修 士 論 文

画像の境界値表現と
その応用における省メモリアルゴリズム

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

今野賢

2013 年 9 月

修 士 論 文

画像の境界値表現と
その応用における省メモリアルゴリズム

指導教官　浅野哲夫 教授

審査委員主査　浅野哲夫 教授
審査委員　平石邦彦 教授
審査委員　緒方和博 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

0910752 今野賢

提出年月: 2013 年 8 月

概 要

近年，省メモリアルゴリズムの重要性が増々高まっている．省メモリアルゴリズムとは限定的な作業メモリ量内で効率的な演算アルゴリズムについての総称である．

かつてのムーアの法則が支配的であった時代は過ぎ去り，コンピュータ性能およびメモリ容量の劇的な向上はもはや期待できない．最近のスマートフォンなどの組み込み機器についてはサプチップ搭載による数値演算能力の向上は顕著である．しかしながら依然として搭載されるメモリ容量にかなりの制限がある．またサーバサイドでも，ビックデータに代表される入力データ数の指数関数的に爆発的な要求に対して，単体サーバーのメモリ容量は有限であり十分ではない．

本稿では，省メモリアルゴリズムの題材として，近年のスマートフォンアプリで需要の高い画像認識分野で，境界値表現検出を取り扱った．本稿で取り扱う境界値表現検出とは，入力画像の輪郭を追跡し検出結果として複数の閉じた多角形集合を出力する処理を指す．

最初に既存の境界値表現検出アルゴリズム [8] を検証し，その結果を踏まえて改良アルゴリズムの考案および検証を行った．既存の境界値表現検出アルゴリズムは単純に実装すると処理の分岐が多く煩雑であった．また実装段階で条件分岐処理が固定化されるため入力画像の探索方向成分分布により偏った実行速度となる懸念もあった．本稿では既存の省メモリアルゴリズムの特徴を更に押し進め，並列コンピューティング環境への適用も念頭に置いた改良アルゴリズムの考案および検証を行った．

また最適な検出結果を得るために入力画像に対する前処理手法，検出後に出力された境界値表現の簡素化についても考案および検証を行った．

目 次

第1章　はじめに

1.1　背景

1.1.1　省メモリアルゴリズムを取り巻く環境

近年，限られたメモリ領域内で効率的に稼働する省メモリアルゴリズムの重要性が増々高まっている．メモリ容量と処理性能はコンピュータ登場時から 2005 年頃まではムーアの法則 [1] の将来予測通りに年々劇的に向上してきた．この背景から省メモリアルゴリズムは組み込み機器などの限られた一部プラットフォーム以外を除いて，一般的には脚光を浴びず衰退していたとも言える技法であった [2].

時代は変わり，近年ではコンピュータ集積回路の微細化によるクロック向上も限界に近づきつつあり，処理性能向上も既に鈍化傾向である．ムーアの法則についてもその限界が予測されており，以前のような性能およびメモリ容量の劇的な向上はもはや期待できない．特にメモリ領域に関しては，組み込み機器およびサーバーサイドで共通の課題であり，省メモリプログラミングに加えて分散や並列処理を活用したアルゴリズムの重要性が指摘されている [3].

また，パーソナルコンピュータや近年急速に普及しているスマートフォンに代表される組み込み機器では，従来の CPU に加えて SIMD(Sigle Instruction Multiple Data streams) や DSP(Digital Signal Processor) などのサプチップ搭載による数値演算能力の向上が顕著である．スマートフォンレベルでも，その数値演算能力は既に 10 年前のスパコンレベルの処理能力に匹敵するレベルにまで到達している [4][1]．その反面，サブチップで使用可能な作業領域メモリ量は，通常 CPU が利用可能なメモリ領域と比較すると極端に少ない．依然として組み込み機器で活用できるチップのメモリ容量にはかなりの制限がある．

サーバーサイドでもビックデータに代表される入力データ数の指数関数的に爆発的な要求に対しても，有限な単体サーバーのメモリ容量では圧倒的に少ない現状である．Apache Hadoop[5] に代表される複数コンピューターをクラスターとして活用する分散コンピューティング手法と合わせ，簡潔データ構造 [6] などの省メモリデータ構造も注目されている．

このような背景から，省メモリアルゴリズムはパーソナルからクラウドコンピューティング領域までの幅広い分野において，今後もより一層必要とされる活発な領域であると言える．

[1] 2011 年に発表された Apple 社の iPad2 で既に LINPACK が 1500 に到達．この数値は 1998 年の NEC SX-3(地球シュミレータの原型) と同水準である．

1.1.2 画像認識を取り巻く環境

小節 1.1.1 で述べたスマートフォンの性能向上を背景に Apple 社の AppStore や Google 社の GooglePlay など各モバイルプラットフォームでは，カメラで撮影した画像を加工するアプリケーションの人気が高い．画像加工のアプリケーションは，各マーケットで専用のアプリケーションカテゴリが存在することが多く，各モバイルプラットフォーム別のラインキングでも 1 位を獲得するほど活況な分野である [2].

これらの画像加工アプリケーション開発を支える画像認識および画像処理ライブラリとして OpenCV[12] に代表される OSS(Open Source Software) が開発者からの注目を集めている．またプロプライエタリソフトウェア (proprietary software) の分野でも，Apple 社は顔認識や表情認識などの画像認識機能を標準の公式 API(Application Programming Interface) に積極的に追加するなど，アプリケーション用途の拡大に努めている [4]．両者とも，現在も精力的な改良が継続されている．画像認識は，今後も増々の発展が期待できるアプリケーション分野である．

1.2 目的

節 1.1 で述べた背景から，本稿では省メモリアルゴリズムの題材として，画像認識分野の境界値表現検出アルゴリズムを取り扱う．

境界値探索アルゴリズムについては，既存アルゴリズム [8] を基本に検証を進める．既存アルゴリズムの検証結果を踏まえて改良アルゴリズムの考案および検証を行う．また最適な検出結果を得るために入力画像に対する前処理についての手法，検出後に出力された境界値表現の簡素化についての考案および検証を行う．

1.3 境界値表現

本稿の主題となる境界値表現 (contour representation) を関連する 2 値画像 (binary image) および連結性 (connection) と合わせて説明する．

1.3.1 2 値画像

2 値画像 (binary image) とは，画素値が 1(白画像) あるいは 0(黒画像) などとする 2 値の何れかで構成される画像を言う．色々なの画素値で構成されているデジタルカメラの撮影画像などを 2 値画像に変換する処理を 2 値化 (binarization) と言う [7]．2 値画像の例を図 1.5 に示す．

[2] 各モバイルプラットフォームで，全体または各カテゴリーで 1 位を記録したアプリケーションとしては，Instagram，DECOPIC, BeatyPlus，ProCam などがある．

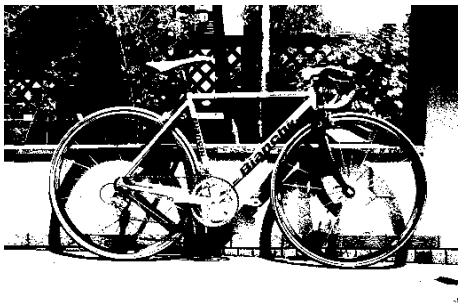
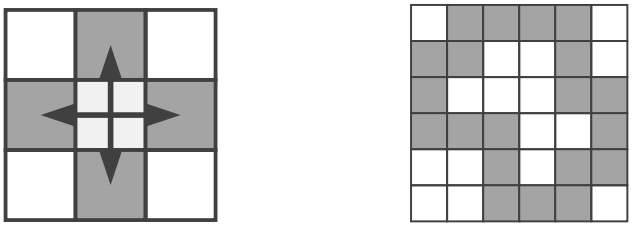


図 1.1: 2 値画像の例

1.3.2 連結性

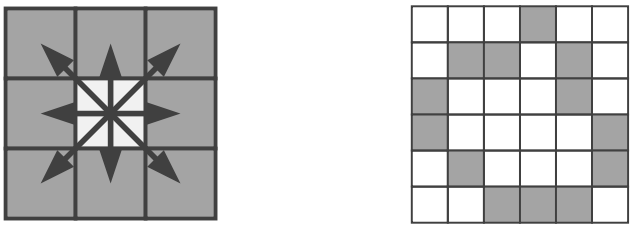
2 値画像で一回りする領域を連結 (connection) していると言う．図 1.2[a] に示すように中心画素に対して上下左右の画素を 4 近傍 (4 connected neighbor)，その近傍に対して連結性を定義したものを 4 連結性 (4 connection) と言う [7].



[a] 4 近傍の定義　　[b] 4 連結性の例

図 1.2: 4 近傍の定義と 4 連結性の例

図 1.3[a] に示すように 4 近傍に斜め方向の近傍を加えたものを 8 近傍 (8 connected neighbor)，その近傍に対して連結性を定義したものを 8 連結性 (8 connection) と言う [7].



[a] 8 近傍の定義　　[b] 8 連結性の例

図 1.3: 8 近傍の定義と 8 連結性の例

1.3.3　境界値表現

4 連結性あるいは 8 連結性で連結している画素の集合を連結性性分 (connected component) と言う．連結性成分の境界を求める事を輪郭追跡 (contour tracking) と言う [7]．この輪郭追跡で得られた軌跡は，画素間を通る水平及び垂直線分系列の境界線で表現できる．線分による境界をエッジ (Edge) と言う．画素の代わりに境界線の集合として 2 値画像を表す方式を境界値表現 (contour representation) と言う．境界値表現の例を図 1.4 に示す．

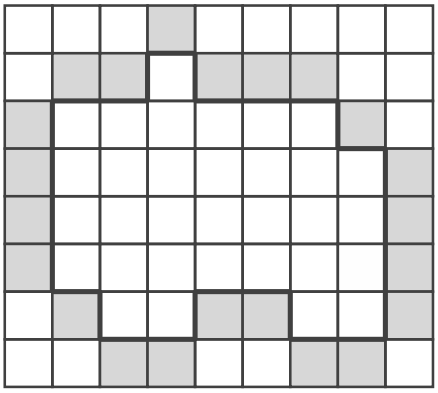


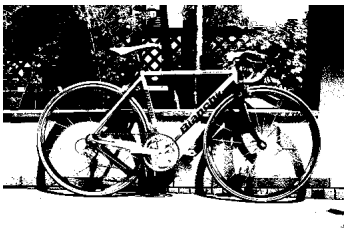
図 1.4: 境界値表現の例

1.4　境界値処理

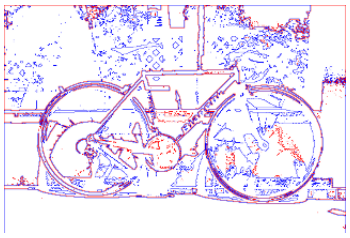
1.4.1　概要

本稿では画像を入力し境界値表現を出力するまでの一連の処理が対象となる．入力画像は境界検出アルゴリズムで処理する前に 2 値化の必要がある．2 値画像を境界検出アルゴリズムで処理し，最終的には境界値表現が出力される (図 1.5)．

[a] 入力画像



[b]2 値化画像



[c] 境界値画像

図 1.5: 入力から境界値検出までの画像遷移の例

本処理の全体フローを図 1.6 に示す．処理フローは，前処理部，2 値化部，境界検出部，簡素化部の順に構成される．

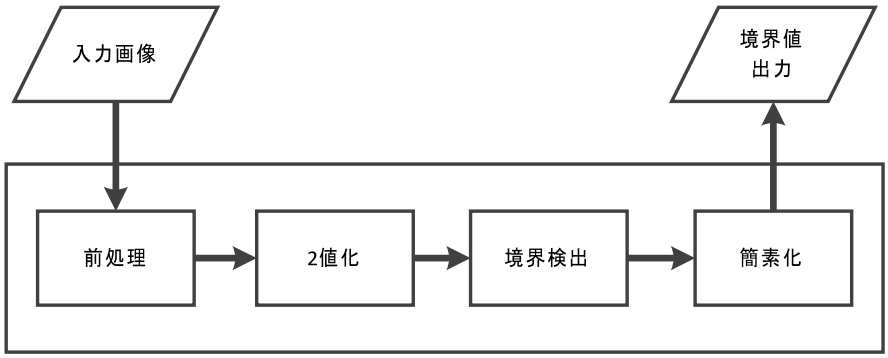
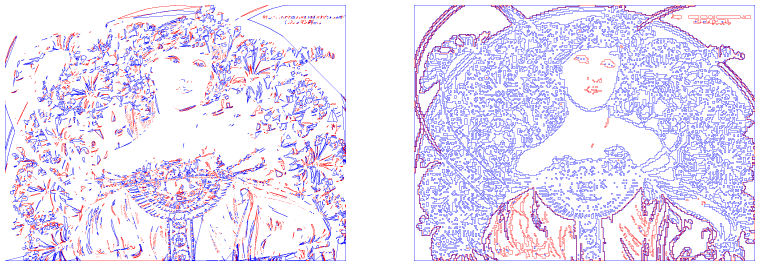


図 1.6: 境界値処理フロー

1.4.2 前処理

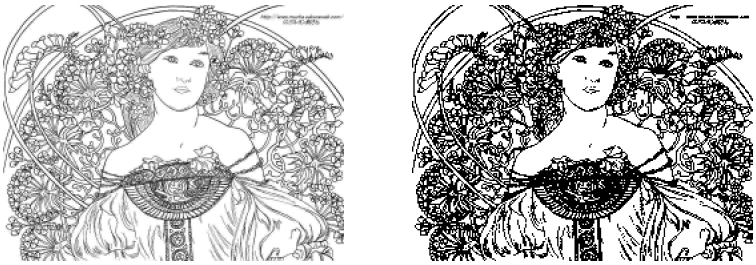
2 値化および境界値検出処理の効率化を目的として，境界検出処理前に入力画像に対してエフェクト処理を試した．境界値探索アルゴリズム [8] は入力画像に対してモザイク処理やバイリニア補間による拡大処理などのエフェクト処理を施した際に検出精度の向上が確認できた (図 1.7)．エフェクト処理有無による境界値検出の効果例を図 1.7 に示す．



[a] エフェクト処理なし　　[b] エフェクト処理あり

図 1.7: エフェクト処理による境界線検出結果の違い

一般的に，エフェクト処理により前処理検出される境界値多角形数が多く，多角形面積も大きい結果が得られた．図 1.7 の入力画像にエフェクト処理としてモザイク処理を施した例を図 1.8 に示す．このモザイクフィルタは 2x2 画素のブロック単位で各画素を各画素値の平均値で置き換えるものである．



[a] 入力画像　　[b] モザイクフィルタ適用後

図 1.8: エフェクト処理の例

このモザイク処理の結果，同じ色調のピクセルが 2x2 画素以上の単位で固まることが検出精度向上の要因と推察される．他の要因として入力画像固有の問題や，実装上の欠陥である可能性もある．ただし，エフェクト処理効果は本稿での主題ではない．そのためエフェクト処理効果の考察については今後の課題とし，本稿での詳細な検証は割愛した．

1.4.3　2 値化

本稿で用いた境界検出アルゴリズム [8] は 2 値画像 (binary image) を対象とするため，デジタルカメラの撮影画像などは境界検出処理前に 2 値化の必要がある．2 値化は CIELAB 色空間で処理する．出力される境界値表現結果に大きな影響があるため，境界値検出結果と合わせ感度分析を行った (図 1.9).

[a] 不適切な 2 値化　　[b] 適切な 2 値化

図 1.9: 2 値化処理の例

1.4.4　境界検出

本稿では画像境界値検出する輪郭追跡アルゴリズムは [8] を基本とした．本アルゴリズムは，省メモリアルゴリズムかつアルゴリズムも比較的簡素なため実装も容易である．また出力される境界値がすべて閉じた多角形である，多角形を構成する線分が他の多角形

辺と交差しないなどの優れた特徴がある．この特徴は，重心や面積などの幾何学的なパラメーターを複雑な例外処理なしに簡潔に算出できる利点がある．

ただし， 本アルゴリズムの単純な実装では探索方向や探索周辺画素により条件分岐が複雑となり，入力画像による条件分岐の偏りの実行速度への影響も懸念される．この課題について，近年のパーソナルコンピュータやスマートフォンに搭載される並列コンピューティング環境への実装を念頭に改良アルゴリズムを考察および検証を行う．

1.4.5 簡素化

出力される境界値多角形は，アルゴリズムの特性上から短い線分から構成される細かい段差の多角形として出力される (図 1.10)．

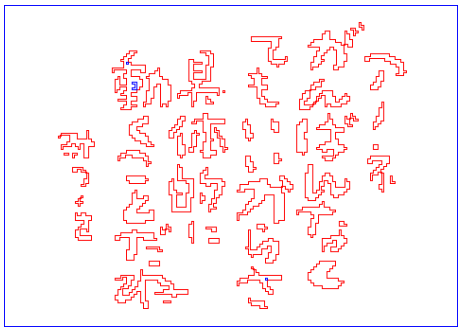


図 1.10: 境界値多角形の出力例

この課題に対して，出力された境界値多角形を簡素化 (simplification) するアルゴリズムを考察および検証を行う．

1.5 実装

1.5.1 検証環境

本稿の実装に用いた検証環境を表 1.1 に示す．検証プログラムは MacOSX 環境で実装した．画像入出力ライブラリとして OSS(Open Source Software) の libGD[9] を利用した．

表 1.1: 実装検証環境

機種	Apple Mac mini Late 2012
CPU	2.6 GHz Intel Core i7
メモリ	8GB
OS	MacOSX 10.8
プログラミング言語	C++
コンパイラ	XCode 4.6.3
利用ライブラリ	libGD 2.0.36

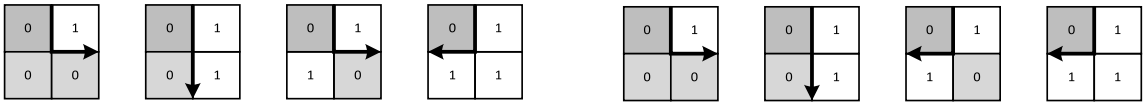
また小節 1.4.2 で説明した前処理のエフェクト処理については，画像処理ソフトウェアの GIMP[10] を活用した.

第2章　境界値検出

2.1　概要

2.1.1　境界検出アルゴリズム

本稿では，入力画像の境界値を検出する境界値探索アルゴリズムは [8] を基本とした．この境界値探索アルゴリズムは 2 値画像 (binary image) を対象とし，入力する画像は前処理で 2 値化 (binarization) が必要である．このアルゴリズムの特徴は，現在探索中の辺の向きおよび周囲画素の位置関係のみで次辺の探索方向が確定できる点である．次辺の探索方向は 4 近傍 (4-connected neighbor) か 8 近傍 (8-connected neighbor) で探索するかで異なる (図 2.1)．



[a] 4 近傍　　[b] 8 近傍

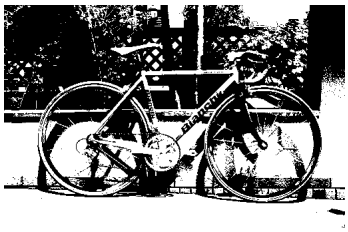
図 2.1: 近傍の種類による辿り方の違い

また，境界値表現の出力は，閉じた多角形かつ各辺が交差しない，多角形同士も交差しないなどの優れた特徴がある．この特徴は出力結果を評価する際に色々な幾何的特徴を容易に計算できる利点がある．

2.1.2　2 値化処理

境界値探索アルゴリズム [8] は，単純な白と黒ピクセルのみで構成される 2 値画像 (binary image) を対象としている．そのため，デジタルカメラの撮影画像などは，このアルゴリズムを適用する前に 2 値化 (binarization) が必要となる (図 2.2)．

[a] 入力画像

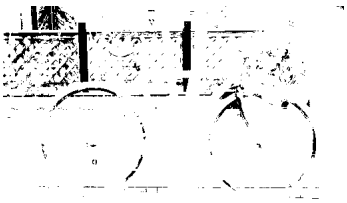


[b]2 値化後の画像

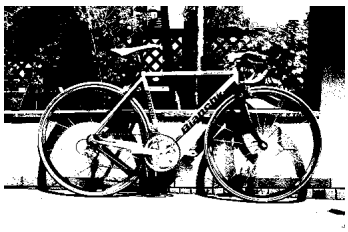
図 2.2: 入力画像の 2 値画像への変換例

2 値化への閾値には，CIELAB 色空間上で色の明度を表す L* 値を用いた．2 値画像への変換は，RGB 色空間で表現される画像を CIELLAB 色空間変換した後に処理する．RGB 色空間から色 Lab 空間への直接変換する式は存在しないため，XYZ 色空間を中間の色空間として RGB 色空間から CIELAB 色空間へ変換した．

CIELAB 色空間は，色の明度を表す L*，赤/マゼンタと緑の間の位置を表す a*，黄色と青の間の位置 b*の 3 つの値から構成される．[1]．バイナリ画像への変換に CIELAB 色空間を用いた利点は，色の明度を表す L*が概ね 0(黒)〜100 (白の拡散色) の範囲にある点である．この L*値に閾値 (threshold) を適用して 2 値画像へ変換した．閾値とする適切な L*値の選択は，検出する境界値全体の精度に影響する (図 2.3)．



[a] 不適切な L* 値の選択例



[b] 適切な L* 値の選択例

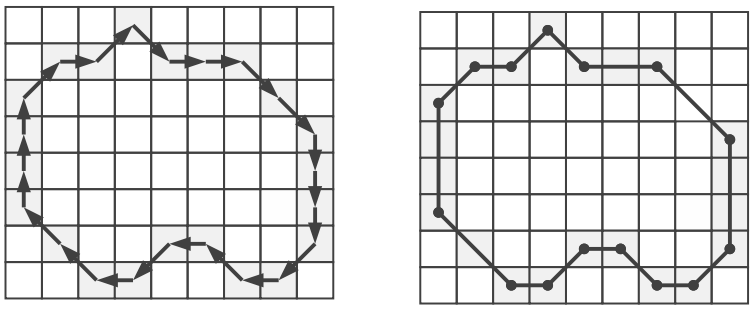
図 2.3: L*値の選択例

2.1.3 境界値表現データ構造

境界値表現のデータ構造としては Freeman チェイン [11] 方式がある (図 2.4[a])．このデータ構造は画像処理ソフトウェアの OpenCV でも採用されており [13] で，次頂点への遷移を連続して保持する方式である．本稿でも当初はこのデータ構造を用いて実装した．しかし，最終的には実装上の観点から境界値多角形 (edge polygon) 方式のデータ構造を用

[1]CIELAB 色空間各値のアスタリスク (*) は，別の色空間である HunterLab 色空間との違い明確化するために付けられている．本稿もこの方式に準拠して記載する．

いた (図 2.4[b]). このデータ構造は，開始点および近傍方向が変化した座標位置 (x, y) のみを保持する方式である.



[a] Freeman チェイン方式　　[b] 境界値多角形方式

図 2.4: 境界値表現方式の種類

本稿が目的とする境界値表現の簡素化では，方向が変化する変異点と変異点を両端点とする直線との距離を指標として用いる．境界値多角形方式は，Freeman チェイン方式と比較し，各変異点に高速にアクセスできるデータ構造である．また連続した近傍の方向データは省略されるため連長圧縮 (run length encoding) と同様な符号化の効果もある.

2.2 検証

最初に，境界値検出の前処理となる 2 値画像 (binary image) の生成について，CIELAB 色空間の L*値を閾値とする感度分析の検証を行った．感度分析は後述する境界値出力密度 [2] を最大とするのを目的とし，その評価指標として出力境界値多角形周囲長の総合計 (総周囲長) を用いた.

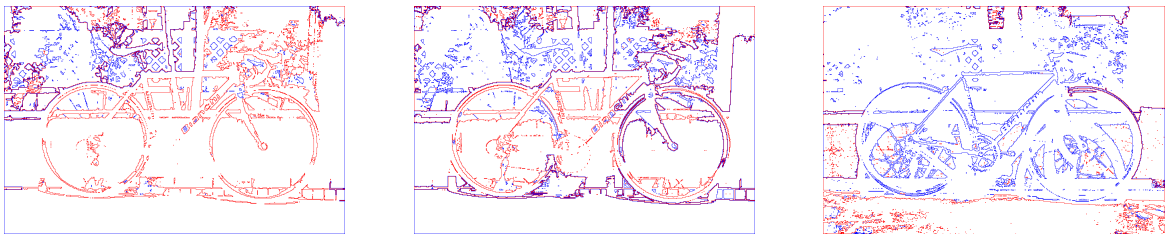
2.2.1 自然画像

図 2.5 のようなデジタルカメラでの撮影画像を入力データとし，L*値を 0〜100 の範囲で遷移させ検証した．評価指標としては総周囲長を用いて，その統計量について検証した.

[2]境界値出力密度 = 総出力境界値周囲長 ÷ 入力画像面積 (横幅 × 縦幅)

図 2.5: 自然画像の入力例

図 2.5 を入力画像として，L*を可変させた境界値検出結果の出力画像および統計例を (図 2.6, 表 2.1) に示す.



[a] L* = 50　　[b] L* = 60　　[c] L* = 80

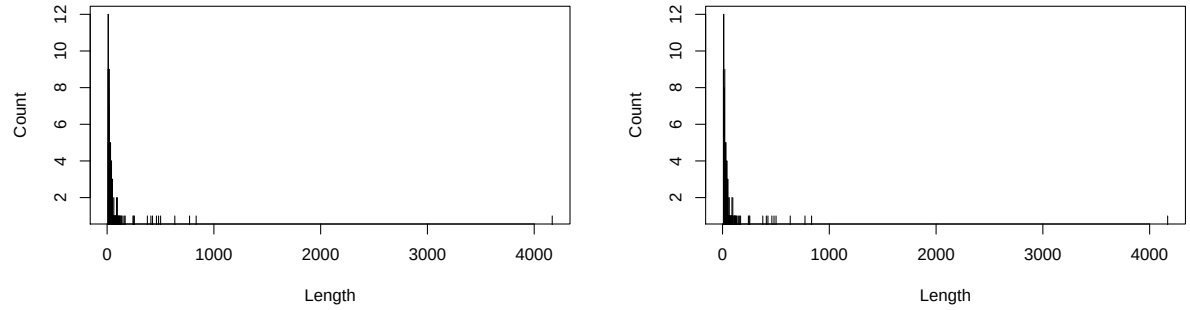
図 2.6: 自然画像の境界検出結果例

表 2.1: 自然画像の境界検出結果例

L*	0	10	20	30	40	50	60	70	80	90	100
境界値数	8	25	217	554	789	996	1206	1604	1974	1031	1
最小周囲長	4	4	4	4	4	4	4	4	4	4	3410
最大周囲長	3410	3414	3870	4330	9298	7214	14246	21104	7304	1668	3410
周囲長中央値	4	4	8	12	8	8	8	4	4	4	3410
周囲長平均	433.75	151.76	39.7143	41.84	57.64	59.15	59.27	48.45	27.66	18.69	3410
総周囲長	3470	3794	8618	23178	45478	58922	71482	77718	54604	19270	3410
周囲長標準偏差	1124.95	666.34	264.92	203.01	377.35	354.18	517.48	590.90	192.70	84.09	0

境界値出力多角形の幾何学特徴パラメーターの代表的なものには，周囲長の他に重心，外接長方形，面積，円形度，オイラー数がある [7]．当初は評価する幾何学特徴パラメーターとしては面積が適切な指標と考えたが，面積を計算する際に凸形状以外や他多角形を内包する穴 (hole) 場合を考慮するなど複雑な計算が必要であった．そのため多角形の面積と比例関係にあり計算が容易な周囲長を検証の指標とした.

出力される境界値数と総周囲長数には高い相関関係が見受けられた．また周囲長はL*に依らず中央値を12を最大とするポアソン分布となり，周囲長12以下の多角形に非常に偏った分布の特徴が見受けられた (図2.7).



[a] L* = 30 の分布　　[b] L* = 60 の分布

図 2.7: 自然画像の総周囲長分布例

2.2.2 境界値密度

自然画像での検証結果から，定性的な評価指標として総周囲長と視覚的な多角形密度の相関性が高い結果が得られた．総周囲長は入力画像の画素数に依存するため，以下の式にて入力画素数に対して正規化する境界値密度を考案した (2.1).

$$\text{境界値密度} = \frac{\text{総周囲長}}{\text{入力画像縦幅} \times \text{入力画像横幅}} \tag{2.1}$$

本境界値検出アルゴリズムの特徴として，境界値が他の境界値と交わらない特徴があり境界値密度は0〜1の範囲をとる．図2.6での総周囲長および境界値密度を検出例を表2.2に示す.

表 2.2: 自然画像の境界密度例

L*	0	10	20	30	40	50	60	70	80	90	100
総周囲長	3470	3794	8618	23178	45478	58922	71482	77718	54604	19270	3410
境界値密度	.0049	.0054	.0123	.0331	.0650	.0842	.1111	.1111	.0780	.0275	.0048

表2.2のL*値と境界値密度の関係グラフを図2.8に示す．自然画像の場合，L*値の変化により境界値密度の変化が見受けられた．また境界値密度が0.1を超えると対象のオブジェクトが認識できる定量的な結果が得られた.

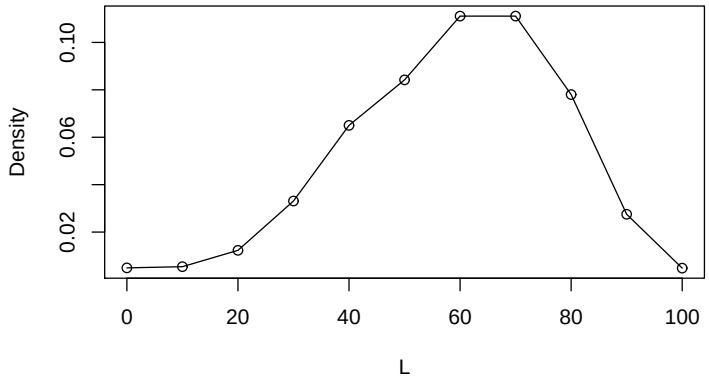


図 2.8: 自然画像の境界密度グラフ例

図 2.8 の L*値変化による境界値密度グラフの山は 1 つであり，近似曲線は 2 次多項式となる結果が得られた．この特性から高速にある程度妥当な L*値を求めるには以下のアルゴリズムとなる (Algorithm1)．このアルゴリズムの引数には，最初に探索を開始する L*閾値 (threshold)，どの程度の間隔精度で探索をするかのオフセット値 (offset) を指定する．

Algorithm 1 妥当な L*値の決定

procedure GetSuitableThreshold($threshold, offset$)
  $circuitLength = GetCircuitLength(threshold)$
  $prevCircuitLength = GetCircuitLength(threshold - offset)$
  $maxCircuitLength = prevCircuitLength$
  $maxThreshold = threshold - offset$
  $nextCircuitLength = GetCircuitLength(threshold + offset)$
  if $maxCircuitLength < nextCircuitLength$ **then**
    $maxCircuitLength = nextCircuitLength$
    $maxThreshold = threshold + offset$
  end if
  if $circuitLength <= maxCircuitLength$ **then**
    return $GetSuitableThreshold(maxThreshold, offset)$
  end if
  return $threshold$
end procedure

2.2.3 手書き画像

その一方，図 2.9 のような習字や漫画などの手書きの画像に関しては，L*値の変化に関係なく境界値密度はほぼ程度一定である特徴が見られた．

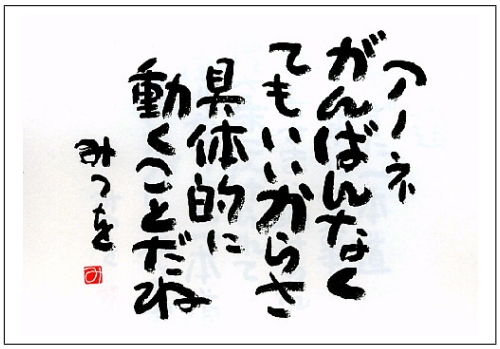


図 2.9: 手書き画像の入力例

自然画像と同じく，手書き画像での境界値検出結果画像および統計例を図 2.10 表 2.3 図 2.11．に示す．

[a] L* = 10　[b] L* = 50　[c] L* = 90

図 2.10: 手書き画像の境界検出結果例

表 2.3: 手書き画像の境界密度例

L*	0	10	20	30	40	50	60	70	80	90	100
総周囲長	7290	8402	8798	8900	9042	9234	9496	9748	9772	9808	1636
境界値密度	.0446	.0514	.0539	.0545	.0554	.0565	.0581	.0597	.0598	.0600	.0100

自然画像と比較すると手書き画像は L*値の変化に関係なく境界値密度はほぼ一定であった．この要因は今回検証に用いた手書き画像が元々量子化レベル数が少ないグレースケール画像 (grayscale image) であり 2 値画像に近かったためと推察される．表 2.3 の L*値と境界値密度の関係グラフを図 2.11 に示す．

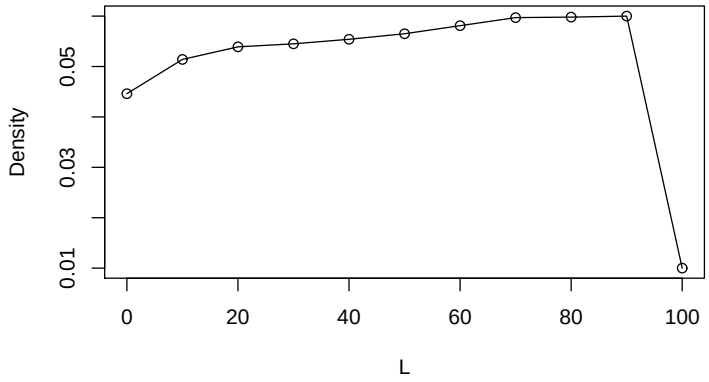


図 2.11: 手書き画像の境界密度グラフ例

図 2.11 の L*値変化による境界値密度グラフの近似曲線は線形近似となる．近似曲線の違いはあるが，手書き画像の場合にも高速にある程度妥当な L*値を求めるアルゴリズムは (Algorithm1) が適用できる．

2.3 考察

今回，2 値化の閾値の L*値を 0〜100 の範囲内で境界値密度の統計を検証した．境界値密度は，自然画像と手書き画像では L*の遷移による傾向の違いが見受けられた．

最良の境界線表現出力を得るためには，境界値密度を最大にするのが望ましい．ただし検証結果からある程度妥当な L*値でも十分な結果が得られることが確認できた．Algorithm1 は入力画像の種類によらず妥当な L*閾値が高速に得るアルゴリズムである．

また 2 値化処理時にある程度画像の種類を判別しておくことにより，Algorithm1 の閾値の引数に，判別された画像種別毎に適切な値を初期値とすることも考察される．適切な閾値が初期値に設定できれば，妥当な L*値による閾値決定処理の更なる高速化も期待できる．

第3章　境界値検出アルゴリズムの改良

3.1　概要

3.1.1　課題

本稿で用いた境界値探索アルゴリズム [8] は，通常の制御構造 (IF, SWITCH 文) による条件分岐で実装される．実装上，輪郭追跡には作業変数として，現座標 (x,y) と探索方向 (search direction) の状態を保持する必要がある．探索中に，現座標は入力画像の縦横範囲内，探索方向は北 (North)，南 (South)，東 (East)，西 (West) の何れかである (図 3.1).

[a] 南向き　[b] 西向き　[c] 北向き　[4] 東向き

図 3.1: 探索方向別の作業変数

4 近傍 (4-connected neighbor) で探索方向が南 (South) である場合の周辺境界パターン (Boundary Edge Pattern) における遷移は (図 3.2) となる．

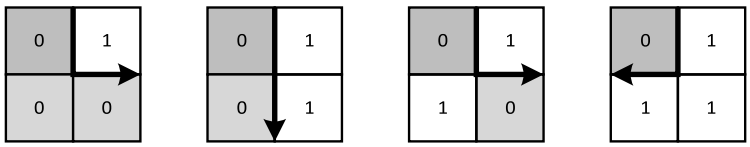


図 3.2: 南向き探索および 4 近傍時の次辺への遷移

この探索処理を単純に実装すると，次辺決定アルゴリズムは探索方向と探索方向周辺の境界パターンの条件分岐の入れ子構造となり煩雑となる．4 近傍時の次辺決定アルゴリズムを Algorithm2 に示す.

Algorithm 2 4近傍ルールでの条件分岐による次辺決定

```
procedure GetNextEdge(x, y, searchDirection)
    fowardDirectionPattern ← GetForwardDirectionPattern(x, y)
    if searchDirection = SOUTH then                    ▷ 南向き探索時のコード
        if fowardDirectionPattern = 0 then
            searchDirection ← EAST
        else if fowardDirectionPattern = 1 then
            y ← y + 1
        else if fowardDirectionPattern = 2 then
            searchDirection ← EAST
        else
            searchDirection ← WEST
            x ← x − 1
            y ← y + 1
        end if
    else if searchDirection = EAST then
        .....                                          ▷ 東向き探索時のコード (省略)
    else if searchDirection = NORTH then
        .....                                          ▷ 北向き探索時のコード (省略)
    else
        .....                                          ▷ 西向き探索時のコード (省略)
    end if
end procedure
```

Algorithm2 は南向き探索時のみ詳細に記述した．東西北探索時の記述は省略したが，図 3.2 に示す通り他の方向の場合にも同じような条件分岐が必要である．省略しない場合の Algorithm2 は 53 ラインである．

Algorithm2 は IF 文で記述されており，各処理分岐の実行順序が実装後のコンパイル時に固定される．SWITCH 文による記述ではコンパイラにより 2 分割法分岐などの最適化が施される場合があるが，いずれにしろ入力画像の探索成分分布により実行速度が偏った傾向となる懸念がある．例えば，分岐が南向きの探索を最初に処理するように実装されていれば，北向きの探索成分が多い入力画像の場合には実行速度の低下が懸念される．この懸念は探索方向周辺の境界パターンの条件分岐にも当てはまる．結果として，各近傍パターン毎の条件分岐は探索方向分岐の 4 種類と探索方向周辺の境界パターンの条件分岐の 4 種類の組み合わせで 16 パターン，近傍種別が 2 種類あるので総計 32 種類の組み合わせが存在する．

現実的には Algorithm2 では入力画像による動的な最適化も難しい．また各条件分岐により次回の探索方向と現座標が確定するが，その分岐ごとの処理も画一ではない．結果と

して改良探索方向と探索方向周辺の境界パターン別に処理を記述する必要があり，より煩雑な実装となる．

3.2 想定

3.2.1 対象モデル

既存アルゴリズムの改良には，現在市場で稼働している並列コンピュータモデル上での実装を念頭に置いた．対象とする並列コンピュータモデルは，現在パーソナルコンピューターやスマートフォンなどのデバイスに数多く搭載されている OpenCL[17] 仕様 v1.2 をベースに，この仕様制限内で動作するように改良を検討した．

OpenCL アーキテクチャモデルの大きな特徴としては，高速にアクセス可能な極少量のローカルメモリが仕様上定義されている点である．このローカルメモリは仕様上必須であり 16KB 以上と定義されている [1]．また，GPU(Graphics Processing Unit) ベースの OpenCL 環境では画像処理用の高速かつ大量のメモリが実装されている場合もあるが，省メモリプログラミングが本稿の主題であるため広域なメモリ領域は考慮の対象とはしない．

また OpenCL の特徴的な仕様として，一般の C 言語などのプログラミング言語と比較すると，ビットフィールドの利用や再帰処理ができないなどの制限事項も検討上の留意点となる．

3.2.2 並列手法

既存アルゴリズムの並列化の手法として，大別するとデー タ並列とタスク並列の技法がある [18]．本稿では省メモリプログラミングの観点でデータ並列を中心に検討した．

タスク並列ベースのアルゴリズムは，クリティカルセク ションなどの処理によりアルゴリズムが複雑となる場合もあり対象外とした．これに対して，データ並列はアルゴリズムおよび実装は簡素になる傾向にあり，定数領域プログラムおよび省メモリプログラミングの副次的な特徴とも合致するため考案の基準手法とした．

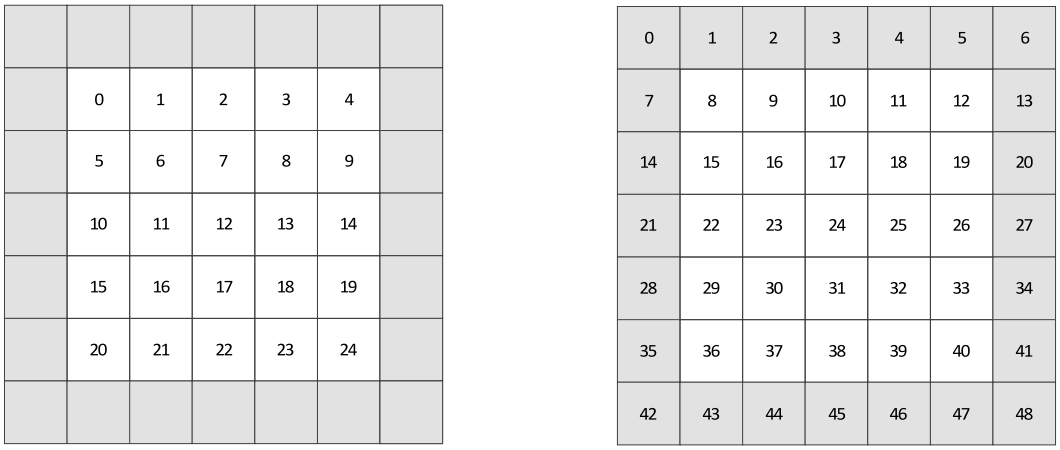
3.3 拡張探索データモデル

既存データモデルでは，現在位置 (x,y) と探索方向 (search direction) の 1 つの独立した変数で処理を実装していた．この現在位置と探索方向の 2 変数を 1 つの変数で処理し、各辺に連続した番号 (index) を割り当てる目的で，新しい拡張データモデルを考案した．

[1] ただし OpenCL バージョン 1.2 からは 32KB 以上に拡張された．

3.3.1 拡張画素モデル

最初に，入力画像の周辺に対して 1 画素ずつ追加する，拡張画素モデルについて説明する (図 3.3).



[a] 既存画素モデル　　[b] 拡張画素モデル

図 3.3: 既存と拡張画素モデルの違い

図 3.3[b] 拡張画素モデルは，既存画素モデルの入力画像画素の周辺に対して 1 画素ずつ加えたものである．拡張画素モデルでは，既存画素モデルに対して横幅および高さが 2 画素分多くなる．拡張画素モデルでも各画素に対して画素番号 (pixel index) が割り当てられる．既存画素モデルの座標 (x, y)，横幅 (width)，高さ (height) と画素番号 (k) との関連を図 3.4，相互変換の関係式を以下に示す.

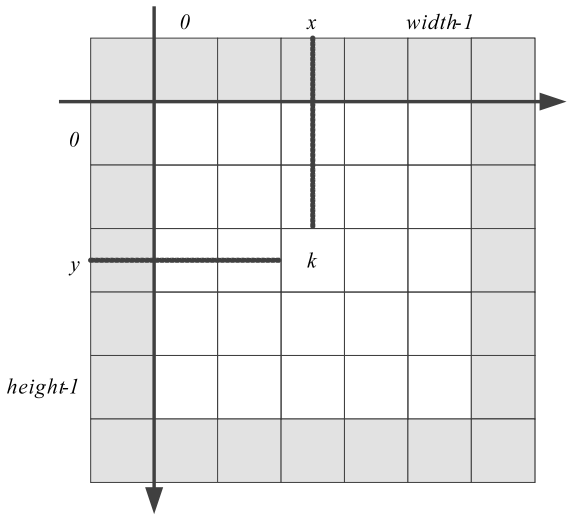


図 3.4: 既存画素モデルと画素番号

$$k = (y \times width) + x \tag{3.1}$$
$$x = k \bmod width \tag{3.2}$$
$$y = floor((k \div width)) \tag{3.3}$$

拡張画素モデルの座標 (x', y')，横幅 (width')，高さ (height') と画素番号 (k') との関連を図 3.5，既存画素モデル (図 3.4) 変数からの変換を含めた関係式を以下に示す.

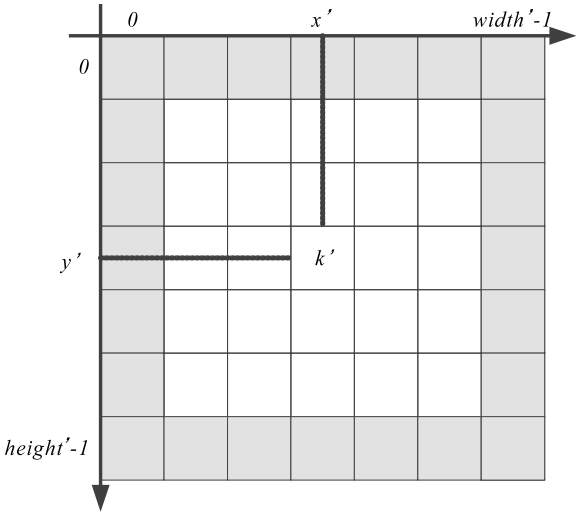


図 3.5: 拡張画素モデルと画素番号

$$
\begin{aligned}
width' &= width + 2 && (3.4)\\
height' &= height + 2 && (3.5)\\
x' &= k' \bmod width' \\
&= x + 1 = (k \bmod width) + 1 && (3.6)\\
y' &= floor(k' \div width') \\
&= y + 1 = floor((k \div width)) + 1 && (3.7)\\
k' &= (y' \times width') + x' \\
&= ((floor((k \div width)) + 1) \times (width + 2)) + ((k \bmod width) + 1) && (3.8)
\end{aligned}
$$

拡張画素モデル (図 3.5) 変数から，既存画素モデル (図 3.4) 変数への関係式を以下に示す.

$$
\begin{aligned}
width &= width' - 2 && (3.9)\\
height &= height' - 2 && (3.10)\\
x &= x' - 1 && (3.11)\\
y &= y' - 1 && (3.12)\\
k &= (y \times width) + x \\
&= ((y' - 1) \times (width' - 2)) + (x' - 1) && (3.13)
\end{aligned}
$$

3.3.2 拡張辺モデル

拡張画素モデルから画素と同様に各辺にも辺番号 (edge number) の割当を考える．辺番号は連続した番号の割り当てを目的とする.

拡張辺モデルでは，拡張画素モデルの画素番号を 2 倍にしたものを基準として辺番号を割り当てる．拡張辺モデルの左辺 (left edge) は対象の画素番号を 2 倍してから 1 引いた値，下辺 (under edge) は対象の画素番号を 2 倍にした値を辺番号として割り当てる (図 3.6).

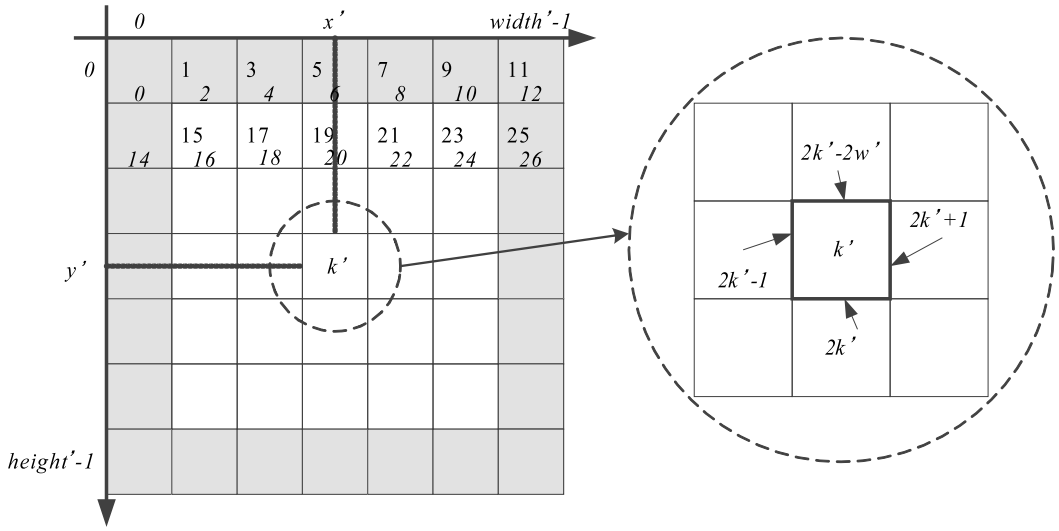


図 3.6: 拡張辺モデル

拡張辺モデルでは，辺番号 (en') および周辺画素の値から，現座標 (x',y') および探索方向 (search direction) の特定が可能である．画素番号 (k ') , 現座標 (x',y') については式 3.6, 式 3.7 より以下の関係式となる．

$$k' = (en' + 1) \div 2 \tag{3.14}$$

$$x' = k' \bmod width' \tag{3.15}$$

$$y' = floor(k' \div width') \tag{3.16}$$

3.4 探索画素周辺の辺番号

この拡張辺モデルをを用いて，現在探索中の画素番号 (k') と探索の結果移動する周辺辺番号との関連性は図 3.7 となる．

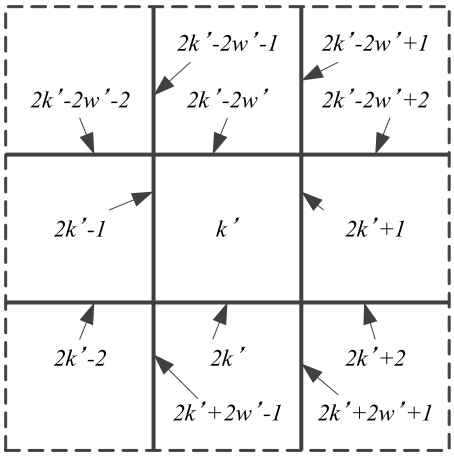
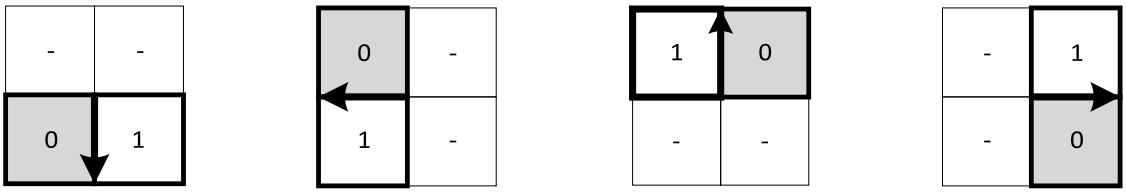


図 3.7: 探索画素周辺の辺番号

3.5 探索方向の特定

境界値探索アルゴリズム [8] は，図 3.6 に示す通り探索方向は東西南北の 4 通りの何れかの値をとる．境界値探索アルゴリズムの特徴として，探索方向の左側には白 (1) 値，右側には黒 (0) の値の画素 (またはその反対の組み合わせ) が必ず存在する．



[a] 南向き　[b] 西向き　[c] 北向き　[4] 東向き

図 3.8: 探索方向別の周辺画素との関係

探索方向を特定するには，まず辺番号 (en') の偶奇により水平辺 (偶数) か垂直辺 (奇数) かを特定する．水平辺の場合には辺番号 (en') に対応する画素，垂直編の場合には辺番号 (en')+1 の画素を確認すれば，探索方向の東西南北が特定となる．辺番号 (en') から探索方向特定のアルゴリズムを Algorithm3 に示す．

Algorithm 3 辺番号からの探索方向特定

```
procedure GetSearchDirection(edgeIndex)
    edgeDirection ← edgeIndex mod 2
    if edgeDirection = 0 then                                   ▷ 水平辺
        if BinaryPixels[(edgeIndex/2)] = 0 then
            return WEST
        else
            return EAST
        end if
    else                                                        ▷ 垂直辺
        if BinaryPixels[((edgeIndex + 1)/2)] = 0 then
            return NORTH
        else
            return SOUTH
        end if
    end if
end procedure
```

3.6 進行方向境界パターンの特定

進行方向境界パターン (fp) は，進行方向に対して左前画素の値を下位ビット (low bit)，右前の画素を上位ビット (high bit) として加算した値となる (図 3.9).

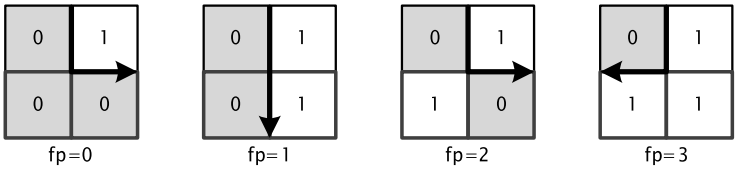


図 3.9: 探索画素周辺の辺番号

進行方向境界パターン (fp) は進行方向の左右前画素のパターンを符号化したものである．進行方向境界パターン (fp) の算出式は以下となる.

$$fp = (\text{進行方向右前画素値} \times 2) + \text{進行方向左前画素値} \tag{3.17}$$

辺番号 (en') から進行方向境界パターン (fp) を特定するアルゴリズムを Algorithm4 に示す.

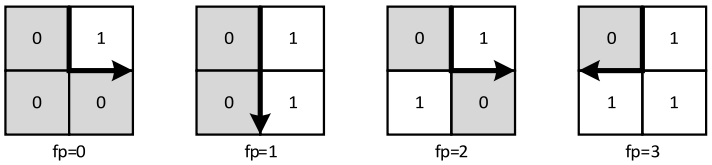
Algorithm 4 辺番号からの進行方向境界パターン特定

procedure GETFORWARDPATTERN($edgeIndex$)
  $searchDirection \leftarrow GetSearchDirection(edgeNumber)$
  if $searchDirection = SOUTH$ **then** ▷ 南向き
    $lowBitEdgeIndex \leftarrow edgeIndex + 2w' + 1$
    $highBitEdgeIndex \leftarrow lowBitEdgeIndex - 2$
  else if $searchDirection = NORTH$ **then** ▷ 北向き
    $lowBitEdgeIndex \leftarrow edgeIndex - 2w' - 1$
    $highBitEdgeIndex \leftarrow lowBitEdgeIndex + 2$
  else if $searchDirection = EAST$ **then** ▷ 東向き
    $lowBitEdgeIndex \leftarrow edgeIndex + 2$
    $highBitEdgeIndex \leftarrow lowBitEdgeIndex + 2w'$
  else ▷ 西向き
    $lowBitEdgeIndex \leftarrow edgeIndex + 2w' - 2$
    $highBitEdgeIndex \leftarrow lowBitEdgeIndex - 2w'$
  end if
  $lowBitPixelIndex \leftarrow lowBitEdgeIndex \div 2$
  $highBitPixelIndex \leftarrow highBitEdgeIndex \div 2$
  return $(binImage[highBitPixelIndex] \times 2) + binImage[lowBitPixelIndex]$
end procedure

3.7 拡張辺モデルによる探索遷移

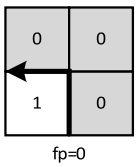
3.7.1 4近傍

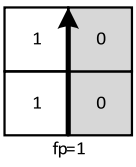
拡張辺モデルで 4 近傍探索時の探索方向 (東西南北) と進行方向境界パターン (fp) 別に次辺 (next edge) および現在辺 (current edge) との差分 (difference) を図 3.10，図 3.11, 図 3.12, 図 3.13 に示す.

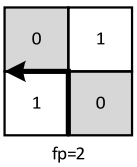


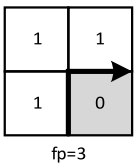
fp	Next Edge	Diffrence
0	2k'	+1
1	2k' +2w' -1	2w'
2	2k'	+1
3	2k' -2	-1

図 3.10: 4 近傍探索の次辺遷移 (南向き:現在辺=2k'-1)



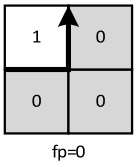


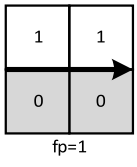


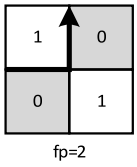


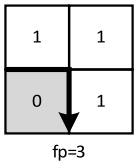
fp	Next Edge	Diffrence
0	2k' -2w'	-2w' -1
1	2k' -2w' +1	-2w'
2	2k' -2w'	-2w' -1
3	2k' -2w' +2	-2w' +1

図 3.11: 4 近傍探索の次境界遷移 (北向き:現在辺=2k'+1)



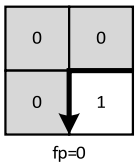


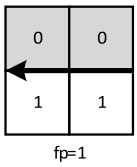


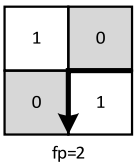


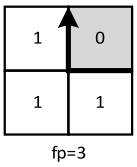
fp	Next Edge	Diffrence
0	2k' +1	+1
1	2k' +2	+2
2	2k' +1	+1
3	2k' +2w' +1	2w' +1

図 3.12: 4 近傍探索の次辺遷移 (東向き:現在辺=2k')







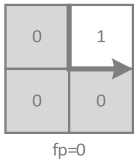


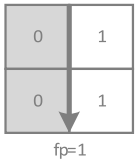
fp	Next Edge	Diffrence
0	2k' -1	2w' -1
1	2k' -2w' -2	-2
2	2k' -1	2w' -1
3	2k' -2w' -1	-1

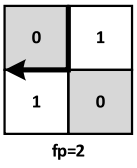
図 3.13: 4 近傍探索の次辺遷移 (西向き:現在辺=2k'-2w)

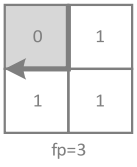
3.7.2 8 近傍

8 近傍は 4 近傍と進行方向境界パターン (fp) が 2 の場合のみ次辺への遷移が異なる．4 近傍との差分を強調した形で，同様に拡張辺モデルで 8 近傍探索時の探索方向 (東西南北) と進行方向境界パターン (fp) 別に次辺 (next edge) および現在辺 (current edge) との差分 (difference) を図 3.14，図 3.15, 図 3.16, 図 3.17 に示す.



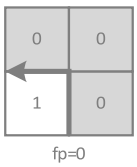


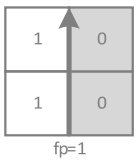


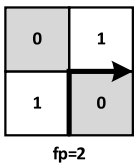


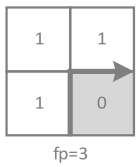
fp	Next Edge	Diffrence
0	2k'	+1
1	2k' +2w' -1	2w'
2	**2k' -2**	**-1**
3	2k' -2	-1

図 3.14: 8 近傍探索の次辺遷移 (南向き:現在辺=2k'-1)



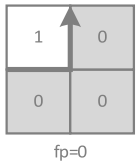


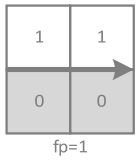


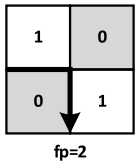


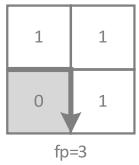
fp	Next Edge	Diffrence
0	2k' -2w'	-2w' -1
1	2k' -2w' +1	-2w'
2	**2k' -2w' +2**	**-2w' +1**
3	2k' -2w' +2	-2w' +1

図 3.15: 8 近傍探索の次境界遷移 (北向き:現在辺=2k'+1)



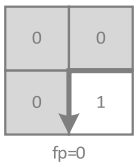


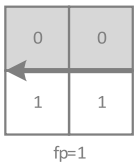


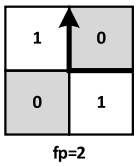


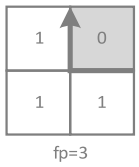
fp	Next Edge	Diffrence
0	2k' +1	+1
1	2k' +2	+2
2	**2k' +2w' +1**	**2w' +1**
3	2k' +2w' +1	2w' +1

図 3.16: 8 近傍探索の次辺遷移 (東向き:現在辺=2k')









fp	Next Edge	Diffrence
0	2k' -1	2w' -1
1	2k' -2w' -2	-2
2	**2k' -2w' -1**	**-1**
3	2k' -2w' -1	-1

図 3.17: 8 近傍探索の次辺遷移 (西向き:現在辺=2k'-2w)

3.8 拡張辺モデルによる次辺差分配列

3.8.1 4 近傍

図 3.10，図 3.11, 図 3.12, 図 3.13 から，探索方向と進行方向境界パターン (fp) をインデックスとする次辺差分配列 (nd) を以下に示す.

$$nd[SOUTH][0] = 1 \tag{3.18}$$
$$nd[SOUTH][1] = 2 \times width' \tag{3.19}$$
$$nd[SOUTH][2] = 1 \tag{3.20}$$
$$nd[SOUTH][3] = -1 \tag{3.21}$$

$$
\begin{aligned}
nd[NORTH][0] &= (-2 \times width') - 1 && (3.22)\\
nd[NORTH][1] &= -2 \times width' && (3.23)\\
nd[NORTH][2] &= (-2 \times width') - 1 && (3.24)\\
nd[NORTH][3] &= (-2 \times width') + 1 && (3.25)
\end{aligned}
$$

$$
\begin{aligned}
nd[EAST][0] &= 1 && (3.26)\\
nd[EAST][1] &= 2 && (3.27)\\
nd[EAST][2] &= 1 && (3.28)\\
nd[EAST][3] &= (2 \times width') + 1 && (3.29)
\end{aligned}
$$

$$
\begin{aligned}
nd[WEST][0] &= (2 \times width') - 1 && (3.30)\\
nd[WEST][1] &= -2 && (3.31)\\
nd[WEST][2] &= (2 \times width') - 1 && (3.32)\\
nd[WEST][3] &= -1 && (3.33)
\end{aligned}
$$

3.8.2 8 近傍

8 近傍は 4 近傍と進行方向境界パターン (fp) が 2 の場合のみ次辺への遷移が異なる．図 3.10，図 3.11, 図 3.12, 図 3.13 から，探索方向と進行方向境界パターン (fp) をインデックスとする次辺差分配列 (nd) を 4 近傍との差分のみ以下に示す.

$$
\begin{aligned}
nd[SOUTH][2] &= -1 && (3.34)\\
nd[NORTH][2] &= (-2 \times width') + 1 && (3.35)\\
nd[EAST][2] &= (2 \times width') + 1 && (3.36)\\
nd[WEST][2] &= -1 && (3.37)
\end{aligned}
$$

3.9 拡張辺モデルによるアルゴリズム改良

次辺差分配列 (nd) を用いて探索方向特定アルゴリズム (Algorithm3) を改良する．改良後のアルゴリズムを Algorithm5 に示す.

Algorithm 5 拡張辺モデルによる次辺決定

procedure GETNEXTEDGE($edgeNumber$)
$searchDirection \leftarrow GetSearchDirection(edgeNumber)$
$fowardDirectionPattern \leftarrow GetForwardPattern(edgeNumber)$
return $edgeNumber + nd[searchDirection][fowardDirectionPattern]$
end procedure

次辺差分配列により，引数は辺番号 (edge number) のみの 1 変数となり，既存アルゴリズムと比較すると関数内も条件分岐を含まない単純なアルゴリズム記述に改良できた．

既存アルゴリズム (Algorithm2) では 53 ラインであった探索方向特定アルゴリズムが，改良アルゴリズム (Algorithm5) では 5 ラインに改善できた．結果として，次辺差分配列により既存アルゴリズムと拡張辺モデルによる改良アルゴリズムを比較するとライン数を大幅に削減できた．

また条件分岐をアルゴリズム除外したことにより，入力画像の探索方向成分分布による実行速度の偏り懸念も解消できた．

3.10 検証

改良アルゴリズム (Algorithm5) でアルゴリズム上の改善は確認できた．実行速度についても表 1.1 の検証環境上で実装し，性能向上効果を含めての検証を行った．検証は次辺決定アルゴリズム実行時間を対象として，既存アルゴリズム (Algorithm3) との相対評価を行った．

3.10.1 計測用入力画像

各アルゴリズムの実行計測は写真などの画像では画像の種類も様々で外部要因となり，アルゴリズム自身の均一的な性能評価が難しい．そのため実行時間計測用に外枠のみの正方形な入力画像を検証用に準備した．この実行時間計測用の入力画像例を図 3.18 に示す．

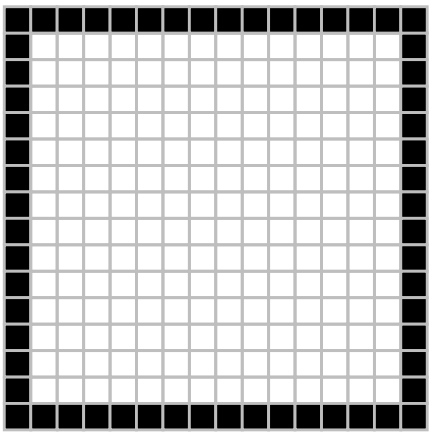


図 3.18: 実行時間計測用の正方形入力画像例

この計測用画像データの特徴としては探索方向の東西南北が均一に存在している点である．既存アルゴリズム (Algorithm3) も入力画像の探索方向成分分布による実行速度の偏りの影響も均一化して評価できる利点がある．

3.10.2 計測区間

実行時間は 3.10.1 の正方形画像を入力画像として全ての画素を開始点する総実行時間を計測した．しかし正方形画像の殆どの画素は境界 (Edge) 上の点ではない．開始点 (start point) 境界上の点でない場合，境界値探索処理はその画素の左方にある境界点 (edge point) に移動してから開始点として処理が開始される (図 3.19)．

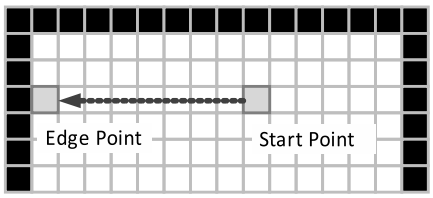


図 3.19: 開始点から境界点への移動

実行時間は既存アルゴリズム (Algorithm3) と拡張辺モデルによる改良アルゴリズム (Algorithm5) の探索方向特定処理実装のみを計測対象とした．境界点への移動処理を含めた他の処理は計測時間に含まれない．結果として探索方向特定アルゴリズムの実行時間のみの計測評価となる．

3.10.3 計測結果

検証は各アルゴリズムの実行時間 (秒) および既存アルゴリズム (Algorithm3) から改良アルゴリズム (Algorithm5) の性能向上比率 (Progress Rate) を指標として評価した．正方形一辺長を 512 画から 16384 画素の範囲内でサンプリングして計測した結果の平均値を表 3.1 に示す.

表 3.1: 正方形入力画像による計測結果

正方形一辺長	既存平均実行時間 (秒)	拡張辺モデル平均実行時間 (秒)	平均性能向上比率
512	0.167338	0.150965	1.10846
1024	0.583105	0.456878	1.27628
4096	10.0906	7.31307	1.37981
8192	43.3256	30.0348	1.44251
16384	178.49	121.09	1.47403

改良アルゴリズムは全ての正方形一辺長で既存アルゴリズムより高速な結果が得られた．表 3.1 の正方形一辺長と性能向上比の関係性を図 3.20 に示す.

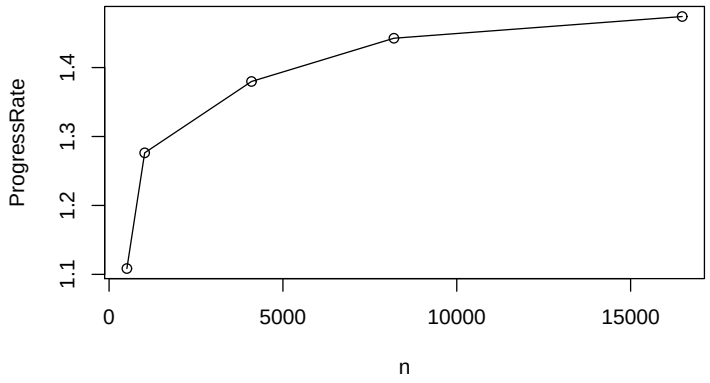


図 3.20: 正方形入力画像による性能向上比

図 3.20 から改良アルゴリズムは入力画像画素が多い場合により高い性能向上が見込め, その最大値は既存アルゴリズムと比較して 1.5 倍 (150%) 程度の性能向上が期待できる結果が得られた.

3.11 考察

拡張辺モデルの導入により次辺決定アルゴリズム処理は，既存アルゴリズムの現座標 (x,y) と探索方向 (search direction) の 2 変数が作業変数として必要であったが，次辺差分配列の導入により辺番号 (en') のみの 1 変数に削減できた．

次辺差分配列により，既存アルゴリズムの条件分岐がなくなり大幅な簡略化，各探索方向の処理速度が均一化が達成できた．既存アルゴリズムでは入力画像の探索方向成分分布による処理速度の依存度が高い．改良アルゴリズムでは各探索方向の処理速度が均一化されており，処理速度の入力画像への依存もより回避できた．

実装による検証においても，既存アルゴリズムと比較して 1.5 倍程度の実行速度となる大幅な性能向上の結果が得られた．検証では，探索方向の東西南北が均一な試験的な入力画像を用いたため，この性能向上比は平均値と見なせる結果である．また，作業変数を 1 変数に削減し条件分岐を極力排除したことにより，並列コンピュータへの実装もより容易になった．

第4章　境界値表現の簡素化

4.1　概要

4.1.1　課題

境界値探索アルゴリズム [8] の特性上，出力される境界値多角形は多くの点および辺で構成されている．視覚的にはギザギザにブロック化されたような印象となる (図 4.1).

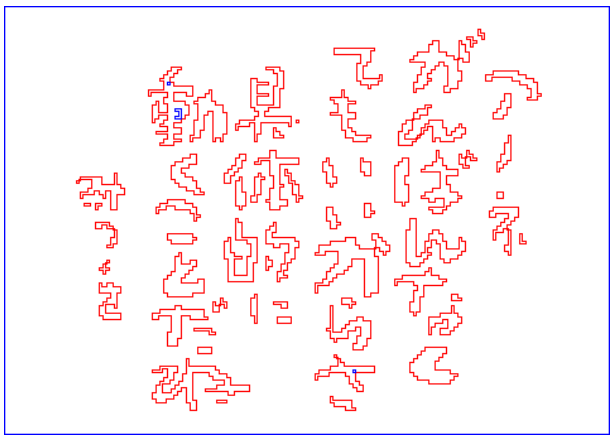


図 4.1: 境界値多角形の出力例

この問題に対して，ベクトル化 (vectorization) による簡素化 (simplification) アルゴリズムについて考察する．

4.1.2　簡素化アルゴリズム

近年，OSS(Open Source Software) の画像処理ライブラリとして OpenCV[12] が著名である．境界値検出およびポリゴン近似の機能も実装されており，ポリゴン近似アルゴリズムには DP(Douglas-Peucker) 近似 [14] が採用されている [13]．DP 近似は，線分による画素の近似アルゴリズムであり，ある直線に対して最遠点を基準にして再帰的に分割していく 2 分割法 (binary decomposition method) である (図 4.2).

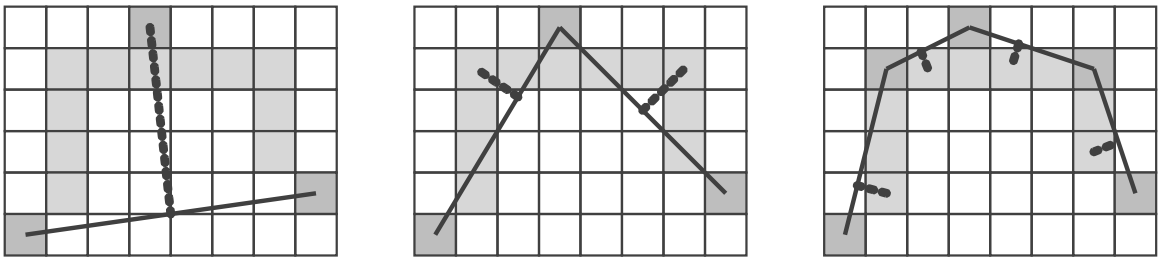


図 4.2: DP 近似による簡素化例

4.1.3 適用

出力される境界値多角形は閉じた多角形であり始点と終点が一致している．始点と終点は境界検出アルゴリズム [8] の基準境界 (Canonical Edge) 点と言われる点である．DP 近似法を適用するは，始点終点が一致しないように多角形を分割し処理する必要がある (図 4.3).

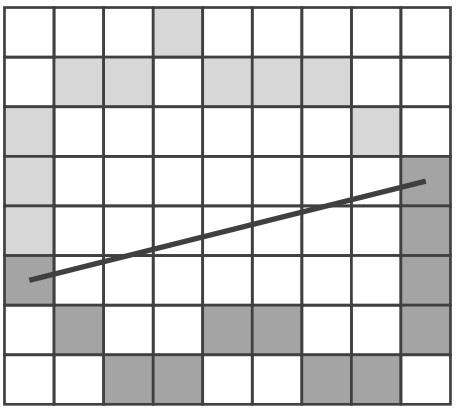


図 4.3: 境界値多角形の 2 分割例

分割は 2 分割とし，一方の線分の始点または終点は基準境界点とした．終点は，始点から最遠点なども候補に挙げられるが，計算量の問題もあり単純に基準境界点から最終点までの中間点とした.

この簡易な中間点選択では閾値以下で簡素化されるべき点が中間点として選ばれてしまう可能性がある．本来は IPAN アルゴリズム [15] のような簡素化の際に特徴的となる点を中間点として選択するのが効果的である．ただし，妥当な中間点の選定は本稿の主題ではないため実装および詳細な検証は割愛した．分割点インデックスの決定アルゴリズムを Algorithm6 に示す.

Algorithm 6 ポリゴン分割インデックス取得

procedure GETDIVIDEPOINTINDEX($polysPointCount$)
  return $ceil((polysPointCount \div 2))$
end procedure

DP 近似法の再帰処理を終了する条件として，直線と最遠点との距離がある閾値を下回った場合とした．境界値出力は閉じた多角形である特徴から，多角形の外接円半径などが閾値の基準として利用できると推察した．DP 近似法をベースとした簡素化アルゴリズムを Algorithm7 に示す.

Algorithm 7 境界値簡素化アルゴリズム

procedure TOSIMPLEPOLYLINES($edgePoints, startIdx, endIdx, thresholdDistance, polyPoints$)
  $maxDistance \leftarrow 0$
  $maxIdx \leftarrow -1$
  for $n \leftarrow (startIdx + 1), (endIndex - 1)$ **do**
    $polyLine \leftarrow edgePoints[startIdx], edgePoints[endIndex]$
    $distance = GetLineToPointDistance(polyLine, edgePoints[n])$
    if $maxDistance < distance$ **then**
      $maxDistance \leftarrow distance$
      $maxIdx \leftarrow n$
    end if
  end for
  if $maxDistance < thresholdDistance$ **then**
    return
  end if
  $polyPoints \leftarrow polyPoints + edgePoints[maxIdx]$
  $ToSimplePolylines(startIdx, maxIdx, thresholdDistance)$
  $ToSimplePolylines(maxIdx, endIndex, thresholdDistance)$
end procedure

4.2 検証

4.2.1 自然画像

境界検出検証時と同様に，最初に自然画像 (図 4.4) を対象に簡素化アルゴリズムを検証した.

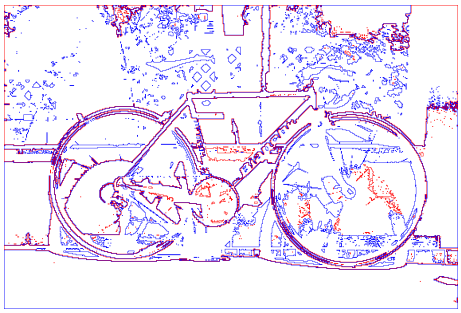
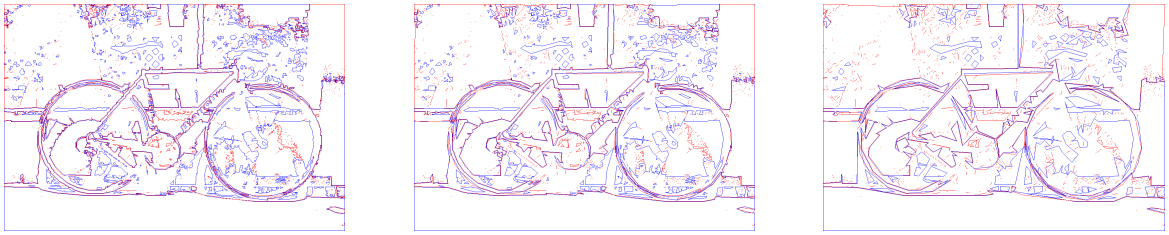


図 4.4: 自然画像の簡素化入力境界例

本簡素化アルゴリズムの自然画像への適用例を (図 4.5) に示す.



[a] 閾値 = 2 の適用例　[b] 閾値 = 4 の適用例　[c] 閾値 = 8 の適用例

図 4.5: 自然画像の静的閾値適用例

検証前には，境界値簡素化アルゴリズム (Algorithm7) の閾値として周囲長などの動的な閾値での正規化を考慮していた．しかし，検証を進める上で，必ずしも動的な閾値でなくとも静的な閾値で視覚的に妥当な結果が得られた．(図 4.5).

動的な閾値が不意必要な理由として，図 4.5 の閾値が 8 の場合に結果に見られる通り，ある一定以上の閾値を適用すると簡素化の効果が大きくなり，視覚的に大ざっぱな印象となるためである．他の自然画像についても検証したが，図 4.5 に示すように，閾値が 8 以下の値が入力画像の特徴を識別可能な適切な範囲との結果が得られた．この考察から入力画像に依らず大きな閾値は不要と判断した．結果として動的な閾値設定も不要と判断し，動的な閾値に関する検証は省略した．図 4.5 の統計的な情報を表 4.1 に示す.

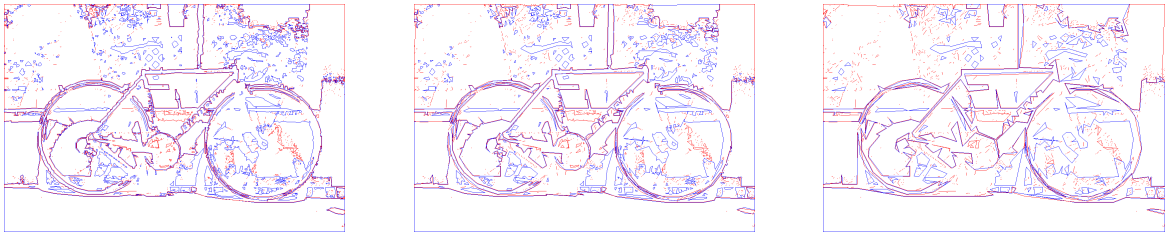
表 4.1: 自然画像の静的閾値適用例

閾値	0	2	4	8
境界値数	1604	1604	1604	1604
最小周囲長	4	2	2	2
最大周囲長	21104	17615	16270	14993
周囲長中央値	4	2	2	2
周囲長平均	48.45	39.27	35.71	32.36
総周囲長	77718	62982	57287	51898
周囲長標準偏差	590.89	493.14	456.12	420.89

簡素化後の周囲長の中央値が 2 となった．これは簡素化により境界値が 2 点から構成される多角形，つまりは直線に変換された結果が多い事を示す．

4.2.2 直線消去

表 4.1 により，境界値簡素化アルゴリズムにより直線に変換された多角形が多く存在する結果を得た．直線は面積が 0 の多角形であるため簡素化により簡略化できる情報と言える．自然画像 (図 4.4) の結果から，直線に変換された多角形を消去した結果を図 4.6 に示す．



[a] 閾値 = 2 の適用例　[b] 閾値 = 4 の適用例　[c] 閾値 = 8 の適用例

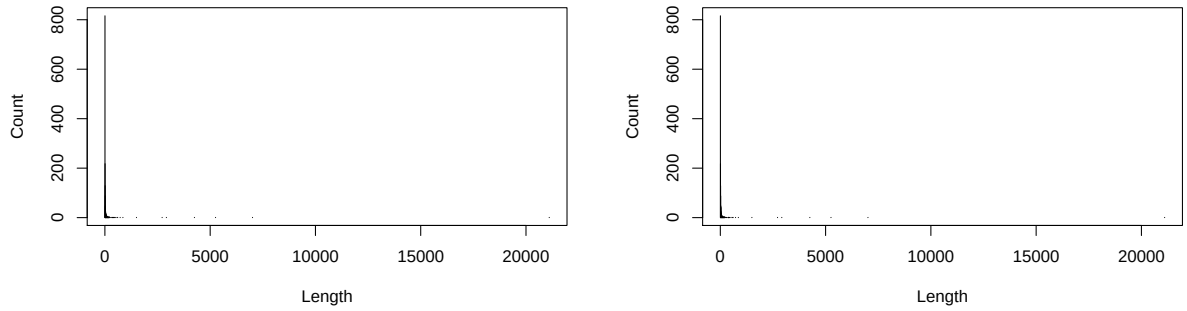
図 4.6: 自然画像の直線消去適用例

図 4.6 の結果から，簡素化の過程で発生した直線を消去しても入力画像の特徴は損なわれない結果が得られた．同様に直線消去した図 4.6 の統計的な情報を表 4.2 に示す．

表 4.2: 自然画像の直線消去適用例 (L* = 70)

閾値	0	2	4	8
境界値数	1604	429	249	99
最小周囲長	4	8	11	25
最大周囲長	21104	17615	16270	14993
周囲長中央値	4	24	34	78
周囲長平均	48.45	135.58	203.15	427.33
総周囲長	77718	58167	50584	42306
周囲長標準偏差	590.89	946.84	1143.17	1644

直線消去の結果，閾値が大きくなるにつれ境界値数および周囲長は減少傾向となる．反対に周囲長の平均および中央値は増加傾向である．この結果により直線消去ありの簡素化効果が統計的にも確認できた．表 4.2 の周囲長と個数の関係グラフを図 4.7 に示す．



[a] 閾値 = 2 の分布　　[b] 閾値 = 8 の分布

図 4.7: 自然画像の直線消去総周囲長分布例 (L* = 70)

簡素化後も中央値に偏ったポアソン分布の傾向は同様である．ただし簡素化により中央値付近により多くの境界値多角形が集中する結果となった．

4.2.3　手書き画像

次に，境界検出検証時と同様に手書き画像 (図 4.8) を対象に本アルゴリズムを検証した．

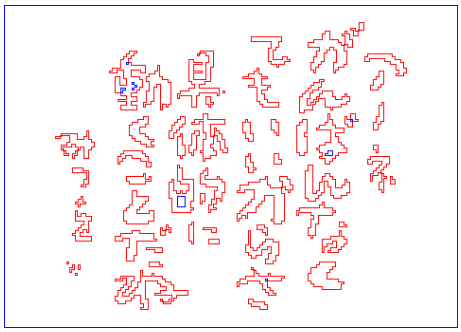
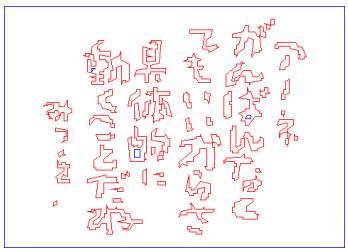
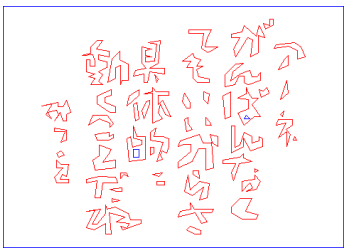


図 4.8: 手書き画像の簡素化入力境界例

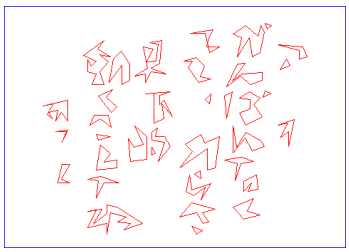
検証は自然画像での結果を踏まえて，静的閾値の適用と直線消去も合わせて実施した．本簡素化アルゴリズムの自然画像への適用例を (図 4.9) に示す．



[a] 閾値：2



[b] 閾値：4



[c] 閾値：8

図 4.9: 手書き画像の静的閾値適用例

手書き画像についても，自然画像検証結果と同様に，閾値が 8 以下の値が入力画像の特徴が識別可能な結果となった．図 4.9 の統計的な情報を表 4.3 に示す．

表 4.3: 手書き画像の静的閾値適用例

閾値	0	2	4	8
境界値数	70	60	57	38
最小周囲長	8	17	16	25
最大周囲長	1636	1636	1636	1636
周囲長中央値	68	79	88	132
周囲長平均	135.65	144.48	137.58	169.95
総周囲長	9496	8669	7842	6458
周囲長標準偏差	209.37	215.82	217.87	252.14

統計的な情報も，自然画像検証結果と同様に境界値数および周囲長は減少傾向となるが，周囲長の平均および中央値は増加傾向であった．

4.3 考察

簡素化は DP(Douglas-Peucker) 近似 [14] をベースにアルゴリズムを考案した．簡素化の閾値については，検証から自然画像および手書き画像でも静的な値での効果が確認できた．この結果から簡素化は動的な閾値 (引数) は不要なアルゴリズムで処理できる事が推察される．また簡素化の過程で発生した直線を消去しても入力画像の特徴は損なわれない結果が得られた．

第5章　まとめ

5.1　考案の検証結果

- 境界値探索アルゴリズム [8] を実装し各種入力画像を検証した．検証結果から境界検出となる 2 値化の閾値を高速に求めるアルゴリズムを考案した．
- 拡張辺モデルの導入により境界値表現探索アルゴリズムの次辺決定処理の改良アルゴリズムを考案した．既存アルゴリズムと比較し作業変数および条件分岐削減により実装ライン数を大幅に削減，各探索方向の処理速度を均一化できた．また実装による性能検証においても大幅な速度向上の結果も得られた．
- DP(Douglas-Peucker) 近似 [14] をベースに境界値表現出力多角形の簡素化アルゴリズムを考案した．検証結果から簡素化の閾値について入力画像の種類によらず固定値での効果を確認できた．また簡素化の過程で発生した直線を消去しても入力画像の特徴は損なわれない結果を得た．

5.2　今後の展望

- 本稿では境界値探索アルゴリズム [8] は次辺決定処理改良処理部のみに注力してアルゴリズムを改良した．他処理部のアルゴリズムの改良の余地については今後の課題する．
- 今回，境界値探索アルゴリズム [8] の検証は一般的なコンピュータ上のみでの実装に留まった．最終的にはアルゴリズム改良の対象モデルとした OpenCL[17] などの並列コンピュータモデル上で更なるアルゴリズム検証については今後の課題とする．
- 近年，画像処理および認識 OSS(Open Source Software) として著名な OpenCV[12] では本稿とは別の境界値表現探索アルゴリズム [19] をベースに実装されている [13]. 今回の境界値検出アルゴリズムの改良結果を踏まえて，OpenCV などで実装されている既存アルゴリズムと比較検証については今後の課題する．有効性を確認した上で OSS へ貢献し多くの利用者に本稿の成果が利用されることを期待したい．

謝辞

本稿を進めるにあたり，厚くご指導を頂いた主指導教員の浅野哲夫教授，多くのご助言を頂いた副指導教員上原隆平教授に深く感謝致します.

参考文献

[1] Gordon E. Moore : ”Cramming more components onto integrated circuits”, Electronics Magazine, 19 April 1965

[2] Charles Weir, James Noble : ” 省メモリプログラミング”, (安藤 慶一 訳), ピアソン・エデュケーション, pp1-12, 2004.

[3] Herb Sutter : ”The Free Lunch Is Over - A Fundamental Turn Toward Concurrency in Software”, Dr. Dobb’s Journal, 2005.

[4] Steve Canon : ”Inside the Accelerate Framework for iOS”, Session 209, WWDC, 2011.

[5] Apache Hadoop : http://hadoop.apache.org/

[6] 定兼邦彦 : ”大規模データ処理のための簡潔データ構造”, 情報処理, 第 48 巻第 8 号, pp899-902, 2007

[7] ディジタル画像処理編集委員会 : ”ディジタル画像処理”, 画像情報教育振興協会, pp177-190, 2004.

[8] Tetsuo Asano, Sergey Bereg, Lilian Buzer : ”Small Work Space Algorithms for Some Basic Problems on Binary Images”, Proc. of the 15th International Workshop on Combinatorial Image Analysis, pp.103-114, Austin, USA, November, 2012.

[9] libGD : http://libgd.bitbucket.org

[10] GIMP : http://www.gimp.org

[11] H. Freeman : ”On the classification of line-drawing data”, Models for the Perception of Speech and Visual Form, pp.408-412, 1967.

[12] OpenCV : http://opencv.org

[13] Grary Bradski, Adraian Kaebler : ”詳解 OpenCV コンピュータビジョンライブラリを使った画像処理・認識”, (松田 晃一 訳), オライリー・ジャパン, pp255-266, 2009.

[14] D. Douglas, T. Peucker : ”Algorithms for the reduction of the number of points required for represent a digitised line or its caricature.”, Canadian Cartographer 10, pp112-122, 1973.

[15] D. Chetverikov. Zs.Szabo : ”A simple and efficient algorithm for detection of high curvature points in planar curves.”, Proceedings of the 23rd Workshop of the Austrian Pattern Recognition Group, pp175-184, 1999.

[16] Clay Breshears : ”並行コンピューティング技法 - 実践マルチコア/マルチスレッドプログラミング”, (千住 治朗 訳), オライリー・ジャパン, pp255-266, 2009.

[17] OpenCL : http://www.khronos.org/opencl/

[18] 株式会社フィックスターズ : ”改訂新版 OpenCL 入門 1.2 対応”, インプレスジャパン, pp27-43, 2012.

[19] S. Suzuki, K. Abe : ”Topological structural analysis of digital binary images by border following.”, ”Computer Vision, graphics and Image Processing”, 1985.