チュートリアル

最近では、情報爆発時代における不可欠な技術として情報可視化技術の開発や先進的利用が注目されています。今回は、高度な可視化技術として位置づけられる有限要素法向けボリュームレンダリングに関するチュートリアル（全4回シリーズ）の第4回目として、京都大学の坂本 尚久先生と小山田 耕二先生に解説していただきます。

可視化プログラミングの基礎（4）
タイルド表示装置を使った大規模分散ボリュームレンダリング

坂本 尚久
小山田 耕二

1　はじめに

有限要素法を使った流体解析や構造解析では、解析対象となる空間を四面体要素や六面体要素などを使った非構造型のボリュームデータとして表現することが多い。一方、計算機科学技術の進歩に伴い、シミュレーションモデルは複雑化・高分解能化していく傾向があり、それに応じてシミュレーション結果データも大規模化している。1台の計算ノードに集約することが困難であるような大規模分散並列環境で計算された有限要素解析結果のボリュームレンダリングを実現するには、効果的な分散化が重要な研究課題のひとつとして位置づけられている。また、大量の情報をもつ可視化結果から重要な知見を見逃さないために複数研究者向けに提示できる高精細表示環境の活用も重要視されている。

本チュートリアルでは、上述のような分散環境に保存された大規模非構造型ボリュームデータに対して、これまでに解説した粒子生成モジュール[1]およびGPUを利用した粒子レンダリングモジュール[2]を用いた分散粒子ベースボリュームレンダリングについて解説する。さらに、高精細表示環境実現例として、タイルドディスプレイ装置を利用した協調可視化環境の構築例を示す。

筆者紹介



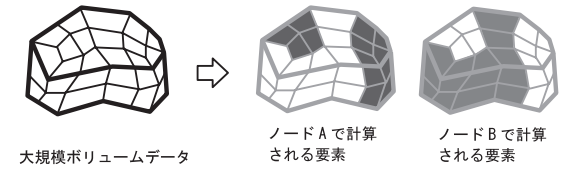
さかもと なおひさ
平成13年（株）ケイ・ジー・ティー入社。平成19年京大大学院工学研究科博士課程了。平成20年京都大学特定助教。情報可視化に関する研究に従事。博士（工学）。電子情報通信学会、可視化情報学会各会員。



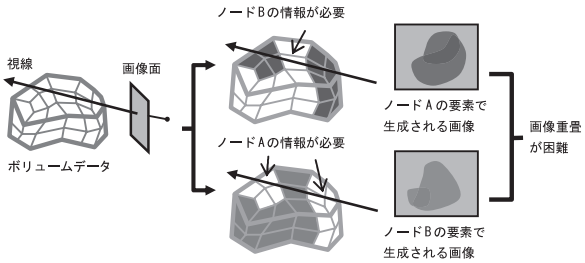
こやまだ こうじ
昭和60年京大大学院工学研究科了。同年日本IBM入社。平成10年岩手県立大学助教授、平成13年京都大学助教授、平成15年同教授。博士（工学）。情報可視化、設計最適化の研究に従事。IEEE CS、日本シミュレーション学会、情報処理学会各員。

2　大規模分散ボリュームレンダリング

分散並列環境で計算された有限要素解析結果は、図1（a）に示すように領域分割されていることが多い。有限差分法による解析結果であれば、それぞれの分割領域ごとにボリュームレンダリングを行い、部分画像を作成し、それらを視線に沿った順序で重ね合わせる方法が有効であるが、有限要素法による解析結果の場合は、その順序が確定しないケースが多い。図1（b）では、順序が決定されず、部分画像の重ねあわせが不可能となるケースを示す。



（a）領域ごとに分割された大規模ボリュームデータ



（b）分散レンダリング

図1　大規模ボリュームデータの分散可視化

この問題を回避するために、我々は、当初、規則格子へのサンプリング[3]を行っていたが、対象とするボリュームデータのメッシュ形状が不均一である場合は、サンプリングを行う格子間隔により、データの欠落やサイズが増大する可能性がある。また、複雑な形

状の場合は、その境界がうまく表現されないといった問題があった。本チュートリアルで説明した粒子ベースボリュームレンダリングを使えば、要素単位での処理が可能であり、かつ視線方向に沿った並べ替え計算も必要がないため、分散環境に保存されるボリュームデータに対しても、効率的にボリュームレンダリングを実現することが可能である(図2)。

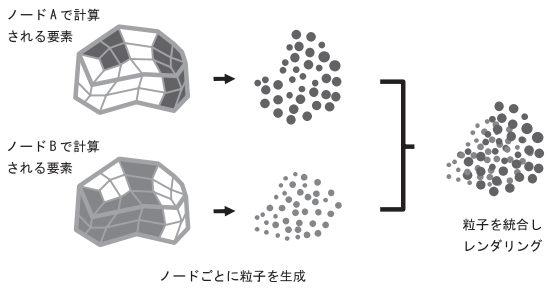


図2　PBVRによる分散可視化

3　分散粒子ベースボリュームレンダリング

粒子ベースボリュームレンダリング(PBVR)では、分散環境下に保存されるボリュームデータであっても、他の部分ボリュームを考慮することなく要素単位での粒子生成処理が可能であるため、効率よくかつ容易に分散レンダリングを行うことができる。本章では、PCクラスタシステムを利用した分散PBVRシステムについて説明する。

本チュートリアルでの分散PBVRシステムの構築には、1台の表示ノードと複数台の計算ノードからなるPCクラスタシステムを利用する。計算ノードは、保存されている部分ボリュームデータを読み込み、粒子生成を行うノードである。表示ノードは、各計算ノードで生成された粒子データを集約し可視化結果画像を表示するノードである。本システムは、次の4つの処理ステージから構成される(図3)。

(1) データ読み込み

本システムが対象とする非構造型ボリュームデータは、要素数、節点数、節点の座標値配列、数値データ、および要素を特定するための接点番号配列から構成される[1]。各計算ノードには、分割された部分ボリュームデータが格納されており、それぞれがひとつの非構造型ボリュームデータとして表現可能なものであるとする。以下に、本ステージでの処理のサンプルコードを示す。

```
void FileLoading(
    const std::string& filename,
    kvs::UnstructuredVolumeObject* volume )
{
    // ボリュームデータを読み込む(インポート)。
    volume = new kvs::UnstructuredVolumeImporter( filename );
}
```

(2) 粒子生成

本ステージでは、読み込んだ部分ボリュームデータに対して、チュートリアル第2稿で解説した粒子生成モジュール[1]を利用して要素ごとに粒子を生成する。

```
void ParticleGeneration(
    const size_t subpixel_level,
    const kvs::UnstructuredVolumeObject* volume,
    const float sampling_step,
    const kvs::TransferFunction& tfunc,
    PointObject* point )
{
    // 粒子を生成する。
    point = new kvs::CellByCellUniformSampling(
        volume, subpixel_level, sampling_step, tfunc );
}
```

(3) 粒子転送

生成された粒子を表示ノードに転送する。このとき、視線方向に沿った並べ替えが不要なため、他の計算ノードとの情報交換は必要なく、非同期的に処理を行うことが可能である。以下では、TCPによる簡単なソケット通信により、データ転送を行うサンプルコードを示す。

```
void ParticleTransmission(
    const kvs::PointObject* point,
    const kvs::IPAddress ip,
    const int port )
{
    // 表示ノードに接続する。
    kvs::TCPSocket client;
    client.open();
```

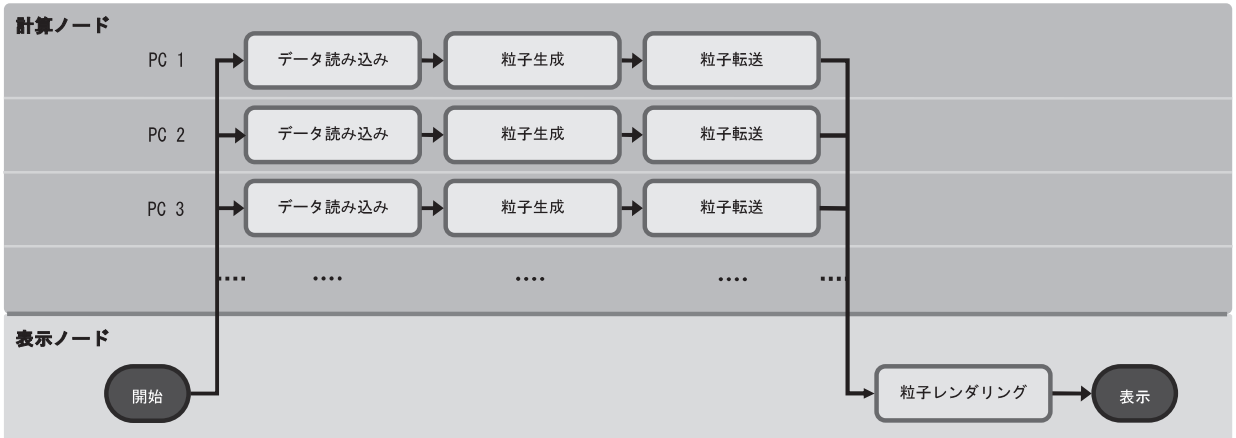


図3　PBVRによる分散可視化システムの処理の流れ

```
    client.connect( ip, port );

    // 転送用データを作成する。
    const int nparticles = point->nvertices(); // 粒子数
    const size_t coord_size = point->coords().byteSize();
    const size_t color_size = point->colors().byteSize();
    const size_t normal_size = point->normals().byteSize();
    const size_t data_size =
        sizeof(int) +
        coord_size +
        color_size +
        normal_size;
    char* data = new char [data_size];
    char* pdata = data;

    // 粒子数をコピーする。
    memcpy( pdata, &nparticles, sizeof(int) );
    pdata += sizeof(int);

    // 座標データをコピーする。
    memcpy( pdata, point->coords().pointer(), coord_size );
    pdata += coord_size;

    // 色データをコピーする。
    memcpy( pdata, point->colors().pointer(), color_size );
    pdata += color_size;

    // 法線データをコピーする。
    memcpy( pdata, point->normals().pointer(), normal_size );

    // 表示ノードにデータを転送する。
    client.send( data, data_size );
}
```

(4) 粒子レンダリング

表示ノードでは、各計算ノードから転送されてくる粒子データを結合し、チュートリアル第3稿で解説したGPUレンダリングモジュール[2]を利用して可視化する。以下に、粒子データを受信し結合する処理のサンプルコードを示す。

```
void ReceiveParticles(
     const int port,
     const int nprocs,
     kvs::PointObject* point )
{
    // 計算ノードからの接続を待つ。
    kvs::TCPServer server;
    server.open();
    server.bind( port );

    // データを受信し全粒子数を計算する。
    int nparticles = 0;
    kvs::MessageBlock message[ nprocs ];
    for ( int i = 0; i < nprocs; i++ )
    {
        kvs::PointObject partial_particles;
        kvs::TCPSocket* socket = server.checkForNewConnection();
        socket->receive( &message[i] );

        nparticles += *((int*)(message[i].pointer()));
        delete socket;
    }

    server.close();

    // 受信したデータを結合し1つの粒子群(kvs::PointObject)を生成する。
    kvs::ValueArray<float> coords( nparticles * 3 );
    kvs::ValueArray<unsigned char> colors( nparticles * 3 );
    kvs::ValueArray<float> normals( nparticles * 3 );

    float* sub_coords = coords.pointer();
    unsigned char* sub_colors = colors.pointer();
    float* sub_normals = normals.pointer();
    for ( int i = 0; i < nprocs; i++ )
    {
        char* block = (char*)message[i].pointer();

        // 粒子数を取得する。
        int sub_nparticles = *((int*)(block));
        block += sizeof(int);

        int element_size = sub_nparticles * 3;

        // 座標データを取得する。
        int byte_size = element_size * sizeof(float);
        memcpy( sub_coords, block, byte_size );
        sub_coords += element_size; block += byte_size;

        // 色データを取得する。
        byte_size = element_size * sizeof(unsigned char);
        memcpy( sub_colors, block, byte_size );
        sub_colors += element_size; block += byte_size;

        // 法線データを取得する。
        byte_size = element_size * sizeof(float);
        memcpy( sub_normals, block, byte_size );
        sub_normals += element_size;
    }

    point = new kvs::PointObject( coords, colors, normals );
}
```

4 分散可視化システムの実装

本章では、可視化基盤ライブラリKVSを利用した簡単な分散可視化システムを構築する。前章で示したサンプルコード内で「kvs::」が付いているクラス(名前空間kvs内で定義されているクラス)はすべてKVSに実装されているクラスであることを示している。また、PBVRを利用した分散可視化の方法については、奥行き値をもったサブピクセルを伝送する方式[5]でも計算可能であるが、本チュートリアルでは、計算ノードで生成される粒子データを単純に表示ノードに伝送するだけの仕様とした。そのため、データ伝送にはTCPによるソケット通信を利用したクライアント・サーバ方式のシステムとして構築する。通信部分については、実行環境などに応じて、MPIなど既存の通信ライブラリを利用することで頑健性や高速性に優れたデータ伝送を実現することも可能である。

4.1 計算ノード上での処理

計算ノード上で実行されるプログラムは、前章で述べたステージ1、2および3の処理からなり、サーバプログラムとして機能する。以下にソースコードを示す。

```
#include <string>
#include <kvs/PointObject>
#include <kvs/UnstructuredVolumeObject>
#include <kvs/UnstructuredVolumeImporter>
#include <kvs/CellByCellUniformSampling>
#include <kvs/IPAddress>
#include <kvs/TCPServer>
#include <kvs/TCPSocket>
#include <kvs/MessageBlock>


int main( int argc, char** argv )
{
    // 引数で指定されるパラメータ(入力パラメータ)を取得する。
    //      第1引数:表示ノードのIPアドレス
    //      第2引数:接続ポート番号
    //      第3引数:サブピクセルレベル
    //      第4引数:サンプリングステップ長
    //      第5引数:ボリュームデータファイル名
    //      第6引数:伝達関数ファイル名
    kvs::IPAddress ip( argv[1] );
    int port = atoi( argv[2] );
    int subpixel_level = atoi( argv[3] );
    float sampling_step = atof( argv[4] );
    std::string data_filename = std::string( argv[5] );
    std::string tf_filename = std::string( argv[6] );

    // 表示ノードからの接続を待つ。
```

```
    kvs::TCPServer server;
    server.open();
    server.bind( port );
    server.listen();

    kvs::MessageBlock message;
    server.receive( &message );
    if ( message.toString() != "START" ) return( 1 );

    // (1) データ読み込み
    kvs::UnstructuredVolumeObject* volume = NULL;
    FileLoading( data_filename, volume );

    // ※本プログラムでは、簡単化のため、あらかじめボリュームデータの数値
    //   データの最大・最小値は計算しておくこととし、以下の関数を利用して
    //   その値を取得できるものとする。
    const double min = GetMinValues();
    const double max = GetMaxValues();
    volume->setMinMaxValues( min, max );

    // (2) 粒子生成
    kvs::PointObject* point = NULL;
    kvs::TransferFunction tfunc( tf_filename );
    ParticleGeneration( volume,
                        subpixel_level,
                        sampling_step,
                        tfunc,
                        point );

    // (3) 粒子転送
    ParticleTransmission( point, ip, port );

    delete volume;
    delete point;

    return( 0 );
}
```

4.2 表示ノード上での処理

表示ノード上では、前章で述べたステージ4の処理が実行され、クライアントプログラムとして、受信する粒子データを表示する機能を持つ。今回は、指定される伝達関数(カラーマップ)をビューワ上に表示する。そのため、前回までは、kvs::glut::Globalクラスおよびkvs::glut::Screenクラスを利用して可視化アプリケーション開発を行っていたが、今回は、それらのベースとして実装されているkvs::glut::GlobalBaseクラスおよびkvs::glut::ScreenBaseクラスを利用し、可視化アプリケーションの開発を行う。以下にソースコードを示す。

```
#include <cstdio>
#include <vector>
#include <string>
#include <kvs/TransferFunction>
#include <kvs/SocketAddress>
#include <kvs/Math>
#include <kvs/TCPServer>
#include <kvs/TCPSocket>
#include <kvs/MessageBlock>
#include <kvs/PointObject>
#include <kvs/glut/GlobalBase>
#include <kvs/glut/ScreenBase>
#include <kvs/glut/LegendBar>
#include <kvs/glew/ParticleVolumeRenderer>


// 大域変数を格納するためのクラス
class Global : public kvs::glut::GlobalBase
{
public:

    static std::vector<kvs::SocketAddress> ip_list;
    static int port;
    static int subpixel_level;
    static kvs::TransferFunction tfunc;
    static kvs::glut::LegendBar legend;

    Global( int argc, char** argv ):
        kvs::glut::GlobalBase( argc, argv )
    {
        // IPアドレスファイルを読み込む
        //    1:計算ノード1のIPアドレス:ポート番号
        //    2:計算ノード2のIPアドレス:ポート番号
        //    ...
        std::string filename = std::string( argv[1] );
        FILE* fp = fopen( filename.c_str(), "r" );
        char buf[32];
        while ( fgets( buf, 32, fp ) != NULL )
        {
            kvs::SocketAddress ip( buf );
            ip_list.push_back( ip );
        }

        // 接続ポートの読み込み
        port = atoi( argv[2] );

        // サブピクセルレベルの読み込み
        subpixel_level = atoi( argv[3] );

        // 伝達関数ファイルの読み込み
        tfunc.read( std::string( argv[4] ) );
    }
};

std::vector<kvs::SocketAddress> Global::ip_list;
int Global::port;
int Global::subpixel_level;
kvs::TransferFunction Global::tfunc;
kvs::glut::LegendBar Global::legend;

// スクリーンクラス
class Screen : public kvs::glut::ScreenBase
{
public:

    Screen( void )
    {
        // 初期化関数(init)と描画イベント(paint)を登録する。
        addInitializeFunc( init );
        addPaintEvent( paint );
    }

    // 初期化関数
    static void init_func( void )
    {
        // 計算ノードに接続し処理開始のメッセージを伝送する。
        for ( unsigned int i = 0; i < Global::ip_list.size(); i++ )
        {
            kvs::TCPSocket client;
            client.open();
            client.connect( Global::ip_list[i].ip(),
                            Global::ip_list[i].port() );

            kvs::MessageBlock message("START");
            client.send( message );
        }

        // 粒子データを受信し結合する。
        kvs::PointObject* point = NULL;
        ReceiveParticles( Global::port,
                          Global::ip_list.size(),
                          point );

        // (4) 粒子レンダリン
        //     粒子レンダリング用のレンダラを準備する。
        int subpixel_level = 1;
        int repeat_level =
                    kvs::Math::Square( Global::subpixel_level );
        kvs::glew::ParticleVolumeRenderer* rend =
            new kvs::glew::ParticleVolumeRenderer(
                point, subpixel_level, repeat_level );

        // データとレンダラを関連付ける。
        int obj_id = Global::object_manager->insert( point );
        int rend_id = Global::renderer_manager->insert( rend );
        Global::id_manager->insert( obj_id, rend_id );

        // 表示用カラーマップを準備する。
        // ※本プログラムでは、簡単化のため、あらかじめボリュームデータの数値
        //   データの最大・最小値は計算しておくこととし、以下の関数を利用して
        //   その値を取得できるものとする。また、数値データ名もあらかじめ設定
        //   しておくものとする。
```

```
        const double min = GetMinValues();
        const double max = GetMaxValues();
        const std::string name = GetValueName();
        Global::legend.setTitle( name );
        Global::legend.setMinValue( min );
        Global::legend.setMaxValue( max );
        Global::legend.setColorMap( Global::tfunc.colorMap() );
    }

    // 描画イベント
    static void paint ( void )
    {
        // カラーマップを表する。
        const int x = 40;
        const int y = 20;
        const int width = 150;
        const int height = 20;
        Global::legend.setPosition( x, y );
        Global::legend.setSize( width, height );
        Global::legend.draw();
    }
};

int main( int argc, char** argv )
{
    // 引数で指定されるパラメータを取得する。
    //      第1引数: 計算ノードのIPアドレスリストのファイル名
    //      第2引数:接続ポート
    //      第3引数:サブピクセルレベル
    //      第4引数:伝達関数ファイル名

    Global* global = new Global( argc, argv );

    Screen* screen = new Screen();
    screen->setGeometry( 0, 0, 800, 600 );
    screen->show();

    delete global;
    delete screen;

    return( 0 );
}
```

4.3 実験

本システムを利用して、大規模口腔内気流シミュレーション結果と大規模ポンプ自重解析結果の可視化を行った。実験では、1台の表示ノードと8台の計算ノードからなるPCクラスタシステムを利用した(表1)。

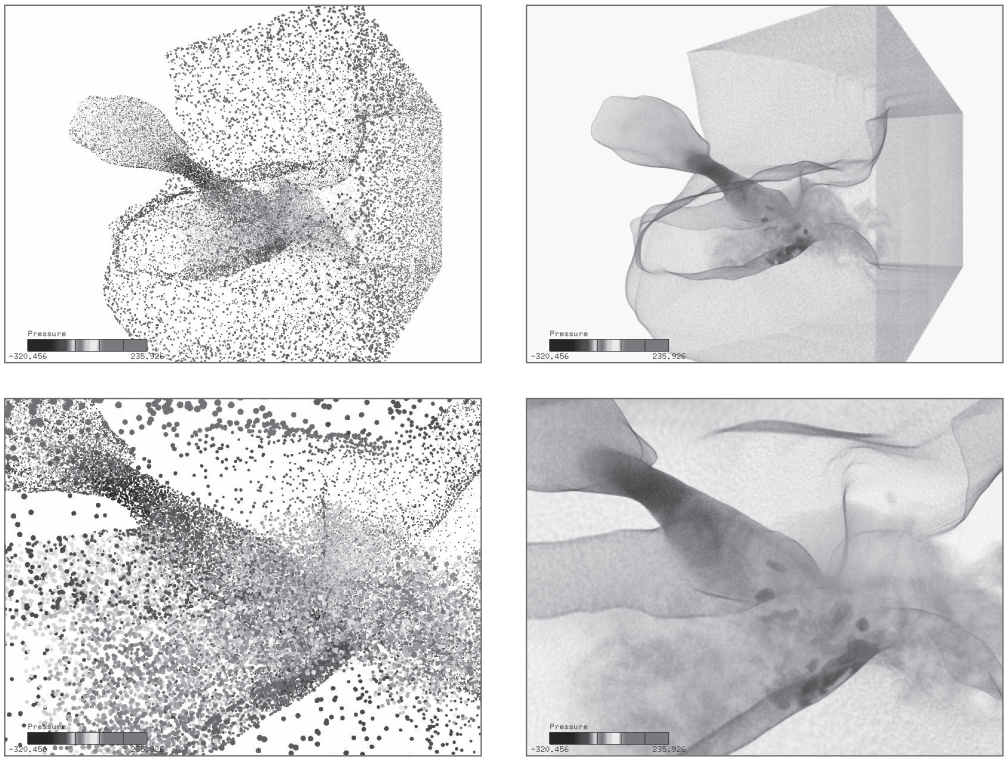
表1 利用したPCの仕様

表示ノード(1台)	
CPU	Intel Core 2 Duo 2.66 GHz
RAM	2.0 GB
GPU	NVIDIA GeForce 8500 GT 256 MB
計算ノード(A×6台、B×2台)	
A	
CPU	Intel Core 2 Duo 1.86 GHz
RAM	2.0 GB
GPU	NVIDIA GeForce 8500 GT 256 MB
B	
CPU	Intel Core 2Duo 2.2 GHz
RAM	2.0 GB
GPU	NVIDIA GeForce 8500 GT 256 MB

(1) 大規模口腔内気流シミュレーション結果への適用

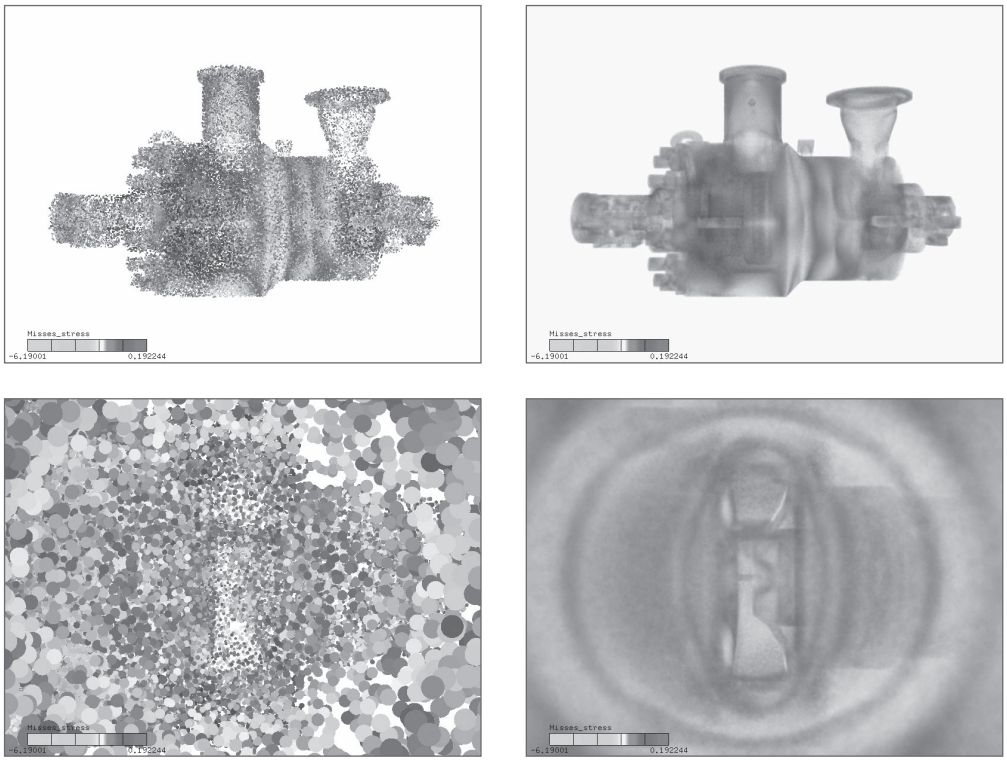
シミュレーションは、LES(Large Eddy Simulation)に基づき流体解析ソフトウェアFrontFlow/Blue[4]を利用して計算されており、7,215万個の六面体1次要素からなる非構造型ボリュームデータとして解析結果が出力される。データは16領域に分割され、並列に計算され結果がそれぞれ出力される。

可視化結果を図4に示す。実験では、16分割された



(a) リピートレベル1で描画(2.1〜5.8 frames per second)　(b) リピートレベル144で描画(0.8〜1.9 frames per second)

図4 大規模口腔内気流シミュレーション結果(圧力分布)の可視化(粒子数:683万点、粒子生成時間:22.3秒)
(粒子数683万点のうち384万点は口腔形状を表す粒子でありあらかじめ生成されたものであり、その処理時間は上記の粒子生成時間には含まれていない。)



(a) リピートレベル1で描画(1.2〜3.3 frames per second)　(b) リピートレベル144で描画(0.6〜0.9 frames per second)

図5　大規模ポンプ自重解析結果の可視化(粒子数：1,500万点、粒子生成時間：44.2秒)

データを8台の計算ノードで処理するために、1計算ノードあたり2つのデータを処理することで分散可視化を行った。さらに、シミュレーションの対象となる口腔内形状モデル(コーンビームCTで得られた画像から等値面生成処理を行い生成されるメッシュデータ)を表示ノードで読み込み、計算された圧力分布との合成表示を行った。本システムを利用することによって、大規模非構造型ボリュームデータに対して高速に可視化処理を行うことができ、さらに、口腔内形状モデルを粒子群で表すことで半透明効果が得られ、口腔内の空間的な気流の圧力分布を直観的に解析することができる。

(2) 大規模ポンプ自重解析結果への適用

並列有限要素解析ソフトウェアFrontSTR[4]を利用して計算された原子炉給水ポンプの自重解析結果に対して本システムを適用した。本実験で利用したポンプデータは、16の領域に分割されており、2,629万個の四面体2次要素からなる非構造型ボリュームデータとして表現されている。

可視化結果を図5に示す。実験では、1計算ノードあたり2つの分割領域を読み込み、処理を行った。これまでは、データが大規模であったため断面コンターや表面ポリゴンに対して値をマッピングして可視化を行っていたが、本システムを利用してボリュームレンダリングすることによって、データ内部のデータ値の分布を空間的に解析することが可能となる。また、LOD制御を行うことで、リアルタイムに可視化結果の視点を変更することが可能であり、対話的な速度での解析も可能となる。

5　タイルドディスプレイ装置を用いた協調可視化環境の構築

大規模有限要素解析結果を解析する段階において、そこから新たな知見を発見するためには、複数の研究者たちが一堂に会し、可視化結果をもとに議論できる場が必要であり、そのような環境を構築するためには複雑な視覚情報をより分かりやすく可視化する大規模表示システムが有効であるとされる。本チュートリアルでは、LCDディスプレイやプロジェクタなどを複数並べることによって、安価にスケーラブルな大規模表示環境を構築することができるタイルドディスプレイ装置(TDW, Tiled-Display Wall)を用いた協調可視化環境[6]の構築方法について説明する。

5.1　SAGE

タイルドディスプレイ装置向けの基盤ソフトウェア環境としてはカリフォルニア大学サンディエゴ校で開発されているCGLX(Cross-Platform Cluster Graphics Library)[7]やソースコードも公開されGLR(GL DLL Replacement)機能を利用して容易にタイルドディスプレイ向け表示が可能なChromium[8]など様々環境が提供されているが、ここではイリノイ大学EVLで開発されているSAGE(Scalable

Adaptive Graphics Environment)[9]を利用したシステム構築例を示す。

SAGEは、複数台のレンダリング用PC上に仮想の高解像度フレームバッファを構築し、リモートから転送される2次元映像をピクセル情報のストリームとして適切に制御することによって、タイルドディスプレイ上の任意の位置に任意の大きさで描画することができる。

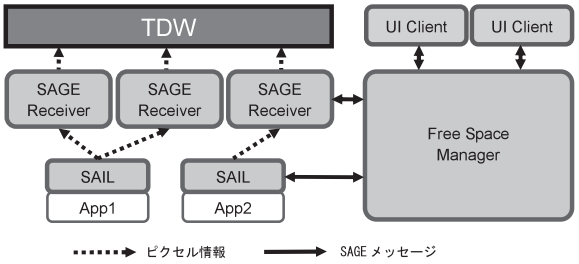


図6　SAGE (Scalable Adaptive Graphics Environment)

SAGEの簡単な構成を図6に示す。タイルドディスプレイに接続されるレンダリング用PC上では、SAGE Receiverが起動しており、伝送されてくる複数のピクセル情報を受け取り担当ディスプレイに表示する。利用者が開発するアプリケーションは、SAIL (SAGE Application Interface Library) を利用してSAGE Receiverに対してピクセル情報を送信する。このとき、FreeSpace ManagerがSAIL-SAGE Receiver間のピクセル情報を制御することによって、表示位置と大きさなどの管理を行っている。

5.2　構築例

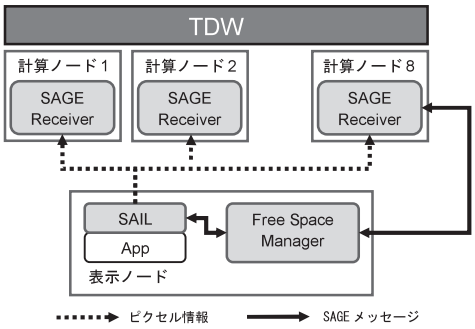
我々の研究室では、前章で述べたPCクラスタシステムの各計算ノードにそれぞれ2台のLCDディスプレイを接続し、4×4 (16面) の実験用タイルドディスプレイ装置を構築している (図7)。本実験では、表示ノードにFree Space Managerを動作させ、表示ノードでレンダリングされた可視化結果をピクセル情報としてSAILを利用し再度計算ノードに伝送することによってTDWに表示する。

通常、SAGEを利用したTDW向けの可視化ソフトウェアを開発する場合、SAGEが提供するAPI (Application Program Interface) を利用して、フレームバッファに描かれたピクセル情報を読み込み (Read-back)、SAGE Receiverに伝送する処理を書き加える必要がある。KVSでは、SAGEをサポートしており、最小限の書き換えで、TDWへ可視化結果を出力することが可能である。

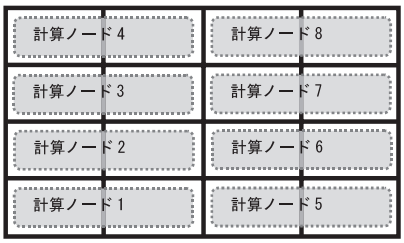
前章で示した分散可視化システムの表示ノード上で実行されるプログラムを以下の手順に従い書き換えることにより、簡単にTDW向けの可視化システムを構築することができる。



(a) タイルドディスプレイ装置とPCクラスタ



(b) システム構成



(c) 4x4タイルドディスプレイ

図7　タイルドディスプレイ装置

手順1　SAGE用のクラスを利用するためのヘッダファイルをインクルードする (ヘッダファイルのディレクトリ名をglutからsageに変更する)。

```
#include <kvs/sage/GlobalBase>
#include <kvs/sage/ScreenBase>
```

手順2　SAGE用のGlobalBaseクラスとScreenBaseクラスを継承する (名前空間をglutからsageに変更する)。

```
class Global : public kvs::sage::GlobalBase
{
...
    Global( int argc, char** argv ):
        kvs::sage::GlobalBase( argc, argv )
...
};

Class Screen : public kvs::sage::ScreenBase
{
...
}
```

本システムを利用してTDW上に可視化結果を表示し、解析結果に対する議論を行っている様子を図8に示す。大規模表示装置を利用することで、複数人でデータ全体を概観することが可能であり、かつそれぞれに異なる興味領域を同時に観察することもできるため、効果的な議論を行うことが期待できる。



図8　タイルドディスプレイ上に可視化された大規模ポンプ自重解析結果(ミゼス応力分布)について議論をしている様子

6　おわりに

本チュートリアルでは、PBVRを用いた大規模非構造格子データ向けの分散可視化システムについて解説し、簡単なサンプルコードを示しながらその具体的な実装方法について説明した。さらに、タイルドディスプレイ装置(TDW)を利用した協調可視化環境の構築方法についても説明し、SAGEを利用したTDW向けの可視化システムの構築方法について述べた。実際にリアルタイムでレンダリングしている様子は、http://www.youtube.com/user/kyotoviz で確認することができる。また、チュートリアルを通して利用してきた可視化基盤ソフトウェアKVSは、http://code.google.com/p/kvs でオープンソースとして公開されており、自由にダウンロードして利用することが可能である。

本チュートリアルでは、有限要素法プログラムの経験者がとっかかりやすいように解説を試みたが、いかがだったであろうか？尚、ボリュームレンダリング画像生成における重要な役割を果たす特異点探索法や時系列データ処理のためのプログラミングについては紙面が足らず、割愛した。機会があれば、ベクタデータ可視化プログラミングとあわせ解説できればと思っている。会員の皆様の忌憚のない意見を伺えれば幸いである。

1970年代ノーベル経済学賞受賞の学者ハーバート・サイモンが「情報の豊富さは注意の貧困を生み出す」と予言した状況がいたるところで発生している。このような情報過剰の時代にあって、泥から砂金を探し当てる可視化技術は多くの分野で注目されており、特に計算工学研究の大きな柱と位置づけられるものと考えている。可視化技術は、スパコンや計測機器から出力される大量の情報をわかりやすく映像化し、シミュレーションモデルの理解・新しい振る舞いの発見・新しい可視化技術の発明を促すきっかけを与えると期待できる。欧米では、可視化技術が博士号取得の研究分野になっているにも関わらず、残念ながら日本では研究分野としての認知度がまだまだ低いのが現状である。これを機会に計算工学会員のなかから新進気鋭の可視化技術研究者が現れることを期待している。

謝辞

口腔気流シミュレーションデータを提供していただいた大阪大学サイバーメディアセンター野崎一徳氏と原子炉給水ポンプデータを提供していただいた東京大学人工物工学研究センター奥田洋司教授に深く感謝する。

■参考文献

[1] 坂本尚久、小山田耕二、可視化プログラミングの基礎(2)粒子ベースボリュームレンダリング、計算工学会、Vol.14, No.1, pp.40-46,(2009).

[2] 坂本尚久、小山田耕二、可視化プログラミングの基礎(3) GPUを利用した粒子ベースボリュームレンダリングの高速化、計算工学会、Vol.14, No.2, pp.24-31,(2009).

[3] N. Sakamoto, K. Koyamada, K. Sakai, M. Kikugawa, Voxelization of Hexahedral Cell with the Two-Pass Rasterization Technique, The 4th IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP2004), pp.178-181,(2004).

[4] 文部科学省次世代IT基盤構築のための研究開発「革新的シミュレーションソフトウェアの研究開発」プロジェクト, http://www.rss21.iis.u-tokyo.ac.jp/

[5] N. Sakamoto, H. Kuwano, T. Kawamura, K. Koyamada, and K. Nozaki, Visualization of Large-scale CFD Simulation Results Using Distributed Particle-Based Volume Rendering, International Journal of Emerging Multidisciplinary Fluid Sciences, to appear,(2009).

[6] H. Kuwano, T. Kawamura, N. Sakamoto, K. Koyamada, and K. Nozaki, A Collaborative Visualization System for Complex CFD Results on a Tiled Display Wall, The 3rd International Workshop on Process Tomography (IWPT-3), CD-ROM,(2009).

[7] CGLX, http://vis.ucsd.edu/~cglx/

[8] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, Chromium：a stream-processing framework for interactive rendering on clusters, International Conference on Computer Graphics and Interactive Techniques, pp.693-702,(2002).

[9] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, and J. Leigh, High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment, Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, pp.24,(2006).