

大規模有限要素解析における プレポスト処理に関する調査

河合浩志

平成 17 年 8 月 29 日

目次

第 1 章	要旨	4
第 2 章	はじめに	5
第 3 章	対話型システムの特徴	6
3.1	対話型操作の階層性	6
3.2	リアルタイム操作	6
3.3	コマンド操作	7
3.4	セッション開始終了操作	7
第 4 章	大規模解析のための計算機環境	8
4.1	計算サーバ	8
4.1.1	ハードウェアアーキテクチャ	9
4.1.2	OS と開発環境	9
4.1.3	使い勝手	10
4.1.4	性能	10
4.1.5	プレポスト処理	11
4.2	クライアント端末	11
4.2.1	ハードウェアアーキテクチャ	11
4.2.2	OS と開発環境	12
4.2.3	性能	13
4.3	高速化	13
4.3.1	スカラープロセッサでの高速化	13
4.3.2	ベクトルプロセッサでの高速化	15
4.3.3	並列化	15
4.3.4	描画処理の高速化	16
第 5 章	大規模有限要素解析	18
5.1	定常解析	18
5.2	非定常解析	19
5.3	構造解析	20
5.3.1	解析対象となる問題と使用される要素	20
5.3.2	プレポスト処理	20
5.4	流体解析	21
5.4.1	解析対象となる問題と使用される要素	21
5.4.2	プレポスト処理	21
第 6 章	既存アプローチのサーベイ	23
6.1	主にクライアント側で処理する場合	23
6.2	主にサーバ側で処理する場合	24

第 7 章	ブレバースト処理のための要素技術	26
7.1	ハッシュ検索	26
7.2	バケット検索	26
7.3	要素から節点への振り分け	26
7.4	低次要素化	26
7.5	要素辺の抽出	26
7.6	要素面の抽出	27
7.7	境界表面の抽出	28
7.8	境界表面のグループ化	28
7.9	任意断面の抽出	29
7.10	等値面の抽出	30
7.11	矢線の抽出	30
7.12	パーティクルトレースの抽出	31
第 8 章	データ抽出処理のパフォーマンス設計	33
8.1	性能見積りの前提条件	33
8.2	有限要素モデル	33
8.2.1	四面体一次要素	35
8.2.2	流体解析	35
8.2.3	四面体二次要素	36
8.2.4	構造解析	36
8.3	要素から節点への振り分け	37
8.3.1	四面体一次要素	38
8.3.2	流体解析	38
8.3.3	四面体二次要素	38
8.3.4	構造解析	39
8.4	低次要素化	39
8.4.1	構造解析	40
8.5	要素辺の抽出	41
8.5.1	構造解析	42
8.5.2	流体解析	42
8.6	要素面の抽出	43
8.6.1	構造解析	44
8.6.2	流体解析	44
8.7	境界表面の抽出	45
8.7.1	構造解析	47
8.7.2	流体解析	48
8.8	任意断面の抽出	48
8.8.1	構造解析	50
8.8.2	流体解析	51
8.9	等値面の抽出	51
8.9.1	構造解析	53
8.9.2	流体解析	53
8.10	矢線の抽出	54
8.10.1	流体解析	55
8.11	パーティクルトレースの抽出	55
8.11.1	流体解析	57
第 9 章	描画処理のパフォーマンス設計	58
9.1	性能見積りの前提条件	58
9.2	描画プリミティブ	59
9.3	アニメーション	60
9.4	境界表面の描画	61
9.4.1	構造解析	61

9.5 境界表面のアニメーション	62
9.5.1 構造解析	63
第 10 章 実行例と性能評価	64
10.1 問題設定	64
10.2 境界表面の抽出	64
10.3 境界表面のグループ化	64
10.4 任意断面の抽出	64
10.5 描画	64
第 11 章 おわりに	65

第1章 要旨

大規模な有限要素解析向けのプレポスト処理に関する調査報告。構造解析、熱伝導解析、電磁場解析および流体解析について、これらの定常問題および非定常問題を対象とした。主に、ボリウム（ソリッド）四面体要素を検討した。

ここでのプレポスト処理では、メッシュ生成については考慮していない。メッシュ生成後の解析条件設定と解析結果の可視化がその主な検討対象である。具体的な機能については、以下の通り。

- メッシュ図
- 表面上の要素または節点ごとの選択と境界条件設定
- 表面上の面グループごとの選択と境界条件設定
- 表面上のコンター図
- 変形図
- 任意断面上のコンター図
- 任意断面上の矢線図
- 等値面図
- パーティクルトレース図

大規模解析を考慮した省メモリかつ高効率な設計について検討した。構造解析については、四面体二次要素を用いた一億自由度の非定常解析が扱えることが判明した。流体解析については、四面体一次要素を用いた一千万自由度の非定常解析が扱えることが判明した。どちらについても、32 ビット OS を搭載した PC 一台を用いたものである。

第2章 はじめに

地球シミュレータを始めとする超並列計算機 (MPP) や大規模 PC クラスタ、あるいは各種の計算グリッド環境など、ハイパフォーマンスコンピューティング環境を用いた大規模な有限要素解析において、そのプレポスト処理が困難になりつつある。

ここでは、数千万から数億自由度クラスの大規模有限要素解析に適用可能なプレポスト処理システムを開発することを念頭におき、まずは、その要求仕様や設計を検討するための下地として、プレポスト処理技術に関する調査検討を行う。

最初に、プレポスト処理の要件を整理するにあたり、一般的な対話型システムの性質について述べる。続いて、解析およびそのプレポスト処理に関連する計算機環境、および個々の解析問題の性質について分析する。

次に、大規模有限要素解析をサポートするプレポスト処理について、各種の要素技術を検討する。これらは、大きく分けてデータ処理と描画処理に分類される。同時に、メモリや HDD などの各種計算機資源への負荷や処理速度について分析する。

第3章 対話型システムの特性

対話型システムでは、各処理のレスポンスタイムは処理の目的や性質に応じて適切に設定されている。もし、作業セッション開始やファイル転送などのすべての処理をリアルタイムで行わなければならないとしたら、比較的小規模な解析モデルしか扱えないが、一方で、視点変更やオブジェクト選択操作に数分あるいは数時間もかかるようでは、対話型システムとして成立しない。

ということで、一口に対話処理とは言っても、実際にはこれをいくつかのパターンに分類することができる。便宜上これを3つに分類するとすれば、リアルタイム操作、コマンド操作および、セッション開始終了操作となる。リアルタイム操作は理想的には1 / 10 秒以内で、最悪でも数秒まで、コマンド操作は理想的には1 秒以内、最悪でも数分まで、そして、セッション開始および終了操作は理想的には1 分以内、最悪でも数時間まで、といったレスポンスタイムが期待される。

3.1 対話型操作の階層性

対話型システムにおける操作を、ここではセッションとコマンド、および入力操作により分類する。

セッションはユーザーからみて比較的最長続く一連のコマンド操作の過程である。一つのセッションはコマンドにより開始され、同様にコマンドによって終了するものとする。あるいは、アプリケーションが起動してから終了するまでを一つのセッションと見なすこともできる。一つのセッションを開始または終了させる際に行われる操作が、セッション開始終了操作に対応する。あるセッションの中に、複数のサブセッションを含むことができる。

コマンドは一つのまとまったタスクを遂行するために比較的最短時間で実行される操作である。一つのコマンド実行がコマンド操作に対応する。各コマンドはそれぞれ単一の目的のために存在し、各コマンドにはそのタスクを表現する名前が付いていることが多い。アプリケーションのマニュアルは、通常これらのコマンドについて説明している。あるコマンドを実行中に、複数のサブコマンドを実行することができる。

なお、一つのコマンドの実行にはマウスやキーボードなどの入力デバイスによる一回または複数回の入力操作が必要とされる。これらの入力操作がリアルタイム操作に対応する。

3.2 リアルタイム操作

GUI ベースのアプリケーションにおいてマウスやキーボードにより画面を更新する操作は、通常一秒以内のレスポンスが期待され、理想的には1 / 10 秒以内で行えることが望ましい。以下では、これをリアルタイム操作と呼び、他の対話型操作と区別することにする。

描画対象が三次元モデルの場合には、マウスやキーボード操作によるスムーズな視点移動や画面の拡大縮小が期待される。

また、描画対象が時間依存モデルであれば、時間方向変化をアニメーション表示する際には短い時間間隔で画面を更新できなければならない。さらに、そのアニメーション

ンの時間進行をマウスやキーボード操作によりコントロールする機能があればそのほうが望ましい。

一方、モデルをただ表示するだけでなく、モデルの各部を選択／編集することが必要とされる場合、まず描画されているモデルの一部をマウスによりピックし、そこで選択されたオブジェクトがハイライト表示される。こういったピック・選択操作もまた、リアルタイム操作の一つである。

3.3 コマンド操作

解析モデル全体あるいはその選択された一部に対して、より大がかりな修正変更や比較的重いデータ評価計算を行いたい場合がある。このような操作は通常、マウス操作によりメニューよりコマンドを発行するか、あるいは、アプリケーションのコマンド入力端末ウィンドウからキーボード操作によりコマンドを入力する。このとき、ダイアログボックスやコマンドライン引数により付加的なパラメータを追加入力することが多い。

こういったある程度入力の手間がかかるような対話操作の場合、これに必要とされるレスポンスタイムは通常数秒程度以内、理想的には一秒以内が期待される。

ただし、扱うデータ量が多いためそれ以上の処理時間がかかることもありうる。こういった場合にも最悪数分以内にコントロールが戻るようであればならない。また、砂時計やプログレスバー表示により作業の処理状況が分かるようにしたり、任意のタイミングで処理を中断することができることが望ましい。

3.4 セッション開始終了操作

対話型システムの操作では通常、ユーザーの作業をいくつかのセッションに分割して管理することが多い。例えば、ある一つの解析作業について、これをソルバーを起動する前までのプレ処理と、ソルバーの計算が終了した後のポスト処理という、二つのユーザーセッションに分割することは一般的である。

一つのユーザーセッションは、通常は対話型アプリケーションの起動と終了や、モデルファイルのロードやセーブ操作と連係している。そして、対話型アプリケーションを起動したり終了したりするごとに、あるいはその中で操作対象モデルを切替えるごとに、ファイルの読み書きやネットワーク通信を含んだより長時間の待ちが生まれることがある。

このような一つのユーザーセッションの開始時や終了時に行われる処理では、そのレスポンスタイムとしては通常数十秒から数分以内が期待される。ただし、状況によってはあるいはランチタイム中に、とか、明日の朝までに、という形で、最悪数時間ということも許されるようである。

第4章 大規模解析のための計算機環境

大規模シミュレーションのための計算機環境は、主に、メイン部分、つまり解析本体を実行するための計算サーバと、そのプレポスト処理を行うためのクライアント端末より構成される。以下の議論では、これらの計算機システムをそれぞれバックエンドおよびフロントエンドと表現する。つまり、フロントエンド側にはプレポスト処理における対話操作のためのクライアント端末が、また、バックエンド側ではメインの大規模解析を行うための計算サーバが、それぞれ存在するものとする。

フロントエンドとバックエンドをそれぞれどのような計算機環境上で行うか、また、これらをどのように繋ぐかについては、さまざまな構成が考えられる。そして、それらの構成の良否を検討する際には、その性能や使い勝手に関する多くの工学的トレードオフが存在する。

なお、ここで検討する問題の定義上、メインの解析そのものは大規模なため計算サーバでしか実行することができず、クライアント端末上では解析を実行できないものとする。したがって、解析のための入力ファイルについては、それがどこで生成されようとも、解析が始まる直前にはそれは計算サーバ上に置かれている必要があり、同様に、解析の出力ファイルはまず計算サーバ上に置かれることになる。プレポスト処理において対話操作を行うクライアント端末からは、これらのファイルに関して計算サーバとの間のデータ転送が必要となる。

理想的には、端末を有する高速計算機を用い、メインとプレポストをすべてそこで行ってしまえばよいことになるが、これを実現することは、主に性能およびコストの点から難しい。具体的には、SGIのOnyxなどのようにグラフィックス機能を有するSMP型の計算サーバを用いるか、あるいは、PCクラスタの一部あるいはすべてにグラフィックスカードを装着するようなケースがありうる。このようなシステム構成は、プレポスト処理環境単体として見れば優れた環境であるが、メインの計算能力と言う点で不十分である。一方で、その計算能力を無理矢理補おうとすれば、今度はメインの計算に関して利用効率が低くなってしまったため、ユーザーあたりのコストがきわめて高くなってしまったといった欠点を有する。

ということで、以下の議論ではその前提として、クライアント端末と計算サーバがそれぞれ独立した計算機システム、すなわちフロントエンドシステムおよびバックエンドシステムとして分離されており、その間をネットワークあるいはその他のデバイスを用いてデータのやりとりを行うことを想定する。

基本的にはこの問題は、プレポスト処理自体をどのように分割し、そのうちのどこをクライアント端末で、また、どこを計算サーバで実行するのか、を検討することである。すなわち、対話操作を管理するためのフロントエンド上のソフトウェアモジュールと、データ処理を高速に行うためのバックエンド上のソフトウェアモジュールの設計問題となる。

4.1 計算サーバ

計算サーバは計算能力に優れ、また、メモリ、HDD容量とも巨大である。ただし、それが高性能であればあるほど、高価なものとなるため、多くのユーザーにより共有

されることになる。共有されるということは、その利用にはなんらかの制限が科せられることが多い。これは、主に、計算能力に関する制約と、計算時間に関する制約とに分かれる。特に、バッチ型の計算実行が要求されるならば、その対話性はユーザーにとって最悪のものとなる。また、計算サーバは計算機ルームや計算機センターといったユーザーから離れた場所に配置される。ユーザーはこれを使うためにわざわざそこまで通うか、あるいは、近くの端末からネットワークを介して計算サーバにアクセスする。

4.1.1 ハードウェアアーキテクチャ

現状の計算サーバについては、そのほとんどが並列型の計算機システムである。それらのうち特に大規模なものは、複数のノードにより構成される分散メモリ型の並列計算機となっている。また、各ノードがデュアルあるいはそれ以上の共有メモリ型のマルチプロセッサ構成になっていることも多い。

計算サーバ上での性能を左右する要因としては、並列化のほかに、プロセッサの種類がある。これを分類すると、スカラー型とベクトル型とに分かれる。現在では PC クラスタおよび超並列計算機 (MPP) のほとんどでスカラー型のプロセッサが利用されているが、日立の SR8000、NEC の SX シリーズや地球シミュレータなど一部の MPP ではベクトル型が用いられている。(日立の SR8000 は疑似ベクトルであり、スカラーに分類することもできる。)

計算サーバのほとんどは、グラフィックス機能をまったく持たないか、持っていたとしてもきわめて貧弱なものである。また、OpenGL などのグラフィックスライブラリが使えることもまれである。したがって、計算サーバ上でのレンダリングはソフトウェアベースのものとなる。

計算サーバは基本的に並列型なので、ノード間のネットワーク通信はきわめて高速である。通常、通信バンド幅は数 GB/s から始まり、大規模なものではそれ以上におよぶこともある。一方、計算サーバには外部システムとのデータをやりとりするためのネットワークインターフェイスがあるが、これはそれほど高速なわけではない。

4.1.2 OS と開発環境

計算サーバでの OS は、そのほとんどが UNIX をベースとした OS である。特に最近では、主に PC クラスタを中心として、Linux ベースのものも増加している。基本的には 64 ビット OS であるが、PC クラスタの場合には、32 ビットのものもまだ多い。

開発環境は、主に Fortran および C、C++ 言語ユーザー向けに整備されている。プログラミングモデルとしては、分散メモリ型と共有メモリ型のどちらかあるいは両方が採用されている。分散メモリ型の場合は MPI ライブラリを用いたメッセージパッシングによるプログラミングモデルとなる。一方、共有メモリ型の場合には、p-thread ライブラリを用いてスレッドを陽に制御するか、または、OpenMP コンパイラディレクティブなどによる自動並列化機能を用いる。さらに、プロセッサのタイプがベクトル型である場合には、コンパイラディレクティブを併用した自動ベクトル化機能を用いる。

このように、プログラミングモデルや開発ツールには多種多様なものが存在するが、ギリギリのチューニングが求められるような極端な場合を除き、一般的には、最小限 MPI による分散メモリ型のプログラミングモデルを用いておけば、ほとんどの計算サーバ上でそこそこの性能を得ることができる。これは、フラット MPI モデルと呼ばれている。ただし、ベクトル型プロセッサの場合には、さらにベクトル化が必要である。

4.1.3 使い勝手

計算サーバは高価であり、通常は複数のユーザーにより共有される。したがって、ほとんどの場合、計算サーバはユーザーの普段いる場所からある程度離れた場所に配置される。それは、例えば専用の計算機ルームや別フロア、別の建物に置かれるか、あるいは遠く離れた場所に独立して計算機センターが設定されることもある。大学、国立研究所あるいは企業の計算機センターがそれにあたる。ユーザーは、計算機センター内部のクライアント端末からセンター内 LAN 経由で、またはユーザー側のクライアント端末から LAN 経由または場合によりインターネット越しに、遠隔ログインすることにより計算サーバを用いる。

計算機センターなどに配置されるような大規模な計算サーバでは、通常 TSS 環境とバッチ処理環境の両方が用意されている。TSS 環境では、プログラムのソースファイルや入出力データファイルの遠隔転送、プログラムのコンパイル、そして小規模データを用いた実行とテストを行うことができる。その他、プレポスト処理に関連する比較的小規模なデータ処理だけが許されている。この場合、TSS 環境でユーザーが実行可能なプロセスのメモリ容量、並列実行のプロセッサ数、および実行時間には厳しい制約が科せられている。したがって、計算サーバの能力をフルに利用したり、より長時間のジョブを実行するためには、バッチ処理環境にジョブを投入する必要がある。バッチジョブキューが空いている場合を除き、通常は、ジョブの投入後、ジョブ自体の実行時間の数倍程度の待ち時間を経た後にジョブが実行される。

計算サーバへのネットワーク接続は、もっぱら telnet、rsh または ssh などの遠隔ログインと、ftp、rcp または scp、sftp などの遠隔ファイル転送により行う。計算機センター内部のクライアント端末からはセンター内 LAN 経由で telnet または rsh が、一方、インターネット経由ではセキュリティ上の理由から SSH が用いられる。これ例外の方法によるネットワークアクセス、例えば HTTP や SMTP、あるいは独自の TCP ポート ID によるアクセスなどは、基本的に許されていないことが多い。

なお、計算機センターや計算機を保有管理する研究所のポリシーによっては、その LAN 内においてであるが、X Window System や OpenGL におけるクライアントサーバ通信が許されているところもある。このときは、X や OpenGL のクライアントを計算サーバの TSS 環境で実行し、これが LAN に接続されたクライアント端末上の X サーバにネットワーク接続することにより、ある程度対話的な環境を得ることができる。一方、これを SSH のポートフォワーディング機能を用いてインターネット越しに行うことも、サイトによっては可能である。ただし、アプリケーションの設計によっては、ネットワーク転送速度が遅すぎてその利用が難しくなることもありうる。

4.1.4 性能

計算サーバの性能スペックとしては、数 Tflops の浮動少数点演算性能、数 TB のメモリ容量と数 PB の HDD 容量が現状における標準的なものとなっている。メモリアクセス速度や HDD アクセス速度についても、インターリーブや RAID によりある程度高速化されている場合が多い。

とはいえ、これを複数ユーザーで共有する以上、TSS およびバッチ処理において一つのプロセスやジョブが利用可能な CPU 資源はある程度制限されたものとなる。

概算ではあるが、バッチ環境において投入可能なジョブの実行速度、および利用可能な計算速度やメモリ容量は、それをユーザー端末で実行する場合と比較して、ユーザー端末の能力の十倍から百倍程度と考えてよい。すなわち、浮動少数点演算性能で 10 から 100Gflops、メモリ容量で 10 から 100GB、HDD 容量で 1TB 程度である。それ以上だとユーザーあたりの計算機コストが高価すぎる一方で、それ以下では今度はわざわざこの計算機を使ってくれるユーザーがいなくなってしまう。

ただし、一つの計算サーバの利用期間は通常 4 年から 5 年程度であるが、この間にクライアント端末である WS や PC の性能が一桁以上向上する場合がある。そうすると、利用期間の後期にもなると、待ち時間を含めた実効的な計算時間についてクライアント端末側で実行するのとそれほど差がなくなってしまう。したがって、利用可能

なメモリ容量の多さが、計算サーバを利用する際の最大のメリットとなる。逆に言えば、本来計算サーバ側で実行すべきデータ処理を無理にクライアント端末側で行おうとすれば、かなり厳しい省メモリ設計を強いられることになる。

4.1.5 プレポスト処理

計算サーバ側でプレポスト処理を行わせる場合、簡単なのは、計算サーバの TSS 環境を利用することである。制限されているとは言え、各種のプレポスト処理に要する計算時間は解析時間そのものに比べるとかなり短く、TSS 環境上での制約時間内である程度のデータ処理が可能である。また、クライアント端末に比べればより多くのメモリ容量が利用できることが多い。特に、解析により出力される大量の解析結果データファイルに対して、それをネットワークを介してクライアント端末まで持ち帰る必要なしに、自由自在にアクセス可能なところがメリットである。

計算サーバ側でプレポスト処理を行わせる際の問題点としては、まず、クライアント端末からの対話性やレスポンスの問題が挙げられる。特に、クライアント端末上でのグラフィックスによる解析モデルデータの可視化操作や選択編集操作については、X Window System や OpenGL アプリケーションを起動できるのでない限り、あまり期待できない。

また、プレポスト処理がより大規模になれば、最終的にはこれをジョブとしてバッチ処理環境に投入することになる。この場合はさらに、並列化やベクトル化など、その計算サーバ向けのパフォーマンスチューニング作業が必要となる。このとき、バッチ処理環境のターンアラウンドタイムはきわめて長いので、解析ジョブとの一体化あるいは連続実行が出来ることが望ましい。

4.2 クライアント 端末

クライアント端末の利用パターンとしては、通常各ユーザーがこれを占有する形をとる。クライアント端末は常にユーザーのそばに置かれており、いつどのようなときにも使うことができる。一方、計算速度やメモリ、HDD 容量の観点から言えば、計算サーバに比べ端末は非力である。ただし、グラフィックス性能については、クライアント端末のほうが優れていることが多い。

4.2.1 ハードウェアアーキテクチャ

ハードウェア構成で見たクライアント端末の分類としては、パーソナルコンピュータおよびワークステーションという分類がある。パーソナルコンピュータとしてはいわゆる PC、つまり IBM PC/AT 機コンパチブルおよび、Apple 社の Macintosh が挙げられる。一方、ワークステーションとしては、その市場は主に UNIX ワークステーションと Windows ワークステーション (いわゆる PC ワークステーション) に分けられる。前者は Sun、HP、IBM、SGI 社などの RISC チップベースのものであり、また、64 ビットアーキテクチャとなっている。一方、後者は基本的には比較的高級な PC であり、当然、x86 チップベースであるため、そのほとんどは 32 ビットアーキテクチャのままである。

プレポスト処理の環境としては、一部のユーザーは PC や Macintosh などのパーソナルコンピュータを用いているが、基本的にはワークステーション市場を想定すればよいと思われる。なお、現在この市場の大部分は 64 ビットアーキテクチャを必要とする一部のハイエンド市場を除いて、ほとんど Windows ワークステーションで占められている。

クライアント端末はある程度のグラフィックス機能を有する。これはチップセットに内蔵されているか、または別に装着されたグラフィックスカードにより機能する。

したがって、ノート PC や省スペース PC よりも、ミドルタワー PC やワークステーションのほうがより高性能なグラフィックスカードが搭載できるため、描画性能が高いことが多い。

クライアント端末の LAN 環境は、長らく 100BaseT が標準であったが、最近では GbE も普及しつつある。ちなみに、100BaseT のピーク通信バンド幅は 12.5MB/s、1000BaseT (GbE) の場合、125MB/s である。

4.2.2 OS と開発環境

クライアント端末向けの OS としては、Windows、MacOS および各種商用 Unix を想定する。ただし、主に大学などを中心に一部で Linux が用いられている。

UNIX ワークステーションでは Sun の Solaris、HP の HP-UX、IBM の AIX、SGI の Irix など各社独自の商用 UNIX が用いられている。Macintosh では MacOS が用いられている。ただし、現在の MacOS (MacOS-X) は基本的にはほぼ UNIX コンパチブルといってよい。一方、PC については、Windows と Linux の選択子が存在する。Windows については、Cygwin などのフリーソフトを導入することにより、UNIX コンパチブルな環境を実現することができる。

すべての商用 UNIX、および最近の Linux、MacOS および Windows では、64 ビット環境が利用できる。ただし、PC の場合には AMD64 あるいは EM64T が有効であるものについて、MacOS ならば PowerPC G5 以降のマシンにおいて、これらが利用できる。したがって、現状利用されているクライアント端末の大部分は、32 ビット環境である。

クライアント端末上で主に用いられる開発言語としては、Fortran、C、C++ 言語を中心に、その他 Java、C#、Visual Basic、Objective-C、そして Perl、Tcl/Tk などのスクリプト言語など、多様な選択子が存在する。

ただし、GUI および三次元グラフィックスを用いるアプリケーション開発の場合は、一部の例外を除き、もっぱら C および C++ が選択されるようである。Java、C# および Visual Basic などについては、これらを用いた高効率なアプリケーションの開発は多くの場合きわめて困難であるため、現状としてパッケージソフトウェアの開発に用いられている例は少ない。同様の理由により、Perl や Python などのスクリプト言語もまた除外される。

GUI ライブラリについては、OS および計算機言語の選択により多様な選択子が存在する。多くの商用 UNIX では Motif が用いられる一方、Linux ではこれに加えて Gtk+ および Qt がメジャーである。Macintosh では主に Apple 社標準の Cocoa および Carbon が用いられるが、X Window System を導入すれば Linux と同様の環境が利用できる。Windows においては、言語の選択に関しきわめて複雑な状況となっている。C 言語を用いるならば、低レベル API の WIN32/64 がある。一方、C++ を用いる場合には、さらに MFC および .NET 環境が選択子として加わる。これに加えて、Cygwin の導入により、Motif、Gtk+ および Qt の利用が可能である。

グラフィックスライブラリとしては、OpenGL および DirectX が用いられている。このうち、前者はクロスプラットフォームであるが、後者の DirectX は専ら Windows 環境で主にゲームやコンテンツ開発向けに用いられている。

Java については、これ自身が独自のプラットフォームを構成している。GUI として Swing、三次元グラフィックスとして Java3D を利用する場合、これらを用いたアプリケーションはきわめて移植性が高く、ここで想定しているクライアント端末のほとんどで動作する。

Visual Basic および C# については、これらはもっぱら Windows 環境のみで動作する。GUI として .NET、三次元グラフィックスとして DirectX を用いることができる。

4.2.3 性能

クライアント端末の計算能力については、1Gflops 前後の浮動少数点演算性能、1GB から 2GB 程度のメモリ容量と数百 GB から 1TB 程度の HDD 容量が現状における標準的なものとなっている。また、メモリアクセス速度および HDD アクセス速度については、それぞれ数 GB/s、数十 MB/s 程度である。

ただし、32 ビット OS では、一つのプロセスは最大でも 2GB 程度のメモリしか利用できない。また、一つのファイルサイズに 4GB の制約がある可能性がある。

描画性能としては、現状では数千万ポリゴン / 秒が標準であるが、高速なグラフィックスカードの場合には、数億ポリゴン / 秒もの性能を発揮することがある。ただし、プレポスト処理アプリケーションについては、グラフィックスカードそのものの描画性能よりも、グラフィックスカードへの通信経路である AGP あるいは PCI Express バスのデータ転送バンド幅がボトルネックとなっていることが多い。

計算サーバとのネットワーク通信については、もし計算サーバが同じ LAN 内に存在すれば、GbE ベースの通信をさせることも可能である。ただし、現状の PC での実装では、最高でもピークの半分の性能、すなわち約 60MB/s 程度のスループットしか望めない。実際には、多くの場合、通信経路の途中で 100baseT のスイッチやルータを経由し、場合によりインターネットを介することもあるため、通信速度としては 1MB/s から 10MB/s 程度が妥当であろう。

4.3 高速化

計算やデータ処理、あるいは描画を高速化すること、いわゆるパフォーマンスチューニングを行うためには、計算機ハードウェアの構造を考慮しながらプログラムの設計を変更する必要がある。考慮すべきハードウェア要素として、ここでは主に、プロセッサの種類ごとのチューニングと並列化におけるポイントについて述べる。続いて、グラフィックスカードやチップセット内蔵のグラフィックス機能を用いて、描画処理を高速化する際のポイントについても説明する。

4.3.1 スカラープロセッサでの高速化

スカラープロセッサではその記憶システムがアクセス速度に関して階層化されていることが特長である。まず、比較的高速なプロセッサと低速なメモリが接続され、その速度ギャップを複数段のキャッシュが埋める構造になっている。また、プロセッサの内部においては、複数の演算器が並列に動作できるようになっているが、これらの稼働率を向上するために、なるべく多くのレジスタにデータが配置されるような設計が望ましい。

近年の傾向では、コンシューマ市場における動画や音声など、ストリーミング型のマルチメディアアプリケーションの需要増大により、スカラープロセッサにおいてもメモリアクセス速度が無視できないものとなりつつある。メモリクロックおよび FSB クロックの向上、DDR および DDR2 メモリの採用やデュアルチャネル化などによるメモリアクセス性能の向上がなされている。とはいえ、プロセッサ自体においても高クロック化に加えてハイパースレディング、デュアルコア、さらにクワドコアといったマルチコア化が進行しつつあり、メモリアクセス速度とプロセッサの演算能力とのギャップはむしろ広がる傾向がある。

スカラープロセッサにおける性能向上のポイントは、データ処理におけるデータアクセスのパターンを階層化し、その結果データアクセスの多くがメモリではなくレジスタまたはキャッシュ上で行われるようにすることである。これを実現するには、データ処理の順番を変更したり、ループの構造を書き直すなどの、設計上の変更が必要となることがある。

コンパイラの最適化能力だけではそれほど効果的なチューニングを行うことはで

きない。とはいえ、最終的な機械語コード生成はコンパイラの仕事である。したがって、チューニング戦略の基本は、コンパイラにうまく仕事をさせることができるように、プログラムを再構成することになる。ポイントとして、以下のようなものが挙げられる。

- 複雑なデータ構造における最適化

多くのコンパイラは配列への順次アクセスなど、比較的単純なデータ構造へのアクセスパターンについてはこれを認識し、それなりの最適化をかけてくれるようになっている。しかし、間接インデックス参照やポインタアクセスが入るようなより複雑なデータアクセスパターンを有するコードでは、コンパイラは互換性を保証するため最適化に関して保守的になってしまうことが多い。

- レジスタの利用

レジスタの有効利用には、ループアンローリングが有効である。単純なデータ構造ならば、コンパイラが元のコードを自動的にループアンローリングしてくれる場合があるが、複雑なデータ構造を用いている場合、それはほとんど期待できない。ただし、コンパイラはループ内のスカラー変数（配列やポインタでないもの）についてはうまくレジスタ割当を行ってくれるようである。プログラムは手動でループアンローリングを行い、その際に、配列やポインタによるデータ参照をいったんテンポラリなスカラー変数へ明示的に格納してやる必要がある。

- キャッシュの利用

キャッシュの有効利用としては、ブロック化が有効である。単純なデータ構造ならば、コンパイラが元のコードをキャッシュを有効利用するようなコードに変換してくれる場合があるが、複雑なデータ構造を用いている場合、それはほとんど期待できない。プログラムは明示的にブロック化をおこなってやる必要がある。つまり、キャッシュに入らないような比較的大きなサイズのデータ領域へのアクセスについて、これをブロック分割する。さらに、各ブロックをいったんちょうどキャッシュに格納できるような容量のデータ領域に明示的に格納し、そこで演算を行うようにする。

- メモリアクセス量の削減

レジスタやキャッシュに載せられないようなメモリアクセス中心のアルゴリズムにおいては、プロセッサでの演算能力ではなく、メモリへのアクセスが性能上のボトルネックである。したがって、少しでもメモリアクセス回数やデータ量を少なくすることができれば、その分実行速度は向上すると考えて良い。例えば、複数のループを融合したり、整数や浮動小数点データの精度を十分なレベルまで落とすことでデータサイズを削減する、などである。

通常、上記のようなスカラーチューニングを行った場合、行わない場合と比べて多くの場合二倍から三倍程度の性能向上が期待できる。これ自体はそれほど大した数字ではないとしても、これに共有メモリ型マシンでの並列化を組み合わせた場合に、大きな差が現れることがある。デュアルコア、デュアルプロセッサマシンを含め、共有メモリ型の並列計算機ではシステムバス速度がボトルネックとなりやすいため、レジスタやキャッシュをどれだけ有効利用しているかが性能向上のためのキーとなるからである。

逆に、スカラーチューニングを試みることにより、そのコードがキャッシュを有効利用できるのか、それともメモリアクセスがネックとなっているのが判別出来る。それにしたがって並列処理の戦略を立てることができる。キャッシュの有効利用が可能ならば、SMP サーバーなどの共有メモリ型並列計算機でもある程度まではスケラブルな性能向上が得られる。一方、もしそうでなければ、PC クラスタなどの分散メモリ型並列計算機を選択すべきである。また、この場合、各ノードをデュアルプロセッサ構成にする意味はあまりないことが分かる。

4.3.2 ベクトルプロセッサでの高速化

スカラープロセッサと違い、ベクトルプロセッサは高速なプロセッサと高速なメモリシステムによって構成される。したがって、そのハードウェア構成は高価なものになる一方で、多くの数値計算アプリケーションにおいて比較的性能は出しやすい。ただし、性能向上にはベクトル化が不可欠である。

ベクトル化とは何かを理解するためには、ベクトルプロセッサマシンのハードウェア構成の理解が不可欠である。ベクトルプロセッサでは整数および浮動小数点演算パイプラインについてこれらを非常に細かいステージに分割し、高クロック化を達成している。また、最近のベクトルプロセッサはこのパイプラインを複数保持しているものが多い。スカラープロセッサの場合と異なり、ベクトルプロセッサの各パイプラインは連続して同じ内容の演算を処理する場合についてのみ、その性能を発揮することができる。これは、配列上のデータを同じパターンで処理していくような場合に相当する。また、配列への規則的なデータアクセスを効率化するため、きわめて多数のバンクを用いたメモリアンタリーピングを用いている。さらに、メモリアクセスがボトルネックになることを防ぐため、レジスタの配列であるベクトルレジスタをいくつかが有している。以上のように、ベクトルプロセッサでは大規模配列に対してある程度規則的なデータアクセスパターンを有するような数値演算やデータ処理に限って高速化することができる。これは多くの数値計算アプリケーション、特にメインの解析プログラムにおいて頻繁に登場するパターンであるが、プレポスト処理については必ずしもそうではない。

ベクトル化は基本的にはコンパイラが行う仕事であるが、プログラマはコンパイラをアシストするために、ソースコード上にベクトル化のためのコンパイラディレクティブを挿入しなければならないことが多い。ベクトル化は各ループについて適用される。多重ループの場合は、最内側ループがベクトル化される。ベクトル化したいループについて、基本的に避けるべきポイントを以下に述べる。

- 複雑な手続き呼び出し。単純なものならばインライン展開される。
- ループからの脱出とやり直し (goto 文、break と continue など)。
- ポインタアクセス。したがって、オブジェクトへの参照やリスト構造などはそのままではベクトル化できない。

また、性能を向上させるためのポイントについては、以下のようになる。

- ループの実行回数、すなわちベクトル長を長くする。
- 多重ループについては、内側ループのベクトル長が長くなるようにする。
- なるべく間接インデックス参照をしないようにする。
- なるべく条件分岐をしないようにする。
- メモリ参照を減らし、ベクトルレジスタを有効利用する。
- コンパイラがベクトル化を拒絶するループについては、もしベクトル化が可能であれば、プログラマがディレクティブを挿入して強制ベクトル化を行う。

ベクトルプロセッサの問題点は、対応するスカラー性能が相対的に低いため、コードをベクトル化した場合としない場合とで性能差がきわめて大きいことである。最近のプロセッサの場合では、性能差は有に百倍を超えることがある。したがって、ベクトル化されていないプログラムをベクトルプロセッサで実行することはやむを得ない場合を除き、なるべく避けたほうがよい。

4.3.3 並列化

現状、計算サーバのほとんどは並列型の計算機システムである。したがって、メインの解析にはなんらかの並列計算が不可欠となっており、そのために領域分割型アプ

ローチが用いられることが多い。領域分割型アプローチは MPI ライブラリを用いた分散メモリ型のメッセージパッシングによるプログラミングモデルとも相性がよい。並列化のポイントを簡潔に述べる。

- 各プロセッサの計算負荷をバランスさせる。そのために、動的負荷分散や領域再配分が必要となるかも知れない。
- プロセッサ間通信の負荷を削減する。プロセッサ間で通信されるデータ量を減らす。また、プロセッサ間で同期をとる回数を減らす。
- 並列化はあくまで高速化のための一手段に過ぎない。並列化を行うまえに、まず単一プロセスでのスカラーまたはベクトルチューニングの検討を忘れないこと。

もし、プレポスト処理の一部あるいはすべてをここで行うならば、メインの場合と同様に並列化が要求されることになる。このとき、メインと同じような領域分割型アプローチが取れば、メインと同じデータ構造およびメモリレイアウト、あるいは入出力ファイル形式を用いることが出来るので、計算効率上都合がよい。

4.3.4 描画処理の高速化

有限要素解析のための可視化や選択編集処理において必要とされるコンピュータグラフィックス技術は、CAD や景観シミュレーション、映画やゲームなど他分野におけるものとある程度の共通部分はあるものの、異なる部分もまた多い。

基本的に、プレポスト処理におけるグラフィックスとは専ら、メッシュやその表面、任意断面など作業中に現れる各種解析データを描画し、あるいはその一部をマウスによりピック・選択させることである。まず、解析データはワールド座標系で定義されている。次に、解析データから、多数の線分、三角形、また場合によりさらに文字列やイメージアイコンなどが生成される。これらは同じくワールド座標系において定義されている。これらの描画プリミティブ群が順にウィンドウ内にレンダリングされていくことになる。なお、レンダリングそのものはクライアント端末のグラフィックスハードウェアによって行われる。また、プリミティブ群のピック処理はレンダリングとほぼ同じ過程に従いハードウェアベースで行われる。

ここで、クライアント端末のグラフィックス能力であるが、最近では比較的高速なグラフィックスカードを備えたものが増えてきている。グラフィックスカード上に搭載された GPU とよばれるグラフィックス専用チップのここ数年における著しい性能向上により、すでに数億ポリゴン/s もの描画性能が期待できるようになっている。ただし、プレポスト処理におけるグラフィックスの使用パターンの場合には、性能のボトルネックは GPU 性能ではなく、むしろシステムメモリとグラフィックスカード間の通信経路である AGP あるいは PCI Express バスのデータ転送バンド幅にある。場合によってはさらに、CPU での計算やメモリへの読み書きもまた無視できなくなることがありうる。現状ではこれらの制約により、プレポスト処理におけるグラフィックス性能は数千万ポリゴン / 秒程度に制限される。したがって、レンダリング性能の向上に必要なのは、まず、CPU で余計な計算やメモリアクセスを行わないこと、次に、グラフィックスカードへ送るプリミティブ群のデータサイズをできる限り削減すること、の二つである。

さて、プレポスト処理におけるリアルタイム操作の高速化を計るにあたり、もっとも優先順位が高いのは、視点移動や拡大縮小などのビューイングパラメータ変更操作である。ビューイングパラメータが変更されると、ワールド座標系におかれたレンダリング対象オブジェクトは視野座標系を経由してデバイス座標系まで変換され、続いて光源処理、ラスタライゼーションと続く。ここで、ビューイングパラメータ変更操作を何度も行っている間は、レンダリングされているオブジェクトに関する座標や法線方向ベクトル、あるいは色などの情報は、それが定義されるワールド座標系上に留まっていっさい変更されず、静的な存在となっている。したがって、レンダリング対象オブジェクトを構成する描画プリミティブ群をメモリ上にキャッシュしておくことは、対話性を向上するための有効な手段となる。

また、可能であれば、これをメモリではなくグラフィックスカード側のメモリや VRAM に格納できれば、さらなる描画性能の向上が期待できる。ただし、グラフィックスカードの VRAM 容量には制限があるため、大規模解析では注意する必要がある。

一方、ユーザーの行うリアルタイム操作がビューイングパラメータの変更ではなく、その他の可視化パラメータを変更する場合があります。例えば、時間方向変化のアニメーション表示や、変形図での変形拡大率の変更、断面位置のスライドなどである。このような場合には、描画プリミティブ群のキャッシングはあまり意味が無く、個々のケースごとに、元の解析データから描画プリミティブ群を高速に評価生成できるようなアルゴリズムとデータ構造が必要となる。

プレポスト処理においてレンダリングやピック処理を高速化するためのポイントをまとめると、以下のようになる。

- もしメモリに余裕があれば、グラフィックス API をすぐに呼び出せるようなデータ形式ですべてのプリミティブデータをメモリ上に保存する。それができなければ、このデータ形式を出来る限り高速に生成できるように可視化編集対象となる解析モデルのデータ構造を構成する。
- メッシュ境界表面情報のように三角形データに裏表の区別があるものは、その情報を用いる。
- なるべく同じ種類のプリミティブが連続してレンダリングされるようにする。描画プリミティブ群を線分、三角形ごとにグループ分けする。
- レンダリング中は、色、透明度などの描画属性設定の変更をなるべく避ける。属性変更はグループ化されたプリミティブ群の先頭で行う。
- グラフィックスカードに送り込むデータ量を圧縮する。可能ならば、プリミティブ群をストリップ化する。

第5章 大規模有限要素解析

大規模な有限要素解析では、問題の規模や自由度の増加とともに、プレポスト処理が扱うデータ量やそのデータ処理時間が増加し、解析規模によってはそのうち扱いきれなくなってくる。特に、非定常解析や非線形解析などの複数ステップを有する解析については、特別な扱いが必要となる。

ここでは、大規模有限要素解析におけるプレポスト処理の問題点を、その解析の種類ごとに分析、検討する。最初に、有限要素解析についての一般的な問題について、これらを定常解析と非定常解析に分けて取りあげ、続いて、構造解析および流体解析について順に検討する。

なお、本報告におけるプレポスト処理は、三次元ボリウム (ソリッド) 要素を用いた有限要素解析をその対象とする。線形あるいは非線形解析について、これらの定常問題または非定常問題をサポートする。要素の種類としては、四面体の一次要素あるいは二次要素を用いるものとする。

また、本研究ではメッシュ生成機能については考慮しない。つまり、四面体要素のメッシュがすでに与えられているものとする。本研究ではこのメッシュに対してプレ処理として、解析条件、すなわち、境界条件、初期条件や材料物性を設定する機能および、ポスト処理として、解析終了後、その解析結果を可視化する機能とを対象としている。

5.1 定常解析

ここで扱われる問題は基本的には三次元問題である。二次元解析の、特に定常問題では、ある程度対話性を犠牲にしてもプレポスト処理ではそれほど困らない。しかしながら、三次元解析モデルに対するプレポスト処理では対話性は必須のものである。特に、モデルの移動、回転や拡大縮小などのビューイングパラメータ変更にはリアルタイムレベルのレスポンスが要求される。また、このようなレスポンスが得られなければ、プレ処理において解析条件の設定作業を効率的に行うことはできない。

一方、三次元ボリウム領域に分布する各種のスカラー、ベクトル量に対して、任意断面、等値面、パーティクルトレースやボリウムレンダリングなど、多くの可視化手法が存在する。これらの可視化手法では、例えば、断面位置、等値面の値やパーティクルの初期位置など、それぞれ変更可能な可視化パラメータが存在する。三次元空間上の複雑な物理量分布を効果的に把握するには、ビューイングパラメータに加えて、これらの可視化パラメータの変更についても、きわめて対話的な環境が要求される。

さて、大規模有限要素のプレポスト処理における問題点は、基本的には、データの巨大さに対しクライアント端末の能力が非力すぎるため、普通の方法ではそれを扱えないことである。まず、単純に描画やオブジェクトのピック、選択操作のレスポンスが遅くなることがある。表示すべき描画プリミティブ数が増加すれば、そのうちに対話処理を行うのに十分なレスポンスを得られなくなってくる。

しかしながら、より深刻なのは、データが大きすぎてクライアント端末のメモリ上に載らないため、必然的に HDD とのやりとりが増加してアプリケーションのパフォーマンスや安定性が急激に低下するという問題である。これに対しては、クライアント端末の能力を向上するという方法とともに、プレポスト処理について省メモリ設計を徹底することで、ある程度規模に対してスケーラブルな環境を確保することが出来る。

ここで想定する数千万から数億自由度の解析規模では、そのボトルネックの多くがメモリ量である。逆に、処理に必要なメモリ量の問題さえなんとかなれば、同時に、データ処理速度および描画処理速度をうまく効率化することにより、一般に普及しているクライアント端末の能力の範囲内でも、このような大規模解析に対して使用に耐えうるプレポスト処理環境を提供することができる。

もう一つの問題は、計算サーバからクライアント端末へのデータ転送である。計算サーバ上で取り扱われるデータ量と比較して、ネットワーク上の転送バンド幅はきわめて狭いのが現状である。このため、解析の前後についてクライアント端末と計算サーバとの間で毎回数時間から最悪数日といったオーダーの時間ひたすらデータ転送を行う必要がある。かといって、巨大なデータを計算サーバにしばらく置きっぱなしにしておけば、今度は計算サーバ上で保持するデータ量に関する課金が問題となる。

これに対しては、計算サーバ上でプレポスト処理に必要なデータだけを選別し、それらをクライアント端末にネットワーク転送するというアプローチが有効である。すなわち、本来プレポスト処理に分類されるデータ処理のうち、どこまでを計算サーバ上で行い、どこからをクライアント端末上で行うかの機能分割の問題となる。なお、計算サーバ上で行えるデータ処理のうち、解析開始前にあらかじめそれらを詳細に指定することが可能であれば、これらのデータ処理をメインの解析本体と連結してバッチ処理環境で一つのバッチジョブとして実行することもありうる。

5.2 非定常解析

全節では定常解析におけるプレポスト処理の問題について触れたが、非定常解析の場合にはそのポスト処理に関してより深刻な問題が生じる。すなわち、複数の時間ステップにわたる大量の解析結果が生成されることと、これらの時系列アニメーション表示の必要性である。

非定常解析では、有限要素解析の内部で取り扱われる時間ステップ間隔と、その出力結果として吐き出される時間ステップ間隔とが異なる場合が多い。特に、衝突問題や塑性加工問題においてよく用いられる陽解法ベースの構造解析や、流速について陽的な扱いをするスキームに基づく流体解析では、その時間ステップは通常きわめて細かい。一つの解析ケースごとに、数万ステップから数百万ステップにのぼることもある。一方、出力可能な解析結果ファイルのサイズに制限がある場合も多い。このような場合、本来解析コード内で扱っている時間ステップについて、そのいくつかおき、例えば 100 ステップや 1000 ステップに一回程度、でしか出力できないことになる。

したがって、時間方向への精度が要求されるすべてのポスト処理は、本来はメインの解析コード内でそれを処理すべきである。特に、時間進行を考慮したパーティクルトレース図については、精度上無視できないような場合もありうるため、解析コードでのパーティクルトレース生成というのは比較的普通に行われる。

また、かろうじて計算サーバの HDD 上に出力することができた大量の解析結果ファイルについても、今度はそれをどうやってプレポスト処理を行うクライアント端末に転送するのか、さらに、クライアント端末の HDD 上にこの巨大なデータが収まるのか、といった各種の問題がある。したがって、たとえメインの解析本体とポスト処理との連結が出来ない場合にも、ポスト処理の一部を計算サーバ上で行うことにはある程度意味がある。

以上の問題については、前節での議論と同様に、メインの解析と連結してバッチ処理環境内で処理できるものについてはそれを行い、そうでなくても、プレポスト処理の一部で対話性を必要としない部分のデータ処理については、これを計算サーバ上で行えるようにしておくことが必要である。

一方、時系列アニメーション表示についても、その扱うデータ量の巨大さからくる各種の問題がある。ここでは、アニメーション画像生成ではなく、クライアント端末上でのリアルタイムのアニメーション表示について検討する。時間ステップごとに各種物理量を可視化していく際に、可視化に必要なデータが各ステップごとに用意されているものとする。このとき、全ステップのデータをあらかじめメモリ上にロードし

ておけるのか、それとも各ステップごとに HDD 上から読み出しながら描画していくのか、という選択がある。前者にはクライアント端末でのメモリ容量制約による表示ステップ数の限界があり、一方、後者では HDD からの読み出し時間が表示速度のボトルネックとなる。

5.3 構造解析

原子炉圧力容器や自動車のエンジンブロック、発電用タービンといった大規模複雑構造物は、数十、数百から小部品を含めると数万に至るようなきわめて多数の部品より構成されるアセンブリ構造物である。特にこれらは、その部品群が互いに溶接、接着やリベット、ボルト接続などにより比較的剛に拘束されている。そのため、一部の部材だけの部分的な変形や応力を求めたいときにも、構造物全体レベルでの解析が要求される。一方でこれらの構造物は、高層建築物や航空機、自動車のボディーなどのような、梁構造または薄肉構造として近似できるものとは異なり、各部材がどちらかといえばバルク形状である。そのため、基本設計を終えてより詳細な解析が必要な段階では、これらのビームやシェルなどの構造要素によるモデル化は難しく、連続体ソリッド要素による大規模有限要素モデルの作成が要求される。

5.3.1 解析対象となる問題と使用される要素

ここでは、大規模複雑アセンブリ構造物を三次元ソリッド要素によりモデル化した上で、これらの応力解析あるいは熱応力解析を対象とする。時間方向の取扱いとして静解析および動解析の双方をカバーする。ただし動解析の場合には、陰解法ベースで比較的粗い時間ステップをとるものとする。また、弾塑性解析や大変形解析といった増分法ベースの非線形解析も考慮にいれるものとする。さらに、その解析規模は数千万から数億自由度程度をターゲットとする。

三次元ソリッド要素としては、ここではもっぱら四面体二次要素を用いるものとする。数億自由度規模の構造解析が行える場合には、上記のように解析対象構造物について、その全体スケールをカバーする、いわゆる「まるごと解析」が行われる。また、その構造物を構成する部品のすべて、あるいはほぼ大部分について、これらをビームやシェルのような構造要素ではなく、連続体ソリッド要素として簡略化せずにモデル化するものとする。このような大規模な有限要素モデルのベースとなる形状モデルもまたきわめて複雑なものとなり、手動あるいは半自動によるメッシュ生成は事実上不可能である。したがって、メッシュは CAD モデルより自動生成するほかなく、三次元ソリッド四面体二次要素が選択されるものとする。

例えば、構造解析で一億自由度 (100M DOFs) のモデルでは、各節点は変位 u 、 v 、 w の 3 自由度を持つので、節点数は、 $N_{2nd} = 33M$ となる。なお、これは二次節点数である。

5.3.2 プレポスト処理

構造解析では、解析結果について以下のような可視化を扱う必要がある。

- メッシュ境界表面のスカラーコンター図。スカラー量としては、温度、相当応力、相当歪みなどがある。あるいは、応力テンソル、歪みテンソル、主応力ベクトル、主歪みベクトルについて、これらの各成分をスカラーとしてプロットする。
- 変形図。メッシュ境界表面における変位ベクトルを用いる。
- 任意断面でのスカラーコンター図。メッシュ境界表面と同様のスカラー量を扱う。

なお、これ以外にも、解析モデルの境界表面ではなく内部、すなわち三次元ボリューム領域での可視化が求められることがある。ただし、亀裂解析のようなケースを除けば、構造解析では一般にその物理量分布の性状は比較的単純であり、変形図や境界表面上でのスカラーコンターを見れば、ある程度の見当がついてしまうことが多い。

任意断面を切るニーズにしても、それはもともとの解析形状が複雑なためであることが多い。特に、管状あるいは球殻形状の構造物や、多数の薄肉部材が層状に配置されるような構造物の場合、解析形状の内側、裏側などその内部部材の詳細を把握するために、部材ごとの表面コンター表示と併用して用いられる。

つまり、どちらかと言えば、大規模三次元ソリッドの構造解析では、解析条件や解析結果における各種物理量の空間分布からくる複雑さよりは、その解析モデル形状自体の構成の複雑さからくるプレポスト処理の負荷が高い。これは、ポスト処理における解析結果の可視化だけでなく、プレ処理における境界条件や材料物性の設定の困難さを意味する。

5.4 流体解析

超並列計算機や PC クラスターの普及とともに、三次元流体解析シミュレーションもようやく実用期を迎えようとしている。ここでは基本的に有限要素法を用いた流体解析を扱うが、ここで扱われるプレポスト処理技術の多くは、同様に有限体積法の非構造格子セルを用いた解析においても適用可能である。

5.4.1 解析対象となる問題と使用される要素

流体問題として、主に三次元非圧縮性ナビエストークス方程式の解法を対象とする。これは、水や空気などについて、圧縮性を考慮する必要がない場合、すなわちマッハを越えない程度の比較的低速な流れ場について広く適用可能なケースである。

三次元流体解析においても、四面体要素のニーズは高い。また、比較的低次元な要素が用いられることが多いようである。特に、四面体の 4 節点一次要素について、各節点に流速三成分と圧力を保持し、流速と圧力について同次補間を行うような、p1p1 要素と呼ばれるタイプの要素の利用が増えている。以下の議論では、流体解析の場合には四面体一次要素を用いることとして、各節点は流速の u 、 v 、 w および圧力 p の計 4 成分を保持するものとする。なお、このタイプの要素の場合は四面体向けの体積積分公式により厳密積分ができるので、一般には積分点は存在しない。しかし、これを要素中心での一点積分と考えることもできる。どちらにせよ、要素ごとの物理量、および要素中心に配置される物理量については、プレポスト処理においてこれを考慮するものとする。

例えば、流体解析で一千万自由度 (10M DOFs) のモデルでは、各節点は流速 u 、 v 、 w および圧力 p の 4 自由度を持つので、節点数は、 $N_{1st} = 2.5M$ となる。なお、これは一次節点数である。

5.4.2 プレポスト処理

流体解析では、解析結果について以下のような可視化を扱う必要がある。

- メッシュ境界表面のスカラーコンター図。スカラー量としては、圧力、流速ノルムや温度などがある。
- 任意断面でのスカラーコンター図。メッシュ境界表面と同様のスカラー量を扱う。
- 任意断面での流速ベクトルの矢線図。
- ボリューム領域でのスカラー等値面図。メッシュ境界表面と同様のスカラー量を扱う。

- ボリューム領域での流速ベクトルのパーティクルトレース図。

三次元流体解析では、通常その解析形状はバルク形状をとることが多い。物体回りの流れはもとより、管流流れの場合にも、構造解析で頻繁に現れるような細い部材や薄い箇所はあまり出現せず、たとえあったとしても、その部分のメッシュ分割は非常に細かいことが多い。

したがって、構造解析の場合と比較すれば、メッシュで考えた場合の体積に対する表面積の割合が小さいため、メッシュ境界表面での物理量の可視化や境界条件編集における負荷は比較的小さい。三次元流体解析におけるプレポスト処理の計算負荷ボトルネックは、主に、これが三次元ボリューム領域を扱う必要性から生じてくる。すなわち、ボリューム領域に分布する各種物理量の可視化である。これは、例えば任意断面上におけるスカラー場のコンター図やベクトル場の矢線図、ボリューム領域内でのスカラー場の等値面生成やベクトル場のパーティクルトレース生成などがある。これらと非定常解析との組合せにより、ボリューム領域全体をカバーし、しかも多数の時間ステップより構成されるきわめて膨大な解析結果データを扱う必要性が頻繁に生じることが、三次元流体解析のプレポスト処理における深刻な問題となっている。

さらに、流体解析における物理量の分布性状は一般に複雑である。流体解析では流速分布が大小さまざまな渦を構成し、これらが時間とともに流れによって変化成長していく。また、これらの渦の動きにそって圧力分布がダイナミックに変化する。なんにせよ、構造解析での変形や応力集中、温度などの分布と比べれば、著しく複雑である。そのため、解析モデルの境界表面だけでなく、そのボリューム内部領域に対するさまざまな可視化手段が必要とされる。

三次元問題では、これが三次元空間方向で展開される。したがって、可視化手段にもまた高度な対話性が求められる。移動や回転、拡大縮小操作のレスポンスにはリアルタイムレベルが要求される。これに加えて、断面コンターや等値面、矢線やパーティクルトレースに関してそれらの各種可視化パラメータをダイナミックに変化させるような、ユーザーの試行錯誤的な操作パターンを効率的に支援していく必要がある。

なお、三次元空間に配置された複数オブジェクトの空間関係を把握するために、立体視あるいはその発展技術である仮想現実感技術もまた流体解析における重要な可視化支援手段となる。特に、矢線図やパーティクルトレース図のような線画ベースの可視化手法では、立体視が空間関係の把握に大きな役割をはたすことが多い。

第6章 既存アプローチのサーベイ

大規模複雑構造物の有限要素解析では、その解析規模は巨大であり、解析には常にその時代の最大クラス的能力をもつ計算サーバが必要とされてきた。これは、古くは大型計算機やベクトル型スーパーコンピュータであり、近年ではスカラーあるいはベクトル型 SMP ノードの超並列計算機 (MPP)、あるいは大規模 PC クラスタがその主流である。それに伴いメッシュや解析条件、解析結果を保持するために膨大な HDD スペースが必要とされる。

これに対し、クライアント端末としての PC やワークステーションにおいては、そのメモリおよび HDD 容量ともに比較的非力であり、状況によっては 1 つの解析ケースデータすらそこに保持できないこともありうる。

しかしながら、クライアント端末は計算サーバと異なり個人的に占有できるうえに、対話型操作環境が提供されている。しかも、グラフィックス表示能力に関しては計算サーバよりも優れていることが多い。これに対し、計算サーバでは主にバッチ処理による利用形態がメインとなっている。

以上のように、大規模複雑構造物の有限要素解析のためのプレポスト処理には、特別な工夫が要求されることが多い。大規模構造解析のためのプレポスト処理システムについては、これまでさまざまなアーキテクチャやシステム構成が提案されている。これは大雑把に分けて、クライアント端末側で処理するものと、計算サーバ側で処理するものとの二つの方法に分類されるものと思われる。ただし、プレポスト処理の機能タスクを適当に分割し、結果としてクライアント端末および計算サーバの両方を用いる場合もありうる。

6.1 主にクライアント側で処理する場合

まず、主にクライアント側で処理する場合には、クライアント端末のメモリおよび HDD 容量の少なさが問題となる。また、その端末のグラフィックス表示能力に比べて、そもそもの解析規模がこれを対話的に操作するには大きすぎる可能性がある。

これに対する一つの方法としては、解析データ全体を空間方向あるいは時間方向に簡略化、間引きすることで、全体スケールを扱えるようにしながらもデータ容量を削減するものがある。空間方向の簡略化では、より粗いメッシュにマップする「河合、吉村、吉岡、矢川ら、WH」、ボクセル化する「VCAD」¹⁾ 平野竜広、河合、吉村ら、AutoVK²⁾、二次から一次というようにより低次の要素に戻す、などの方法がある。一方、時間方向では、もっぱら時間ステップを間引くことになる。このような方法の欠点としては、それが構造解析における応力値のピークをばやかしてしまったり、あるいは精度を落してしまう可能性がある。

また、別の方法として、メッシュの一部、例えば特定の部品や空間領域のデータを取り出すものや、メッシュの境界表面や指定の断面上での情報に注目するというものがある。任意断面や等値面、パーティクルトレースについては、この幾何形状情報をまず計算サーバ上で評価、抽出しておき、次にこの幾何形状情報だけをクライアント端末へ転送して表示することができる「GeoFEM」³⁾。このような方法の欠点としては、ビューイングパラメータを除く各種の可視化パラメータをあらかじめ指定しておく必要があることである。可視化パラメータとは、例えば、注目する部品の種類や ID、抽出領域の範囲、断面の位置や方向、等値面の値、そしてパーティクルの初期位置や出現タイミングなどである。その変更には計算サーバでの作業や、場合によっては解析

のやり直しが必要とされる。ただし、二次元画像ではなく三次元幾何情報だけを計算サーバで生成するので、ビューイングパラメータはクライアント端末で変更することができる。したがって、モデルの移動や回転、拡大縮小は対話的に行うことができる。

さらに、問題となっているクライアント端末の基本ハードウェア能力の非力さに対し、思い切ってそのハードウェア構成を強化するといった方向もありうる。例えば、SGI の Onyx のようなグラフィックス表示機能を有する SMP 型の計算サーバを導入したり、あるいは、PC クラスタを用いて可視化を行う「ADV/Visual」「AVS」といったものがある。これにより、一旦解析データさえクライアント端末側に転送してしまえば、メモリや HDD 容量、計算能力の不足といった問題はある程度解決される。これに加えて、三次元立体視プロジェクターや仮想現実感デバイスが備わっている場合もありうる。ただし、このような強力なクライアント端末環境は特殊かつ比較的高価なものになってしまうため、現在はまだそれほど普及してはいない。

6.2 主にサーバ側で処理する場合

次に、主にサーバ側で処理する場合について述べる。この場合、計算サーバ上でプレポスト処理の大部分を行うことになる。そして、その出力としてレンダリング対象となる線分やポリゴンなどの幾何形状データを生成したり、あるいは、その場でさらにレンダリングを行い、最終出力として画像データを生成する。

前者の場合、これはすでに上で述べた部分抽出と同じものである。幾何形状データをいったんクライアント端末に転送し、そこでクライアント端末のグラフィックス表示機能を用いて対話的な操作を行うことができる。「GeoFEM」この場合の、後者のイメージ出力による方法に対する欠点として、データ量が比較的大きくなり、したがって通信量や HDD 容量もまた多くなる。さらに、データ抽出パラメータや可視化パラメータの変更には、再度計算サーバ側での処理が必要となる。

一方、後者の場合、三次元の幾何形状ではなく、二次元画像またはそのアニメーション動画像だけがクライアント端末に転送される。グラフィックスハードウェアを有するクライアント端末と比較すれば、計算サーバ自体は画像を生成することが苦手である。これに対し、並列化およびベクトル化（計算サーバがベクトル型の場合）により画像生成処理を高速化するという方法も存在する。「GeoFEM」なんにせよ、これらの画像データはあらかじめ決められたビューイングパラメータによるものであるため、時間方向表示のコントロールを除けば、クライアント端末上での対話操作の自由度はほとんどない。

また、計算サーバ上でのプレポスト処理に関する別の分類方法として、解析コードと同一のジョブ内において、解析コードの実行中またはその前後に行うものと、解析の開始前あるいは終了後に計算サーバ上で解析本体とは別プロセス、別ジョブとして行うものがある。

前者は、計算サーバでの HDD 容量が極端に少ない場合や、HDD でのデータ保持時間が極端に短い場合、あるいは、計算サーバをその計算能力の限界までフルに用いたような場合に行われることが多い。あるいは、流体解析において高精度なパーティクルトレースを出力したい場合などにも有効である。このとき、解析を行う直前に、プレポスト処理に必要なコマンドやパラメータを入力しておく必要がある。したがって、たとえ解析が TSS 環境で行われたとしても、そのレスポンスタイムは解析時間そのものになってしまう。そして、バッチ処理環境で行われる場合には、これにさらにジョブ待ち時間が加わり、ユーザーは数時間から場合により数日もの間待たされることになる。

一方、後者は計算開始前や計算終了後に計算サーバ上の HDD に保持してあるデータを対象として行う。この場合、前者にくらべれば比較的自由度が大きくなるが、HDD 上でのデータ容量制限や保持時間制限、あるいはそのための課金システムにより利用上の制約を受ける。また、計算サーバでは一般に限定的な TSS 環境が提供されている場合が多いが、これは主に解析プログラムのコンパイルやデバッグ、および、比較的小規模なデータ処理のために用意されている。したがって、TSS 環境で実行可能な

プロセスについては並列処理機能や使用可能なメモリ容量、および実行時間に厳しい制限を受ける。

なお、これらの中間として、解析中に計算サーバ上で画像を生成しながら、それをリアルタイムにクライアント端末に送り表示することができるシステムがいくつか存在する。「原研」また、計算サーバの利用条件によっては、トラッキング・ステアリング技術により解析をコントロールすることも可能である。

第7章 プレポスト 処理のための要素技術

7.1 ハッシュ検索

7.2 バケット 検索

7.3 要素から節点への振り分け

要素上の物理量を節点上に振り分けることで、精度の低下と引き替えにデータ量の圧縮が可能となる。ただし、可視化や解析条件編集のためのアルゴリズムによっては、節点上の物理量しか扱えないことも多いため、可視化や解析条件編集を目的とするならばこのデータ抽出操作はそこそこ意味のある方法である。

なお、ある要素内の指定位置での物理量評価を行いたいとき、たとえば、表やグラフ形式での出力を行いたいときには、節点上のデータを用いずに、もとの要素積分点上のデータに戻って再評価すべきである。

これを実際に行うアルゴリズムを検討する。物理量を要素から節点へ移す処理は、基本的には、各節点についてその周囲の要素の値を単純平均、あるいはなんらかの重み付き平均することになる。アルゴリズムとしては、要素ごとのループを回し、その要素から関連する節点へ適当な形で物理量を加算していくことになる。このアルゴリズムの演算量は節点数、要素数に比例する。また、このデータ処理の性能上のボトルネックは、もしすべてのデータがメモリ上に載っていればメモリへのアクセス、さもなければ、HDD への読み書きになる。

7.4 低次要素化

低次要素化とは、データ量削減のため、二次要素のメッシュから中間節点を省いて一次に落すことである。この操作は、前節と同様、可視化や解析条件の編集を目的にした場合においてはある程度妥当な方法である。

これを実際に行うアルゴリズムを検討する。物理量を二次節点から一次節点へ移す処理は、基本的には、比較的単純な節点番号のリナンバリング操作によって行える。この演算量は節点数に比例する。前節と同様、このデータ処理の性能上のボトルネックは、もしすべてのデータがメモリ上に載っていればメモリへのアクセス、さもなければ、HDD への読み書きになる。

7.5 要素辺の抽出

要素辺の抽出操作では、三次元ボリュームメッシュ内のすべての要素辺を抽出し、これらと要素との参照関係を明らかにする。三次元ボリューム (ソリッド) 要素は、要素ごとに何本かの辺 (稜線) を有する。例えば、四面体要素は 6 本の要素辺を持つ。一

方、各要素辺は、これを囲む複数の要素から共有される。あるメッシュ内に存在するすべての要素辺を重複無く数え上げ、また、これらと要素との参照関係を明らかにするのが、要素辺抽出操作である。

要素辺抽出操作は、主にメッシュの細分割や高次要素の生成に必要とされる。例えば、要素を複数に分割したり、高次要素のための高次節点を生成する際には、その要素辺上に節点を新たに生成する必要がある。そこで、抽出された各要素辺上に必要な節点を発生させれば、節点位置の重複や要素との参照関係について心配する必要がある。

要素辺抽出操作を行うアルゴリズムとして、ここではハッシュ検索による方法を採用する。まず、要素辺のテーブルを初期状態を空として作成する。次に、各要素の各要素辺について、以下の処理を行う。要素辺テーブルの中から、これと同一の要素辺がすでに登録されているかどうかを検索する。もし同じものが見つければ、要素からこの要素辺への参照関係を記録する。一方、同じものが見つからなければ、要素辺を新たに生成し要素辺テーブルに登録する。そして、要素からこの新しく生成された要素辺への参照関係を記録する。本処理の終了後、要素辺テーブルにはこのメッシュの全要素辺が重複無しに蓄えられることになる。要素辺テーブルの検索にはハッシュ法を用いる。したがって、本アルゴリズムではその計算時間は要素数に比例することになる。

なお、要素辺抽出操作を行うためのもう一つの方法として、ソートとそれに続く重複アイテムのふるい落としによるものがある。ちょうど、UNIX コマンドの `sort` と `uniq` によるものにあたる。この方法では、`sort` が $N \log N$ の計算時間を必要とするため前者に比べ多くの実装で遅くなるようである。また、重複するものも含めてすべての要素辺をメモリ上に集める必要があるため、ハッシュ検索に比べてより多くのメモリを消費する傾向がある。ただし、実際のケースにおける性能比較については、おのおの実装詳細と計算機ハードウェアの特性、および解析規模に依存するファクターが大きい。

7.6 要素面の抽出

要素面の抽出操作では、三次元ボリュームメッシュ内のすべての要素面を抽出し、これらと要素との参照関係を明らかにする。三次元ボリューム (ソリッド) 要素は、要素ごとに何枚かの面を有する。例えば、四面体要素は 4 枚の要素面を持つ。一方、各要素面は、これをはさむ 2 個の要素から共有される。ただし、隣接する要素が 1 個だけしかなければ、この要素面はメッシュの境界表面に存在する。あるメッシュ内に存在するすべての要素面を重複無く数え上げ、また、これらと要素との参照関係を明らかにするのが、要素面抽出操作である。

要素面抽出操作は、主に、メッシュの境界表面を取り出したり、要素間の隣接関係を求めたい場合に必要とされる。ここで求められたメッシュ境界表面はさらに、メッシュおよび表面物理量の可視化や境界条件設定の目的で用いることができる。また、要素間の隣接関係については、例えば、流体解析などでパーティクルトレースを高速に生成するために用いることができる。

要素面抽出操作を行うアルゴリズムとして、ここではハッシュ検索を用いた方法を採用する。まず、要素面のテーブルを初期状態空として作成する。次に、各要素の各要素面について、以下の処理を行う。要素面テーブルの中から、これと同一の要素面がすでに登録されているかどうかを検索する。もし同じものが見つければ、要素からこの要素面への参照関係を記録する。一方、同じものが見つからなければ、要素面を新たに生成し要素面テーブルに登録する。そして、要素からこの新しく生成された要素面への参照関係を記録する。本処理の終了後、要素面テーブルにはこのメッシュ内の全要素面が重複無しに蓄えられることになる。要素面テーブルの検索にはハッシュ法を用いる。したがって、本アルゴリズムではその計算時間は要素数に比例することになる。

7.7 境界表面の抽出

境界表面の抽出操作では、三次元ボリュームメッシュの境界表面を抽出する。ここでいうメッシュの境界表面とは、三次元ボリューム (ソリッド) 要素で近似された解析領域について、その境界表面上に存在するすべての要素面を集めたものを意味する。四面体要素の場合には、これはいわゆる三角形表面パッチに相当する。

メッシュの境界表面を抽出することにより、モデル表面物理量の可視化および境界条件の編集における対話性能を向上させることができる。

境界表面上の要素面抽出操作を行うアルゴリズムとして、ここではハッシュ検索を用いた方法を採用する。まず、要素面のテーブルを初期状態空として作成する。次に、各要素の各要素面について、以下の処理を行う。要素面テーブルの中から、これと同一の要素面がすでに登録されているかどうかを検索する。もし同じものが見つければ、それを要素面テーブルから削除する。一方、同じものが見つからなければ、要素面を新たに生成し要素面テーブルに登録する。本処理の終了後、要素面テーブルにはこのメッシュ内の境界表面上に存在する要素面だけが蓄えられることになる。要素面テーブルの検索にはハッシュ法を用いる。したがって、本アルゴリズムではその計算時間は要素数に比例することになる。

これは、前節での要素面の抽出とほぼ同じアルゴリズムであるが、大きく異なるのは、二つ目の要素面が見つかった場合にその二つともテーブルから削除することである。この場合、境界表面上にある要素面だけしかテーブルに登録されないため、すべての要素面を抽出するケースに比べ、より少ないメモリで、しかもより短時間でデータ処理を行うことが出来る。したがって、メモリ使用量に制約がある場合や、境界表面データしか必要とされない場合には、前節の全要素面の抽出よりもこちらを用いるべきである。

7.8 境界表面のグループ化

有限要素解析のためのプレポスト処理では、メッシュ境界表面上のさまざまな属性や物理量に対する編集操作が行われる。特に、境界条件の設定および可視化操作においては、メッシュ境界表面上の個々の要素面や節点に対しての設定が必要とされる。しかし、大規模解析の場合、選択すべき要素面や節点は膨大なので、これらのある程度の扱いやすい塊ごとにグループ化したものがあれば便利である。解析モデルの多くは CAD やモデラーからの形状情報を元に生成されているので、解析モデルのもともとの幾何形状の頂点、稜線や表面を復元するような形でグループ化されていると、多くの場合都合が良い。

ここでは、抽出されたメッシュ境界表面 (表面パッチ) の情報を用いてグループ化を行う。基本的には、表面パッチを構成する各表面三角形要素についてその隣接関係を調べ、このうち比較的滑らかに接続する表面三角形要素同士をグループ化すると、もともとの幾何形状の表面とほぼ同じものが得られる。そして、これらの幾何形状表面間のちょうど境界が幾何形状の稜線に相当し、そこからさらに幾何形状の頂点が認識される。隣り合う要素面同士が滑らかに接続するかどうかの判定 (二面挟角判定) は、ユーザーが指定する二面挟角のトレランス角度により行う。なお、ここで認識されたそれぞれの幾何形状表面を、ここでは表面パッチの面グループと呼ぶ。

具体的に境界表面のグループ化を行う手順としては、

1. メッシュ境界表面情報からシェル要素のメッシュとしての表面三角形メッシュ (表面パッチ) を構成する。
2. 表面メッシュについて、全ての要素辺を抽出する。
3. 表面メッシュの要素間隣接関係を求める。
4. 表面要素隣接関係と指定された二面挟角の角度を用いて、それぞれの表面要素辺が幾何形状稜線を構成するかどうかを調べる。

5. 表面要素のグループ化を行い、各幾何形状表面 (面グループ) とそれらを構成する表面要素を認識する。このとき、幾何形状稜線を構成する表面要素辺が隣り合う幾何形状表面間の境界となるようにする。
6. 隣り合う 2 つの幾何形状表面 (面グループ) 間の境界として、各幾何形状稜線とそれらを構成する表面要素辺を認識する。
7. 複数の幾何形状稜線がマージする接合点として、各幾何形状頂点とそれらに相当する表面節点を認識する。

7.9 任意断面の抽出

任意断面の抽出は、三次元ボリューム領域に分布するスカラー量の可視化手段としての、断面スカラーコンター図の作成に必要となる。

任意断面の抽出では、まず、任意断面とメッシュの各要素との交差を調べ、もしその要素が断面と交差するならば、同時にその交差面における物理量を抽出する。なお、ここでの任意断面は、平面によるものに限定する。ただし、平面は任意の点を含み、任意の法線方向を向くものとする。

任意断面の平面と四面体要素との交差面は、通常は三角形または四辺形のポリゴンである。ここでは、三角形の場合はそのまま、四辺形の場合にはこれを二つの三角形として保持する。これら断面三角形の各頂点は、生成元の四面体要素の要素辺上に存在する。ある断面三角形の各頂点での物理量は、この要素辺上の節点群、特に、四面体一次要素の場合は、端点上の 2 節点での物理量から内分比を用いて補間すれば良い。したがって、断面三角形の表現としては、断面三角形の各頂点ごとに、その頂点座標の他に、対応する上記 2 節点への参照と、要素辺上での内分比を記憶している。

続いて、任意断面の平面とある四面体要素との交差面を求める方法を説明する。まず、任意断面についてそれを表現する平面方程式、例えば、 $f(x, y, z) = ax + by + cz + d = 0$ が与えられるものとする。交差面上の任意の点では、その点の座標値 x 、 y 、 z をこの平面方程式に代入すれば、 $f = 0$ になる必要がある。よって、この交差面を、平面方程式のスカラー場 $f(x, y, z)$ に関する、値 0 の等値面と見なし、等値面生成を行うことにより、交差面の形状を求めることができる。なお、等値面生成のアルゴリズムとしては、マーチングキューブ法の四面体版を用いている。マーチングキューブ法の詳細については、次節を参照。

等値面生成では、四面体要素の各節点上でその座標値に対する平面方程式の $f(x, y, z)$ の値を求める必要がある。これをすべての四面体のすべての節点で行うのは計算量として無駄が多い。これに対して、記憶容量をいくらか犠牲にすることで、ある程度的高速化が達成できる。つまり、まずあらかじめメッシュの各節点ごとに平面方程式の値を評価しておき、各要素の等値面を調べる段階で再度これを用いる。

次に、ユーザーが適当な断面を切ったあとで、今度はこの断面を平行にスライドしていくことを考える。新しい断面の平面方程式では、その法線方向が前のものと同じになる。つまり、平面方程式 $ax + by + cz + d = 0$ について、 a 、 b 、 c が同じで d だけが異なるものになる。したがって、今度は切り出す等値面の値を 0 とせずに、 d が変化した分のオフセット値に対応する値で等値面を切り出せばよい。このとき、最初の断面抽出時に計算しておいたメッシュの各節点での平面方程式値を再利用できる。また、次節で述べる、一次元バケットを用いた等値面的高速生成アルゴリズムを用いることができる。本アルゴリズムを用いることにより、断面生成における計算時間は、メッシュの全要素数ではなく、実際に断面と交差するかそれに近い要素の数に比例する。これにより、通常、全要素を探索する場合に比べ、数十倍からそれ以上の高速化を達成することが出来る。結果として、一億自由度の構造問題や一千万自由度の流体問題について、断面をほぼリアルタイムにスライドしていくことが可能である。

7.10 等値面の抽出

等値面の抽出は、三次元ボリューム領域に分布するスカラー量の可視化手段としての、等値面図の作成に必要となる。

等値面の抽出では、スカラー場についてユーザーが指定する値 (等値面の値) に対応する等値面を構成する、すべてのポリゴン群を求める。このとき、スカラー場自体はメッシュの各節点ごとのスカラー値として表現されているものとする。

等値面の可視化操作では、ユーザーは等値面の値をすこしずつ増減させながら次々と新しい等値面形状を表示していくことで、領域内に分布する対象スカラー場の性質を把握することができる。このとき、等値面の抽出および描画速度はリアルタイムレベルであることが望ましい。

四面体要素の場合、等値面生成により各要素ごとに三角形または四辺形が生成される。ただし、一般のスカラー場による等値面の場合、後者はその頂点が同一平面上にある保証はない。そのため、もし四辺形が生成される場合には、これを二つの三角形に分割する。したがって、各四面体ごとに、もしこの四面体が等値面を含む場合には、等値面を構成する等値面三角形が一つまたは 2 つ生成される。

ある四面体について、ここから等値面を切り出す方法としては、代表的なアルゴリズムである、マーチングキューブ法 (その四面体版) を用いる。まず、四面体の各頂点のスカラー値について、これからユーザー指定の等値面の値を引いた値を求め、この符号を調べる。ここで、頂点ごとの値の符号の正負を分類し、分類結果から等値面パターン表を参照することにより、等値面の有無とその等値面切り出しパターンを参照する。ここで、等値面パターン表とは、四面体の頂点ごとの符号の並びを整数コード化したものと、等値面を切り出すパターンとの対応関係をテーブル化したものである。また、個々の等値面の切り出しパターンでは、等値面ポリゴンが存在するのか、もしあればそれは三角形か四辺形か、さらに、その三角形または四辺形の各頂点は四面体のどの辺上に存在するかといった情報が扱われる。パターンが分かれば、辺の 2 端点頂点における上記の値の絶対値より内分比を求め、結果として、抽出された等値面三角形の各頂点の座標が求まる。なお、内分比は、前節で触れた任意断面または等値面上において、他の任意の物理量をマッピングする際に用いることもできる。

次に、一次元バケットを用いて等値面の抽出を高速化する方法について説明する。

準備段階として、まず、ユーザーが選択したスカラー場について、各節点ごとにそのスカラー量がメモリ上に保存されているものとする。ここで、スカラー量の最大最小レンジに関して一次元バケットを構築する。さらに、各要素について、その要素を構成する節点スカラーの最大最小を求め、一次元バケットの要素内最大最小レンジの部分にこの要素を登録する。

次に、等値面の値が指定された後では、この一次元バケットを用いて指定の等値面の値に対応するバケットを求め、このバケットに所属する要素群だけを検索対象とする。

このように、一次元バケットを用いることにより、全要素の中から等値面を含んでいるような要素だけに検索対象を絞り込むことができる。通常は、全要素を検索する場合に比べ、数十倍からそれ以上の高速化を達成することが出来る。結果として、一億自由度の構造問題や一千万自由度の流体問題について、等値面の値をほぼリアルタイムに変更していくことが可能である。

7.11 矢線の抽出

矢線の抽出は、三次元ボリューム領域に分布するベクトル量の可視化手段としての、任意断面上での矢線図の作成に必要となる。

一般に、ベクトル場の矢線表示はユーザー指定の任意断面上で行われる。表示される各矢線は基本的にはなんらかの矢印である。各矢印はその矢印の原点位置でのベクトル値と同じ方向を向き、またその長さはそのベクトル値のノルムについて、ユーザー指定の均一倍率でこれをスケールしたものである。

一方、矢線図においてベクトル値の方向や値を分かりやすく表示するためには、その断面上において矢線の原点を格子状に配置したほうが望ましい。格子の細かさ、すなわち格子間隔または格子分割数についてはユーザーが指定する。また、断面上での格子の向きについては、グローバル座標系を断面に投影することで自動計算するか、またはユーザーが指定する。

矢線図での各格子点は必ずしもメッシュの節点上や要素積分点上にあるとは限らない。したがって、各格子点について、それがどれかの要素に含まれているかどうかをチェックし、そうであればその位置でのベクトル量を補間して求める必要がある。

次に、矢線の原点でのベクトル量を求める方法を説明する。各格子点についてループを回す。まず、その格子点を含む要素を探索する。もしあれば、そこから矢線の矢印を描くことができる。そうであれば、次に、その要素に関する格子点位置の要素局所座標を求める。さらに、その要素局所座標を用いて要素内でベクトル量を有限要素補間する。なお、ある要素が特定の点を含むかどうかをチェックするためには、要素に関するその点の要素局所座標を求め、その各成分が正規の値域にあるかどうかを調べればよい。この結果、要素局所座標も自動的に求まる。ここで、四面体要素の場合、要素局所座標とは体積座標のことである。

この手順での性能上のボトルネックは、格子点座標を含む要素の検索である。これには、三次元バケットが有効である。まず、メッシュを覆う三次元バケットを生成する。そして各要素を、それぞれを含むバウンディングボックスを用いて三次元バケットに登録しておく。次に、格子点座標が与えられたとき、その点の座標でのバケットを取り出し、そこに登録された要素群について、上記のようなより詳細な包含チェックを行う。通常は、三次元バケットを用いることにより、メッシュの規模にかかわらずリアルタイムでの矢線表示が可能である。問題は、一般のクライアント端末においてこのバケットがオンメモリで構築できるかどうかであるが、一億自由度の構造問題や一千万自由度の流体問題についてはそれが可能である。

7.12 パーティクルトレースの抽出

パーティクルトレースの抽出は、二次元または三次元ボリューム領域に分布するベクトル量の可視化手段としての、パーティクルトレース図の作成に必要となる。

一つのパーティクルトレースは、その始点に置かれた質量ゼロの粒子（パーティクル）が、ここで流れ場と見なされたベクトル場に従って流されていく軌跡（トレース）、すなわち流れ場に沿った粒子の軌跡である。通常、一つまたは複数の粒子が、それぞれユーザーの指定した始点位置に配置される。複数の粒子を扱う場合には、これらを格子状に配置したほうが分かりやすい。次に、各粒子ごとに、その始点位置、指定開始時刻からの軌跡を計算する。なお、時間ステップ間隔はユーザーが与えるか、あるいは自動でも求まる。

各粒子の軌跡を求めるには、基本的には粒子の始点から始めて対象としているベクトル場を時間方向に線積分していけばよい。実際には、これは適当な時間ステップを切って数値的に求められる。開始時間から始めて各時間ステップごとに、粒子が現在いる場所でのベクトル場の値を求め、これを次のステップでのその粒子の流速とする。これがある短い時間、時間ステップ間隔だけ移動したとすれば、単純に考えれば、流速の方向に、流速ノルムと時間ステップ間隔の積にあたる分の長さだけ、粒子は移動する。

上記のアルゴリズムは、時間積分でいえばもっとも単純な、前進オイラー法に相当する。一般には、軌跡計算の精度を向上するために、ルンゲクッタ法などのより高精度な時間積分法が、アダプティブな時間ステップ間隔制御とともに用いられる。なお、積分精度が悪い場合の弊害としては、流した粒子の一部が壁付近に到達したとき、流速固定境界条件をほどこされた壁にトラップされてそれ以上動かなくなったり、あるいは壁を越えて解析領域外部にはみ出してしまふことがある。

粒子の軌跡計算にどのような方法が用いられようとも、必要となるのは各タイムステップでの粒子の位置におけるベクトル値である。一般に各粒子の位置はボリュー

ム内を節点や要素積分点とは無関係に動き回るので、ボリウム内の任意座標でのベクトル場の補間評価が必要である。これは、ちょうど前節の矢線抽出の場合と同じである。

任意座標でのベクトル値の評価には、矢線抽出の場合と同様に、三次元バケットを用いることができる。少なくとも、粒子が置かれた始点における、最初の時間ステップでの計算ではこれを用いればよい。

一方、続く各時間ステップにおける計算では、単純に三次元バケットを用いるだけのものよりも効率的な方法が存在する。通常、ステップ間では粒子はそれほど長い距離を移動せず、その前のステップにいた要素に留まるか、その隣接要素にいる可能性が高い。したがって、三次元バケットを検索する前に、まず粒子が前ステップにいた要素内に留まっているかどうかをチェックし、続いて要素面を挟んだ4つの隣接要素をチェックする。これらになければ、検索した隣接要素のさらに隣接要素を、再帰的にチェックしていくこともできる。ここでは、二段階または三段階の隣接要素チェックを行い、それで見つからなかった場合にのみ三次元バケットを検索することにする。なお、このとき、前節の要素面抽出において求められた要素隣接関係情報が必要となる。これらの方法により、三次元バケットや要素隣接関係情報をメモリ上に置いておくことさえ可能であれば、通常はメッシュの規模にかかわらずリアルタイムでのパーティクルトレース表示が可能である。

ところで、パーティクルトレースの抽出では、対象とするベクトル場が定常場か、それとも非定常場であるかによって考慮が必要である。解析結果が定常問題の解であれば、上の方法で十分である。一步、非定常場の場合、パーティクルトレース計算における時間進行を実際の流れ場の変化時間と対応づけるか、それとも非定常流れ場のある1つの時間ステップデータを対象に、これを定常場と見なしてパーティクルトレースを計算するかの違いがある。厳密には、前者は流跡線(ストリークライン)、後者は流線(ストリームライン)と呼ばれ、それぞれ区別されている。後者は特定のタイムステップにおける流れ場の分布を理解するのには便利であるが、一般には前者で計算される実際の軌跡とは異なるため、注意が必要である。

なお、前者の計算にはベクトル値の時間履歴が必要となる。これはアニメーション表示の状況と同様に、膨大な解析結果データの計算サーバからクライアント端末へのデータ転送と可視化アプリケーションへの入力を伴うことになる。

さらに、通常は解析結果ファイルには時間ステップが間引きされて出力されているので、パーティクルトレース計算の精度はある程度劣化している。パーティクルトレースを時間方向により精密に求めたい場合には、解析コードの内部で扱われているベクトル値を用いて、有限要素解析で本来用いられている時間ステップごとに、パーティクルトレースを計算する必要がある。これは、少なくともパーティクルトレースの抽出計算部分については、計算サーバ上で、しかも解析コード内部で計算することの必要性を意味する。このとき、各粒子をどのタイミングでどこに出現させるかについては、有限要素解析を始める前にあらかじめ決めておかなければならない。

第8章 データ抽出処理のパフォーマンス設計

大規模有限要素モデルのプレポスト処理を実現するためには、そこで使用されるデータ処理アルゴリズムの計算量が節点数や要素数に対して $O(N)$ あるいは $O(N \log N)$ の特性を持っている必要がある。まずこれを前提とした上で、さらにデータ処理がオンメモリで行なわれることが望ましい。そのため、各データ処理に必要とされるメモリ量をあらかじめ検討しておかなければならない。

以下では、三次元ボリウム (ソリッド) 要素を用いた有限要素モデルの可視化・選択編集において必要とされるデータ処理について、その使用メモリ量や演算回数、データ入出力量などを算出する。

要素の種類については、四面体の一次および二次要素双方を考慮する。流体解析では、一次要素が用いられることが多い。また、構造解析のように解析自体は四面体二次要素を用いて行われたとしても、そのプレポスト処理の各段階において一次要素が用いられることも多いので、このようにした。

8.1 性能見積りの前提条件

まず、基本データ単位を定義する。必要とされるメモリ容量を計算する際には、整数値について 32 bit (4 byte) 符号付きの整数表現を用い、一方、実数値について 64 bit (8 byte) IEEE 倍精度浮動小数点表現を用いる。

ただし、レンダリングやピック処理を目的にした場合に限れば、例えば、4 byte 整数を 1 byte にする、倍精度を単精度にする、浮動小数点数をよりサイズの少ない整数で近似するなど、データ量を削減するために実装時に精度を描画に必要な程度まで落とすことは場合によっては十分ありうる。

また、メモリ上への操作をディスク上への操作に置き換えることにより、メモリ使用量をさらに削減することが可能な場合もある。

一方、整数を 64 bit 表現、8 byte にするかどうかの問題がある。対象としている数千万から数億自由度までの問題については、少なくともプレポスト処理に関しては整数を 32 bit で扱っても特に問題はないようである。また、主にクライアント端末上で現在広く普及している 32 ビット OS 上でプレポスト処理を実行したいということもある。同様な理由より、4 倍精度の浮動小数点数 (128 bit、16 byte) は用いないものとする。

今回は基本的にデータの利用目的を問わない形での見積りを行いたいため、整数は 4 byte、実数は 8 byte とする。また、もし、可視化に特化するといったような例外があれば、そこではそれを明記することにする。

8.2 有限要素モデル

有限要素法で用いられる非構造格子メッシュは、節点座標、すなわち、各節点ごとの空間座標データと、要素コネクティビティ、すなわち、各要素ごとの節点番号リス

トより表現されているものとする。

三次元ボリウム (ソリッド) 要素の場合、節点座標データでは、各節点ごとに x 、 y 、 z の 3 成分が必要とされる。また、四面体要素の場合、要素コネクティビティデータでは、各要素ごとに 4 個 (一次要素) または 10 個 (二次要素) の節点番号が必要とされる。まとめると、個々のデータ単位ごとに必要とされるメモリ量は以下のようになる。

- 節点座標
 x 、 y 、 z の 3 成分をそれぞれ実数値として、 $8byte \times 3 = 24byte$
- 要素コネクティビティ (一次要素)
節点番号を整数値として、 $4byte \times 4 = 16byte$
- 要素コネクティビティ (二次要素)
節点番号を整数値として、 $4byte \times 10 = 40byte$

一方、解析結果については、時間ステップ、固有モード、増分ステップ、反復ステップなどの各種ステップあるいはデータセットごとについて考慮する。このとき、ステップデータ内の個々のデータ単位、すなわち、スカラー値、ベクトル値およびテンソル値 (対称テンソルとする) について、

- スカラー
実数値として、8 byte
- ベクトル
各成分をそれぞれ実数値として、 $8byte \times 3 = 24byte$
- テンソル
各成分をそれぞれ実数値として、 $8byte \times 6 = 48byte$

これらのデータ単位は、基本的に節点上、要素上あるいは要素積分点上に配置されるものとする。

まず、要素積分点については、四面体一次要素、二次要素ごとにそれぞれ以下のものとする。

- 一次要素
要素積分点数は 1 個 (要素中心) とする。
- 二次要素
要素積分点数は 4 個とする。

次に、節点数と要素数との関係については、以下の比率をとるものとする。前提として、一次節点数 N_{1st} を四面体一次要素のメッシュにおける全節点数、二次節点数 N_{2nd} を四面体二次要素のメッシュにおける全節点数、とする。ここでは、一次節点は要素頂点上の節点であり、一方、二次節点は要素頂点および稜線 (中間節点) 上の節点を合わせたものとなる。したがって、中間節点数は、 $N_{2nd} - N_{1st}$ となる。なお、四面体要素の場合、要素数 N_{el} が同じならば、二次節点数 N_{2nd} は一次節点数 N_{1st} のちょうど 8 倍である。

$$N_{2nd} = 8N_{1st} \quad (8.1)$$

さらに、一次節点数 N_{1st} と要素数 N_{el} の比率を R_{el} とおけば、

$$N_{el} = R_{el}N_{1st} \quad (8.2)$$

$$N_{el} = 1/8R_{el}N_{2nd} \quad (8.3)$$

以降の議論では、仮にこれを、 $R_{el} = 5.6$ とする。これは経験値であり、実際の値は個々のメッシュの性質ごとに 2、3 割程度の範囲で変動する。バルク形状であれば、だいたいこのぐらいの値で安定し、一方、薄肉形状では、より小さな値をとる傾向がある。

すると、

- 一次要素
要素数 N_{el} は一次節点数 N_{1st} の 5.6 倍程度とする。

$$N_{el} = 5.6N_{1st} \quad (8.4)$$

- 二次要素
要素数 N_{el} は二次節点数 N_{2nd} の 0.7 倍程度とする。

$$N_{el} = 0.7N_{2nd} \quad (8.5)$$

8.2.1 四面体一次要素

ここでは、四面体一次要素を用いるものとする。四面体一次要素では、積分点は要素中心に一つだけ存在するため、要素積分点ごとのデータと要素ごとのデータは同じものとして扱う。

メッシュや解析結果を記憶するのに必要なメモリ量は以下ようになる。

- 節点座標
 $24byte \times N_{1st} = 24N_{1st}byte$
- 要素コネクティビティ
 $16byte \times N_{el} = 16N_{el}byte$
- 節点スカラー
 $8byte \times N_{1st} = 8N_{1st}byte$
- 節点ベクトル
 $24byte \times N_{1st} = 24N_{1st}byte$
- 節点テンソル
 $48byte \times N_{1st} = 48N_{1st}byte$
- 要素スカラー
 $8byte \times N_{el} = 8N_{el}byte$
- 要素ベクトル
 $24byte \times N_{el} = 24N_{el}byte$
- 要素テンソル
 $48byte \times N_{el} = 48N_{el}byte$

8.2.2 流体解析

例えば、流体解析で一千万自由度 (10M DOFs) のモデルでは、各節点は流速 u 、 v 、 w および圧力 p の 4 自由度を持つので、節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = R_{el} \times 2.5M = 14M$ となる。したがって、以下のようなメモリがそれぞれ必要となる。

- 節点座標
 $24byte \times 2.5M = 60MB$
- 要素コネクティビティ
 $16byte \times 14M = 220MB$
- 節点スカラー
 $8byte \times 2.5M = 20MB$
- 節点ベクトル
 $24byte \times 2.5M = 60MB$

- 節点テンソル
 $48\text{byte} \times 2.5M = 120MB$
- 要素スカラー
 $8\text{byte} \times 14M = 110MB$
- 要素ベクトル
 $24\text{byte} \times 14M = 340MB$
- 要素テンソル
 $48\text{byte} \times 14M = 670MB$

8.2.3 四面体二次要素

ここでは、四面体二次要素を用いるものとする。
メッシュや解析結果を記憶するのに必要なメモリ量は以下ようになる。

- 節点座標
 $24\text{byte} \times N_{2nd} = 24N_{2nd}\text{byte}$
- 要素コネクティビティ
 $40\text{byte} \times N_{el} = 40N_{el}\text{byte}$
- 節点スカラー
 $8\text{byte} \times N_{2nd} = 8N_{2nd}\text{byte}$
- 節点ベクトル
 $24\text{byte} \times N_{2nd} = 24N_{2nd}\text{byte}$
- 節点テンソル
 $48\text{byte} \times N_{2nd} = 48N_{2nd}\text{byte}$
- 要素スカラー
 $8\text{byte} \times N_{el} = 8N_{el}\text{byte}$
- 要素ベクトル
 $24\text{byte} \times N_{el} = 24N_{el}\text{byte}$
- 要素テンソル
 $48\text{byte} \times N_{el} = 48N_{el}\text{byte}$
- 要素積分点スカラー
 $8\text{byte} \times 4 \times N_{el} = 32N_{el}\text{byte}$
- 要素積分点ベクトル
 $24\text{byte} \times 4 \times N_{el} = 96N_{el}\text{byte}$
- 要素積分点テンソル
 $48\text{byte} \times 4 \times N_{el} = 192N_{el}\text{byte}$

8.2.4 構造解析

例えば、構造解析で一億自由度 (100M DOFs) のモデルでは、各節点は変位 u 、 v 、 w の 3 自由度を持つので、節点数は、 $N_{2nd} = 33M$ 、要素数は、 $N_{el} = 1/8 R_{el} \times 33M = 23M$ となる。したがって、以下のようなメモリがそれぞれ必要となる。

- 二次節点座標
 $24\text{byte} \times 33M = 800MB$
- 要素コネクティビティ
 $40\text{byte} \times 23M = 930MB$

- 節点スカラー
 $8\text{byte} \times 33M = 270MB$
- 節点ベクトル
 $24\text{byte} \times 33M = 800MB$
- 節点テンソル
 $48\text{byte} \times 33M = 1.6GB$
- 要素スカラー
 $8\text{byte} \times 23M = 180MB$
- 要素ベクトル
 $24\text{byte} \times 23M = 560MB$
- 要素テンソル
 $48\text{byte} \times 23M = 1.1GB$
- 要素積分点スカラー
 $32\text{byte} \times 23M = 750MB$
- 要素積分点ベクトル
 $96\text{byte} \times 23M = 2.2GB$
- 要素積分点テンソル
 $192\text{byte} \times 23M = 4.4GB$

8.3 要素から節点への振り分け

原則として、もし、ある処理が以後の作業で必要となることがあらかじめ分かっているとしたら、データアクセスの効率化の観点からは、なるべくそれを先に行っておくべきである。ただし、その後ユーザーの気が変わってそれ以外のことも必要となるかも知れないため、自由度や柔軟性を保つという観点からは議論の余地がある。

要素から節点への振り分けにおけるパフォーマンス設計について言えることは、

- 要素積分点上の出力だけでなく、節点ごとの出力を別に用意しておいたほうがいい。
- もし、あらかじめ検討したいスカラー値やベクトル値の種類が分かっている場合には、これを別に評価しておいたほうがよい。構造解析の例では、例えば、相当応力、相当歪みや主応力ベクトルなどである。
- テンソルやベクトルの個々の成分が必要ならば、これを分離しておいたほうがよい。たとえば、各成分ごとに分離してそれぞれ別のスカラー値ファイルにしておく。あるいは、バイナリファイルで出力し、そのとき同一成分、例えば x 、 y 、 xx 、 xy 成分等をそれぞれファイルの 1 セクションにまとめることで、あとでそのセクションだけにランダムアクセスできるようにする。
- データが計算サーバ上にあるならば、データ抽出操作も計算サーバ上でおこなったほうがよい。その後、すでに計算サーバの HDD 上に出力されている抽出結果ファイルをクライアント端末にネットワーク転送する。
- 以上の処理をあらかじめメインの解析ソルバー側で行えるならば、そのほうが望ましい。これは、解析コードの入力ファイルで解析結果の出力指定をすることに対応するため、比較的常識的な考え方である。

したがって、ガイドラインとしては、もし大量のデータ出力をしばらくの間保持することが可能であるならば、まず、要素積分点ごとのデータを出力し、詳細なデータ解析やより広範な用途に備えてこれらを保存しておく。また、それとは別に、頻繁に必要なようなスカラーやベクトルは別途解析ソルバーから節点上の値として出力しておく。

8.3.1 四面体一次要素

ここでは、四面体一次要素を用いるものとする。

まず、このデータ抽出操作によるデータ量の変化を算定する。前節での結果より、もし、要素ごとのデータを節点へ移せば、節点数 N_{1st} は要素数 N_{el} よりも少ないので、データ量は 0.18 倍に圧縮される。

次に、データアクセス量については、

- 要素スカラーを節点へ
読み込み $8N_{el}byte$ 、書き出し $8N_{1st}byte$ で、計 $53N_{1st}byte$ 。
- 要素ベクトルを節点へ
読み込み $24N_{el}byte$ 、書き出し $24N_{1st}byte$ で、計 $160N_{1st}byte$ 。
- 要素テンソルを節点へ
読み込み $48N_{el}byte$ 、書き出し $48N_{1st}byte$ で、計 $320N_{1st}byte$ 。

8.3.2 流体解析

より具体的に、一千万自由度 (10M DOFs) の流体解析モデルを例に、HDD 上のデータを対象として要素から節点へデータを移す場合を考える。節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = 14M$ である。このとき、各処理は以下ようになる。

- 要素スカラーを節点へ
 $53byte \times 2.5M = 130MB$
- 要素ベクトルを節点へ
 $160byte \times 2.5M = 400MB$
- 要素テンソルを節点へ
 $320byte \times 2.5M = 800GB$

ここで、仮に HDD へのアクセス速度を 50MB/s とすると、HDD へのデータアクセス時間は、

- 要素スカラーを節点へ
 $130MB/50MB/s = 2.6sec.$
- 要素ベクトルを節点へ
 $400MB/50MB/s = 8.0sec.$
- 要素テンソルを節点へ
 $800MB/50MB/s = 16sec.$

8.3.3 四面体二次要素

ここでは、四面体二次要素を用いるものとする。

まず、このデータ抽出操作によるデータ量の変化を算定する。前節での結果より、もし、要素ごとのデータを節点へ移せば、節点数 N_{2nd} は要素数 N_{el} より若干多いので、データ量は 1.4 倍ほど余計に必要となる。一方、要素積分点ごとのデータを節点へ移せば、逆にデータ量は 0.35 倍に圧縮される。

次に、データアクセス量については、

- 要素スカラーを節点へ
読み込み $8N_{el}byte$ 、書き出し $8N_{2nd}byte$ で、計 $14N_{2nd}byte$ 。
- 要素ベクトルを節点へ
読み込み $24N_{el}byte$ 、書き出し $24N_{2nd}byte$ で、計 $41N_{2nd}byte$ 。

- 要素テンソルを節点へ
読み込み $48N_{el}byte$ 、書き出し $48N_{2nd}byte$ で、計 $82N_{2nd}byte$ 。
- 要素積分点スカラーを節点へ
読み込み $32N_{el}byte$ 、書き出し $8N_{2nd}byte$ で、計 $30N_{2nd}byte$ 。
- 要素積分点ベクトルを節点へ
読み込み $96N_{el}byte$ 、書き出し $24N_{2nd}byte$ で、計 $90N_{2nd}byte$ 。
- 要素積分点テンソルを節点へ
読み込み $192N_{el}byte$ 、書き出し $48N_{2nd}byte$ で、計 $180N_{2nd}byte$ 。

8.3.4 構造解析

より具体的に、一億自由度 (100M DOFs) の構造解析モデルを例に、HDD 上のデータを対象として要素積分点から節点へデータを移す場合を考える。節点数は、 $N_{2nd} = 33M$ 、要素数は、 $N_{el} = 23M$ である。このとき、各処理は以下のようになる。

- 要素積分点スカラーを節点へ
 $30byte \times 33M = 1.0GB$
- 要素積分点ベクトルを節点へ
 $90byte \times 33M = 3.0GB$
- 要素積分点テンソルを節点へ
 $180byte \times 33M = 6.0GB$

ここで、仮に HDD へのアクセス速度を $50MB/s$ とすると、HDD へのデータアクセス時間は、

- 要素積分点スカラーを節点へ
 $1.0GB/50MB/s = 20sec.$
- 要素積分点ベクトルを節点へ
 $3.0GB/50MB/s = 60sec.$
- 要素積分点テンソルを節点へ
 $6.0GB/50MB/s = 120sec.$

8.4 低次要素化

ここでは、基本的には前節の要素から節点への振り分けとまったく同じことが言える。つまり、もしあらかじめ必要だとわかっているものについては、用意しておくべきである。

特に、解析コードから一次節点での解析結果だけを直接出力できる場合には、そのほうが望ましい。節点番号の並びについて、例えば、まず前半が一次節点、後半が二次節点となっていれば、通常、解析コードの入力ファイルにおいて指定することができる、出力節点番号の範囲を前半部分だけに制限することで、一次節点上のみの出力が可能となる。

一次節点数 N_{1st} は二次節点数 N_{2nd} の $1/8$ である。したがって、必要なメモリ量は、

- 一次節点座標
 $24byte \times N_{1st} = 24N_{1st}byte$
- 一次要素コネクティビティ
 $16byte \times N_{el} = 16N_{el}byte$

- 一次節点スカラー
 $8byte \times N_{1st} = 8N_{1st}byte$
- 一次節点ベクトル
 $24byte \times N_{1st} = 24N_{1st}byte$
- 一次節点テンソル
 $48byte \times N_{1st} = 48N_{1st}byte$

データアクセス量を以下にまとめる。

- 二次節点スカラーを一次節点へ
読み込み $8N_{2nd}byte$ 、書き出し $8N_{1st}byte$ で、計 $9N_{2nd}byte$ 。
- 二次節点ベクトルを一次節点へ
読み込み $24N_{2nd}byte$ 、書き出し $24N_{1st}byte$ で、計 $27N_{2nd}byte$ 。
- 二次節点テンソルを一次節点へ
読み込み $48N_{2nd}byte$ 、書き出し $48N_{1st}byte$ で、計 $54N_{2nd}byte$ 。

8.4.1 構造解析

例えば、一億自由度 (100M DOFs) の構造解析モデルでは、二次節点数は、 $N_{2nd} = 33M$ 、一次節点数は、 $N_{1st} = 33M/8 = 4.1M$ 、一方、要素数は変わらず、 $N_{el} = 23M$ となり、

- 一次節点座標
 $24byte \times 4.1M = 100MB$
- 一次要素コネクティビティ
 $16byte \times 23M = 370MB$
- 一次節点スカラー
 $8byte \times 4.1M = 33MB$
- 一次節点ベクトル
 $24byte \times 4.1M = 100MB$
- 一次節点テンソル
 $48byte \times 4.1M = 200MB$

これに、要素から節点への振り分け処理や、よく用いられるスカラー値の評価処理などを加えることにより、大幅なデータ圧縮が可能になる。例えば、一億自由度 (100M DOFs) の構造解析モデルにおいて、

- 変位ベクトル
二次節点ごとの変位ベクトルは 800MB 必要だが、これを一次節点で評価すれば、8 分の 1 の 100MB まで縮まる。
- 温度
二次節点ごとの温度は 270MB 必要だが、これを一次節点で評価すれば、8 分の 1 の 33MB まで縮まる。
- 応力テンソル
要素積分点ごとの応力テンソルは 4.4GB 必要だが、これを一次節点で評価すれば、22 分の 1 の 200MB まで縮まる。
- 相当応力
要素積分点ごとの相当応力は 740MB 必要だが、これを一次節点で評価すれば、22 分の 1 の 33MB まで縮まる。要素積分点応力テンソルから換算すれば、130 分の 1 の圧縮率となる。

より具体的に、一億自由度 (100M DOFs) の構造解析モデルを例に、HDD 上のデータを対象として要素積分点から節点へデータを移す場合を考える。このとき、各処理は以下ようになる。

- 二次節点スカラーを一次節点へ
 $9\text{byte} \times 33M = 300MB$
- 二次節点ベクトルを一次節点へ
 $27\text{byte} \times 33M = 900MB$
- 二次節点テンソルを一次節点へ
 $54\text{byte} \times 33M = 1.8GB$

ここで、仮に HDD へのアクセス速度を 50MB/s とすると、HDD へのデータアクセス時間は、

- 要素積分点スカラーを節点へ
 $300MB / 50MB/s = 6\text{sec.}$
- 要素積分点ベクトルを節点へ
 $900MB / 50MB/s = 18\text{sec.}$
- 要素積分点テンソルを節点へ
 $1.8GB / 50MB/s = 36\text{sec.}$

8.5 要素辺の抽出

要素辺の抽出処理は、メッシュが与えられた段階で始めることが出来る。プレ処理セッションまたはポスト処理セッションの開始時に行うことが出来るし、あるいは、このデータが必要になるまで抽出処理を遅らせることもありうる。

要素辺の抽出操作に必要とされるメモリ量は以下ようになる。まず、各構成データごとのメモリは、

- 要素ごとに、要素辺への参照
 要素は 6 本の要素辺を持つ。要素辺番号を整数値として、 $4\text{byte} \times 6 = 24\text{byte}$
- 要素辺ごとに、端点上節点への参照
 要素辺は 2 つの端点を持つ。節点番号を整数値として、 $4\text{byte} \times 2 = 8\text{byte}$
- ハッシュ表データ
 ハッシュ表は各エントリからの単方向リストより構成される。平均値としては、要素辺ごとにエントリとさらにリンクが 2 つほど必要になる。合わせて、要素辺ごとに整数値が 5 つ程必要となり、 $4\text{byte} \times 5 = 20\text{byte}$

次に、メッシュ内の要素辺数 N_{ed} を求める。要素辺数は、二次要素の中間節点の数と等しい。

$$N_{ed} = N_{2nd} - N_{1st} \quad (8.6)$$

したがって、

$$N_{ed} = 1.3N_{el} \quad (8.7)$$

要素辺の抽出には、要素コネクティビティデータが必要となる。ただし、一次要素のものでよい。低次要素化されているデータがあれば、これを用いる。以下に、必要な入力データを示す。

- 一次要素コネクティビティ
 $16\text{byte} \times N_{el} = 16N_{el}\text{byte}$

次に、作業領域データについては、

- ハッシュ表データ
 $20byte \times N_{ed} = 20N_{ed}byte$

一方、出力される要素辺に関する各種データについては、以下ようになる。

- 要素から要素辺への参照
 $24byte \times N_{el} = 24N_{el}byte$
- 要素辺から端点上節点への参照
 $8byte \times N_{ed} = 8N_{ed}byte$

なお、メモリが不足する場合には、一次要素コネクティビティデータおよび要素から要素辺への参照データを記憶しないようにする。一次要素コネクティビティデータについては、要素データをファイルから順次読み込みながら処理する。このとき同時に、この要素について要素から要素辺への参照データを作成し、ファイルに順次書き込む。結果として、作業領域および要素辺から端点上節点への参照データのみをメモリ上に記憶することになり、必要なメモリ容量を半分近く減らすことができる。

8.5.1 構造解析

例えば、一億自由度 (100M DOFs) の構造解析モデルでは、二次節点数は、 $N_{2nd} = 33M$ 、一次節点数は、 $N_{1st} = 4.1M$ 、要素数は、 $N_{el} = 23M$ 、要素辺数は、 $N_{ed} = 1.3 \times 23M = 29M$ となり、要素辺抽出に必要な入力データについては、

- 一次要素コネクティビティ
 $16byte \times 23M = 370MB$

作業領域データについては、

- ハッシュ表データ
 $20byte \times 29M = 580B$

出力される要素辺に関する各種データについては、

- 要素から要素辺への参照
 $24byte \times 23M = 560MB$
- 要素辺から端点上節点への参照
 $8byte \times 29M = 230MB$

8.5.2 流体解析

続いて、一千万自由度 (10M DOFs) の流体解析モデルでは、節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = 14M$ 、要素辺数は、 $N_{ed} = 1.3 \times 14M = 18M$ となり、要素辺抽出に必要な入力データについては、

- 要素コネクティビティ
 $16byte \times 14M = 220MB$

作業領域データについては、

- ハッシュ表データ
 $20byte \times 18M = 360MB$

出力される要素辺に関する各種データについては、

- 要素から要素辺への参照
 $24byte \times 14M = 340MB$
- 要素辺から端点上節点への参照
 $8byte \times 18M = 140MB$

8.6 要素面の抽出

要素面の抽出処理は、メッシュが与えられた段階で始めることが出来る。プレ処理セッションまたはポスト処理セッションの開始時に行うことが出来るし、あるいは、このデータが必要になるまで抽出処理を遅らせることもありうる。

要素面の操作に必要とされるメモリ量は以下ようになる。まず、各構成データごとのメモリは、

- 要素ごとに、要素面への参照
要素は 4 枚の要素面を持つ。要素面番号を整数値として、 $4byte \times 4 = 16byte$
- 要素ごとに、隣接要素への参照
要素の要素面ごとに、隣接要素が対応する。ただし、要素面がメッシュの境界表面にあれば、隣接要素は存在しない。要素番号を整数値として、 $4byte \times 4 = 16byte$
- 要素面ごとに、頂点上節点への参照
要素面は 3 つの頂点を持つ。節点番号を整数値として、 $4byte \times 3 = 12byte$
- 要素面ごとに、要素への参照
裏表 2 つのサイドについて、もし隣接する要素があれば、要素番号および要素に関する局面番号をそれぞれ整数値として、 $(4byte + 4byte) \times 2 = 16byte$
- ハッシュ表データ
ハッシュ表は各エントリからの単方向リストより構成される。平均値としては、要素面ごとにエントリとさらにリンクが 2 つほど必要になる。合わせて、要素面ごとに整数値が 5 つ程必要となり、 $4byte \times 5 = 20byte$

次に、メッシュ内の要素面数 N_{fc} を求める。ここで、まず要素数 N_{el} に対する要素面数 N_{fc} の比を、 R_{fc} とすると、

$$N_{fc} = R_{fc} N_{el} \quad (8.8)$$

なお、 R_{fc} であるが、ここではこれを仮に 2.0 であるとする。これは経験値であり、実際の値は個々のメッシュの性質ごとに多少変動する。
したがって、

$$N_{fc} = 2.0 N_{el} \quad (8.9)$$

要素面の抽出には、要素コネクティビティデータが必要となる。ただし、一次要素のものでよい。低次要素化されているデータがあれば、これを用いる。以下に、必要な入力データを示す。

- 一次要素コネクティビティ
 $16byte \times N_{el} = 16N_{el}byte$

次に、作業領域データについては、

- 要素面から頂点上節点への参照
 $12byte \times N_{fc} = 12N_{fc}byte$
- ハッシュ表データ
 $20byte \times N_{fc} = 20N_{fc}byte$

一方、出力される要素面に関する各種データについては、以下ようになる。

- 要素から要素面への参照
 $16byte \times N_{el} = 16N_{el}byte$
- 要素から隣接要素への参照
 $16byte \times N_{el} = 16N_{el}byte$
- 要素面から要素への参照
 $16byte \times N_{fc} = 16N_{fc}byte$

なお、メモリが不足する場合には、これを 3 バスに分解する。

第一バスでは、作業領域のハッシュテーブルを作成しながら、要素から要素面への参照データのみを生成する。一次要素コネクティビティデータについては、要素データをファイルから順次読み込みながら処理する。このとき同時に、この要素について要素から要素面への参照データを作成し、ファイルに順次書き込む。

第二バスでは、要素から要素面への参照データを読み込み、要素面から要素への参照データを作成する。

第三バスでは、要素から要素面への参照データおよび要素面から要素への参照データを読み込み、要素から隣接要素への参照データを作成する。

この場合、第一バスにおいて記憶する作業領域のサイズが最大必要メモリ容量となり、必要なメモリ容量を半分近く減らすことができる。

8.6.1 構造解析

例えば、一億自由度 (100M DOFs) の構造解析モデルでは、二次節点数は、 $N_{2nd} = 33M$ 、一次節点数は、 $N_{1st} = 4.1M$ 、要素数は、 $N_{el} = 23M$ 、要素面は、 $N_{fc} = R_{fc} \times 23M = 46M$ となり、要素面抽出に必要な入力データについては、

- 一次要素コネクティビティ
 $16byte \times 23M = 370MB$

作業領域データについては、

- 要素面から頂点上節点への参照
 $12byte \times 46M = 560MB$
- ハッシュ表データ
 $20byte \times 46M = 930MB$

出力される要素面に関する各種データについては、

- 要素から要素面への参照
 $16byte \times 23M = 370MB$
- 要素から隣接要素への参照
 $16byte \times 23M = 370MB$
- 要素面から要素への参照
 $16byte \times 46M = 740MB$

8.6.2 流体解析

続いて、一千万自由度 (10M DOFs) の流体解析モデルでは、節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = 14M$ 、要素面数は、 $N_{fc} = R_{fc} \times 14M = 28M$ となり、要素面抽出に必要な入力データについては、

- 要素コネクティビティ
 $16byte \times 14M = 220MB$

作業領域データについては、

- 要素面から頂点上節点への参照
 $12byte \times 28M = 340MB$
- ハッシュ表データ
 $20byte \times 28M = 560MB$

出力される要素面に関する各種データについては、

- 要素から要素面への参照
 $16\text{byte} \times 14M = 220MB$
- 要素から隣接要素への参照
 $16\text{byte} \times 14M = 220MB$
- 要素面から要素への参照
 $16\text{byte} \times 28M = 450MB$

8.7 境界表面の抽出

メッシュ境界表面上の要素面、すなわち表面パッチの抽出処理は、メッシュが与えられた段階で始めることが出来る。表面パッチそのものについては、すでに境界条件の編集操作において必要となるため、プレ処理のセッション開始時において抽出処理を行う。

一方、境界表面上の物理量を可視化するニーズに備えて、解析結果から境界表面上に存在するものだけを抜きだしておくことは有益である。解析終了後、あらかじめ抽出しておいた表面パッチを用いて、解析結果全体から可視化に必要な境界表面上の物理量だけを抽出する。

境界表面の抽出操作に必要とされるメモリ量は以下ようになる。まず、各構成データごとのメモリは、

- 要素面ごとに、頂点上節点への参照
要素面は 3 つの頂点を持つ。節点番号を整数値として、 $4\text{byte} \times 3 = 12\text{byte}$
- 要素面ごとに、要素への参照
メッシュ境界表面上にある要素面は、一つの要素だけを参照する。要素番号および要素に関する局所面番号をそれぞれ整数値として、 $4\text{byte} + 4\text{byte} = 8\text{byte}$
- ハッシュ表データ
ハッシュ表は各エントリからの単方向リストより構成される。平均値としては、要素面ごとにエントリとさらにリンクが 2 つほど必要になる。また、一旦ハッシュ表に登録したアイテムを削除する必要があるので、フリーになったアイテムのための単方向リストを別に用意する。合わせて、要素面ごとに整数値が 6 つ程必要となり、 $4\text{byte} \times 6 = 24\text{byte}$

次に、メッシュ境界表面上の要素面数 (表面パッチ数) N_{pch} を求める。ここで、まず要素数 N_{el} に対する表面パッチ数 N_{pch} の比を、 R_{pch} とすると、

$$N_{pch} = R_{pch} N_{el} \quad (8.10)$$

なお、 R_{pch} であるが、基本的にこれは体積に対する表面積の比となるため、大規模になるほど小さくなる一方で、形状が複雑になることで全体的に薄く/細くなればなるほど大きくなる。

極端なケースとして、バルク形状に近いメッシュを考える。例えば、長さ方向に n 分割した立方体メッシュについては、立方体数が n^3 、メッシュ境界表面の正方形数が $6n^2$ となる。この場合、四面体要素数 $N_{el} = 6n^3$ に対して、三角形表面パッチ数は $N_{pch} = 12n^2$ となる。数千万から数億自由度のメッシュでは n は 100 以上なので、 R_{pch} は $1 / 50$ 以下となる。

もう一方の極端なケースとしては、薄板上のメッシュを考慮する。例えば、長さ n 分割の正方形メッシュを厚さ m 層にしたメッシュについては、立方体数が mn^2 、メッシュ境界表面の正方形数は側面を無視してほぼ $2n^2$ となる。この場合、四面体要素数 $N_{el} = 6mn^2$ に対して、三角形表面パッチ数は $N_{pch} = 4n^2$ となる。 R_{pch} については、 n とは無関係に、 $R_{pch} = 2/(3m)$ となる。きわめて極端な $m = 1$ の場合、要素数と表面パッチ数はほぼ同じオーダーの値となってしまう。

ここでは、 R_{pch} を仮に 0.1 であるとする。これは数千万から数億自由度の構造解析用のメッシュでは経験的に妥当な数字である。なお、流体解析や電磁場解析の場合は、よりバルク形状に近いくなるため、 R_{pch} はより小さな値になる。

したがって、

$$N_{pch} = 0.1N_{el} \quad (8.11)$$

境界表面の抽出には、要素コネクティビティデータが必要となる。ただし、一次要素のものでよい。低次要素化されているデータがあれば、これを用いる。以下に、必要な入力データを示す。

- 一次要素コネクティビティ
 $16byte \times N_{el} = 16N_{el}byte$

次に、作業領域データについては、

- 要素面から頂点上節点への参照
 $12byte \times N_{pch} = 12N_{pch}byte$
- ハッシュ表データ
 $24byte \times N_{pch} = 24N_{pch}byte$

また、表面パッチおよびその上の物理量の抽出には、同様に低次要素メッシュの節点座標および各種物理量データが必要となる。

- 一次節点座標
 $24byte \times N_{1st} = 24N_{1st}byte$
- 一次節点スカラー
 $8byte \times N_{1st} = 8N_{1st}byte$
- 一次節点ベクトル
 $24byte \times N_{1st} = 24N_{1st}byte$
- 一次節点テンソル
 $48byte \times N_{1st} = 48N_{1st}byte$

一方、境界表面抽出の出力は、基本的には三角形シェル要素のメッシュと同じデータ構造をしている。入力四面体一次要素であるため、これらの要素は 3 節点の三角形一次要素である。このとき、表面要素数は、 N_{pch} となり、また、表面節点数は、表面要素数の半分として、 $1/2N_{pch}$ となる。

出力される境界表面上の各種データについては、以下のようになる。

- 表面節点座標
 $24byte \times 1/2N_{pch} = 12N_{pch}byte$
- 表面要素コネクティビティ
 $12byte \times N_{pch} = 12N_{pch}byte$
- 表面要素から要素への参照
 $8byte \times N_{pch} = 8N_{pch}byte$
- 表面節点スカラー
 $8byte \times 1/2N_{pch} = 4N_{pch}byte$
- 表面節点ベクトル
 $24byte \times 1/2N_{pch} = 12N_{pch}byte$
- 表面節点テンソル
 $48byte \times 1/2N_{pch} = 24N_{pch}byte$

なお、メモリが不足する場合には、これを複数のパスに分割する。

まず、第一パスでは、作業領域のハッシュテーブルを作成しながら、表面要素コネクティビティ情報のみを生成する。このときにメモリにロードする必要があるデータは、作業領域および表面要素コネクティビティである。一次要素コネクティビティデータについては、要素データをファイルから順次読み込みながら処理する。

第二パスでは、表面節点から一次節点へのマッピング情報を生成する。さらにこの後、マッピング情報を用いて表面節点座標を生成する。

以降のパスでは、必要があれば、各種スカラー、ベクトル、テンソル量を同様に生成する。

8.7.1 構造解析

例えば、一億自由度 (100M DOFs) の構造解析モデルでは、二次節点数は、 $N_{2nd} = 33M$ 、一次節点数は、 $N_{1st} = 4.1M$ 、要素数は、 $N_{el} = 23M$ 、表面パッチ数は、 $N_{pch} = R_{pch} \times 23M = 2.3M$ となる。まず、境界表面抽出に必要な入力データについては、

- 一次要素コネクティビティ
 $16byte \times 23M = 370MB$

作業領域データについては、

- 要素面から頂点上節点への参照
 $12byte \times 2.3M = 28MB$
- ハッシュ表データ
 $24byte \times 2.3M = 84MB$

一方、表面パッチおよびその上の物理量の抽出に必要な入力データについては、

- 一次節点座標
 $24byte \times 4.1M = 100MB$
- 一次節点スカラー
 $8byte \times 4.1M = 33MB$
- 一次節点ベクトル
 $24byte \times 4.1M = 100MB$
- 一次節点テンソル
 $48byte \times 4.1M = 200MB$

さらに、出力される境界表面上の各種データについては、表面節点数は 1.2M、表面要素数は 2.3M となり、

- 表面節点座標
 $12byte \times 2.3M = 28MB$
- 表面要素コネクティビティ
 $12byte \times 2.3M = 28MB$
- 表面要素から要素への参照
 $8byte \times 2.3M = 19MB$
- 表面節点スカラー
 $4byte \times 2.3M = 9.2MB$
- 表面節点ベクトル
 $12byte \times 2.3M = 28MB$
- 表面節点テンソル
 $24byte \times 2.3M = 55MB$

8.7.2 流体解析

続いて、一千万自由度 (10M DOFs) の流体解析モデルでは、節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = 14M$ 、表面パッチ数は、 $N_{pch} = R_{pch} \times 14M = 1.4M$ となる。まず、境界表面抽出に必要な入力データについては、

- 要素コネクティビティ
 $16byte \times 14M = 220MB$

作業領域データについては、

- 要素面から頂点上節点への参照
 $12byte \times 1.4M = 17MB$
- ハッシュ表データ
 $24byte \times 1.4M = 34MB$

一方、表面パッチおよびその上の物理量の抽出に必要な入力データについては、

- 節点座標
 $24byte \times 2.5M = 60MB$
- 節点スカラー
 $8byte \times 2.5M = 20MB$
- 節点ベクトル
 $24byte \times 2.5M = 60MB$
- 節点テンソル
 $48byte \times 2.5M = 120MB$

さらに、出力される境界表面上の各種データについては、表面節点数は 0.70M、表面要素数は 1.4M となり、

- 表面節点座標
 $12byte \times 1.4M = 17MB$
- 表面要素コネクティビティ
 $12byte \times 1.4M = 17MB$
- 表面要素から要素への参照
 $8byte \times 1.4M = 11MB$
- 表面節点スカラー
 $4byte \times 1.4M = 5.6MB$
- 表面節点ベクトル
 $12byte \times 1.4M = 17MB$
- 表面節点テンソル
 $24byte \times 1.4M = 34MB$

8.8 任意断面の抽出

任意断面の抽出処理は、断面の位置や向きが任意なので、これらについては切り出したい断面ごとにユーザーが指定する必要がある。断面のいくつかについては、解析形状やメッシュを考慮しながら、解析直前あるいはポスト処理開始時点でユーザーがあらかじめ必要なものを指定しておくことが出来る。または、試行錯誤的にビジュアルデータマイニングを行う際に、対話セッションにおいてユーザーがその場その場で任意に指定することもありうる。

対話操作において断面を切り出す場合には、あらかじめ断面の法線方向を定めておき、断面の位置を平行にスライドしていくようなことが可能であると、対話性の向上

に役に立つ。このとき、断面を表現する平面方程式は定数係数を除いて変化しないので、これを各節点で評価したものについてメモリ上にキャッシュしておく、断面位置のみを変化させる場合の切り出し速度をある程度向上させることができる。さらに、この断面方程式の値のレンジで一次元バケットを構築し、各要素をあらかじめこの一次元バケットに登録することで、切り出し速度の著しい高速化が可能である。

一方、断面の向きを変化したい場合には、節点での断面方程式の評価値や、一次元バケットを再度作り直す必要が生じる。

任意断面の抽出操作に必要とされるメモリ量は以下ようになる。まず、各構成データごとのメモリは、

- 断面三角形ごとに、要素への参照
断面三角形は、一つの要素だけを参照する。また、断面三角形は 3 つの頂点を持つ。各頂点は、その要素に関する局所要素辺番号とその内分比より構成される。要素番号および局所要素辺番号を整数値、内分比を実数値として、 $4byte + (4byte + 8byte) \times 3 = 40byte$
- 断面三角形ごとに、頂点座標
断面三角形は 3 つの頂点を持つ。x、y、z の 3 成分をそれぞれ実数値として、 $24byte \times 3 = 72byte$
- 断面三角形ごとに、頂点スカラー
断面三角形は 3 つの頂点を持つ。スカラー量を実数値として、 $8byte \times 3 = 24byte$
- 一次節点ごとに、断面方程式の値
方程式値を実数値として、 $8byte$
- 一次元バケットデータ
一次元バケットは一次元方向に分割したバケット群および、各バケットからの単方向リストより構成される。平均値として、各要素がそれぞれ二つのバケットに登録されるものとすれば、要素ごとにリンクが 2 つ、すなわち、整数値が 4 つ程必要となり、 $4byte \times 4 = 16byte$

次に、任意断面上の断面三角形数 N_{sec} を求める。ここで、まず要素数 N_{el} に対する断面三角形数 N_{sec} の比を、 R_{sec} とすると、

$$N_{sec} = R_{sec} N_{el} \quad (8.12)$$

なお、 R_{sec} であるが、基本的にこれは前節での表面パッチ数の比率 R_{pch} と同様な傾向を示す。通常、表面パッチ数の比率 R_{pch} に対し、小さめの値となる

極端なケースとして、バルク形状に近いメッシュを考える。立方体メッシュでは、軸方向に垂直に切った断面はちょうど表面一枚分に相当するので、表面パッチ数の比率 R_{pch} の $1/6$ となる。ただし、一つの四面体要素からは通常 1 つまたは 2 つの断面三角形が生成されるので、これを $1/4$ から $1/3$ 程度に修正したほうがよい。

もう一方の極端なケースとしては、薄板上のメッシュを考慮する。もし、形状が薄肉形状でかつ平板に近い場合、断面の法線方向が厚さ方向に平行であれば、表面パッチ数の比率 R_{pch} の半分あるいは同じぐらいの値となる。

ここでは、 R_{sec} を仮に 0.05 であるとする。前節で用いた表面パッチ数の比率 R_{pch} の半分としている。これは数千万から数億自由度のメッシュでは経験的に妥当な数字である。

したがって、

$$N_{sec} = 0.05 N_{el} \quad (8.13)$$

任意断面の抽出には、要素コネクティビティデータおよび節点座標が必要となる。ただし、一次要素のものでよい。低次要素化されているデータがあれば、これを用いる。以下に、必要な入力データを示す。

- 一次節点座標
 $24byte \times N_{1st} = 24N_{1st}byte$

- 一次要素コネクティビティ
 $16\text{byte} \times N_{el} = 16N_{el}\text{byte}$

次に、作業領域データについては、

- 一次節点ごとの断面方程式値
 $8\text{byte} \times N_{1st} = 8N_{1st}\text{byte}$
- 一次元バケットデータ
 $16\text{byte} \times N_{el} = 16N_{el}\text{byte}$

また、任意断面上のスカラー量の抽出には、同様に低次要素メッシュの各種スカラーデータが必要となる。

- 一次節点スカラー
 $8\text{byte} \times N_{1st} = 8N_{1st}\text{byte}$

一方、出力される任意断面上の各種データについては、以下のようになる。

- 断面三角形の要素への参照
 $40\text{byte} \times N_{sec} = 40N_{sec}\text{byte}$
- 断面三角形の頂点座標
 $72\text{byte} \times N_{sec} = 72N_{sec}\text{byte}$
- 断面三角形の頂点スカラー
 $24\text{byte} \times N_{sec} = 24N_{sec}\text{byte}$

なお、対話セッションを保持しながらもメモリを節約したい場合には、要素コネクティビティ、節点座標および節点スカラーのみをメモリ上に保持する。このとき、実数値についてはその精度を単精度に落すこともできる。なお、節点ごとの断面方程式値と一次元バケットはその使用をあきらめる。ただし、一次元バケットを用いない場合には、全要素を探索する必要があるため、通常に比べて一桁以上の処理時間が必要になる。一方、出力される断面三角形データについては、記憶せずにそのまま描画プリミティブとして画面出力する。

さらにメモリが不足する場合には、節点座標と節点スカラーのみを記憶する。要素コネクティビティについてはこれを記憶せず、ファイルからの逐次読み出しにより要素ごとに断面三角形を生成する。ただし、大量のファイル入出力が必要となるため、対話性は損なわれる。

8.8.1 構造解析

例えば、一億自由度 (100M DOFs) の構造解析モデルでは、二次節点数は、 $N_{2nd} = 33M$ 、一次節点数は、 $N_{1st} = 4.1M$ 、要素数は、 $N_{el} = 23M$ 、断面三角形数は、 $N_{sec} = R_{sec} \times 23M = 1.2M$ となる。まず、任意断面抽出に必要な入力データについては、

- 一次節点座標
 $24\text{byte} \times 4.1M = 100MB$
- 一次要素コネクティビティ
 $16\text{byte} \times 23M = 370MB$

作業領域データについては、

- 一次節点ごとの断面方程式値
 $8\text{byte} \times 4.1M = 33MB$
- 一次元バケットデータ
 $16\text{byte} \times 23M = 370MB$

一方、任意断面上のスカラー量の抽出に必要な入力データについては、

- 一次節点スカラー
 $8\text{byte} \times 4.1M = 33MB$

さらに、出力される任意断面上の各種データについては、

- 断面三角形の要素への参照
 $40\text{byte} \times 1.2M = 48MB$
- 断面三角形の頂点座標
 $72\text{byte} \times 1.2M = 84MB$
- 断面三角形の頂点スカラー
 $24\text{byte} \times 1.2M = 28MB$

8.8.2 流体解析

続いて、一千万自由度 (10M DOFs) の流体解析モデルでは、節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = 14M$ 、断面三角形数は、 $N_{sec} = R_{sec} \times 14M = 0.70M$ となる。まず、任意断面抽出に必要な入力データについては、

- 節点座標
 $24\text{byte} \times 2.5M = 60MB$
- 要素コネクティビティ
 $16\text{byte} \times 14M = 220MB$

作業領域データについては、

- 節点ごとの断面方程式値
 $8\text{byte} \times 2.5M = 20MB$
- 一次元バケットデータ
 $16\text{byte} \times 14M = 220MB$

一方、任意断面上のスカラー量の抽出に必要な入力データについては、

- 節点スカラー
 $8\text{byte} \times 2.5M = 20MB$

さらに、出力される任意断面上の各種データについては、

- 断面三角形の要素への参照
 $40\text{byte} \times 0.70M = 28MB$
- 断面三角形の頂点座標
 $72\text{byte} \times 0.70M = 51MB$
- 断面三角形の頂点スカラー
 $24\text{byte} \times 0.70M = 17MB$

8.9 等値面の抽出

等値面の抽出処理は、任意断面の場合と同様、対象とするスカラー値の種類や等値面の値が任意なので、その生成パラメータをユーザーが指定する必要がある。そのいくつかについては、選択するスカラー値の値域や分布の性質を考慮しながら、解析直前あるいはポスト処理開始時点でユーザーがあらかじめ必要なものを指定しておくことが出来る。または、試行錯誤的にビジュアルデータマイニングを行う際に、対話セッションにおいてユーザーがその場その場で任意に指定することもありうる。

対話操作において等値面を切り出す場合には、対象となるスカラー値について、値域に関する一次元バケットを構築し、各要素をあらかじめこの一次元バケットに登録することで、抽出速度の著しい高速化が可能である。

等値面の抽出操作に必要なとされるメモリ量は以下ようになる。まず、各構成データごとのメモリは、

- 等値面三角形ごとに、頂点座標
等値面三角形は3つの頂点を持つ。x、y、zの3成分をそれぞれ実数値として、 $24\text{byte} \times 3 = 72\text{byte}$
- 等値面三角形ごとに、頂点法線方向ベクトル
等値面三角形は3つの頂点を持つ。ベクトルの各成分をそれぞれ実数値として、 $24\text{byte} \times 3 = 72\text{byte}$
- 一次節点ごとに、スカラー値の勾配ベクトル
ベクトルの各成分を実数値として、 24byte
- 一次元バケットデータ
一次元バケットは一次元方向に分割したバケット群および、各バケットからの単方向リストより構成される。平均値として、各要素がそれぞれ二つのバケットに登録されるものとすれば、要素ごとにリンクが2つ、すなわち、整数値が4つ程必要となり、 $4\text{byte} \times 4 = 16\text{byte}$

次に、等値面を構成する等値面三角形数 N_{iso} を求める。ここで、まず要素数 N_{el} に対する等値面三角形数 N_{iso} の比を、 R_{iso} とすると、

$$N_{iso} = R_{iso}N_{el} \quad (8.14)$$

なお、 R_{iso} であるが、基本的にこれは前節での断面三角形数の比率 R_{sec} と同様な傾向を示す。

ここでは、 R_{iso} を仮に 0.05 であるとする。前節で用いた断面三角形数の比率の比率 R_{sec} と同じものを用いる。これは数千万から数億自由度のメッシュでは経験的に妥当な数字である。

したがって、

$$N_{iso} = 0.05N_{el} \quad (8.15)$$

等値面の抽出には、要素コネクティビティデータ、節点座標、選択したスカラー値に関する節点スカラーと節点スカラー勾配ベクトルが必要となる。ただし、一次要素のものでよい。低次要素化されているデータがあれば、これを用いる。以下に、必要な入力データを示す。

- 一次節点座標
 $24\text{byte} \times N_{1st} = 24N_{1st}\text{byte}$
- 一次要素コネクティビティ
 $16\text{byte} \times N_{el} = 16N_{el}\text{byte}$
- 一次節点スカラー
 $8\text{byte} \times N_{1st} = 8N_{1st}\text{byte}$

次に、作業領域については、

- 一次節点スカラー勾配ベクトル
 $24\text{byte} \times N_{1st} = 24N_{1st}\text{byte}$
- 一次元バケットデータ
 $16\text{byte} \times N_{el} = 16N_{el}\text{byte}$

一方、出力される等値面上の各種データについては、以下ようになる。

- 等値面三角形の頂点座標
 $72\text{byte} \times N_{iso} = 72N_{iso}\text{byte}$

- 等値面三角形の頂点法線方向ベクトル

$$72\text{byte} \times N_{iso} = 72N_{iso}\text{byte}$$

なお、対話セッションを保持しながらもメモリを節約したい場合には、要素コネクティビティ、節点座標および節点スカラーのみをメモリ上に保持する。このとき、実数値についてはその精度を単精度に落すこともできる。また、一次元バケットはその使用をあきらめる。ただし、一次元バケットを用いない場合には、全要素を探索する必要があるため、通常に比べて一桁以上の処理時間が必要になる。一方、出力される等値面三角形データについては、記憶せずにそのまま描画プリミティブとして画面出力する。

また、スカラー勾配ベクトルについては、画像クオリティの劣化と引き替えに省略してもよい。結果として、等値面三角形の頂点ごとの法線方向ベクトルが得られないので、各等値面三角形の描画にはフラットシェーディングを用いる。通常この規模での等値面表現は画面解像度に比べて十分細かいため、画像の一部を拡大しなければ十分な画質が得られる。

さらにメモリが不足する場合には、節点座標と節点スカラーのみを記憶する。要素コネクティビティについてはこれを記憶せず、ファイルからの逐次読み出しにより要素ごとに等値面三角形を生成する。ただし、大量のファイル入出力が必要となるため、対話性は損なわれる。

8.9.1 構造解析

例えば、一億自由度 (100M DOFs) の構造解析モデルでは、二次節点数は、 $N_{2nd} = 33M$ 、一次節点数は、 $N_{1st} = 4.1M$ 、要素数は、 $N_{el} = 23M$ 、等値面三角形数は、 $N_{iso} = R_{iso} \times 23M = 1.2M$ となる。まず、等値面抽出に必要な入力データについては、

- 一次節点座標
 $24\text{byte} \times 4.1M = 100MB$
- 一次要素コネクティビティ
 $16\text{byte} \times 23M = 370MB$
- 一次節点スカラー
 $8\text{byte} \times 4.1M = 33MB$

作業領域データについては、

- 一次節点スカラー勾配ベクトル
 $24\text{byte} \times 4.1M = 100MB$
- 一次元バケットデータ
 $16\text{byte} \times 23M = 370MB$

一方、出力される等値面上の各種データについては、

- 等値面三角形の頂点座標
 $72\text{byte} \times 1.2M = 84MB$
- 等値面三角形の頂点法線方向ベクトル
 $72\text{byte} \times 1.2M = 84MB$

8.9.2 流体解析

続いて、一千万自由度 (10M DOFs) の流体解析モデルでは、節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = 14M$ 、等値面三角形数は、 $N_{iso} = R_{iso} \times 14M = 0.70M$ となる。まず、等値面抽出に必要な入力データについては、

- 節点座標
 $24\text{byte} \times 2.5M = 60MB$
- 要素コネクティビティ
 $16\text{byte} \times 14M = 220MB$
- 節点スカラー
 $8\text{byte} \times 2.5M = 20MB$

作業領域データについては、

- 節点スカラー勾配ベクトル
 $24\text{byte} \times 2.5M = 60MB$
- 一次元バケットデータ
 $16\text{byte} \times 14M = 220MB$

一方、出力される等値面上の各種データについては、

- 等値面三角形の頂点座標
 $72\text{byte} \times 0.70M = 51MB$
- 等値面三角形の頂点法線方向ベクトル
 $72\text{byte} \times 0.70M = 51MB$

8.10 矢線の抽出

矢線の抽出処理は、ユーザーが指定する任意断面上で行われる。断面の位置や向きに加え、対象とするベクトル値の種類および、矢線原点格子の位置や間隔などのパラメータをユーザーが指定する必要がある。断面のいくつかについては、解析形状やメッシュ、および選択するベクトル場の分布の性質を考慮しながら、解析直前あるいはポスト処理開始時点でユーザーがあらかじめ必要なものを指定しておくことが出来る。または、試行錯誤的にビジュアルデータマイニングを行う際に、対話セッションにおいてユーザーがその場その場で任意に指定することもありうる。

対話操作において矢線を抽出する場合には、メッシュの三次元バケットを構築し、各要素を三次元バケットに登録しておくことで、抽出速度の著しい高速化が可能である。

矢線の抽出操作に必要とされるメモリ量は以下になる。まず、各構成データごとのメモリは、

- 矢線ごとに、原点位置
座標の各成分を実数値として、 24byte
- 矢線ごとに、要素への参照
矢線は、一つの要素だけを参照する。また、矢線原点のこの要素に関する局所要素座標が必要となる。要素番号を整数値、座標の各成分を実数値として、 $4\text{byte} + 24\text{byte} = 28\text{byte}$
- 三次元バケットデータ
三次元バケットは三次元空間の各軸方向に分割したバケット格子および、各バケットからの単方向リストより構成される。平均値として、各要素がそれぞれ二つのバケットに登録されるものとすれば、要素ごとにリンクが2つ、すなわち、整数値が4つ程必要となり、 $4\text{byte} \times 4 = 16\text{byte}$

なお、抽出される矢線の本数、すなわち断面上の格子点数はユーザーがなんらかの方法で指定するものとする。通常、この数は数百から数千程度であり、その他のデータに比べて少ないため無視できる。

矢線の抽出には、要素コネクティビティデータ、節点座標、選択したベクトル値に関する節点ベクトルが必要となる。ただし、一次要素のものでよい。低次要素化されているデータがあれば、これを用いる。以下に、必要な入力データを示す。

- 一次節点座標
 $24byte \times N_{1st} = 24N_{1st}byte$
- 一次要素コネクティビティ
 $16byte \times N_{el} = 16N_{el}byte$
- 一次節点ベクトル
 $24byte \times N_{1st} = 24N_{1st}byte$

次に、作業領域データについては、

- 三次元バケットデータ
 $16byte \times N_{el} = 16N_{el}byte$

一方、出力される矢線の各種データについては、データ量が比較的小規模なのでここでは考慮しない。

なお、対話セッションを保持しながらもメモリを節約したい場合には、要素コネクティビティ、節点座標および節点ベクトルのみをメモリ上に保持する。このとき、実数値についてはその精度を単精度に落すこともできる。また、三次元バケットはその使用をあきらめる。ただし、三次元バケットを用いない場合には、全要素を探索する必要があるため、抽出する矢線数の数によってはきわめて長い待ち時間が必要になることもある。

さらにメモリが不足する場合には、節点座標と節点ベクトルのみを記憶する。要素コネクティビティについてはこれを記憶せず、ファイルからの逐次読み出しにより要素ごとに矢線を生成する。ただし、大量のファイル入出力が必要となるため、対話性は損なわれる。

8.10.1 流体解析

例えば、一千万自由度 (10M DOFs) の流体解析モデルでは、節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = 14M$ となる。矢線抽出に必要な入力データについては、

- 節点座標
 $24byte \times 2.5M = 60MB$
- 要素コネクティビティ
 $16byte \times 14M = 220MB$
- 節点ベクトル
 $24byte \times 2.5M = 60MB$

作業領域データについては、

- 三次元バケットデータ
 $16byte \times 14M = 220MB$

8.11 パーティクルトレースの抽出

パーティクルトレースの抽出処理では、対象とするベクトル値の種類および、パーティクルの始点位置やパーティクル群の間隔などのパラメータをユーザーが指定する必要がある。そのいくつかについては、解析形状やメッシュ、および選択するベクトル場の分布の性質を考慮しながら、解析直前あるいはポスト処理開始時点でユーザーがあらかじめ必要なものを指定しておくことが出来る。または、試行錯誤的にビジュアルデータマイニングを行う際に、対話セッションにおいてユーザーがその場その場で任意に指定することもありうる。

対話操作においてパーティクルトレースを抽出する場合には、メッシュの三次元バケットを構築し、各要素を三次元バケットに登録しておくことで、抽出速度の著しい高速化が可能である。

これに加えて、要素の隣接関係を利用すれば、第一ステップ以降の検索処理がさらに高速化される。

パーティクルトレースの抽出操作に必要なメモリ量は以下ようになる。まず、各構成データごとのメモリは、

- パーティクルごとに、現在の位置
座標の各成分を実数値として、 24byte
- パーティクルごとに、要素への参照
矢線は、一つの要素だけを参照する。また、パーティクルのこの要素に関する局所要素座標が必要となる。要素番号を整数値、座標の各成分を実数値として、 $4\text{byte} + 24\text{byte} = 28\text{byte}$
- 三次元バケットデータ
三次元バケットは三次元空間の各軸方向に分割したバケット格子および、各バケットからの単方向リストより構成される。平均値として、各要素がそれぞれ二つのバケットに登録されるものとすれば、要素ごとにリンクが2つ、すなわち、整数値が4つ程必要となり、 $4\text{byte} \times 4 = 16\text{byte}$
- 要素ごとに、隣接要素への参照
要素の要素面ごとに、隣接要素が対応する。ただし、要素面がメッシュの境界表面にあれば、隣接要素は存在しない。要素番号を整数値として、 $4\text{byte} \times 4 = 16\text{byte}$

なお、パーティクル数はユーザーがなんらかの方法で指定するものとする。通常、この数は一つあるいは数十個、多くても数千個程度であり、その他のデータに比べて少ないため無視できる。

パーティクルトレースの抽出には、要素コネクティビティデータ、節点座標、選択したベクトル値に関する節点ベクトルが必要となる。ただし、一次要素のものでよい。低次要素化されているデータがあれば、これを用いる。以下に、必要な入力データを示す。

- 一次節点座標
 $24\text{byte} \times N_{1st} = 24N_{1st}\text{byte}$
- 一次要素コネクティビティ
 $16\text{byte} \times N_{el} = 16N_{el}\text{byte}$
- 一次節点ベクトル
 $24\text{byte} \times N_{1st} = 24N_{1st}\text{byte}$

次に、作業領域データについては、

- 三次元バケットデータ
 $16\text{byte} \times N_{el} = 16N_{el}\text{byte}$
- 要素から隣接要素への参照
 $16\text{byte} \times N_{el} = 16N_{el}\text{byte}$

一方、出力されるパーティクルトレースの各種データについては、データ量が比較的小規模なのでここでは考慮しない。

なお、対話セッションを保持しながらメモリを節約したい場合には、要素コネクティビティ、節点座標および節点ベクトルのみをメモリ上に保持する。このとき、実数値についてはその精度を単精度に落すこともできる。また、三次元バケットや要素隣接関係については、場合によりその使用をあきらめる。ただし、これらを用いない場合には、全要素を探索する必要があるため、きわめて長い待ち時間が必要になることもあるため、すくなくともどちらかは残しておいたほうがよい。

さらにメモリが不足する場合には、節点座標と節点ベクトルのみを記憶する。要素コネクティビティについてはこれを記憶せず、ファイルからの逐次読み出しにより要素ごとに矢線を生成する。ただし、大量のファイル入出力が必要となるため、対話性は損なわれる。

8.11.1 流体解析

例えば、一千万自由度 (10M DOFs) の流体解析モデルでは、節点数は、 $N_{1st} = 2.5M$ 、要素数は、 $N_{el} = 14M$ となる。パーティクルトレース抽出に必要な入力データについては、

- 節点座標
 $24byte \times 2.5M = 60MB$
- 要素コネクティビティ
 $16byte \times 14M = 220MB$
- 節点ベクトル
 $24byte \times 2.5M = 60MB$

作業領域データについては、

- 三次元バケットデータ
 $16byte \times 14M = 220MB$
- 要素から隣接要素への参照データ
 $16byte \times 14M = 220MB$

第9章 描画処理のパフォーマンス設計

9.1 性能見積りの前提条件

コンピュータグラフィックスでは通常、座標や法線ベクトルの各成分を始めとしてほとんどの実数は単精度で表現される。実数値は 32 bit(4 byte) を占める。

ただし、色の RGB 各成分の表現については、単精度浮動小数点表現以外に、0 から 255 までの 1 バイト整数で表現することもできる。もっとも、比較的最新のグラフィックスカードにおいて、特にピクセルシェーダが浮動小数点数を扱える場合には、色の RGB 各成分を最終段階まで単精度浮動小数点表現のまま保持するようである。

一方、必要に応じて、アニメーション表示やネットワーク転送におけるデータ圧縮のために実数を近似するような整数表現をとることがありうる。このとき、データサイズは 1 バイトあるいは 2 バイトとなり、それぞれ 256、65536 階調の表現が可能である。なお、これによりいくらか精度が犠牲にされるので、この技法は問題に応じて注意深く用いられなければならない。

実数の整数近似表現の具体的な方法として、ここでは、これで表現されるスカラー値ごとにその最大最小レンジを用い、この範囲で 1 バイトあるいは 2 バイトの符号無し整数により階調表現を行う。浮動小数点表現から整数表現にエンコードする際には、各スカラー値についてその最大最小のレンジを求め、この範囲内で符号無し整数に変換する。一方、整数表現を元の浮動小数点表現にデコードするには、その整数値を別途保存しておいた最大最小レンジを用いて浮動小数点値に復元する。この方法では、少なくとも最大値と最小値についてはその精度が保証される。

実数の整数近似表現を利用するためのガイドラインとしては、

- 座標
画面の解像度が数百のオーダーなので、オブジェクトの位置やサイズを表すには 1 バイト整数表現では不十分である。多くの場合には最低でも 2 バイト表現が必要となるだろう。
- 色の成分
多くの場合、RGB の各成分は最終的にフレームバッファ上で 1 バイト符号無し整数で表現される。したがって、ほとんどのケースににおいて、すなわち、ライティングやシェーディングなど色に関する比較的単純な演算しか行われない場合であれば、色の成分は 1 バイトで十分である。ただし、半透明表示やボリウムレンダリングのように透明度が絡む演算では、注意が必要である。
- スカラー値
多くの場合には最低でも 2 バイト表現が必要となるだろう。ただし、そのスカラー量がコンター表示または等値面表示でのスカラーとして用いられ、さらに、アニメーション作成などにおいて、ユーザーによる画面の拡大縮小操作やコンターレンジ変更操作が行われないような状況に限り、1 バイト整数表現が利用できる。
- ベクトル値の各成分
多くの場合には最低でも 2 バイト表現が必要となるだろう。ただし、そのベクトル量が、変形図における変形オフセットや、矢線表示、パーティクル表示で

のベクトルとして用いられ、さらに、アニメーション作成などにおいて、ユーザーによる画面の拡大縮小操作が行われないような状況に限り、1 バイト整数表現が利用できる。

以下の議論では、すべての実数について基本的には単精度表現を用いる。ただし、必要に応じて整数近似について言及し、そのメモリ削減効果や性能への影響を検討する。

9.2 描画プリミティブ

ハードウェアベースのレンダリングにおける最も基本的な描画プリミティブは、線分と三角形である。ここでは、特に断りがない限り、以下のような形でこれらを用いる。

三角形をレンダリングするには、基本的には 3 つの頂点座標と一つの法線方向ベクトルの指定が必要である。座標は x 、 y 、 z の三成分から構成され、これらは実数値とする。また、法線方向ベクトルは nx 、 ny 、 nz の三成分から構成され、これらはもまた実数値とする。なお、このとき三角形の塗りつぶし色はあらかじめ指定されているものとする。これと、光源方向と法線方向の向きからライティングにより本当の色が決定される。この色を用いて、三角形全体が単色で塗りつぶされる。これは、フラットシェーディングと呼ばれる。

場合により、これに加えて 3 頂点の各々ごとに法線方向ベクトルが指定されることがありうる。同様に、 nx 、 ny 、 nz の実数値三成分から構成される。各頂点ごとに異なる法線方向ベクトルが指定できるので、それぞれの頂点でライティングが行われ、これは頂点ごとに異なる色を生成する。そして、最終的には三角形内で 3 頂点間のスムーズな色補間が行われる。これは、グーローシェーディングと呼ばれる。頂点ごとの法線方向ベクトル指定は、等値面表示の際に用いられる。

場合により、さらに、3 頂点の各々ごとに色が指定されることがありうる。それぞれの色は RGB(red, green, blue) 三成分より構成され、これらは実数値とする。各頂点で異なる色が指定できるが、三角形の法線方向ベクトルを用いてこれらがそれぞれライティングされ、上と同様頂点ごとに異なる色を生成する。そして、上と同様に、三角形内で 3 頂点間のスムーズな色補間が行われる。これにより、ライティングされたスムーズコンターが生成できるので、立体感を保ちながらスカラー場の可視化を行うことが出来る。頂点ごとの色指定は、表面または断面スカラーのコンター表示で用いられる。

次に、線分については、これは 2 つの端点座標より構成される。線分の塗りつぶし色はあらかじめ指定されているものとする。また、線分についてはライティングやシェーディングを行わず、指定色でそのまま描画する。

以上の定義は、これらの描画プリミティブをグラフィックスライブラリの API を通して指定する際に適用される。また、グラフィックスライブラリがこれらのプリミティブデータを AGP または PCI Express バスを通してグラフィックスカードに転送する際にも引続き有効である。多くの場合、描画性能のボトルネックはこのグラフィックスカードへの転送過程で生じるため、データ転送量と AGP または PCI Express バスのデータ転送バンド幅より描画速度をある程度見積もることができる。このときのデータ転送量の計算では上記の定義をそのまま用いることとする。

まとめると、以下のようになる。

- 三角形の頂点座標
 $4\text{byte} \times 3 \times 3 = 36\text{byte}$
- 三角形の法線方向ベクトル
 $4\text{byte} \times 3 = 12\text{byte}$
- 三角形の頂点法線方向ベクトル
 $4\text{byte} \times 3 \times 3 = 36\text{byte}$
- 三角形の頂点色
 $4\text{byte} \times 3 \times 3 = 36\text{byte}$

- 線分の端点座標

$$4\text{byte} \times 3 \times 2 = 24\text{byte}$$

一方、三角形や線分データのメモリ上での表現は、上記の描画プリミティブ定義とまったく同じ形式をとる場合もあれば、ある程度形式が異なる場合もありうる。

- 描画プリミティブをそのままの形式でメモリ上に記憶する。これをメモリより読み出し、グラフィックスライブラリの API コールを行う。CPU による計算負荷が少ないので、経験上、この方法を用いた場合の描画速度が最も高速である。
- 描画プリミティブをまず別の形式でメモリ上に記憶しておき、グラフィックスライブラリの API コール時に CPU による計算により上記の定義に変換する。アニメーションなど、多数のシーンデータをメモリ上あるいは HDD 上に記憶する必要がある場合には、必要な記憶容量を圧縮できるため効果的である。
- 等値面抽出や矢線抽出などのような、何段かの比較的重いデータ変換過程の最終段階において、描画プリミティブをオンザフライで生成し、これらを記憶せずそのままグラフィックスライブラリの API コールを行う。断面や等値面について、生成パラメータを少しずつ変更しながら生成と描画を連続して行う場合には、このほうがメモリを節約できる。

9.3 アニメーション

時系列データやマルチステップの結果について、これらを連続して表示することで、解析結果のダイナミックな性質について理解を深めることができる。

大規模解析においてアニメーションを実現することは困難であり、計算と描画、およびそのための記憶領域の利用に関して難しいトレードオフが存在する。

大規模解析では大量の描画プリミティブデータが生成される。これに対して、なるべくリアルタイムレベルの描画速度を維持したいので、データ変換などの余計な CPU 負荷を避ける必要があり、出来れば描画プリミティブデータをそのままの形式でメモリ上に保存しておきたい。

しかしながら、アニメーション表示で描画されるオブジェクト群は基本的にダイナミックであり、ステップを通してまったく変化しないようなスタティックなデータは少ない。したがって、アニメーション表示において最高の描画性能を得るためには、すべてのステップにおけるすべての描画プリミティブをメモリ上に保存し、これらを連続表示する必要がある。ところが、大規模解析では必然的にメモリが足りなくなる。

これに対する選択子としては、アニメーション表示でのステップ数をメモリに入り切るまで削るか、あるいはデータを HDD から読むように変更する。後者の HDD から読み出す場合には、これはステップ数の制約がない一方で、HDD からの読み出し速度がメモリ読み出しよりも最悪 100 倍近く遅いことを考慮しなければならない。

データをメモリ上に置くか、それとも HDD 上に置くか、どちらの場合にも、結局は各ステップのデータを出来る限り圧縮することになる。ここでは、これまでとは逆に CPU の計算能力を積極的に用いて、圧縮されたデータを CPU パワーによって伸長し、描画プリミティブに変換することを考える。もし、データをメモリ上に置くことができる場合には、このデータ伸長処理における CPU 負荷が描画のボトルネックとなる。一方、データを HDD 上に置く場合には、HDD からのデータ読み込み時間に対して無視できる速度で CPU が伸長処理を行えばよい。

ちなみに、HDD からデータを読み出す場合には、HDD からの読み出し速度を 50MB/s とすると、1 秒間に 5 つのステップを表示するためには、ステップごとのデータサイズを 10MB に制限しなければならない。このレベルでデータ圧縮を行うには、4 バイトの単精度浮動小数点表現では足りず、1 バイトあるいは 2 バイトの固定浮動小数点表現が必要になることもある。

9.4 境界表面の描画

ここでは、メッシュ境界表面形状の可視化、およびその上の物理量の可視化について検討する。メッシュ表面上の物理量表示として、具体的には、スカラー値のスムーズコンター表示とベクトル値を変位場と見立てた場合の変形表示の二つについて考慮する。なお、メッシュ境界表面を表示するには、先に境界表面において抽出した表面パッチデータを用いる。

なお、ここでは時系列に関するアニメーション表示は考慮しない。解析モデルの形状やメッシュ表示、および、決められたある特定のステップにおけるコンター図や変形図に関して、これらについて移動、回転および拡大縮小を行う状況を想定する。また、これを行えるだけ高速に行えるようにすることを考える。なお、アニメーションについては、次節を参照。

もし、解析モデルの形状を表示したいならば、表面パッチの三角形を適当な単色でただ単純に描画すればよい。

次に、スカラー値のスムーズコンタープロットが必要であれば、これらの三角形の各頂点についてコンターに応じた色を RGB で指定し、スムーズシェーディングを行う。

一方、ベクトル値の変形プロットが必要な場合には、表面パッチの三角形の各頂点について、その位置をあらかじめ適当なオフセット量だけ変形させておき、これを単純に表示すれば良い。オフセット量については、まず選択するベクトル量を変位と見なし、これにユーザーが指定するスケールをかけてオフセットとする。

それぞれのプロットについてさらに、有限要素メッシュの個々の要素を区別するために、要素境界の稜線をプロットに重ねて表示することについても検討する。もし、解析モデル形状の表示に加え、これとは別の色を用いて要素稜線を表示すれば、これは通常メッシュ図と呼ばれるものとなる。

さらに、メッシュ表面上の任意の要素、各節点あるいは面グループに関して、それらのマウスによるピック / 選択についても同時に考慮する。

ここで、境界表面の描画において必要なデータ量について検討する。

三角形数は表面パッチ数と同じ、 N_{pch} である。各三角形は頂点座標と法線方向ベクトルを有する。また、コンター表示の場合にはさらに、各頂点ごとに RGB の色情報を持つ。

一方、要素稜線を表現する線分群については、各三角形につき 3 本の線分が存在し、各線分は 2 端点の座標より構成される。すべての線分は同じ色で描画されるものとする。

ここでは、描画速度を最大にするため、描画プリミティブをそのままの形式でメモリ上に記憶することにする。このとき、

- 三角形の頂点座標
 $36byte \times N_{pch} = 36N_{pch}byte$
- 三角形の法線方向ベクトル
 $12byte \times N_{pch} = 12N_{pch}byte$
- 三角形の頂点色
 $36byte \times N_{pch} = 36N_{pch}byte$
- 稜線線分の端点座標
 $24byte \times 3 \times N_{pch} = 72N_{pch}byte$

9.4.1 構造解析

例えば、一億自由度 (100M DOFs) の構造解析モデルでは、二次節点数は、 $N_{2nd} = 33M$ 、一次節点数は、 $N_{1st} = 4.1M$ 、要素数は、 $N_{el} = 23M$ 、表面パッチ数は、 $N_{pch} = 2.3M$ となり、

- 三角形の頂点座標
 $36\text{byte} \times 2.3M = 83MB$
- 三角形の法線方向
 $12\text{byte} \times 2.3M = 28MB$
- 三角形の頂点色
 $36\text{byte} \times 2.3M = 83MB$
- 稜線線分の端点座標
 $72\text{byte} \times 2.3M = 170MB$

より具体的に、ソリッド表示、コンター表示およびそれらに稜線を加えたケースについて、レンダリング性能予測を行う。ここでは、性能のボトルネックがメモリアクセス速度およびグラフィックスカードへのデータ転送速度であると仮定する。このとき、データ転送バンド幅をそれぞれ実効で 3GB/s、1GB/s とおく。また、転送時間はそれぞれにかかるデータアクセス時間の和とする。トータルでは、0.75GB/s の実効バンド幅となる。

- ソリッド表示を行う場合
この場合、三角形の頂点座標と法線方向が必要となる。合わせて 100MB が転送される。7.5 フレーム/s の描画が可能である。
- 稜線つきソリッド表示 (メッシュ表示) を行う場合
この場合、ソリッド表示に対してさらに稜線が加わる。合わせて 270MB が転送される。2.8 フレーム/s の描画が可能である。
- コンター表示を行う場合
この場合、三角形の頂点座標、法線方向および頂点色が必要となる。合わせて 180MB が転送される。4.2 フレーム/s の描画が可能である。
- 稜線つきコンター表示を行う場合
この場合、コンター表示に対してさらに稜線が加わる。合わせて 360MB が転送される。2.1 フレーム/s の描画が可能である。
- ピックを行う場合
この場合、セレクション判定を行うため、三角形の頂点座標と法線方向が必要となる。また、選択された部分をハイライト表示するために、再度ソリッド表示が行われる。合わせて 200MB が転送される。3.8 フレーム/s のピック操作が可能である。

なお、変形図については、基本的にソリッド表示と同じものとなる。

9.5 境界表面のアニメーション

ここでは、境界表面上のスカラー値およびベクトル値について、そのマルチステップデータのアニメーション表示について検討する。特に、構造解析においてニーズの強い、変形図とコンター図を同時に表示する場合を考える。

ここでは、各ステップのデータをメモリまたは HDD 上に記憶する。後者の場合には、ステップを通じて変化しないスタティクなデータはメモリ上に置き、ステップごとに化するダイナミックなデータのみを HDD 上に置く。ダイナミックなデータには、表面節点上のスカラー量とベクトル量が配置される。このとき、スカラー量はコンターのグレードに、ベクトル量は変形のオフセットとして用いられる。

ここで、表面要素数は表面パッチ数と同じ、 N_{pch} である。また、表面節点数は、 $1/2N_{pch}$ である。まず、スタティクなデータについては、以下のようになる。

- 表面要素コネクティビティ
境界表面抽出と同じく、 $12N_{pch}\text{ byte}$

- 表面節点座標
各成分ごとに実数値として、 $4byte \times 3 \times 0.5N_{pch} = 6N_{pch}byte$

次に、ダイナミックなデータについては、以下のようになる。

- 表面節点ベクトル
各成分ごとに実数値として、 $4byte \times 3 \times 0.5N_{pch} = 6N_{pch}byte$
- 表面節点スカラー
実数値として、 $4byte \times 0.5N_{pch} = 2N_{pch}byte$

なお、各三角形の法線方向ベクトルはその頂点座標より CPU 側で計算する。ただし、CPU 側で正規化を行うのは計算負荷が大きいため、CPU 側ではその正規化までは行わず、グラフィックスハードウェアに任せる。(OpenGL では、GL_NORMALIZE と併用する。)

9.5.1 構造解析

例えば、一億自由度 (100M DOFs) の構造解析モデルでは、二次節点数は、 $N_{2nd} = 33M$ 、一次節点数は、 $N_{1st} = 4.1M$ 、要素数は、 $N_{el} = 23M$ 、表面パッチ数は、 $N_{pch} = 2.3M$ となる。

まず、スタティックなデータについては、以下のようになる。

- 表面要素コネクティビティ
境界表面抽出と同じく、28MB
- 表面節点座標
 $6 \text{ バイト} \times 2.3M = 14MB$

次に、ダイナミックなデータについては、以下のようになる。

- 表面節点ベクトル
 $6 \text{ バイト} \times 2.3M = 14MB$
- 表面節点色
 $2 \text{ バイト} \times 2.3M = 4.6MB$

これを 100 ステップ分記憶するには、1.9GB のメモリが必要である。一方、これらを HDD 上に置くものとすれば、HDD の読み込み速度を 50MB/s とすれば、ステップごとの読み出しに 0.38 秒必要となる。

なお、これらのダイナミックなデータについては、実数の整数近似表現を用いることもできる。それぞれを 2 バイト表現とすれば、データ量を $1/2$ に、また、それぞれを 1 バイト表現とすれば、データ量を $1/4$ まで削減できる。

第10章 実行例と性能評価

10.1 問題設定

10.2 境界表面の抽出

10.3 境界表面のグループ化

10.4 任意断面の抽出

10.5 描画

第11章 おわりに

関連図書

- [1] 河合他 1 名、計算力学講演会講演論文集、17(2004),??-??.
- [2] <http://adventure.q.t.u-tokyo.ac.jp/>
- [3] 吉村、計算工学、4-4(1999), 210-218.
- [4] 河合他 1 名、日本計算工学会論文集、9(2004),859-862.