動的に描画点数を考慮した
非同期数値地図情報表示システムの設計と実装

島根大学　総合理工学部

数理・情報システム学科　計算機科学講座　卒業論文

田中研究室

S023006　今井拓也

2007 年 2 月 6 日

目次

第1章　序論

近年はコンピュータとネットワークの性能が数年前に比べて格段と高くなったが、WWW での 3 次元コンテンツは普及しているとは言えない。

そこで、本研究では、先行研究[1]での河川流域境界データを使用し、実用的なサーバ型数値地図情報表示システムの実装を目的とした。

先行研究において、既に島村和義[1]は流域境界データと海岸線データを加工し、河川流域境界データとして X3D（eXtensible 3D）によってブラウザに表示することを試みて成功している。しかし、「河川流域境界データの法線が逆向きになる部分がある」、「出力されるファイルサイズが大きい」といった問題点が残されている。今回は、この河川流域境界データに対して法線の向きの修正・統一及び、描画点数の間引きを行った。更に、非同期なファイルアクセスをシステムに組み込んだ。

河川流域境界データの描画点数の間引きを行うにあたり、データ帯域を測定し、その測定結果に依存したファイルサイズを出力するようにした。ここで、データ帯域とはクライアントのマシンスペックとネットワーク性能の両方を考慮した実データ転送速度を指し、以後、本研究において「データ帯域」はこの実データ転送速度のことを意味することとする。

データ帯域に依存したファイル出力を行い、描画点数を落とすことで、マシンスペックやネットワーク性能が比較的低い環境の利用者でも本システムを利用することができるようになる。

また、非同期通信は Ajax（Asynchronous JavaScript + XML）の XMLHttpRequest を利用し、異なる河川流域境界データへの切り替えをページ遷移なしに行えるようにすることで実現した。

第2章　法線方向の統一

2.1　概要

島村[1]が作成した 3 次元流域境界データをクライアントのブラウザに表示させると、法線が逆向きになって表示される箇所が多く存在する（下図)。そこで、法線が下向きに表示されている部分を修正し、全ての法線が上を向くように統一した。

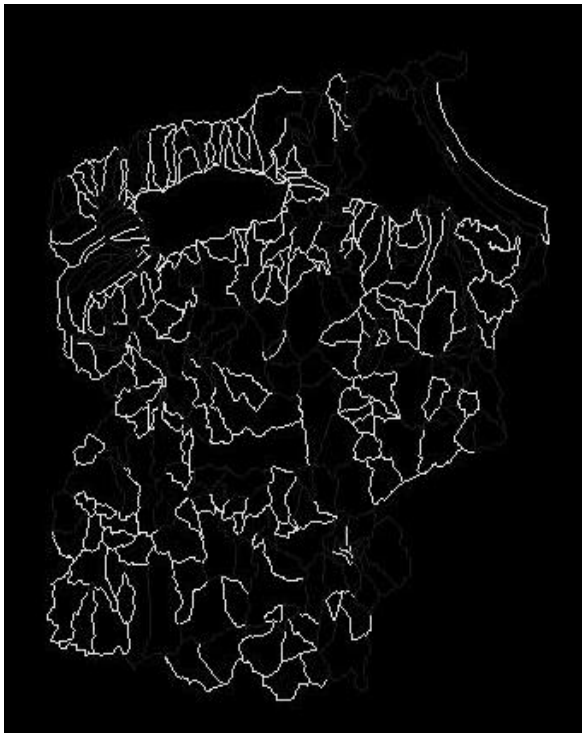


図 2-1-1：法線上向きの部分

図 2-1-2：法線下向きの部分

2.2　法線不統一の原因

法線の向きはデータの並びの順番に依存する。下図より、ちょうど右ねじの法則のように、ある点から次の点へ線を引く回転方向が右回りであれば法線は上向きになる。逆に、

左回りであれば法線は下を向く。

なお、*x,z* 軸が地表を表し、*y* 軸が高さとなっているのは X3D の仕様による。

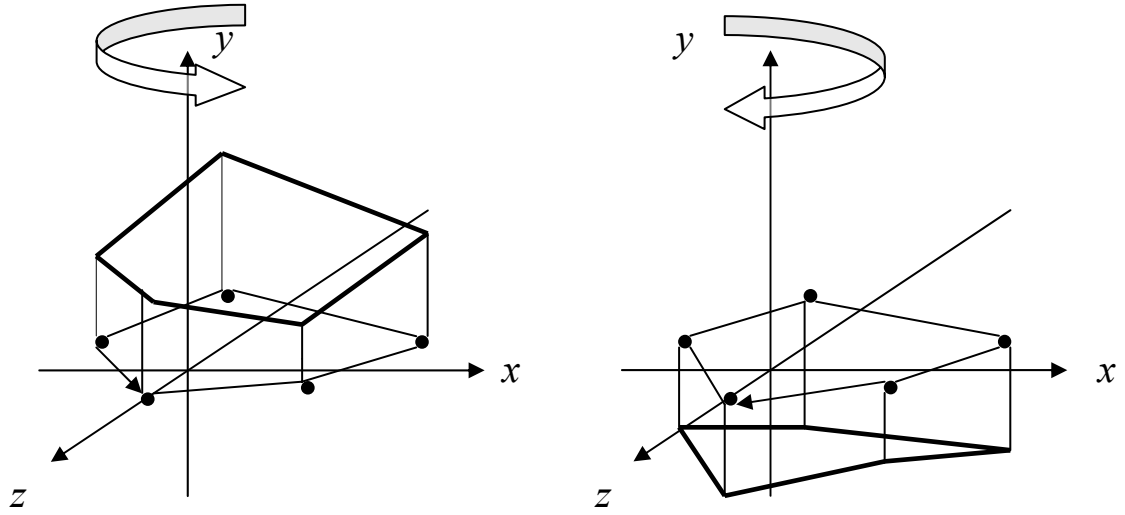


図 2-2-1：データの並び方と法線の向き

2.3　修正方法・修正結果

図 2-2-1 より、法線の向きは、ポリゴンデータの流れが左右のどちらであるかを判定することで知ることができる。その判定方法には外積ベクトルの公式を用いる。

外積の公式では、ベクトル *a,b* を辺とする平行四辺形の面積と、面積を求める際のベクトルの回転方向を知ることができる。

$$a \times b = |a||b|\sin\theta$$

公式より、外積 $a \times b$ の符号が分かれば、$\sin\theta$ の符号が分かる。つまり、$a \times b$ と θ の符号は等しいため、外積を計算した結果が正であるか負であるかで θ の回転方向が判定できる。以上より、

$a \times b > 0$ であれば、$\theta > 0$ で、左回り

$a \times b < 0$ であれば、$\theta < 0$ で、左回り

となる。これを図で示すと図 2.3.1 のようになる。

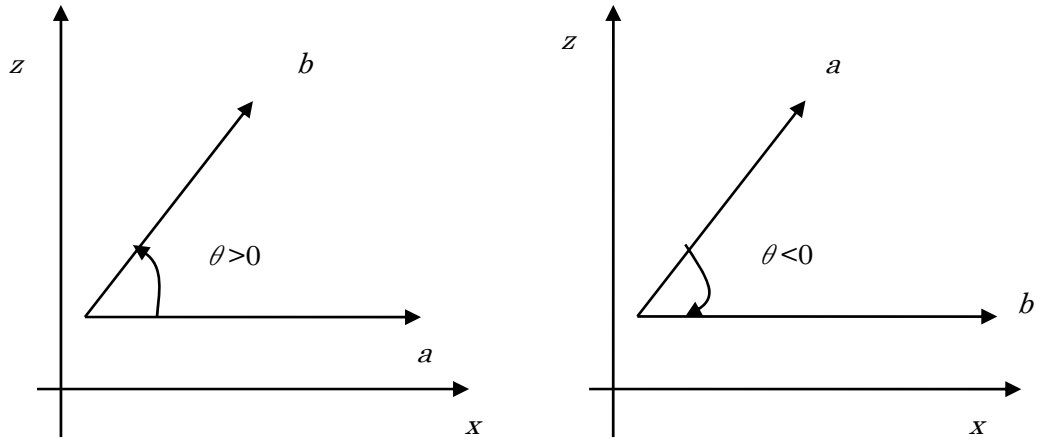
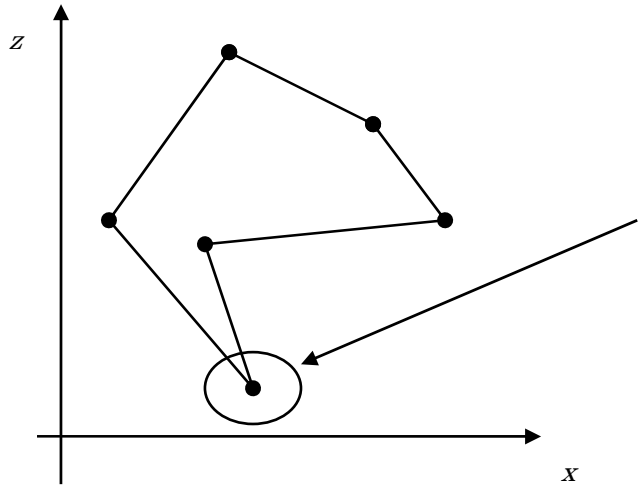


図 2-3-1：ベクトルの回転方向と θ の符号の関係

次に、この外積の考え方を実際にポリゴンデータに用いる方法について説明していく。

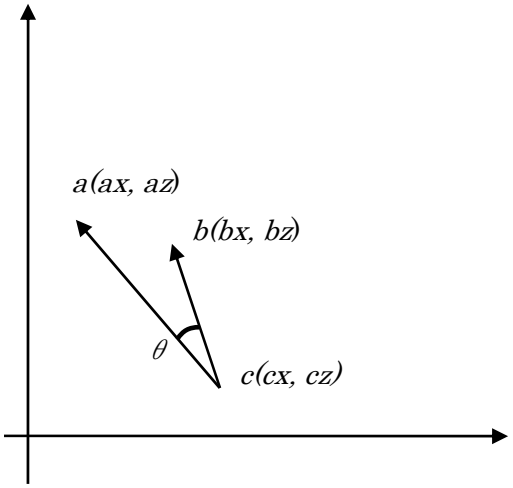
1. ポリゴンデータの中で z 軸の座標値が最小の点を見つける



x	z
906681	1006526
905231	1004865
906437	**1003329**
906100	1004500
906700	1006300
906681	1006526

図 2-3-2：ポリゴンデータの z 軸の座標値が最小の点

2.　*z*座標値最小の点と、その前後 2 点の 3 点で外積の計算をする



x	*z*	
906681	1006526	
905231	1004865	*a*
906437	**1003329**	***c***
906100	1004500	*b*
906700	1006300	
906681	1006526	

図 2-3-3：*z*軸の座標値が最小の点と前後 2 点

ここで、*z* 座標値最小の点が始点であれば、始点より前の点は存在しないため、始点、始点+1、終点-1 の 3 点での外積を考える。*z*座標値最小の点が終点であった場合も同様に、終点、終点-1、始点+1 の 3 点で外積を考える。

3 点 *a,b,c* の外積の計算式は

$$a \times b = (ax - cx) * (by - cy) - (ay - cy) * (bx - cx)$$

となる。計算結果が正であれば θ も正であり、データの流れが右回りであるので、法線下向きと判断し、データの流れを修正する。上図の例で外積を計算すると

$a \times b$ = (905231 - 906437) ＊ (1004500 - 1003329)
　　－ (1004865 - 1003329) ＊ (906100 - 906437)
　= －499967 < 0-

となり、負であるので、データには修正を加えない。この計算を全てのポリゴンデータに行い、法線が逆向きになっているポリゴンには修正を加えていく。

注意点として、上図の場合だと x 軸は右方向、z 軸は画面手前から奥向きの方向が正方向であるが、X3D の標準では、z 軸は、逆に奥から手前へ向けての方向が正方向となっている。実際に外積を計算する際には、どちらの回転方向が θ にとって正であるのかということを各軸の向きから判断する必要がある。

修正結果を図に示す。

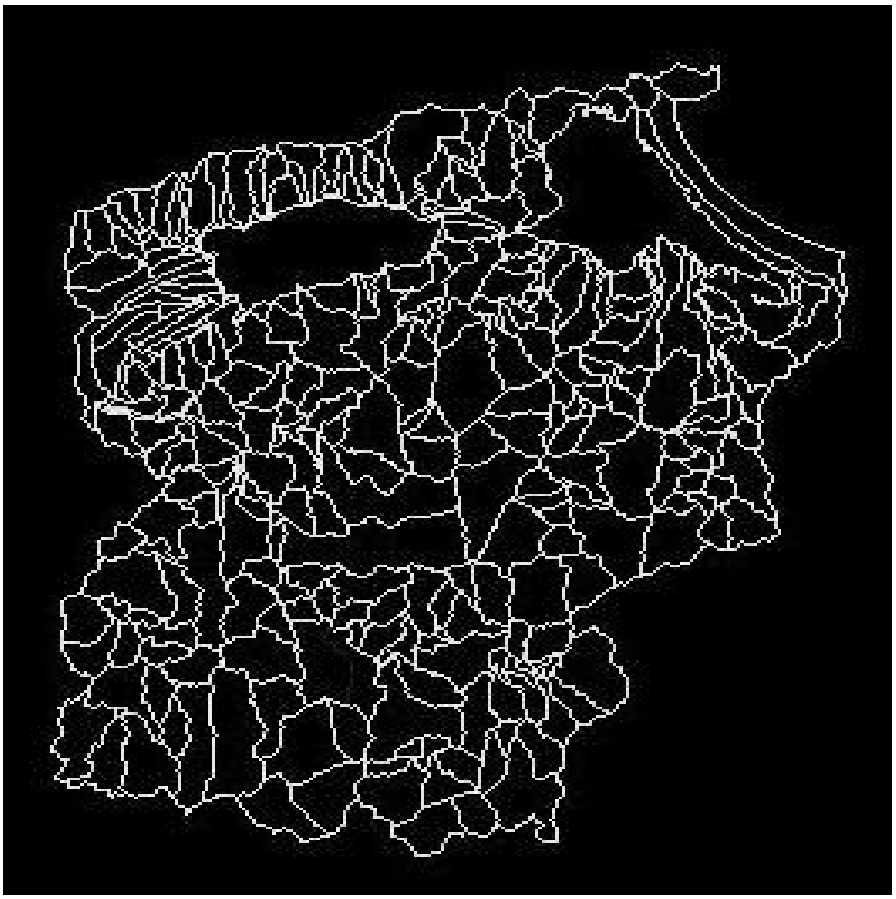


図 2-3-4：法線を修正した河川流域境界データ

第3章　描画点数の間引き

本研究の目的は、データ帯域を考慮した描画点数の間引きである。本章はその目的のために、グラフィックデータ量の変更方法を論じる。

3.1　データ構造の概要

島村[1]が作成した河川流域境界データは、ポリゴンデータ（ベクター型）と標高値データ（ラスター型）の２つの型のデータにより構成されている。よって、それぞれのデータ形式において、描画点数の間引き方法を考えなければならない。ベクター型データとラスター型データの概要を説明する。

ベクター型データ

ベクター型のデータでは、画像は、点、直線、円、などの幾何的な記述によって表現される。そのため、拡大や縮小を行っても情報が失われることはないが、写真のようなデータを表現するのには向いていない。

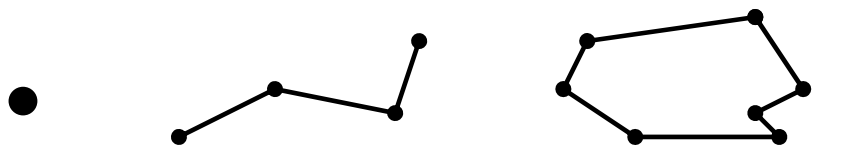
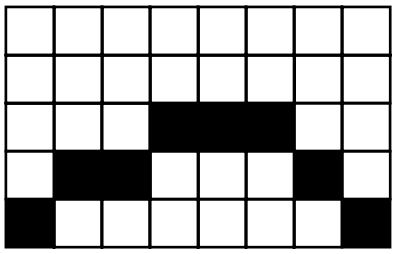


図 3-1：ベクター型のデータ構造

ラスター型データ

ラスター型のデータでは、画像を規則的な 2 次元の格子に分割し、各格子（メッシュ）の属性値をそれぞれのデータとして保存する。写真のようなデータの表現に向いているが、一般にファイルサイズは大きくなる。



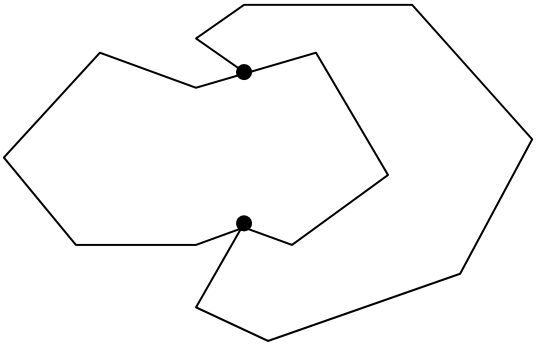
0　0　0　0　0　0　0　0
0　0　0　0　0　0　0　0
0　0　0　1　1　1　0　0
0　1　1　0　0　0　1　0
1　0　0　0　0　0　0　1

図 3-2：ラスター型のデータ構造

3.2　ポリゴンデータ（ベクター型）の間引き

3.2.1　ノード判定

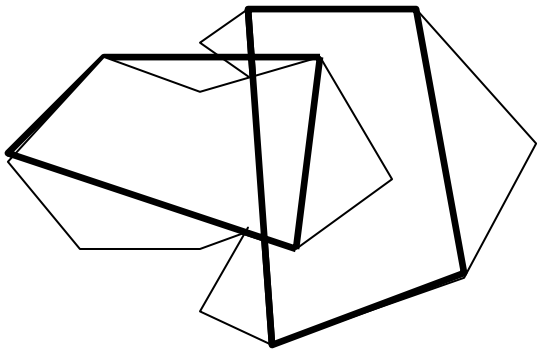
図 7-1-1 のように、ポリゴン同士の曲線が交わる、あるいは分岐する点を特に「ノード」と呼ぶ。そして、それ以外の点は「中間点」と呼ぶことにする。



黒丸：ノード
その他の点：中間点

図 3-2-1 ノードと中間点

描画点数を間引く際にノードを全く無視すると、間引き後のポリゴンデータ同士で曲線が交差してしまう（下図 7-1-2)。そこで、各点でノードであるかどうかの判定を行い、座標値データの前にノードタグを付け加えた。ノードであれば、その点は間引きの対象外とする。



ノードを無視して間引くと
曲線同士が交差してしまう。

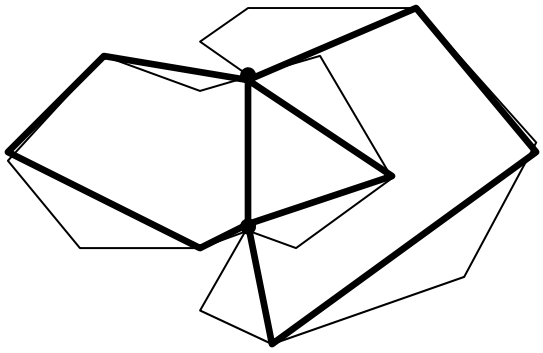
図 3-2-2 ノードを無視して間引いた例

ノードタグ	*x*	*y*	*z*
1	767151	7730	500000
2	768263	8510	498568
0	769743	9220	498167
0	771948	9940	496758
0	774153	11140	495576
2	776736	12500	493641
0	780064	10680	494467
1	767151	7730	500000

図 3-2-3：ノードタグを付け加えたポリゴンデータ

図のようにポリゴンデータにノードタグを付け加えていく。ノードタグはそれぞれ「０：中間点」、「１：始終点」、「２：ノード」とし、ノードタグが０、即ち中間点であれば、その点は間引きの対象となる。

しかし、中間点を適当に間引くと、間引いた中間点がポリゴンデータの形を特徴付ける特徴点であった場合、間引き後のポリゴンデータは元のデータと大きく異なるものになってしまう。



ノードを残しても間引き後のポリゴンの形が元の形と大きく異なる

図 3-2-4 ノードを残して中間点を適当に間引く例

そこで、中間点には Douglas-Peucker のアルゴリズム[5]を用いることで、ポリゴンデータの特徴点を残しつつ間引きを行った。

尚、今回は、点がノードであるかどうかの判定を一部手作業で行った。河川流域境界データは 19,589 行のデータからなるが、手作業で修正を加えたのはそのうち 600 行程度である。次より、手作業で行った理由を述べる。まず、ノードである点は以下の２つのどちらかの条件を満たしている。

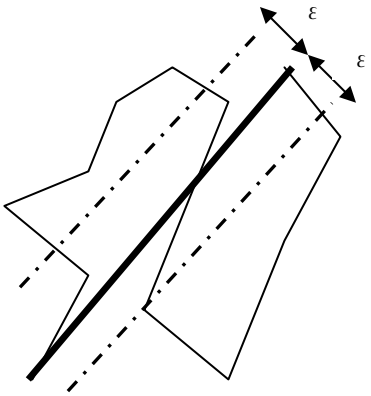
1. ポリゴン同士で共通する曲線の両端の点である
2. x,y,z 座標値が全く同じ点が他に２つ以上ある（ノード自身を含めると３つ以上）

条件２に関してはプログラムでノードタグを付け加えていったが、条件１の点をプログラムで判定するのは困難であった。

その理由は、ポリゴンデータは座標値ごとに順番に並んでいるわけではないため、対象としているポリゴンの左右に位置するポリゴンが分からないというものである。

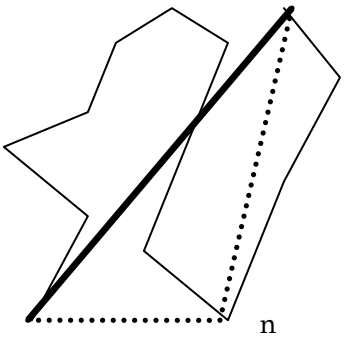
3.2.2 Douglas-Peucker のアルゴリズム[5]

Douglas-Peucker のアルゴリズムとは、複数の点からなるデータの間引きを考える際に、特徴となる点を残しつつ近似曲線を引くためのアルゴリズムであり、地理情報システム（Geographic Information System）で標準的に用いられている。その手順を以下に説明する。

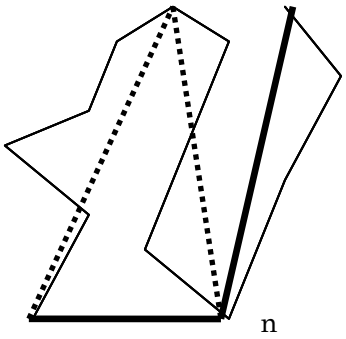


１． 始点と終点による近似曲線を引き、各頂点と近似曲線との距離（垂線の距離）を計算する。
ポリゴンデータの場合は、始点と終点の一つ前の点で近似曲線を引く。

全ての点において近似曲線からの距離が ε よりも小さければアルゴリズム終了。

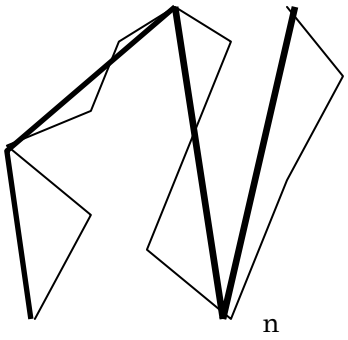


２． 精度 ε よりも大きく、近似曲線から最も距離がある点（n 番目とする）に次の近似曲線を引く。



３．　始点から n 番目の点
　　n 番目の点から終点

の２つで近似曲線を引き、それぞれにおいて手順 1.2 同様に近似曲線から各点への距離を計算し、ε より大きい点があれば次の近似曲線を引く。

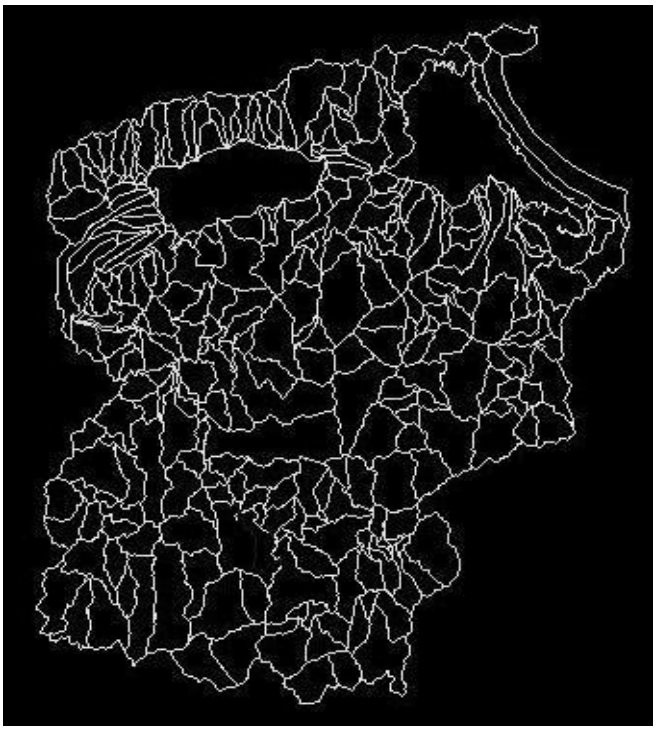


４．　全ての点がεに収まるまで再帰的に繰り返す。
　　全ての点がεに収まれば、アルゴリズム終了。

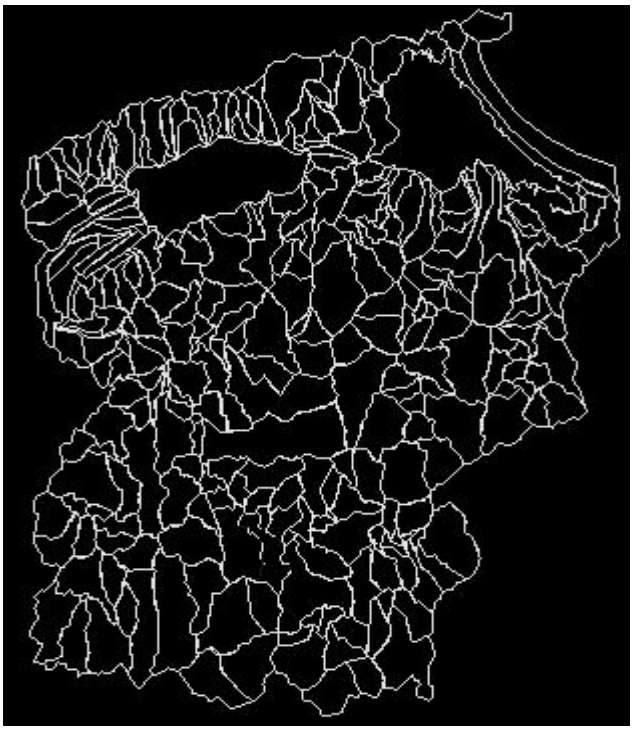
Douglas-Peucker のアルゴリズムで精度ε内に収まる点であっても、ノードタグより、ノードであると判定された点は間引きの対象外とする。

精度εを変更しながら間引いた幾つかのデータを次ページの図に示す。εの値を大きくすればするほど、ファイルサイズは小さくなり、データの精度が落ちていくのが分かる。

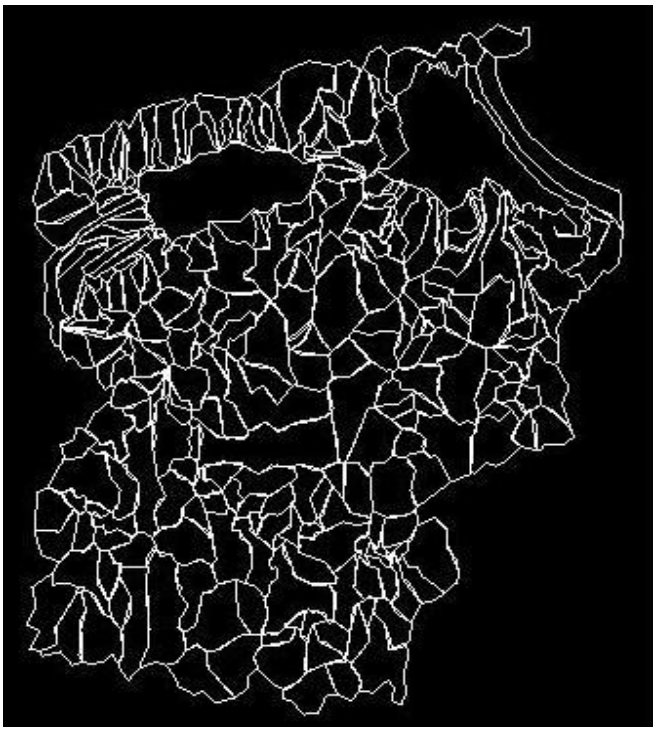
なお、ポリゴンデータの座標値は 0.1m 単位で表現されている（ε=1000 であれば、精度 100.0m である）。



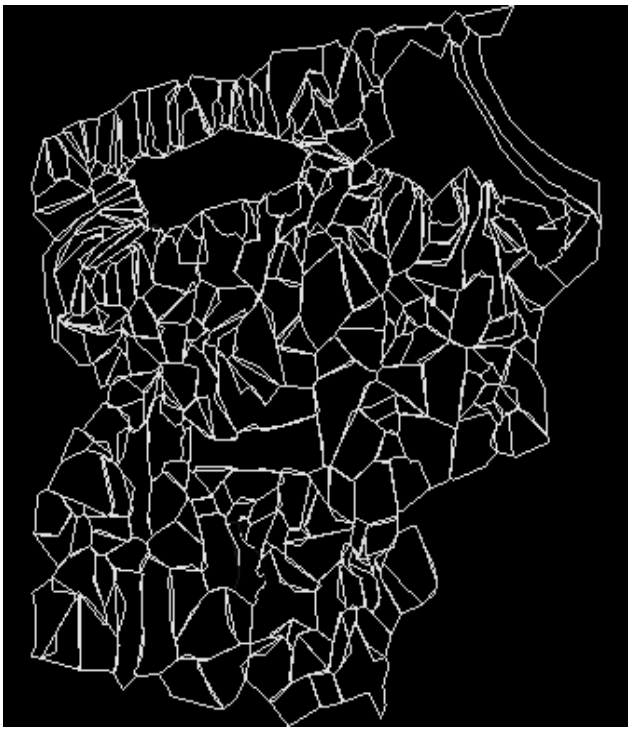
ε =500　365KB



ε =1500　245KB



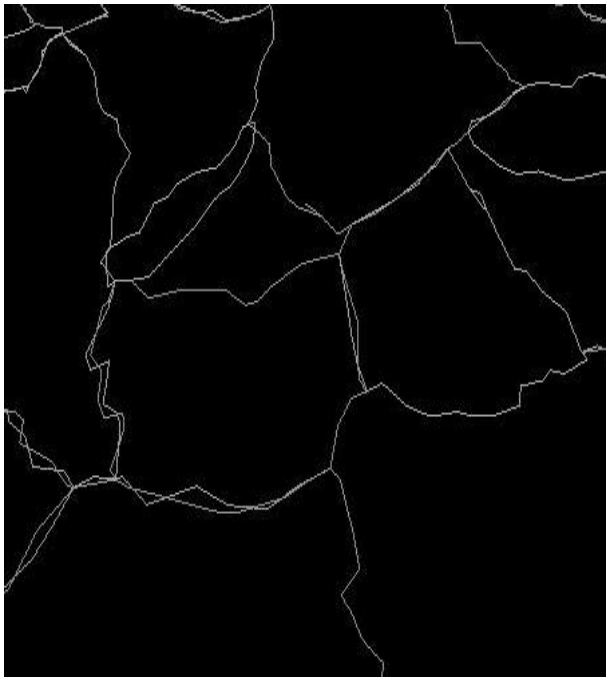
ε =3000　212KB



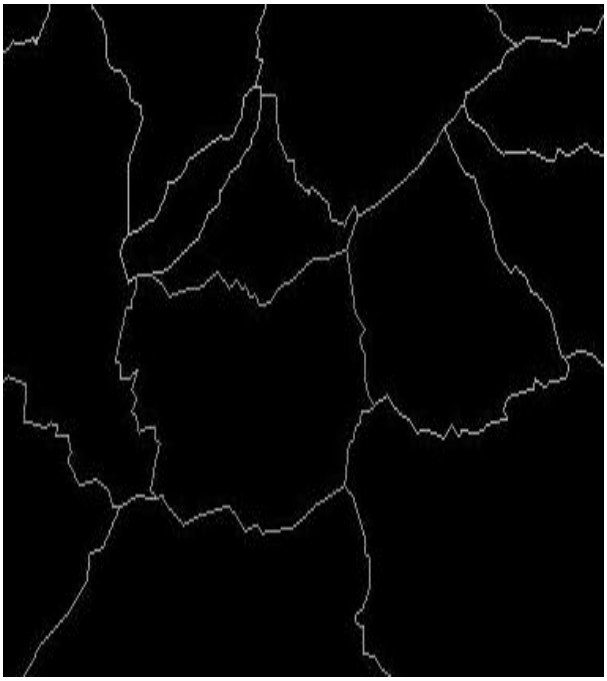
ε =10000　192KB

図 3-2-5：Douglas-Peucker を用いて間引いた河川流域境界データ

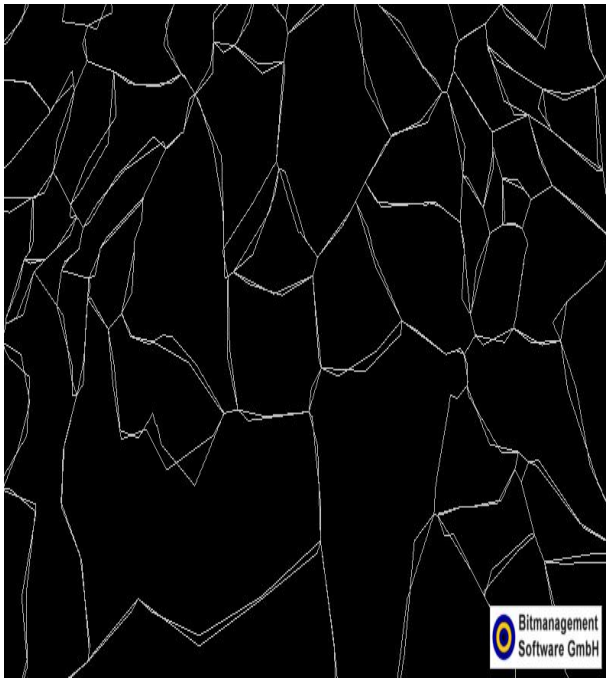
比較のため、Douglas-Peucker のアルゴリズムを用いて間引いたデータと、そのデータとほとんど同じファイルサイズになるよう適当に中間点を間引いたデータを下図に示す。



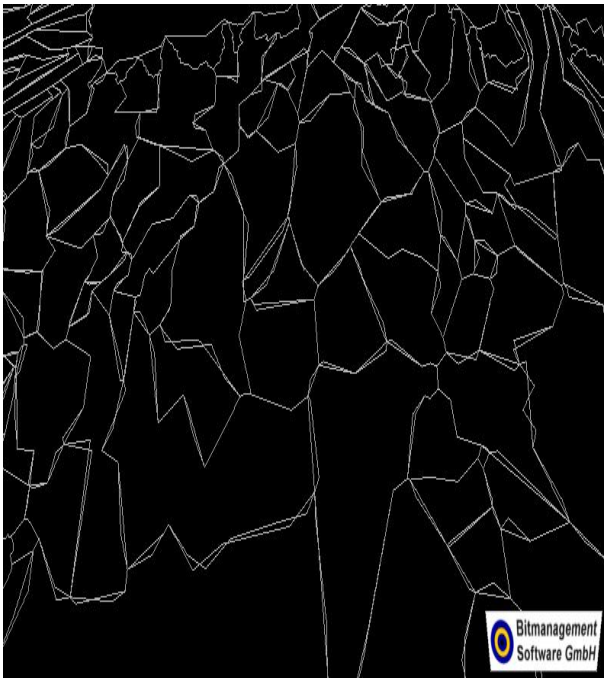
Douglas-Peucekr なし　　351KB



Douglas-Peucker　ε=500　　365KB



Douglas-Peucker なし　　207KB



Douglas-Peucker　ε=3000　　212KB

図 3-2-6：中間点を適当に間引いたデータとの比較

図 3.2.6 より、ファイルサイズが 351KB（Douglas なし）と 365KB（Douglas あり）では、Douglas-Peucker のアルゴリズムを用いない場合だとポリゴン同士の曲線が多く交わるのに対し、用いた場合は曲線の交わりはほとんどなかった。

しかし、ファイルサイズを 220KB 程度まで落とすと、Douglas-Peucker のアルゴリズムの適用あるなしに関わらず、ポリゴン同士での曲線の交差が多く見られた。

3.3　標高値データ（ラスター型）の間引き

メッシュデータを間引く際には、元の地形データの形を保持させるために、等間隔でデータを間引いていく必要がある。下図 7-2-1 のように、求めるファイルサイズに合わせてデータを間引く間隔を変化させる。

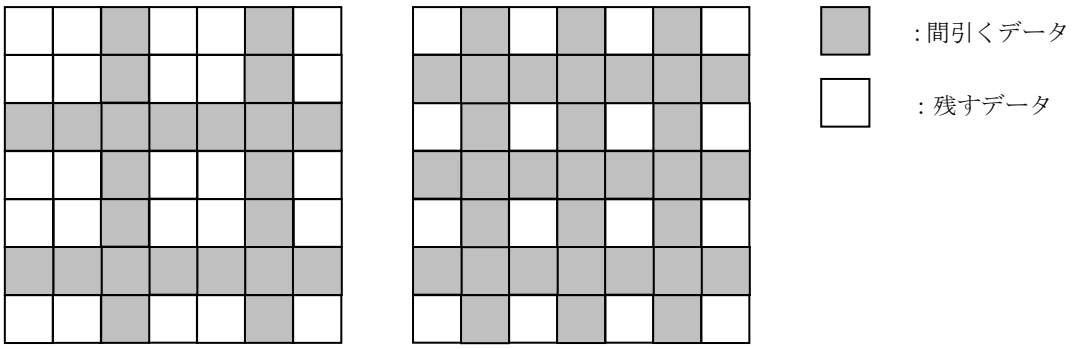


図 3-3-1：ラスター型データ（メッシュデータ）の間引き例

3.3.1　座標値の調整

間引きを行った標高値データは、当然、元の標高値データよりも少ないメッシュ数で地形を表現することになる。そのため、間引いた標高値データにポリゴンデータをそのまま重ね合わせると、お互いのデータの表示位置がずれてしまう。そこで、標高値データの間引きの間隔に合わせてポリゴンデータの座標値を修正する必要がある。

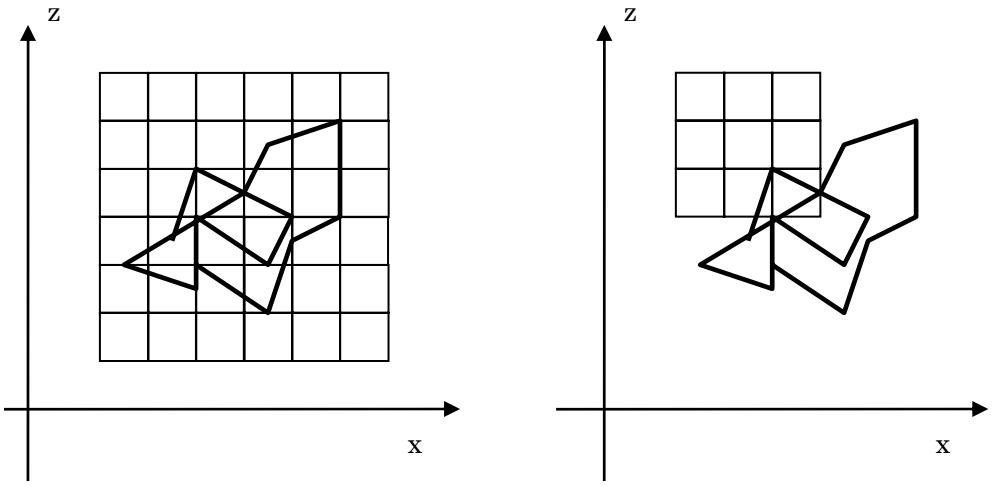


図 3-3-2：メッシュデータを間引いて座標値変更なしの例

ここで、X3D の ElevationGrid 機能を用いて、各メッシュデータの間隔を調整することで、ポリゴンデータの座標値を修正することなく 2 つのデータを正しく重ね合わせることにした。

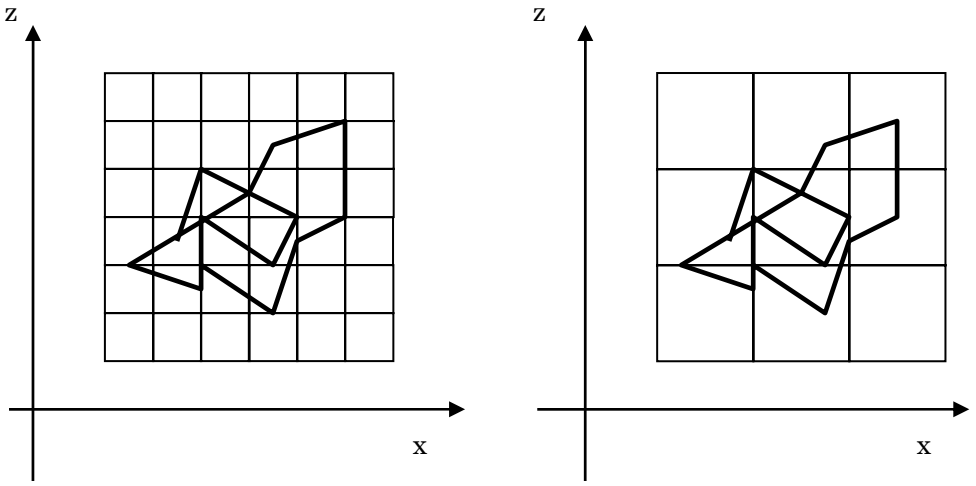
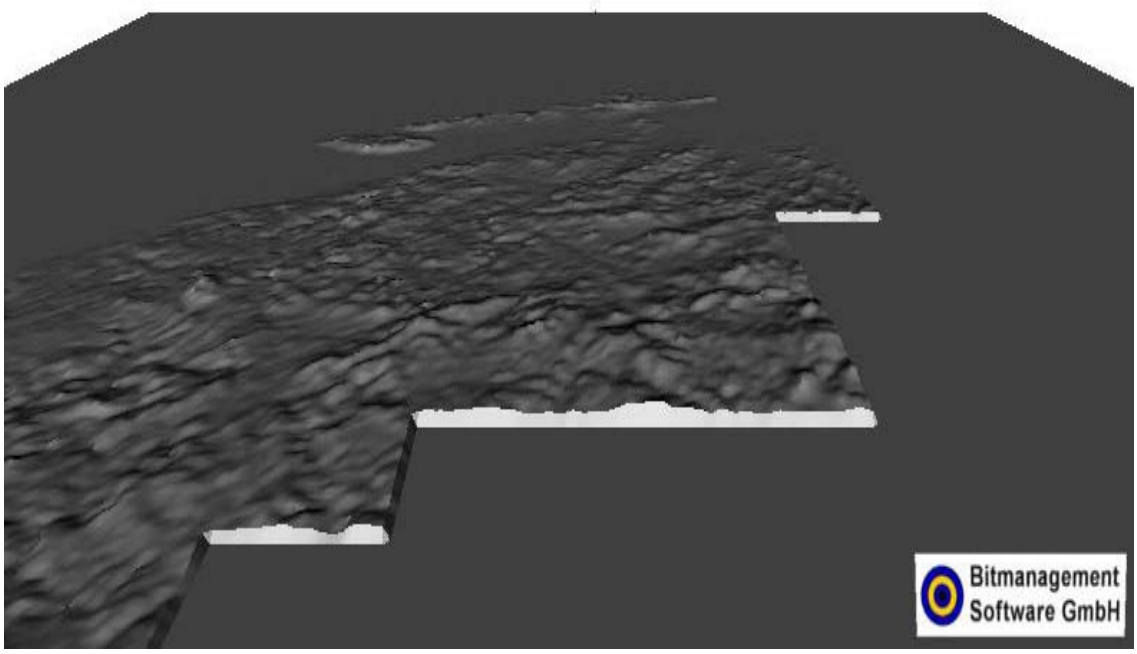
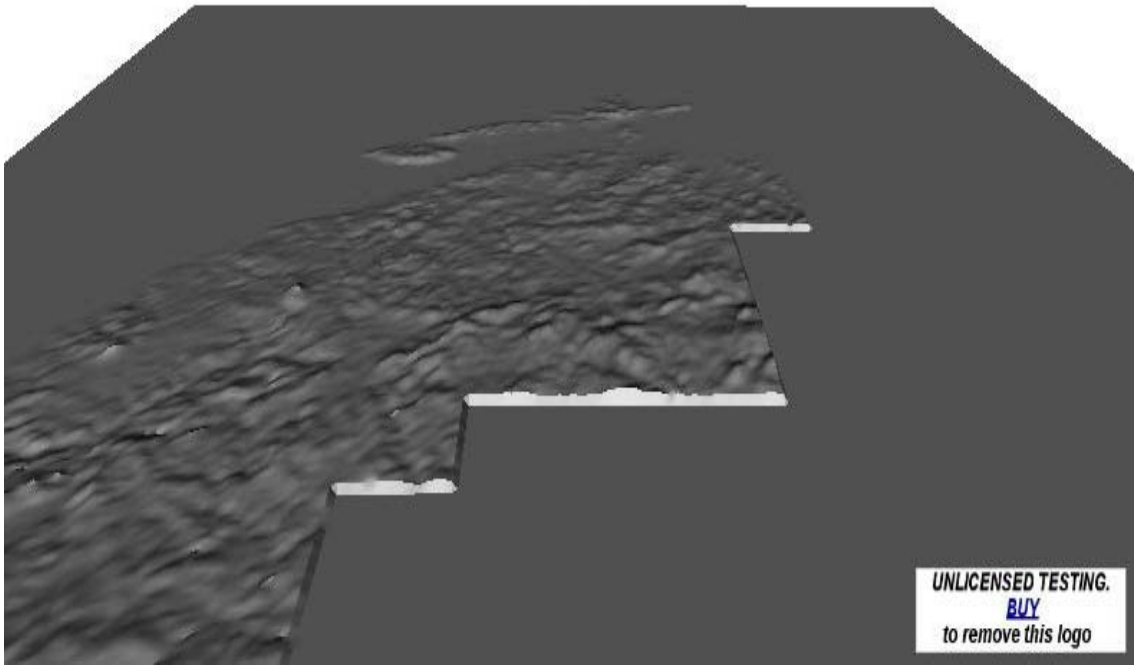


図 3-3-3：メッシュデータを間引いて間隔を調整する例

次ページの図に幾つかの間引き例を示す。

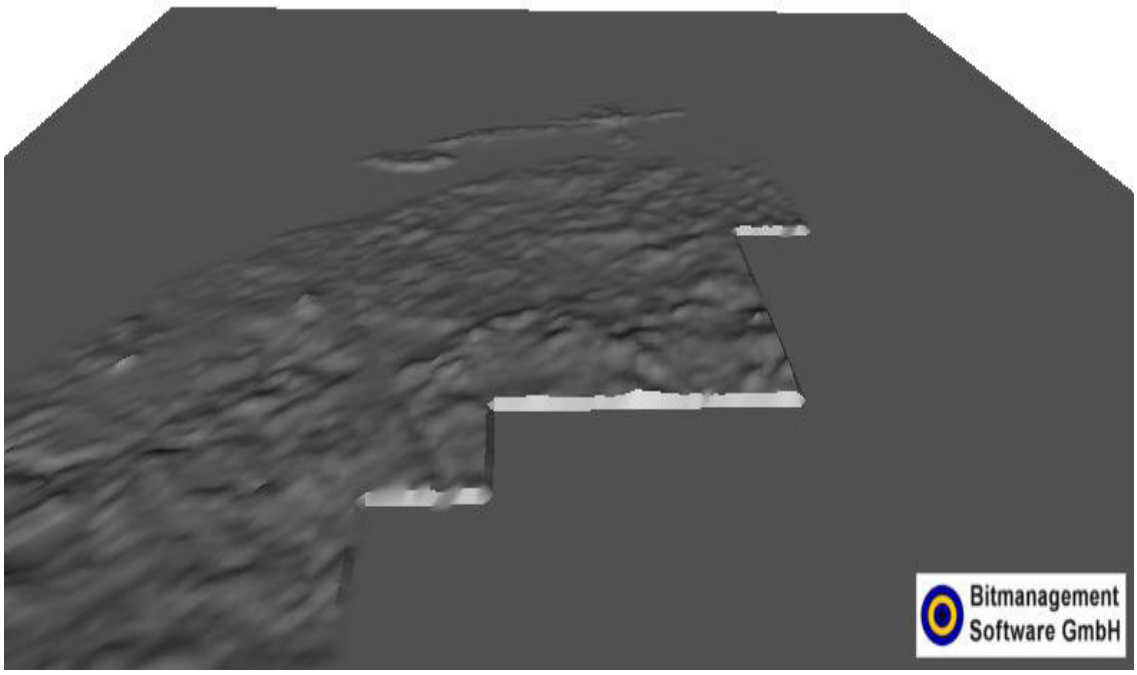


426*426　464.0KB

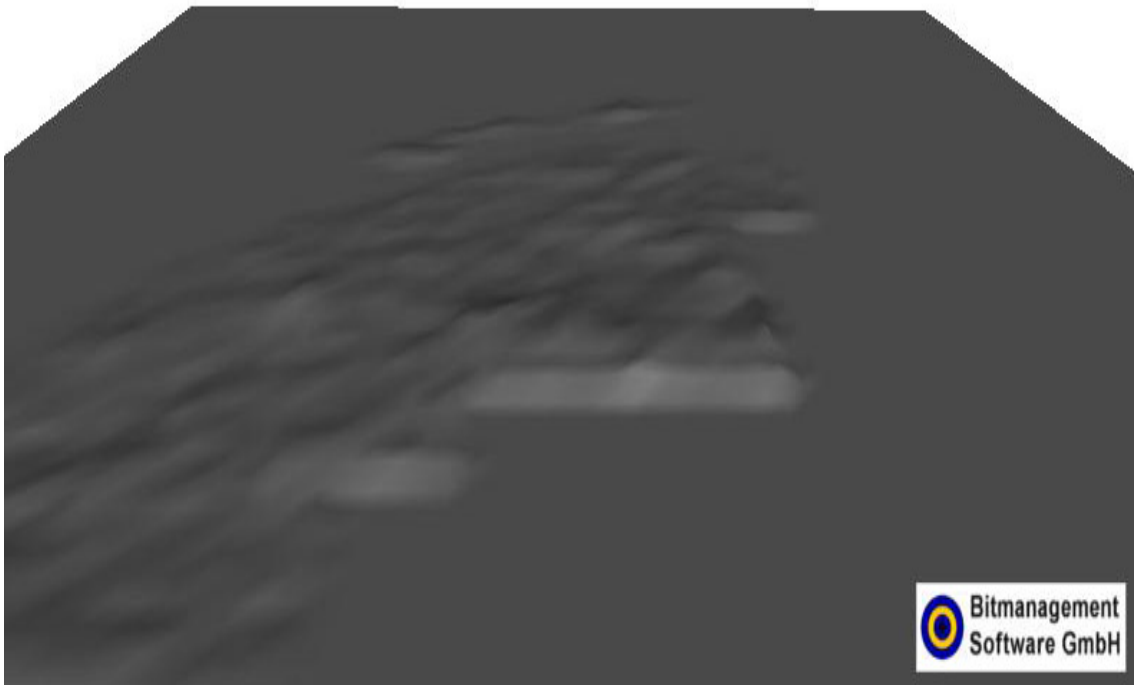


320*320　263.0KB

図 3-3-4：標高値データ間引き例 1



214*214　117.0KB



80*80　16.8KB

図 3-3-5：標高値データ間引き例 2

ElavationGrid ノードの xDimension、zDimension でそれぞれ *x* 軸と *z* 軸のメッシュ数を指定し、xSpacing、zSpacing によりメッシュの間隔を指定する。指定したメッシュ数、メッシュの間隔で、河川流域境界データに間引きを行った X3D データをクライアントに表示する。

メッシュ数 9、メッシュの間隔 50m の標高値データを表示する方法を例に示す。

例. メッシュ数：3*3　メッシュの間隔：50m の標高値データの表示方法

```
<Shape>
  <ElevationGrid   xDimension='3'   xSpacing='500'
                   zDimension='3'   zSpacing='500'>
   height=' 0     0     10
            30    50    20'
            40    29     0
  '/>
</Shape>
```

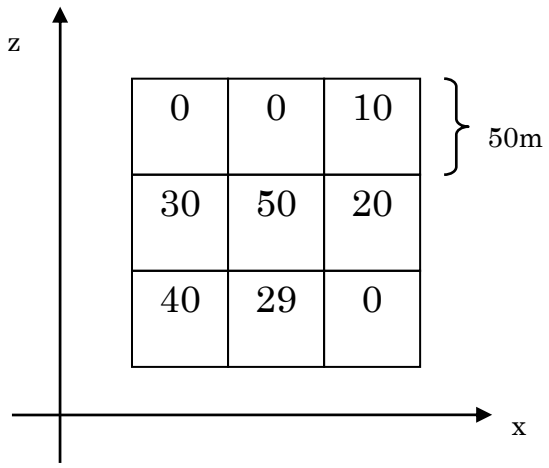


図 3-3-6：例の表示のされ方

先行研究[1]では、メッシュ数 640*640、メッシュの間隔 250m の標高値データが使用されている。この標高値データのメッシュ数を 320*320 まで落とす場合、メッシュの間隔は 500m に指定する。つまり、間引き後のメッシュ数が 1/2 になるときは、メッシュの間隔を 2 倍に指定し、メッシュの数が 1/3 になるときは、メッシュの間隔を 3 倍に指定してやればよい。このように、間引いた間隔に合わせてメッシュの間隔を調整することにより、ポリゴンデータと標高値データをずれることなく表示させる。

第４章　データ帯域測定

4.1　測定方法

動的に X3D データのファイルサイズを決定するためにデータ帯域を測定する必要がある。

まず、サーバマシンに 1Mbyte 程度の画像ファイルを置いておく。クライアントからの画像読み込み要求を受けた後、サーバ側では、JavaScript プログラムによって、画像タグ<img>が読み込まれる直前を読み込み開始時刻として取得しておく。また、<img>タグ中に onLoad イベントハンドラを記述することで、クライアント側が画像ファイルの読み込みを終えた瞬間の時刻を読み込み終了時刻として取得しておく。

終了時刻　－　開始時刻　＝　経過時間

となる。ファイルサイズを経過時間で割った商をデータ帯域とする。

ファイルサイズ　／　経過時間　＝　データ帯域

距離を時間で割ると速度が求まるのと同じ考え方である。比較的ファイルサイズの大きな画像ファイルを読み込むことで、誤差が少なく、マシンスペックも考慮した実データ転送速度を計測することができる。

なお、測定は以下の条件で行う。

- 単位は Mbps に変換する。
- JavaScript プログラムにより、5 回リロードを行いその平均の値を返す。
- 他のアプリケーションはなるべく起動させない。
- 画像のキャッシュは残さない。

画像のキャッシュを残すと、2 回目からの測定結果に誤差が含まれる。画像を読み込む度に、画像ファイルの後ろにクエリ「？」と、ユニークな文字列として、読み込んだ際の現在時刻を付け加えることで、これを回避する。

```
<script>
now = new Date().getTime();

img1 = new Image();
img1.src = "example.jpg?" + escape(now);
・・・
document.writeln('<img src = " ' +img1.src +' ">');
</script>
```

図 4-1-1：ユニークなキャッシュを残す方法

```
example.jpg?1162360574171
example.jpg?1162360573890
example.jpg?1162360573625
example.jpg?1162360573500
example.jpg?1162360573218
```

図 4-1-2：5 回画像ファイルを読み込んだ例（Temporary Internet File フォルダの中身）

4.2 測定結果

学外からサーバにアクセスし測定を行った。結果比較のために、サーバと同一 LAN である研究室の各クライアントからも測定を行った。

表 4-2-1：学外からの測定結果

番号	回線速度 (Mbps)	CPU クロック数(GHz)	CPU	帯域測定結果 (Mbps)
1	100	2.21	Athlon(tm) 64 processor	6.93934
2	100	3.00	Pentium 4	6.52111
3	100	3.06	Celeron	6.47088
4	100	1.86	Intel Core(tm) 2CPU 6300	6.34710
5	54	2.00	Pentium M Processor	6.29752
6	100	1.66	Athlon	5.15136
7	100	1.60	Celeron	4.85807
8	100	1.20	？（不明）	3.50098

9	100	0.30	x86 Family 6 Model 6 Stepping 10	1.70752
10	10	2.40	Celeron	0.74185
11	$64*10^{-3}$	1.50	Celeron	0.02771

表 4-2-2：研究室内（同一 LAN 内）からのアクセス

番号	CPU クロック数（GHz）	CPU	帯域測定結果（Mbps）
1	3.20	Pentium 4	25.84786
2	3.20	Pentium 4	25.06868
3	3.00	Pentium 4	20.74413
4	2.80	Celeron	17.39577
5	2.40	Celeron	16.18347
6	2.00	Celeron	10.13259
7	1.60	Celeron	7.40687
8	0.33	Pentium 2	4.70123
9	0.45	AMD-K6(tm) 3D Processor	3.91182
10	0.45	AMD-K6(tm) 3D Processor	2.85292
11	0.30	X86 Family 6 Model 6 Stepping10	2.14584

4.3　測定結果の考察

測定結果より

- データ帯域は回線速度と CPU クロック数のどちらにも依存し、比例の関係である。
- CPU クロック数の大きなマシンほど、回線速度に対するデータ帯域の傾きが大きくなる。
- メモリデータ容量は結果にあまり影響を与えない

ということが言える。その根拠を以下に示す。

データ帯域が CPU クロック数と回線速度に依存する

測定結果より、どちらの表も、CPU クロック数が大きいマシンほどデータ帯域の値が大きくなっている。表 4-2-1 の番号 1 と番号 9 の例で見ると、回線速度はどちらも同様に 100M

であるが、CPU クロック数はそれぞれ 2.21GHz、0.30GHz であり、測定結果は 6.93934Mbps、1.70752Mbps である。CPU の種類が異なるため純粋な比較はできないが、この結果より測定結果が CPU クロック数に依存していることが分かる。

同様に、回線速度が大きな環境ほどデータ帯域の値が大きくなる。表 4-2-1 の番号 7 と番号 11 はそれぞれ、Celeron 1.60GHz と Celeron 1.50GHz であるが、回線速度は 100Mbps、$64*10^{-3}$Mbps と大きな差があり、測定結果も 4.85807Mbps と 0.02771Mbps となっている。この結果より回線速度が大きい環境であれば測定結果が大きくなることが分かる。

データ帯域は CPU クロック数に比例する

図 4-3-1 は、同一のネットワーク環境下（100Mbps）での 3 台のマシンの測定結果比較である。図より、CPU クロック数とデータ帯域が比例の関係になっている。

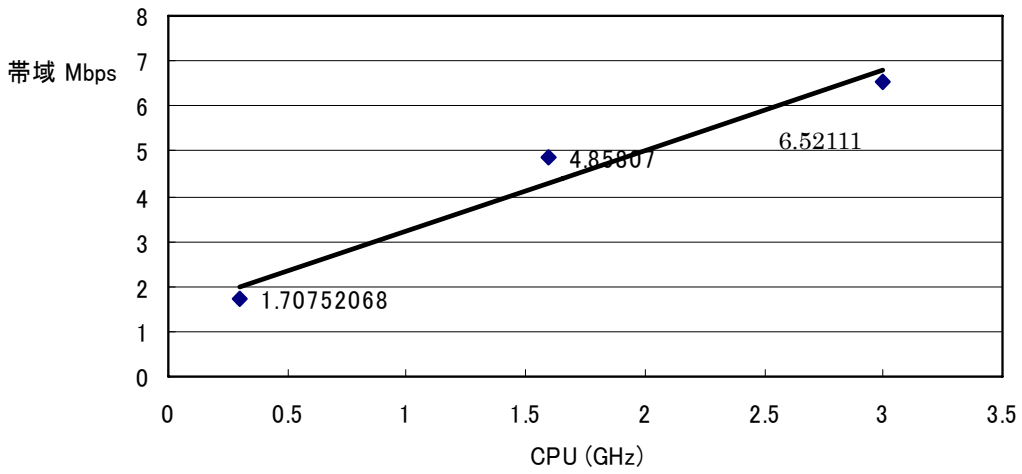


図 4-3-1：同一回線上での 3 台のマシンでの結果の違い

CPU クロック数の大きなマシンほど回線速度に対するデータ帯域の傾きが大きくなる

図 4-3-2 は、2 台のマシンでのネットワーク環境下の違いによる結果の違いである。図より、回線速度が大きいほど、データ帯域も大きくなり、比例の関係であることが分かる。また、CPU クロック数の大きなマシンの傾きが大きくなっている。

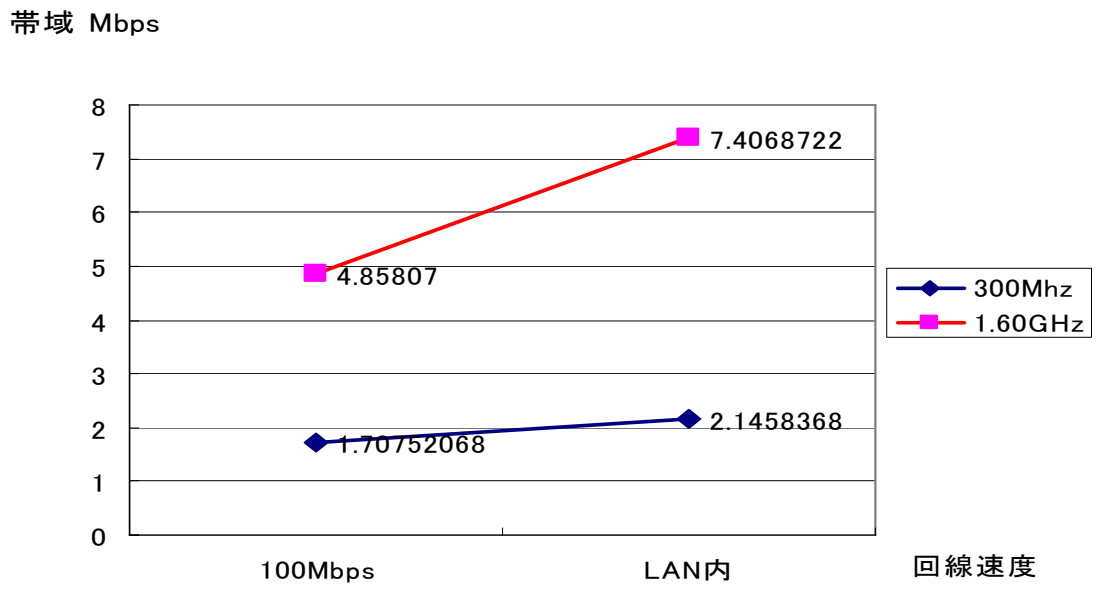


図 4-3-2：同一マシンでの測定場所による結果の違い

メモリデータ容量は結果にあまり影響を与えない

両表の結果とも、メモリデータ容量を全く考慮せずに測定を行った。結果として、ほぼ CPU スペック数の大きな順にデータ帯域は降順に並ぶため、充分なメモリデータ容量を有する通常の PC では、メモリデータ容量が結果に与える影響は小さいと判断できる。

第５章　非同期通信

より実用的なシステム構築のため、動的なファイル出力を非同期的な通信で行う。非同期通信[6]とは、クライアントとサーバ間で同期を取らずに通信することである。

同期的な通信の場合、現在ブラウザに表示しているページから他のページへ移動する際にはリロードを行う必要があるのに対し、非同期的な通信の場合は、他のページへ移動する際にも絶えずサーバとの通信を行うため、ページ遷移が発生しない。このように、非同期通信はシステム利用者にサーバの存在を意識させず、ブラウザの制御を奪うことなしに HTTP 通信を行うことができる。

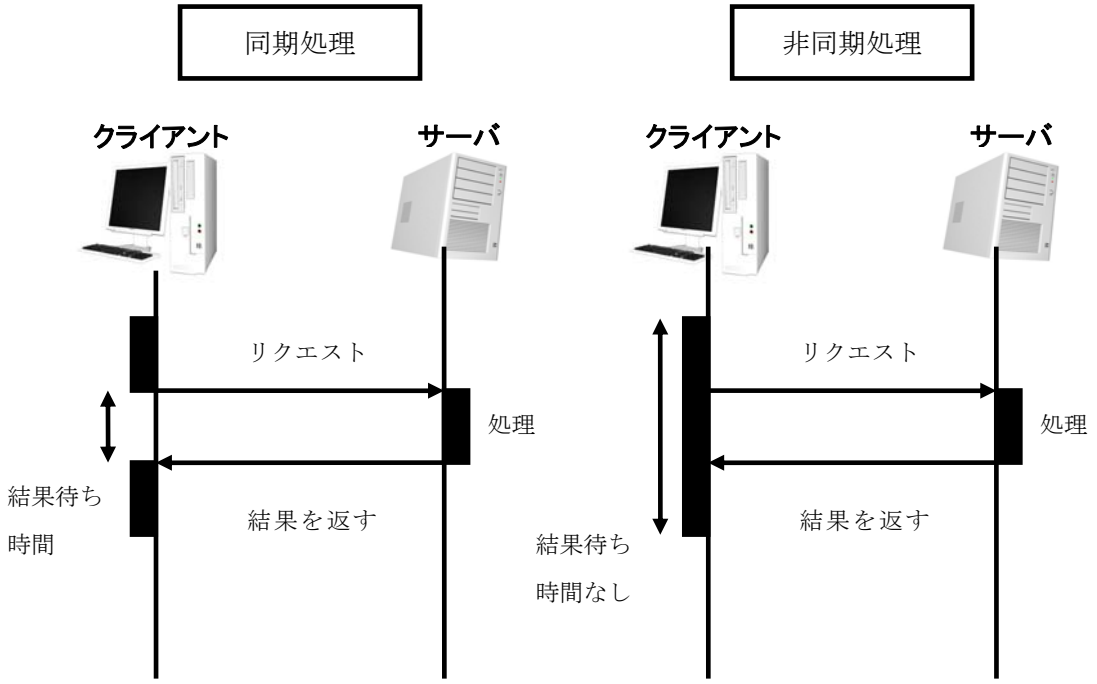


図 5-1：同期通信と非同期通信の違い

5.1 Ajax について[7]

Ajax とは Asynchronous JavaScript + XML の略語であり、JavaScript、Dynamic HTML、XML などの既存技術を組み合わせる手法のことを指す。

Ajax では、JavaScript の組み込みクラスである XMLHttpRequest を利用することで非同期通信を実現できる。XMLHttpRequest により、Web ページのリロードを行わずにサーバと XML 形式のデータのやりとりを行い、CSS（Cascading Style Sheets）で定めたレイアウトによりブラウザへ情報を表示する。Ajax の利点を次に示す。

・プラグインを必要としない

Ajax は特別新しい技術ではなく既存技術の組み合わせであるため、ブラウザにプラグインをインストールする必要がない。

・最小限のデータを必要なときに取得すればよい

非同期通信では、必要になったときに必要なデータだけを読み込めばよいため、通信に無駄が生じない。

次に、問題点を示す。

・セキュリティ上の制限から異なるドメイン間の通信ができない
・ブラウザ間の仕様差を考慮する必要があり実装に手間がかかる

実際に Ajax を用いて実装を行っている Web アプリケーションの例として、Google Maps や Google Earth、Google Suggest 等が挙げられる。それらを実際に使用してみれば非同期通信のイメージを掴みやすい。

5.2 非同期的なファイルアクセス

Ajax を使用し、X3D ファイルへのアクセスを非同期的に行えるようにした。非同期通信を用いない場合、一度 X3D ファイルを表示した後に違うファイルへアクセスするためには、ブラウザの「戻る」でページのリロードを行い、再び全ての情報を読み込む必要がある。

今回実装したシステムが X3D ファイルを出力する手順を次に説明する。

1. データ帯域測定プログラム(JavaScript)が帯域を測定する。
2. 測定結果を X3D ファイル表示プログラム（Ajax）に渡す。
3. X3D ファイル表示プログラムは、測定結果より、最適と思われるサイズの X3D ファイルを表示スペースに出力する。
4. 出力した X3D ファイルのサイズが大きすぎたり、逆に小さすぎたりする場合、サイズの異なるファイルへのアクセスを非同期で行う。

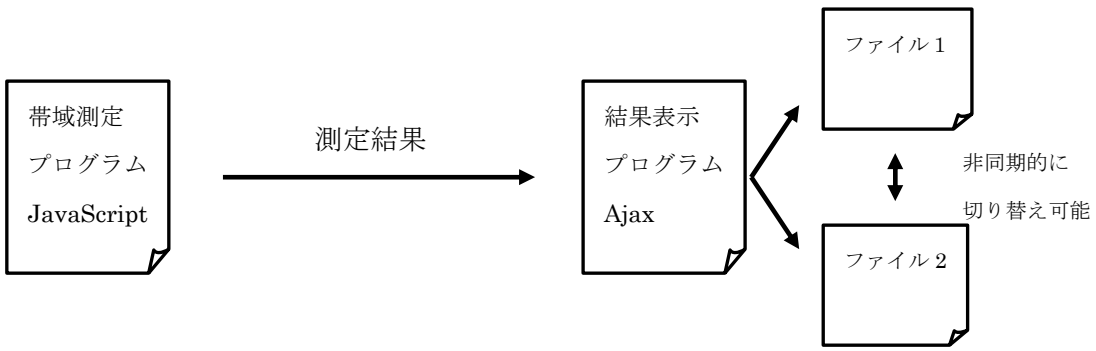


図 5-2-1：サーバ上のプログラムの動き

ここで、手順 2 では、データ帯域測定結果を JavaScript プログラムから Ajax プログラムに直接渡している（どちらのプログラムもサーバに置いておく）。JavaScript ファイルが測定結果を文字列として Ajax ファイルに送信し、Ajax ファイルはそれを数値に変換、判定を行う。

サーバには予めサイズの違う 5 種類の X3D ファイルを用意しておき、Ajax ファイルが最適なファイルを選択して出力する。用意した X3D ファイルは以下の 5 つである。

表：5-2-1　サーバに用意しておく 5 つの X3D データファイル

番号	ポリゴンデータのサイズ	標高値データのサイズ	合計サイズ
1	438 KB　(ε=300)	464 KB　(426*426)	906 KB
2	365 KB　(ε=500)	262 KB　(320*320)	630 KB
3	211 KB　(ε=3000)	262 KB　(320*320)	474 KB
4	202 KB　(ε=4000)	117 KB　(214*214)	321 KB
5	193 KB　(ε=8000)	66　KB　(160*160)	257 KB

Java Servlet や PHP を用いれば、間引く前の元のデータだけ用意しておくことで動的に間引いたデータを出力することも可能であるが、今回はそれらの技術を用いなかった。

その理由は、Java Servlet と JavaScript 間のデータ通信が非常に複雑・困難であったからである。両者の間に Java Applet を挟むことで、JavaScript→Applet→Servlet という流れで容易にデータ通信を行うことができるが、Java Applet の起動が重たいことや、起動自体ができない環境にある利用者もいること等から汎用性に欠けると判断し、使用しなかった。

PHP の場合も同様に、サーバ側のプログラムが非常に複雑になるため、使用しなかった。本研究での目的はあくまでもファイルサイズ間での非同期通信の実現であり、より複雑な非同期通信を実装する場合にはこれらの技術を用いる必要がある。

実際のインタフェースは次ページのようになる。

図 5-2-2：本システムのインタフェース

第 6 章　考察・今後の課題

本研究により、法線方向の向きの統一、ファイルアクセスの非同期化は実現できた。また、データ帯域を測定し、その測定結果に依存して動的にファイルサイズを出力することで、比較的マシンスペックやネットワークの性能が低い環境の利用者でもストレスを感じることなく利用できるシステムを実装することができた。

今回は実現できなかった機能を、今後の課題として以下に示す。

- **Ajax による 3D データの非同期通信**

  本研究で実現した Ajax の非同期通信機能は、一度 X3D ファイルを出力した後に他の X3D ファイルへアクセスする場合、そのアクセスを非同期にするというものである。よって、3D データを部分的に非同期に読み込んだり、読み終えたりしている訳ではない。ファイルアクセスだけでなく、この部分も非同期で通信できるようにしたい。丁度 Google Earth のような動作が目指すところである。

- **ポリゴン同士の曲線の交差の回避**

  ノードは間引かず、Douglas-Peucker のアルゴリズム[5]を用いて間引きを行ったが、ポリゴンデータの精度が落ちれば落ちるほど、ポリゴン同士の曲線が交わることが多くなる。隣接行列を用意したデータ構造を作成すれば、隣り合うポリゴンとの重なり合いの判定を行うことで、曲線同士の重なり合いを回避できると考えられる。

- **DBMS の利用による検索機能の追加**

  先行研究の段階で挙げられていた課題であったが今回も実現はできなかった。人口や道路、その他のデータを加え、システムに検索機能を付け加えたい。例として、

  ・斐伊川の流域左右 2km における総人口の算出
  ・過去に水害が起こった流域を通る国道の検索
  ・河川の汚染の被害拡大予想

  といった機能が挙げられる。

第７章　謝辞

謝　辞

本研究にあたり、最後まで熱心な御指導をいただきました田中先生には、心より御礼申し上げます。また、田中研究室の皆さんからは本研究に関して数々の御協力と御助言を、学外からのデータ帯域測定に協力して下さった方々からは貴重な実験データをいただきました。厚く御礼申し上げます。なお、本論文、本研究で作成したプログラム及びデータ，並びに関連する発表資料等のすべての著作権を、本研究の指導教官である田中教授に譲渡致します。

文献

[1]　島村和義「ブラウザ環境における河川流域境界データ表示システムの試作」
島根大学総合理工学部　数理・情報システム学科　卒業論文　2005

[2]　数値地図ユーザーズガイド　監修　建設省国土地理院　（平成 10 年 1 月 1 日）

[3]　JMC マップ（日本）（財）日本地図センターCD-ROM　Readme.txt

[4]　数値地図５０ｍメッシュ（標高）日本-Ⅲ 国土地理院 CD-ROM Index.htm

[5]　David H. Douglas and Thomas K. Peucker, “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature,” Canadian Cartographer Vol 10 No 2 December 1973, pp. 112-122.

[6]　Web プログラミングチュートリアル
http://grape.sapid.org/tutorial/lec7.html

[7]　羽田野太巳「Ajax　Web アプリケーション　アイデアブック」
株式会社　秀和システム　2006 年 4 月　pp.12-26