2003－ARC－154　(11)
2003／8／5

汎用グラフィクスハードウェアを用いた並列ボリュームレンダリングの実装

丸山悠樹[†]　中田智史[†]　高山征大[†]
津邑公暁[†]　五島正裕[†]　森眞一郎[†]
中島康彦[†]　富田眞治[†]

本稿では汎用グラフィクスハードウェアを用いて, 並列ボリュームレンダリングを行うシステムの実装について報告する. システムではさらに大規模かつ高速な描画を行うために, 適応的サンプリングによるデータの圧縮と, 中間画像の圧縮による通信時間の削減を行った. この結果, $512 \times 512 \times 1024$ のボリュームデータに対し, グラフィクスカード 4 台を用いたシステムにおいて 40 倍の速度向上が得られ, ほぼ実時間でのレンダリングが可能となった.

Parallel Volume Rendering with Standard Graphics Hardware

Yuki Maruyama ,[†] Satoshi Nakata ,[†] Motohiro Takayama,[†]
Tomoaki Tsumura ,[†] Masahiro Goshima ,[†] Shinichirou Mori ,[†]
Yasuhiko Nakashima[†] and Shinji Tomita [†]

In this paper, we report implementation of the system which performs parallel volume rendering using standard graphics hardware. We performed compression of the data based on adaptatively sampling and curtailment of communication time by compression of subimage to perform more larger scale and more higher speed drawing. Consequently in this system. Consequently, it was able to get about forty times speed up and to draw in real time in the system using using four standard graphics hardwares.

1. はじめに

近年の計算機処理能力の向上による大規模シミュレーションシステムの実用化に伴ない, より大規模な 3 次元データの解析を支援する可視化システムの実用化が求められている. このような大規模な 3 次元データの解析を支援する可視化方法の一つとしてボリュームレンダリングが挙げられる. ボリュームレンダリングを用いることにより, 複雑な 3 次元構造の理解が容易となるため, 工学, 医学などの分野で幅広く利用されている. しかし, 膨大な計算量が必要とされるため, 専用ハードウェアを用いる場合を除き, リアルタイムに可視化することは一般に困難であった. しかし, 汎用グラフィクスハードウェアの機能の向上とともに, その機能を利用してボリュームレンダリングを行うことが可能となってきており, これを並列化して用いることで大規模なデータの可視化が可能となる.

本稿では RADEON9700PRO を用いた並列ボリュームレンダリングシステムの実装を行った. 以下, 2 章で研究の背景となるボリュームレンダリングについて説明し, 3 章で汎用グラフィクスハードウェアによるボリュームレンダリング手法について述べる. 4 章では並列化手法について述べ, 5 章で具体的な実装について述べた後, 6 章で評価を, 7 章で結論を述べる.

† 京都大学
Kyoto University

2. Volume Rendering

我々が対象としているボリュームレンダリングは 3 次元のスカラー場をボクセルの集合として表現し, 2 次元平面へ投影することにより, 複雑な内部構造や動的特性を可視化する手法である. ボリュームレンダリングは大別して, すべてのサンプル点の寄与を計算して全体を表示する直接法と, 前処理によって表示する情報を抽出してデータの一部分を表示する間接法の二種類に分類され, 通常ボリュームレンダリングという場合は直接法のことを指す. 直接法によるボリュームレンダリングでは対象空間内のボクセルすべての寄与を計算して 2 次元平面へ投影する. このため表示像が正確であり, はっきりとした境界を持たない雲や炎といった自然現象やエネルギー場の可視化に適用できるという特徴をもつ. このようにボリュームレンダリングを用いると, 複雑な 3 次元構造の理解が容易となるため, 工学, 医学などの分野で幅広く利用されている.

しかし, 膨大な計算時間と記憶容量が必要とされ, 大型計算機や特殊ハードウェアを用いる場合に利用が限られており, リアルタイムに可視化することは一般に困難であった.

3. Texture Based Volume Rendering

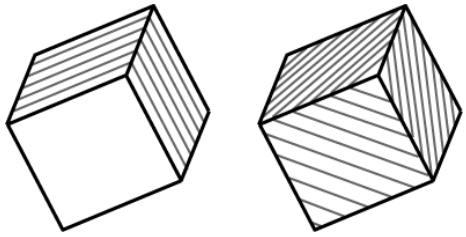
ボリュームレンダリングは描画面の各画素から視線方向に沿ってボクセル値の持つ色情報を積分していく. これを離散化すると, スクリーン上の各ピクセルごとに発生する視線に沿って, 視線と交差するボクセル値のサンプリングを視線上のボクセルがなくなるまで繰り返し, ピクセル値を求めることになる. この方法は視点から近い順にサンプリングする方法 (front to back) と, 視点から遠い順にサンプリングする方法 (back to front) に分けられる. back to front の場合, ボクセルの値を視点に近い順から, v_0, v_1, $\cdots$, v_n とし, RGB の各色情報 c_k と不透明度 α_k がボクセル値 v_k の関数で表されるとすると, ピクセル値は

$$P = \sum_{i=0}^{n} \alpha(v_i)c(v_i) \prod_{j=0}^{i-1}(1 - \alpha(v_j)) \qquad (1)$$

と表される. このピクセル値計算式は累積値 C_k を用いて次式のような漸化式に変形される.

$$C_{k-1} = \alpha(v_i)c(v_k) + (1 - \alpha(v_i))C_k \qquad (2)$$

ここで $P = C_0$ である. 式 (2) はボリュームのサンプリングを視線方向に従って一定のサンプリングで行い, 描画面に遠い方から順に RGB 値を α ブレンディングすることでボリュームレンダリングができることを示している. α ブレンディングとは二枚の画像の持つ RGB 値を, α 値の示す比率で線形内挿して, 合成画像の RGB 値を算出する方法である. ボリュームをある軸に対して垂直なスライスの重ね合わせで表現する. そのスライスをテクスチャとしてポリゴンにマッピングし, それらを視点から遠い順に順次 α ブレンディングすることでボリュームレンダリングを行う. この手法を用いることで, 汎用グラフィクスハードウェアの機能を利用した高速処理が可能である[1]. グラフィクスハードウェアが 3 次元テクスチャをサポートしている場合には, 視線に対して垂直な面を用意し, ボリュームデータを 3 次元テクスチャとして扱う方法が可能である (図 1-(b)). 3 次元テクスチャが利用できない場合は, ボリュームデータを各座標軸に対して垂直なスライスとして 3 つ用意し, 視線方向とスライスの法線のなす角が一番小さい軸のスライスに対して, 2 次元テクスチャとしてマップする方法を用いる (図 1-(a)). 本稿で用いるグラフィクスハードウェア RADEON9700PRO(表 1) は 3 次元テクスチャをサポートしているおり, ボクセル値を 8bit の RGBA データとして描画した場合, $256 \times 256 \times 256$ のボリュームデータをリアルタイムに描画することができる. したがって, 本稿では 3 次元テクスチャによるボリュームレンダリングを扱うことにする.



(a) 2 次元テクスチャ　(b) 3 次元テクスチャ

図 1 テクスチャベースのボリュームレンダリング

しかし, さらに大きいサイズのボリュームデータを描画する場合, 演算速度, メモリバンド幅, メモリ容量の制限などがボトルネックとなり, 描画速度が低下する. RADEON9700PRO の場合, Core Clock: 325MHz, Pipeline Unit: 8, Memory Pipeline: 310MHz DDR, Memory: 256bit 128MB という仕様である. 一番のボトルネックはメモリ容量で $256 \times 256 \times 256$ のボリュームデータを描画することが限界であるが, 仮にメモリ容量の制限がなかったとしても, 演算速度は 2.6Gpixel/sec, メモリバンド幅は 19.8GB/sec であるため, 1024^3 のボリュームデータの場合, たかだか 5fps 程度しか描画できないということがわかる. このように, 現状の汎用グラフィクスハードウェアの性能は大規模なボリュームデータの描画を行うにはまだまだ性能が不足しており, 複数の汎用グラフィクスハードウェアを用いた並列化を行う必要がある.

4. 並 列 化

汎用グラフィクスハードウェアのテクスチャマップ機能と α ブレンディングを用いることにより, ボリュームレンダリングの高速処理が可能になる. しかし, テクスチャを保持するグラフィクスハードウェアのメモリ容量には制限があるため, メモリ容量を上回るサイズのデータを描画することはできない. 描画するデータがグラフィクスハードウェアのメモリ容量を超えない大きさのテクスチャに分割して描画させることにより, そのままでは描画できない大きなサイズのデータを描画することが可能となる. しかし, 保持しているテクスチャデータのサイズがグラフィクスメモリの容量を超える場合, テクスチャデータはメインメモリーに格納され, テクスチャは描画に使われる度にグラフィクスカードに転送される. このため, テクスチャデータの転送時間がボトルネックとなり描画速度が急激に低下する. そこで複数の汎用グラフィクスハードウェ

アを用いて並列化を行う．これによりデータサイズの大きなボリュームデータを扱うことができる[2]．

4.1 基本方針

まず，$N_x \times N_y \times N_z$ のボリュームデータを x, y, z 方向に d 等分に分割し，P 台のノードそれぞれで発生させた d^3/P 個のプロセスにそれぞれを割り当てる．各サブボリュームをそれぞれのグラフィクスカードに与えてボリュームレンダリングを行い，中間画像を生成する (図 2)．このようにして生成された d^3 枚の中間画像を 2 次元テクスチャとして扱い，視点からの距離の遠いものから α ブレンドして一つの画像にまとめることにより，最終結果を得る (図 3)．

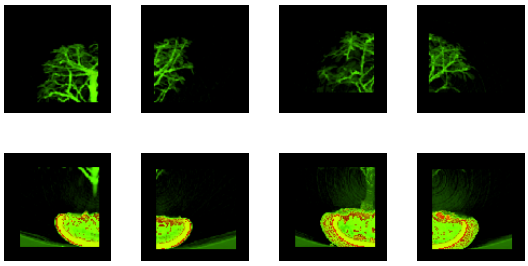


図 2 中間画像 ($d = 2$)

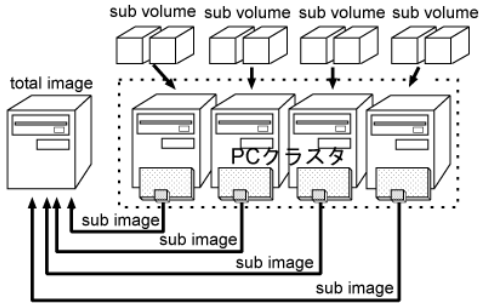


図 3 並列化

4.2 適応的サンプリング

並列化により大きなデータを扱うことができるが，さらに大きなデータを扱う方法として，ボリュームデータをブロックで分割し，ブロック内のデータの局所性を利用して，データ量を削減することでメモリ利用の効率化を行う適応的サンプリングという方法が提案されている[3]．この手法を用いることにより，ほとんど画質の劣化させることなく，データ量を大幅に削減することができる．

$N_x \times N_y \times N_z$ のボリュームデータを x, y, z 方向それぞれに b 等分に分割した場合を考える．ただし，N_x/b, N_y/b, N_z/b は，OpenGL の制約上，2 のべき乗である必要がある．このようにして分割された各ブロックに対して，x, y, z 方向それぞれのデータサイズを 1/2 のサイズとすることで解像度を順に一つずつ減少させ，その際に生じる元のデータとの誤差を計算していく．解像度を下げる前の各ブロックのサンプル点のうち，解像度を一つ下げたときの各ブロックのサンプル点に含まれるボクセル値の平均値を，解像度を一つ下げたときのボクセル値と定義する．また，誤差はサンプリングした値との自乗誤差と定義する．このようにして解像度の変更された各ブロックを視点からの距離の遠いブロックから順に描画していく．図 4 に 2 次元平面に対して行った適応的サンプリングの例を示す．ブロック全体の誤差が 0 の場合には画質を全く変えずに，データ量を削減することができる．全てのブロックで誤差計算を行い，誤差が許容値以下であるブロックの解像度を下げていく．この操作を繰り返すことで，全体としてデータのサイズを減少させていき，グラフィクスハードウェアで使用するメモリを最小限に抑えることができる．

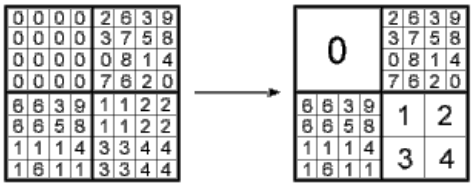


図 4 適応的サンプリング

4.3 中間画像の圧縮による高速化

一般に断層撮影などから得られるボリュームデータは全体の 70% から 95% が透明領域である．このことから，生成される中間画像には透明領域が多数含まれていることが多い．この透明領域では RGB の値はどのような値をとっても影響はない．したがって，不透明度 A が 0 であるピクセルの情報を圧縮することにより，通信データ量を削減することができる．生成された中間画像のデータはすべてのピクセルの RGBA 値 (各 1Byte) が順に並んだ 1 次元配列である．$A = 0$ となるピクセルがいくつか連続するとき，最初のピクセルの RGB 成分に相当する 3Byte 分の情報を使って，連続するピクセルの個数を表し，4Byte 目を 0 として圧縮することにより通信データ量の削減を行う (図 5)．例えば，$A = 0$ となるピクセルが N 個続いた場合，$4N$Byte から 4Byte へデータを圧縮することができる．この手法を用いることにより，中間画像の通信によるオーバヘッドを低下させ，描画の高速化を図る．

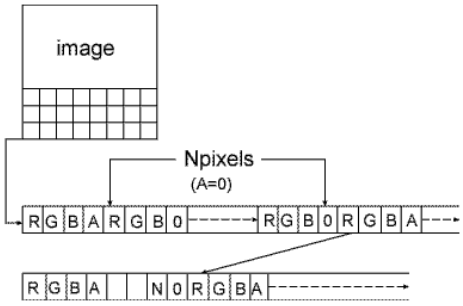


図 5 中間画像の圧縮

5. 実　　装

OpenGL グラフィクスライブラリを用いて PC クラスタ上で実験を行った. 実装環境を表 1 に示す. Master は生成された中間画像を 2 次元テクスチャとして扱い, α ブレンドして一つの画像にまとめるためのノード, Slave はサブボリュームのボリュームレンダリングを行うためのノードである.

表 1 実装環境

	Master
CPU	Pentium4 1.8GHz
Memory	512MB
Gfx Card	GeForce4 Ti4600
Gfx Memory	128MB DDR
OS	Red Hat Linux
	Slave
CPU	Pentium3 1.0GHz
Memory	512MB
Gfx Card	RADEON9700PRO
Gfx Memory	128MB DDR
OS	Red Hat Linux
NODE	4 台
Network	100BASE Ethernet PVM3.4

OpenGL では, α ブレンディングを行う際に前後の 2 枚の画像が持つ RGBA 値に乗じる α 値を混合係数として設定する. 色情報 (source) が, バッファに保存されている色情報 (destination) と, α ブレンドされて処理されるとすると, source と destination の RGB 値を C_s, C_d, α 値を A_s, A_d, 混合係数 (B_s, B_d) をとして, α ブレンド後のバッファの RGB 値と α 値は次式のように表される.

$$C = B_s C_s + B_d C_d \quad (3)$$

$$A = B_s A_s + B_d A_d \quad (4)$$

並列化を行う場合, スクリーンに対して後方にあるサブボリュームから生成した画像と, その前方にあるサブボリュームから生成した画像とを重ね合わせなければならない. そのため, 各サブボリューム全体の不透明度を計算しておく必要がある. RGB 値と α 値の計算式は RGB 値の混合係数を $(A_s, 1-A_s)$ とし, α 値の混合係数を $(1, 1-A_s)$ として次式のように表される.

$$C = A_s C_s + (1-A_s) C_d \quad (5)$$

$$A = A_s + (1-A_s) A_d \quad (6)$$

OpenGL では混合係数は RGBA 値共通の値として設定され, RGBA の各要素ごとに異なる混合係数を設定することができないため, 式 (5) の RGB 値と式 (6) の α 値を同時に求めることはできない. RGB 値と α 値を別々に求める方法も考えられるが, この方法だと α ブレンディングを 2 回行うことになるため処理速度が低下する. そこで, テクスチャデータが透明度を考慮したカラー値 (intensity) を持つというモデルであると考え, 各ボクセルの持つ RGB 値に予め α 値を乗じておくようにする. このように, テクスチャデータの RGBA 値を (AR, AG, AB, A) としておき, 混合係数を $(1, 1-A_s)$ として計算することで正確な画像を得ることができる.

6. 評　　価

6.1 仕様・測定方法

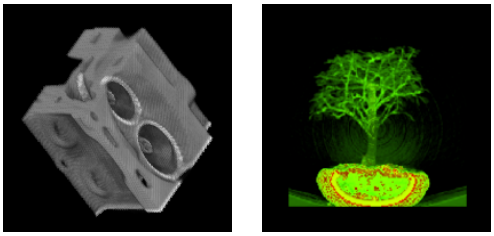
さまざまなボリュームデータを用いて評価を行った. 使用したデータは次の 4 種類である.

- Engine
  エンジンのボリュームデータでサイズは $256 \times 256 \times 128$ である (図 6-(a)).
- Bonsai
  盆栽のボリュームデータでサイズは $256 \times 256 \times 256$ である (図 6-(b)).
- Chest
  人間の胸部のボリュームデータでサイズは $512 \times 512 \times 512$ である (図 6-(c)).
- Tree
  クリスマスツリーのボリュームデータでサイズは $512 \times 512 \times 1024$ である (図 6-(d)).

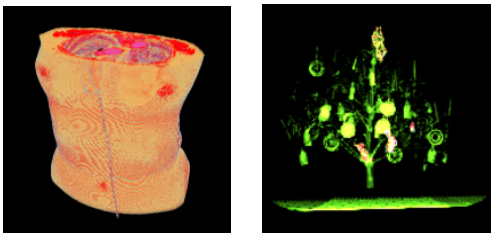
不透明度については各ボリュームデータともにボクセル値と等しい値をとっている. スクリーンのサイズは各ボリュームデータともに 256^2, 中間画像を生成する各ノードのサブスクリーンのサイズは 128^2 とした. スライスの数はボリュームデータの各軸方向のサイズの最大値とした. ボリュームレンダリングにかかる描画速度は視点を変更しながら複数枚 (数百枚程度) の描画を行い, それに要した時間から平均の描画速度を求めている.

6.2 単純な並列化の効果

並列化を行わずに 1 台で実行させた時の描画速度と, 適応的サンプリングも中間画像の圧縮も行わずに並列化を行ったときの描画速度を表 2 に示す. 非分割はボリュームデータを単一の 3 次元テクスチャとしてグラフィクスカードで描画させた時の描画速度を, 分割は



(a) Engine (b) Bonsai



(c) Chest (d) Tree

図 6 ボリュームデータ

ボリュームデータを 4^3 個の 3 次元テクスチャに分割して描画させた時の描画速度を示している．Engine, Bonsai についてはボリュームデータがグラフィクスメモリに入りきるサイズであるため, 分割することによる描画処理の増加や, 並列化による通信時間がオーバヘッドとなり, 並列化した時の方が描画速度が低下している．一方, Chest, Tree は非分割の場合, 描画することができないが, 分割することで低速度ながらも描画できるようになり, 並列化によって扱うことのできるグラフィクスカードのメモリが増えたため, 100 倍以上描画速度が向上していることがわかる．

表 2 単純な並列化の効果

Data	描画速度 [fps]			
	1 台		4 台	
	非分割	分割	非分割	分割
Engine	102.4	39.9	17.7	18.0
Bonsai	78.3	34.8	17.2	17.6
Chest	-	0.066	-	13.2
Tree	-	0.007	-	0.77

6.3 適応的サンプリングの効果について

適応的サンプリングを行うブロックの大きさは, 小さいと描画時のオーバーヘッドや使用メモリが増加し, 大きいとデータの局所性を利用しにくくなるため, 最適値は処理系とデータに依存する．誤差の許容値を 5%, 分割数を $d = 2$, slave マシンを 4 台とし, 分割してできるブロックの数を変えながら, 描画速度, 元データに対する適応的サンプリング後のデータサイズの比率 (縮小率), 適応的サンプリングにかかった時間を比較した結果を表 3 に示す．ブロックの数が 2^3 の時において Chest, Tree の描画ができないのは, サブボリュームのデータがグラフィクスカードのメモリに入りきらなかったためである．また, Engine, Bonsai はブロックの数が 4^3 の時, Chest, Tree はブロックの数が 8^3 の時が最も効率が良いことがわかる．Engine, Bonsai については元々のボリュームデータがグラフィクスメモリに入りきるサイズであったのに対し, Chest, Tree は元々のボリュームデータのサイズはグラフィクスメモリに入りきるサイズではない．このため, Chest, Tree は Engine, Bonsai より分割するブロックの数の最適値が大きくなったと考えられる．

ブロックの数が 4^3 の時の圧縮前と圧縮後の描画速度を比べてみると, 圧縮後の方が描画速度が向上しており, 適応的サンプリングを用いてデータサイズを圧縮することが効果的であることがわかる．しかし, 今回使用しているボリュームデータは時系列によって変化の起こらない静的なデータを用いており, 時系列によって変化するの起こりうる動的なデータの場合は適応的サンプリングにかかる時間を考慮しなければならない．適応的サンプリングにかかる時間は Engine では約 1 秒であるが, Tree では約 20 秒もかかっていることがわかる．このため, 適応的サンプリングは静的なデータに対しては有効な手法であるといえるが, 動的なデータに対しては処理時間がかかりすぎて有効な手法ではないと考えられる．

表 3 適応的サンプリングの効果

Data	ブロック数	描画速度 [fps]	縮小率 [%]	処理時間 [sec]
Engine	2^3	17.8	100	0.74
	4^3	18.2	93.6	0.79
	8^3	16.9	50.2	1.14
	16^3	11.3	34.2	1.48
Bonsai	2^3	17.2	100	0.93
	4^3	17.8	84.4	1.30
	8^3	15.8	53.9	1.66
	16^3	10.4	37.5	2.30
Chest	2^3	-	-	-
	4^3	13.5	100	3.69
	8^3	15.8	77.2	6.03
	16^3	10.4	66.8	8.81
Tree	2^3	-	-	-
	4^3	1.11	71.9	19.9
	8^3	12.7	44.9	20.9
	16^3	8.14	22.3	22.5

6.4 中間画像の圧縮効果

中間画像の圧縮は, 透明領域が多ければ通信データ量を減らすことができるが, 透明領域が少ない場合は圧縮にかかる時間が逆にオーバヘッドとなってしまう．slave マシンを 4 台とし, 中間画像の圧縮を行った時と行わない時の描画速度の比較を行った結果を表 4 に

示す. 縮小率は中間画像が最も小さなサイズに圧縮された時と最も大きなサイズに圧縮された時の縮小率を示している. 分割数は $d = 2$, 誤差の許容値は 5%, ブロックの数は Engine, Bonsai が 4^3, Chest, Tree は 8^3 とした. Engine, Bonsai, Chest に関しては約 2 倍近い速度向上が得られていることがわかる. Tree に関しては他の 3 つのデータ性能向上が得られていないが, これは Tree のデータがグラフィクスカードメモリ容量をオーバーしており, 3 次元テクスチャの入れ替えによるオーバヘッドの影響が強かったためであると考えられる. Tree のボリュームデータの解像度を半分にし, サイズを $256 \times 256 \times 512$ として描画させたみたところ, 中間画像の圧縮により約 2 倍近い速度向上が得られた. このことから中間画像の圧縮がどのボリュームデータにおいても効果的であることがわかる.

表 4 中間画像の圧縮効果

Data	描画速度 [fps]		縮小率 [%]
	非圧縮	圧縮	
Engine	18.2	34.6	48 ～ 79
Bonsai	17.8	38.1	37 ～ 64
Chest	15.8	27.5	42 ～ 82
Tree	12.7	14.4	41 ～ 73

6.5 並列化の効果

最後に適応サンプリングと中間画像の圧縮の両方を適用した場合の評価を行う. 並列化を行わずに 1 台で実行させた時の描画速度と並列ボリュームレンダリングを行ったときの描画速度を表 5 に示す. 分割数は $d = 2$, 誤差の許容値は 5%, ブロックの数は Engine, Bonsai では 4^3, Chest, Tree では 8^3 とした. Engine, Bonsai, Chest に関してはほぼリアルタイムに可視化できていることがわかる. データのサイズが大きくなると描画速度が著しく低下してしまうが, これはグラフィクスハードウェアが保持するテクスチャデータのサイズがグラフィクスメモリの容量を超えてしまったためだと考えられる. 逆に, $512 \times 512 \times 512$ のサイズ以下のボリュームデータは各ノードの汎用グラフィクスハードウェアのメモリ容量以下であったため, 高速な描画が可能であることがわかる.

表 5 並列化の効果

Data	描画速度 [fps]		縮小率 [%]
	1 台	4 台	
Engine	34.5	34.6	93.8
Bonsai	34.7	38.1	84.4
Chest	0.45	27.5	77.2
Tree	0.33	14.4	44.9

7. 結　　論

ブロック化による適応的サンプリングと中間画像の圧縮を用いて, 汎用グラフィクスハードウェアによる並列ボリュームレンダリングの実装行った. ブロック化による適応的サンプリングを用いることにより, 単純な並列化で描画できるデータサイズの約 2 倍の大きさのデータをほぼ同じ速度で描画することができた. また, 中間画像の圧縮を行って通信データ量の削減を行うことにより, 約 2 倍の速度向上を得ることができた. 本研究で用いたデータは時系列によって変化の起こらない静的なデータを用いているが, シミュレーション結果の可視化のような時系列によって変化するの起こりうる動的なデータの可視化は今後の課題である.

8. 謝　　辞

日頃より御討論いただく京都大学大学院情報学研究科富田研究室の諸氏に感謝します.

本研究の一部は文部省科学研究費補助金 (基盤研究 (B)(2) 課題番号 13480083 ならびに 特定領域研究 (C)(2)「情報学」課題番号 13224050) による.

本稿で用いた胸部のボリュームデータは, (株) ケイジーティー 宮地英生 氏より御提供いただいたものである. また, エンジン, 盆栽のボリュームデータは volvis ☆から, クリスマスツリーのボリュームデータは The Computer Graphics Group XMas-Tree Project ☆☆から利用させて頂いた.

参 考 文 献

1) C.Rezk-Salama, K.Engel, M.Bauer, G.Greiner, T.Ertl: "Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Texturesand Multi-Stage Rasterization", In Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware, 2000.

2) S.Muraki, M.Ogata, K.Ma, K.Koshizuka, K.Kajihara, X.Liu, Y.Nagano and K.Shimokawa, "Next-Generation Visual Supercomputing using PC Clusters with VolumeGraphics Hardware Devices", Proceedings of IEEE/ACM Supercomputer Conference, 2001.

3) 山崎, 加瀬, 池内: "PC グラフィクスハードウェアを利用した高精度・高速ボリュームレンダリング手法", 情報処理学会 CVIM-130-10, Nov. 2001.

☆ http://www.volvis.org/

☆☆ http://ringlotte.cg.tuwien.ac.at/datasets/XMasTree/