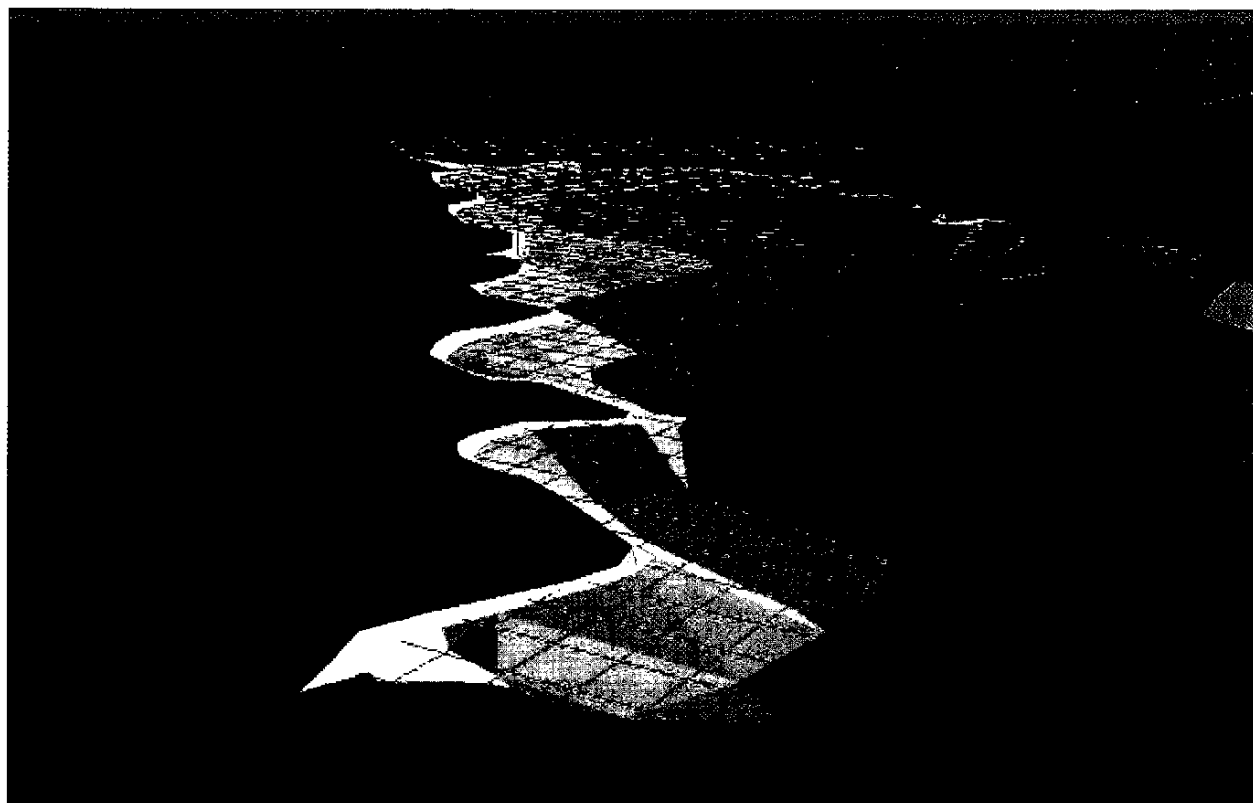


図 6-2：ポリゴン地形データ

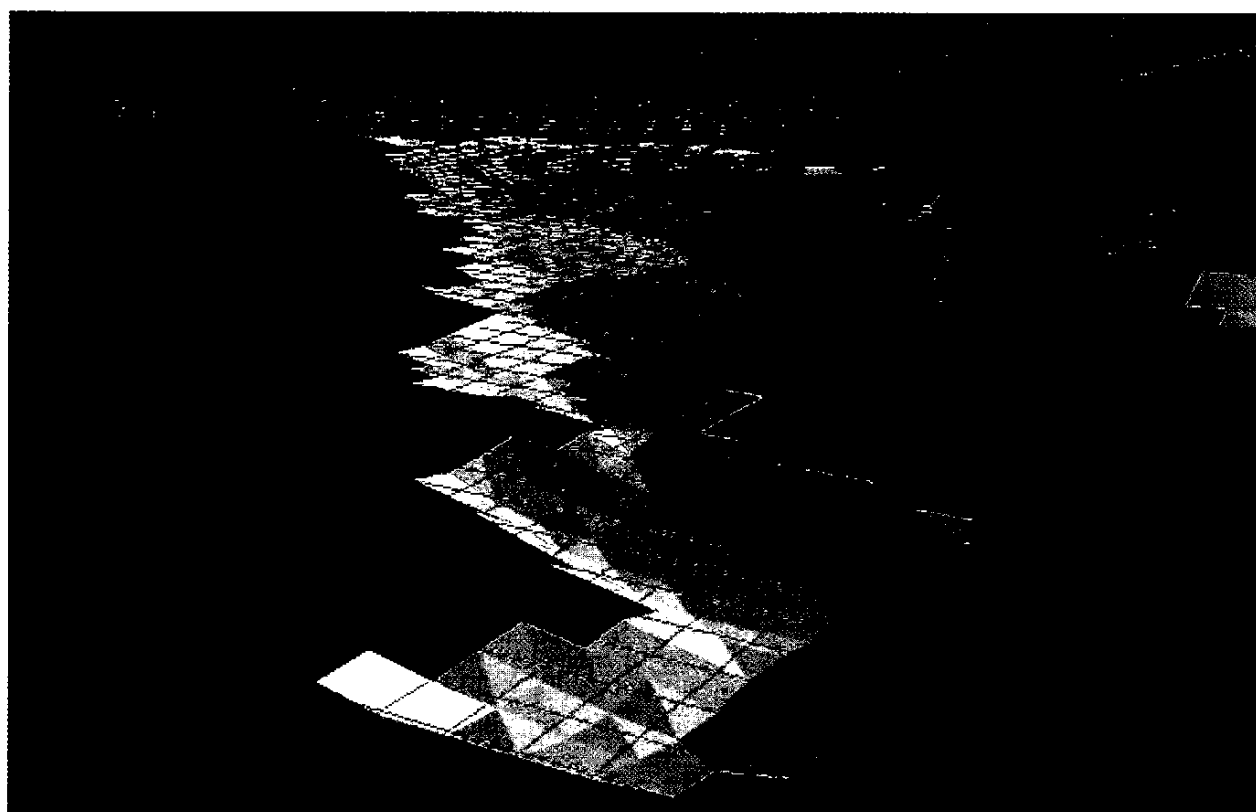


図 6-3：「粗」メッシュ地形データ(メッシュ間隔50m)

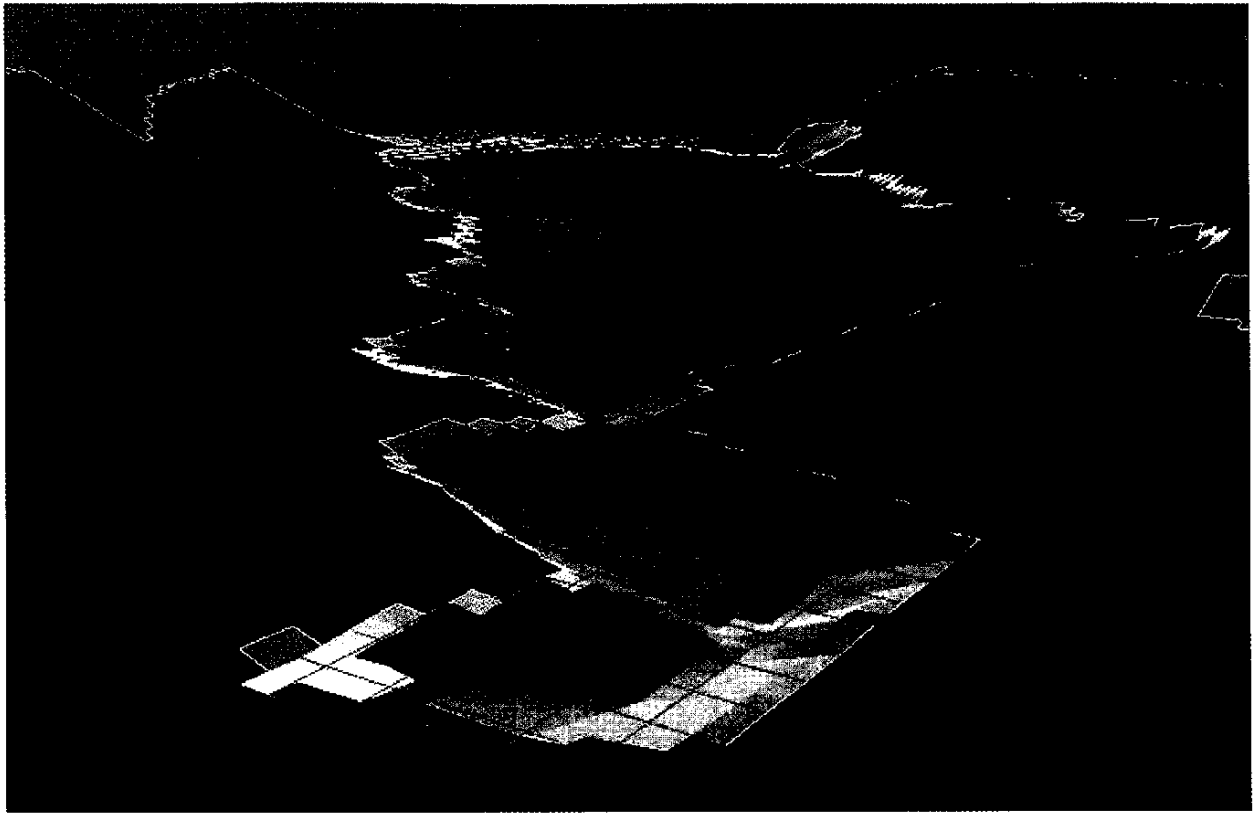


図 6-4：「中」メッシュ地形データ(メッシュ間隔25m)

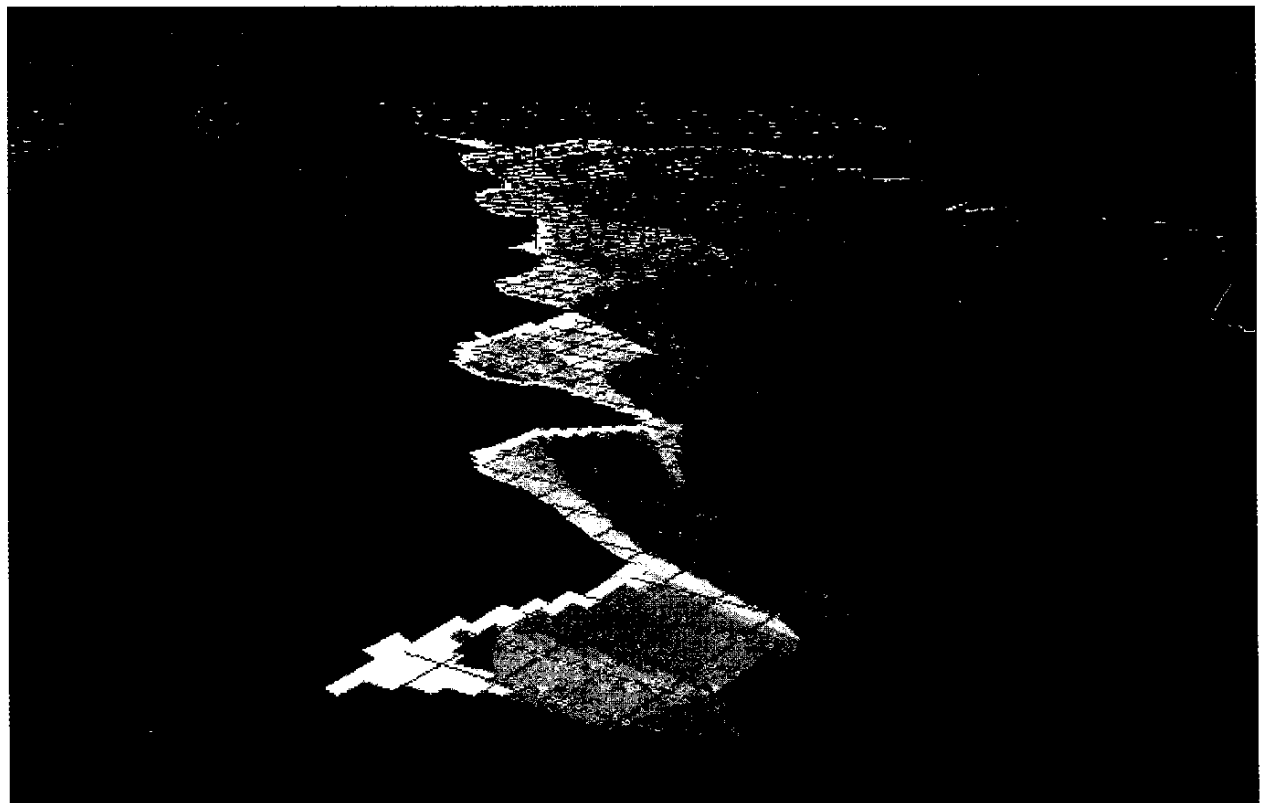


図 6-5：「細」メッシュ地形データ(メッシュ間隔12.5m)

(3) 考 察

結果からメッシュ形式と比較したポリゴン形式の利点を考察する。

ポリゴン形式(図6-2)では立体図左下の川が表現されている。しかし、同等のメモリ容量のメッシュ形式(図6-3 メッシュ間隔50m)では表現されていない。同等なメモリ容量ではポリゴン形式の方が表現能力に優れているといえる。

また、立体図左下の川が途切れないようにメッシュ形式のメッシュ間隔を決める(図6-5 メッシュ間隔12.5m)と、メモリ容量がポリゴン形式に比べて10倍以上大きくなってしまう(図6-1)。同等な表現能力ではポリゴン形式の方がメモリ容量を抑えることができる。

さらに、より高低差の小さい市街地や、単純な植生形状の地形情報においてメモリ容量の差がさらに大きくなると考えられる。ポリゴン形式ではこの場合、広範囲を少ないポリゴンと頂点で表現できるのでメモリ容量が少なくなるが、メッシュ形式は定められた範囲に対して地形情報を記録しているのでメモリ容量が変化することはない。

以上からポリゴン形式は従来のメッシュ形式にくらべて、メモリ容量を節約しつつ高品位の地形情報をデータ化できると言える。

7. まとめ

立体地形情報のポリゴン形式による表現を広く利用可能にするために、地形図に記載された地形情報をポリゴン形式に効率よく変換するための基礎技術開発を行った。

効率よく地形情報をポリゴンデータ化する手法を検討した。標高データと植生データを独立に生成し、両者を合成することで立体地形ポリゴンデータを得る手法を提案し、以下のライブラリプログラムを開発した。

- 等高線データから等高線ポリゴンデータを作成する。
- 植生ポリゴンデータと等高線ポリゴンデータから立体地形ポリゴンデータを作成する。
- 植生ポリゴンデータと標高メッシュデータから立体地形ポリゴンデータを作成する。

提案したデータ形式の有効性を検証するために、同一地形図をポリゴン形式とメッシュ形式でデータ化し比較検討した。その結果、採用したポリゴン形式のほうがデータ容量を節約しつつ高品位に地形データを電子データ化できることを示した。

— 禁無断転載 —

平成12年3月発行

発行 財団法人 データベース振興センター
東京都港区新橋2丁目13番8号
新橋東和ビル5F
TEL 03-3508-2430

委託先 株式会社セタ
東京都江東区有明3丁目1番25号
有明フロンティアビルB棟
TEL 03-3599-2562

印刷所 有限会社 三愛システム 大明社
東京都 大田区 西蒲田 7丁目5番3号
TEL 03-3733-2531

