非構造四面体格子用等値面描画手法の高速化

杉 原 光 太†

著者は数千万，数億規模の大規模な非構造格子上の計算結果を並列ベクトル型スーパコンピュータなどの並列計算サーバ上で高速に可視化表示する等値面描画手法を開発した．本論文では，アルゴリズムの詳細と NEC 製のベクトル並列スーパコンピュータ SX-6 上での性能結果を報告する．本等値面描画手法は，ネットワーク分散環境下での大規模計算結果の高速可視化を可能にすることを目的とし，非構造格子のような不規則格子上で処理負荷が膨大であるスムーズシェーディング処理をハッシュ法の使用とベクトル化により高速処理している点に特徴がある．さらに並列処理にも向いている．SX-6 上で性能評価した結果，スムーズシェーディング処理をした半透明等値面の描画 CPU 時間は 4000 万格子で 7.14 秒，3 億格子で 47.5 秒になり，格子規模依存性は，格子数の 2/3 乗のオーダ以上線形オーダ未満の範囲に抑えられたことを確認した．

Speed-up of Generating Isosurfaces from Volume Data on Tetrahedron Grids

Kouta Sugihara†

In this research, the author and his colleagues developed a solution to make smooth shading process of an isosurface faster by reducing computing cost. Using the hash method and developing a new algorithm suited for vector and parallel computing, we reached our goal: we shortened the total time for isosurface generation with unstructured grid and reduced the dependency of visualization time on the number of grid elements. Numerical experiments with the new algorithm on the NEC's SX-6 supercomputer have shown the following effectiveness: 1) When the number of elements is about 300,000,000, the time required to generate a smooth-shaded isosurface from the data on a tetrahedron mesh will be 47.5 seconds, which is 41 times faster than the prototype visualization function of isosurfaces. 2) The dependency of visualization time on the number of elements will become nearly linear, while it was on the order of 4/3 before.

1. はじめに

コンピュータの処理能力の向上にともない，数千万，数億点格子の大規模非定常数値シミュレーションが行われている．このような大規模シミュレーションの可視化システムとして，NEC では，リアルタイム可視化システム RVSLIB [1)]（Real-time Visual Simulation Library）を開発している．RVSLIB は計算サーバ上でシミュレーション実行と同時にその結果に対する動画を作成し，圧縮された画像をネットワークを通してクライアント端末で可視化表示するシステムである．本システムはネットワーク負荷が画像サイズによってのみ規定され，サーバ側の数値シミュレーション規模によらず小さくほぼ一定に抑制できる点に特徴がある．この特徴は，ネットワーク分散環境下での大規模データの可視化には効率的である．ここでネットワーク分散環境とはインターネットなどのネットワーク上で複数の計算/可視化サーバや PC などの端末表示装置がつながれている環境を指す．

RVSLIB が採用している可視化方式では，画像生成をベクトル計算機，並列計算機上などの計算サーバ上で行うため，シミュレーション時間に対して可視化処理時間を小さくする必要があり，計算サーバ上での高速な可視化技法が必須になる．非構造格子上での可視化処理時間は，格子間の接続関係に規則性がないため，構造格子上での可視化処理時間より長い．また 2 次元図である断面上の等値線図の可視化処理時間よりも 3 次元的な物理分布を表現する等値面の可視化処理時間の方が長い．さらに等値面の描画処理においてはスムーズシェーディング処理の高速化が課題となる．等値面の描画処理は，等値面を構成するポリゴンを抽出

† 日本電気株式会社 HPC 販売推進本部
HPC Marketing Promotion Division, NEC Corporation

するマッピング処理とそのポリゴンに対するレンダリング処理に分かれる．スムーズシェーディング処理をしない場合，マッピング処理の CPU 時間は格子数の線形オーダ，レンダリング処理の CPU 時間はポリゴン数に比例し，格子数の 2/3 乗のオーダになると見積もられる．しかしスムーズシェーディング処理を行うと，ポリゴン頂点上の法線ベクトルをその頂点を共有する全ポリゴンの法線ベクトルの和を規格化した値として計算する必要がある（**図 1**）．このため各ポリゴン頂点を共有するポリゴン番号と対応するポリゴン内のローカルな頂点番号を検索する処理が必要になり，ポリゴン頂点上の法線ベクトルの計算量は最悪ポリゴン数の 2 乗のオーダになると推定され，格子数に対しては 4/3 乗のオーダになると推定される．よってスムーズシェーディング処理の際のポリゴン頂点上の法線ベクトルの計算が等値面の描画計算のボトルネックとなることが分かる．

これらの背景をふまえ，課題解決のため今回の研究では以下のことを行った．

- ポリゴン頂点番号は隣接ポリゴン間で共有するようにし，ポリゴン頂点の検索処理をハッシュ法により高速処理し，その計算量の格子規模依存性を格子数の 2/3 乗のオーダに抑制．
- 負荷が大きく，ポリゴン頂点の検索処理が含まれるスムーズシェーディング処理での法線ベクトルの計算の大部分をベクトル処理により高速化し，等値面描画全体の時間を短縮すると同時に，その格子規模依存性を格子数の 2/3 乗のオーダ以上線形オーダ未満の範囲に抑制．

開発した描画手法の処理速度を評価した結果，スムーズシェーディング処理をした半透明等値面の描画 CPU 時間は，NEC 製スーパコンピュータ SX-6 上で 38,811,960 格子の場合は 7.14 秒，310,495,680 格子の場合は 47.5 秒になり，格子規模依存性は，格子数の 2/3 乗のオーダ以上線形オーダ未満の範囲になった．なお，今回は非構造格子のうち，四面体格子用等値面描画手法の高速化を実施した．

本論文は非構造四面体格子用の等値面描画手法の高速アルゴリズムとその性能評価結果について報告する．本論文の 2 章では関連する従来技術について触れ，これらの問題点を明らかにする．3 章では，ポリゴン頂点上の法線ベクトルの計算手法を検討する．4 章では今回開発した等値面描画手法のアルゴリズムを述べる．5 章では SX-6 上での等値面描画手法の性能評価結果を述べる．なお，本論文の等値面描画手法は Fortran で実装している．

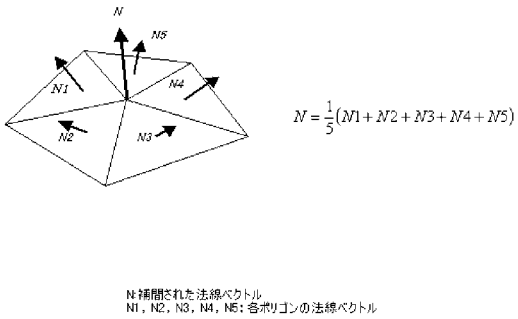

図 1 ポリゴン頂点上の法線ベクトルの補間
Fig. 1 Interpolation of the normal vector on a vertex of a polygon.

2. 関連する従来技術

非構造格子上での等値面描画手法の高速化を図る研究として，解像度を考慮して等値面を構成するポリゴン数を削減したり，計算結果を階層構造に格納しておき，可視化したい部分や可視化精度に応じて必要なデータをネットワーク転送して可視化サーバ上で可視化したりすることにより，描画時間の短縮を図る手法がある[2)~5)]．これらの方法は，ネットワーク上を転送し可視化サーバ上で扱うべきデータ量を削減する点で描画時間の短縮には有効である．しかしながら，非構造格子への適用となると計算結果を階層構造に格納する手法は，可視化したい部分や可視化精度に応じて階層化された計算結果を再構成する処理速度がインタラクティブな可視化のためには十分ではなく，実用化には至っていない．一方，等値面を構成するポリゴンの抽出時間を短縮することで等値面の描画時間の短縮を図る研究[6)~8)]がある．いずれの手法も計算領域内の格子の中で等値面を構成する格子はごく一部であることが多いため，等値面と交差しない格子の多くに対する処理を省略することによって等値面生成処理の高速化を実現している．しかしながら，文献 8) の手法は等値面生成を高速処理するための前処理の計算時間が格子数の線形オーダ以上になるという問題点がある．一方，文献 6) はこの前処理の計算時間が格子数の線形オーダ以下という利点があるが，非構造格子の場合，格子ごとに隣接する格子番号を記憶する必要がある[9)]．このために必要なメモリ容量は格子数の線形オーダで増加するため，格子規模が大きいときに必要なメモリ容量が膨大になるという問題点がある．文献 10) は等値面と交差する格子内にポリゴンを生成する際に，ポリゴン頂点を隣接するポリゴン間で共有するように処理を実装している．これによりポリゴン頂点を隣接するポリゴン間で共有しない場合と比較して，必要な

メモリ容量の低減とポリゴン頂点の座標値や法線ベクトルの計算時間の短縮を実現している．ポリゴン頂点を共有するように実装するためには隣接ポリゴン間でポリゴン頂点を検索する処理が必要だが，ハッシュ法によってこの検索処理の高速化を図っている．この手法は有効だが，文献 10) の実装では，ポリゴン頂点上の法線ベクトルの計算がベクトル処理に不向きであるという問題点がある．文献 7) は文献 6) が提案する高速等値面生成手法を取り込み，さらにポリゴン頂点の検索処理を行わずに，隣接ポリゴン間のポリゴン頂点の共有を実現している．具体的には等値面と交差する格子の内部におけるポリゴンを生成したときに，そのポリゴンの頂点が接する格子辺について，格子辺を共有するすべての格子を探索し，探索した格子の内部に生成されるポリゴンに対して，同じポリゴン頂点を登録する．この手法はハッシュ法によるポリゴン頂点の検索処理を行う手法に比べ，一度登録されたポリゴン頂点を検索する必要がないため，隣接ポリゴン間のポリゴン頂点の共有を実現させた，より高速な等値面生成手法であることが報告されている．しかしながら，文献 6) と同様に必要なメモリ容量が格子数の線形オーダという問題点がある．著者は格子数が数千万以上の大規模問題に対しては，必要なメモリ容量の低減が重要と考え，文献 7) が提案するポリゴン頂点の共有化処理の速度向上を犠牲にして，必要なメモリ容量を格子数の 2/3 乗のオーダに抑えた．本論文では文献 10) と同様に，ハッシュ法によるポリゴン頂点の検索処理を行うが，この処理の過程でポリゴン頂点上の法線ベクトルの計算をベクトル処理向きにする配列を生成する．この配列は 4.2 節で説明するポリゴン頂点番号テーブルに相当する．本論文のベクトル処理による高速化手法は，文献 7) に対して犠牲にしたポリゴン頂点の共有化処理の速度向上分を補うだけでなく，文献 10) に対しても描画時間全体を短縮するという利点がある．

3. ポリゴン頂点上の法線ベクトルの計算手法

ここでは計算サーバ上でスムーズシェーディング処理した場合，数千万以上の格子規模では等値面描画処理全体において大部分を占めるポリゴン頂点上の法線ベクトルの計算手法を検討する．検討のため，本章で述べる四面体格子用等値面描画手法のプロトタイプを実装し，SX-6 上で処理速度の性能評価を行う（以後この手法をプロトタイプ版と呼ぶ）．ポリゴン頂点上の法線ベクトルの計算では，ポリゴン頂点を共有するポリゴン番号を抽出する処理の計算量が膨大となるため，この部分の計算手法を工夫することが処理の高速化において必須である．ポリゴン頂点の検索処理はポリゴン頂点を共有するポリゴン番号と対応するポリゴン内のローカルな頂点番号の抽出処理を意味する．さらにポリゴン内のローカルな頂点番号とは各ポリゴン内の 3 つの頂点に割り振られた頂点番号（1，2，3）を指す．プロトタイプ版のポリゴン頂点の検索処理ならびにポリゴン頂点上の法線ベクトルの計算のアルゴリズムを説明する．まず計算領域を 32^3 個の部分空間に分割して，同一部分空間中のポリゴン頂点の座標値の比較によりポリゴン頂点の検索処理を行う．ポリゴン頂点上の法線ベクトルの計算方式は，ポリゴン頂点の検索処理をしながら逐次処理で行う（**図 2**）．プロトタイプ版は従来技術のハッシュ法を用いずに，検索処理の対象とするポリゴン頂点を同一部分空間にあらかじめ絞り込むことで処理の高速化を図る．ここではプロトタイプ版の有効性を吟味するため，等値面描画の処理速度を評価した．1 辺の長さ 100 の立方体領域に直交格子を張り，各直交格子を 5 分割して四面体格子を生成した計算領域内に，各節点上のスカラ値が立方体領域の中心からの距離であるテストデータを使った．スムーズシェーディング処理をともなう等値面の描画に要する CPU 時間は格子数が 38,811,960 の場合は 2 分，格子数が 310,495,680 の場合は 30 分となった．ここで最も計算コストの高い処理は等値面を構成するポリゴン頂点上の法線ベクトルの計算部分であり，4,851,495 格子，38,811,960 格子，310,495,680 格子で各々全体の描画の CPU 時間の 75%，91%，96%を

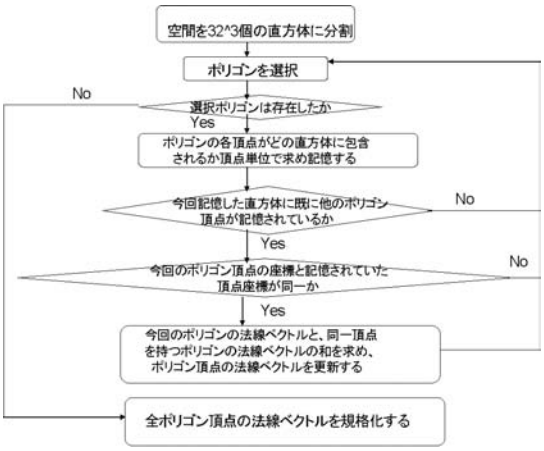

図 2 プロトタイプ版のポリゴン頂点上の法線ベクトルの計算のフロー

Fig. 2 Flowchart of calculating the normal vector on the vertices of the polygons with the prototype visualization function of isosurfaces.

表 1 プロトタイプ版のポリゴン頂点上での法線ベクトルの計算の割合
Table 1 Cost of calculating the normal vector on the vertices of the polygons with the prototype visualization function of isosurfaces.

格子数	描画時間に対する法線ベクトルの計算時間の割合
4,851,495	75%
38,811,960	91%
310,495,680	96%

占めることが明らかになった（**表 1**）．このテストデータを用いた場合，ポリゴン頂点の検索処理の CPU 時間は格子数の 4/3 乗のオーダで増加するが，その理論的根拠を説明する．32^3 個に分割した部分空間のうち，ポリゴン頂点を含む部分空間の数は格子数によらず一定でこの数を m とする．等値面は球面になり，球面と交差する格子数は立方体領域内の格子数の 2/3 乗のオーダになる．したがってポリゴン数 $npolygon$ と全ポリゴン頂点数 $nvertex$ は格子数の 2/3 乗のオーダに見積もられる．ここで全ポリゴン頂点数は，ポリゴン頂点を隣接ポリゴン間で共有させたときの等値面を構成するポリゴンの頂点数を指す．検索対象のポリゴンの頂点数は $3 * npolygon$ なので，テストデータを用いた場合，ポリゴン頂点の検索処理の計算量は $(3 * npolygon) * (nvertex/m)$ と見積もられることから，その格子規模依存性は格子数の 4/3 乗になる．ポリゴン頂点の検索処理の計算量の格子規模依存性は 1 章で述べたように，理論上格子数の 4/3 乗以上のオーダにはならないので，プロトタイプ版のポリゴン頂点の検索処理の手法ならびに法線ベクトルの計算手法は有効とはいえない．

ポリゴン頂点上の法線ベクトルの計算は，ポリゴン頂点の検索処理と，各ポリゴンの法線ベクトルを計算し，共有ポリゴンの法線ベクトルの和をとってポリゴン頂点上の法線ベクトルを求める部分に分かれる．ポリゴン頂点の検索処理はベクトル処理に向かないので，それ以外の部分をベクトル処理できるようにする．さらにポリゴン頂点の検索処理の計算オーダを削減する必要がある．ポリゴン頂点上の法線ベクトルの計算は，ポリゴン頂点を共有しているポリゴン番号とポリゴン内のローカルな頂点番号の情報が前もってあれば，ポリゴン数をループ長とするベクトル処理が可能である．そこでポリゴン頂点上の法線ベクトルの計算を以下の手順で行う．

- 法線ベクトルの計算の前にポリゴン番号とポリゴン内のローカルな頂点番号に対する全体ポリゴン頂点番号を表す配列を計算．
- ポリゴン番号とポリゴン内のローカルな頂点番号に対する全体ポリゴン頂点番号を表す配列を利用して，ポリゴン頂点上の法線ベクトルの計算をポリゴン数をループ長としてベクトル処理．

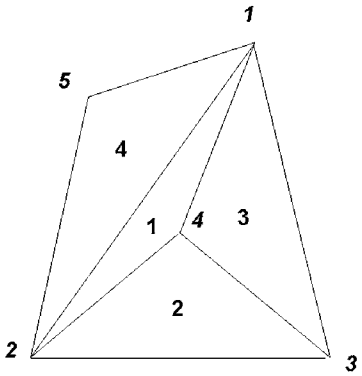

図 3 全体ポリゴン頂点番号
Fig. 3 Global numbering of vertices of polygons.

図 3 で全体ポリゴン頂点番号とポリゴン番号，ポリゴン内のローカルな頂点番号の対応を説明する．仮に等値面が 4 つのポリゴンから構成されるとする．4 つのポリゴンに対して，1，2，3，4 のポリゴン番号が割り振られる．等値面を構成するポリゴン頂点の数は 5 つで斜体文字で 1，2，3，4，5 の全体ポリゴン頂点番号が割り振られる．ポリゴン内のローカルな頂点番号とは各ポリゴンの 3 つの頂点に 1，2，3 と割り振られる番号である．たとえばポリゴン 1 のローカルな頂点番号 1 は全体ポリゴン頂点番号 1 に対応し，ローカルな頂点番号 2 は全体ポリゴン頂点番号 2 に対応し，ローカルな頂点番号 3 は全体ポリゴン頂点番号 4 に対応する．ポリゴン 2，ポリゴン 3，ポリゴン 4 についても同様である．

ポリゴン頂点の検索処理は 1 つの等値面あたり，四面体格子上の辺には多くて 1 つのポリゴン頂点しか乗らないことを利用し，ポリゴン頂点が乗る辺（両端の節点番号対）により行う．ポリゴン頂点が乗る辺の両端の節点番号ならびに全体ポリゴン頂点番号の生成，管理にはハッシュ法を用いた．ハッシュ長をポリゴン数に比例させた場合，ポリゴン頂点の検索処理の計算量はポリゴン数の線形オーダに抑えられる．ポリゴン頂点の検索処理をポリゴン頂点が乗る辺（両端の節点番号対）の比較により行うが，この比較の前にポリゴン頂点が乗る辺番号（格子内のローカルな辺番号）と等値面が通る格子内のポリゴン数（1 か 2）を求める必要がある．各格子単位での，ポリゴン頂点が乗る辺番号と等値面が通る格子内のポリゴン数を基本的なポリゴンデータと呼ぶことにする．以後今回開発した非構造格子用等値面描画手法を高速版等値面描画手法と

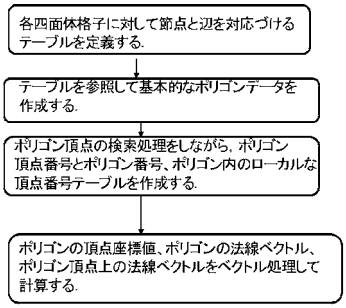

図 4 高速版等値面描画手法のポリゴン頂点上の法線ベクトルの計算のフロー
Fig. 4 Flowchart of calculating the normal vector on the vertices of the polygons with the improved isosurface function.

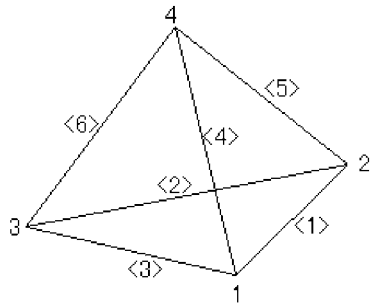

図 5 四面体上のローカルな頂点番号と辺番号
Fig. 5 Local numbering of vertices and edges on a tetrahedron.

表 2 辺番号と辺の両端の節点番号の対応
Table 2 Correspondence between an edge and its end points.

辺番号	節点対
1	1-2
2	2-3
3	3-1
4	1-4
5	2-4
6	3-4

呼ぶ.

高速版等値面描画手法の，ポリゴン頂点上の法線ベクトルの計算アルゴリズムは以下の **Step 1**，**Step 2**，**Step 3** から成る（**図 4**）.

Step 1 基本的なポリゴンデータの作成.

Step 2 ポリゴン頂点の検索処理を行いながら，ポリゴン頂点番号とポリゴン番号，ポリゴン内のローカルな頂点番号テーブルを作成し，ポリゴン頂点の座標値を計算.

Step 3 ポリゴン頂点上の法線ベクトルの計算，規格化.

高速版（図 4）はプロトタイプ版（図 2）と比較して，以下の要因によって処理の最適化が図られている.

- ポリゴン頂点の検索処理には，空間分割による手法ではなく，ハッシュ法を使用し，さらにハッシュ長をポリゴン数に比例させることにより，計算量をポリゴン数の 4/3 乗のオーダから線形オーダに改善.
- 法線ベクトルの計算はポリゴンに関する逐次処理ではなく，全体ポリゴン頂点番号とポリゴン番号，ポリゴン内のローカルな頂点番号テーブルを用いる方式に変更．これによりポリゴン数，全ポリゴン頂点数のループ長でベクトル処理可能．ポリゴン数，全ポリゴン頂点数は格子数の 2/3 乗のオーダであり，格子数が大規模なほどループ長が長くなるため，法線ベクトルの計算はベクトル処理により高速化可能.

Step 1，**Step 2** のポリゴン頂点の座標値の計算，および **Step 3** はそれぞれ格子数，全ポリゴン頂点数，ポリゴン数，全ポリゴン頂点数をループ長とする DO ループの中でベクトル処理される．**Step 1**，**Step 2**，**Step 3** の処理の後，ポリゴン頂点での輝度値を計算する．ボリゴン内部点の輝度をポリゴン頂点での輝度を線形補間して求めることによりポリゴンの平滑化を行う（グーローシェーディング法）．最後に，ポリゴンを視点から遠い順にソートし，等値面の透明度を考慮して，フレームバッファに色を重ねていく.

4. 高速アルゴリズム

4.1 基本的なポリゴンデータの作成法

図 5，**表 2** が示すように，前もって各四面体格子の節点 1〜4 と辺 1〜6 を対応づけるテーブルを定義しておく.

各四面体格子内の等値面ポリゴンのパターンは，4 節点上での物理量の等値面値に対する大小関係の組合せにより，16 通り（2 の 4 乗）に分類される．**表 3** の「格子—ポリゴン」テーブルは，各パターンにおける，ポリゴンの個数，各ポリゴンの頂点が乗る辺番号を示す．このテーブルで「節点値大小」とは，各節点での物理量が等値面値以上の場合「1」，等値面値未満の場合「0」とし，節点 1 から 4 に左から並べたものである．また各 id 番号は「節点値大小」を 2 進数値と解釈して，それを 10 進数に変換した値である.

アルゴリズムの実装においては，4 節点での値の大小をもとに id 番号を計算し，表 3 を参照してポリゴン頂点が乗る辺番号を求める．この一連の計算は全四

表 3 id 番号とポリゴン数とポリゴン頂点が乗る辺の両端の節点番号の対応
Table 3 Correspondence between ID number, the number of polygons, and the node number on the edge.

id	節点値大小	ポリゴン数	ポリゴン頂点
0	0000	0	
1	0001	1	(4,5,6)
2	0010	1	(2,3,6)
3	0011	2	(2,3,4)(2,4,5)
4	0100	1	(1,2,5)
5	0101	2	(1,2,6),(1,6,4)
6	0110	2	(1,3,6),(1,6,5)
7	0111	1	(1,3,4)
8	1000	1	(1,4,3)
9	1001	2	(1,5,3),(3,5,6)
10	1010	2	(1,4,2),(2,4,6)
11	1011	1	(1,5,2)
12	1100	2	(2,5,3),(3,5,4)
13	1101	1	(2,6,3)
14	1110	1	(4,6,5)
15	1111	0	

面体格子数をループ長とする DO ループの中でベクトル処理される．ポリゴン頂点の検索処理の中でポリゴン頂点が乗る辺番号は，その両端のローカルな節点番号対に変換され，最終的に辺両端のグローバルな節点番号対が計算される．またポリゴン頂点の座標値の計算は，次節で説明する共有ポリゴン頂点データの生成後に行われる．

本章のアルゴリズムは共有メモリ型並列計算機上や分散メモリ型並列計算機上で並列処理を行っても，プロセッサ間通信が必要ないのでプロセッサ台数分の並列化効果が得られる．

4.2 共有ポリゴン頂点データの生成

基本的なポリゴンデータを作成した後，ポリゴン頂点の検索処理を行う．高速版等値面描画手法ではメモリ節約，およびスムーズシェーディングにおけるポリゴン頂点上の法線ベクトルの計算のベクトル処理のため，次のようなデータ構造を用い，全ポリゴンの頂点データを隣接ポリゴン間で共有化して格納している．

- integer npolygon 全ポリゴン数
- integer nvertex 全ポリゴン頂点数（重複して数えない）
- integer ivertex(3,npolygon) ポリゴン頂点番号テーブル
- double precision vertex(3,nvertex) ポリゴン頂点座標値

ip 番目のポリゴンの各頂点（頂点 1〜3）の座標値は以下のように参照される．

```
iv1 = ivertex(1,ip) ! 頂点 1 の全体ポリゴン頂点番号
iv2 = ivertex(2,ip) ! 頂点 2 の全体ポリゴン頂点番号
iv3 = ivertex(3,ip) ! 頂点 3 の全体ポリゴン頂点番号
x1 = vertex(1,iv1) ! 頂点 1 の x 座標値
y1 = vertex(2,iv1) ! 頂点 1 の y 座標値
z1 = vertex(3,iv1) ! 頂点 1 の z 座標値
```

! 頂点 2 の座標値 (x2,y2,z2)，頂点 3 の座標値 (x3,y3,z3) も同様

ポリゴン頂点が乗る格子辺の両端にある節点番号を登録する配列として，節点対記録用配列 v_node_table(3*npolygon,2) を用意する．ポリゴン頂点番号テーブル ivertex の計算方法を以下の **Step 1** から **Step 3** に示す．**Step 1** から **Step 3** までの処理は四面体格子，格子中のポリゴン，ポリゴン頂点についての 3 重ループの中で行う．

Step 1 ポリゴン頂点が乗る格子辺の両端の節点番号 (node1,node2) を計算．

Step 2 節点番号 (node1,node2) が v_node_table に登録済みの場合は，ivertex に (node1,node2) を両端とする格子辺上の全体ポリゴン頂点番号を登録．

Step 3 節点番号 (node1,node2) が v_node_table に登録されていない場合は，全体ポリゴン頂点番号を更新し，その新規の全体ポリゴン頂点番号を ivertex に，節点番号 (node1,node2) を v_node_table に登録．

v_node_table から，ポリゴン頂点が乗る格子辺の両端の節点番号を取り出し，この格子辺上のポリゴン頂点の座標値 vertex を補間計算により求める．このポリゴン頂点の座標値の計算はポリゴン頂点数のループ長で処理される．

v_node_table の管理および重複チェックには，2 変数ハッシュ関数を使用したハッシュ法を用いる．2 変数を引数にとり，1〜HSIZE の間の整数値を返すハッシュ関数 h(x,y) を準備しておく．与えられた節点番号 (node1,node2) に対して，まずハッシュ値 h=h(node1,node2) を計算し，その値により節点番号を HSIZE 個の組に分類する．同一ハッシュ値を持つ登録済みの節点番号は連結リストとして格納する．もし与えられた節点番号が連結リストの末尾まで探索して見つからなかった場合は未登録と判断し，リストの末尾に追加登録する．本アルゴリズムの実装においては，v_node_table のほかに，連結リスト中において次セルの位置を格納するポインタ配列 next が必要にな

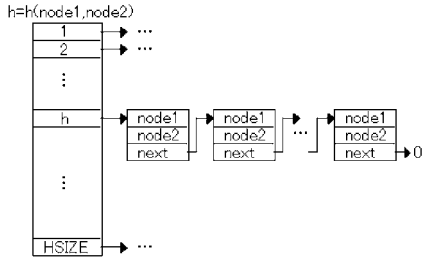

図 6 ポリゴン頂点を通る辺の両端の節点番号を利用してポリゴン位置を管理する方法
Fig. 6 Method of storing the locations of polygons using the node numbers of both the end points of those edges on which their vertices lie.

る．各連結リストの末尾はポインタ next が「0」を指すことにより終端される（**図 6**）．

ここでポリゴン頂点の検索処理の計算量が，ポリゴン数の線形オーダになる根拠を説明する．簡単のため，描画する等値面が 1 つの場合を考え，全ポリゴン数を $npolygon$ とすると，検索処理の対象となるポリゴン頂点の総数は $3*npolygon$ となる．一方，ハッシュ関数が理想的な場合，すなわちハッシュ長を m としたとき，ハッシュ関数の値域が 1 から m に均等に分布する場合を考えると，連結リストの長さの上限は $nvertex/m$ と見積もられる．検索処理の計算量が最悪となる場合は，毎回図 6 の連結リストの最後まで探索する場合でその計算量は $(3*npolygon)*(nvertex/m)$ と見積もられる．全ポリゴン頂点数 $nvertex$ とポリゴン数 $npolygon$ の格子数に対するオーダは同じと見積もられるので，計算量は $npolygon^2/m$ と見積もられる．m をポリゴン数 $npolygon$ とほぼ同じ値としたとき，計算量はポリゴン数 $npolygon$ のオーダになる．

本章のアルゴリズムを並列処理した場合について触れる．ポリゴン頂点の座標値計算部分は共有メモリ型並列計算機上や分散メモリ型並列計算機上でもプロセッサ間通信が必要ないのでプロセッサ台数分の並列化効果が得られる．節点対記録用配列，ポリゴン頂点番号テーブルの作成は各プロセッサ単位で並列に行うことができる．分散メモリ型並列計算機上で可視化処理をする場合のみ，ソルバが持っているプロセッサ間の境界上の節点番号情報から複数のプロセッサで共有されるポリゴン頂点番号テーブルを作成する必要があるが，この情報作成の処理コストは小さい．

4.3 スムーズシェーディング

ポリゴンの平滑化にはグーローシェーディング法を用いた．グーローシェーディング法では，頂点での法線ベクトルをもとに，頂点での輝度を計算する．最後にポリゴン内部の点での輝度は，そのポリゴンの 3 頂点での輝度を線形補間することにより求める．

高速版等値面描画手法では，4.2 節のポリゴン頂点番号テーブルを生成して，この配列を利用することにより，ポリゴン頂点上の法線ベクトル計算はポリゴン数と，全ポリゴン頂点数をループ長とする DO ループでベクトル処理される．ポリゴン頂点上の法線ベクトル計算手法を以下の **Step 1** から **Step 4** に示す．

Step 1 ポリゴン頂点上の法線ベクトル成分を格納する配列 vnormal(3,nvertex) を宣言し，ポリゴン番号 ip の値を 1 と初期化．

Step 2 ポリゴン頂点番号テーブル ivertex からポリゴン ip の 3 つの全体ポリゴン頂点番号を抽出．

Step 3 ポリゴン ip の法線ベクトル成分 (nx,ny,nz) を計算．

Step 4 ポリゴン ip の 3 つの頂点の法線ベクトル成分を表す vnormal にポリゴン ip の法線ベクトル成分 (nx,ny,nz) を加算．ip の値が全ポリゴン数 npolygon 未満なら，ip の値を更新して **Step 2** に戻り，ip の値が全ポリゴン数 npolygon に等しければ，vnormal の計算は終了．

この後，全体ポリゴン頂点に関するループ処理で vnormal を規格化する．

本アルゴリズムでは，ポリゴン頂点での法線ベクトルの計算は共有メモリ型並列計算機上ではループを分割すればよいので，プロセッサ台数分の並列化効果が得られる．分散メモリ型並列計算機上では前節で作成した ivertex 以外に複数のプロセッサで共有されるポリゴン頂点番号テーブルを使えば並列化は容易である．プロセッサ間通信はプロセッサ間の境界上のポリゴン頂点での法線ベクトルの計算の場合にのみ発生する．各プロセッサに割り当てられるポリゴン頂点数に比べ，プロセッサ間の境界上のポリゴン頂点数は十分小さく，プロセッサ間の通信量は小さいので，本節のアルゴリズムを分散メモリ型並列計算機に適用してもプロセッサ台数分に近い並列化効果は期待できる．

4.4 等値面描画手法の隠面処理および半透明化

視線上で重なっている各ポリゴンの色を，等値面ごとに指定された透明度に従って視点から遠い順に混ぜ合わせることにより，プロトタイプ版ならびに高速版等値面描画手法では等値面の半透明化を実現している．また隠面処理のアルゴリズムとして Z バッファ法を用いた．

表 4 プロトタイプ版と高速版等値面描画手法の処理の CPU 時間（sec）：4,851,495 格子
Table 4 Necessary CPU time (sec) for the prototype and improved isosurface functions (number of grids; 4,851,495).

描画条件	プロトタイプ版	高速版	速度向上率
フラット/不透明	3.06	0.968	3.16 倍
フラット/半透明	3.32	1.38	2.41 倍
スムーズ/不透明	9.50	1.02	8.87 倍
スムーズ/半透明	9.95	1.43	6.96 倍

表 5 プロトタイプ版と高速版等値面描画手法の処理の CPU 時間（sec）：38,811,960 格子
Table 5 Necessary CPU time (sec) for the prototype and improved isosurface functions (number of grids; 38,811,960).

描画条件	プロトタイプ版	高速版	速度向上率
フラット/不透明	13.3	4.92	2.70 倍
フラット/半透明	15.4	6.78	2.27 倍
スムーズ/不透明	112	5.21	21.5 倍
スムーズ/半透明	114	7.14	16.0 倍

表 6 プロトタイプ版と高速版等値面描画手法の処理の CPU 時間（sec）：310,495,680 格子
Table 6 Necessary CPU time (sec) for the prototype and improved isosurface functions (number of grids; 310,495,680).

描画条件	プロトタイプ版	高速版	速度向上率
フラット/不透明	69.8	35.1	1.99 倍
フラット/半透明	79.1	44.3	1.79 倍
スムーズ/不透明	1880	37.8	49.7 倍
スムーズ/半透明	1930	47.5	40.6 倍

5. 性能評価結果

本章では，プロトタイプ版と高速版等値面描画手法の SX-6 上での性能評価結果を示す．高速版の性能評価のテストデータは 3 章のプロトタイプ版の性能評価のテストデータと同じである．フラットシェーディング，スムーズシェーディング，半透明，不透明のケースで SX-6 上での描画 CPU 時間と，その格子規模依存性について性能評価を行った．

5.1 描画 CPU 時間

描画条件（フラット/不透明，フラット/半透明，スムーズ/不透明，スムーズ/半透明）の 4 ケースについて，プロトタイプ版と高速版の描画 CPU 時間を評価する．各表の速度向上率はプロトタイプ版の描画 CPU 時間を高速版の描画 CPU 時間で割ったものである．**表 4**，**表 5**，**表 6** から以下のことが確認された．

- スムーズ・不透明の場合，高速版等値面描画手法の処理の CPU 時間は 4 千万格子規模の場合は 5.21 秒，3 億格子規模の場合は 37.8 秒になった．
- 描画 CPU 時間は，プロトタイプ版の場合は格子数の 4/3 乗のオーダであったが，高速版の場合は格子数の増加よりも小さいオーダに改善された．
- 今回スムーズシェーディングの高速化にあたり，法線ベクトルの計算をベクトル処理するため，ポリゴン頂点番号テーブルを生成した．このテーブルによって，ポリゴン頂点の座標値の計算がポリゴン頂点数のループ長でベクトル処理でき，かつ各ポリゴン頂点上の法線ベクトルの計算もポリゴン数のループ長でベクトル処理できるため，フラットシェーディングの場合も高速版では 2 倍程度速度向上が見られた．

表 7 各処理部と CPU 時間の格子規模依存性
Table 7 Dependency of necessary CPU time (sec) for each visualization step on the number of elements.

処理内容	格子規模依存性
基本的なポリゴンデータの作成	N
ポリゴン頂点の検索処理	$N^{2/3}$
ポリゴンの法線ベクトルの計算	$N^{2/3}$
ポリゴン頂点上の法線ベクトルの計算	$N^{2/3}$
半透明化処理	$N^{2/3}\log(N^{2/3})$

5.2 高速版等値面描画手法の処理速度の格子規模依存性

前章で述べた高速版等値面描画手法の処理速度の格子規模依存性について考察する．表 4，表 5，表 6 から，描画の CPU 時間の格子規模依存性が格子数の増加よりも小さいことを確認した．これを計算量の観点から分析する．等値面の描画計算の各処理部とその CPU 時間の格子数に対するオーダを**表 7** に示す．ここで格子数を N で示す．

表 7 においては，等値面を構成するポリゴン数は N の 2/3 乗のオーダとした．また半透明化処理は等値面を構成するポリゴンを視点から遠い順にソートする必要があり，その処理にヒープソートを用いているので，$N^{2/3}\log(N^{2/3})$ のオーダとした．基本的なポリゴンデータの作成は格子数のループ長でベクトル処理，ポリゴン頂点上の法線ベクトルの計算はポリゴン数と全ポリゴン頂点数のループ長でベクトル処理される．全ポリゴン頂点数は N の 2/3 乗のオーダに見積もられ，格子数が 4,851,495 格子の場合でもループ長はベクトル化の効果を引き出すのに十分な長さとなる．このため，基本的なポリゴンデータの作成の CPU 時間は N の線形オーダ，ポリゴンの法線ベクトルの計算とポリゴン頂点上の法線ベクトルの計算の CPU 時間は N の 2/3 乗のオーダに比例した．各処理の格子規模依存性から，等値面の描画の CPU 時間の格子規模依存性は N の 2/3 乗のオーダと線形オーダの間と

表 8 プロトタイプ版と高速版等値面描画手法のポリゴン頂点のスムーズシェーディング処理までの CPU 時間
Table 8 Necessary CPU time (sec) for the smooth shading process in the prototype and improved isosurface functions.

格子数	プロトタイプ版	高速版	速度向上率
4,851,495	8.103	0.533	15.2 倍
38,811,960	108.8	3.731	29.2 倍
310,495,680	1862	28.1	66.26 倍

推定され，理論と実際の評価結果がほぼ一致した．

5.3 スムーズシェーディング処理部分の分析

スムーズシェーディングの場合，陰面処理および半透明化処理以外の部分を性能分析する．**表 8** からプロトタイプ版と高速版を比較し，以下のことが確認された．

- プロトタイプ版において，負荷が 85%以上であったスムーズシェーディング処理までの計算部分の高速化により，等値面の描画の CPU 時間が短縮された．
- スムーズシェーディング処理までの CPU 時間の N 依存性が 2/3 乗のオーダ以上線形オーダ未満の範囲に改善された．

6. おわりに

本研究では数億規模以上の大規模非構造格子上での等値面描画手法の高速アルゴリズムの概要とその性能評価結果について述べた．

高速化においては，処理負荷が格子規模の増大とともに大きくなるスムーズシェーディング処理部分の高速化に重点をおいた．スムーズシェーディングに必要なポリゴン頂点の検索処理をハッシュ法により高速処理し，ポリゴン頂点上の法線ベクトルの計算をベクトル処理向きにするために，ポリゴン頂点の検索処理の過程でポリゴン頂点番号テーブルを生成した．このポリゴン頂点番号テーブルを使うことにより，ポリゴン頂点上の法線ベクトルの計算はポリゴン数，全ポリゴン頂点数をループ長とするベクトル処理が可能になった．大規模問題に対しては必要なメモリ容量の低減が重要であるため，その容量を格子数の 2/3 乗のオーダに抑えた．開発した高速アルゴリズムを数千万から数億規模の格子の問題に適用し，SX-6 上で描画速度の性能評価を行った．評価の結果，スムーズシェーディング処理を行った場合，描画 CPU 時間が 310,405,680 格子で 47.5 秒になり，描画 CPU 時間の格子規模依存性は，格子数の 2/3 乗のオーダ以上線形オーダ未満になった．描画 CPU 時間の格子規模依存性は問題がより大規模化するにつれ，今回のアルゴリズムの効果が出ることを示す．今回はベクトル化による高速化を行ったが，開発した高速アルゴリズムは並列処理にも向いている．今後は六面体格子用等値面描画手法の高速化や今回開発した高速アルゴリズムの並列化とその性能評価を実施し，従来の可視化ソフトでは可視化が困難な大規模問題に対し，効率的可視化の手段を提供していく予定である．

参考文献

1) 武井，松本，土肥：大規模非定常数値シミュレーションのためのリアルタイム可視化—並列計算サーバによる可視化方式の実用化に向けて，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.41, No.SIG 8(HPS 2), pp.107–118 (2000).
2) Cignoni, P., et al.: Multiresolution Representation and Visualization of Volume Data, *IEEE Trans. Visualization and Computer Graphics*, Vol.3. No.4, pp.352–369 (1997).
3) Lane, D.A.: Scientific Visualization of Large Scale Unsteady Fluid Flows, *Scientific Visualization Overviews Methodologies Techniques*, Nielson, G.M., Hagen, H. and Muller, H. (Eds.), pp.125–145, IEEE Computer Soc. Press (1997).
4) Van Sint Jan., et al.: Morphology-based data elimination from medical image data, *IEEE Computer Graphics and Applications*, pp.46–52 (March-April 2000).
5) http://www.llnl.gov/terascale-vis/
6) Itoh, T., et al.: Fast Isosurface Generation Using the Volume Thinning Algorithm, *IEEE Trans. Visualization and Computer Graphics*, Vol.7, No.4, pp.32–46 (2001).
7) 伊藤，山口，小山田：等値面生成のための高速ポリゴン構築方法，情報処理学会論文誌，Vol.42, No.5, pp.1076–1083 (2001).
8) Livat, Y., et al.: A Near Optimal Isosurface Extraction Algorithm Using the Span Space, *IEEE Trans. Visualization and Computer Graphics*, Vol.2. No.1, pp.73–84 (1996).
9) Koyamada, K.: Visualization of simulated airflow in a clean room, *IEEE Visualization '92*, pp.156–163 (1992).
10) Doi, A. and Koide, A.: An Efficient Method of Trigulating Equi-valued Surfaces by Using Tetrahedral Cells, *IEICE Trans.*, Vol.E74, No.1, pp.214–224 (1991).

(平成 15 年 7 月 24 日受付)
(平成 15 年 12 月 9 日採録)

杉原 光太

昭和 42 年生．平成 5 年京都大学大学院理学研究科数理解析専攻修士課程修了．同年 NEC 入社．現在 NEC HPC 販売推進本部勤務．