修士論文

実時間インタラクティブ
シミュレーション環境の構築

指導教官　　富田 眞治 教授

京都大学大学院情報学研究科
修士課程通信情報システム専攻

丸山 悠樹

平成16年2月6日

実時間インタラクティブ
シミュレーション環境の構築

丸山 悠樹

内容梗概

近年の計算機性能の急速な向上に伴い，大規模かつ高精度な数値シミュレーションへの期待が高まっている．このような中，次世代のシミュレーション技術として，従来の実験の代替手段となり得る「仮想実験型/仮想体験型のシミュレーション環境」の構築が望まれている．ここでは，オペレータによるシミュレーション対象へのインタラクティブな操作に対応して，実時間でシミュレーションを行うとともに，即刻その結果を，視覚その他の手段により提示することが求められる．

しかし，計算時間と計算精度はトレードオフの関係にある．実時間インタラクティブシミュレーション環境ではこの「リアルタイム性」と「高精度なシミュレーション」という相反する二つの要求に対応しなければならない．そこで我々はシミュレーションの状態に応じて，シミュレーションの精度を動的に変更できる環境を構築する．本論文ではオペレータの操作頻度に応じた精度制御手法と場の変化に応じた精度制御手法を提案し，シミュレーション精度の動的制御を目指す．

また，実時間インタラクティブシミュレーション環境における可視化システムでは，高精細かつ，高速な描画速度が要求される．そこで，本論文では汎用グラフィクスカード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い，リアルタイム可視化システムの構築を行った．この結果，サイズ $512 \times 512 \times 512$ のボリュームデータをリアルタイムに可視化することができた．

そして，ポアソン方程式と拡散方程式を解析対象として，状態に応じた精度制御を行う実時間インタラクティブシミュレーション環境の実装を行った．この結果，ポアソン方程式ではオペレータの操作頻度に応じた精度制御手法を用い，シミュレーションの精度を段階的に上げることができた．拡散方程式では場の変化に応じた精度制御手法を用いることにより，シミュレーション精度の動的な制御を可能とした．

Implementation of Real-Time Interactive Simulation System

Yuuki Maruyama

Abstract

Recently, large-scale numerical simulation is coming to be done in the various fields, caused by the rapid improvement of the computer performance. Under this circumstance, construction of "the simulation environment of a virtual experiment type and a virtual experience type" is desired as simulation technology of the next generation, which may serve as an alternative means of the conventional experiment.This system needs to performe a simulation in real-time corresponding to the interactive operation by the operator and to show the result at once by the means of vision and others needs.

But, calculation time and calculation accuracy have the relation of a trade-off. A real-time interactive simulation system have to correspond to two opposite demands called "real-time performance" and "highly precise simulation." Then, we implement the system where dynamically change the accuracy of a simulation, according to the state of a simulation. In this paper, we propose the accuracy control technique according to an operation frequency and change of a place, and aim at dynamic control of simulation accuracy.

Moreover, the visualization system in a real-time interactive simulation system requires high definition picture and high-speed drawing speed. In this paper, we implement parallel volume rendering system by standard graphics hardware, and this system attained the real time performance. Consequently, this system can visualize the volume data which size is $512 \times 512 \times 512$ on real time.

And we implement a real time interactive simulation system using poisson equation and diffusion equation. Consequently, the accuracy of a simulation was able to be gradually raised with poisson equation using the accuracy control technique according to an operation frequency. With the diffusion equation, dynamic control of simulation accuracy was enabled by using the accuracy control technique according to change of a place.

実時間インタラクティブ
シミュレーション環境の構築

目次

第 1 章　はじめに　1

第 2 章　実時間インタラクティブシミュレーション環境　3

2.1　システムの構想 3

2.2　シミュレーションと可視化系 4

2.3　リアルタイム可視化システム 6

2.3.1　3 次元データの可視化方法 6

2.3.2　ボリュームレンダリング 7

2.4　状況の分類 9

2.5　問題の提起 10

第 3 章　数値シミュレーションにおける精度制御　12

3.1　操作に応じた精度制御 12

3.1.1　制御手法 12

3.1.2　解の補間 13

3.1.3　初期ベクトルの選定方法 14

3.1.4　方針 15

3.1.5　計算の流れ 16

3.2　可視化の頻度に応じた精度制御 17

3.2.1　制御手法 17

3.2.2　方針 19

3.2.3　計算の流れ 19

3.2.4　解法の選択 22

3.3　本章のまとめ 23

第 4 章　汎用グラフィクスカードを用いた並列ボリュームレンダリングシステムの実装　24

4.1　汎用グラフィクスカードを用いたボリュームレンダリング 24

4.2　並列化 26

4.2.1　基本方針 26

4.2.2 適応的サンプリング 26
4.2.3 中間画像の圧縮による高速化 29
4.3 実装 29
4.4 評価 31
4.4.1 仕様・測定方法 31
4.4.2 単純な並列化の効果 33
4.4.3 適応的サンプリングの効果について 34
4.4.4 中間画像の圧縮効果 35
4.4.5 並列化の効果 36
4.5 本章のまとめ 37

第 5 章 システムの実装 **39**
5.1 ポアソン方程式 39
5.1.1 ポアソン方程式の離散化 39
5.1.2 実装 40
5.1.3 評価 40
5.2 拡散方程式 42
5.2.1 前進差分による離散化 42
5.2.2 後退差分による離散化 43
5.2.3 実装 43
5.2.4 評価 44
5.3 本章のまとめ 46

第 6 章 まとめ **48**

謝辞 **49**

参考文献 **50**

付録 **A-1**
A.1 100BASE Ethernet 環境における汎用グラフィクスカードによる並列ボリュームレンダリングシステムの評価 A-1
A.1.1 仕様・測定方法 A-1
A.1.2 単純な並列化の効果 A-2
A.1.3 適応的サンプリングの効果について A-3

A.1.4　中間画像の圧縮効果 A-4
A.1.5　並列化の効果 A-4
A.2　マルチグリッド法 A-4
A.3　共役勾配法 A-6
A.3.1　アルゴリズム A-6
A.3.2　実装 A-7
A.4　Courant 条件 A-7

第1章　はじめに

近年の計算機性能の急速な向上に伴い，大規模かつ高精度な数値シミュレーションへの期待が高まっている．中でも実時間内での数値シュミレーションは，大規模な3次元データを必要とする医療などの分野において高度な技術が要求されている分野である．これまでも，PC クラスタ等の並列計算機環境を利用した高精度なシミュレーションシステムが開発されてきたが，次世代のシミュレーション技術として，従来の実験の代替手段となり得る「仮想実験型/仮想体験型のシミュレーション環境」の構築が望まれている．ここでは，オペレータによるシミュレーション対象へのインタラクティブな操作に対応して，実時間でシミュレーションを行うとともに，即刻その結果を，視覚その他の手段により提示することが求められる．また，仮想現実感を提供し得るだけの高精度なシミュレーションもまた必要である．

しかし，計算時間と計算精度はトレードオフの関係にある．このため，ゲームのような対話的なシミュレーション環境では，シミュレーションの精度よりもリアルタイム性が第一に要求されてきた．一方，スーパーコンピュータ等の超高速計算機による大規模な数値シミュレーションでは，高速な計算が望まれるものの，リアルタイム性よりはシミュレーションの精度が要求されている．

実時間インタラクティブシミュレーション環境ではこの「リアルタイム性」と「高精度なシミュレーション」という相反する二つの要求に対応しなければならない．そこで我々はシミュレーションの状態に応じて，シミュレーションの精度を動的に変更できる環境を構築する．これにより，オペレータへのシミュレーション結果の提示は十分に行いつつも，計算資源を最大限まで有効利用することができる．

また，実時間インタラクティブシミュレーション環境における可視化システムでは，高精細かつ，高速な描画速度が要求される．そこで我々は自然現象といった，数値シミュレーションで頻繁に扱われる対象の可視化に適している，ボリュームリングを用いたリアルタイム可視化システムを構築する．

本論文では実時間インタラクティブシミュレーション環境として，シミュレーション精度の動的制御を行うシステム，並びに汎用グラフィクスカードを用いたリアルタイム可視化システムを実装し，その評価を行った．まず第2章において，我々が現在検討している実時間インタラクティブシミュレーション環境

の構想，構成方式，並びにリアルタイム可視化システムについて述べる．第 3 章ではシミュレーションの精度を動的に制御する方法についてについて述べる．第 4 章で本システムに実装する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システムについて述べる．

第 5 章では実際に実装した解析対象について説明し，評価を行い，第 6 章でまとめる．

第2章　実時間インタラクティブシミュレーション環境

本章ではまず，我々が実現しようとしている実時間インタラクティブシミュレーション環境の構想とその構成方式について述べ，検討すべき課題について述べる．

2.1　システムの構想

実時間インタラクティブシミュレーション環境とはオペレータによるシミュレーション対象へのインタラクティブな操作に対応して実時間[1)]でシミュレーションを行うとともに，即刻その結果を視覚その他の手段により提示することのできる仮想実験型/仮想体験型のシミュレーション環境のことである．

通常の数値シミュレーションでは，初期状態と幾つかのパラメータをシミュレーションを開始する前に設定して，大型計算機等で計算を行い，後にシミュレーションの最終結果の可視化を行うというものがほとんどであり，そのパラメータを計算の途中で変更することは極めて稀である．しかし，医療分野で必要とされている手術シミュレーション [10] や触診シミュレーション [9] といった対話性の要求されるシミュレーション等では，計算途中にオペレータの判断によって，パラメータを変更することがしばしば起こり，それに伴ってそのような時々刻々と変わる状況を実時間でシミュレーションしていかなければならない (インタラクティブ性)．このような対話性の要求されるシミュレーション環境では高精度なシミュレーションはもちろんのこと，従来の数値シミュレーションではあまり重視されていなかったリアルタイムでの応答[2)]が必要とされる．

また，実時間インタラクティブシミュレーション環境の構築に際し，計算機システムとして検討すべき課題は，シミュレーションの高速化のための「スループット」，ならびにユーザの操作性向上のための「レスポンス」といった，両立が困難な要求への対応である．そこで我々は，計算負荷の高いシミュレーションと多次元データの可視化までを PC クラスタベースの計算サーバで行い，オペレータの操作や結果の呈示を行うユーザインタフェース端末としてのクライアントを用いたクライアントサーバモデルの環境を構築する (図 1)．これによ

---

1) 本論文ではオペレータへのシミュレーション結果の提示を視覚的手段を用いて提示する．以後,「実時間な応答」を **30fps**(frame per second) の描画速度で結果を提示できる性能 であることと定義する．

2) 「実時間」と同様に 30fps の描画速度で結果を提示できる性能であることと定義する．

り，計算サーバを煩雑な入出力処理から解放するとともに，柔軟なユーザインタフェース環境の実現を目指す [2].

また，クライアントと計算サーバのデータのやりとり間に通信遅延が生じたり，高精度なシミュレーションのために計算サーバのスループットが足りなくなるという可能性も考えられる．そこで，クライアントでは状況に応じて簡易シミュレーションを行い，計算サーバからのシミュレーション結果の提示の遅れを隠蔽する．こうすることにより，オペレータがリアルタイムな操作感覚を維持しつつシミュレーションの可能なシステムを目指す.

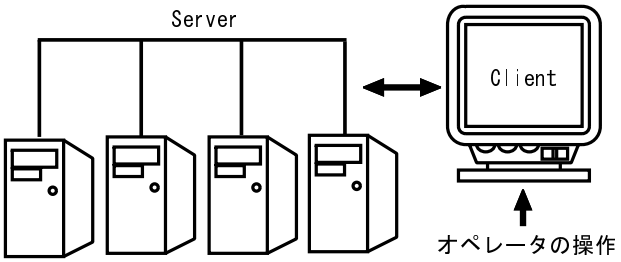


図 1: クライアントサーバモデル

2.2 シミュレーションと可視化系

シミュレーション結果のデータをどこで可視化し，どのようにしてオペレータに提示するのが望ましいかというのは扱う問題，構成するシステムの仕様によって様々である．本項ではクライアントで一括して可視化する場合と，各計算機がそれぞれ可視化する場合の 2 つに絞り，その特徴を詳しく述べる．なお，シミュレーションを行う部分を**シミュレーション系**，可視化処理を行う部分を**可視化系**と定義とする.

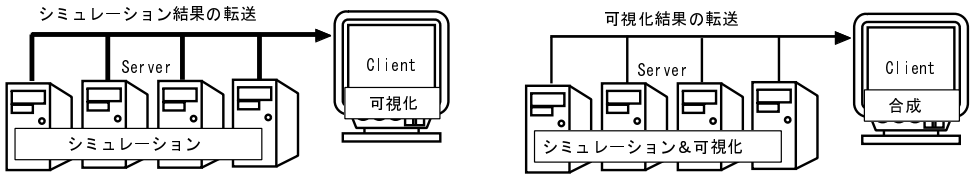
A) クライアントで一括して可視化

まず考えられるのが，計算サーバではシミュレーションのみを行い，それとは別にクライアントで可視化処理を行ったり，可視化専用ハードウェアのようなものを設け，そこで生成データを一括して可視化する方式である (図 2-(a))．従来の数値シミュレーションにおける可視化対象は最終結果が出てから可視化を行うものがほとんどである．このため，シミュレーショ

ン系から可視化系への可視化対象データの受け渡しは一度きりであり，シミュレーション系と可視化系との間の通信にはそれほど高い性能は求められない．また，視点変更のような可視化パラメータの変更を行う時に，計算サーバとのデータの受け渡しをする必要がない分高速である．また，後述するクライアントで簡易シミュレーションを行うような構成の場合は，計算サーバから集めた結果を利用して，簡易シミュレーションの解を補間するなど，シミュレーションの精度を上げたり，計算時間を短縮することができると考えられる．

B) 各計算機がそれぞれ可視化

各計算機ごとに可視化系を設け，画像のような中間データを生成し，これをクライアントに集めて最終結果を可視化するという方式である (図 2-(b))．シミュレーション系から可視化系へのデータの受け渡しが頻繁に起こる場合はこちらの方法が適している．ただし，可視化パラメータを変更する時に計算サーバとのデータの受け渡しが必要である．



(a) クライアントで一括して可視化　　(b) 各計算機がそれぞれ可視化

図 2: 構成方式

可視化対象がシミュレーションの途中経過であったり，時間的に変化するシミュレーションなどである場合は，時々刻々と対象データの変化する様子を可視化しなければならない．この場合，シミュレーション系から可視化系への高速なデータの受渡しが求められる．したがって，データの受け渡しの少ない方式の方が有効である．

先ほど説明した二つの方式のうち，どちらがデータの通信が少ないかを例を挙げて説明する．今，n^3 個の 64bit のシミュレーションを行い，これをそのまま通信する場合と，8bit の RGBA で表されたサイズ m^2 の画像データに変換し

て通信する場合を考える．シミュレーション結果を可視化せずにそのまま通信した方が通信量が少なくすむのは

$$64n^3 \quad < \quad 8m^2 \times 4 \tag{1}$$

が成り立つ時である．上式を変形すると

$$m \quad > \quad n\sqrt{2n} \tag{2}$$

である．$n = 16$ ならば $m > 64\sqrt{2}$，$n = 32$ ならば $m > 512\sqrt{2}$ である．以上のことから，B) 各計算機がそれぞれ可視化した方がデータのやりとりが少なくてすむことがわかる．したがって，実時間インタラクティブシミュレーション環境においては方式 B の方が有効であるといえる．ただし，計算サーバの各ノードが可視化処理を行うことによってシミュレーションの妨げとなることはできる限り避けなければならない．このため， 計算サーバの各計算機がそれぞれ可視化アクセラレータを装備する方法が最も適切であると考えられる．

2.3 リアルタイム可視化システム

実時間インタラクティブシミュレーション環境における可視化システムでは，高精細かつ，高速な描画速度が要求される．本項では，まず実時間インタラクティブシミュレーションの可視化方法，主にボリュームレンダリング処理についての説明を行う．

2.3.1 3 次元データの可視化方法

仮想現実感を提供する実時間インタラクティブシミュレーション環境が扱うシミュレーションは主に 3 次元現象である．したがって，ここでは 3 次元現象を可視化方法についてのみ述べる．3 次元現象を可視化するには 2 つの方法がある．物体の表面を表示するサーフェイスレンダリングと，対象を 3 次元的に中身の詰まった立体要素の集合体と考えるボリュームレンダリングである．

サーフェイスレンダリング

サーフェイスレンダリングは 3 次元空間上の物体を多角形の平面の集合として近似し，物体の表面を形成するポリゴンを多数表示する事によって実現される．しかし，このポリゴンを用いた幾何学的 3 次元表示法は，対象となる 3 次元データの簡単なモデル化から始まっている．このため，境界や表面を抽出し，それらをどのような幾何学的要素で表現するかは決定す

る人やアプリケーションに左右されてしまう．

ボリュームレンダリング

ボリュームレンダリングは3次元的に中身の詰まったボリュームとして表現された対象の複雑な内部構造や動的特性を表示するための技術である．ボリュームデータは空間の最小単位であるボクセルの集合で表されている．ボクセルは3次元座標上の点からサンプリングされたもので，元の対象物について，その地点での強度，温度といった観測値や計算値を持たせることにより，空間全体をデータとして保持している．そして，格子点以外の地点のデータは，その地点近傍の格子点から補間を用いて求める．これにより，空間内のすべての地点におけるデータを得ることができる．ボリュームレンダリングではこれらの3次元ボクセルを幾何形状に変換することなく2次元平面へ投影するため，以下のような特徴を持つ．

- 表示までの間に幾何形状への近似を入れないので，表示像が正確である．
- 雲や炎，エネルギー場などといったはっきりとした境界を持たない自然現象に適用できる．
- 物体の透過を表現することが可能なため，物体の内部構造を伴った半透明の画像を容易に作成することが可能である．

ボリュームレンダリングによって複雑な3次元構造の理解が容易となる．このためボリュームレンダリングは工学，医学，娯楽等様々な分野で幅広く利用されている．

以上のことからも，ボリュームレンダリングが数値シミュレーションの可視化において非常に有効な手段であることがわかる．次項ではボリュームレンダリングのアルゴリズムについて詳しく述べる．

2.3.2 ボリュームレンダリング

ボリュームレンダリングは3次元のスカラー場やベクトル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．

直接法のボリュームレンダリングでは，数値シミュレーションにより得られた3次元空間上の数値データを，RGB値と不透明度に対応付けて可視化するこ

とで，3 次元空間内部のデータの分布状況を可視化する．具体的に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 (図 3)．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，ピクセル値 P はボクセルの値を視点に対して遠い順から，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

$$P = \sum_{i=0}^{n} \alpha(v_i)c(v_i) \prod_{j=0}^{i-1}(1-\alpha(v_j)) \tag{3}$$

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

$$C_k = \alpha(v_i)c(v_k) + (1-\alpha(v_i))C_{k-1} \tag{4}$$

$$A_k = \alpha(v_i)A_{k-1} \tag{5}$$

ここで $P = C_n$ である．このピクセル値計算は，演算区間をいくつかの部分区間に分割し，それぞれの区間に対する計算結果に対して，再度ピクセル値計算を行うことが可能であるという性質がある．したがって，シミュレーションサーバの各ノードが，そのノード内で生成されたデータに対する部分 3 次元空間 (以下**サブボリューム**と呼ぶ) に対してピクセル値計算を行い，そこで得られた画像と画素毎の透明度を視点からの距離の順番に従って，合成を行っていくという手法で並列処理が可能である．

しかし，ボリュームレンダリングには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る．我々はこのようなボリュームレンダリング処理の実装に際しては，可視化処理に対してどれだけの投資が可能かに応じて，3 つの側面を考え，それぞれ研究を行ってきた．高速を目指した専用ハードウェアでの実現 [1][3]．処理対象に応じて処理内容を柔軟に変更でき，処理の高機能化も容易であるソフトウェアによる実現 [4]．比較的安価に手に入る汎用グラフィクスカードによる実現 [5] である．本論文では汎用グラフィ

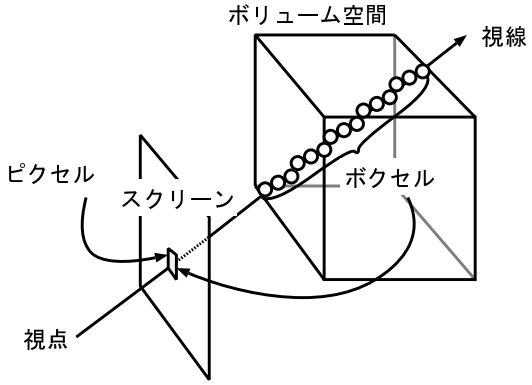


図 3: ボリュームレンダリング

クスカード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．具体的な実装に関しては第 4 章で詳しく述べる．また，このシステムは第 5 章で述べる実時間インタラクティブシミュレーション環境におけるリアルタイム可視化システムとしても用いる．

2.4 状況の分類

クライアントと計算サーバが置かれる状況としては，様々な状況が想定される．クライアントと計算サーバ間に通信遅延が存在する場合もあり，その場合は通信遅延を考慮したシステムの構築が必要となる．また，高いシミュレーション精度をオペレータから要求されたために，計算サーバのスループットが足りないという状況も考えられる．このような問題が生じ，オペレータの操作に対してインタラクティブ[1)]にシミュレーション結果を提示することができない場合は，例えばクライアントで簡易シミュレーションを行うなどして，計算サーバからのシミュレーション結果の提示の遅れを隠蔽し，オペレータができるかぎり柔軟にシミュレーションを行えるような環境を提供しなければならない．以下ではこれらの問題によって，構成方式にどのような差異が生じ，どのように解決すべきか述べる．

A) スループットが足りていて，通信遅延が存在しない場合

計算サーバで行われたシミュレーションの結果は実時間内でオペレータに提示することができる．

[1)] 実時間内で提示できることと定義する．

B) 通信遅延は存在しないが，スループットが間に合わない場合

計算サーバが高いシミュレーション精度を要求されたために，シミュレーションの結果を実時間内でオペレータに提示することができないような場合を想定している．

C) スループットは足りているが，通信遅延が存在する場合

計算サーバが遠隔地に存在し，クライアントとのデータの通信の際に通信遅延が起こってしまうような場合を想定している．

D) スループットが足りていなくて，通信遅延が存在する場合

計算サーバが高いシミュレーション精度を要求されたために，シミュレーションの結果を実時間内でのオペレータに提示することができない．さらに，計算サーバが遠隔地に存在し，クライアントとのデータの通信の際に通信遅延が起こってしまう，という状況を想定している．

状況 A において，クライアントは計算サーバのシミュレーション結果をオペレータに提示することと，オペレータからの操作にだけ専念すればよい．一方，状況 B，C，D では，オペレータから操作があって，計算サーバからシミュレーション結果が返ってくるまでに実時間以上の時間がかかってしまう．このため，クライアントで実時間内で計算できる精度でのシミュレーションを行うなどして，計算結果の提示までの遅延時間を隠蔽する必要がある．

遅延が存在せず，スループットが足りている場合は何も問題は発生しない．しかし，スループットが足りなかったり，通信遅延が存在すると，計算結果の提示までに遅延時間が生じてしまう．本研究の目的はこのような理由によって生じた，シミュレーション結果を提示するまでの遅延時間を隠蔽し，オペレータがリアルタイムな操作感覚を維持しつつ，シミュレーションを続けられるようなシステムを構築することである．本論文では，主にスループットが間に合わないという問題に焦点をあて，この遅延時間を隠蔽する手法について述べていく．

2.5 問題の提起

高精度な数値シミュレーション行う場合，計算規模は精度とともに増大し，シミュレーションの精度を上げれば，それだけオペレータの操作に対してレスポンスを返す時間は遅くなってしまう．このため，リアルタイム性を確保するためににはシミュレーションの精度を制限しなければならない．ゲームはこのようなシミュレーションの一種といえる．このように計算時間と計算精度はトレー

ドオフな関係にある．実時間インタラクティブシミュレーション環境ではこの「リアルタイム性」と「高精度なシミュレーション」という相反する二つの要求に対応しなければならない．そこで我々はシミュレーションの状態に応じて，シミュレーションの精度をゲームのように低いレベルから大規模数値シミュレーションような高レベルまで動的に変更できる環境を構築する (図 4)．このようにすることにより，オペレータへのシミュレーション結果の提示は十分に行いつつも，計算資源を最大限まで有効利用することができる．

スループットの足りない計算機環境において，処理量を削減し，スループットを確保するための手法としては，モデルを簡易化する手法 [10]，注視領域のみでシミュレーションを行う手法 [11]，2 次元データに次元を下げる手法 [12] など，数多く存在する．

次章ではシミュレーションの精度を動的に制御する方法についてに詳しく述べていく．

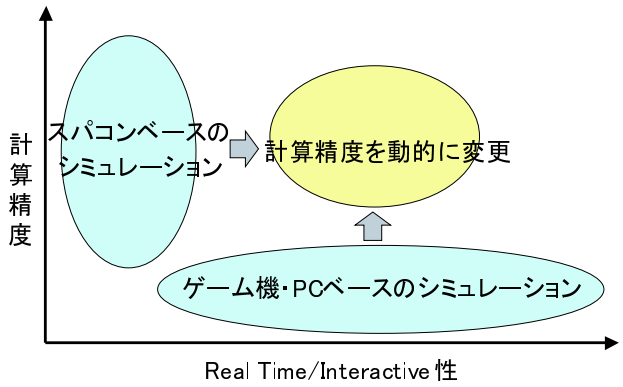


図 4: シミュレーション精度を動的に制御

第3章　数値シミュレーションにおける精度制御

本章ではシミュレーションの状態に応じて，シミュレーションの精度を動的に変更する方法として，オペレータ操作頻度に応じた精度制御，系の変化に応じた精度制御を提案する [6]．オペレータの操作頻度に応じた精度制御手法のみで支配方程式に時間微分項の含まれない問題はシミュレーション精度の動的制御が可能である．しかし，時間微分項が含まれる問題では後述する系の変化に応じた精度制御を行う必要がある．以下ではこの二つの手法についてそれぞれ詳しく述べる．

なお，具体的な実装について述べた話題では，表2で示したクライアントサーバモデルを想定して話を進めている．

3.1　操作に応じた精度制御

本項では，数値シミュレーションにおいて粗さの異なる複数のグリッドを利用する手法の説明が行われる．以後，対象となっている数値シミュレーションにおいて，最も粗いグリッドを使用したシミュレーションから順に，レベル1のシミュレーション，レベル2のシミュレーション･･･と定義する．

3.1.1　制御手法

実時間インタラクティブシミュレーション環境では，オペレータの操作に対応したリアルタイムな応答と，高精度なシミュレーションが必要とされる．しかし，常に秒間30回のシミュレーション結果を提示する必要があるわけではない．オペレータの操作があり，現在シミュレーションを行っている対象の変化があった場合にのみ，ただちにオペレータに対し，その結果を知らせることができれば十分である．同様に，オペレータの操作を計算機が受け取る頻度も常に秒間30回のシミュレーション結果を提示する必要があるわけではなく，オペレータの操作する情報に一定以上の変化が見られたときのみ，その情報を計算機が受け取ることができれば，リアルタイムな応答は十分可能である．

さて，システムがオペレータの操作に対して以上のような反応を行うとすると，オペレータの次の操作までの時間間隔に大小の差が生じる．つまり，オペレータの操作する情報が急激に変化するときは次の操作までの時間間隔が小さくなり，緩やかに変化するときは次の操作までの時間間隔が大きくなる．そこで，この時間間隔の差を利用して，シミュレーションの精度を変更させる．具

体的には，時間間隔が小さいときはシミュレーションの精度を粗くし，時間間隔が大きいときはシミュレーションの精度を細かくするわけである．以上のような方法を用いることにより，オペレータ操作頻度に応じたシミュレーションの精度制御を行うことができる．

しかし，次のオペレータの操作までの時間間隔をシミュレーションを行っているその時点で知ることは不可能である．そこで，まず一番精度の低いレベル 1 のシミュレーションを行い，その計算が終了した時点で，オペレータからの次の操作がなければ，レベル 2，レベル 3$\cdots$ とシミュレーションの精度を上げていく．これを繰り返すことにより，段階的にシミュレーションの精度を上げることができる．

3.1.2 解の補間

支配方程式に時間微分項の含まれない問題の解法は連立一次方程式の求解に帰結する．連立一次方程式を解く方法は大別して直接法と反復法に分類される．一般的に小規模な計算の場合は直接法が大規模な計算では反復法が適していると言われる．本研究では，大規模な問題のシミュレーションを想定しているため，数値シミュレーションにおける連立一次方程式を解法には一貫して反復法を扱うことにする．

さて，段階的に精度を上げていきながらシミュレーションを行っていく場合，その解法に反復法を用いることにより，以下の方法によって計算の高速化が可能である．まず，レベル n のシミュレーションからレベル $n+1$ シミュレーションに移るときに，レベル n のシミュレーションで得られた解をレベル $n+1$ のシミュレーションのグリッド上に補間する．次に，これを反復法の初期ベクトルとしてレベル $n+1$ のシミュレーションを行う．この概念はマルチグリッド法 [13] において，グリッド間でのベクトルの補間を行う手法を導入したものである (付録 A.2 参照)．細かいグリッドのある点における補間後の値 V^{n+1} は，その点に寄与する粗いグリッド上の点の値 V_i^n と寄与する割合 e_i を用いて

$$V^f = \sum V_i e_i \tag{6}$$

となる．ただし

$$\sum e_i = 1 \tag{7}$$

である．サイズ $2^3 n$ のグリッドからサイズ $2^3(n+1)$ のグリッドへ補間する場合

を考える．図 5 は，2^3n のグリッド上にある 1 点 ($\bigtriangledown$ で表示) とその点に対応する $2^3(n+1)$ のグリッド上の隣接する 8 個のデータ ($\bigcirc$ で表示) を示したものである．グリッドのサイズから判断して，$\bigcirc$ に寄与する粗いグリッド上の点は $\bigtriangledown$ のみである．したがって，2^3 個のデータはすべて中心の $\bigtriangledown$ の値と同じ値をとる．このようにして，補間は行われる．反復解法においては一般的に粗いグリッドの方が解の伝播が高速で，収束が早い．このため，単純に細かいグリッドのみを用いて計算を行うよりも，粗いグリッドで解の伝播をさせておいて，その解を細かいグリッドへ補間し，これを初期ベクトルとして反復法を実行させた方が解の収束が早くなる．ただし，グリッド間での補間を行っているため，シミュレーションの場が連続な領域内のみでしか補間を行うことはできない．

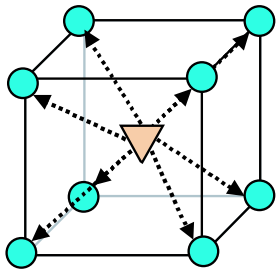


図 5: 解の補間

3.1.3 初期ベクトルの選定方法

従来，支配方程式に時間微分項の存在しない静的な場の解析では，一回解いてしまえばシミュレーションは終了であった．このため，反復解法における初期ベクトルは固定値であった．一方，対話性の要求される静的なシミュレーションでは従来の静的なシミュレーションと違い，シミュレーション対象に境界条件を与えて解いてしまえば終了というわけではない．オペレータによる操作の続く間はシミュレーションが何度も連続して繰り返される．

シミュレーションにおいてオペレータが境界条件などのパラメータを連続的に変化させるとき，シミュレーションの結果も連続的に変化し，直前の結果と近い結果になることが多い．反復法における収束率は初期ベクトルの選定にも依存しており，収束値に近い値を初期ベクトルとして用いた方が計算時間を短縮できる．つまり，オペレータの操作が連続的である場合は，反復解法における初期ベクトルとして直前の操作によるシミュレーション結果を利用すること

で，収束を早めることができると考えられる．さらに残差の収束が単調減少であるモデルに対しては途中まで計算されていた解を利用することでも，収束を早めることができる．

しかし，本手法はシミュレーションの精度を段階的にあげる手法を併用することを考慮しなければならない．単純にレベル 1 のグリッドの初期ベクトルに 1 操作前の解を初期ベクトルとして用いただけでは，レベル 1 のシミュレーションの収束は早くなったとしても．他のレベルのシミュレーションの収束は早くならないからである．そこで，以下のような手法で初期ベクトルの選定を行う．まず，レベル 1 のグリッドでは普通に前の操作で求めた解を初期ベクトルにして求める．そしてレベル n のグリッドでは，レベル $n-1$ のシミュレーションの解を補間したものに，レベル n の 1 操作前のシミュレーションの解から差し引いた値を残しておく．次に，レベル 1 のシミュレーションから順に解が計算され，細かいグリッドに補間されていく．この補間された値を先ほど残しておいた解に加え，これを初期ベクトルとする．

レベル $n-1$ のグリッドにおける解を $\boldsymbol{x}_{n-1}$ で表し，レベル n のグリッドにおける解を $\boldsymbol{x}_n$ で表す．1 つ前の操作によって求めた解を $\boldsymbol{x}^{pre}$，現在の求めるべきシミュレーションの解を $\boldsymbol{x}^{now}$ で表す．また，レベル $n-1$ のグリッドからレベル n のグリッドへ解を補間する関数を行列 $\boldsymbol{P}^n_{n-1}$ で表すとする．このとき，レベル n のグリッドで残しておくべき値は

$$\boldsymbol{x}^{pre}_n - \boldsymbol{P}^n_{n-1}\boldsymbol{x}^{pre}_{n-1} \tag{8}$$

となる．そして，細かいグリッドにおいて，反復法の初期ベクトルとすべき値は

$$\boldsymbol{P}^n_{n-1}\boldsymbol{x}^{now}_{n-1} + \boldsymbol{x}^{pre}_n - \boldsymbol{P}^n_{n-1}\boldsymbol{x}^{pre}_{n-1} \tag{9}$$

となる．このようにすることで段階的に解の精度を上げる場合にも 1 操作前の解を有効に使うことができる．

3.1.4 方針

オペレータの操作があってからシミュレーション結果を可視化し提示するまでの流れを図 6 に示す．図では 3 レベルのグリッドの段階的な表示方法を説明する．レベル 1 のグリッドはリアルタイム性を確保する必要があるため，クライアントで計算を行う．また，そのグリッドの粗さはリアルタイム性を確保できるレベルまで粗くしておく．まず，オペレータの操作があった場合，その情

報はただちにクライアントと計算サーバの両方に送られる (1)．そしてクライアントとサーバの両方でレベル 1 のグリッドの計算がが実行される (2)．クライアントでのシミュレーション結果は，即刻可視化され，オペレータに提示される (3)．一方，サーバ側ではレベル 1 のシミュレーションが終了すると，その解をレベル 2 のグリッドに補間し (4)，レベル 2 のシミュレーションが開始される (5)．レベル 2 のシミュレーションが終了すると，その結果をクライアントに送信し (6)，その結果がオペレータに提示される (7)．さらに，計算サーバではレベル 2 の解をレベル 3 のグリッドに補間し (8)，レベル 2 のシミュレーションが開始される (9)．そして，レベル 3 のシミュレーションが終了すると，その結果をクライアントに送信し (10)，その結果がオペレータに提示される (11)．以上のような流れで段階的にシミュレーションの精度が上げられる．

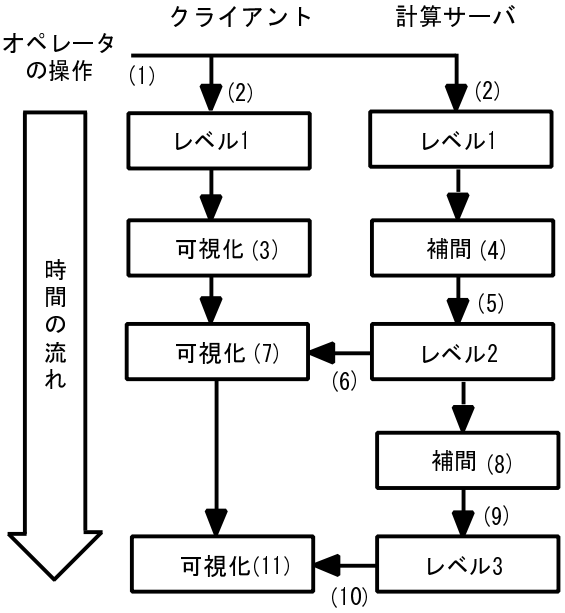


図 6: 段階的に精度を変更

3.1.5 計算の流れ

ここではオペレータからの操作が連続した場合にシステムがどのようなインタラクションを起こすか説明する．図 7 に示すような時間間隔でオペレータからの操作があるとする．まず一つめの操作があると，その情報はただちにクライアントと計算サーバの両方に送られる (1)．そしてクライアントとサーバの両

方で粗いグリッドの計算がが実行される (2)．クライアントでのシミュレーション結果は，即刻可視化され，オペレータに提示される (3)．ここで，オペレータから二つ目の操作がある (4)．クライアントでは計算が終了しているため，すぐに次のシミュレーションを開始する (5)．計算サーバでは 1 操作前のシミュレーションを中断し，次のシミュレーションを開始する (6)．一つめの操作と二つ目の操作の時間間隔は短かったため，計算サーバの結果を返すことができなかったが，二つ目の操作と三つ目の操作の時間間隔が長いため，計算サーバの結果を返すことができる (7)．以上のような流れでオペレータの操作頻度に応じたシミュレーションの精度制御を行うことができる．

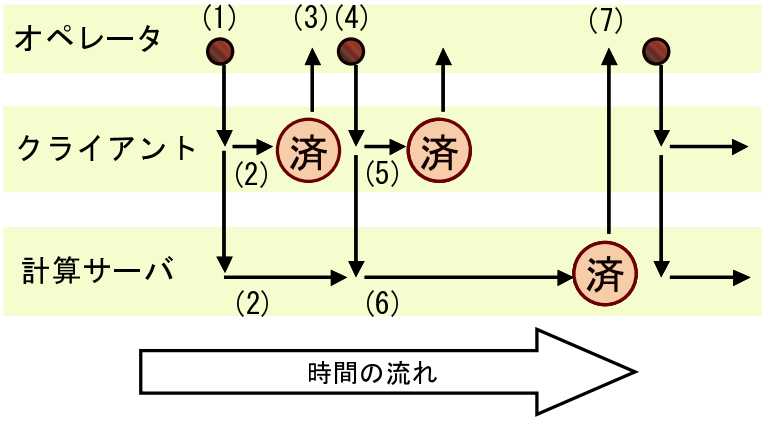


図 7: 操作頻度に応じた精度制御

3.2 可視化の頻度に応じた精度制御

ここでは，シミュレーション上の時間の進行と現実の時間の進行両方を用いた説明を行う．以下，この違いを明確に区別するために，シミュレーション上の時刻の進行を t で表現し，現実の時間の進行を T で表現する．そして，時間刻み幅 ΔT で離散化された現実の時間軸上の点 T_0，T_1，T_2，$\cdots$ を**タイムステップ**と呼ぶ．また，特にクライアントのシミュレーション上の時間刻み幅を指す場合は Δt_c，クライアントのシミュレーション上の時間刻み幅は Δt_s と表記する．

3.2.1 制御手法

実時間インタラクティブシミュレーション環境では，オペレータの操作に対応したリアルタイムな応答と同様に，時々刻々と変化するシミュレーション結

果をリアルタイムにオペレータに提示していかなければならない．しかし，前に提示したシミュレーション状態から変化のないシミュレーションの結果はオペレータに提示する必要はない．したがって，系の変化が人間に区別のつかないほど小さい場合に，系の状態を可視化せずとも，オペレータへのシミュレーション結果の提示は十分に行うことができる．

システムが系の変化に対して上記のような反応を行う場合，系の変化をオペレータに提示する次の時刻までの時間間隔に大小の差が生じる．この時間間隔の差を利用して，シミュレーションの精度を変更させる．以上のような方法を用いることにより，系の変化に応じたシミュレーションの精度制御を行うことができる．

さて，時間的変化の見られる数値シミュレーションでは，現在のシミュレーションの状態から時刻 Δt 後の状態を求める．このような問題を実時間インタラクティブシミュレーション環境で扱う場合，系が時々刻々と変化する状態を時間刻み幅 ΔT ごとに計算していかなければならない．

今，精度の高いシミュレーションと，精度の低いシミュレーションの時間刻み幅 Δt_s，Δt_c を等しくとり，シミュレーションを行うとする．このとき，精度の低いシミュレーションが ΔT 以内ぎりぎりで計算を終了できたとしても，精度の高いシミュレーションでは計算量が大きいため，ΔT 以内の時間で計算を終了することはできない．これでは精度の高いシミュレーションの結果がオペレータに提示できず，シミュレーションの精度制御を行うことはできない．

そこで，シミュレーション上の時間刻み幅 Δt をシミュレーションの精度によって変更することを考える．まず，精度の低いシミュレーションではリアルタイム性を確保できるレベルに Δt を設定する．また，シミュレーションの精度は ΔT 以内で計算できる精度とする．次に，精度の高いシミュレーションでは si シミュレーションの計算時間以上となるように Δt を設定する．系の変化が微小である場合には可視化結果が更新されないため，精度の高いシミュレーションの結果をオペレータに提示することができる．自然現象などのシミュレーションにおいては，系の状態の変化は系に何らかの変化があった瞬間は激しく変化するものの，時間の経過につれて次第に変化量は減少していくものが多い．また，系の状態が収束しなくても，一時的に変化量のごく少ない状態が続くこともあり得る．したがって，以上のような手法で，系の変化に応じたシミュレーションの精度制御を行うことができると考えられる．

3.2.2 方針

時間的変化の見られる数値シミュレーションの解法は大別して陽解法と陰解法に分類される[1)]．陰解法では連立一次方程式を解くのに対し，陽解法では連立一次方程式を解かず，直接的に次の時刻の状態を求める．このため，一般に陽解法は陰解法より計算量が少ない．しかし，安定に解析が行われるためには，Δt をある限界値 Δt_{max} よりも小さくとる必要がある (Courant 条件)．つまり，時間刻み幅 Δt が Courant 条件を満たしているかどうかによって，そのシミュレーションに適した解法は異なる．

精度の低いシミュレーションにおいて，陽解法を用いるか陰解法を用いるかは，Δt の安定限界値 Δt_{max} と，結果を提示するまでの時間 ΔT の関係に依存する．$\Delta t_{max} \geq \Delta T$ の場合は陽解法を使うことができる．しかし，$\Delta t_{max} < \Delta T$ の場合は陰解法で解くか，ΔT を時間分割して陽解法で複数回計算する必要がある．

一方，細かいシミュレーションでは Δt を大きくとる場合を想定しなければならない．また，粗いシミュレーションより，細かいシミュレーションの方が Δt_{max} が小さくなる (付録 A.4)．このため，陽解法は適しておらず，陰解法を使う方が望ましい．

次に，系の変化が微小であるという判断を何の変化を基準にして行うかを述べる．オペレータへのシミュレーション結果の提示を視覚的手段で行っている場合は，可視化データの変化を基準にするのが最も適当であるが，それでは「系の変化が微小である場合に可視化結果を更新しない」という前提に矛盾する．そこで，現時点から系がオペレータの操作を全く受けないと過程したときの $T = \infty$ の値と，現時点でのシミュレーションの値の差のノルムを判断の基準値として用いることにする．

3.2.3 計算の流れ

ここでは系の状態が時々刻々と変化していった場合にシステムがどのようなインタラクションを起こすか説明する．ただし，シミュレーションのモデルは拡散方程式のような $t = \infty$ で系の状態が安定するような問題とする．

図 8 に示すような一連のインタラクションがあったとする．オペレータから操作を $\bigtriangledown$，クライアントからシミュレーション結果の提示を $\bigtriangleup$，計算サーバか

[1)] 本論文では，陽解法には前進差分によるオイラー法を用い，陰解法では後退差分によるオイラー法を用いている．

らのシミュレーション結果の提示を ◯ で表す．一番上に書かれた T_0，T_1，T_2，$\cdots$ は現実の時間の進行を，一番下に書かれた t_0，t_4，t_8，$\cdots$ は計算サーバにおけるシミュレーション上の時刻の進行を表している．t の添え字が T の添え字と等しい場合，計算サーバにおけるシミュレーション上の時刻は現実の時刻に対して遅れておらず，t の添え字が T の添え字より小さい場合は添え字の差の分だけ遅れていることを表している．また，クライアントの粗いシミュレーションは 1 タイムステップ以内に計算することができ，計算サーバでの細かいシミュレーションはシミュレーション上の時刻が現実の時刻に対して遅れたりしないかぎりは，4 タイムステップで計算できるものとする．

以下では時刻 T_0 から順にオペレータの操作に対して行われたシステムの動作を説明していく．

時刻 T_0**:** オペレータからの操作があり，その情報がクライアントと計算サーバの両方に送られる．

時刻 $T_1 \sim T_3$**:** クライアントで計算された結果が可視化され，オペレータに提示される．

時刻 T_4**:** 計算サーバで計算された結果が可視化され，オペレータに提示される．

時刻 T_5**:** クライアントで計算された結果が可視化され，オペレータに提示される．

時刻 T_6**:** クライアントで計算された結果は 1 ステップ前の時刻のシミュレーション結果から一定以上の変化が見られないので，可視化されない．オペレータの提示されているのは時刻 T_5 のクライアントでのシミュレーション結果である．

時刻 T_7**:** クライアントで計算された結果が可視化され，オペレータに提示される．

時刻 T_8**:** 計算サーバで計算された結果が可視化され，オペレータに提示される．

時刻 T_9**:** クライアントで計算された結果が可視化され，オペレータに提示される．また，オペレータからの操作があり，その情報がクライアントと計算サーバの両方に送られる．サーバでは時刻 t_{12} の状態の計算が開始されていたが，オペレータの操作によって系の状態が変わってしまったため，計算を中断し，時刻 t_9 の状態の計算を開始する．

時刻 $T_{10} \sim T_{11}$**:** クライアントで計算された結果が可視化され，オペレータに提示される．

時刻 T_{12}**:** クライアントで計算された結果が可視化され，オペレータに提示される．また，計算サーバでは時刻 t_9 の状態が計算され，次の時刻の計算が開始される．その時刻は現在遅れているシミュレーション上の時刻が現実の時刻に追いつくような値としなければならない．そこで，計算サーバでは現実の時刻に追いつくように 5 タイムステップ先の時刻 t_{17} の状態の計算が開始される．

時刻 T_{13}**:** クライアントで計算された結果が可視化され，オペレータに提示される．

時刻 T_{14}**:** クライアントで計算された結果は 1 ステップ前の時刻のシミュレーション結果から一定以上の変化が見られないので，可視化されない．オペレータの提示されているのは時刻 T_{13} のクライアントでのシミュレーション結果である．

時刻 T_{15}**:** クライアントで計算された結果が可視化され，オペレータに提示される．

時刻 T_{16}**:** クライアントで計算された結果は 1 ステップ前の時刻のシミュレーション結果から一定以上の変化が見られないので，可視化されない．オペレータの提示されているのは時刻 T_{15} のクライアントでのシミュレーション結果である．

時刻 T_{17}**:** 計算サーバで計算された結果が可視化され，オペレータに提示される．

時刻 T_{18}**:** クライアントで計算された結果は 1 ステップ前の時刻のシミュレーション結果から一定以上の変化が見られないので，可視化されない．オペレータの提示されているのは時刻 T_{17} の計算サーバでのシミュレーション結果である．

時刻 T_{19}**:** クライアントで計算された結果が可視化され，オペレータに提示される．

時刻 T_{20}**:** クライアントで計算された結果は 1 ステップ前の時刻のシミュレーション結果から一定以上の変化が見られないので，可視化されない．オペレータの提示されているのは時刻 T_{19} のクライアントでのシミュレーション結果である．

時刻 T_{21}**:** 計算サーバで計算された結果が可視化され，オペレータに提示される．

時刻 T_{22}**:** クライアントで計算された結果は 1 ステップ前の時刻のシミュレーション結果から一定以上の変化が見られないので，可視化されない．オペ

レータの提示されているのは時刻 T_{21} の計算サーバでのシミュレーション結果である．

2 回目の操作以降のインタラクションの方が，計算サーバからのシミュレーション結果をオペレータに提示できる機会が増えている．このように，オペレータの操作が途絶え，系の変化が少なくなればなるほど，計算サーバからのシミュレーション結果を提示する機会が増え，その結果シミュレーションの精度を上げることができる．以上のような流れで，系の変化に応じたシミュレーションの精度制御を行う．

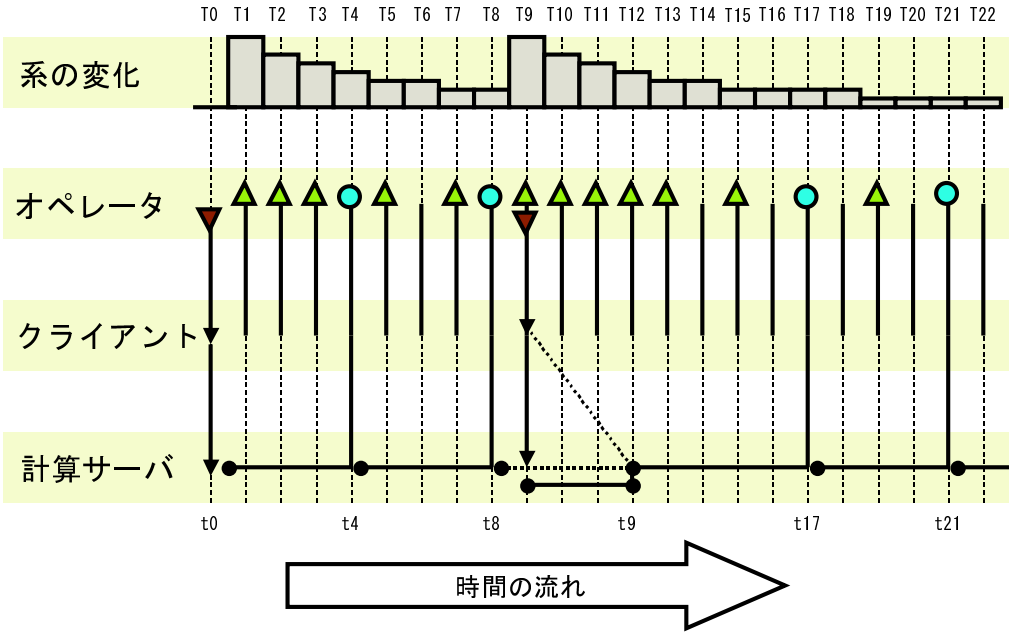


図 8: 系の変化に応じた精度制御

3.2.4 解法の選択

オペレータからの操作があった場合，クライアントも計算サーバも操作のあった時刻の状態を計算しなければならない．したがって，オペレータの操作が連続した場合，図 8 における時刻 T_9 のような動作が何度も行われることになる．クライアントではオペレータの操作にリアルタイム応答して，シミュレーション結果の提示を行うことができる．しかし，計算サーバでは 1 タイムステップ内の時間で 1 タイムステップ後の状態を求めることはできない．このため，計算機サーバ上の時刻は現実の時刻に対してどんどん遅れていくことになる．し

かし，オペレータの操作が途切れてしまえば陰解法で一気に追いつくことができる．そこで，なるべく計算時間が少なくて済む手法を用いて，連続した操作に対する計算サーバ上の時刻の遅れを軽減することを考える．

計算サーバがオペレータの操作のあった時刻の状態を計算するとき，現実の時刻に対する計算サーバ上の時刻の遅れが $n \times \Delta t_{max}$ で表されるとする．この時の計算サーバにおける計算方法は以下の二つが考えられる．

- $\Delta t_s = \Delta t_{max}$ として，陽解法を用いてで n 回を行う．
- $\Delta t_s = n \times \Delta t_{max}$ とし，陰解法を用いる．

どちらの計算時間の方が短いかは n の値によって決まる．すなわち，オペレータの操作があり，計算サーバが操作のあった時刻のシミュレーションの状態の計算しなければならいときの解法は前述の二つの方法のうち，計算時間の短い方を選択した方が，計算時間を短くすることができる．

3.3 本章のまとめ

本章ではシミュレーションの状態に応じて，シミュレーションの精度を動的に変更する方法として，オペレータ操作頻度に応じた精度制御，系の変化に応じた精度制御を提案した．静弾性解析，静磁界解析のような支配方程式に時間微分項の存在しない問題ではオペレータ操作頻度に応じた精度制御を行うことによりシミュレーション精度の動的制御が可能である．また，系の変化に応じた精度制御を行うことにより，熱伝導方程式のような支配方程式に時間微分項の存在する問題にも対応している．ただし，系の変化が常に一定であったり，定常的に振動現象を起こしているような系では本手法によって，精度を細かいシミュレーション結果をオペレータに提示することはできない．また，$\Delta t = \infty$ で発散するような系には適応できない．

第4章　汎用グラフィクスカードを用いた並列ボリュームレンダリングシステムの実装

本章では，第2章で述べたリアルタイム可視化システムの実現を目指し，汎用グラフィクスカード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う．まず，汎用グラフィクスカードを用いたボリュームレンダリング手法の説明をし，続いて並列化手法を行い，実装とその評価を行う．

4.1　汎用グラフィクスカードを用いたボリュームレンダリング

本項では汎用グラフィクスカードでサポートされているテクスチャマッピングとαブレンディング機能を用いたボリュームレンダリング手法について述べる．

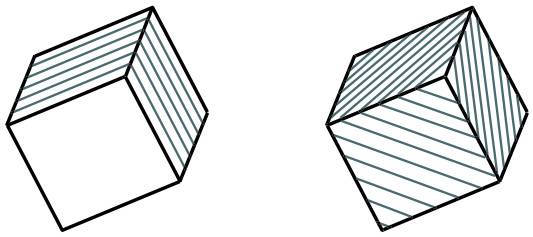
αブレンディングとは前面の画像要素の色と後面の画像要素の色を，α値の示す比率で線形内挿して，合成画像の色を算出する方法である．α値とは各ピクセルの色を決定するRGB以外のもう一つの要素で，素材の不透明度を決定するものであり，通常0.0から1.0の間の値をとる．1.0は素材が不透明で後ろにあるものは隠されるということを示し，0.0は素材が完全に透明で見えないことを表す．

一方，テクスチャマッピングとは物体の表面の質感を表現するために貼り付けるテクスチャと呼ばれる画像を，物体表面に貼り付ける方法である．テクスチャマッピングは，ピクセルを図形オブジェクトに結合させて，非常に複雑な画像を，大掛かりな図形モデルを構築することなく生成できる．ポリゴンの各頂点がテクスチャ画像上のどの点に対応しているのかをあらかじめ関数として持っており，ポリゴンを描画するときに，それを元にポリゴン上の一点一点がテクスチャ上のどの位置を参照しているのかを求めて，その位置に対応している色でポリゴンにマッピングしていく．

さて，式(4)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．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，汎用グラフィクスカードの機能を利用した高速処理が可能である[14]．グラフィク

スカードが 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 9-(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 9-(a))．

多くのグラフィクスカードでは 3 次元テクスチャとして扱うよりも，2 次元テクスチャとして扱う方が高速である．しかし，各座標軸ごとに 3 つのテクスチャを用意しなければならないため，3 次元テクスチャを使用する場合に比べて多くのメモリを必要とする．汎用グラフィクスカードのメモリ容量はボリュームレンダリングで必要とされるデータ量に比べてまだまだ少ないため，3 次元テクスチャをサポートしている場合にはボリュームデータを 2 次元テクスチャとして扱うよりも，3 次元テクスチャとして扱う方が有効である．本稿で用いるグラフィクスカード，GeForceFX5950 Ultra(表 1 参照) は 3 次元テクスチャをサポートしており，ボクセル値を 8bit の RGBA データとして描画した場合で $256 \times 256 \times 256$ のボリュームデータをリアルタイムに描画する性能を持つ．したがって，本論文では 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 9: テクスチャベースのボリュームレンダリング

4.2 並列化

汎用グラフィクスカード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となる．しかし，テクスチャを保持するグラフィクスカード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カード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は可能である．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カード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 [15].

4.2.1 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 10)．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 11).

4.2.2 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，ボリュームデータの局所性を利用してデータ量を削減することにより，メモリ利用の効率化を行うことを考える．他にもグラフィクスカードのメモリ容量を上回るサイズの大きなデータを描画する方法として，テクスチャデータをメインメモリに保持しておき，グラフィクスメモリに容量を越えない量だけデータを転送させる方法が考えられる．しかし，この方法を用いる場合，1 回の描画ごとにグラフィクスカードにテクスチャデータを転送する必要があるため高速な描画処理を行うことはできない．リアルタイムに可視化するためには，常に全てのデータをグラフィクスメモリに収まりきる形で保持しておくほうが望ましい．そこで，データを固定解像度のブロックで表現し，ブロック

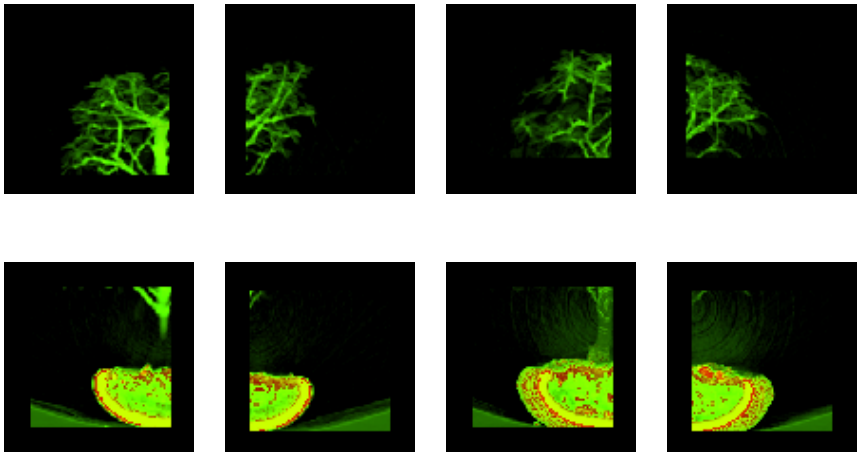


図 10: 中間画像 ($d = 2$)

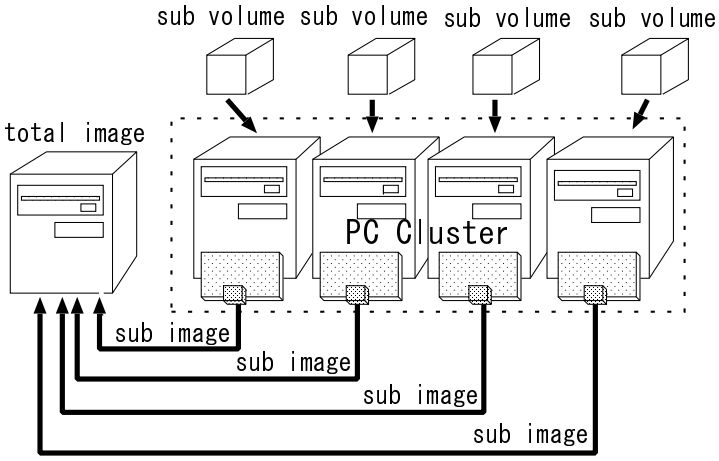


図 11: 並列化

内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 [16].

$N_x \times N_y \times N_z$ 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．誤差はサンプリングした値との自乗誤差と定義する．ブロックに分割されたボリュームを V とし，ブロック内の座標 (i, j, k) におけるボクセル値を $v_n(i, j, k)$，解像度を一つ下げたときのボクセル値を $v_{n-1}(i, j, k)$ とすると，各ブロックの誤差は

$$\sum_{(i,j,k)\in V} |v_n(i, j, k) - v_{n-1}(i, j, k)|^2 \tag{10}$$

と定義される．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 12 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，グラフィクスカードで使用するメモリを最小限に抑えることができる．

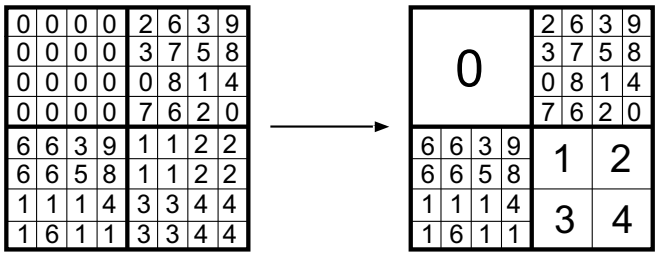
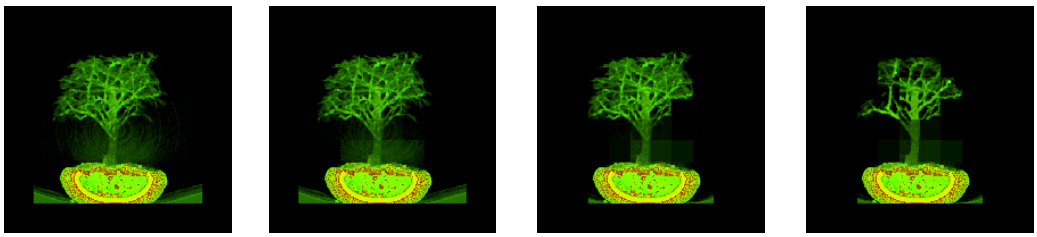


図 12: 適応的サンプリング

図 13 に実際のデータに対して誤差の許容値を変えながら，ブロック単位の適応的サンプリングを行い，データを削減した時の画質の変化を示す．ブロックの数は 8^3 である．この時，元データのサイズに対する適応的サンプリング後のデータのサイズは許容誤差 10% で 47.5%，20% で 30.1%，40% で 18.2% にまで縮小できている．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% が透明領域であり [17]，この部分のデータ量をブロック化により効果的に削減でき，ブロック単位でこの手法を適用した場合，分割するブロック数に応じてデータを縮小することができる．シミュレーション結果の可視化に対しても変化量の少ない領域に対しては，この手法により効果的にデータ量を削減することができる．一般にボリュームの縮小化を進めるほど画質は劣化

するが，この手法を用いるとほとんどが質の劣化なく，データ量を大幅に削減することができる．



(a) 元データ　(b) 許容誤差 10%　(c) 許容誤差 20%　(d) 許容誤差 40%

図 13: 適応的サンプリング

4.2.3　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 [17]．このことからも，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 14)．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．

4.3　実装

OpenGL1.2 グラフィクスライブラリを用いて PC クラスタ上に実装した．実装環境を表 1 に示す．なお，Client 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，Server 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．Server の

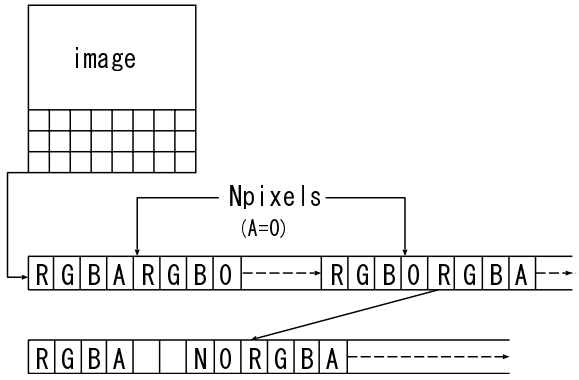


図 14: 中間画像の圧縮

ノード数は特に断りのない場合は 4 台で測定したデータを用いている．

表 1: 実装環境

	Client	Server
CPU	Pentium4 3.0GHz	Pentium4 3.0GHz
Memory	1.0GB	1.0GB
Gfx Card	GeForceFX5950 Ultra	GeForceFX5950 Ultra
Gfx Memory	256MB DDR	256MB DDR
NODE	1 台	4 台
OS	Debian Linux	
Network	1000BASE Ethernet	
Compiler	gcc+LAM6.5.9	

OpenGL1.2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．

$$C = B_s C_s + B_d C_d \tag{11}$$

$$A = B_s A_s + B_d A_d \tag{12}$$

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．

$$C = A_s C_s + (1 - A_s) C_d \tag{13}$$

$$A = A_s + (1 - A_s) A_d \tag{14}$$

OpenGL1.2 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[1)]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．

4.4 評価

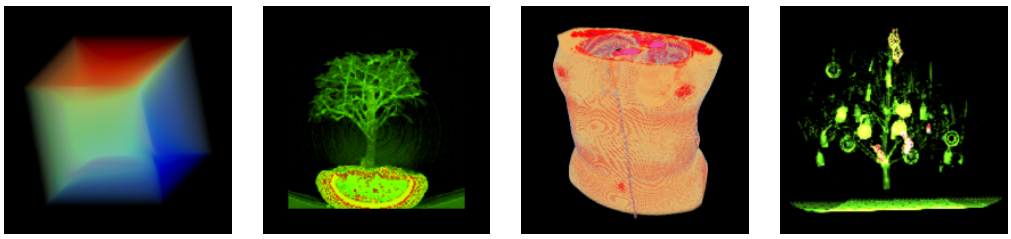
4.4.1 仕様・測定方法

様々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．

- Poisson
  ポアソン方程式を解いた結果をボリュームデータに変換したもので，サイズは $128 \times 128 \times 128$ である (図 15-(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 15-(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 15-(c))．

[1)] 最新の OpenGL では glBlendFuncSeparateEXT 機能を用いて，RGB 値と α 値に異なる係数を指定することが可能である．

- 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 15-(d)).



(a) Poisson (b) Bonsai (c) Chest (d) Tree

図 15: ボリュームデータ

不透明度については，Poisson のみすべてのボクセル値に対して 0.0625 とし，他のボリュームデータは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の測定方法を説明する．グラフィクスカードにおいてレンダリング処理はパイプラインで実行されるため，レンダリングの関数の実行時間はパイプラインへレンダリング命令の発行を行っている時間であり，正確なレンダリング時間とは異なる．そこで，連続してレンダリング要求を送ることにより，グラフィックスカードのパイプラインを渋滞させる．このようにすることにより，要求待ち間隔が定常状態になったときの間隔がレンダリングにかかる時間と等しくなる．また，グラフィクスカードによっては視点によって描画速度が異なるという性質を持つものも存在する．さらに，中間画像の圧縮効果は視点の位置に依存する．このため，レンダリングするオブジェクトを回転させながら，連続してレンダリング命令を発行させて求めた要求待ち間隔の平均値を描画速度とした．

4.4.2 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，並列化を行ったときの描画速度を表 2 に示す．ただし，適応的サンプリングと中間画像の圧縮は行っていない.「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．

Chest，Treen については 1 台非分割で描画することができなかった．これは生成されたテクスチャデータがグラフィクスカードに乗り切らなかったため，グラフィクスカードがテクスチャデータを受け付けなかったと考えられる．一方，1 台非分割の場合，Chest は低速ながらも描画できていることがわかる．生成されたテクスチャデータの一つ一つがグラフィクスカードに保持できても，すべてのテクスチャデータ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したと考えられる．Tree に至っては分割しても，プロセスが強制終了されて，描画できなかった．これは OS レベルの問題であると思われ，付録 A.1 で実装した環境においては Tree も 1 台非分割で描画可能である．

並列化を行うことにより，すべてのサンプルデータの描画速度が速くなっていることがわかる．ただし，ボリュームデータのサイズが小さいほど並列化効率は悪い．これは 1 台でも十分高速に描画できるサイズのデータを並列化したために，逆に通信処理など並列化に必要な処理がオーバヘッドになってしまったからであると考えられる．

通信性能について述べる．中間画像の 1 枚のデータサイズが 128^2 の 8bit の RGBA である．これがマスターノードに 8 枚通信されるわけであるから，中間画像の通信量は 1 回の描画に付き，512kB である．したがって通信にかかる時間は 4.0msec となる．ただし，実際には待ち時間などのオーバヘッドでさらに時間がかかっている．また，通信の実効性能のことを考えると，通信時間はさらにかかっていると考えられる．通信時間が大きなオーバヘッドなるぐらい高速な描画が可能な場合は並列化しない方が高速な描画が可能であると思われる．

しかし，並列化して描画速度が悪くなったとしても，実時間内での描画性能は十分に満たしている．第 2 章で述べたように，実時間インタラクティブシミュ

レーション環境においては，各計算機がそれぞれ可視化した方が望ましい．したがって，実時間インタラクティブシミュレーション環境の可視化システムとして用いる場合は，実時間内での描画性能が十分に満たされているならば，並列化した方が望ましい．

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
Poisson	126.4	106.9	144.4	138.8
Bonsai	35.6	29.8	89.9	80.4
Chest	-	0.83	32.2	27.1
Tree	-	-	0.75	7.54

4.4.3 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 $d = 2$，Server 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．

Poisson はブロックの数が 2^3 の時，Bonsai，Chest はブロックの数が 8^3 の時，Tree はブロックの数が 16^3 の時が最も効率が良い．Poisson，Bonsai，Chest については元々のボリュームデータが単純に並列化しただけで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，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，分割するブロックの数が大きい方がデータサイズを削減でき，高速な描画が可能になったと思われる．このように，ボリュームデータがグラフィクスメモリに入りきらないデータに関してはデータを圧縮することにより，高速な描画が可能となる．また，ボリュームデータがグラフィクスメモリに入りきるデータに関しては，分割して描画処理が増加したことによる描画速度の低下はほぼ見られない．この

ため，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，Poison に関してほとんどボリュームデータを圧縮できていない．これはボリュームデータの値が連続的に変化しており，ボリュームデータの圧縮が効かなかったためであると考えられる．誤差の閾値をさらに許容すれば圧縮は可能であるが，誤差の閾値を 30% まで上げて 74.7%，50% で 54.2% の縮小率である．このように，値が連続的に変化しており，その変化量が誤差の閾値を越えるようなデータには本手法は効きにくいと考えられる．

また，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，シミュレーション結果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．動的なデータに対してはデータの更新頻度が適応的サンプリングにかかる時間よりも大きい場合のみ有効である．したがって，実時間インタラクティブシミュレーション環境におけるリアルタイム可視化システムで適応的サンプリングを実装した方が望ましいのは

- シミュレーション結果のデータがグラフィクスカードに入りきらない．
- 適応的サンプリングにかかる時間が実時間内で終了する．

以上の二つを満たしたときであると考えられる．

4.4.4 中間画像の圧縮効果

中間画像の圧縮効果は，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．Server 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 $d = 2$，適応的サンプリングの誤差の許容値は 5% とした．ブロックの数は Poisson が 2^3，Bonsai，Chest8^3，Tree は 16^3 と適応的サンプリングの最も効果のあった数とした．

ボリュームデータのサイズが小さいほど描画速度が向上していることがわかる．これはレンダリングにかかる時間に対する，圧縮・通信にかかる時間の割合が増えているため，圧縮効果が大きくなっていると考えられる．一方，大きなデータに関しては中間画像の圧縮効果による描画速度の向上はあまり見られない．したがって，中間画像の圧縮効果はあるものの，レンダリングにかかる

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]	処理時間 [sec]	縮小率 [%]
Poisson	2^3	144.4	-	100
	4^3	138.8	0.024	100
	8^3	133.8	0.070	98.5
	16^3	68.8	0.103	93.8
Bonsai	2^3	89.9	-	100
	4^3	85.4	0.28	84.4
	8^3	96.4	0.47	53.9
	16^3	61.1	0.70	37.5
Chest	2^3	32.2	-	-
	4^3	27.1	0.81	100
	8^3	32.3	2.02	77.2
	16^3	30.3	2.89	66.8
Tree	2^3	0.74	-	-
	4^3	8.77	6.67	71.9
	8^3	13.8	8.16	44.9
	16^3	18.6	10.0	22.3

時間に対する，圧縮・通信にかかる時間の割合が少ない場合には描画速度の向上に及ぼす影響は少ないと考えられる．

本手法は表 1 に示す実装環境では 1000BASE Ethernet を用いているため，若干の速度向上しか見られなかったが，100BASE Ethernet の環境においては 2 倍近くの速度向上が得られている．100BASE Ethernet を用いた汎用グラフィクスカードによる並列ボリュームレンダリングシステムの評価については付録 A.1 を参照されたい．

4.4.5 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 $d = 2$，適応的サンプリングの誤差の許容値は 5%，ブロックの数は中間画像の圧縮効果の評価を行っ

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	
Poisson	144.4	161.7	59 ∼ 89
Bonsai	96.4	103.9	37 ∼ 64
Chest	32.3	33.1	42 ∼ 82
Tree	19.2	21.4	41 ∼ 73

たときと同じ数である．Poisson，Bonsai，Chest に関しては 30fps 以上の性能を出しており，リアルタイムに可視化できていることがわかる．また，Tree のようにグラフィクスメモリに入りきらないほどデータサイズが大きなボリュームデータは，描画速度が著しく低下してしまうが，適応的サンプリングを行い，ボリュームデータの圧縮を行うことにより，高速化が可能である．

表 5: 並列化の効果

Data	描画速度 [fps]		縮小率 [%]
	1 台	4 台	
Engine	109.1	161.7	93.8
Bonsai	58.9	103.9	84.4
Chest	1.98	33.1	77.2
Tree	-	22.4	22.3

4.5 本章のまとめ

本章では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，汎用グラフィクスカードによる並列ボリュームレンダリングの実装行った．この結果，サンプルとして使用したボリュームデータを 4 台のクラスタを用いてほぼリアルタイムに描画することができた．これは実時間インタラクティブシミュレーションの可視化システムの性能としては十分であると思われる．しかし，時系リュームデータの圧縮には今回実装した適応的サンプリングによる方法は向かないと考えられる．一方，中間画像の圧縮によって，若干の速度向上

が見られなかった．したがって，本論文で実装する実時間インタラクティブシミュレーション環境の可視化システムには中間画像の圧縮手法のみ実装し，適応的サンプリングは実装しないこととする．次章以降は，本章で説明したリアルタイム可視化システムを実装した，実時間インタラクティブシミュレーションの評価を行っていく．

第5章　システムの実装

本章では前章までに説明した，シミュレーションの精度を動的に制御するシステムと汎用グラフィクスカードによるリアルタイム可視化システムを実装した，実時間インタラクティブシミュレーション環境の構築を行い，その評価を行う．解析対象にはポアソン方程式と拡散方程式を用いた．以下ではこの二つのシミュレーションの実装・評価をそれぞれ行っていく．

5.1　ポアソン方程式

前章の提案システムに基づいた評価用のシステムを実装した．解析対象には3次元ポアソン方程式

支配方程式:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + Q = 0 \tag{15}$$

境界条件:

$$u = u_0 \tag{16}$$

の差分解法を用いる．

5.1.1　ポアソン方程式の離散化

ここでは式 (15) の離散化の過程を示す．領域は x，y，z 方向とも刻み幅 h で n 点に離散化し，

$$\begin{aligned} x_i &= ih \\ y_j &= jh \\ z_k &= kh \\ u_{i,j,k} &= u(x_i, y_j, z_k) \\ Q_{i,j,k} &= Q(x_i, y_j, z_k) \end{aligned}$$

とする．このとき，各偏導関数は中心差分をとることにより，以下のように離散化される．

$$\frac{\partial^2 u}{\partial x^2} \Rightarrow \frac{u_{i-1,j,k} - 2u_{i,j,k} + u_{i+1,j,k}}{h^2} \tag{17}$$

$$\frac{\partial^2 u}{\partial y^2} \Rightarrow \frac{u_{i,j-1,k} - 2u_{i,j,k} + u_{i,j+1,k}}{h^2} \tag{18}$$

$$\frac{\partial^2 u}{\partial z^2} \Rightarrow \frac{u_{i,j,k-1} - 2u_{i,j,k} + u_{i,j,k+1}}{h^2} \tag{19}$$

このため，ポアソン方程式は以下のような式に離散化される．

$$\frac{u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k} + u_{i,j,k-1} + u_{i,j,k+1} - 6u_{i,j,k}}{h^2} = -Q_{i,j,k} \tag{20}$$

式 (20) は n 元の連立一次方程式を表している．

5.1.2 実装

以上のように導出された式を用いてポアソン方程式を解く実時間インタラクティブシミュレーション環境を構築する．ポアソン方程式は支配方程式に時間微分項の存在しない問題であるため，オペレータの操作頻度に応じた精度制御を行えばよい．まず，クライアントではオペレータの操作に対し，リアルタイムな応答ができるように，16^3 のサイズのグリッドを用意して粗いシミュレーションを行う．計算サーバでは，16^3，32^3，64^3 の 3 つのグリッドを用意し，32^3 のシミュレーション結果と 64^3 のシミュレーション結果を段階的に可視化させる．式 (20) の連立一次方程式を解く反復解法には共役勾配法 (付録 A.3 参照) を用いた．解析領域は 1.0×1.0×1.0 の立方体で，領域の端を固定境界条件としている．また，領域内の 1 点に Q し，オペレータはこの Q を自由に動かすことができる．

オペレータによる Q の操作はキーボード入力によって行われ，Q の位置が 1 メッシュ分に相当するだけ移動したときに，位置の移動を計算機側が情報を受けとるようにしている．このようにすることで，オペレータの操作が緩やかである時に，計算機側が受けとる操作の数を減らすことができる．

また，シミュレーション結果は OpenGL グラフィクスライブラリによるボリュームレンダリング手法によって可視化する．クライアントでは逐次で，計算サーバでは 4 章で述べた並列化方法を用い，その結果をクライアントに送信する．実装した PC クラスタの環境は表 1 の通りである．

5.1.3 評価

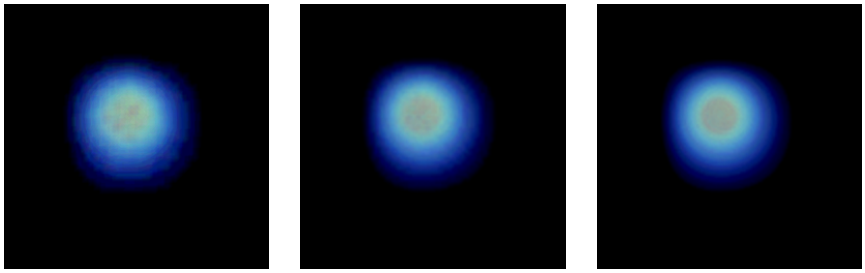
1 操作前の解を初期ベクトルに用いた場合と，反復法の初期ベクトルを常に一定値 (0) とした場合それぞれの，各レベルのシミュレーション結果が提示されるまでの処理時間を表 6 に示す．計算時間にバラつきがあるのは Q の位置に

よって系の収束の度合いが異なるからである．16^3 のシミュレーションに関しては，0.012 秒とリアルタイムなシミュレーションを可能としている．また，計算サーバから 32^3 の結果を得られるまでの計算時間は平均 0.26 秒，64^3 になると平均 1.43 秒である．オペレータの操作が多いときは，計算サーバでのシミュレーション結果はオペレータに提示されないが，オペレータの操作数が少なくなればなるほど，精度の細かいシミュレーション結果がオペレータに提示されており，オペレータの操作頻度に応じたシミュレーションの精度制御ができていることがわかる．また，1 操作前の解を初期ベクトルに用いた場合の方が，初期ベクトルを一定とした場合に比べて計算時間が短縮されていることがわかる．

なお，16^3，32^3，64^3 で行ったシミュレーション結果を図 16 に示しておく．

表 6: 処理時間

	処理時間 (sec)	
	初期ベクトル可変	初期ベクトル一定
Client(16^3)	0.012 ∼ 0.013	0.013 ∼ 0.015
Server(32^3)	0.19 ∼ 0.34	0.44 ∼ 0.70
Server(64^3)	1.13 ∼ 1.90	2.29 ∼ 2.67



(a) 16^3 の結果　(b) 32^3 の結果　(c) 64^3 の結果

図 16: ポアソン方程式の可視化

5.2 拡散方程式

次に 3 次元拡散方程式

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + Q = \frac{\partial u}{\partial t} \tag{21}$$

境界条件:

$$u = u_0 \tag{22}$$

の差分解法を対象とした実時間インタラクティブシミュレーション環境の実装を行う．拡散方程式は前項で実装したポアソン方程式と異なり，支配方程式に時間微分項の存在する時間依存型の問題である．拡散方程式の離散化は時間微分項を前進差分によって離散化するか，後退差分によって離散化するかによって，離散化された式を解く解法が陽解法となるか陰解法となるか異なってくる．以下では前進差分で離散化した場合と後退差分で離散化した場合の二つの離散化方法を示す.

5.2.1 前進差分による離散化

刻み幅 Δt で時間方向を離散化し，各タイムステップの時刻を $t_0, t_1 \cdots t_n \cdots$ とし，その時刻におけるシミュレーションの値を $u^0, u^1 \cdots u^n \cdots$ と表わす．空間方向の離散化はポアソン方程式の離散化と同様に，中心差分をとるとする．また，時間方向の離散化には前進差分をとる．このとき，式 (21) の時間微分項は以下のように離散化される

$$\frac{\partial u}{\partial t} \quad \Rightarrow \quad \frac{u^{n+1}_{i,j,k} - u^n_{i,j,k}}{\Delta t} \tag{23}$$

$$\tag{24}$$

ここで，式 (21) の各要素を空間方向に離散化した式

$$\frac{u^n_{i-1,j,k} + u^n_{i+1,j,k} + u^n_{i,j-1,k} + u^n_{i,j+1,k} + u^n_{i,j,k-1} + u^n_{i,j,k+1} - 6u^n_{i,j,k}}{h^2} \tag{25}$$

$$\tag{26}$$

の和を行列とベクトルの積で表し，$\boldsymbol{K}^n \boldsymbol{u}^n$ とすると，式 (21) は以下のような式に離散化される.

$$\boldsymbol{K}^n \boldsymbol{u}^n + \boldsymbol{Q}^n = \frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} \tag{27}$$

したがって，次式が導出される．

$$\boldsymbol{u}^{n+1} = (\Delta t \boldsymbol{K}^n + \boldsymbol{E})\boldsymbol{u}^n + \boldsymbol{Q}^n \tag{28}$$

ここで $\boldsymbol{E}$ は単位行列を表している．式 (28) は t^n の解を用いて時刻 t^{n+1} の解を直接的に計算できることを示している (陽解法)．

5.2.2 後退差分による離散化

前進差分同様に，空間方向の離散化には中心差分を用いる．一方，時間方向の離散化には後退差分を用いるとすると．式 (21) は以下のような式に離散化される．

$$\boldsymbol{K}^{n+1}\boldsymbol{u}^{n+1} + \boldsymbol{Q}^{n+1} = \frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} \tag{29}$$

となる．したがって

$$(\boldsymbol{E} - \Delta t \boldsymbol{K}^{n+1})\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \boldsymbol{Q}^{n+1} \tag{30}$$

となる．式 (30) は時刻 t^{n+1} の解を求めるために時刻 t^{n+1} の関係を使わなければならない．このため，連立一次方程式を解く必要がある (陰解法)．

5.2.3 実装

以上のように導出された式を用いて拡散方程式を解く実時間インタラクティブシミュレーション環境を構築する．拡散方程式は支配方程式に時間微分項の存在する時間依存型の問題であるため，系の変化に応じた精度制御を行う．解析領域は $1.0\times1.0\times1.0$ の立方体で，領域の端を固定境界条件としている．また，領域内の 1 点に Q し，オペレータはこの Q を自由に動かすことができる．

クライアントでは 32^3 のグリッドを用意し，式 (28) を用いた計算を行う．また，計算サーバでは，64^3 のグリッドを用意し，陽解法を用いた計算を行う．クライアント側の時間刻み幅 Δt_c の値は Courant 条件によって制限され，上式の拡散方程式の場合，$\Delta t_{max} = 1/6$ である (付録 A.4 参照)．本項では，$\Delta t_c = 1/6$ とし，これを秒間 30 回オペレータに提示することとする．ただし

$$|\boldsymbol{K}^{n-1}\boldsymbol{u}^{n-1} + \boldsymbol{Q}^{n-1}| - |\boldsymbol{K}^n\boldsymbol{u}^n + \boldsymbol{Q}^n| < 0.1 \tag{31}$$

を満たすタイムステップにおけるシミュレーション結果は可視化を行わない．な

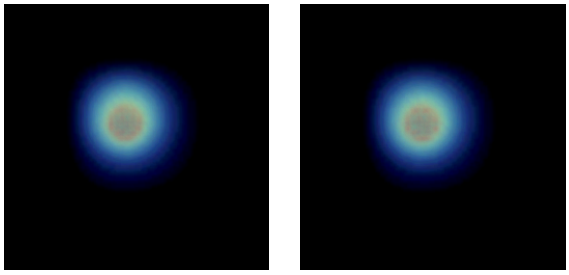


図 17: 変化量が 0.1 である 2 つのシミュレーション結果

お，$|\boldsymbol{K}^{n-1}\boldsymbol{u}^{n-1}+\boldsymbol{Q}^{n-1}|$ と $|\boldsymbol{K}^{n}\boldsymbol{u}^{n}+\boldsymbol{Q}^{n}|$ 差が 0.1 である二つの時刻のシミュレーション結果を可視化したものを図 17 に示す．視覚的判断ではほとんど区別がつかないことが見て取れる．

一方，計算サーバ側の時間刻み幅 Δt_s は計算サーバのシミュレーション上の時刻が現在の時刻に対してどれだけ遅れているかによって変化する．また，Δt_s 後の状態を求める場合に，Δt_s が一定値以下の場合は，陰解法を用いずに陽解法を用いる．ただし，Δt_s 後の状態は $\Delta t_s/\Delta t_max$ 回だけ陽解法計算を行うことにより求めるとする．図 18 にこの二つの解法の計算時間を示す．陽解法を用いた方が計算の高速化が望まれるのは $\Delta t_s \leq 11$ の時であることがわかる．また，陰解法で用いる反復解法には共役勾配法を用いた．実装環境はポアソン方程式の時と同様である．

5.2.4　評価

シミュレーション結果の変化によって，システムがどのように動作をするかを評価する．Q をタイムステップ 1，100，150 のときに操作した．この時のシステムのインタラクションを以下で述べる．まず，図 19 は各タイムステップにおける式 (31) の左辺の値を表示したものである．式 (31) を満たしたタイムステップにおけるシミュレーション結果が可視化されていないことがわかる．

次に計算サーバからのインタラクションについて述べる．オペレータの操作，計算サーバの動作をタイムステップ別に以下に示す．

時刻 T_0**:**　オペレータの操作で Q が移動する．

時刻 T_{33}**:**　サーバーからの結果が提示される．

時刻 T_{69}**:**　サーバーからの結果が提示される．

時刻 T_{100}**:**　オペレータの操作で Q が移動する．

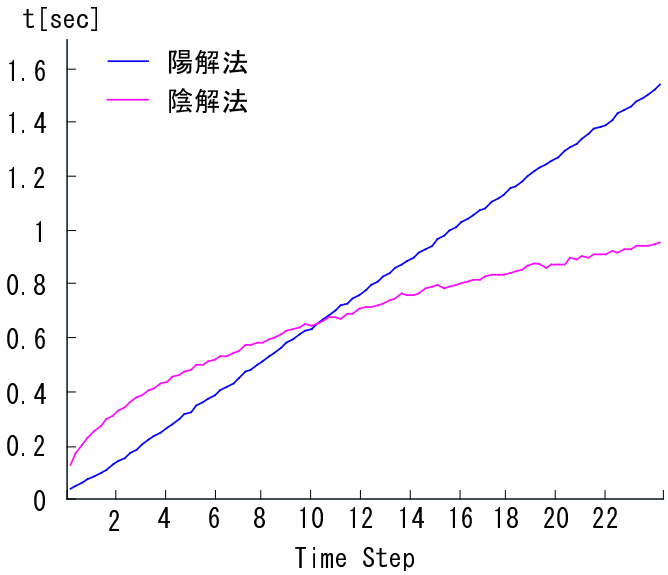


図 18: 解法による計算時間の違い

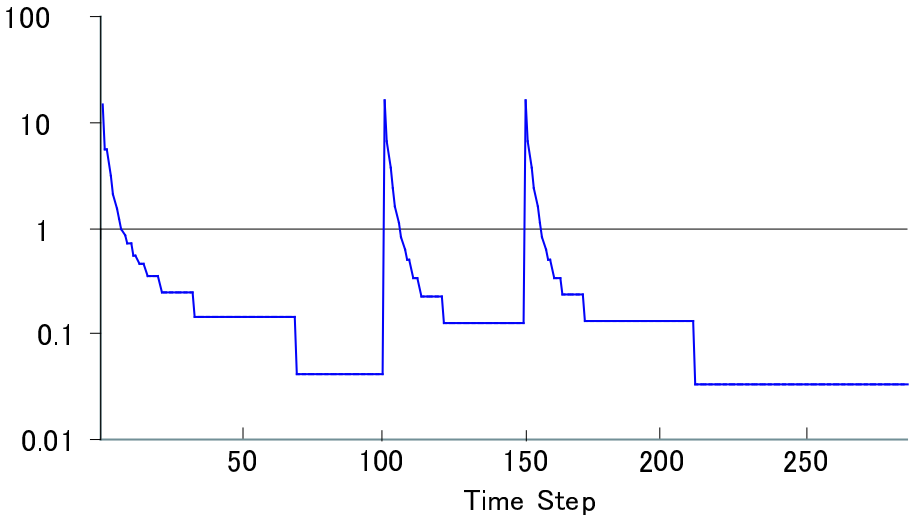


図 19: 可視化されたデータの変化量

時刻 T_{117}**:** サーバでタイムステップ 100 の状態が求まる.

時刻 T_{150}**:** オペレータの操作で Q が移動する.

時刻 T_{182}**:** サーバで時刻 150 の状態が求まる.

時刻 T_{224}**:** サーバーからの結果が提示される.

時刻 T_{264}**:** サーバーからの結果が提示される．

オペレータの操作が存在しない場合，約 33~42 タイムステップごとにサーバからの結果がオペレータに提示されていることがわかる．また，オペレータから操作のあった時刻 T_{100}，時刻 T_{150} の状態を計算サーバが求めるために，それぞれ 17 タイムステップ，32 タイムステップかかっていることがわかる．この違いはオペレータの操作があった時に計算サーバのタイムステップがどれだけ遅れているかによる．時刻 T_{100} でオペレータの操作があった時に計算サーバのシミュレーション上の時刻は T_{69} である．一方，T_{150} でオペレータの操作があった時に計算サーバのシミュレーション上の時刻は T_{100} である．図 18 に示すように，Δt が大きいほど，Δt 後の状態を求めるための計算時間は大きくなる．このため，計算サーバが，時刻 T_{150} の状態を求める時間は時刻 T_{100} の時に比べて長くなってしまったと考えられる．

次にオペレータの操作が連続した時のシステムの動作を評価する．Q をタイムステップ 1 から 40 まで連続して操作した．計算サーバが陰解法のみを用いて解いた場合，時刻 T_{40} の状態を計算サーバが求めたのが時刻 T_{190} であった．一方，陽解法と適宜使い分ける方法の場合，時刻 T_{40} の状態を計算サーバが求めたのが時刻 T_{59} であった．陽解法と陰解法を適宜使い分けることでオペレータの連続した操作に対する計算サーバの時刻の遅れを約 8 分の 1 に削減できていることがわかる．

5.3 本章のまとめ

本章では前章までに説明した，シミュレーションの精度を動的に制御するシステムと汎用グラフィクスカードによるリアルタイム可視化システムを実装した実時間インタラクティブシミュレーション環境の構築を行い，その評価を行った．

ポアソン方程式ではオペレータの操作頻度に応じた精度制御手法を用い，16^3，32^3，64^3 のサイズのシミュレーションを段階的にオペレータに提示することができた．

拡散方程式では系の変化に応じた精度制御手法を用い，計算サーバのシミュレーションの時間刻み幅を動的に変更することにより，シミュレーション精度の動的な制御を可能とした．ただし，オペレータの操作が連続した場合に，計算サーバのシミュレーション上の時刻がどんどん遅れてしまう．このため，陽解法と陰解法を適宜使い分け，計算時間が短い方を選択するようにした．この

ようにすることで，連続したオペレータの操作に対する，計算サーバ上の時刻の遅れを約 8 分の 1 に削減できた.

第6章　まとめ

我々は「仮想実験型/仮想体験型のシミュレーション環境」を提供する実時間インタラクティブシミュレーション環境の研究を行っている．

本論文ではオペレータの操作頻度に応じた精度制御手法と系の変化に応じた精度制御手法を取り上げ，シミュレーション精度の動的制御方法を提案した．

また，リアルタイム可視化システムとして，汎用グラフィクスカード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い，サイズ $512 \times 512 \times 512$ のボリュームデータをリアルタイムに可視化することができた．

そして，ポアソン方程式と拡散方程式を解析対象として，状態に応じた精度制御を行う実時間インタラクティブシミュレーション環境の実装を行った．ポアソン方程式ではオペレータの操作頻度に応じた精度制御手法を用い，シミュレーションの精度を段階的にオペレータに上げることができた．拡散方程式では系の変化に応じた精度制御手法を用いることにより，シミュレーション精度の動的な制御を可能とした．

今後の課題は，より実践的なシミュレーション対象に本手法を適用し，評価することである．

謝辞

本研究の機会を与えて頂いた，本研究室の富田眞治教授に深く感謝の意を表します．また，本研究に関して数々の有用な御指導，御意見を頂いた，森眞一郎助教授，中島康彦助教授，五島正裕助手，津邑公暁助手に深く感謝致します．さらに，共同研究者である高山征大氏，中田智史氏，篠本雄基氏，牧田俊明氏をはじめ，日頃様々な角度から助力して下さった京都大学大学院情報学研究科通信情報システム専攻富田研究室の諸兄に心より感謝致します．

参考文献

[1] 山内聡，他：”アクティブボリュームレンダリングに基づくシミュレーションステアリング”, 信学技報 CPSY2001-35，pp.1-8，Aug. (2001)

[2] 生雲公啓，他: ”実時間インタラクティブシミュレーションのための並列ボリュームレンダリング環境”, 平成14年度情報処理学会関西支部大会，pp.121-124，Nov. (2002)

[3] 森眞一郎，他: ”大規模データの並列可視化を支援するFPGA搭載PCIカード”, 第1回リコンフィギャラブルシステム研究会論文集，pp.15-20，Sep. (2003).

[4] 高山征大，他: ”非構造格子ボリュームデータの可視化における動的負荷分散”, 平成15年度情報処理学会関西支部大会，Nov. (2003)

[5] 丸山悠樹，他: ”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”, 情報処理学会 ARC-154-11，Aug. 2003.

[6] 丸山悠樹，他: ”実時間インタラクティブシミュレーションのための遅延隠蔽手法”，平成15年度情報処理学会関西支部大会，Nov. (2003)

[7] 矢川元基，他: ”有限要素法”，培風館，July. (1991)

[8] 森正武 著: ”数値解析 第2版”，共立出版，feb. (2002)

[9] 山本 恭弘 他: ”有限要素法を用いた心臓大動脈の触診シュミレーション”, 日本バーチャルリアリティ学会第6回大会論文集，2001.

[10] 中尾 恵 他: ”組織切開・開創シミュレーションによる手術計画支援”, 日本バーチャルリアリティ学会論文誌 TVRSJ Vol.8 No.2，2003.

[11] 中尾 恵 他: ”物理特性に基づいた高精細かつ対話的な軟組織切開手法”, 情報処理学会論文誌，Vol. 44，No.8，pp.2255-2265，2003.

[12] W. Pasman and F.W. Jansen: ”Latency layered rendering for mobile augmented reality.”, Proceedings of the 21th symposium on information theory, May 25-2, 2000

[13] 建部修見，小柳義夫 (1993): ”マルチグリッド前処理付き共役勾配法の並列化”, 並列処理シンポジウム JSPP98 論文集， pp.387-394.

[14] C.Rezk-Salama: ”Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Texturesand Multi-Stage Rasterization”, In Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware, 2000.

[15] S.Muraki: ”Next-Generation Visual Supercomputing using PC Clusters with VolumeGraphics Hardware Devices”, Proceedings of IEEE/ACM Supercomputer Conference, 2001.

[16] 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.

[17] M.Levoy: ”Efficient ray tracing of volume data.” ACM Trans. on Graphics, pp. 245-261, July 1990.

付録

A.1 100BASE Ethernet 環境における汎用グラフィクスカードによる並列ボリュームレンダリングシステムの評価

ここでは 100BASE Ethernet 環境における汎用グラフィクスカードによる並列ボリュームレンダリングシステムの評価を行う.

A.1.1 仕様・測定方法

特に断りのないかぎり，論文中の第 4 章と同様の仕様・表記で評価を行うこととする．実装環境を表 A.1 に示す.

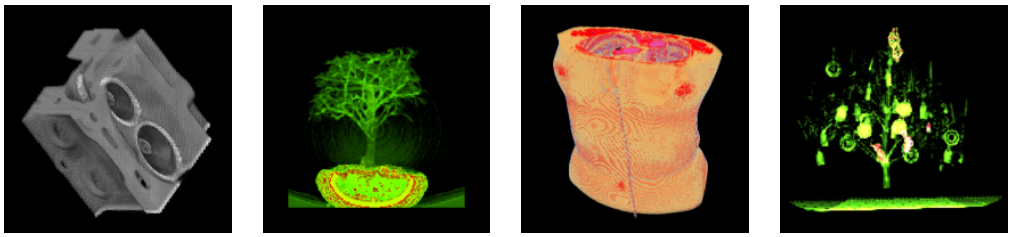
表 A.1: 実装環境

	Client	Server
CPU	Pentium4 1.8GHz	Pentium3 1.0GHz
Memory	512MB	512MB
Gfx Card	GeForce4 Ti4600	RADEON9700PRO
Gfx Memory	128MB DDR	128MB DDR
NODE	1 台	4 台
OS	Red Hat Linux	
Network	100BASE Ethernet	
Compiler	gcc+LAM6.5.9	

使用したボリュームデータのサンプルは次の 4 種類である．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.

- 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 A.1-(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 A.1-(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 A.1-(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 A.1-(d)).



(a) Engine (b) Bonsai (c) Chest (d) Tree

図 A.1: ボリュームデータ

A.1.2 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 A.2 に示す．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っていることがわかる，また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，劇的に描画速度が向上していることがわかる．

表 A.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

A.1.3 適応的サンプリングの効果について

誤差の許容値を 5%，分割数を $d = 2$，Server 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 A.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．

表 A.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	<u>18.2</u>	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	<u>17.8</u>	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	<u>15.8</u>	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	<u>12.7</u>	44.9	20.9
	16^3	8.14	22.3	22.5

A.1.4　中間画像の圧縮効果

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 A.4 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 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 つのデータほど性能向上が得られていない．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．

表 A.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

A.1.5　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 A.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．

A.2　マルチグリッド法

マルチグリッド法は反復法の一種であり，複数のグリッドを用いて収束を加速させる方法である．ガウスザイデル法や SOR 法に代表される反復法は漸化式を立て，ある初期値から始めて収束するまで漸化式を計算し続けるという方

表 A.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

法であるが，誤差の低周波成分の収束が悪いという性質を持つ．マルチグリッド法では，この低周波成分の収束率改善のために，まず解析を行うグリッドよりも粗いグリッドを用いる．粗いグリッドでは，元のグリッドでは低周波成分だったものが，高周波寄りになるため，元のグリッドでは収束しにくかった成分が収束しやすくなる．そのため，従来の反復法のみの場合に比べて収束が改善される．

$\boldsymbol{A}_1$，$\boldsymbol{u}_1$，$\boldsymbol{b}_1$ を解析対象から得られた係数行列，解ベクトル，右辺ベクトルであるとする．解くべき連立一次方程式は

$$\boldsymbol{A}_1\boldsymbol{u}_1 = \boldsymbol{b}_1 \tag{A.1}$$

となる．ここでマルチグリッド法では粗いグリッドを階層的に用意し，上記のルーチンを再帰的に呼び出すことで，収束の改善を行う．ここでは粗いグリッドを一つだけ用いた 2grid 法の説明を行う．まず，上式に対して反復法を適用し，数回反復を行った後，一時的な近似解を求める．この操作を pre-smoothing と呼ぶ．次にこの段階での残差 $\boldsymbol{r}_1$ を求める．これを粗いグリッド Ω_2 に渡して反復計算を行う．粗いグリッドにあう形に残差を変形する．この操作を restriction と呼ぶ．元のグリッドでの残差を $\boldsymbol{r}_1$， restriction 後の残差を $\boldsymbol{r}_2$ として，この restriction 演算子を $\boldsymbol{R}_{1,2}$ と表せば，この操作は

$$\boldsymbol{r}_2 = \boldsymbol{R}_1\boldsymbol{r}_1 \tag{A.2}$$

残差を粗いグリッドに渡したら，次はこのグリッドでスムージングを行う．このグリッドでの係数行列を $\boldsymbol{A}_2$，求める解ベクトルを $\boldsymbol{u}^2$ と置くと，restriction 後の残差を右辺として，以下の式をスムージングすることになる．

$$\boldsymbol{A}_2\boldsymbol{u}_2 = \boldsymbol{r}_2 \tag{A.3}$$

このスムージングにより，誤差の中で元のグリッドでは，低周波成分に当たる部分が低減されることになる．そして，粗いグリッドでのスムージングが終わったら，この解 $\boldsymbol{u}_2$ で元のグリッドの $\boldsymbol{u}_1$ を補正する操作を行う．これは解の prolongation と呼ばれる．prolongation 演算子を $\boldsymbol{P}_{2,1}$ と表すと，この操作は

$$\boldsymbol{u}_1 \leftarrow \boldsymbol{u}_1 + \boldsymbol{P}_{2,1}\boldsymbol{u}_2 \tag{A.4}$$

と表すことができる．そしてこの $\boldsymbol{u}_1$ を初期値として，再び

$$\boldsymbol{A}^1\boldsymbol{u}^1 = \boldsymbol{b}^1 \tag{A.5}$$

についてスムージングを行い，収束判定を行う．この操作は post-smoothing と呼ばれる．

以上の操作が，グリッドを二つ用いた 2grid 法と呼ばれる方法であるが，これを拡張して粗いグリッドを階層的に複数用意し，2grid 法を再帰的に呼び出すことで反復計算の収束の改善を行う方法がマルチグリッド法である．この呼び出し方にはいくつかの種類があり，一番簡単なものは，最も粗いグリッドまで残差を渡して求解した後，各段階において答えを修正しながら細かいグリッドへと戻っていく方法で，これは V-cycle と呼ばれる．その他に W-cycle， μ-cycle 等がある．

A.3 共役勾配法

A.3.1 アルゴリズム

共役勾配法 (CG 法) は Stiefel-Henstens が考案した降下法型の反復解法であり，理論上は n 元 1 次方程式 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ に対して n 回で真の解に収束することが保証されている．アルゴリズムを以下に示す．

1) 適当な初期ベクトル $\boldsymbol{x}^{(0)}$ を選んで次の計算を行う．

$$\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)} \tag{A.6}$$

$$\boldsymbol{p}^{(0)} = \boldsymbol{r}^{(0)} \tag{A.7}$$

2) $k = 0, 1, 2, 3, \cdots$ について $||\boldsymbol{r}^{(k+1)}||$ が収束するまで，次の手順を繰り返す．

$$\alpha_k = \frac{(\boldsymbol{r}^{(k)}, \boldsymbol{r}^{(k)})}{(\boldsymbol{p}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)})} \tag{A.8}$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)} \tag{A.9}$$

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{A}\boldsymbol{p}^{(k)} \tag{A.10}$$

$$\beta_k = \frac{\left(\boldsymbol{r}^{(k+1)}, \boldsymbol{r}^{(k+1)}\right)}{\left(\boldsymbol{r}^{(k)}, \boldsymbol{r}^{(k)}\right)} \tag{A.11}$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}k \tag{A.12}$$

A.3.2 実装

支配方程式:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + Q = 0 \tag{A.13}$$

境界条件:

$$u = u_0 \tag{A.14}$$

の差分解法を CG 法を用いて解いた結果を表 A.6 に示す．なお，評価は表 1 の Server マシン 4 台で行った．

表 A.6: CG 法を用いたポアソン方程式の計算時間

サイズ	時間 (sec)	反復回数
16^3	0.11	51
32^3	0.32	104
64^3	1.81	208

A.4 Courant 条件

拡散方程式

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \frac{\partial u}{\partial t} \tag{A.15}$$

を例に陽解法におけるの安定条件を求める．上式を離散化すると

$$\frac{u_{s-1,t,r} + u_{s+1,t,r} + u_{s,t-1,r} + u_{s,t+1,r} + u_{s,t,r-1} + u_{s,t,r+1} - 6u_{s,t,r}}{h^2} = \frac{u^{n+1}_{s,t,r} - u^n_{s,t,r}}{\Delta t} \tag{A.16}$$

となる．ここで $u^n_{s,t,r}$ を空間方向にフーリエ分解した値

$$u^n_{s,t,r} = \sum_{\theta}\sum_{\phi}\sum_{\varphi} g^n e^{ih(s\theta+t\phi+r\varphi)} \tag{A.17}$$

を考える．式 (A.17) を式 (A.16) へ代入すると

$$g^n \frac{e^{ih\theta} + e^{-ih\theta} + e^{ih\phi} + e^{-ih\phi} + e^{ih\varphi} + e^{-ih\varphi} - 6}{h^2} = \frac{g^{n+1} - g^n}{\Delta t} \tag{A.18}$$

$$\tag{A.19}$$

となる．したがって次式

$$g^{n+1} = g^n(1 - 2\frac{\Delta t}{h^2}(sin^2\frac{h\theta}{2} + sin^2\frac{h\phi}{2} + sin^2\frac{h\varphi}{2})) \tag{A.20}$$

が求められる．ここで，任意 θ，ϕ，φ に対して

$$|1 - 2\frac{\Delta t}{h^2}(sin^2\frac{h\theta}{2} + sin^2\frac{h\phi}{2} + sin^2\frac{h\varphi}{2})| < 1 \tag{A.21}$$

であれば安定である．したがって式 (A.15) の安定条件 (Courant 条件) は次式のようになる．

$$\Delta t < \frac{1}{6}h^2 \tag{A.22}$$

式 (A.22) は空間方向のシミュレーションの精度を上げるほど Δt の安定条件が制限されることを示している．ただし，式 (21) のように Q が存在するような場合は，安定条件が若干変化する．