GPGPU を用いた地形データの特徴量の並列計算

法政大学　理工学部　経営システム工学科

経営数理工学研究室

指導教官　五島洋行

2014 年 2 月

10x4026　　加藤諒磨

論文要旨

本研究では，まず大島の標高 10m ごとの数値標高モデルから座標と角度を抽出し，そのデータから特徴量の計算を行う実験を行い, CPU のみで計算を行った場合と GPGPU で計算を行った場合で比較し，その差を測った.

しかし，実際の標高ごとの面積や周長は調べることができないため，次に，楕円の座標データを用いて，面積の計算を行い，その計算精度を測った． また，その際にモンテカルロ法でも面積を計算し，時間と計算精度を比較した.

結果は，今回用いた大島の 10m 座標標高データのデータ数では CPU の方がわずかに速いという結果になった．しかし，座標数を増やすことによって, GPGPU の方が速く計算できるという結果が出て，また，同計算時間でモンテカルロ法と線積分の結果を比較すると，常に線積分で計算を行った方が精度が良いという実験結果がでた.

この結果，座標数が多く，精度を必要とするような面積の計算では本研究の提案している手法を用いると高速で，精度のよい計算が行えるということが判明した.

目次

1 序論

1.1 背景

この研究の背景は 2 つの問題から成り立っている.

1 つ目に, 近年自然災害が頻繁に起こっているという事実があり, それに対する対策が十分なされていないという問題がある. 例えば, 近年の大きな災害と言えば, 2011 年 3 月 11 日にあった東日本大地震があげられる. その被害は想定を遥かに超え, 想定している震度に耐えられるとされていた物件でさえもいくつも崩壊してしまった. そして, その原因は予期せぬ自然災害の対策を怠ったことにあったと考えられる. しかし, 現実には起きる前に被害を予測して, シミュレーションを行い, 対策を講じることは震源地やその大きさ等の要素を事前に予測することができないと難しく, 現在の技術ではその予測はまだ不可能であるとされている. そこで, 今は自然災害が起きた後どうするか, ということに注目すべきである.

そして, 今注目されているのがハザードマップという非常時に使われる地図であり, これは国や各自治体で既に配布されていて, 主に自然災害の発生時に利用されている, しかし, この地図は東日本大地震前にも既に配布されていたはずで, 多くの人が存在を知っていたにも関わらず, 実際にはあまり活かされることがなかった, それは常に見られるような状態にしていなかったという理由もあるが, 当時想定していた被害予測と, 実際の被害があまりにもかけ離れていたからという理由もある. そこで, 震度も想定出来ていないハザードマップより, 震度が判明してからハザードマップを作成し, それを多くの人が見ることが出来れば, もっと活躍するはずである.

また, その被害を予測するには地形データをコンピュータで計算することが必要不可欠である, そして, もし, 想定外の大きな災害が起きた場合に津波の被害がどこまで及ぶか等の地図を即座にテレビやインターネットで公表することができれば自然災害での人の被害は激減するはずであり, そのためにはコンピュータでの計算を高速化することが必要になる. そこで, 次にコンピュータでの計算の速さの問題について解説する.

また, 近年, 家庭用のコンピュータが普及し, スマートフォン等を含めたその所有率は 80%に近いとされている[1]. また, それに伴いユーザーの用途も多様化し, ライトユーザとヘビーユーザでは利用目的が違い, 求める仕様も大きく違ってきている. それに合わせてコンピュータの性能も初期の物と比べて年々そのスペックも上がっている. ここではスペックを CPU 等の能力をまとめた, コンピュータの総合的な能力とする. しかし, 近年では CPU の処理能力の成長が発熱等の物理的な理由で頭打ちになり, 性能の面を伸ばすわけではなく, コアをマルチコアに変えて能力を上げようとしている, しかし, ソフトウェア側の対応がなされていないのが現状であり, このままではいつかユーザーの要求する能力に対応できなくなってしまうという問題がある.

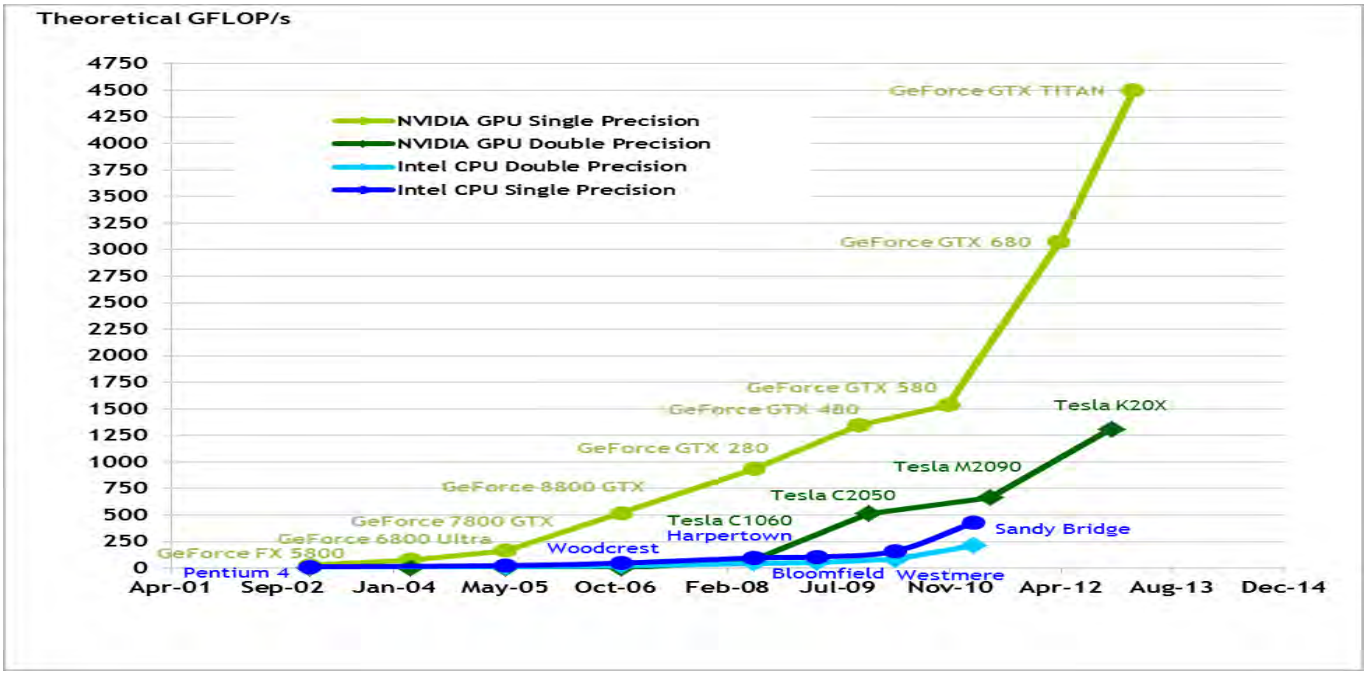

図 1-1　CPU と GPU の帯域幅の成長率[2][1]

そこで最近では，まだ能力の伸び代が十分にある，GPU(Graphics　Processing　Unit)というパソコンのパーツが注目されている．その GPU と CPU との成長の度合いの比較を図 1-1 に示す．図 1-1 の黄緑のグラフが通常の GPU で，緑のグラフが GPGPU 専用に作られた GPU であり，薄青と青は CPU のグラフである．図から最新の GPU では CPU と比較して性能に何倍もの差が開いていることがわかる．

また，その計算能力を活かすために，GPU を使ったグラフィック以外の通常のプログラミングにも注目が集まっている，本来 GPU はグラフィック処理専用のパーツであったが，GPGPU(General-purpose computing on graphics processing units)というグラフィックの分野以外にも GPU を用いる概念も広まりグラフィックの処理以外にも GPU を使おうとする流れができ，既に様々な分野に用いられている，例えば，東京工業大学では TSUBAME　2. 5 というスーパーコンピュータに GPU を導入して天気予報や粒子の計算等の様々なシミュレーションを行っており，TOP500 というスーパーコンピュータを独自に評価したランキングでは 2013 年 11 月現在では世界 11 位にランクインしている．その他にも多くの研究が行われており，GPGPU を利用する利点は十分にあることがわかる．

そのため本研究では GPGPU を用いて，ハザードマップを作成するときに必要になる面積を求める．特に今回は GPGPU を用いて実在するデータの面積の計算を行い，その計算時間を短縮することが可能かを明かすことを目標とする．

[1]　NVIDIA の CUDA　TOOLKIT　DOCUMENTATION より　2014 年 1 月現在のデータ

2　背景知識

2.1　関連技術，知識

2.1.1　ビデオカード

1990 年代から, GUI(graphical　user　interface)というインターフェイスが広がり始めた, GUI を使うと従来の仕様であったテキストのみでの処理とは違い，モニターを使用することができ，マウス等のデバイスを使い視覚的にコンピュータを操作することができるようになった．このことによりモニターに何かを描画するということが増え，また満足にその能力を利用しようとなると当時の標準であった 640×400 ドットという描画範囲では不十分だった，それを解決するためにビデオカードがパソコンに搭載されるようになった．ビデオカードは描画範囲を広げるだけではなく，描画の速度も高速化することができるようになり，すぐにほとんど全てのコンピュータに搭載されるようになった[3]．ビデオカードは現在でも使われていて，最近では求める用途によって使うビデオカードが変わり，用途によっては 1 つのパソコンに複数枚搭載されるほど重要な存在になっている，さらに，ビデオカードはコンピュータだけではなくゲーム機等の描画が必要な機器には必ず搭載されており，近年ではノートパソコンやNexus7などのタブレットにも搭載されるようになっている．

2.1.2　GPU/GPGPU

GPU(Graphics　Processing　Unit)とはNVIDIA 社が Geforce256 発売時に提唱し始めた単語で，ビデオカード上の VRAM(Video　Random　Access　Memory)を利用するための集積回路であり，マザーボード上の CPU のような存在である．

GPU が誕生する以前では 3DCG の描画は CPU のみで行われていたが, 3DCG には影をつける機能や物理演算を同時に多くの物体に作用させる機能等の様々な機能があり，その機能を十分に使うには処理能力が十分ではなかった．そこで当時は，映画や CAD 等の CG を作成する際には 3D のグラフィック機能に特化した 3D 専用ワークステーションを利用していた，しかし，ワークステーションは性能がいいが，値段は高く一般のコンピュータやゲーム機に入れられる程の大きさではなかったため，パソコンやゲーム機では CPU による処理しかできず，パソコン用の 3DCG のソフトウェア等の実現は実現しづらかった．そこで, NVIDIA はパソコンで 3DCG の作業を行えるようにするため, Geforce256 という世界初の GPU を完成させた．これにより，パソコンでも映画や CAD の CG 作成を行うことが可能になり，ゲーム機でも昔より格段に綺麗な 3DCG が利用可能になった，また, GPU とビデオカードの能力の発展はめざましく，図 2-1 にある通り 5 年でその処理能力は約 5 倍以上になっている．

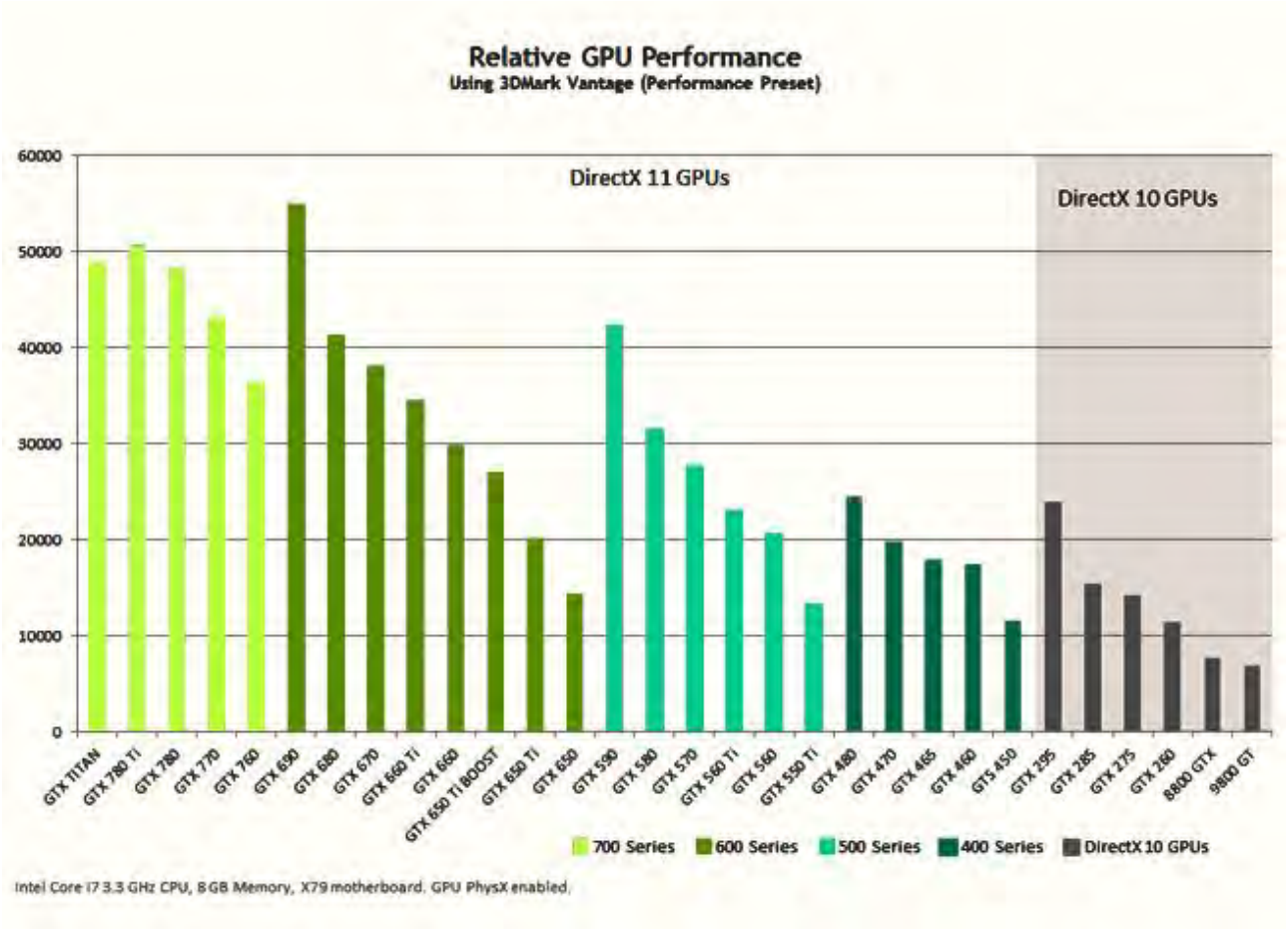

図 2-1　GPU の性能の進歩[4]

次に，GPU が搭載されているビデオカードの構造について解説する，ビデオカードは新しい機種が発売される度や，発売している会社によってその構造は変化しているが，大体の部品は共通している，その部品の構造はビデオカード全体の熱暴走を防ぐための冷却装置，主な処理を行うためのメモリである VRAM(ビデオメモリ)，ビデオメモリでの処理を管理する GPU，ビデオカードを PC と接続するためのインターフェイス，電源の供給を受けるための電源コネクタ，モニターに映像を出力するための出力端子といった部品からなり図 2-2 のような構造になっている，その能力によって求める電源が違う，また GPU は内部で CPU のデュアルコア等のマルチコアのようにコアで分けられていて，それを CUDA コアと呼び，その数は最新の GeforceGTX780Ti では 2880 個も搭載されていて[4]，さらに実際の処理はそのコアの中でさらに細分化されて行われている．

そして，現在では GPU の処理能力の速さ，今後の伸び代に目をつけ GPU を利用した GPGPU(General-purpose computing on graphics processing units)という技術にも注目が集まってきている，これは GPU をグラフィックのみで利用するだけでなく，様々な汎用的な目的で使うという技術である，GPU は前述の通り多数のコアを持っているため，多数の動作を並列的に行うことを得意としている，そこで GPGPU では複数の計算の処理を分散させ，並列的に処理させることで一般的なプログラムの処理も高速化を目指して研究されている．GPGPU は近年では様々なところで使われており，例えば MATLAB では GPU が導入されている PC で使うための関

数がいくつか用意されている，また, NAMD という分子の動きをシミュレーションするソフトウェアでもGPGPUは利用されており，特にこのような同時に多数の物体の座標の動きを計算しなくてはならない研究は GPGPU で高速化できるとされている.

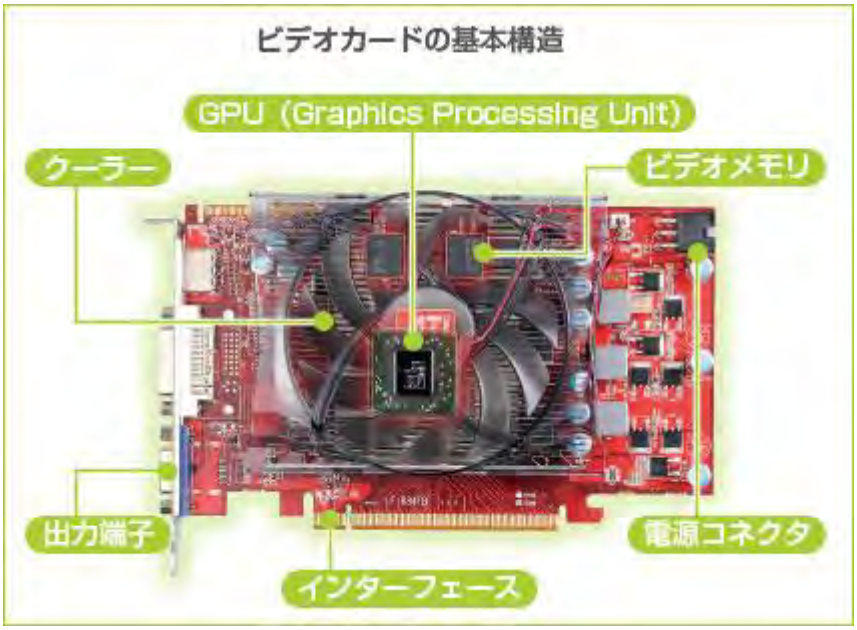

図　2-2 ビデオカードの基本構造[5]

2. 1. 3　CUDA/OpenCL

NVIDIA は 2006 年にグラフィックのみでなく汎用的なプログラムを動かせる GPU を開発した，この GPU で動かす，専用のプログラムを開発するために NVIDIA は同時に CUDA という GPU のための統合環境を発表した, CUDA はNVIDIA 製の GPU で利用できる環境でWindows ならVisual Studio に組み込んで C++を拡張することで併用して使うことが出来る, CUDA を用いると, GPU の指定した箇所に指定した処理を行わせることが出来る.

また，類似の技術として OpenCL がある, OpenCL を簡潔に表現すると「ヘテロジニアスな並列計算機環境に適した並列プログラミングのためのフレームワーク」[6]であり，要するに，様々なプロセッサが搭載された環境で並列計算をすることができるプログラミング言語であるということである, OpenCL も CUDA と同様に GPU を扱うためのプログラミング言語であり, CUDA のライブラリを利用して C++を拡張して用いることができる, CUDA との主な違いは使う GPU を限定しないことにある，さらに上手く使うことができれば GPU に限らず，マルチコア CPU でも, 2 台のパソコンでも並列的に計算することが出来るようになる，要は NVIDIA 社の GPU のために作ったプログラムをそのまま AMD 等の別のメーカーが作った GPU や PS3 の中にある Cell 等でも同じように使うことができるということである，このことによって複数の言語を覚える必要も，新しくプログラムを作成する必要もなくなるという大きなメリットがある.

両者が一般的な C++のプログラムと違う箇所はいくつかあるが，主要なポイントは図 2-3 のよう

にプログラムがホスト（CPU）側とカーネル（GPU）側に処理が別れることとプログラムの処理を並列化できる点である，ホストコードはメインのプログラムで処理の順序や計算部分以外のGPU に処理を割り当てたり，データを読み込んだり，その他の処理を行わせ，カーネルコードでは並列処理を行う部分だけを記述し，そこで処理を行わせ値をホストに返す．

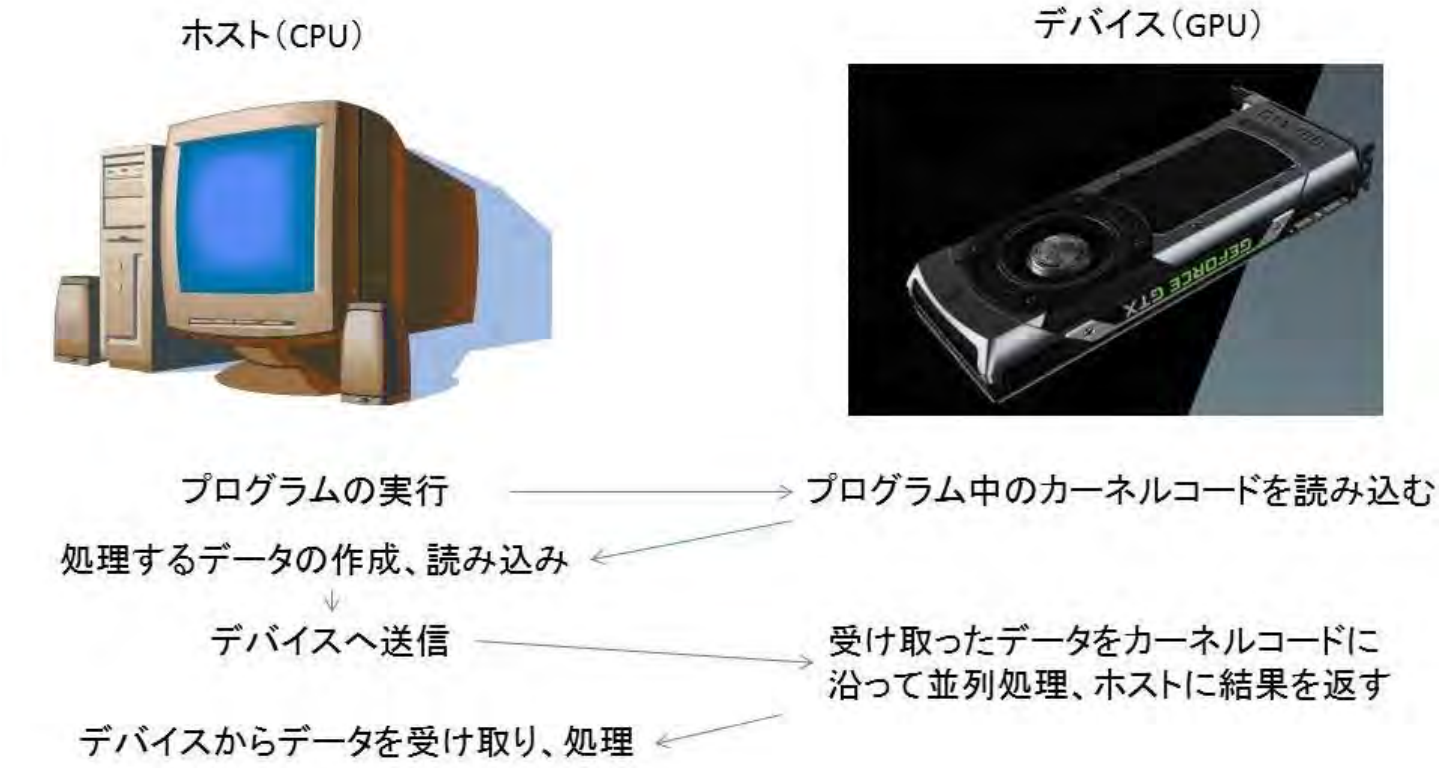

図　2-3　GPU を用いたプログラムの処理手順[2]

2. 1. 4　地理情報システム(GIS)

「地理情報システム（GIS : Geographic　Information　System）は，地理的位置を手がかりに，位置に関する情報を持ったデータ（空間データ）を総合的に管理・加工し，視覚的に表示し，高度な分析や迅速な判断を可能にする技術である．」[7]つまり，地形のデータを測り，それを利用する技術のことを指している．しかし，そのデータも様々で三次元の座標データ，人口，建物のデータ，土地の値段に関するデータ等のとても多くの種類のデータがあり，そのデータの種類に合わせて用途も数多くある．例えば，政府が都市開発を行う際に利用したり，各自治体が避難経路や危険な場所を指定するハザードマップを作る際に利用したり，業者が商業施設を建てる位置を決める際に利用したり，個人がカーナビに入っているデータを利用したりと使い道は無数にある．

その中でも今回は三次元の座標データを用いたハザードマップについて考える，ハザードマッ

[2]　デバイス(GPU)の部分の画像は http://www. nvidia. co. jp/gtx-700-graphics-cards より

プとは台風や地震などで発生する被害を予測し，自然災害が発生した場合の危険地域や避難経路や避難所が描かれている地図であり，地図の種類も自然災害ごとに洪水ハザードマップ，内水ハザードマップ，津波ハザードマップ等多く作られている．例えば，小金井市作成の法政大学周りの総合的なハザードマップは図 2-4 のようになっている．

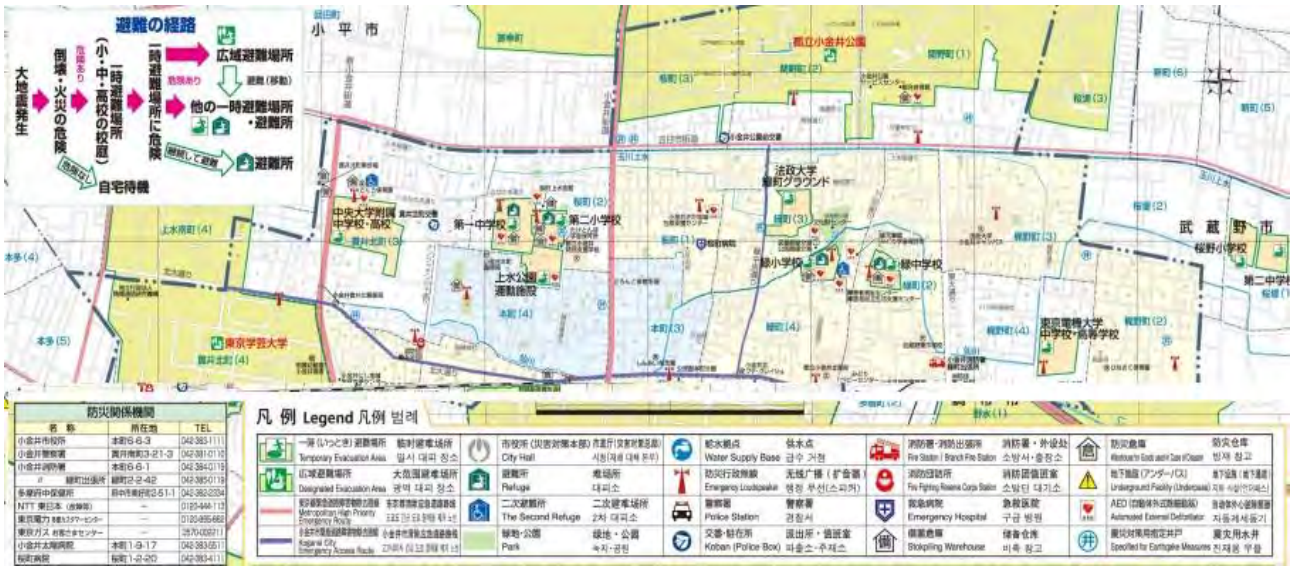

図　2-4　法政大学周りのハザードマップ[8] [3]

三次元の座標データが活かせるハザードマップとして，山間部のような凹凸の大きい地形での洪水ハザードマップがその一例に上げられる，座標データから精度の高い面積や体積を求めることが可能であれば，雨がどの程度降ると氾濫して洪水になるかがわかり，どこが危険なのかが判断でき，ハザードマップの作成に役に立つためである，今回はその一歩として大島のデータの三次元座標データを扱った．

2. 2　関連研究

本研究の関連研究として，「GPU を用いた行列演算の高速化」(桑山 2010)[9]をあげる，桑山の研究ではタイトルの通り行列の計算を, GPU を用いて高速化を行っている．この研究の結果として，64*64 の行列以上の大きさの行列の乗算では, CPU のみの場合と比較して処理の高速化が行えることがわかり, 2048*2048 の行列の乗算の場合では CPU のみの場合と比べて約 600 倍の高速化が行われている．しかし, 64*64 以下の場合は CPU のみの場合の方が処理時間は短い，この研究から GPU を用いた並列化では，計算回数が増えれば増えるほど高速化が行え，計算回数が少ないと CPU と変わらない，またはそれ以下の結果がでてしまうということがわかった．

また，各自治体の研究で，本研究で用いたような数値標高モデルを使い，災害の予想を行っている研究として，「数値標高モデルを用いた傾斜地カンキツ園の斜面崩壊危険度の評価」(福本，島崎，吉村 2008)[10]がある，福本らの研究では数値標高モデルから，傾斜度，集水度というパラメータを

3　小金井市公式 HP のハザードマップより，画像が大きいため法政大学周りと凡例の部分を抜粋

作成し，そのデータから台風等による災害で，どの辺りの斜面が崩壊しやすいかを調べ，どこに対策を施すべきかを探る研究である，この研究より，数値標高モデルを災害対策に活かすことは十分に可能であるということがわかる．本研究では体積から，窪地に貯まる水量を求める，という研究の一歩として面積を求めているため，類似の研究としてあげる．

3 提案手法

3.1.1 並列化の手法の評価手順

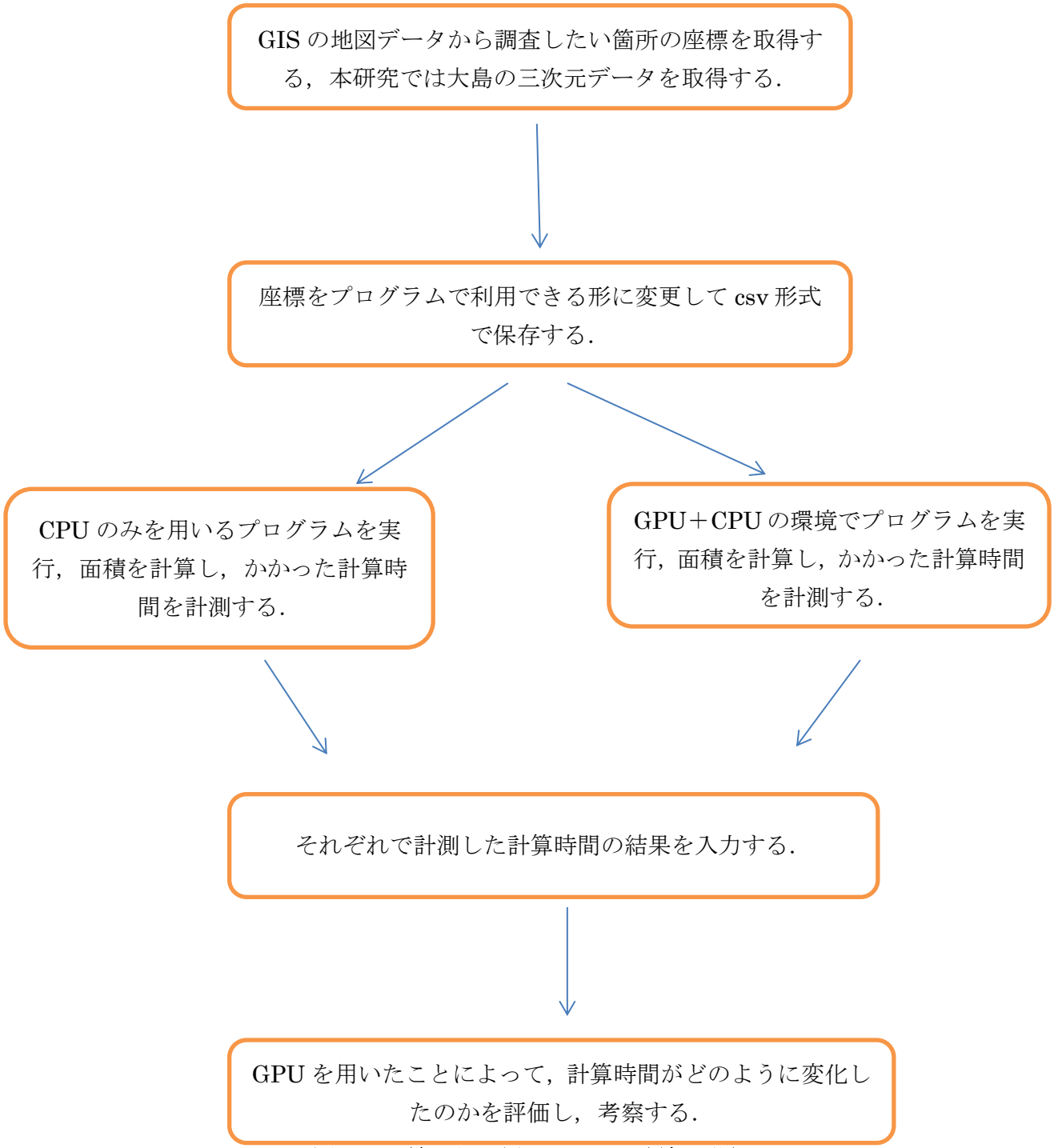

図 3-1　並列化を用いたことの評価手順

図 3-1 にあるように，まず, GIS を用いて，大島の数値標高モデルから大島の座標を抽出し，その形式を，プログラム中で利用するために，ヘッダー行, x 座標, y 座標，角度という要素にわけ, csv

ファイルに保存する．そして，それを CPU のみの場合と CPU＋GPU の場合の 2 つの環境でそれぞれ特徴量の計算を行い，それぞれの計算にかかった時間を計測し，考察する．

3. 1. 2　線積分のアルゴリズムの評価手順

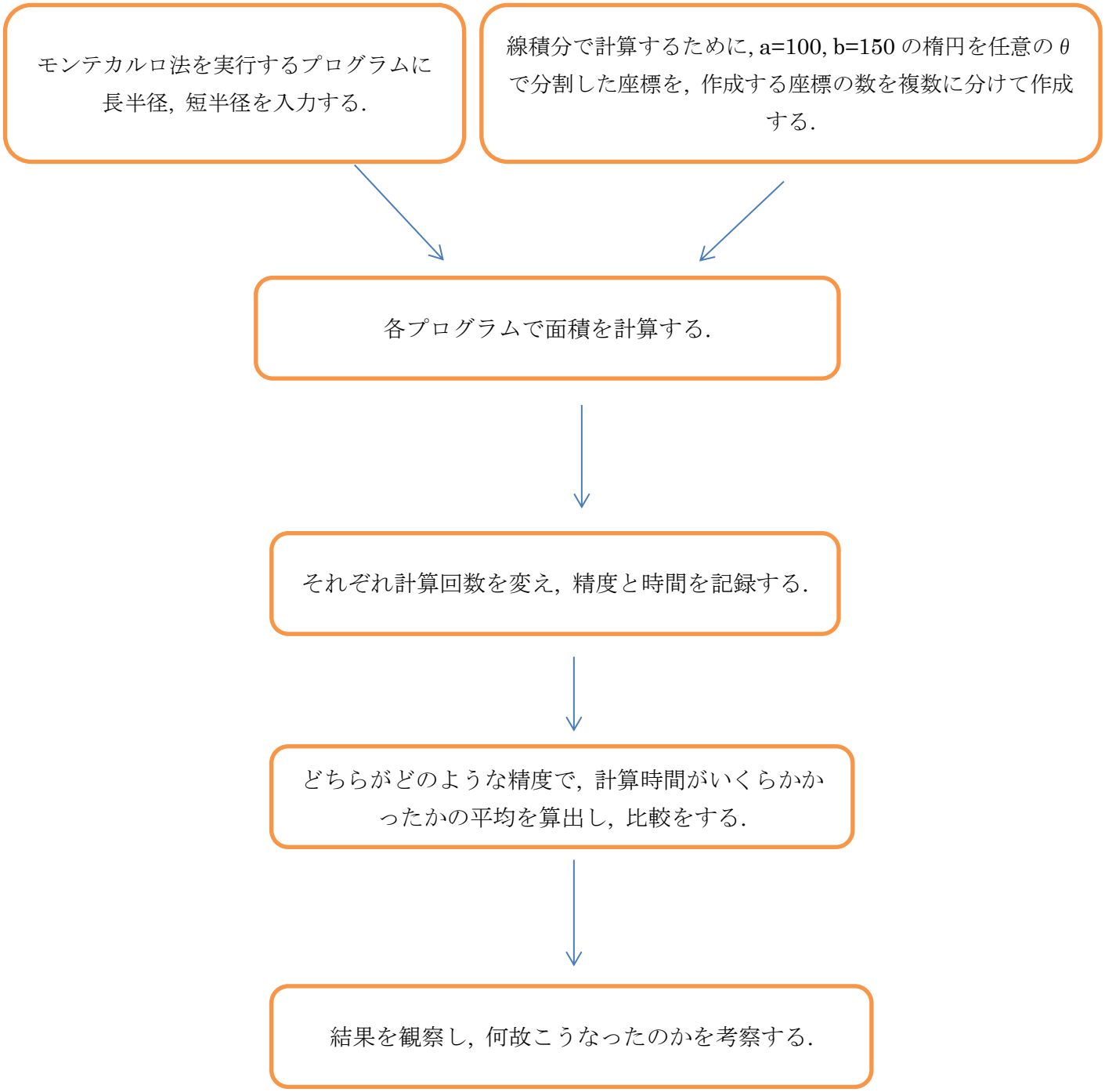

図　3-2 線積分を用いたアルゴリズムの評価手順

図 3-2 にあるように，まず線積分での面積の計算を行うために，別のプログラムで楕円の座標の作

成を行い，モンテカルロ法のプログラムには長形と半径，楕円の公式を入力する，次に，作成したデータを元に各プログラムで面積の計算を行い，計算時間を測る，これを座標数や発生させる乱数の数を変えて，複数回実行し，その結果を時間と精度に分け，評価と考察を行う．

3. 2 実行したプログラムの原理

3. 2. 1　線積分の特徴量の計算

まず，プログラム内で用いる特徴量の計算方法について解説する．ここでは特徴量を面積と周長とするが，特徴量を計算するために線積分の概念を用いるため，ここではまず線積分について説明する．線積分とは主に物理学で使われる積分で，向きが決められている経路に沿って求める範囲を細かく刻んで計算する数学の概念である．この積分の特徴は経路が決められており，定めた経路によって，計算する範囲は同一である必要があり,さらに，今回はグリーンの定理を適用し，周回積分を行う，また，今回は面積と周の長さを計算するため，始点と終点が同じで無くてはならない，このような線積分を周回積分という．この方法で面積を求めた理由は，後述するモンテカルロ法と比べて計算回数が少ないからであり，また，座標間の範囲を細かく刻むことで，他の同回数の計算回数のアルゴリズムと比べて計算の精度も高くなることが期待されるからである．ここで座標間の補間は円弧近似で行うとする．また，線積分を用いる際には座標間の補間が必要であるため，本研究では座標間の式は円弧で近似している，今回は座標(x,y)と座標間の角度 θ のみが入力データとして与えられている．これらのデータから面積を求める方法を以下で説明する．今回の計算で使った文字を図 3-1 に示す．

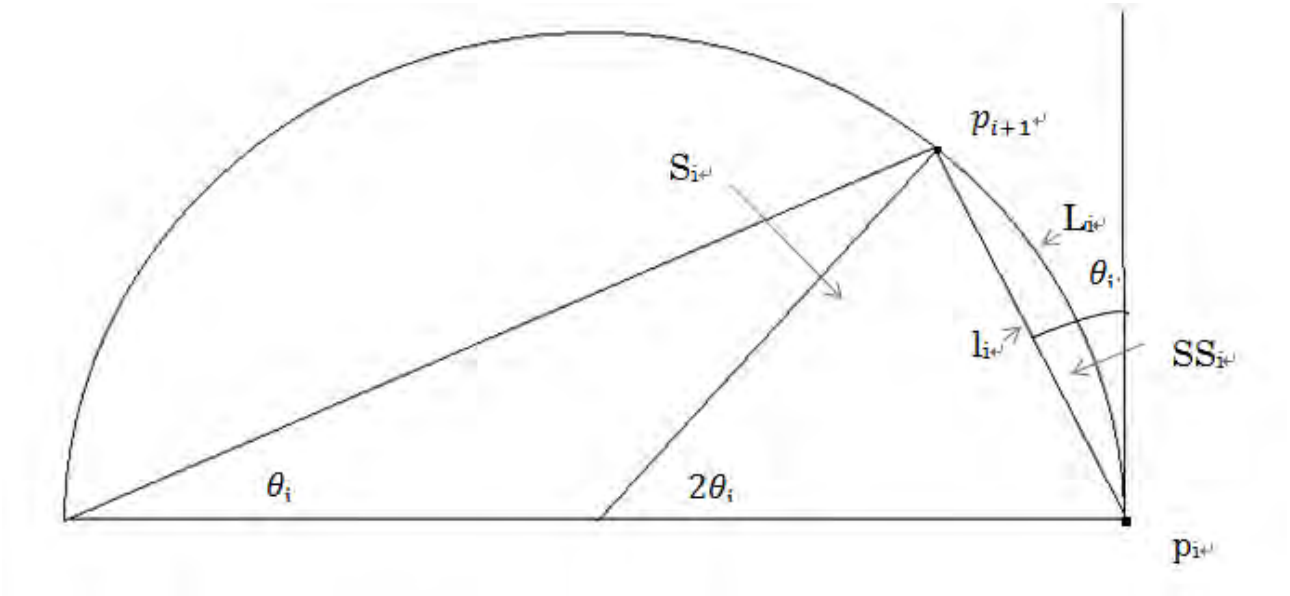

図 3-3 p_i と p_{i+1} 間の計算に必要な文字の説明

i 番目の面積を求めるとする．制御点 2 点を $p_i(x_i, y_i)$, $p_{i+1}(x_{i+1}, y_{+1})$とし, 2 点を円弧の座標としたときに中心点から p_i に引かれた線分と p_i, p_{i+1} の座標間の線分とがなす角を θ_i とすると, 2 点が円弧を

挟んでなす角は $2\theta_i$ となる．よって，中心と制御点 2 つを結んで作られる三角形の面積 S_i は三角形の面積の公式により次のように表される．

$$S_i = \frac{1}{2}(x_i y_{i+1} - x_{i+1} y_i) \tag{1}$$

次に三角形の外の弓形の円弧の面積 SS_i を求める，その面積は半径を r とすると

$$SS_i = \frac{r^2}{2}(2\theta_i - \sin 2\theta_i) \tag{2}$$

と表すことができる，次に r を求めるために，まず座標間の長さを求め，それを弦として r を求める．座標間の長さを l_i とすると r は以下のように求められる．

$$2r\sin\theta_i = l_i \tag{3}$$

また　$l_i = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$ であるため，これを(3)に代入し，整理すると

$$r = \frac{\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}}{2\sin\theta_i} \tag{4}$$

となり，式(4)を式(2)に代入すると

$$SS_i = \frac{1}{2}\frac{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}{4\sin{\theta_i}^2}(2\theta_i - \sin 2\theta_i)$$

となり，さらにこれを整理すると

$$SS_i = \frac{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}{4}\frac{\theta - \sin\theta_i\cos\theta_i}{\sin{\theta_i}^2} \tag{5}$$

となり，式(1)と式(5)を足すことで求めたい座標間の面積を求めることができる．

また，座標の数が多いと座標間の角度 θ は非常に小さくなる，このため，$\theta^2 \leq e^{-10}$ の場合は(5)を 0 の周りでテイラー展開した式(6)で計算する．

$$SS_i = \frac{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}{6}\theta_i \tag{6}$$

また，周の長さ L も同様に線積分を用い，細かく計算する．i 番目の座標間の弧 L_i は円周の公式より

$$L_i = 2r\theta_i \tag{7}$$

となる，係数として 2 がついているのは，中心角を 2θ としているからである．また，式(7)に式(4)を代入すると，

$$L_i = \frac{\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}}{\sin\theta_i}\theta_i \tag{8}$$

となり，周長も面積と同様に 0 の周りでテイラー展開を行うと

$$L_i = \frac{\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}}{6} \times (1 + \theta^2) \tag{9}$$

最後に，i=0 から i=n-1 までの結果を合計し，面積，周長の値を出す．

次に，プログラムの全体の処理の説明に移る，まずエラー等の処理を省いた簡易なフローチャートを図 3-2 に示す．左側の処理はホストで，右側の処理はデバイスに区分される．

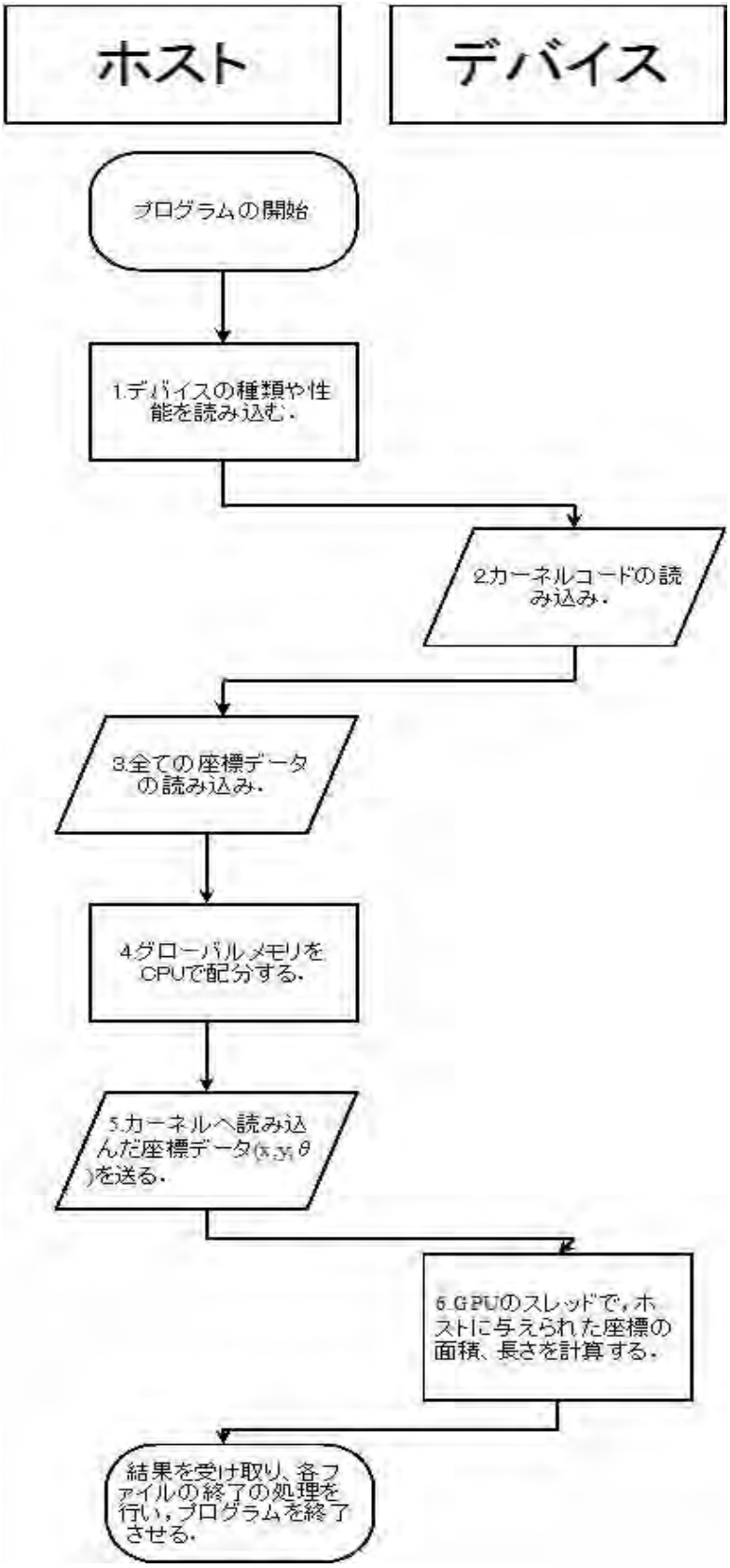

図 3-4　OpenCL における面積計算プログラムのフローチャート

OpenCL で面積計算のプログラムを組むと図 3-2 のようなフローチャートとなる．このプログラムの中で並列化されている箇所は「GPU のスレッドで，ホストに与えられた座標の面積，長さを計算する」という部分である．この部分は他の一般的なプログラムとは特に変わっている点があるため，以下で解説する．

GPU におけるプログラムの処理は多くの処理を同時に行うために，その構造も一般的な処理とは異なっている．まず，その処理部分について説明する，一番小さい処理部分の単位として，GPU で最終的な計算処理を行うスレッドという単位があり，スレッドは関連研究で紹介した CUDA コアという箇所で処理されている，さらにそのスレッドをまとめて格納するものにブロックという単位があり，一つのブロックには 512 個のスレッドを格納することが可能である，さらにそれを格納するものとして，GPU では一番大きい単位であるグリッドがあり，その中には最大で 65535 個のブロックを格納することができ，図にすると図 3-3 のような構造になっている．しかし，実際のプログラムではグリッドは一つの GPU として扱われており，一つの GPU だけが入った PC ではプログラムの中で指定出来るのはブロックとその中のスレッドの番号だけである．

また，2 章の関連知識で述べた通り CUDA や OpenCL でプログラムのコードを書く場合には CPU 側で処理を行うホストコードと GPU 側で処理を行うカーネルコードに分けて書く必要がある．本研究のプログラムでは面積計算ではホストコード上で処理するブロック数とスレッド数を指定し，座標データを GPU に送り，計算処理のみを行わせる，また，並列的に処理を行うとしているが，通常スレッドでの処理は 1 クロックずつ処理時間のずれが生じるため，計算結果を再利用するプログラムでは同期という作業が必要である，今回は面積を計算したあと，その数字を再利用することはないため，詳細な説明は省略する．

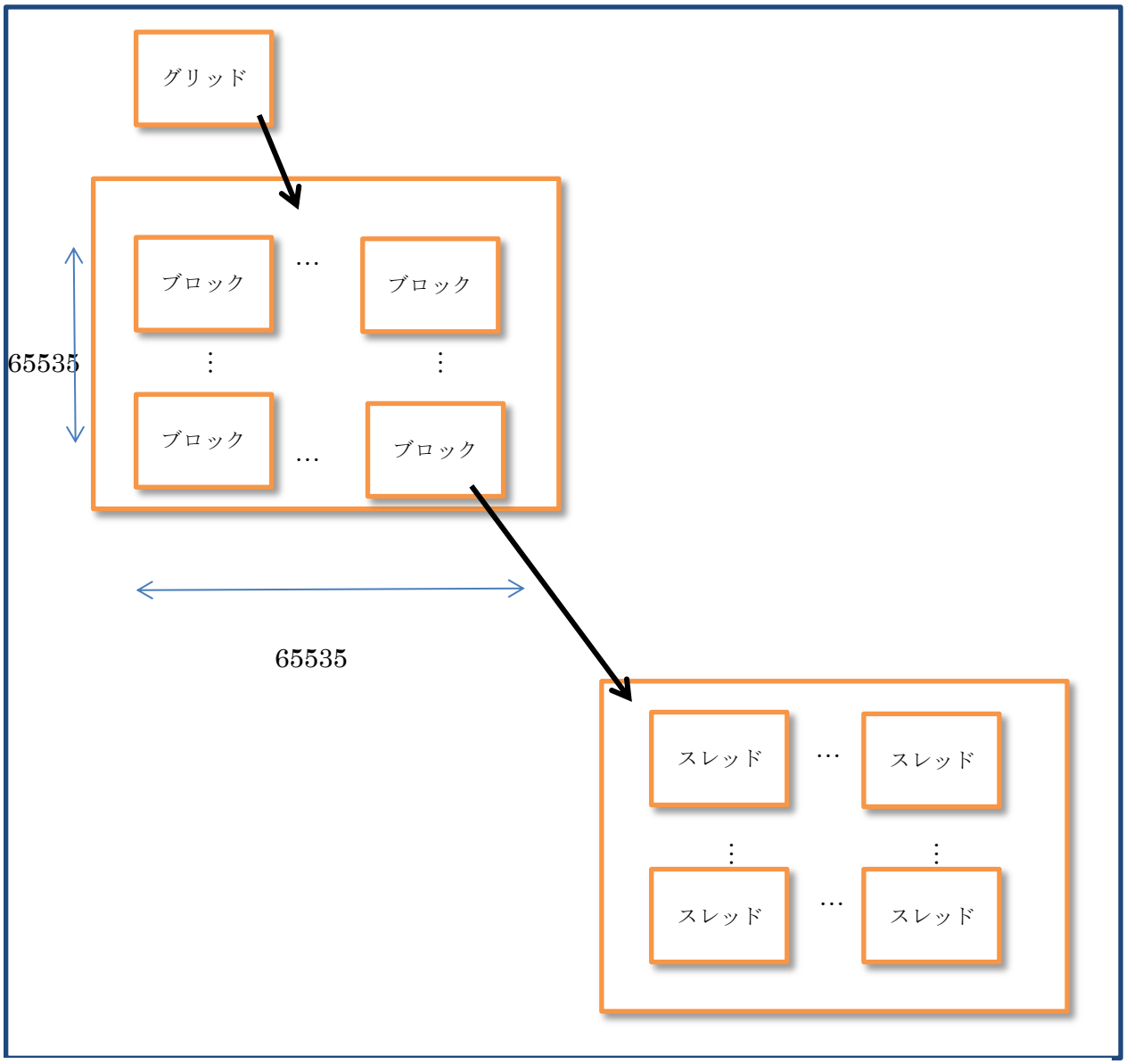

図 3-5　GPU 内部の処理系

また CPU のみを用いるプログラムでは，上記の並列計算を行わず，全てのデータの計算を終えるまでループするプログラムとなっており，CPU のみで行うメリットは GPU との通信を行う必要がなく，各々のコアの性能が GPU と比較して格段に優れているということである．そのため，扱うデータが少ない場合や，プログラムの内容が複雑な場合は CPU のみの方が計算を早く行えることもある．そこで，本研究では GIS の地形データから面積を計算する場合に CPU のみで行うべきなのか，GPU を用いるべきなのかを計算時間で判断する．

さらに，GIS の地形データの面積計算を行う際に，もう一つ重要なことはその精度であるが，実際の地形データからは正確な面積を知ることが出来ないため，今回円弧近似の線積分で求めた値が正しいかどうかを判断することが不可能である．そこで本研究では，今回のプログラムの結果が実

際の面積とどれほどの差が発生するのかを測定し，面積を計算する他の手法による計算時間と精度を比較するためにもう一つ実験を行う．

実験内容は，面積がわかっている楕円を用いて，提案法である円弧近似の線積分を用いた面積計算とモンテカルロ法を用いた面積計算を行い，計算時間とその精度を測り，その差を比較し考察するという内容である．計算に用いる楕円は長軸が 150，短軸が 100 の楕円であり，式は次のように表される．

$$\frac{x^2}{150^2}+\frac{y^2}{100^2}=1 \tag{10}$$

また，その正しい面積は円周率を 3. 141592 とすると 47123. 88 であり，実際に行った計算結果との差を求め，このプログラムの計算精度を測る．

3. 2. 2　モンテカルロ法での面積計算

次に，今回用いるモンテカルロ法についての解説を行う．モンテカルロ法とは乱数を用いてシミュレーションを繰り返し，近似的に解を求める手法の総称であり，ここでは求めたい解は面積に，用いる乱数は座標の値となる．その内容は x 座標, y 座標にあたる乱数をコンピュータ内で作成し，楕円の内側に存在するか外側に存在するかを判定して，カウントし，実行した回数のうちいくつ楕円の内側に存在したかという判定結果から面積を求めるというプログラムであり，乱数は 0 から 150 までの x, 0 から 100 までの y をメルセンヌツイスタという擬似乱数作成アルゴリズムを用いて作成している．

メルセンヌツイスタという擬似乱数作成アルゴリズムは，松本・西村[11]により作成されたアルゴリズムで，周期の長い乱数を作成することができる．また，数をカウントするだけでは，100×150 の平面空間のうちの楕円が占める割合を求めているだけであるため，この値を面積に変換する．カウントされた回数を n，計算回数を N とすると面積 S は以下の式となる．

$$S=4\times\frac{n}{N}\times 150\times 100 \tag{11}$$

最初に 4 を乗じている理由は原点を中心点とした楕円の第一象限でシミュレーションを行っているため，それに 4 を乗じ，求めたい楕円の面積を求めるためである．

理解しやすいように，以下の例を示す，例えば，(10. 98, 70. 17)という座標が乱数で作成されたとする，式(6)の楕円の式に作成された乱数(x, y)の数値を代入し，その左辺が 1 より小さければ楕円の範囲に入っているということになるため，例の乱数を入れると左辺の値が 0. 4924……となり，1 より小さいため楕円の中に入っていると判定され，プログラム内部でカウントされる，もう 1 つの例として(134, 98)という乱数が与えられ，式(11)に代入したとすると，式(11)の左辺は 1. 75844 となるため楕円の範囲外であると判定されカウントされない，以上の計算回数 2 回でプログラムを終了すると，カウント数は 1 で，計算回数は 2 であるため，式(7)に代入すると S は 30000 となり，実際の面積との誤差は 17123. 88 となる．2 回の場合ではこのような誤差になってしまうが，計算回数

を増やしていく内に時間はかかるが，実際の値に近づいていく，視覚的に表すと図 3-4 のようになる．

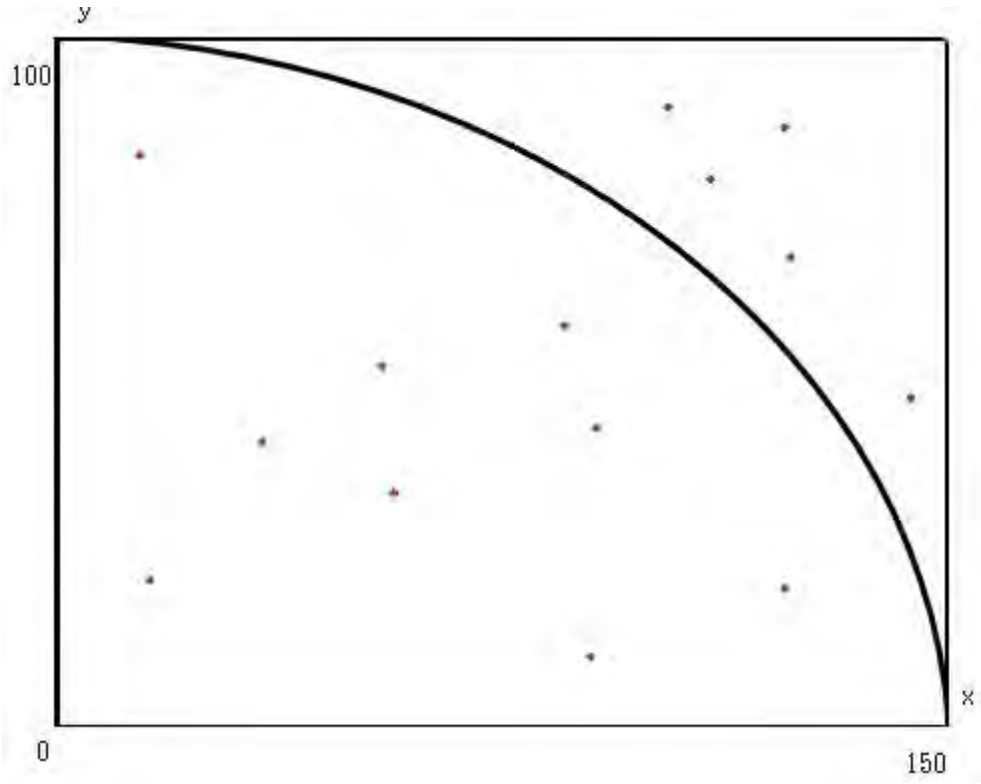

図 3-6　モンテカルロ法を視覚的に表現

4　実験

4.1　GPU の有用性の評価

4.1.1　実験環境

本研究では実験を行ったパソコンの環境が実験結果に直結するため，そのスペックの情報を表 4-1 に記載する．

表 4-1　実験を行った環境

OS	Windows 7 64bit
メモリ容量	4GB
CPU	Intel Corei7 860
コア数	4
スレッド数	8
動作周波数	2.8GHz
GPU	Geforce GTS250
搭載 GPU 数	1 基
CUDA コア数	128
プロセッサ周波数	1836Hz
単精度演算	最大 470GFlops
搭載メモリ	512MB
ホスト接続	PCI Express x16
開発環境	Visual C++ 2010 express

4.1.2　実験に使用したデータ

本研究では，GIS の座標データや楕円の座標データを用いる，また，使うデータによっても結果が異なるため，実験でどのようなデータを用いたのかを記載する．まず，使用した GIS の座標データを紹介する，今回用いた座標データは大島を 10m 標高で高さを区切った三次元座標データであり，図 4-1 に示すような形式になっており，データ形式は csv ファイルで，一行目にヘッダー行があり，プログラム中ではそのヘッダー行で標高 0 の座標数を読み込み，その数の行の面積を計算させ，次のヘッダー行で次の標高分の座標数を入力させる，という処理をファイルの終わりまで繰り返すようになっている．

	A	B	C	D	
1	5865	0	0	0	⇦ ヘッダー行
2	9444.417	6840.302	−0.20668	0.012742	⇦ x座標，y座標，座標間の角度，ダミーデータ
3	9443.449	6839.276	−0.15784	0.013054	
4	9442.179	6838.66	−0.08584	0.015651	
5	9440.808	6838.33	−0.04378	0.019212	
6	9439.408	6838.152	−0.02797	0.02316	
7	9437.498	6838.031	−0.00949	0.028878	
8	9435.585	6837.966	−4.90E−05	0.034932	
9	9432.962	6837.892	0.007036	0.043705	
10	9430.34	6837.792	0.013751	0.052909	
11	9426.825	6837.581	0.015983	0.065652	
12	9423.319	6837.261	0.023237	0.078423	
13	9418.657	6836.649	0.026064	0.094722	
14	9414.032	6835.801	0.040743	0.109229	
15	9407.885	6834.235	0.052145	0.124596	
16	9401.921	6832.076	0.094964	0.134182	
17	9393.911	6827.706	0.135629	0.139323	

図 4-1　大島の三次元データ

また，この座標データを，MATLAB を用いて座標をプロットすると図 4-2, 3 のようになる，両方の図における色の区分は，青，水，黄緑，黄，橙，赤色の順番に標高が 200m ずつ高くなっている．ただし，立体的に表した図 4-3 の図形では縦横の縮尺が実物とは違うため，実際の地形とは異なっている．そして，実際の図形と比較するために，図 4-4 に気象庁　東京航空地方気象台　大島空港分室のホームページにある大島の地形図を記載する．

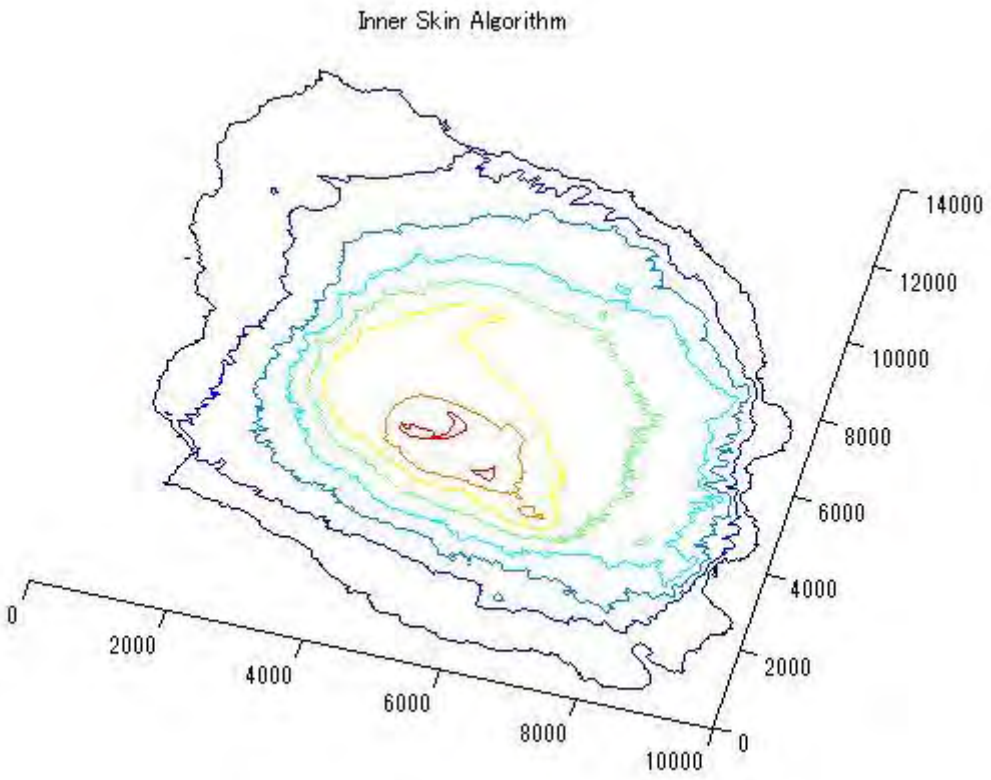

図　4-2　大島の座標データを平面的にプロット

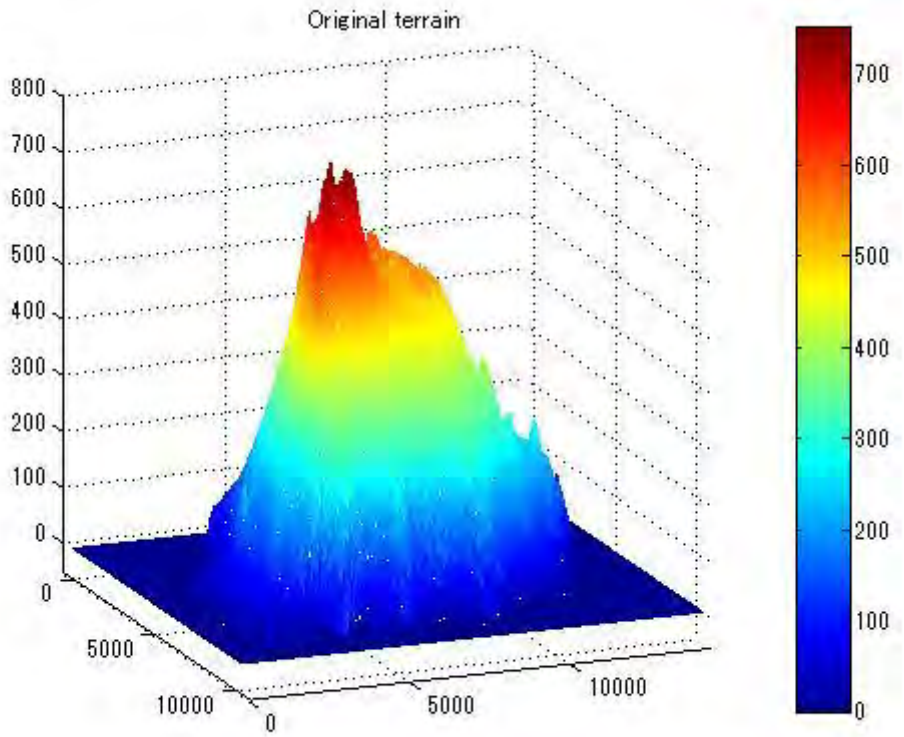

図　4-3　大島の座標データを立体的にプロット

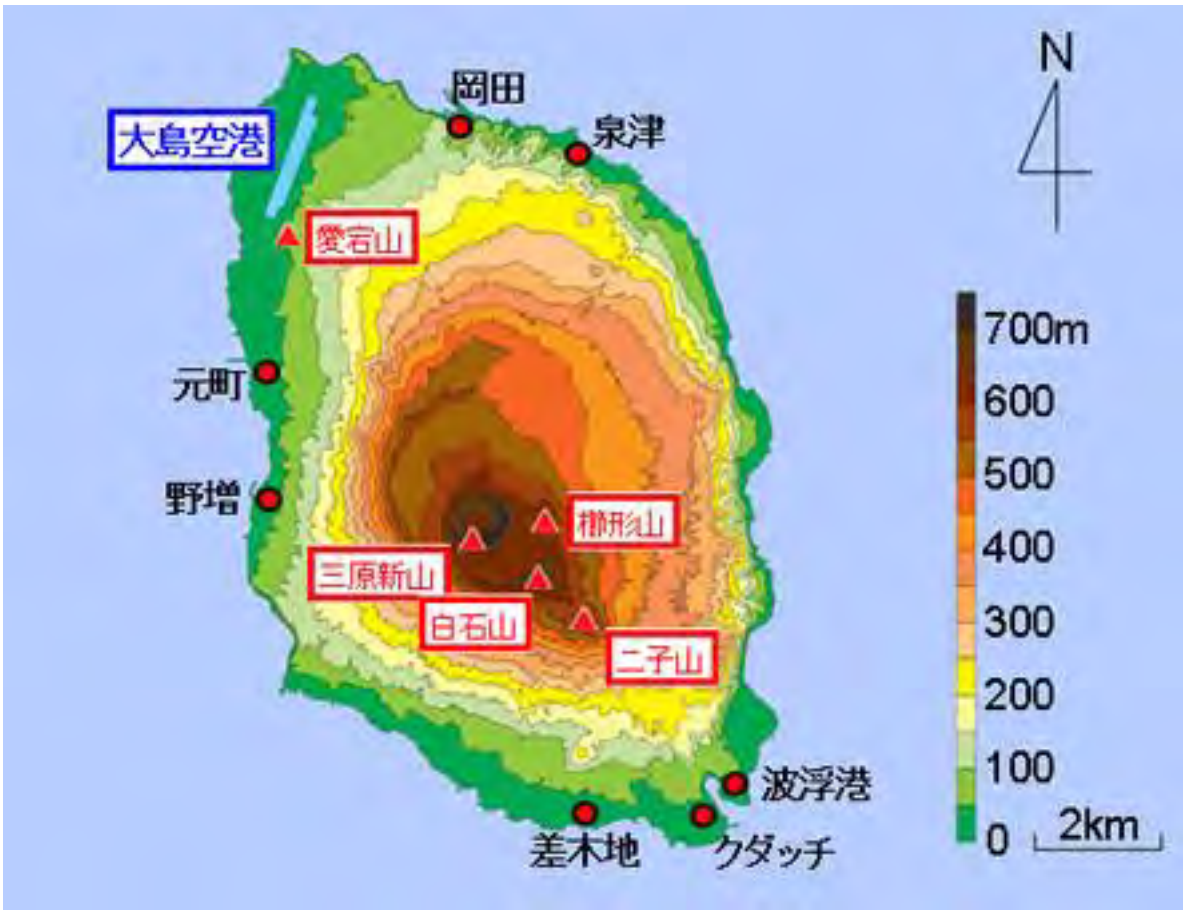

図　4-4　気象庁　東京航空地方気象台　大島空港分室 HP にある大島の地形図[12]

このように図 4-2 の GIS データを MATLAB でプロットした図形と図 4-4 の実際の地形の形や等高線が一致するため，使用した GIS データは概ね正しいと言える．

4. 1. 3　実験結果と考察

まず，CPU のみの環境と，CPU+GPU の環境で大島の GIS の三次元座標データの面積・周の長さを計算する実験を行った．本研究では，両方共の計算プログラムは同一であるため，計算結果自体は等しく，また，計算結果が実際の地形の結果に合っているかを判定することが不可能なため，面積と周の長さの計算結果は評価の対象外とする．代わりに，後述するアルゴリズムの有用性を比較する実験で実際の面積がわかっているものと計算結果の面積を比較する．

表 4-2　大島の面積計算にかかった時間の比較

	CPU	GPU	
1 回目	4567	4653	
2 回目	4552	4642	
3 回目	4502	4664	
4 回目	4556	4644	
5 回目	4568	4615	
6 回目	4503	4689	
7 回目	4488	4666	
8 回目	4550	4643	
9 回目	4490	4667	
10 回目	4475	4606	
			CPU の何倍の計算時間か
平均計算時間	4525.1	4648.9	1.027359
中央値	4526.5	4648.5	1.026952
最大値	4568	4689	1.026489
最小値	4475	4606	1.029274

表 4-2 は CPU と GPU で面積計算を行い，そのプログラムの最初と最後で時間を測り，どれだけかかったかを計測した結果である，結果を見ると，想定した通りの結果にはならず，CPU の方が平均約 124ms 程度プログラムの終了が早かったという結果になった，この結果になった原因として考えられるのは OpenCL でプログラムを記述すると，どうしても CPU 単体でプログラムを記述する場合より処理が増えてしまうことがあげられ，また，GPU と CPU の通信に時間がかかり，それが計算時間より長くかかってしまっているのではないか，という点も考えられる．

そこで，プログラムの始まりと終わりにタイムを測るのではなく，面積や周の長さを計算している時間のみを計測する実験を行い，その結果を表 4-3 に示す．

表 4-3　面積計算のみの時間の合計

	CPU	GPU	
1 回目	41	51	
2 回目	42	59	
3 回目	40	51	
4 回目	40	54	
5 回目	43	46	
6 回目	41	62	
7 回目	42	58	
8 回目	40	51	
9 回目	44	50	
10 回目	42	51	
			CPU の何倍の計算時間か
平均計算時間	41. 5	53. 3	1.284337349
中央値	41. 5	51	1.228915663
最大値	44	62	1.409090909
最小値	40	46	1.15

この実験の結果，大きな差はないが，CPU の方が少し計算を早く終えることがわかった，このプログラムでは，大島の座標データを 10m の等高線ごとに区切って面積を計算していて，その等高線ごとの計算時間を見ると CPU の方も GPU の方も全て 0 か 1 であり，殆ど計算時間に差が生じず，また，GPU では CPU との通信時間がどうしてもかかってしまうため，このような少しの計算時間の差が生まれてしまったと考えられる，これは各等高線間の座標数が少ないため，CPU と GPU での計算時間が殆ど変わらないせいで起きたと考えられる．

また，表 4-2 の計算時間となぜこれ程違うのかを調べるために，様々な場所で時間を計測した結果，面積計算時間以外では殆どファイルを読み込む箇所でかかっていた，上記の 2 つの理由からこの GIS の座標数では計算時間自体に差がでず，ファイルを読み込む箇所で時間をとられてしまうため，GPU を採用するメリットはあまりないという結果になった．次に，どのような場合に GPU を用いると良いのかを調べるために，次の実験で用いる楕円のデータを使い，計算時間の上昇率を調べ，どれだけ座標の数があれば CPU より早くなるのかを調べる．そのために，CPU と GPU で座標数が 201, 2001, 20001, 100001, 1000001 の楕円の面積を計算し，その計算時間を計測し，グラフを作成する．

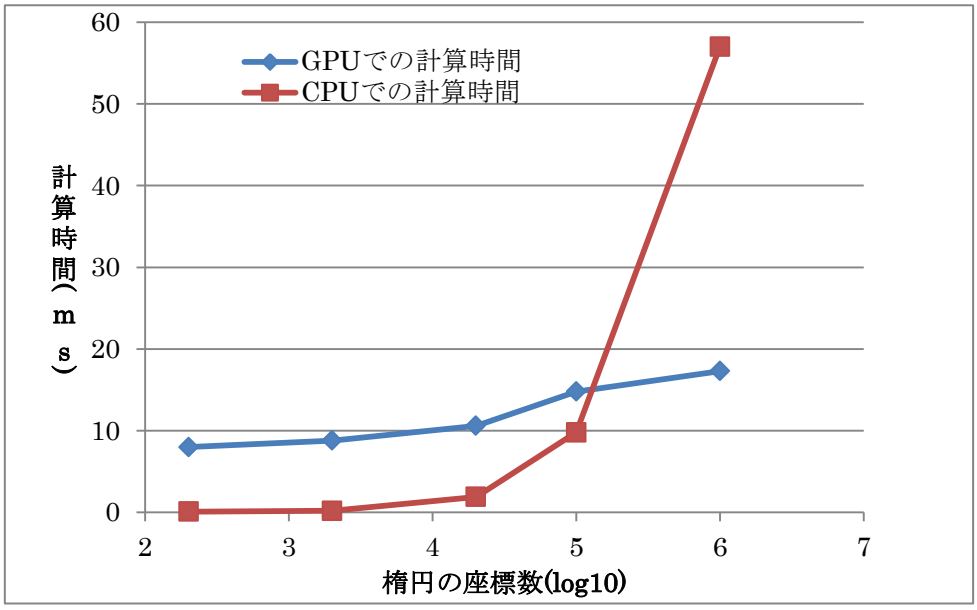

図　4-5　楕円の面積を CPU と GPU で計算した時のそれぞれの計算時間

結果は図 4-5 のようになり，一度に約 200000 個の座標を計算する場合に CPU と GPU が同等の計算時間で計算を行うことができ，それを超えると GPU の方が計算時間を短縮できることがわかった．つまり，この面積の計算アルゴリズムで用いる場合は同一の，等高線上の座標が 200000 個を超えると GPU を用いた方がいいという結果が得られた．

しかし，このプログラム単体では，面積を計算することは出来ても，その精度がどれほどの物であるかがわからないため，正解がわかっている図形でなくては精度がわからず，実際の大島の面積の等高線ごとの面積の正解はわかっていない．そのため，次の楕円を例にした実験によってこの線積分のプログラムの精度を確認する，尚，計算した特徴量のデータは付録に記載する．

4. 2　線積分のアルゴリズムの精度を評価

4. 2. 1　実験環境

4. 1. 1 の実験環境と同様の環境で行った．

4. 2. 2　実験に使用したデータ

本実験では，楕円の面積の計算を，それぞれ線積分で面積を計算するアルゴリズムと，モンテカルロ法を用いて面積を計算するアルゴリズムで行い，座標数を変えることによる計算時間とその精度の動きを比較する，そのために，今回は人為的に正しい面積のわかる楕円を作成し，その座標データを作成する．作成する楕円は長軸 a が 150，短軸 b が 100 の楕円で，その楕円から作成する座標データは中心から任意の角度 θ で区切り座標を設定する．例えば$\theta = \frac{\pi}{100}$と設定すると，始点が(150, 0)から始まり終点が(150, 0)の 201 個の座標の csv 形式の座標データが作成される．また，Csv 形式のデータには面積を計算するために必要な角度も記載され，る．

4. 2. 3　実験結果と考察

今回行う実験は線積分を用いたアルゴリズムで GPU を用いた場合で面積を計算した結果とモンテカルロ法で面積を計算した結果の精度と計算時間を比較し，線積分と GPU を用いたことに利点があったかどうかを考察する実験である．

まず，表 4-4 にそれぞれのアルゴリズムで面積を計算した部分のみの計算時間を比較した表，図 4-6 にグラフを載せる，ただし，図 4-9 の x 軸, y 軸はlog_{10}で対数を取ることとし, csv ファイルの要素数の都合で線積分の座標数は 1000001 までとする．

表 4-4　各手法における面積の計算時間の比較

モンテカルロ法	
打点数	計算時間の平均(ms)
201	0. 03960396
2001	0. 356435644
20001	3. 702970297
100001	18. 14851485
1000001	182. 5643564
10000001	1735. 455446
100000001	17651. 85149

線積分	
座標数	計算時間の平均(ms)
201	8
2001	8.8
20001	10.6
100001	14.8
1000001	17.3

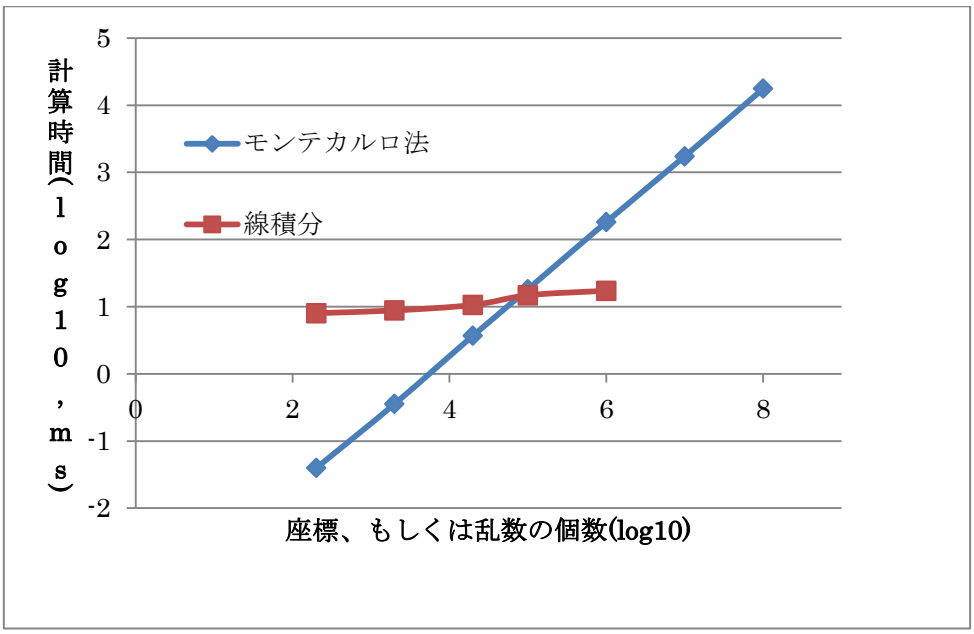

図 4-6　モンテカルロ法と線積分の計算時間の比較

図 4-6, 表 4-4 から分かる通り，計算時間はモンテカルロ法と線積分は共に線形的に計算時間が増えていて，途中まで線積分の方が計算時間の点で劣っている，しかし， モンテカルロ法ではプログラム中に乱数を発生させ，計算を行っているため，一定の個数を超えると線積分の方が計算時間を短く済ませることが出来たと考えられる．

次に，計算した面積の精度を測定した図表を，表 4-5，図 4-10 に記載する，図 4-6 と同様に x 軸,y 軸はlog_{10}で対数を取る．

表 4-5　各手法の面積計算精度の比較

モンテカルロ法		
打点数	正しい面積との差の平均	誤差(%)
201	1301. 38955	2. 761634971
2001	452. 7710527	0. 960810215
20001	127. 2044715	0. 269936328
100001	62. 80106969	0. 133268037
1000001	20. 54436628	0. 043596508
10000001	6. 067718356	0. 012876101
100000001	1. 895131931	0. 004021596

線積分		
座標数	正しい面積との差	
201	9. 054531526	0. 019214317
2001	0. 100263762	0. 000212766
20001	0. 010712207	2. 2732E-05
100001	0. 009841388	2. 08841E-05
1000001	0. 0098049	2. 08066E-05

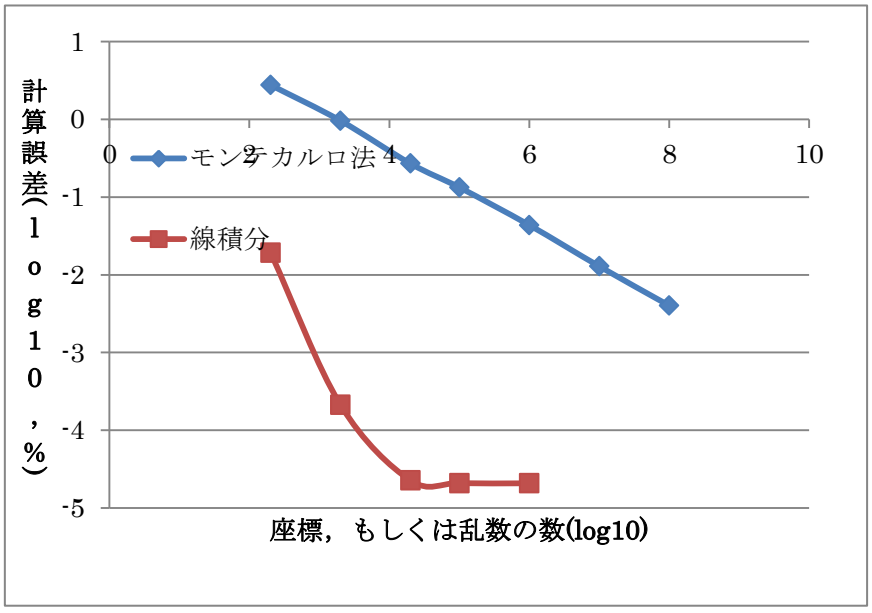

図 4-7　各手法の面積計算精度の比較

図 4-7, 表 4-4 から計算精度は線積分の方が格段に精度がいいことが判明した, その差は座標数 201 の場合で 143 倍も違い, 線積分の座標数 201 と同精度になるにはモンテカルロ法では

10000001 個点を打たなくてはならず，本研究の想定するハザードマップでは精度が何より重要視されるため，線積分を使うべきである．

また，同じ計算時間の場合では，精度がどちらの方がいいかという点も重要であると考えられるため，図 4-8 に縦軸を計算精度，横軸を計算時間としたグラフを記載する．

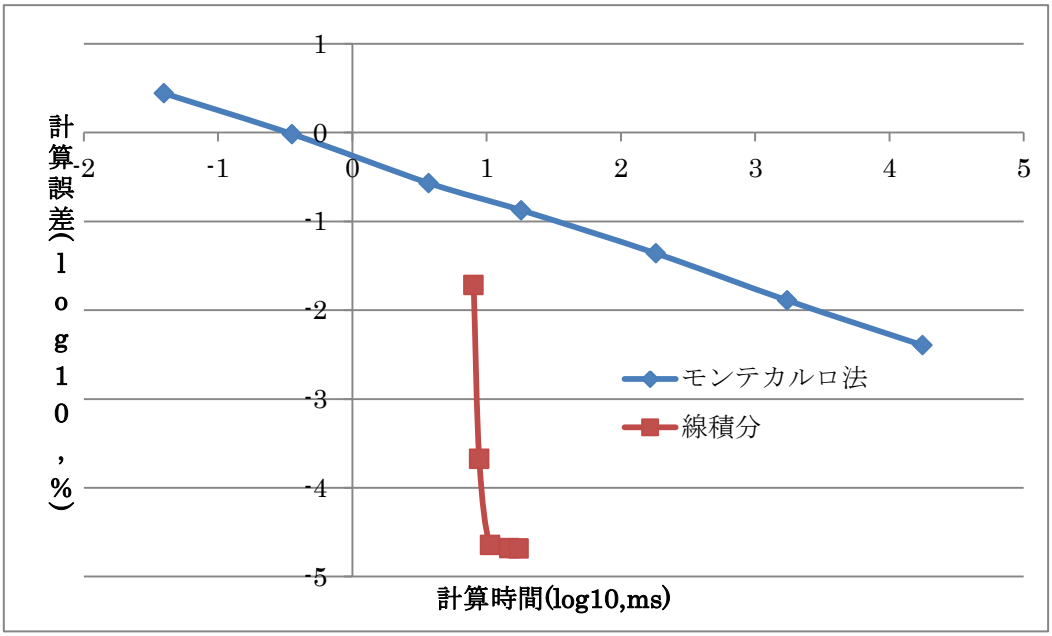

図 4-8　各手法の面積計算精度と計算時間の比較

図 4-8 から同じ計算時間では線積分の方が精度がいいことがわかった，この結果から，よほど精度を求めない計算でない限りは線積分を用いた方が効率が良いことがわかった．つまり，本研究で線積分を用いたことは正解だったと言える．

5　結論

本研究では将来的にハザードマップの作成で使う面積と周の長さを，従来の方法より高精度かつ高速に計算にするということを目標として研究を行った，そのために従来と違う手段で計算を行うことにした，その特徴として，モンテカルロ法よりも計算回数の少ない線積分を用い，その精度を上げるために座標間の補間を直線ではなく，円弧近似で行い，また，最新の技術である GPU を用いてプログラミングを行い，計算時間の短縮と同時に精度の向上をねらった．

そのための実験として 2 つの実験を行った, 1 つ目は大島の三次元座標データを CPU のみで計算を行った場合と，GPU を用いて計算を行った場合で計算時間の計測を行った．最初に，プログラムの最初と最後で時間を行った結果，大差はなかったが，GPU を用いた方が遅いという結果が出た，しかし，この計測時間の殆どはデータを読み込む時間で，実際の計算時間ではないと考えられたため，計算の直前と直後で再度時間を計測し直した，その結果計算時間は極めて短く，CPU との通信にかかる時間が少しあるため，わずかに GPU の方が計算時間がかかった．

そこで，次に楕円で座標数を変えながら時間を計測することにし，座標数と計算時間のグラフを作成し，どの程度の座標数から GPU を用いた方が有利になるのかを検証することにした．結果は，座標数が 200000 を超えたあたりから, GPU の方が計算結果が早いことがわかった．

2 つ目の実験として，円弧近似で座標間を補完した線積分の精度を，他の面積を求める手法であるモンテカルロ法と比べて評価する実験を行った．その結果，同じ計算回数ではモンテカルロ法の方が計算時間が短いことが分かった，しかし精度を比較すると，線積分のほうが圧倒的に優れていて，同じ計算時間のときの計算精度を比較しても線積分の方が 10 倍精度がいいことがわかった．

まとめると, 200000 を超える座標数の面積を計算する場合は GPU を用いたプログラムを使用した方が計算時間が短くなり，また，モンテカルロ法と比較すると線積分を用いた面積計算の方が計算精度を優先する場合では計算時間が優れているという結果になった．

つまり，本研究での目的であった，地形データの特徴量の計算においては，一度に 20 万以上の座標データを計算する場合には GPU を用いた方が高速化を行うことが可能で，それ以下の場合も大きな差はなかったため，全て GPU で計算を行っていればよく，また，ハザードマップ等の災害時の計算には精度が求められるため，モンテカルロ法よりも精度が良い線積分を使った方が良いという結論に至った．

6　今後の課題

今後の課題として，今回の研究では，あまり GPU を用いた利点を示すことが出来なかったため，今後の研究にはこれを解消したいと考える．

その案として，csv ファイルでは要素の数に限界があり，またプログラム上で一番時間が割かれている部分が計算時間ではなく，ファイルの読み込み部分であることが判明し，ファイルからデータを読み込むプログラムでは，どのようなプログラムにしても確実に時間がかかってしまうため，次の研究ではプログラム上でデータの作成，計算を行えるようなプログラムを作成することが可能になれば，excel 等の拡張子の限界のデータ数より多くのデータ数を扱うことが可能になり，また，それに伴い GPU の性能をより活かすことが可能になるはずであるため，その手法を実装したい．

また，本研究では，実際の大島のデータを扱ったが，大島の実際の標高ごとの面積はわからず，その精度が判明しないため，実際の面積との比較を行うことが出来なかったことが心残りである．そのため，今後はどうにか正しい面積を計測することが可能になり，実際の面積における精度を求めることが可能になることが課題になっている．

参考文献

[1] 総務省「平成 23 年通信利用動向調査」

[2] Web「Programming Guide :: CUDA Toolkit Documentation」<http://docs. nvidia. com/cuda/cuda-c-programming-guide/index. html>(2014/1/10 アクセス)

[3] 小山田 耕二, 岡田 賢治 :「CUDA 高速 GPU プログラミング入門」株式会社 秀和システム(2010)

[4] Web「Kepler テクノロジー搭載 GTX 780 Ti グラフィックスカード| GeForce | NVIDIA」<http://www. nvidia. co. jp/object/geforce-gtx-780-ti-jp. html>(2014/1/10 アクセス)

[5] Web「新世代ビデオカード 徹底攻略 | 新世代ビデオカード スーパーガイド | DOS/V POWER REPORT」<http://www. dosv. jp/other/1007/>(2014/1/10 アクセス)

[6] 株式会社フィックスターズ(土山 了士, 中村 孝史, 飯塚 拓郎, 浅原 明広, 三木 聡) : OpenCL 入門 マルチコア CPU・GPU のための並列プログラミング, 株式会社 インプレスジャパン

[7] Web「GIS とは・・・｜国土地理院」<http://www. gsi. go. jp/GIS/whatisgis. html>(2014/1/10 アクセス)

[8] Web「防災マップ : 小金井市公式 WEB へようこそ」<http://www. city. koganei. lg. jp/kakuka/soumubu/bousaikoutsuuka/info/bousaimap. html>(2014/1/10 アクセス)

[9] 桑山 裕樹(2010) :「GPU を用いた行列演算の高速化」 平成 21 年度 立命館大学 理工学部 電子情報デザイン学科卒業論文 <http://www. hpc. se. ritsumei. ac. jp/papers/b11/kuwayama. pdf>(2014/1/10 アクセス)

[10] 福本 昌人, 島崎 昌彦, 吉村 亜希子(2008) :「数値標高モデルを用いた傾斜地カンキツ園の斜面崩壊危険度の評価」 近畿中国四国農業研究センター研究報告 7 号, pp.119-129

[11] M. Matsumoto and T. Nishimura(1998), 「Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator」ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 (1998)

[12] Web「気象庁大島空港分室ホームページ(空港分室案内図)」<http://www. jma-net. go. jp/haneda-airport/oshimakuko/tizu/rjto-tizu. htm>(2014/1/10 アクセス)

付録

本研究では大島の面積を計算したが，結局，計算結果が正しいかが不明であったため，本文中には載せず，付録に記載することとする，

表　大島の 10m 標高ごとの面積と周長

標高	座標数	面積	周長
0	5865	−9. 1E+07	61166. 55
10	2526	−8. 9E+07	55930. 62
20	2767	−8. 7E+07	59161. 33
30	2701	−8. 3E+07	58539. 51
40	2716	−8E+07	58261. 4
50	2847	−7. 6E+07	59965. 36
60	2864	−7. 3E+07	59770. 71
70	2757	−7. 1E+07	57743. 15
80	2775	−6. 8E+07	58247. 45
90	2759	−6. 6E+07	56709. 18
100	2840	−6. 4E+07	56273. 18
110	2504	−6. 2E+07	53830
120	2501	−6. 1E+07	53254. 93
130	2468	−5. 9E+07	52744. 7
140	2658	−5. 7E+07	53473. 87
150	2586	−5. 6E+07	52567. 48
160	2297	−5. 4E+07	50369. 99
170	2486	−5. 3E+07	50808. 76
180	2480	−5. 1E+07	51193. 37
190	2469	−5E+07	50385. 79
200	2449	−4. 8E+07	49876. 87
210	2472	−4. 7E+07	49917. 52
220	2562	−4. 5E+07	49965. 56
230	2348	−4. 4E+07	48127
240	2238	−4. 2E+07	46911. 01
250	2140	−4. 1E+07	45513. 62
260	2215	−3. 9E+07	45044. 32

270	2128	-3.8E+07	44423.3
280	2249	-3.7E+07	45477.86
290	2169	-3.6E+07	44233.12
300	2160	-3.5E+07	43383.55
310	2174	-3.4E+07	42496.36
320	2055	-3.2E+07	41401.61
330	2057	-3.1E+07	41746.12
340	2098	-3E+07	41357.09
350	2309	-2.9E+07	42292.3
360	2298	-2.8E+07	42247.07
370	2224	-2.6E+07	41690.56
380	2368	-2.5E+07	41891.62
390	2032	-2.3E+07	37131.85
400	1931	-2.2E+07	35599.98
410	1576	-2.1E+07	33141.37
420	1560	-1.9E+07	33031.15
430	1541	-1.8E+07	32294.87
440	1256	-1.7E+07	30053.33
450	1143	-1.6E+07	28347.25
460	1125	-1.5E+07	27519.28
470	1157	-1.4E+07	27764.23
480	1142	-1.3E+07	28035.8
490	1071	-1.2E+07	26946.35
500	957	-1.1E+07	25569.85
510	950	-1E+07	25388.79
520	1058	-9262535	26183.51
530	1083	-8188378	26753.33
540	765	-6425050	21856.08
550	662	-5667547	20865.73
560	518	-4952195	19125.18
570	500	-4467686	18689.55
580	464	-4005663	18127.43
590	484	-3603459	17720.13

600	410	-3161748	16038.35
610	331	-2875900	15315.84
620	391	-2550909	16130.38
630	258	-569322	13482.25
640	238	-371454	12358.67
660	135	-841353	11464.62
670	141	-758549	11272.71
680	172	-640537	11172.48
690	184	-287340	11539.22
700	169	-205239	11442.24
710	105	-62307.7	9453.024
720	64	-35966	8987.996
730	47	-17387.7	8588.107
740	44	-3695.14	8206.358
750	5	-12636.5	8301.058