99-01098

VRMLを用いた可視化サーバ構築に関する研究

小山田　耕二　　京都大学大型計算機センター

研究調査の目的・意義

本研究では、インターネットを介して数値シミュレーション結果（本研究では、計算数値流体力学計算(CFD)結果に焦点をあてる。）を遠隔地で効率よく可視化するための幾何形状データ処理に関する技術開発を行う。現在、インターネット上でのコミュニケーションにテキスト、画像を利用することはごく日常的に行われている。最近ではPCの性能が向上し、３次元グラフィックス表示も十分可能となってきているため、インターネット上で３次元グラフィックスデータを介したコミュニケーションの実現に対する期待も高くなってきている。インターネット上での３次元グラフィックスデータの表現については標準作業が進行し、その大きな成果がVirtual Reality Modeling Language(VRML)である。

本研究の目的は、数値シミュレーション結果からVRML形式の幾何形状を「適切に」生成する手法を開発することである。インターネットを介した可視化にVRMLを採用するのはひとつの有効な戦略である。数値シミュレーション結果から変換した幾何形状をVRML形式で記述しておけば、VRMLに対応した様々なインフラの恩恵を得ることができる。残念ながら、VRMLは、可視化を特に意識した仕様となっているわけではない。

本研究の意義は、これまで専門家しか触れる機会のなかった数値シミュレーションの世界をより多くの人々に開放することにある。本研究成果により、多くの人々が数値シミュレーション結果を自らの目的のために活用することができるようになると期待する。

CFD解析技術の向上の結果、その利用分野が拡大すると同時にその解析結果の恩恵を受ける人も増えている．例えば、営業担当者が解析結果を使って、顧客に商品をプレゼンテーションするケースも出初めている。しかし、多くの解析結果は、技術論文・報告書中の参照図という形で公表され、さらに入手するにも困難な場合が多い。 WWWのサーバに解析結果ファイルを格納し、これをインターネットを介して可視化する仕組み（リモート可視化）を構築すれば、多くの人々が、WWWのクライアント端末を使ってその恩恵にあずかることが可能になる。

また、計算機のパフォーマンスが向上してきたため、数値シミュレーションを使った最適設計を行われるようになってきている。このような最適設計を依頼された解析技術者は、しばしば、遠隔地にいる依頼者に報告を行う必要がある。この場合もリモート可視化システムを使って、依頼者が最適化したいと思う評価関数（重量、コストなど）と変更可能パラメータとの関係を表示することが可能となる。

研究調査の方法

本研究において、「可視化」は、「CFD解析結果から意味のある幾何形状を取り出し、グラフィックスディスプレーに表示すること」を意味するものとする。また、本研究では、CFD解析結果として３次元格子の各頂点で定義されたスカラデータ、ベクタデータを取り上げ、以降これをボリュームデータと呼ぶ。ボリュームデータから抽出された幾何形状データをインターネットで配信するためのフォーマットとしては、インターネット上におけるグラフィックス標準フォーマットであるVirtual Reality Modeling Language(VRML)を採用する。

シミュレーション結果の可視化の観点でVRMLをいかに記述するのがいいのかについて調査を行う。対話機能やアニメーション表示といった動的な表現方法については、VRMLの記述例を示す。これらの調査結果をもとに、我々は、研究のテストベッドとして、適切なVRMLファイルを配信するWWWサーバのプロトタイプを開発する。

VRMLを使って実用的な「可視化」を実現するには、サーバ側でできる限り幾何形状のデータ量を削減しておく必要がある。可視化処理で取り扱う幾何形状として最も頻度の高いデータ付3角形メッシュ(各頂点で数値データが定義されているような3角形メッシュ)を例にとると、実用レベルで用いられているボリュームデータから生成されるデータ付3角形メッシュの数は数万程度である。このような規模の幾何形状データをインターネットで送信し、これをクライアントで受け取って表示するのは困難である。現在、多くの研究者がメッシュに含まれる３角形の数を適切に削減する手法の開発に取り組んでいる。我々は、この問題を解決するため、可視化処理をスライス面に限定し、データ付3角形メッシュを大きな１枚のテクスチャ付４角形で表現するというアプローチを採用する。本アプローチに基づくスライス面作

成プログラムを新規購入する「Linuxマシン」上で実装し、WWWサーバと連動させる。

スカラデータの定義されたボリュームデータ（スカラボリューム）については、複数のスライス面を用いたボリュームレンダリング表示法を開発する。ここで、ボリュームレンダリングは、３次元空間で定義されたスカラデータをデータの分布にしたがって発光する一種の密度雲(発光粒子の集合体)として表示する手法である。ボリュームレンダリングを実現するアルゴリズムのひとつとして、スライスバイスライス法が提案されている。この手法では、視線に垂直な複数のスライス面上で不透明度付テクスチャを作成し、視点から見て奥から順にテクスチャを重ね合わせる。VRMLの仕様では、スライス面の描画順序を制御についての記述がないので、スライス面を奥から順に重ね合わせることが困難である。このため、VRMLシーンでボリュームレンダリング画像を表現することは困難であるとされてきた。ボリュームレンダリングにおける輝度値計算で用いる方程式において、発光粒子の発光量を一定とし、その密度だけを考慮すると任意の順序でスライス面を重ねあわせても問題がないことを確認した。この場合、グレースケール画像をテクスチャとして使うことになり、スカラ場の表現力としては弱くなるというデメリットが出てくる。スカラー値に対応した色つきのボリュームレンダリング表示を行うためにサーバ側でボリュームレンダリング画像を生成し、画像データとしてWWWクライアントに転送する。この画像の生成には、新規購入する「ボリュームレンダリング用PCIボード」を利用する。

ベクタデータの定義されたボリュームデータ（ベクタボリューム）については、特に渦の存在を調べるために有用な技術の開発を行う。渦の存在を調べる場合、渦中心を通過し、その渦軸に垂直な平面でスライスすれば、渦回りの局所的なベクタ場をわかりやすく表現できるものと期待できる。我々は、このようなスライス平面上でベクタ場を投影しベクタテクスチャを生成する。ベクタテクスチャから画像テクスチャの作成には、Line Integral Convolution(LIC)法を採用する。ベクタボリュームにおいて渦は、一般に複数あるので、設定されるスライス面は一般に複数枚となる。お互いのスライス面が干渉しないようにスライス面において渦中心付近以外では高い透明度を設定する。低い透明度を設定するのは、中心近傍から重さ無しの粒子を散布した場合、ある一定時間内に粒子が通過する領域とする。

最適化計算では、数値シミュレーションのパラメータを変化させ、結果であるボリュームデータから計算される評価関数の最小（最大）化を行う。例えば、ノートPC筐体上の指定された点での温度が与えられた温度と一致するようなPC部品の熱伝導率を求める場合、評価関数は、解析結果である温度ボリュームを使って計算される。この場合、評価関数は、パラメータ熱伝導率の陽的な関数となっていないので、最適化計算結果を可視化する場合には、評価関数について回帰曲面を作成する。本研究では、クライアント側で最適化の対象であるパラメータのうちから２つを選択し、評価関数値を2次元格子上の高さデータとして表示するVRML ElevationGridノードを用いて表現する。

本手法の評価を行うためボリュームデータから抽出されたスライス面をデータ付き3角形メッシュそしてテクスチャ付き4角形の2通りの方法で表現し、それぞれのデータサイズを比較する。

結果

本章では、まず、VRMLについての記述方法について、実例をあげながら説明を行う。次に、数値シミュレーションの可視化手法として代表的なものを取り上げ、VRMLを使ってそれらを実装するために必要とされる技術について述べる。可視化手法として、スライス面表示手法を取り上げ、その有用性を転送データ量の観点で評価する。また、可視化サーバシステムをLinuxシステム上でのシステム構築事例についても説明する。

VRMLの記述方法について

VRMLは、ノードとフィールドから構成される。ノードは、仮想3次元空間における全てのオブジェクトを記述するもので、フィールドは、それぞれのノードが表現するオブジェクトの属性を定義する。特に、３次元物体の形状は、Shapeノード

```
Shape {
        geometry    ...
        appearance ...
}
```

によって表現する。このShapeノードは、２つのフィールドgeometry, appearanceから構成される。geometryフィールドは、物体の形そのものをそして appearanceフィールドは、物体の見栄え(色やテクスチャ)を定義する。例えば、半径1.5の球を定義するには、以下のように記述する。

```
#VRML V2.0 utf8
DEF Particle Shape {
```

```
    appearance Appearance {
        material Material { }
    }
    geometry Sphere {
        radius 1.5
    }
}
```

ここで、第１行目は、このファイルがVRMLであることを示すものである。一般的に、#から始まる行はコメント行となる。また、DEFは、ノードに名前をつけるために使用する予約語である。DEFに続く名前は、その次に続くノードを再利用するときに使用できる。見栄えのうち色を定義するMaterialノードについては、記述が省略されている。このような場合、省略値(この場合は白)が割り当てられる。

対話機能

ユーザは、VRMLシーンにおいて、物体をクリックやドラッグしたりすることができる。このようなシーンに対する働きかけは、VRMLのSensorノードを通して実現することができる。Sensorノードには、ユーザの操作を検知して、操作に関する情報（イベント:データ付きメッセージ）を他ノードに送信する機能が実装されている。例えば、シーン中のある物体をクリックすると音楽がスタートするようなシーンでは以下のように記述する。

```
Sound{
    source DEF Music AudioClip{. . . }
}
...
DEF Button TouchSensor{}
...
ROUTE Button.touchTime TO Music.startTime
```

ここで、ROUTEから始まる1行は、Bottonと名付けられたTouchSensorノード(TouchSensor node)がクリックされた時刻をMusicと名付けられたAudioClipノードの音楽開始時時刻フィールドへ受け渡すことを定義している。

アニメーション機能

VRMLにおけるアニメーション手法としては、時間毎に物体の位置を設定し、これらの位置を補間することによって動きを表現するキーフレーム法が代表的である。VRMLでは、キー値としてTimeSensorノードの出力する０から１までの値(fractionと呼ぶ)を以下に示すようなPositionInterpolatorノードに渡すことによりキーフレーム法を実現することができる。

```
PositionInterpolator {
    key [ 0.0,. . .,1.0]
    keyValue [ x y z, . . . ]
}
```

PositionInterpolator{}は、０から１までの値を昇順に並べたkeyというフィールドとkeyで定義された値と同数の座標値を並べたkeyvalueフィールドから構成される。パーティクルアニメーションでは、アニメーションの開始時間を0,終了時間を1と正規化して定義し、各時刻でのパーティクルの位置座標をkeyvalueフィールドに並べる。

以下に、パーティクルアニメーションを実現するVRMLの記述例を示す。

```
DEF P Transform{
translation 0.9 1.9 3.5
children[ USE Particle,
        DEF PT TouchSensor]
},
DEF PTI TimerSensor{},
DEF PP PositionInterpolator{}
ROUTE PT.touchTime TO PTI.set_startTime
ROUTE PTI.fraction_changed TO PP.set_fraction
ROUTE PP.value_changed TO P.set_translation
```

USEは、予約語DEFで名前をつけたノードを再利用する場合に使う予約語である。この場合、Particleという名前をつけたShapeノードを再利用している。Transformノードは、複数のノードをグループ化するためのノードである。グループ化の対象となるノードは、children[]の中に記述する。ParticleとPTと名付けられたTouchSensorノードがグループ化されているので、物体Particleへのクリックは、PTが検出する。

このVRMLファイルの場合、時刻0で座標(0.9,1.9,3.5)に位置するパーティクルをクリックすると、その時刻がPTからPTIと名付けられたTimeSensorノードへ受け渡される。その後、PTIでは、0から1までの値を周期的に出力し、これをPPと名付けられたPositionInterpolatorノードに受け渡す。PPでは、この値をkey値として受け取り、座標の補間計算を行い、その結果をPと名付けられたTransformノードのtranslationフィールドへ受け渡す。このフィールドは、グループ化されている物体の位置ベクタを表すので、球Particleはその位置へ移動することになる。この動作が連続的に行われるので、球ParticleはPositionInterpolator{}のkeyvalueフィールドに並べられた座標に沿って運動していくことになる。

VRMLにおけるもうひとつのアニメーション手法には、あらかじめ定義しておいた複数の幾何形状を順番に表示させるというものがある。この手法では、fractionの値をノードに割り振られた整数値に変換することにより、表示の順序を制御する。この変換には、Scriptノードのもつプログラム機能を利用する。以下に、あらかじめ、Shape0,Shape1,...ShapeNと名付けられたN+1個のShapeノードを順番に表示していくために定義したVRMLファイルの例を示す。

```
DEF SW Switch {
        whichChoice         0
        choice[ USE Shape0,
                USE Shape1,...,
                USE ShapeN
        ]
}
DEF ST TouchSensor {},
DEF STI TimerSensor{},
DEF TOGGLE Script {
    eventIn     SFFloat     frac
    eventOut    SFInt32     item
    url[
        ``javascript:
            function frac(cur,now){
                item = cur*N;
            }"
    ]
}
ROUTE ST.touchTime TO STI.set_startTime
ROUTE STI.fraction_changed TO TOGGLE.fraction
ROUTE TOGGLE.item TO SW.set_whichChoice
```

先ほどの例と同様にクリックされた時刻がSTと名付けられたTouchSensorノードからSTIと名付けられたTimeSensorノードへ受け渡される。その後、*STI*では、0から1までの値を周期的に出力し、これをTOGGLEと名付けられたScriptノードに受け渡す。TOGGLEでは、入力の仮引数がSFFloat(実数)型のfrac, そして出力の仮引数がSFInt32(整数)型のitemであると定義されている。

eventInは、別のノードからのメッセージを受け取ることのできるフィールドであることを、そしてeventOutは、別のノードへメッセージを送信することのできるフィールドであることを宣言する。ここでは、プログラム言語としてJavascriptを使用した例を示したが、Javaを使用することも可能である。関数frac(名前は、仮引数の名前と一致させる)は、入力された0から1までの実数を0からNまでの整数に変換する。

流れ場における渦中心表示

クライアント側で利用する可視化手法を限定することができれば、実質的なデータ削減が可能となる。例えば、流れ場の中で渦中心といった特徴点を探索する場合については、生成される幾何形状は、いくつかの点データであり、その数が多くない限りネットワークを転送されてくるデータ量は小さいものと期待することが出来る。

渦中心を探索するためには、格子毎に速度勾配テンソルを計算し、その特性方程式の解を調べる。３次元速度場の場合、特性方程式は、３次方程式となる。特性方程式の解である固有値の中に共役複素数が含まれる場合、その格子には渦が含まれると判断する。この場合、渦の中心軸は、他の実固有値に対応する固有ベクタに平行となる。

このようにして探索された渦中心を表現する点データは、VRMLの基本形状ノード(Sphere, Cylinder, Box, Cone)を組み合わせて表現することができる。渦中心において定義される速度勾配テンソルの複素固有値の虚部が渦の回転速度に対応するので、この速度の大きさに応じた周波数の音を出すSoundノードを割り当てると効果的である。

ボリュームデータのスライス面表示

スライス面表示というのは、ある平面を設定して、その面上でボリュームデータを可視化する手法のことである。ボリュームデータをスライス面にマッピングするには、スライス面と格子稜線との交点を計算し、交点を頂点とする３角形メッシュを作成する。頂点では、格子頂点で定義された数値データを使って補間計算を行う。３角形メッシュを表現するために、VRMLでは、IndexedFaceSetノードを使う。スライス面上で定義される３角形の頂点は、スライス面と格子の稜線との交点である。その数は、格子の数の2/3乗に比例し、数十万の格子からだと数万個程度となる。

インターネットを通じて、大量の３角形メッシュからなるスライス面を転送するのはパフォーマンスの観点で現実的ではない。この問題を解決するためのひとつの方法は、図１のように３角形メッシュとして表現されたスライス面形状を１個のテクスチャ付き４角形として表現することである。３角形メッシュは、まず、スライス面を投影面とするように座標変換が行われる。この場合スライス面の法線と平行な視線を仮定して、視点座標系への変換を行う。そして引き続き、スクリーン座標系への変換を行う。各画素におけるデータ値は、頂点で定義されたデータを使ったスキャンコンバージョンによって画像データにおける各画素に展開される。

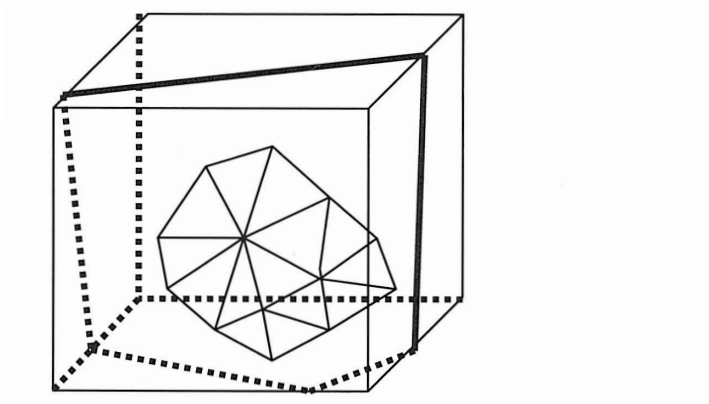

図1　テスクチャ付き4角形として表現されたスライス面

スカラ場の画像化処理

可視化の対象がスカラデータが頂点で定義された3角メッシュである場合、一般的に、等高線表示を行うか、または、カラールックアップテーブルを用いたコンタ表示を行うことが多い。カラールックアップテーブルは、スカラデータとカラーデータの関係付けを行うために使用する変換表である。

等高線表示を行う場合、３角形単位で局所的な等高線を線分として計算し、この線分の集合を画像データにおける各画素に展開する。次に、コンタ表示を行う場合、３角形の各頂点でスカラデータからカラールックアップテーブルを使ってカラーデータに変換し、スキャンコンバージョンアルゴリズムを使って、３角形ごとに面内の塗り潰しを行う。

ベクタ場の画像化処理

ベクタデータが頂点で定義された3角メッシュをスキャンコンバージョンすると

２次元直交格子上で定義されたベクタ場が生成される。２次元ベクタ場を表現するには、スライス面で適切な間隔で設定した場所でベクタデータを表す矢印を描画することが多いが、２次元ベクタ場を画像化する場合に対しては、流線畳み込み（Line Integral Convolution:LIC）法が有効である.

ベクタ場を可視化する場合、しばしば渦の位置を調べることが目標となる。この場合、渦ベクタに垂直で渦中心を通過する平面をスライス面に設定する。一般に与えられた速度ボリュームにおける渦の数は複数あるので、設定されるスライス面は複数となる。図２は、複数個の渦を含むベクタ場を複数のテクスチャ付スライス面で表示したものである。スライス面同士がお互いを隠蔽することを避けるために、LICの入力となるホワイトノイズに関して渦周辺以外についてはその強度をゼロとした。

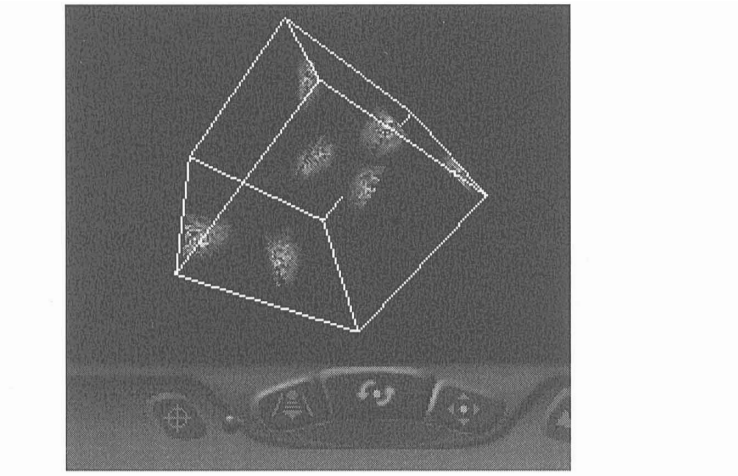

図2　複数のテスクチャ付スライス面で表示された渦

等値面表示

スカラ場の可視化手法のひとつに等値面表示がある。等値面は、スカラボリュームデータf(x,y,z)の定義された３次元空間においてf(x,y,z)=Cを満足する点の集合であり、一般的にマーチングキューブ法を用いて抽出されることが多い。この場合、等値面は、スライス面同様３角メッシュを用いて表現される。しかし、等値面は、スライス面と違って、必ずしも平面とならないために、先ほど述べたテクスチャマッピングを用いた幾何形状データの削減手法を利用することができない。すると、考えられる方法としては、サーバ側で

- 得られた等値面データを削減する
- あらかじめ、ボリュームデータを削減する

のいずれかを行なうことになる。前者だとクライアントのアクセスごとに、等値面データの生成に加えて、その削減処理が必要となる。後者は、一旦、サイズの小さいボリュームデータを作っておけば、クライアントのアクセスに対して、等値面データの生成処理だけとなる。しかも、対象となるボリュームデータのサイズが小さくなっているので、生成処理自身も軽くなるものと推定できる。

このため、等値面を近似的に表現するには、後者のように、あらかじめボリュームデータを適切に削減しておくのがよいと考える。この適切さは、なんらかかの誤差指標で表現する。誤差指標の一例として、もとの格子空間で張られるボリュームデータSoriginal(x,y,z)と削減された格子空間で張られるデータ空間Ssimplified (x,y,z)との差

$$Err = \int_{all} \sqrt{\text{Soriginal}(x,y,z) - \text{Ssimplified}(x,y,z)}$$

で表現することが考えられる。Engelらは、このような誤差指標を使って、与えられたボリュームデータを階層型格子データに変換する手法を提案した。階層型格子データを用いることにより、ユーザは、段階レベルとスカラ値を指定して、現実的な時間で等値面を表示することができる。

ボリュームレンダリング表示

ボリュームレンダリングは、３次元空間で定義されたスカラデータをデータの分布にしたがって発光する一種の密度雲(発光粒子の集合体)として表示する手法である。ボリュームレンダリングでは、ディスプレー上の輝度値Bを求めるために、輝度値方程式(brightness equation)をディスプレー上の画素点毎に積分する。

$$B = \int_{tn}^{t0} c(t)\rho(t)\, e^{\int_{t}^{t0} \rho(\lambda) d\lambda} dt$$

ここで、tは、視線(画素点と視線を結んだもの)上にとられた変数で、t0は、与えられた３次元空間と視線との交差点のうち最も近いもの、そしてt_nは、最も遠いものを表すρ(t)は、単位長さ当たりの粒子数そしてc(t)は、単位粒子当たりの発光量を表す。式は、視線上から発せられた発光エネルギーが指数項で表される減衰を受けて視点に到達する様子を

示している。この式を解析的に計算することは一般的に不可能なので、基本的には区間(t0,tn)をn個の小区間に分けて数値積分を行なうと、

$$B = \sum_{1}^{n} ci\alpha i \prod_{j=1}^{i-1}(1-\alpha j)$$

となる。この計算を実現するアルゴリズムのひとつとして、スライスバイスライス法が提案されている。この手法では、視線に垂直な複数のスライス面上で不透明度付テクスチャを作成し、視点から見て奥から順にテクスチャを重ね合わせる。しかし、VRMLの仕様では、スライス面の描画順序を制御についての記述がないので、スライス面を奥から順に重ね合わせることが困難である。ひとつの解決策として単位粒子当たりの発光量を一定にすることを考える。この場合、式において、c_iは一定値(1とする)となるので、Bの計算結果は

$$B = \sum_{1}^{n} \alpha i \prod_{j=1}^{i-1}(1-\alpha j) = 1 - \prod_{j=1}^{n}(1-\alpha j)$$

となって、視線に沿ったサンプリング点の配置の順序には依存しなくなる。すなわち、スライス面を任意の順序で重ね合わせることができる。

最適化計算結果の可視化

最適化計算手法としてさまざまのものがあるが、本研究では、応答曲面法が採用されていると仮定する。この場合、評価関数は、多項式を用いた回帰曲面として表現されることが多い。最適化の対象であるパラメータのうちから２つを選択すれば、評価関数値を2次元格子上の高さデータとして表示するVRML ElevationGridノードを用いて表現することが可能となる。最適化の対象となるパラメータは、一般的に２より多いので、ユーザの指示（３つ以上のパラメータから２つを選択する）により高さデータが変更することになる。このため、ElevationGridノードにおいて高さデータを更新する記述法を調査した。その記述例を以下に示す。

```
#VRML V2.0 utf8
Group {
        children [
                Shape {
                        appearance Appearance {
                                material Material { }
                        }
                        geometry DEF SP1 Sphere {
                                radius 0.2
                        }
                },
                DEF SW TouchSensor{},
        # Fitness, initially empty
                Shape {
                        appearance Appearance {
                                material Material {
                                        diffuseColor 0.3 0.5 1
                                }
                        }
                        geometry DEF Fitness ElevationGrid {
                                xDimension   10
                                zDimension   10
                                xSpacing     0.500000
                                zSpacing     0.500000
                                solid        FALSE
                                creaseAngle 0.1
                                height [ ]
                        }
                },
        # Fitness maker
                DEF FitnessMaker Script {
```

```
        url "javascript:
        // Generate Height Field
        function calc_height( bool, eventTime ){
                if( bool == true ) {
                        if( ff + 1 < 3 ) {
                                ff = ff + 1;
                        } else {
                                ff = 0;
                        }
                        if( ff == 0 ) {
                                k[ 0 ] = 1;k[ 1 ] = 1;k[ 2 ] = 1;
                    k[ 3 ] = 1;k[ 4 ] = 1;k[ 5 ] = 1;
            } else
                        if( ff == 1 ) {
                                k[ 0 ] = 1;k[ 1 ] = -1;k[ 2 ] = -1;
                    k[ 3 ] = 1;k[ 4 ] = 1;k[ 5 ] = 1;
            } else
                        if( ff == 2 ) {
                                k[ 0 ] = 1;k[ 1 ] = 1;k[ 2 ] = 1;
                    k[ 3 ] = -1;k[ 4 ] = -1;k[ 5 ] = 1;
            }
                        calc_fit( k );
                }
        }
        function calc_fit( k ){
                for( i = 0; i < 10; i++ ){
                        for( j = 0; j < 10; j++ ){
                          x=0.1*i; y=0.1*j;
                                height_changed[10*i+j]=

k[0]+k[1]*x+k[2]*y+k[3]*x*x+k[4]*y*y+k[5]*x*y;
                        }
                }
        }"
            field       SFInt32 ff              0
            field       MFFloat k               [1,1,1,1,1,1]
```

```
                    eventIn   SFTime   set_time
                    eventIn   SFBool   calc_height
                    eventOut MFFloat height_changed
                }
        ]
}
ROUTE SW.isActive TO FitnessMaker.calc_height
ROUTE FitnessMaker.height_changed TO Fitness.set_height
```

シーン上に表示された白い玉をクリックすることにより、True と設定された変数 isActive と名付けられた Script ノードへ受け渡される。その後、Script ノードで定義された関数 calc_height が周期的に回帰曲面の係数 k[6] を変更する。このあと、関数 calc_fit が呼び出され、変更された係数 k[6] を使って高さデータ height_changed[100] を更新する。最終的に Fitness と名付けられた ElevationGrid ノードの高さフィールドにこの更新済み高さデータ height_changed[100] が受け渡される。この結果、 ElevationGrid ノードにより表現される回帰曲面の形状が変化する。

システム開発

我々は、WWW サーバに格納された計算流体力学シミュレーション結果をクライアント側で表示するためのプロトタイプシステムを開発した。ローエンドクライアント端末においても実用的な「可視化」を実現するには、サーバ側でできる限り幾何形状のデータ量を両面コンタ表示に限定し、データ付3角形メッシュを大きな1枚のテスクチャ付4角形で表現する手法を採用している。

図3はシステムの概要図である。クライアントでは WEB ブラウザ経由で現在サーバで保存されているボリュームデータの一覧を表示する。ユーザは、 Java アプレットをダウンロードし、断面の指定などに必要とされる可視化パラメータを入力する。入力されたパラメータは、サーバー側でロードされている Serviet に渡される。 Serviet は Java Native Interface (JNI)経由で C 言語で開発された可視化用 Native Module を呼び出す。このModule は、ボリュームデータからテスクチャ付4角形をVRML 形式で生成する。生成された VRML ファイルは、クライアントのブラウザ上で実行されている Java アプレットに転送され、そこから Extern Authoring Interface(EAI)を用いて VRML ビューワに受け渡される。

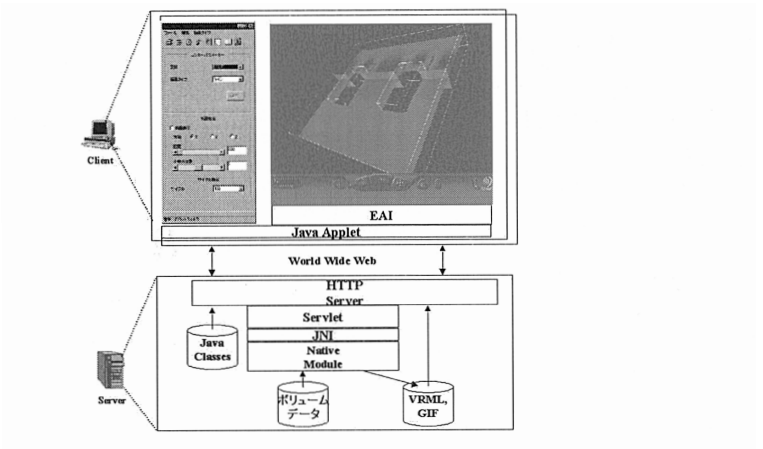

図3　VRMLを用いた可視化システムの概要

本手法の有効性を検証するために、直交格子で定義されたスカラー上から2通りの方法でスライス面を計算した。ひとつは、1つのテスクチャ付4角形を使って、そうしてもうひとつは、複数のカラーデータ付3角形を使ってスライス面を表現する。前者において、 GIF ファイルで表現されたテスクチャの解像度は、428*画素で、そのサイズは、1枚あたり9.87KB であった。 VRML ファイル中テスクチャ付4角形を1つ表現するのに、0.74KB、そしてその他の記述の部分

で、4.05KB 必要とした。

テスクチャを表現する GIF ファイルのサイズは、指定する解像度に依存するが、 VRML ファイル自身のサイズは一定となる以上から、N枚のスライス面を表現するために必要となるディスクスペース S は、以下のように計算される。

$$S = 4.05 + 10.61 \times N \ (KB)$$

スライス面をカラーデータ付3角形（頂点の総数が1246）で表現した場合、1枚のスライス面を表示するために150KB必要となる。

VRML ファイルを校正する Coordinate ノードの point フィールド Color ノードの color フィールド、そして頂点の接続情報を表す IndexedFaceSet ノードの coordIndex フィールドについては定義される3角形の数に比例すると考えて差し支えない。したがって、この場合、N枚のスライス面を表現するために必要となるディスクスペースは

$$S = 4.05 + 150.0 \times N \ (KB)$$

となる。したがって、以下のあげる理由によりディスクスペースの観点のおいて本手法が有利であることがわかる。

- 解像度を一定と仮定すると、スライス面を表現するテスクチャサイズは、3角形の数とは関係しない
- 現実のボリュームデータから計算されるスライス面は、今回の検証で得られた3角形より 多くの3角形から構成される

＜ 発　表　資　料 ＞

題　　名	掲載誌・学会名等	発表年月
VRMLを用いたCFD解析結果の可視化技術	日本機械学会論文集	2000年６月
Visualzation of Vortical Fiow Using Multiple Slices.	IEEE ICPAD-MMNS2000	2000年７月
数値風洞におけるレーザーライトシート 法の実験	日本シミュレーション学会論文誌	2000年９月