ハイパフォーマンス
コンピューティング 48−16
（１９９３．８．２０）

3次元数値シミュレーション結果のための「計測」システム

小山田 耕二
伊藤 貴之
日本アイビーエム 東京基礎研究所

本報告において、3次元数値シミュレーション結果のための「計測」システムについて述べる。本システムにおいて、ユーザは、3次元空間中の任意点に計測器を配置をして、物理量を計測する感覚で可視化処理を行なうことができる。数値シミュレーション結果では、データが格子単位に定義されているのでユーザの指示する任意点を含む格子を効率良く探索する必要がある。このため、我々は、数値シミュレーションのもつ特徴を利用した効率の良い探索手法を開発した。また、格子間の隣接情報を参照して、プローブ点を通る流線や等高面を生成する方法を開発した。

A Measurement System for 3-D Numerical Simulation Results

Koji Koyamada
Takayuki Ito
Tokyo Research Laboratory, IBM Japan Ltd.

In this paper, we propose a measurement system for three dimensional numerical simulation results. In the proposed system, an end user can explore the simulation results as if he locates a measurement device at an arbitrary position in a real laboratory to measure a physical quantity. In order to realize it, a grid cell which includes the user-specified position should be efficiently searched for because numerical simulation results are defined on grid cells. For such searching, we developed a method on the basis of a feature in the simulation results. In addition, we developed methods to generate a stream line and an isosurface which pass through the specified position.

はじめに

1987年、Visualization in Scientific Computingの必要性が提唱されて[McCormick 87]以来、3次元数値シミュレーション結果についての可視化技術が多くの研究者によって開発され、商用ソフトウエアパッケージも流通し始めている。とりわけ、ビジュアルプログラミングを応用した可視化システムが主流になっている。これらパッケージにおいては、可視化技術のとりそろえに焦点が当てられユーザインタフェースはもっぱら、どの可視化技術を使うのかということに重点がおかれている。ビジュアルプログラミングタイプの可視化システムでは、モジュールと呼ばれるデータ可視化機能をディスプレー上で組み合わせることによりユーザが必要とする可視化システムを構築することができる。開発元は、モジュールの組合せ作業のしやすさ、モジュールの作りやすさ、そしてデータの取り込みやすさに焦点をあてた改良を行なっており、高級言語とグラフィックスライブラリをつかって可視化ソフトウエアを作成していたプログラマは、膨大なプログラミング作業から解放されるようになった。しかしながら、このビジュアルプログラミング型の可視化システムは、必ずしも物理現象を数値シミュレーションを使って解明しようとするエンドユーザの立場にたったものではない。これに対して、我々は、「エンドユーザにとって、使いやすい可視化システムとは何か?」という視点に立ってシステム開発を行なってきている。エンドユーザーの多くは、物理現象を解明するため、関連する物理量を計測したという経験を持っている。このような計測においては、どのように計測するのかと同様、どこで計測するのかも重要である。3次元数値シミュレーション結果を、数学モデルを使って再現された物理現象と仮定するならば、「可視化」は、「計測」に対応するものと考えることができる。したがって、彼らは、ごく自然に、3次元空間中の任意点に計測器を配置して物理量を計測する感覚で、可視化処理を行ないたいと考えるであろう。残念ながら、従来の可視化システムは、エンドユーザが連続的に指示する3次元位置座標をパラメータとした可視化が行なえるようなユーザインタフェースを提供していなかった。このような背景をもとに、われわれは、3次元空間におけるデータ計測に類似した作業環境を提供する可視化システムの開発を行なうことにした。本報告では、まず、システム開発にあたっての目標、そして開発されたシステムの概要について述べる。次に、3次元位置座標を含む格子を高速に検索し、データ値を補間計算するデータプローブ法を提案する。最後に、与えられた座標点を通る流線や等高面を生成するためのアルゴリズムについて述べる。

システム開発目標

これまでに述べた従来の可視化システムの現状および問題点を考慮に入れて、システム開発に当たり以下の目標を設定した。

1. **エンジニアリングワークステーションに標準装備されている周辺装置を用いて3次元空間中の1点が指示できる。**
   現在、3次元空間中の1点(以下、プローブ点)を指示するためにさまざまな装置が開発されており、いわゆる「バーチャルリアリティー」実現のために利用されている。これら装置は、まだまだ高価であり、可視化システムのエンドユーザが手軽に購入できる状態には至っていない。したがって、エンドユーザのエンジニアリングワークステーションに標準装備されているマウスを使った空間点指示を目標としている。動きが2次元平面に限られるマウスを使ってプローブ点を指示するために、以下の3つのモードを設定する。

   - キーボードを使って座標値を直接入力(点モード)
   - マウスの動きをエンドユーザの指定する直線上に限定(線モード)
   - マウスの動きをエンドユーザの指定する平面上に限定(面モード)

2. **対話的に指示された点におけるデータ値（スカラ、ベクタ）を表示する。**
格子点で出力されている数値シミュレーション結果をもとにして任意点におけるデータ値を補間計算（データプローブ）する。この補間計算は、以下のステップで行なわれる。

   - マウスにより指示されたプローブ点を含む格子を探索する。
   - 格子の各頂点でのデータ値を使って補間計算を行なう。

3. **3次元座標をパラメータとした可視化処理（断面／等高面／流線表示）を行なう。**
プローブ点を含む断面／等高面、プローブ点を出発する流線を生成し、マウスの動きに合わせて生成形状を移動表示する。

4. **数値シミュレーションプログラム（ソルバー）と協調しながら可視化処理を行なう。**
数値シミュレーションにおける陽的解法の途中段階での収束状況をモニタしたり、非定常数値シミュレーションにおいて、タイムステップごとにリアルタイムに可視化処理を行なう。

システム概要

本システムにおいて処理の対象となる数値シミュレーション結果としては、有限要素法、有限差分法が出力したものであると仮定する。これらシミュレーション手法は、3次元空間を格子単位に分割することを特徴とする。したがって、本システムの前提となるデータ構造は、格子点情報、格子情報から構成される。格子点情報は、格子点番号により管理されるデータであり、格子点座標、格子点で定義されるシミュレーション結果（スカラ／ベクタ）からなる。格子情報は、格子番号により管理されるデータであり、格子を構成する格子点番号および格子を構成する各面を通じて隣接する格子番号からなる。有限差分法解析結果には、規則性があるので、格子情報は、陽には存在しない。
本システムでは、システムの状態を初期状態、イベント監視状態、イベント処理状態群として定義することができる。初期状態においては、システム実行に必要なシステム資源の確保やプローブ点座標の初期値設定を行なう。イベント監視状態において、次に述べるイベントを確認したあとで、イベント処理状態に遷移する。

- マウスの位置座標の変化
- マウスボタンの押下
- キーボードからの入力
- ソルバーからのデータ処理要求

このなかで、マウスの位置座標の変化について少し述べる。我々が、グラフィックスアプリケーションプログラムインタフェースとして利用している graPHIGS[1] を使うと、マウスの位置が一定時間毎に3次元座標として返される。この場合の座標は、視点から投影方向に向かって発せられる直線とビュー定義ボリュームの前面との交点を表す。この交点は、必ずしも格子集合体の内部に存在するとは限らない。計測作業と類似したシステムとするには、プローブ点が常に格子集合体の内部にある必要があるので、本システムは、この点をプローブ点としてそのまま使うのではなく、前回獲得した交点との差分をプローブ点の移動ベクタとして利用している。この移動ベクタの大きさがある閾値以上の時、位置座標が変化したと考えている。
イベント処理状態においては、以下に述べる処理を行なった後、再びイベント監視状態に戻る。

- マウスの位置座標が変化すると、その座標を含む格子を探索する。

[1] graPHIGS は、IBM の登録である。

- マウスボタンを押すことにより、可視化ツール（断面表示／等高面表示／流線表示）を起動する。
- コマンドがキーボード入力されると、ツールに対するパラメータ設定を行なう。
- ソルバーからの最新データが到着すると、処理対象のデータを更新する。

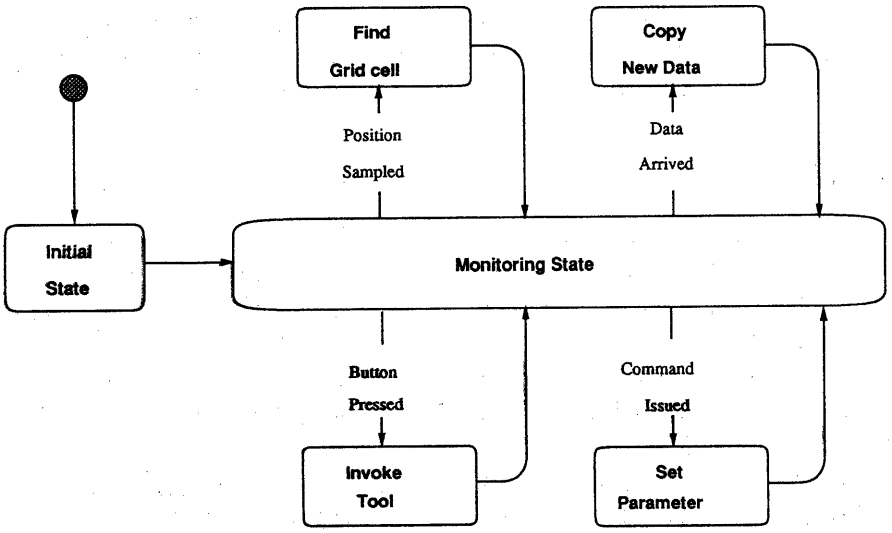

図 1: システムの概要

対話的データプローブ

格子探索

データプローブ処理において、まず最初に行なわなければならないことは指示された点（プローブ点）を含む格子を探索することである。このような格子探索問題に対し、Preparata らは、$O(log^2 n)$ の計算時間で処理する手法を開発している [Preparata 89] が、格子数が大きくなった時の対話性の確保が困難である。Neeman は、この問題に対し、メディアンカット法を適用している [Neeman 90]。格子の稜線が座標軸と平行するようなデータについては、計算効率が高いものの、そうでないデータでは計算効率が低くなってしまう特徴がある。これらの手法は、それぞれの格子を独立して扱っているため、数値シミュレーション結果のもつ有用な特徴を利用していない。数値シミュレーション結果の定義される空間は、重なることなくそして隙間を作ることなく、格子によって埋めつくされており、また隣接する格子は、格子点を共有している。われわれは、この特徴を利用して効率の良い格子探索法を開発している。我々の手法では、まず、1つの格子を「適当に」（メモリ内に最初に格納されているもの／前回の探索で見つかったもの）決めて、その内部点（探索開始点）からプローブ点に向かって半直線を引く。半直線は、探索開始点において t=0、そしてプローブ点において t=1 となるように、パラメータ表現するものとする。探索開始点から半直線に沿っ

て順次、格子を構成する面（格子面）との交点でのパラメータtの値を計算していく。格子面は、一般的には、平面であるとは限らない。曲面と直線との交点計算は、時間がかかるので、本システムでは、格子面の三角形分割により交点計算を高速化している。格子を順次探索するうえで、交点におけるtの値が始めて1を越えた時、その格子は、与えられたプローブ点を含むものと考える。以上が基本的な考え方であるが、定義領域に凹部がある場合、tの値が1を越える前に領域の境界に到達することがあり得る。この場合、探索開始点を変更して、再探索を行なう。本システムでは、再探索の回数を減らすために、以下の条件を満足する格子に含まれる探索開始点から再探索を行なっている。

- 格子面が領域の境界を構成する。
- ある平面を考えた時、プローブ点のこの平面への投影点が格子の投影部分に含まれる。

データ値補間

格子内部においてデータ値Sは、格子点で定義されているデータ値S_iおよび補間関数$N_i(\vec{u})$を用いて、計算される。

$$S(\vec{X}) = \sum_{i=0}^{n-1} S_i \times N_i(\vec{u}) \tag{1}$$

ここで、iは、格子で定義される格子点番号(格子点数n)、$\vec{u}$は、データ値の計算される格子内部点の局所座標(u,v,w)を表す。
局所座標は、一般に曲面で構成される格子が単位長さの稜線を持つ正方格子に変換される正規直交座標系である。4面体格子の場合、局所座標として、体積座標が用いられる。体積座標は、全4面体体積に対する、内部点と各3角形とから作る部分4面体体積の比として、定義される。体積の計算は、もとの4面体を正4面体に変換した後に行なわれる。補間関数の値は、その関数の定義されている格子点では、1.0をとり、それ以外の格子点では、0.0をとる。すなわち、

$$N_i(\vec{u_j}) = \begin{cases} 1.0, & \text{for } i = j \\ 0.0, & \text{for } i \neq j \end{cases}$$

したがって、$N_i(\vec{u})$は、格子点数nに等しい項数をもつ多項式により、表現される。例えば、8格子点を持つ格子では、

$$N_i(\vec{u}) = a_0 + a_1 \times u + a_2 \times v + a_3 \times w + a_4 \times vw + a_5 \times wu + a_6 \times uv + a_7 \times uvw$$

と表される。8個の格子点で補間関数を評価すると連立方程式が得られる。8個の係数$(a_0 \ldots a_7)$は、この連立方程式を解くことにより得られる。
補間関数は、局所座標の関数となっているのに対し、格子内部の評価点は、全体座標系で与えられることが多い。したがって、補間計算を行なうためには、まず座標変換を行なう必要がある。この座標変換は、一般にニュートン法に基づく繰り返し計算を用いておこなう。以下に、格子内部の評価点における補間計算アルゴリズムを示す。

1. 与えられた評価点の座標$\vec{X}$を各格子点の座標$\vec{X_k}$および補間関数$N_k(\vec{u})$を用いて表現する。

$$\vec{X} = \sum_{k=0}^{n-1} \vec{X_k} \times N_k(\vec{u})$$

2. 求めるべき局所座標の初期値を$\vec{u_0}$そして繰り返し回数を表すインデックスiを0に初期化する。

3. 局所座標 u_i に対する全体座標 $\vec{X_i}$ を計算する。

$$\vec{X_i} = \sum_{k=0}^{n-1} \vec{X_k} \times N_k(\vec{u_i})$$

4. i 番目に計算された全体座標と与えられた評価点座標の差 $\Delta\vec{X_i} = \vec{X} - \vec{X_i}$ をヤコビアン行列 [J] と微小変位ベクタ $\Delta\vec{u_i}$ との積で表す。i 番目に計算された局所座標をわずかに変化させると与えられた評価点座標に一致するとして考える。

$$\Delta\vec{X_i} = [J]^t \times \Delta\vec{u_i}$$

但し、

$$[J] = \begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \\ \frac{\partial x}{\partial w} & \frac{\partial y}{\partial w} & \frac{\partial z}{\partial w} \end{pmatrix}$$

であり、$[J]^t$ は、[J] の転置行列を表す。行列の各成分は、式 () に基づいて計算される。例えば、(1,1) 成分は、

$$\frac{\partial x}{\partial u} = \frac{\partial}{\partial u} \sum_{k=0}^{n-1} x_k \times N_k(\vec{u}) = \sum_{k=0}^{n-1} x_k \times \frac{\partial N_k(\vec{u})}{\partial u}$$

となる。

5. 前ステップで得られた方程式を微小変位ベクタ $\Delta\vec{u_i}$ について解く。

$$\Delta\vec{u_i} = [J]^{-t} \times \Delta\vec{X_i}$$

6. i 番目に計算された局所座標に求められた微小変位ベクタを加え、新たな局所座標とする。

$$\vec{u_{i+1}} = \vec{u_i} + \Delta\vec{u_i}$$

7. 微小変位ベクタが十分に小さいとき、$\vec{u_{i+1}}$ を局所座標の最終結果とし、これを式 (1) に代入して、補間計算を完了する。そうでない時は、インデックス i をインクリメントし、ステップ 3. に戻る。

プローブ点からの可視化

流線追跡

流線は、以下の方程式を用いて定義される。

$$\frac{dx}{v_x} = \frac{dy}{v_y} = \frac{dz}{v_z}$$

ここで、$\vec{v} = (v_x, v_y, v_z)$ は、3 次元空間内における速度ベクタを表す。数値シミュレーション結果から流線を計算する時、格子単位に速度ベクタを積分する。1 つの格子で計算された出口点およびそこでの脱出速度ベクタは、次に接続する格子にとっての入口点およびそこでの進入速度ベクタとなる。あるプローブ点で流線表示ツールが起動されると、ツールには、プローブ点を含む格子の番号およびプローブ点で補間計算された速度ベクタが受け渡される。その格子を出発点として、格子間の隣接関係を参照しながら、つぎつぎと格子単位で脱出点が計算される。この計算は、流線が境界面に達するかまたは、流線上で速度ベクタがゼロになるまで続けられる。

1 つの格子において、進入点から脱出点を計算するアルゴリズムを以下に示す。

1. 格子内において、速度が一様と考えて、脱出点$\vec{p}_{out}^{old}$ および所要時間Δt を求める。

2. Δt を半分にし、繰り返しのためのインデックス値 i を初期化する。

$$\begin{aligned} \Delta t* &= 0.5; \\ i &= 0; \end{aligned}$$

3. i 番目の点$\vec{p}_i$において、速度ベクタ$\vec{v}_i$を補間計算する。

4. 速度が一様と考えて、現在点$\vec{p}_i$を出発点として、脱出点$\vec{p_{out}}$ および所要時間 t_iを求める。

5. t_iがΔt よりも大きいとき、現在点をタイムステップ（Δt）分進め、インデックス値 i をインクリメントして、ステップ 3 に戻る。

$$\begin{aligned} \vec{v} &= \vec{v}_i; \\ \vec{p_{i+1}} &= \vec{p}_i + \Delta t \times \vec{v}; \\ i+ &= 1; \end{aligned}$$

6. 前回の脱出点$\vec{p}_{out}^{old}$および今回の脱出点 $\vec{p}_{out}$ が十分に近いといえない時、$\vec{p}_{out}^{old}$を更新して、ステップ 2 に戻る。

$$\vec{p}_{out}^{old} = \vec{p}_{out};$$

計算精度を重視する時、ステップ 5 は、以下のように変更する。

$$\begin{aligned} \vec{v} &= \vec{v}_i; \\ \vec{p}_{i+1}^{*} &= \vec{p}_i + \Delta t \times \vec{v}; \\ \vec{v} &= \frac{1}{2} \times (\vec{v} + \vec{v}_{i+1}^{*}) \\ \vec{p}_{i+1} &= \vec{p}_i + \Delta t \times \vec{v}; \\ i+ &= 1; \end{aligned}$$

ここで、$\vec{v}_{i+1}^{*}$は、点$\vec{p}_{i+1}^{*}$において、補間された速度ベクタである。

等高面抽出

等高面は、以下の方程式を満足する点の集合として定義される。

$$S(\vec{X}) - C = 0$$

ここで、$S(\vec{X})$ は、3 次元空間内の 1 点$\vec{X}$におけるスカラデータ値であり、そして C は、与えられたスカラ値である。数値シミュレーション結果において、等高面抽出は、格子単位に行なわれる。通常、格子単位に、各頂点が等高面と格子稜線との交差点であるような三角形を生成していく。格子の稜線が等高面と交差するかどうかを効率良く調べるために、格子点 i 毎にスカラデータ値 S_iと C との差 $g_i = S_i - C$を計算する。格子の稜線上でスカラデータが線形に分布していると仮定すると、稜線を構成する格子点で g_iの符号が変化した場合、その稜線は等高面と交差点をもつと考えることができる。n 個の格子点から構成されている格子では、このような符号の組み合わせは、有限（2^n通り）なので、符号の組に対して交差点をもつ稜線の組をあらかじめテーブルに記述している。等高面と交差点をもつ格子は、格子全体のうちの一部であるので、抽出処理を効率化するためには、交差点を持たない格子は、処理の対象にしないことが重要である。

このために、稜線の組の他に交差する面の組もテーブルに記述しておき、交差面を通じて接続する格子だけを処理の対象とするようにしている。交差稜線の組及び交差面の組は、格子点における符号の組から一意に決まるインデクス値にしたがって登録されている。本システムでは、プローブ点でのスカラデータ値Cおよびプローブ点を含む格子（シード格子）の番号が与えられるので、プローブ点を通る等高面は、以下の手順にしたがって抽出することができる。

1. FIFO キューにシード格子の番号を登録する。
2. FIFO キューから格子番号を取り出す。キューが空なら抽出処理を終える。
3. 格子点でのデータ値およびプローブ点でのスカラデータ値からインデクス値を計算し、交差稜線、交差面を調べる。
4. 交差稜線上で交差点が計算済みかどうかを調べる。計算済みでなければ、交差点を計算し、交差点番号を与える。計算済みであれば、交差点番号を獲得する。
5. 交差点番号の組として、三角形データを出力する。
6. 格子情報を参照して、交差面に接続する格子の番号を取り出す。
7. 取り出された番号の示す格子が登録済みでなければ、FIFO キューに登録する。
8. ステップ2に戻る。

交差稜線は、通常複数の格子によって共有されているので、交差点計算を効率良く行なうために、ステップ4は、重要である。ステップ4の処理を実現するため、本システムでは、計算済みの交差点をハッシュテーブルに登録している。

まとめ

我々は、可視化処理と計測作業のアナロジーに基づき、数値シミュレーション結果のための「計測」システムを開発した。我々は、対話的データプローブにおいて、プローブ点を含む格子を効率良く捜し出すために、数値シミュレーション結果のもつ有用な特徴を利用した格子探索手法を開発した。また、データプローブ処理中、プローブ点を含む格子（シード格子）の番号がわかることを利用して、シード格子から格子間の隣接関係を参照しながら、効率良く流線追跡や等高面抽出を行なう手法を開発した。今後、ソルバーとの協調処理に重点をおいた開発を進めていく予定である。

参考文献

[McCormick 87] McCormick, B. H., DeFanti, T. A. and Brown, M. D., 'Visualization in Scientific Computing,' Computer Graphics, Vol.21, No.6, 1987.

[Neeman 90] Neeman, H., 'A Decomposition Algorithm for Visualizing Irregular Grids,' Computer Graphics, Vol.24, No.5, pp.49-56.

[Preparata 89] Preparata, P. and Roberto T., 'Efficient Point Location in a Convex Spatial Cell Complex,' Technical Report 89-47, Dept. of Computer Science, Brown University, 1989.