修士論文

インタラクティブ流体シミュレーション

指導教員　　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

小松原 誠

平成18年2月3日

インタラクティブ流体シミュレーション

目次

インタラクティブ流体シミュレーション

小松原 誠

内容梗概

近年の計算機性能の急速な向上に伴い，インタラクティブな実時間シミュレーションへの期待が高まっている．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．流体シミュレーションは，工学，医学等の分野で重要視されており，研究が進められているが，計算量が多く，オペレータによる操作に動的に対応することが難しいため，流体シミュレーションをインタラクティブに実行しようとする試みはほとんど例がない．インタラクティブな流体シミュレーションを実現するためには，オペレータの操作に動的に対応できる数値計算モデルを使用する必要があり，実時間での計算結果の提示のためには計算の高速化が必要である．

本稿では，数値計算力学の新しい手法として注目され始めている格子ボルツマン法による流体シミュレーションをグラフィクスプロセッサユニット (GPU) 上に実装することで，これらの課題を解決できることを示した．

まず，格子ボルツマン法の計算を GPU のテクスチャマッピングの機能を利用することで効果的に GPU 上に実装できることを示し，GPU を用いることによって CPU による計算よりもはるかに高速な計算を実現できることを確認した．さらに，触覚デバイスを用いて流体シミュレーションに操作を加え，結果を視覚及び力覚としてオペレータに提示するシステムを実装したことにより，インタラクティブシミュレーションの適用範囲を広げ，オペレータがより現実に近い仮想現実を体験できるシステム開発の可能性を示した．また，並列計算を適用する手法を示し，実装報告を行うことで，大規模なインタラクティブシミュレーションシステムに流体計算を組み込める可能性を示唆した．

Interactive Fluid Simulation

Makoto KOMATSUBARA

Abstract

Recently, interactive simulation is coming to be done in the various fields, caused by the rapid improvement of the computer performance. This system needs to perform a simulation in realtime corresponding to the interactive operation by the operator and to show the result at once by means of vision and others needs. Fluid simulation is regarded as important in many fields such as engineering or medicine, nevertheless fluid simulation is rarely applied for interactive simulation, because of its high computational cost and difficulty of coping with interactive operation. To be applied for fluid simulation to interactive simulation, it is necessary to use suitable numerical model and to accelerate computing.

This paper shows that the lattice Boltzmann method and using the graphics processing unit (GPU) make it possible to solve this problem.

To begin with, this paper shows a technique of implementation of fluid simulation by lattice Boltzmann method onto GPU by using a function of texture mapping. The experimental results based on the current implementation show that the GPU outperforms the CPU. Implementation of interactive fluid simulation by using GPU and a device for the sense of touch shows possibility of effective interaction between a simulation and an operator. We also examine a technique of parallel computation connected with this research for the realization of large-scale interactive simulation.

第1章　はじめに

近年の計算機性能の急速な向上に伴い，インタラクティブな実時間数値シミュレーション [1] への期待が高まっている．中でも実時間内での数値シミュレーションは，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える「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．

我々の身の回りには水や空気をはじめとした流体が多く存在し，特に工学や医学等の分野で流体の振舞いを知ることは重要な課題となっている．流体の振舞いに関しては，ごく単純な場合を除き，解析的に解を求めることは不可能であるため，計算機を用いた数値シミュレーションが利用される．しかし，流体シミュレーションをインタラクティブに実行しようとする試みは，ほとんど例がない．この要因として，流体計算は計算量が多く，実時間での実行が困難であることが挙げられる．また，格子の生成などにおいて，境界形状などに応じた問題の定式化が困難であり，オペレータの動的な操作に対応することが難しい．

インタラクティブな流体シミュレーションを実現するためには，オペレータの操作に動的に対応しうる数値計算モデルを使用することが必要となる．格子ボルツマン法 (Lattice Boltzmann Method)[2][3][4] は，数値流体力学の比較的新しい手法であり，局所的で単純な規則に従って計算が進む手法であることから，インタラクティブな操作に対応することが比較的容易であると考えられる．

また，実時間で計算結果を提示するためには，計算を高速化する必要がある．近年，汎用グラフィクスカードに搭載されるグラフィクスプロセッサユニット (Graphics Processing Unit，GPU) の性能は著しく向上しており，浮動小数点演算を正式にサポートするとともに，従来固定であったグラフィクス演算処理フローのプログラム制御までも可能となっている [5][6]．このため，GPU の本来の用途であるグラフィクス用途にとどまらず，GPU を汎用の数値計算にも応用しようとする試みが始められている [7]．

本研究では，計算モデルとして格子ボルツマン法を用い，計算を GPU 上に

実装することによって計算の高速化を図ることで，インタラクティブな流体シミュレーションの実現を目指した．第 2 章では，研究の背景として，現状の課題を概観し，本研究で採用した格子ボルツマン法について説明する．また，GPU を汎用数値計算に応用するための一般的な手法について説明する．第 3 章では，格子ボルツマン法による流体シミュレーションを GPU 上に実装する手法を説明するとともに，目的の実現のための GPU の有用性を確認する．第 4 章で触覚デバイスを用いたインタラクティブ流体シミュレーションの実装報告を行い，第 5 章ではインタラクティブ流体シミュレーションを 3 次元に適用した実装報告を行い，第 6 章ではさらに大規模なシミュレーションへの適用を目指して並列化の検討を行う．

第2章　研究の背景

本章では，インタラクティブシミュレーションが抱える可能性と課題について紹介し，本研究で採用した格子ボルツマン法の計算手法について説明する．また，インタラクティブシミュレーションにおいては計算結果を実時間で提示するための高速計算が必要である．本研究では，近年目覚しく性能が向上しているGPUを用いることによって計算の高速化を目指したが，GPUを数値計算に応用する一般的な手法を紹介する．

2.1　手術シミュレータ

インタラクティブシミュレーションの適用例として，医療分野における手術シミュレータ[8]について紹介する．医療の高度化，多様化が進む中，新たな教育法の確立が求められている．このような中，現実の手術を模擬した仮想環境を用いて手術に必要な知識や技術の習得を効率的かつ安全に行う目的で手術シミュレータの実現に向けた研究が進められている．

人体は体重に対して約92%の体液などの流体で占められており，中でも血液の流れについては，日本での三大死因である心疾患や脳血管障害を引き起こす要因となりうることから，長年研究が進められてきた．近年では，このような人体における流体現象に数値シミュレーションを取り入れる研究が進められている[9]．

インタラクティブな手術シミュレータに流体シミュレーションを取り入れることができれば，シミュレータによってより現実に近い仮想体験が得られ，教育目的にとどまらず，困難な技術や新しい技術を伴う手術におけるリハーサルが可能となり，術前計画にも役立たせることができる．しかし，流体シミュレーションをインタラクティブに実行することは現状では困難であるとされている．

2.2　流体運動の数値シミュレーション

一般に流体の運動は，質量，運動量，エネルギーの保存に関するニュートン力学の基本法則に基づいた，Navier-Stokes方程式によって定式化される．Navier-Stokes方程式は一般に非線形の偏微分方程式となり，一方向の流れ，あるいは環状の流れなど方程式が線形になるものを除いて，厳密解はほとんど知られていない．したがって，差分法，有限要素法，境界要素法などの手法によって，

Navier-Stokes 方程式を単純化した数理モデルを解くことになる．モデル化の過程で物体周りの空間を格子によって離散化するが，境界形状が複雑になれば，適切な格子の生成は非常に困難である．非構造格子 (三次元では通常 4 面体セルの集合を用いる) を使うことで精度や速度が犠牲になるが，格子生成の自由度が高まり，ある程度格子の自動生成は可能になる．しかし，本研究で扱うインタラクティブシミュレーションでは，シミュレーション実行中のオペレータによる操作により，境界位置を動的に変更できることを前提としている．非構造格子を用いたとしても，時間刻み毎に境界位置の変化に応じた格子の再生成が必要となり，実行速度が大きく低下する．

このような Navier-Stokes 方程式を近似によって解く従来のアプローチに対して，比較的新しい手法である格子ボルツマン法が注目されつつある．格子ボルツマン法では，流体の振舞いを局所的で単純な規則による仮想粒子の運動として捉えることで，Navier-Stokes 方程式を解いた場合と同等な結果を得ることができる．そのため，巨視的視点からの格子生成を含めた定式化が困難である場合にも適用できる可能性がある．

2.2.1 格子ボルツマン法

格子ボルツマン法においては，原則として空間は規則的な格子によって一様に離散化される[1)]．時間についても離散化し，時間刻み Δt ずつ時間が進む．流体を，格子上に存在する仮想粒子の集合と捉える．粒子は格子点に存在し，衝突と呼ばれる過程によって一部の粒子は速度の方向を変え，Δt の間に静止，あるいは別の格子点へ移動する（これを並進と呼ぶ)．粒子は個別に捉えるのではなく，時刻 t，位置ベクトル r の格子点上において粒子速度ベクトル e_i を持つ粒子数密度分布を，実数値を持つ分布関数 $f_i(r,t)$ で表す．一連の粒子運動は，

$$f_i(r + e_i\Delta t, t + \Delta t) = f_i(r,t) + \Omega_i(f(r,t)) \tag{1}$$

と表され，これを格子ボルツマン方程式と呼ぶ．式 (1) の左辺は，衝突終了後に粒子が $e_i\Delta t$ 離れた近接の格子点に移動する並進過程を表している．$\Omega_i = \Omega_i(f(r,t))$ は，衝突による粒子分布の変化を表す衝突演算子である．Ω_i は，局所的な分布関数にのみ依存している．

格子点における密度 ρ と運動量 ρu は，粒子速度モーメントとして分布関数 f_i

[1)] 最近では不規則格子への適用も研究されている [10].

を用いて次のように定義される．

$$\rho = \sum_i f_i, \qquad \rho u = \sum_i f_i e_i \tag{2}$$

ここで u は，格子点における巨視的流速である．これら巨視的変数は，式 (1) による衝突・並進過程の後，毎回算出される．

Ω_i は，それぞれの格子点において，質量と運動量を保存するように決定されなければならない．すなわち，

$$\sum_i \Omega_i = 0, \qquad \sum_i \Omega_i e_i = 0 \tag{3}$$

Ω_i に関して，さまざまなモデルが提案されているが，単一緩和時間近似を用いた格子 BGK モデルと呼ばれるモデルが広く使用されている．このモデルでは，格子ボルツマン方程式は単一時間緩和係数 τ を用いて，

$$f_i(r + e_i\Delta t, t + \Delta t) = f_i(r, t) - \frac{f_i(r, t) - f_i^{eq}(r, t)}{\tau} \tag{4}$$

と表される．動粘性係数 ν は τ と音速[1)] c_s を用いて，

$$\nu = \left(\tau - \frac{1}{2}\right) c_s \Delta t \tag{5}$$

と与えられる．$\nu > 0$ であることから，$\tau > 1/2$ でなければならない．格子ボルツマン法では完全な非粘性の流体を扱うことはできず，τ が 1/2 に近づくと，数値安定性が失われる．f_i^{eq} は，局所平衡分布関数と呼ばれ，次に示す Maxwell-Boltzmann 平衡分布関数から導くことができる．

$$g^{eq} = \frac{\rho}{(2\pi RT)^{\frac{D}{2}}} \exp\left[-\frac{(\xi - u)^2}{2RT}\right] \tag{6}$$

D は空間の次元，ξ は粒子速度，R は気体定数，T は温度である．巨視的流体変数は，分布関数 g の速度モーメントによって与えられる．すなわち，

$$\rho = \int g d\xi, \qquad \rho u = \int \xi g d\xi \tag{7}$$

ここで流速 u の大きさが c_s に比べて十分に小さい (すなわち Mach 数が十分小さい) と仮定し，式 (6) を $O(u^2)$ まで展開すると，

$$g^{eq} = \frac{\rho}{(2\pi RT)^{\frac{D}{2}}} \exp\left(-\frac{\xi^2}{2RT}\right)\left[1 + \frac{\xi \cdot u}{RT} + \frac{(\xi \cdot u)^2}{2(RT)^2} - \frac{u^2}{2RT}\right] \tag{8}$$

[1)] ここでの音速とは密度変動の伝播速度であり，熱力学的な変数としての音速とは異なる．

音速を $c_s = c/\sqrt{3}$ として与え，粒子速度を e_i に限定することによって離散化すれば，式 (8) は，

$$f_i^{eq} = w_i \rho \left[1 + \frac{3(e_i \cdot u)}{c^2} + \frac{9(e_i \cdot u)^2}{2c^4} - \frac{3u^2}{2c^2} \right] \tag{9}$$

と書き換えられる．ただし，w_i は重み関数であり，粒子の速さによって決まる．

$$w_i = \frac{1}{(2\pi c^2/3)^{\frac{D}{2}}} \exp\left(-\frac{3e_i^2}{2c^2} \right) \tag{10}$$

e_i，w_i は，Navier-Stokes 方程式に適合するように定められる．ここで用いた $||u|| << c_s$ の仮定は，インタラクティブシミュレーションを実現する上での制限として注意しなければならない．

格子ボルツマン法の計算手法は，便宜上，衝突，並進，巨視化の 3 つの過程の繰り返しであると捉えることができる．このうち他の格子点との情報のやりとりは並進過程においてのみであり，残りの過程の計算は全て格子点内で行われる．格子 BGK モデルにおいては，格子点内の他の速度方向を表す分布関数との干渉すら考慮する必要がない．式 (1) もしくは式 (4) の左辺は並進を表すが，e_i の種類は極めて限られており，使用する速度分布モデルによるが，多くは $e_i \Delta t$ が隣接格子点までの距離のみを指す．このように，格子ボルツマン法は局所的な計算手法であるため，複雑境界の形状に対しても対応が容易であり，並列計算にも向いているといわれている．

2.3 GPU を用いた汎用数値計算

コンピュータゲームに代表されるエンターテインメント系アプリケーションからの需要に支えられ，GPU の性能は著しい向上を続けている．表 1 にグラフィクスカードの主な諸元の例を示す．GeForceFX5900Ultra は，Prescott コアの Pentium4 とほぼ同じトランジスタ数であり，ピクセル計算時の理論最大性能は Pentium4 を上回るほどである．また，GeForce6 では，チップサイズが Pentium4 の 2 倍以上になるとともに，理論最大性能も GeForceFX5900Ultra の約 2 倍弱に達している．このような GPU の高い演算性能を，本来のグラフィクス用途のみならず汎用数値計算に応用しようとする試みが始められている．本節では，GPU の基本アーキテクチャと，汎用数値計算への応用手法を紹介する．

表 1: グラフィクスカードの主な諸元の一例

	GeForceFX 5900Ultra	GeForce6 6800Ultra
Core Clock	450MHz	400MHz
Pipeline Unit	頂点 4/ピクセル 8[1)]	頂点 6/ピクセル 16
頂点計算性能	338M vertex/sec	600M vertex/sec
ピクセル計算性能	3.6G pixel/sec	6.4G pixel/sec
Memory Clock	425MHz DDR2	550MHz DDR3
Memory バンド幅	27.2GB/s	35.2GB/s
Memory bit 幅	256bit	
Memory 容量	256MB	
演算精度	IEEE 32bit 単精度浮動小数点	
API サポート	OpenGL1.4,DirectX9.0	
シェーダ	VS2+/PS2+	VS3/PS3

[1)]32bit float で計算する場合は 4

2.3.1 GPU の基本アーキテクチャ

図 1 に，GPU における基本的なレンダリング処理の概念図を示す．GPU は，頂点プロセッサとフラグメントプロセッサを持つ．CPU から描画のためのデータが送られると，まず頂点プロセッサで頂点座標の幾何変換が行われ，物体がスクリーンに投影されたときの位置および形状が決定される．次に，投影物体をラスタライズ処理により，スクリーンの個々のスキャンラインに対応するフラグメント (投影物体をスキャンラインで切ったときの断片) を計算する．フラグメントに対してテクスチャの貼り付けやフィルタ処理，シェディング処理がフラグメントプロセッサによって行われた後，フレームバッファと呼ばれる，グラフィクスメモリ上のディスプレイに表示するための特定領域に格納される．

テクスチャとは物体の表面の質感を表現するために貼り付けるデータであり，物体を表す図形オブジェクトにテクスチャを貼り付けることをテクスチャマッピングと呼ぶ [11]．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

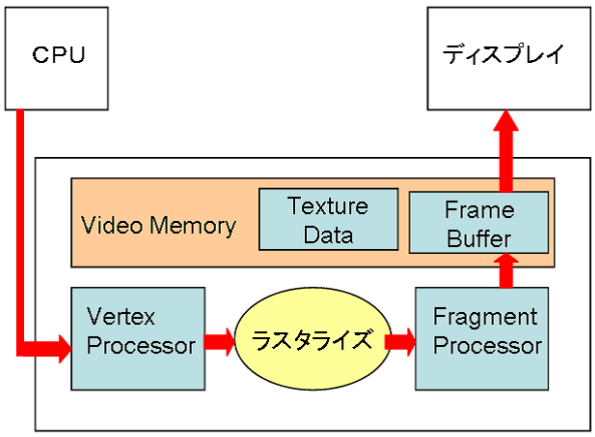

図 1: GPU のレンダリング処理の概念図

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．テクスチャデータとして 1 ピクセルあたり色および透明度を表す RGBA の 4 つの値を持つことができ，フラグメントプロセッサでは，RGBA の 4 つの値を同時に計算することができる．

近年の GPU では，両プロセッサのレンダリングパイプラインをプログラムにより動的に制御することが可能となり，32bit 浮動小数点数を扱うことも可能になった．また，Cg[12] に代表される GPU プログラミングのための高級言語が提案されており，プログラム開発効率が改善されている．このような背景から，GPU で汎用数値計算を行う試みが始められている．

2.3.2 GPU を用いた数値計算手法

GPU での数値計算は，主にフラグメントプロセッサのテクスチャマッピング機能を利用することで行われている．描画の際，フラグメントプロセッサは，テクスチャデータをグラフィクスメモリから読み込むことができる．2 次元配列をテクスチャとして与え，同じ大きさの長方形ポリゴンにマッピングすれば，2 つの 2 次元配列の要素毎の演算が可能になる．図 2 は，テクスチャマッピングを用いた基本演算の例である．$N \times N$ の 2 次元配列 $X[N][N]$ と $Y[N][N]$ に対して，各要素毎の演算を行って $Z[N][N]$ を計算する例であり，フラグメントプロセッサのパイプライン性能を最も発揮できる演算パターンである．前述のように，フラグメントプロセッサでは 1 ピクセルあたり RGBA に相当する 4 つの値を同時に計算することができるため，$N \times (N/4)$ サイズのテクスチャに $N \times N$ の 2 次元配列を埋め込むことで，最大 4 倍の性能を得ることができる．

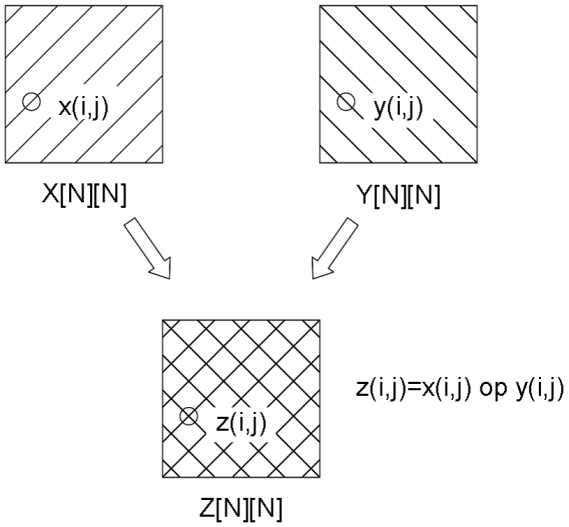

図 2: テクスチャマッピングを用いた基本演算

通常，フラグメントプロセッサでの演算結果はフレームバッファへ出力されるが，フレームバッファは 1 ピクセルの RGBA それぞれにつき 8bit の情報しか持つことができない．一方，フレームバッファとは別に，グラフィクスメモリ内にピクセルバッファと呼ばれるオフスクリーンバッファを持つことができる [13]．ピクセルバッファは 1 ピクセルの RGBA それぞれにつき 32bit の情報を持つことができるため，フラグメントプロセッサの出力先としてピクセルバッファを選択することにより，計算の精度を保つことができる．ピクセルバッファの内容は，テクスチャに上書きすることによって，テクスチャを更新することができる．これによって，描画内容として出力された計算結果の保持が可能となり，反復計算が実現できる．

GPU のグラフィクスメモリから CPU へデータを転送することも可能である．ただし，CPU から GPU へのデータ転送では，AGP8x の場合，2GB/s に近い転送速度が得られるが，GPU から CPU へのデータ転送では，数 100MB/s 程度の転送速度しか得られない．また，GPU のパイプラインを一旦フラッシュした後でなければ GPU から CPU へのデータ転送が行えないという制約がある．したがって，GPU の性能を十分に発揮するためには，GPU から CPU へのデータ転送は極力抑える必要がある．

2.3.3 誤差の問題

前述のように，近年の GPU では 32bit 浮動小数点を扱うことができるが，Hillesland らの報告によれば，加算，減算，乗算，除算のいずれにおいても IEEE 標準に厳密には従っていない [14](表 2)．特に除算における誤差が大きいことがわかる．さらに加算において特に顕著に見られる丸め方向の偏りは，繰り返し演算による誤差の蓄積を顕著にする原因となることが予想できる．

表 2: NV30 での浮動小数点演算における最下位ビットの誤差

Addition	[-1.000, 0.000]
Subtraction	[-0.750, 0.750]
Multiplication	[-0.782, 0.625]
Division	[-1.199, 1.375]

IEEE 標準では，[-0.5, 0.5] となる

第3章　格子ボルツマン法のGPU実装

インタラクティブシミュレーションの実現のためには，入力に対する瞬時の応答が求められるため，計算の高速化は必須である．本章では，インタラクティブシミュレーションへの取り組みの前段階として，格子ボルツマン法による流体シミュレーションをGPU上に実装することによって，計算の高速化を試みる[15]．計算例として，流体シミュレーションにおいて一般に多く計算されている2次元円柱周りの計算を行う．

3.1　計算モデル

3.1.1　格子と粒子速度モデル

2次元の格子BGKモデルとして多く用いられているD2Q9モデル[16]を用いる．このモデルにおいては，正方形格子で空間を分割し，粒子は静止を含む9種類の速度を持つとする．粒子速度e_i $(i = 0, 1, \cdots, 8)$は，以下のように定義される(図3)．

$$e_i = \begin{cases} (0,0) & i = 0 \\ c\,(\cos[(i-1)\pi/2], \sin[(i-1)\pi/2]) & i = 1 \sim 4 \\ \sqrt{2}c\,(\cos[(2i-9)\pi/4], \sin[(2i-9)\pi/4]) & i = 5 \sim 8 \end{cases} \tag{11}$$

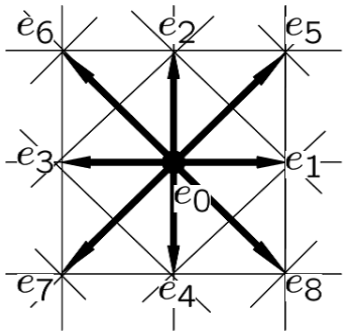

図3: D2Q9モデルにおける仮想粒子の速度ベクトルe_i

重み関数 w_i は，以下のように決められる．

$$w_i = \begin{cases} 4/9 & i = 0 \\ 1/9 & i = 1 \sim 4 \\ 1/36 & i = 5 \sim 8 \end{cases} \tag{12}$$

格子ボルツマン法における衝突，並進，巨視化の一連の過程を 1 タイムステップと呼ぶことにする．K タイムステップ時の時刻は $t = K\Delta t$ となる．

3.1.2 境界条件

固体壁における境界条件

固体壁における境界条件には，bounce-back 境界条件 [17] を採用した．bounce-back 境界条件では，壁面に向かう粒子は，固体壁に衝突することにより 180° 向きを変えるものとする．全ての速度成分について同様の処理を行うと，壁面上での運動量が 0 になる．すなわち，壁面における粘着条件を表している．本実装においては取り扱う固体壁は円柱であるが，格子に沿った形状に近似した．円柱内部の格子点を固体格子点，円柱外部の格子点を流体格子点とし，壁面は，流体格子点に隣接する固体格子点を結ぶ格子上に存在するものとみなした (図 4)．

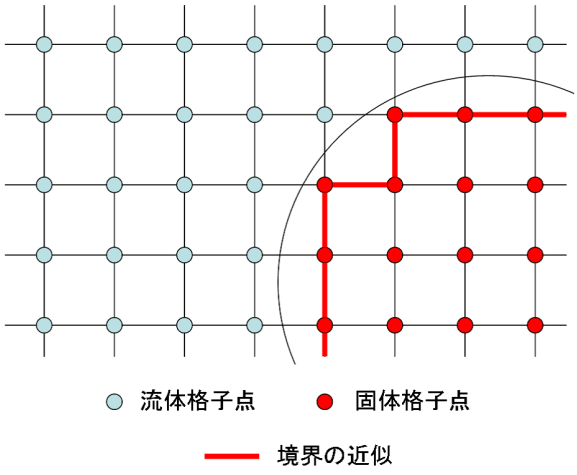

図 4: 固体壁形状の近似

流入，流出境界

3.1.1 のモデルに従って空間を離散化し，$c\Delta t = 1$ とすれば，格子点の座標は整数 m，n で表すことができる．左下を $(0, 0)$ とし，座標 (m, n) の格子点を

$R(m,n)$ で表すことにする．$m_{min}(=0) \leq m \leq m_{max}$，$n_{min} \leq n \leq n_{max}$ であるとき，$R(m_{min},n)$ を流入境界とし，$R(m_{max},n)$ を流出境界とする．

本実装では一様流入を想定している．すなわち，全格子点において初期値として与える流速ベクトル u_0，密度 ρ_0 を用いて導出した局所平衡分布関数 f_i^{eq} を用いて，$R(m_{min},n)$ での分布関数を $f_i(m_{min},n) = f_i^{eq}$ に固定する．

流出境界は自由流出を想定している．$R(m_{max},n)$ での分布関数を $f_i(m_{max},n) = f_i(m_{max}-1,n)$ とすることで流速の勾配を 0 にし，自由流出を表現する．

$R(m,n_{min})$，$R(m,n_{max})$ 間は周期境界とみなす．$f_i(m,n_{min})$，$f_i(m,n_{max})$ をそれぞれ適宜 $f_i(m,n_{max}+1)$，$f_i(m,n_{min}-1)$ とみなすことで，並進を行う．

3.1.3 誤差の低減

格子ボルツマン法は，粒子の存在を分布関数として実数で表し，時刻 t での計算結果は，時刻 $t-\Delta t$ での計算結果に依存する手法であるため，計算機上に実装する場合，時間発展とともに誤差が蓄積する問題が生じる．格子ボルツマン法に限らず，一般に精度が求められる数値計算では倍精度が使われるが，GPU では単精度までしか扱えない．しかも，GPU での浮動小数点演算は，必ずしも IEEE 標準に合致しないという報告もある [14].

式 (9) によって局所平衡分布関数 f^{eq} を求める際，4 個の項を足し合わせる作業が必要である．第 1 項の 1 に，それぞれ u/c，$(u/c)^2$，$(u/c)^2$ に比例する項を足し合わせている．Mach 数が小さい場合，たとえば $u/c \simeq 10^{-3}$ であれば，5, 6 桁の情報落ちが生じることになり，有効桁数が高々8 桁程度である単精度での計算においては，深刻な問題である．

Skordos は，算術的な式変形によって誤差を低減する手法を提案している [18]. この手法では，密度 ρ はほとんど一定値，すなわち初期値 ρ_0 に近いことに着目し，式 (2)(9) の代わりとして次の式 (13)(14) を用いる．

$$\rho = \sum_i f_i + \rho_0 \tag{13}$$

$$f_i^{eq} = w_i \left[(\rho - \rho_0) + \rho \left\{ \frac{3(e_i \cdot u)}{c^2} + \frac{9(e_i \cdot u)^2}{2c^4} - \frac{3u^2}{2c^2} \right\} \right] \tag{14}$$

本研究においてもこの手法を用いることとする．

3.2 計算条件

2次元円柱周りの流れの計算を行う．格子数は256×128とし，円柱の中心座標$(63, 64)$，直径を16とする．初期条件として，全格子点に流速$u = u_0 = (0.01c, 0)$，密度$\rho = \rho_0 = 1$を与える．100000タイムステップの計算を行い，1タイムステップ毎に流速uの大きさを色として表示する．

3.3 実装

計算は，GPU上で行う．まず，計算で用いるデータの初期値をCPUで作成し，32bit浮動小数点テクスチャとして，GPUに転送する．テクスチャの各ピクセルと各格子点を対応させる．すなわち，使用するテクスチャのサイズは256×128とし，格子座標とテクスチャ座標を一致させる．各格子点が保持する必要があるデータは，9個の分布関数f_i $(i = 0 \sim 8)$，流速のx成分u_x，y成分u_y，密度ρの計12個である．前述のように，テクスチャには1ピクセルにつきRGBAの4個のデータを保持することができるため，テクスチャは3枚必要になる．本実装では，固体境界位置情報を保持するためのテクスチャを1枚加え，計4枚のテクスチャを用いた．これらをtex1 ∼ tex4と呼ぶことにする．f_iの初期値は，$u = u_0$，$\rho = \rho_0$によって導出された局所平衡分布関数f_i^{eq}を用いて，$f_i = f_i^{eq}$として与える．f_i $(i = 1 \sim 4)$はtex1，f_i $(i = 5 \sim 8)$はtex2，f_0はtex3のRGBA 4チャネルのうちの1つ，たとえばBに格納する．u_x，u_y，ρ_0はtex3のR，G，Aに格納する．tex4ではたとえば流体格子点を0，固体格子点を1とし，RGBAの内の1チャネルに格納する．tex4の残りの3チャネルは使用しない．

GPUでは，テクスチャマッピングの機能を利用し，計算を行う．すなわち，描画の過程でフラグメントプロセッサへの入力としてテクスチャを利用する．フラグメントプロセッサのパイプライン処理をCgによるフラグメントプログラムで制御することで計算を行う．格子ボルツマン法の1タイムステップにつき，6回の描画を行う．これらの描画はそれぞれ別のフラグメントプログラムによって制御する．Cgによるこれら6つのフラグメントプログラムを，描画順に従ってfp1 ∼ fp6と呼ぶことにする．fp1 ∼ fp5の描画対象はピクセルバッファとし，fp6の描画対象はフレームバッファとする．フラグメントプログラムによる計算処理を具体的に説明する．

fp1

fp1 では，流体格子点での f_i $(i = 1 \sim 4)$ の衝突を行う．前タイムステップの $f_i(i = 1 \sim 4)$ を得るため，tex1 を参照する．衝突に先立って，式 (9) によって局所平衡分布関数 $f_i^{eq}(i = 1 \sim 4)$ を求める必要がある．この際，tex3 の u，ρ の情報が必要となる．衝突過程は式 (4) の右辺に相当する．この右辺の計算結果を f_i' $(i = 1 \sim 4)$ とし，出力とする．固体格子点では，並進過程に先立って，壁面での bounce-back 条件の適用をここで行っておく．すなわち壁面では $f_1' = f_3$，$f_2' = f_4$，$f_3' = f_1$，$f_4' = f_2$ とする．便宜上，固体格子点に相当するピクセル全てでこの操作を行うが，実際に利用されるのは新しく流体格子点へ向かう f_i' のみである．流体格子点と固体格子点では別の計算を行うため，tex4 を参照する．

フラグメントプログラム fp1 を図 5 に示す．フラグメントプログラムは全てのフラグメントに適用される．fp1 では，フラグメント毎に異なるテクスチャ座標と，全てのフラグメントで共通の (uniform 指定子で判別できる)tex1，tex3，tex4，TAU(τ)，RHO(ρ_0) を入力としている．計算結果はピクセルのカラー値として出力される．変数は RGBA に対応して最大 4 次元ベクトルとして扱われる．$i = 1 \sim 4$ に対応する計算が 4 次元ベクトル演算として 1 つの計算式で表現されているが，この表現が示すとおり，GPU では各ベクトル成分 x，y，z，w は同時に計算される．また，各ベクトル成分は，演算子 (.) を用いてコストをかけずに再配置できる．

描画後のピクセルバッファの内容 (f_i') は tex1 に上書きする．式 (4) からわかるように，格子 BGK モデルでは，衝突計算において他の速度方向の分布関数の干渉がないため，他の速度方向の分布関数の衝突計算のために tex1 の内容 (f_i) を残しておく必要がない．テクスチャへの上書きは流入境界を考慮し，$m = 0$ の流入境界を残し，$1 \leq m \leq 255$ の領域を tex1 の同一の領域に上書きする．

fp2

fp2 では，f_i $(i = 5 \sim 8)$ について流体格子点での衝突と，固体格子点での境界条件の適用を行う．使用するテクスチャは tex2，tex3，tex4 であり，e_i，w_i が異なるほかは fp1 とほとんど同じである．描画後の計算結果の f_i' $(i = 5 \sim 8)$ が出力されたピクセルバッファの内容を，fp1 の場合と同様に流入境界以外の領域について tex2 に上書きする．なお，本章で用いた

fp2∼fp6 については，付録にて添付する．

fp3

fp3 では，f_i' $(i = 1 \sim 4)$ の並進を行う．これは，式 (4) の左辺に相当する．e_i $(i = 1 \sim 4)$ に従って tex1 の隣接するピクセルから独立に f_i' $(i = 1 \sim 4)$ に相当する RGBA の値を読み取り，出力とする．境界条件の適用は fp1 で済んでいるため，全てのピクセルで同様の並進を行えばよい．描画後，ピクセルバッファの内容を tex1 に上書きするが，流入境界と流出境界を考慮し，2 度上書きする．1 度目の上書きは流入条件を考慮し，これまで同様 $1 \leq m \leq 255$ の領域を tex1 の同一の領域に上書きする．2 度目の上書きは流出条件を考慮し，$m = 254$ の領域を tex1 の $m = 255$ の領域に上書きする．

fp4

fp4 では，f_i' $(i = 5 \sim 8)$ の並進を行う．e_i $(i = 5 \sim 8)$ に従って tex2 の隣接するピクセルから独立に f_i' $(i = 5 \sim 8)$ に相当する RGBA の値を読み取り，出力とする．fp3 の場合と同様に，流入境界と流出境界を考慮し，tex2 に 2 度上書きする．

fp5

fp5 では，残りの分布関数 f_0 の衝突と，式 (2) に従った巨視化の計算を行う．f_0 の衝突は，tex3 の u_x，u_y，f_0，ρ を用いて計算される．流速ベクトル成分 u_x，u_y と密度 ρ は，今衝突によって更新された f_0 と，tex1，tex2 の f_i $(i = 1 \sim 8)$ を用いて計算される．描画後，上書きによって tex3 を更新する．

fp6

fp6 では，tex3 に保持されている u_x，u_y から流速の大きさ $||u||$ を求め，これを色相に対応させ，RGB に変換し，フレームバッファへ描画する．

3.4 評価

3.4.1 実装環境

CPU は Pentium4 3.2GHz，OS は WindowsXP，GPU は GeForce6800Ultra を用いた．プログラム言語は C，OpenGL，Cg を用いた．

3.4.2 計算結果

100000 タイムステップ後の計算結果を図 6 に示す．一般に知られるとおり，動粘性係数が $3.2c \times 10^{-3}$(代表長さを円柱の直径，代表速さを流入速度とすれば，レイノルズ数 $Re = 50$ に相当する) では上下対称の流れとなり (図 6A)，動粘性係数が $0.8c \times 10^{-3}(Re = 200)$，$0.2c \times 10^{-3}(Re = 800)$ では円柱の上下からカルマン渦が観察された (図 6B，C)．動粘性係数を $0.2c \times 19^{-3}$ とし，同様の計算を CPU でも行い，比較を行った．CPU での計算結果を図 6D に示す．GPU と同じく，単精度で計算を行ったが，これらの計算結果は完全には一致しなかった．

本実装では，固体の形状は tex4 によって定義されている．すなわち，tex4 を変更するだけで，固体の形状を容易に変更することができる．これは，近傍の格子点との相互作用によってのみ進められる格子ボルツマン法の計算の特徴であり，インタラクティブシミュレーションの実現の可能性を示唆するものであると考えられる．2×2 の角柱を千鳥格子上に配置した場合の計算結果を図 6E に示す，

格子数を 512×256 にした場合とあわせて，10000 タイムステップの実行時間を表 3.4.2 に示す．CPU で実行した場合に比べ，GPU を用いることにより，10 倍以上高速に実行することができた．

表 3: 実行時間

格子数	実行時間 (sec/10000timestep)	
	CPU 実行時間	GPU 実行時間
256×128	203	17
512×256	803	69

3.5 本章のまとめ

流体計算の新しい手法である格子ボルツマン法による流体シミュレーションを GPU 上に実装する手法を紹介し，GPU を用いることによって CPU のみでの計算に比べ高速に実行できることを確認した．ただし，計算結果が両者で必ずしも一致しないことも確認した．次章以降では，GPU による計算を前提とし，インタラクティブな流体シミュレーションの実現を目指す．

```
//fp1
struct fpIN{
   float4 texCoord :TEX0;
};
struct fpOUT{
   float4 col :COLOR;
};

fpOUT main(   fpIN IN,
              uniform samplerRECT tex1,
              uniform samplerRECT tex3,
              uniform samplerRECT tex4,
              uniform float TAU,
              uniform float RHO)
{
   fpOUT OUT;
   float4 u=f4texRECT(tex3,IN.texCoord.xy);
   float kabe=f4texRECT(tex4,IN.texCoord.xy).x;
   float4 f=f4texRECT(tex1,IN.texCoord.xy);
   float4 fe,fee;
   int4 e;
   float u2;

   if(kabe<0.5f){
      u2=u.x*u.x+u.y*u.y;
      e=int4(1,0,-1,0);
      fee=e*u.x+(e.yxwz)*u.y;
      fe=(u.w-RHO+u.w*(3.0f*fee+4.5f*fee*fee-1.5f*u2))/9.0f;
      OUT.col=f-(f-fe)/TAU;
   }
   else OUT.col.xyzw=f.zwxy;
   return OUT;
}
```

図 5: フラグメントプログラム fp1

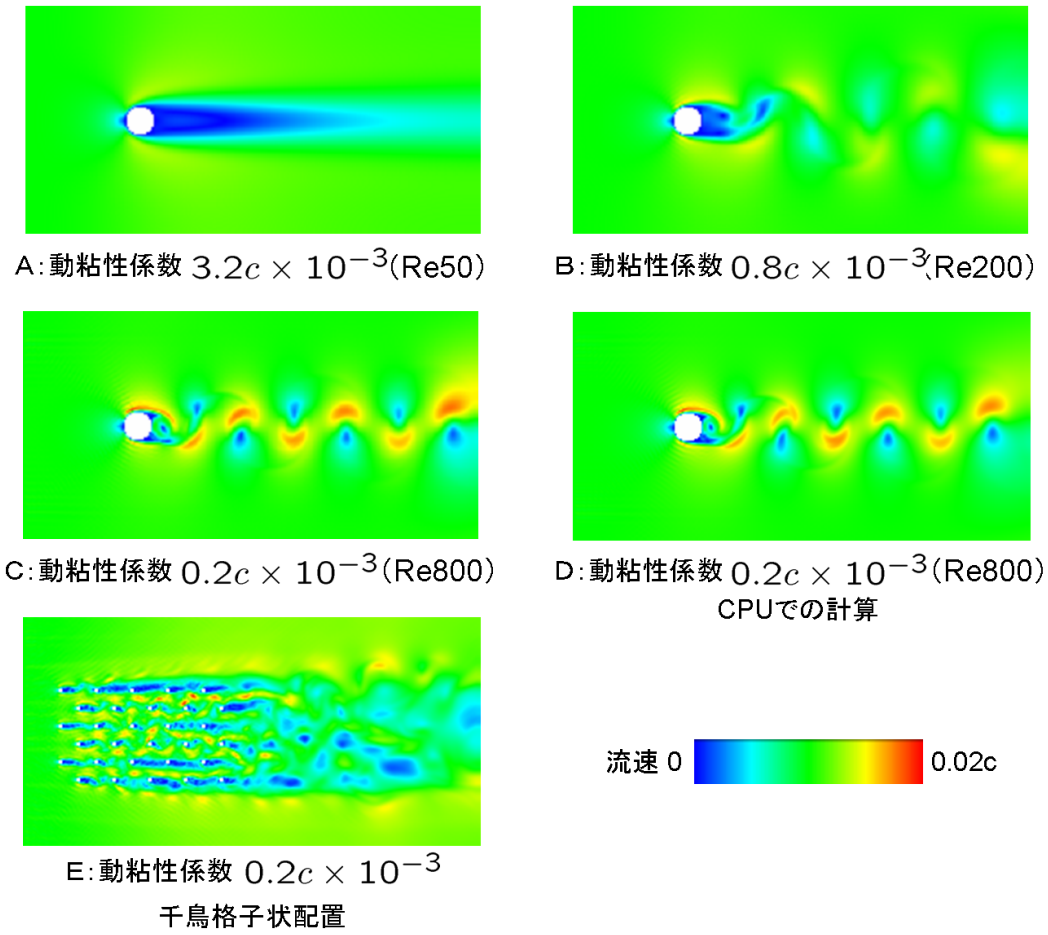

図 6: 計算結果 (100000 timestep)

第4章　インタラクティブシミュレーション

第3章において，GPUによる計算によって高速な流体シミュレーションが実現できたことにより，インタラクティブシミュレーションの実現が期待できる．本章では，固体境界位置を変更する手法を紹介し，触覚デバイスを用いて固体境界位置を動的に変更できるシミュレーションの実装報告を行う．

4.1　計算手法

インタラクティブシミュレーションの実現のために，境界条件に関連した新しい計算手法を導入する．

4.1.1　固体境界の移動

第3章では，固体境界の形状を，格子に沿ったジグザグ状に近似し，単純なbounce-back条件を適用した．このような固体境界形状の近似であっても，直線状の境界面の移動が境界面に沿った平行移動であれば対応できる[19]．しかし，固体格子点から流体格子点，あるいは流体格子点から固体格子点への変更が起こる場合には，仮に境界面の移動速度が小さい場合でも，格子点の変更の瞬間，物理変数の急変が起こってしまう．

Bouzidiらは，流体-固体境界面が連続的な任意の位置をとることを認め，bounce-back条件を修正することで，非合理な物理変数の急変を避けつつ境界面の移動を可能にする手法を提案した[20]．図7において，境界面の位置をDとし，Aを固体格子点，B，Cを流体格子点としたとき，$q = |\mathrm{BD}|/|\mathrm{BA}|$とする．

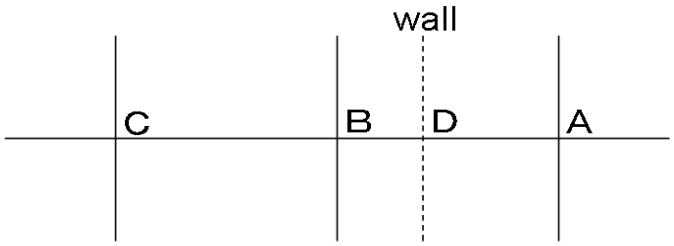

図7: 格子点と固体境界の位置

iが流体格子点から固体格子点へ向かう向き，i'が固体格子点から流体格子点へ向かう向きを表し，iの向きで，点r，時刻tでの分布関数を$f_i(r,t)$と表す場合，

未知の分布関数は次式によって求める．

$$f_{i'}(B, t+\Delta t) = 2q f_i(B,t) + (1-2q) f_i(C,t) \qquad \left(q < \frac{1}{2}\right) \tag{15}$$

$$f_{i'}(B, t+\Delta t) = \frac{1}{2q} f_i(B,t) + \frac{(2q-1)}{2q} f_{i'}(B,t) \qquad \left(q \geq \frac{1}{2}\right) \tag{16}$$

境界が移動する場合，式 (15)(16) に次の項を付加する．

$$\delta f_{i'}^{(1)} = 2\alpha_i e_i v \qquad \left(q < \frac{1}{2}\right) \tag{17}$$

$$\delta f_{i'}^{(1)} = \frac{1}{q}\alpha_i e_i v \qquad \left(q \geq \frac{1}{2}\right) \tag{18}$$

ただし，v は境界の移動速度ベクトルであり，D2Q9 モデルにおいては，

$$\alpha_i = \left\{0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}\right\} \tag{19}$$

である．

境界の移動によって固体格子点が流体格子点に変更された場合，新たな流体格子点では分布関数が定義されていない．本研究では，v と密度の初期値 ρ_0 によって局所平衡分布関数を求め，これを新たな流体格子点での分布関数とする [21].

固定境界においては，通常の bounce-back 条件を適用するものとするが，これは式 (16) における $q = 1/2$ の場合に相当する．

4.1.2 固体にかかる力の計算

格子ボルツマン法において流体中の固体にかかる力を求めるため，圧力積分法と運動量交換法の 2 通りのアプローチが研究されている [22][23]．圧力積分法は，Navier-Stokes 方程式に基づいた解法においても広く使われてきた手法である．一方，運動量交換法は，格子ボルツマン法固有の手法であるが，信頼性が高く，実装が容易であるという長所を持つ．本研究においては運動量交換法を採用した．

壁面へ向かう分布関数 $f_R(t)$ で表現される粒子が壁面で跳ね返ることにより $f_L(t+\Delta t)$ で表現されるとすれば，運動量変化 ϕ は，

$$\phi = f_R(t) + f_L(t+\Delta t) \tag{20}$$

となり，これが固体へと渡される．固体にかかる力は，運動量変化全てについて $\phi_i e_i$ の総和をとることで求めることができる．

4.2 触覚デバイスの利用

オペレータによるシミュレーションへのインタラクティブな入出力を実現するため，SensAble Technologies 社の PHANTOM Omni[24](図 8) を使用した．オペレータは，このデバイスのスタイラス部を移動させることによって，3 次元位置情報を入力できる．また，デバイスは，スタイラス部を通じてオペレータに力覚を返すことができる．PHANTOM Omni の諸元を表 4 に示す．

図 8: PHANTOM Omni

表 4: PHANTOM Omni 諸元

作業空間	160 × 120 × 70 mm
位置分解能	450 dpi (0.055 mm)
最大摩擦力	0.26 N
最大提示反力	3.3 N
硬直性	X 軸：1.26 N/mm Y 軸：2.31 N/mm Z 軸：1.02 N/mm
慣性	45 g 以下
インターフェース	IEEE-1394 FireWire port

4.3 実装

周囲を固体境界とし，初期状態では流体は静止しているものとする．この流体中で PHANTOM によって直径 16 の円柱を移動させた場合の流体の流れの計算を行う．流速の大きさを色として表示するとともに，円柱にかかる力を PHANTOM によってオペレータに返すことを目的とする．なお，本章と第 5 章では，GPU は GeForce6800 を用いた．

4.3.1 円柱の移動

実空間における r_0 mm と格子数 N のシミュレーション空間における格子単位の座標 r との対応は，

$$r = N/200 \times r_0 \tag{21}$$

とした．円柱の中心座標 O は，PHANTOM によって，1 タイムステップ毎に与えられる (2 次元であるため，z 座標は無視する)．時刻 t における円柱の中心座

標を O_t とすれば，時刻 t における円柱の移動速度ベクトル v_t は，

$$v_t = (O_t - O_{t-\Delta t})c \tag{22}$$

となるが，流体の流れは円柱の移動に追随するため，オペレータによる自由な移動を認めた場合，局所平衡分布関数を導出する際の巨視的な流速の大きさが並進速度に比べて十分小さいとする前提が必ずしも守られない．本実装では，v_t の大きさを $0.05c$ に制限する．すなわち，$||v_t|| > 0.05c$ となる場合，$v'_t = v_t$ とおき，次式により v_t と O_t を再定義する．

$$v_t = \frac{0.05cv'_t}{||v'_t||} \tag{23}$$

$$O_t = v_t/c + O_{t-\Delta t} \tag{24}$$

$c = 1/\Delta t$ であり，オペレータの素早い操作に対応するためには，時間刻み Δt を小さくする必要がある．実時間シミュレーションとする前提であれば，Δt を実時間に対応する値であるとみなさねばならず，オペレータの素早い操作に対応するためには計算を高速に実行しなければならないことを意味する．

4.3.2 GPU での計算の流れ

計算は主に GPU で行う．まず 1) 円柱面の位置を捉えるための計算を行い，続いて 2) 衝突 3) 並進 4) 巨視化の計算を行い，5) 表示する．描画回数は，それぞれ 1)2 回，2)2 回，3)2 回，4)1 回，5)1 回であり，計 8 回の描画は，全て異なるフラグメントプログラムによる．1) から 4) については，ピクセルバッファへの描画後，その内容をテクスチャへ上書きすることによってテクスチャを更新する．それぞれの過程について説明する．

1) 円柱面の位置

円柱に対して 4.1.1 節で紹介した境界条件を適用する場合，q の値が必要になる．この境界条件は，固体格子点に隣接する流体格子点において，1 個，あるいは複数個 (最大 4 個) の分布関数に適用する必要があるため，$i = 0$ を除いた各分布関数 f_i に対応した q_i を保持するためのテクスチャを用意する．すなわち，$f_1 \sim f_4$ を保持する tex1，$f_5 \sim f_8$ を保持する tex2，u，ρ を保持する tex3 に加え，$q_1 \sim q_4$ を保持する tex4 と，$q_5 \sim q_8$ を保持する tex5 を用意する．tex4 と tex5 には，q 以外の情報も埋め込む．具体的には，格子ボルツマン法の最初の過程である衝突演算に先立って，以下の計算を行い，tex4，tex5 を更新する．

流体・固体の区別

互いに異なる 3 つの定数 α，β，γ を用意し，円柱の中心座標と半径から各々のピクセルに該当する格子点が固体であるか流体であるかを判断し，固体であれば α，流体であり，直前の状態が固体であれば β，流体であり，直前の状態も流体であれば γ とする．

q の算出

固体境界に隣接する流体格子点 (図 9) において，固体から流体へ向かう向きの分布関数に境界条件を適用する必要があり，この際 q の値が必要となる．境界条件を適用すべき分布関数が位置する格子点において固体表面と格子との交点座標を求め，q を算出する．f_i が必要とする q を q_i とし，前述の β あるいは γ に q_i を加えた値を出力とする．1 格子点につき，複数の q_i が必要となる場合もありうる．

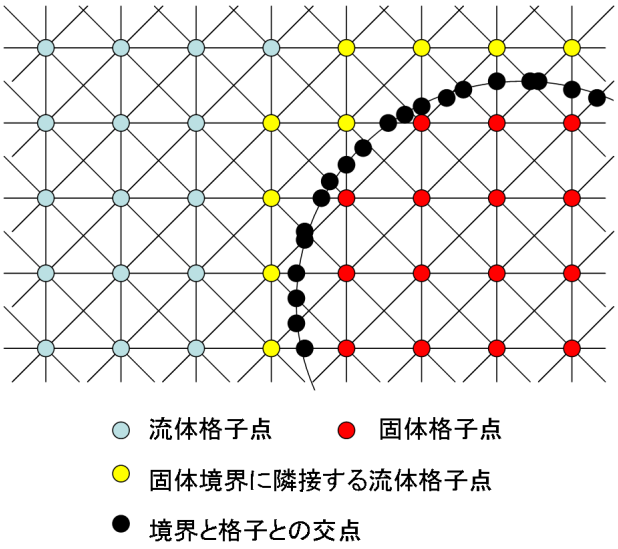

図 9: 格子中の固体表面

$0 < q \leq 1$ であるので，例えば定数 α，β，γ を互いの差が 2 以上の整数とすれば，q の情報との干渉を避けることができる．α，β，γ は，主に新しく生成された流体格子点を検出する目的で利用する．q は並進過程における境界条件の適用の際，さらには CPU での力の計算の際に利用する．

2) 衝突

tex4，tex5 を参照し，該当ピクセルが $\gamma(+q)$ ならば衝突計算を行う．該当ピ

クセルが $\beta(+q)$ であれば新たに生じた流体格子点であり，分布関数，流速，密度が定義されていないことを示す．この場合，円柱の移動速度ベクトル v と密度の初期値 ρ_0 により局所平衡分布関数を求め，これを分布関数として出力する．上書きによって tex1，tex2 を更新する．

3) 並進

この過程では，並進と境界条件の適用を行う．tex1，tex2 の分布関数 f_i について，tex4，tex5 において q_i が定義されていれば境界条件を適用し，q_i が定義されていなければ通常通り e_i に基づいた並進を行えばよい．

4) 巨視化，5) 表示

第 3 章での実装と同様である．

4.3.3 力の計算

力の計算は，CPU で行う．円柱表面での粒子の跳ね返りによる運動量変化 ϕ を求めるためには，境界条件適用前と適用後の分布関数が必要となるが，本実装においては跳ね返る前の粒子を 2 個の分布関数で表現している上，どの分布関数が跳ね返る粒子を表現しているかの判断が難しい．ところが，跳ね返りの前後での分布関数の差は式 (17)(18) であるので，跳ね返りの後を示す分布関数と q_i を GPU から読み出せば，ϕ は計算可能となる．しかも，q_i の定義の有無によって総和をとるべき $\phi_i e_i$ が容易に判別できる．したがって，4.3.2 での 1), 3) の描画後にピクセルバッファの内容を読み取る．GPU から CPU へのデータの転送は低速であるため，データの読み取りは円柱の存在部分周辺の最小限の領域にとどめなければならない．算出された力は，PHANTOM によって提示する．

4.4 力覚の提示に伴う問題

シミュレーションを実行した結果，PHANTOM のスタイラス部が激しく振動し，シミュレーションが破綻する問題が発生した．この振動は，円柱の移動が遅い場合に特に顕著であった．円柱を移動させると，原則的に円柱の移動方向とは逆向きの力が円柱に対して加わることが予想できる．この力を PHANTOM によって提示することにより，逆向きの入力を招く．これが繰り返されることによって，振動が増幅されるものと考えられる．また，この問題は，PHANTOM の位置分解能の限界にも起因していると考えられる．力覚の提示をオフにした状態でシミュレーションを実行し，スタイラスを右向きにゆっくりと連続的に

動かした場合の PHANTOM で認識された x 座標の変化の例を図 10 に示す．この例では，1 タイムステップでの速度約 0.05(格子数 128 のシミュレーションの場合，速度 $0.032c$ に相当する) の移動と，数タイムステップの静止が交互に見られる．このような速度の不連続性が，振動の原因となっているものと考えられる．

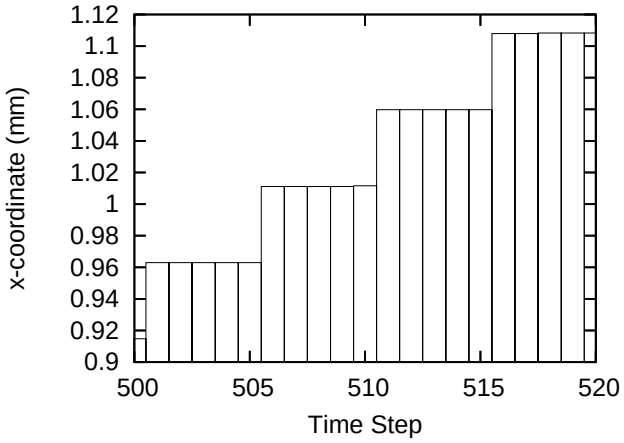

図 10: PHANTOM から得られた座標情報

振動の問題に対して，以下のような解決法が考えられる．

複数回の速度の平均をとる

100 回程度の速度の平均を与えてみたが，問題は改善されなかった．

加速度に制限をかける

前タイムステップでの速度との減算により加速度を求め，加速度の大きさを最大 $0.0005c^2$ 程度に制限したところ，若干の振動は残るものの，かなり改善することができた．

力の平均をとる

シミュレーションの破綻は，微小な振動が出力と入力の連鎖によって増幅されることによって起こる．出力の振動を抑えることによっても問題を改善することができる．100 回程度の力の平均をとることで，かなり改善することができた．

力の変動を制限する

前タイムステップで提示した力との差を制限することにより，かなり改善することができた．この手法を適用したシミュレーションの様子を図 11 に

示す．格子数は 128×128，動粘性係数は $3.2c \times 10^{-3}$ としている．

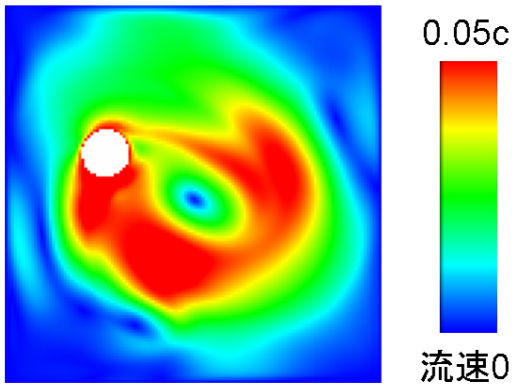

図 11: シミュレーションの様子

4.5 本章のまとめ

格子ボルツマン法での境界を移動させるための計算手法を紹介し，これをGPUに実装する手法について説明し，実装報告を行った．また，触覚デバイスを用いることにより，流体シミュレーションに操作を加えることができ，シミュレーション結果を視覚，力覚で提示することによって，インタラクティブな流体シミュレーションを実現した．次章では，より実用的なシミュレーションを目指し，3 次元空間を対象とする．

第5章　3次元シミュレーション

第4章では，2次元平面をテクスチャに対応させ，2次元のインタラクティブな流体シミュレーションを実装した．しかし，インタラクティブシミュレーションシステムによる仮想現実感を得る目的を実現するためには，3次元空間を扱うことが必要である．本章では，インタラクティブな流体シミュレーションを3次元に適用する手法を説明し，実装報告を行う．

5.1　計算モデル

3次元の格子BGKモデルとして多く用いられているD3Q15モデルを用いる．このモデルにおいては，立方格子で空間を分割し，粒子は静止を含む15種類の速度を持つとする．粒子速度 e_i $(i = 0, 1, \cdots, 14)$ は，以下のように定義される (図12).

$$e_i = \begin{cases} (0,0,0) & i = 0 \\ c\ (\cos[(i-1)\pi/2], \sin[(i-1)\pi/2], 0) & i = 1 \sim 4 \\ (0,0,c), (0,0,-c) & i = 5,6 \\ (\sqrt{2}c\ \cos[(2i-13)\pi/4], \sqrt{2}c\ \sin[(2i-13)\pi/4], c) & i = 7 \sim 10 \\ (\sqrt{2}c\ \cos[(2i-21)\pi/4], \sqrt{2}c\ \sin[(2i-21)\pi/4], -c) & i = 11 \sim 14 \end{cases} \tag{25}$$

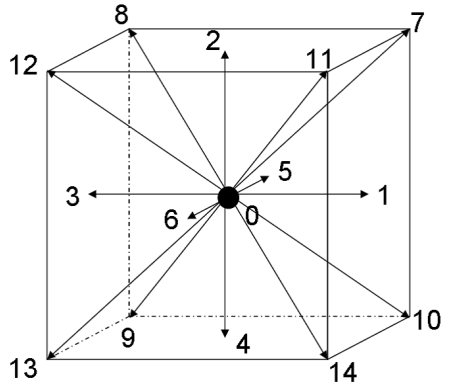

図12: D3Q15モデルにおける仮想粒子の速度ベクトル e_i

重み関数 w_i は，以下のように決められる．

$$w_i = \begin{cases} 2/9 & i = 0 \\ 1/9 & i = 1 \sim 6 \\ 1/72 & i = 7 \sim 14 \end{cases} \tag{26}$$

5.2 実装

本章では，第 4 章と同様のシミュレーションを 3 次元に拡張したシミュレーションの実装を目指す．すなわち，x 方向，y 方向の境界は固体壁とし，z 方向の境界は周期境界とした流体領域の中の z 軸に平行な円柱を PHANTOM によって移動させる．本章以降では力覚の提示は行わないが，第 4 章と同様の手法で容易に実現できるものと思われる．

5.2.1 データの割り当て

D3Q15 モデルにおいては 3 次元空間は立方格子に離散化されるが，この立方格子を 2 次元格子の重なりと捉える．この 2 次元格子をスライスと呼ぶ．z 軸に垂直に z 軸の 1 格子毎に空間を切り分けることで，z 軸方向の格子数分のスライスが並ぶ．これまでと同様に $c\Delta t = 1$ とし，格子点の座標を (l, m, n) とすれば，各スライスは n で区別できる．

計算に用いるテクスチャは，各スライスに対応させる．ただし，テクスチャには上下左右に 1 ピクセル分のマージンをとり，このマージンは計算領域を表さないものとする (理由は後述する)．すなわちサイズ $L \times M$ のテクスチャを用いれば，計算領域の格子サイズは $(L-2) \times (M-2)$ となる．便宜上 z 軸に関してもマージンをとり，計算領域の範囲は $1 \leq l \leq L-2$，$1 \leq m \leq M-2$，$1 \leq n \leq N-2$ とする．各格子点で保持する必要があるデータは，分布関数 f_i $(i = 0 \sim 14)$，流速ベクトルの x 成分 u_x，y 成分 u_y，z 成分 u_z，密度 ρ，第 4 章と同様に $i = 0$ を除いた各分布関数に対応した q_i $(i = 1 \sim 14)$ 等の境界情報の 33 個である．そのため，各スライスで必要なテクスチャは 9 枚となる．この 9 枚のテクスチャを記号 k $(k = 1 \sim 9)$ を用いて区別すれば，スライスの z 座標 n と k を用いて，$\mathrm{tex}(n, k)$ と表せる．使用する変数は，以下のように各テクスチャ，各ピクセルの RGBA のチャネルに割り当てる．ただし $(-)$ のチャネルは使用しないことを表す．

tex$(n, 1)$　f_1，f_2，f_3，f_4

tex$(n, 2)$　f_7，f_8，f_9，f_{10}

tex$(n,3)$　f_{11}，f_{12}，f_{13}，f_{14}

tex$(n,4)$　f_5，f_6，f_0，$-$

tex$(n,5)$　u_x，u_y，u_z，ρ

tex$(n,6)$　q_1，q_2，q_3，q_4

tex$(n,7)$　q_7，q_8，q_9，q_{10}

tex$(n,8)$　q_{11}，q_{12}，q_{13}，q_{14}

tex$(n,9)$　q_5，q_6，$-$，$-$

2 次元の場合，並進の操作は常に同一テクスチャ内で行った．3 次元の場合，スライス間の並進が必要となるため，異なるテクスチャへの並進操作が必要となる．D3Q15V モデルの場合，f_0 は並進無し，$f_1 \sim f_4$ は同一テクスチャ内での並進，f_5，$f_7 \sim f_{10}$ は $n-1$ から n への並進，f_6，$f_{11} \sim f_{14}$ は $n+1$ から n への並進が必要である．これらの並進が効率的に行えるように割り当てを行った．

5.2.2　計算の流れ

1 タイムステップあたりの計算を大まかに分類するとすれば，ほぼ 4.3.2 と同様に分類できるが，異なるテクスチャ間での並進が必要な場合は並進と境界条件の適用の処理を分け，別の描画によって実現する．すなわち，まず 1) 円柱面の位置を捉えるための計算を行い，続いて 2) 衝突 3) 並進 4) 境界条件の適用 5) 巨視化の計算を行い，6) 表示する．それぞれの過程について説明する．

1) 円柱面の位置

計算領域の $1 \leq l \leq L-2$，$1 \leq m \leq M-2$ の範囲で，4.3.2 と同様に，流体・固体の区別と q の計算を行い，上書きによってテクスチャを更新する．同様の処理を $\mathrm{tex}(n,k)$ について $1 \leq n \leq N-2$，$6 \leq k \leq 9$ の範囲で繰り返す．k によってそれぞれ別の 4 つのフラグメントプログラムを用いた．

2) 衝突

4.3.2 と同様だが，描画から上書きまでの処理を $\mathrm{tex}(n,k)$ について $1 \leq n \leq N-2$，$1 \leq k \leq 4$ の範囲で繰り返す．k によってそれぞれ別の 4 つのフラグメントプログラムを用いた．

3) 並進

並進が同一テクスチャ内で済む $f_1 \sim f_4$ については，4.3.2 と同様に並進と境界条件の適用を同一の描画で行う．描画から上書きまでの処理を $\mathrm{tex}(n,1)$ について $1 \leq n \leq N-2$ の範囲で繰り返す．

$f_7 \sim f_{10}$ の並進では，$\mathrm{tex}(n-1,2)$ から $\mathrm{tex}(n,2)$ への並進となるが，$\mathrm{tex}(n,2)$

への上書きにより，$\mathrm{tex}(n+1,2)$ のための並進元の情報が失われる．全てのスライスで無事に並進を完了するため，次の方法を用いる．まず $\mathrm{tex}(0,2)$ を用意し，これに $\mathrm{tex}(N-2,2)$ を単純にコピーする．続いて $\mathrm{tex}(n-1,2)$ から $\mathrm{tex}(n,2)$ への並進を $n=N-2,N-1,\cdots,1$ の順で行う．$f_7 \sim f_{10}$ は z 方向の並進速度成分が全て同一であるため，テクスチャの更新をこの順で行うことで全てのスライスで無事に並進を完了できる．ただし，逆向きの速度 ($f_{11} \sim f_{14}$) を扱っていないことにより，固体境界条件の適用は行えない．そのため，境界 (円柱，外壁) に構わず，全ての格子点で同様の並進を行わなければならない．後で境界条件の適用の際に使用する外壁に向かう分布関数の情報が並進によって消失することを防ぐため，1 ピクセル分のマージンをとっている．

$f_{11} \sim f_{14}$ の並進では，$\mathrm{tex}(N-1,3)$ を用意し，これに $\mathrm{tex}(1,3)$ をコピーする．$\mathrm{tex}(n+1,3)$ から $\mathrm{tex}(n,3)$ への並進を $n=1,2,\cdots,N-2$ の順で行う．$f_7 \sim f_{10}$ の並進と $f_{11} \sim f_{14}$ の並進の違いはテクスチャの扱いの違いのみであり，同一のフラグメントプログラムで実現できる．

f_5，f_6 は空いている A のチャネルを利用して次のような方法で並進させる．まず $\mathrm{tex}(N-1,4)$ を用意し，$\mathrm{tex}(N-2,4)$ の R，$\mathrm{tex}(1,4)$ の G を $\mathrm{tex}(N-1,4)$ の R，A にコピーする．次に，

- $\mathrm{tex}(n,4)$ の G を $\mathrm{tex}(n,4)$ の A へコピー
- $\mathrm{tex}(n-1,4)$ の R を $\mathrm{tex}(n,4)$ の R へコピー
- $\mathrm{tex}(n+1,4)$ の A を $\mathrm{tex}(n,4)$ の G へコピー

とする $\mathrm{tex}(n,4)$ の更新を $n=N-2,N-3,\cdots,1$ の順で行う．$\mathrm{tex}(1,4)$ の更新の際には，$\mathrm{tex}(N-1,4)$ を $\mathrm{tex}(0,4)$ とみなす．本実装では，以上の方法で f_5，f_6 の並進を行ったが，f_5，f_6 では並進速度に x 成分，y 成分が含まれないことに着目すれば，実際にテクスチャを更新すること無しに仮想的な並進で済ませることも可能であろう．この場合，仮想的に K タイムステップ時にスライス n に存在すると考えられる f_5 は実際には $\mathrm{tex}(n-(K \mod (N-2)),4)$，$f_6$ は実際には $\mathrm{tex}(n+(K \mod (N-2)),4)$ に存在することになる．CPU でテクスチャの割り当てを適切に行えば，テクスチャの更新は必要なくなるものと思われる．$f_7 \sim f_{14}$ に関しては，並進速度に x 成分，y 成分が含まれるため，テクスチャの更新は必須となるであろう．

4) 境界条件の適用

$f_5 \sim f_{14}$ に対して境界条件の適用を行う．円柱については 4.1.1 の境界条件を

使うが，すでに並進が完了しているため，式 (15)(16) の代わりに次の式 (27)(28) を用いなければならない．

$$f_{i'}(B, t+\Delta t) = 2q f_i(A,t) + (1-2q) f_i(B,t) \qquad \left(q < \frac{1}{2}\right) \tag{27}$$

$$f_{i'}(B, t+\Delta t) = \frac{1}{2q} f_i(A,t) + \frac{(2q-1)}{2q} f_{i'}(C,t) \qquad \left(q \geq \frac{1}{2}\right) \tag{28}$$

外壁については，マージンの部分へ並進している分布関数を使って bounce-back 条件を適用する．$\mathrm{tex}(n,k)$ について $1 \leq n \leq N-2$，$2 \leq k \leq 4$ の範囲で繰り返す．$k = 2, 3$ については同一のフラグメントプログラムを使うことができる．

5) 巨視化

$1 \leq n \leq N-2$ の範囲で巨視化計算を行い，$\mathrm{tex}(n,5)$ を更新する．

6) 表示

3 次元での計算結果の表示は，これまでと同様の単なる色による表現では効果的ではない．本研究では，ある 1 枚のスライスのみを 2 次元の場合と同様に表示したが，別の表示方法の検討が必要である．

5.3 問題サイズの限界

3 次元シミュレーションでは 2 次元シミュレーションに比べ，格子点数が増大するほか，本研究で使用したモデルでは粒子速度分布が 9 から 15 に増えていることから，計算量がかなり増大している．また，本章の実装に限っても，実行時間は格子点数にほぼ比例している．問題サイズが大規模になると，計算が低速となり，応用例によっては仮想現実を体感する目的の実現が困難になる．

また，本実装で用いた GPU のグラフィクスメモリは 256MB であるが，GPU で計算に用いるテクスチャデータがグラフィクスメモリに入りきらない場合，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メモリに転送される．これが起こると，テクスチャデータの転送時間がオーバヘッドとなり，急激に速度が低下する．インタラクティブシミュレーションを効果的に実行するためには，問題サイズをグラフィクスメモリに入りきるように一定範囲内に抑えなければならない．本実装では格子点 1 個につき 32bit 浮動小数点変数を空きチャネルも含めて 36 個分使用している．すなわち格子点 1 個につき 144B のデータを使用している．格子数 $L \times M = 64 \times 64$

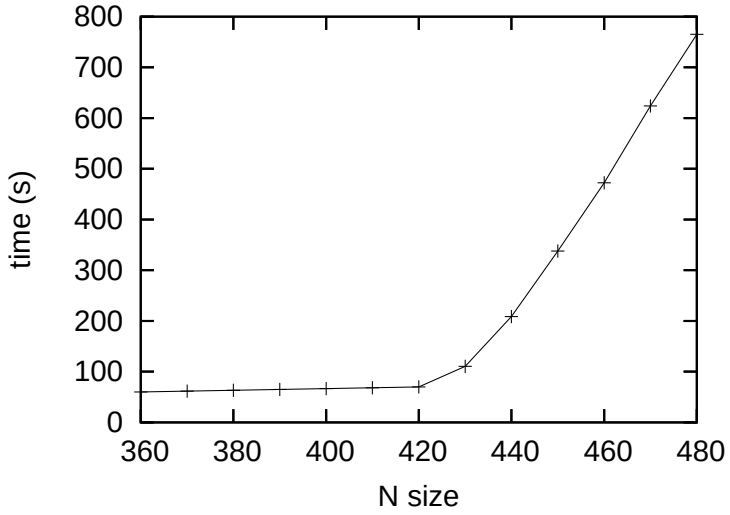

図 13: 格子数 $64 \times 64 \times N$，100 タイムステップの実行時間

とし，N を変化させた場合 (ただし，マージンを含む) の 100 タイムステップの実行時間を図 13 に示す．N が 420 を越えたあたり，すなわち格子点 172 万個程度から急激に速度が低下していることがわかる．

5.4 本章のまとめ

本章では，GPU を用いた格子ボルツマン法による流体シミュレーションを 3 次元に適用する手法を紹介し，実装報告を行った．また，インタラクティブシミュレーションを効果的に実行するための問題サイズの制限について言及した．次章では，さらに大規模なシミュレーションを効果的に実行するために，PC クラスタ上での並列計算について検討する．

第6章　並列化

大規模なインタラクティブシミュレーションの効果的な実行のためには，1台のPCでは限界がある．本章では，PCクラスタを用いた並列計算を適用する手法を紹介し，実装報告を行う．

6.1　処理の分割

処理を8台のノードに分割することを考える．格子ボルツマン法においては，各格子点はほとんどそれぞれ独立に計算することができ，他の格子点との情報のやりとりは，境界条件の適用を含めた並進過程においてのみである．各ノードの負荷をなるべく均等にし，ノード間通信を最小限に抑えるため，x方向，y方向，z方向に関してそれぞれ2分割することによって，計算領域の空間を8等分し，図14に示すように各ノードに番号を割り振る．通信内容は，ほぼ断面に接する格子点の並進情報のみである．本実装では，これまで同様GPUによる計算を前提としているため，通信とは，送信先のノードでGPUが描画した情報をCPUへ転送し，CPU同士でのノード間通信を行い，受信先のノードでCPUからGPUへテクスチャデータとして転送することを意味する．

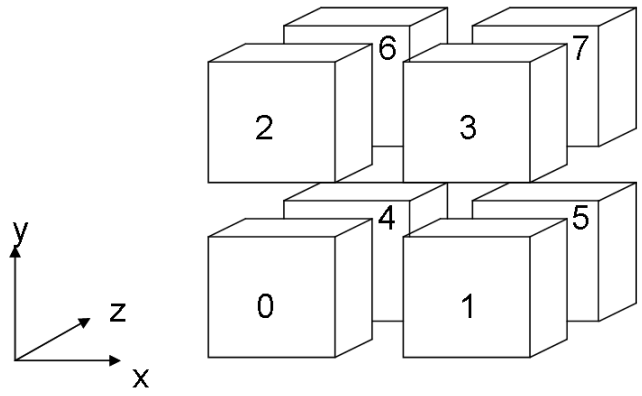

図14: 計算領域の分割と各ノードへの割り当て

6.2　計算条件および実装環境

第5章と同様の計算条件とする．すなわち，x方向，y方向の境界は固体壁とし，z方向の境界は周期境界とした流体領域の中のz軸に平行な円柱をインタ

ラクティブに移動させるシミュレーションを行う．計算条件の z 方向の対称性により，結果的にノード 0，1，2，3 での計算は，それぞれノード 4，5，6，7 での計算と同一となる．なお，本章では PHANTOM を用いず，キーボードによって静止を含めた 9 種類の速度で円柱を移動させる．円柱の移動速度の x 成分，y 成分は 0，$\pm 0.03c$ のいずれか，z 成分は常に 0 とする．

本章での実装環境を表 5 に示す．

表 5: 実装環境

CPU	Pentium4 3.0GHz
Memory	1.0GB
Gfx Card	GeForceFX5950Ultra
Gfx Memory	256MB DDR
OS	Linux 2.4.22
Network	1000BASE Ethernet
Compiler	gcc3.3.5 LAM6.5.9

6.3 実装

1 タイムステップ毎に円柱の中心座標の更新を行うが，円柱の移動のためのキーボードからの入力はノード 0 が管理し，円柱の中心座標，移動速度を決定し，全てのノードへブロードキャストする．以下，各ノードは担当する領域に対して第 5 章と同様の手法で計算を行う．以下，ノード 0 での計算についてのみ説明する．

スライスに割り当てるテクスチャには，これまで同様，上下左右に 1 ピクセル分のマージンをとる．すなわちテクスチャのサイズを $L \times M$(以後 $L \geq M$ と仮定して説明する) とすれば，便宜上 z 軸に関してもマージンをとり，計算領域の範囲は $1 \leq l \leq L-2$，$1 \leq m \leq M-2$，$1 \leq n \leq N-2$ とする．$l = 0$，$m = 0$ のマージン領域は，第 5 章と同様，周囲の固体境界に対して，$l = L-1$，$m = M-1$ のマージン領域は，移動境界 (円柱) に対して境界条件を適用する際に必要となる分布関数を保持する役割を持つ．

並進情報は，分布関数の並進方向に応じて全てのノードと交換する必要があ

る．送信は衝突後に行う必要があり，受信は並進前に完了する必要がある．ノード間の送受信のために，メインメモリに送信用のバッファsend1∼send10(表 6)と受信用のバッファrecv1∼recv6(表 7) を用意する．ただし表中のサイズは 32bit 浮動小数点変数の個数で表している．グラフィクスメモリから送信用のバッファには，ピクセル単位で読み出すため，受信用バッファに比べ容量が大きくなっている．

表 6: ノード 0 での送信用バッファ

バッファ名	サイズ	送信先ノード番号	内容
send1	$(L-2)\times(N-2)\times 4$	2	f_1, f_2, f_3, f_4
send2	$(M-2)\times(N-2)\times 4$	1	f_1, f_2, f_3, f_4
send3	$(L-2)\times(N-3)\times 4$	2,3	f_7, f_8, f_9, f_{10}
send4	$(M-2)\times(N-3)\times 4$	1	f_7, f_8, f_9, f_{10}
send5	$(L-2)\times(M-2)\times 4$	4,5,6,7	f_7, f_8, f_9, f_{10}
send6	$(L-2)\times(N-3)\times 4$	2,3	$f_{11}, f_{12}, f_{13}, f_{14}$
send7	$(M-2)\times(N-3)\times 4$	1	$f_{11}, f_{12}, f_{13}, f_{14}$
send8	$(L-2)\times(M-2)\times 4$	4,5,6,7	$f_{11}, f_{12}, f_{13}, f_{14}$
send9	$(L-2)\times(M-2)\times 4$	4	f_5, f_6, f_0
send10	$(L-2)\times(M-2)\times 4$	4	f_5, f_6, f_0

表 7: ノード 0 での受信用バッファ

バッファ名	サイズ	受信先ノード番号	内容
recv1	$L\times(N-2)\times 2$	1,2	f_3, f_4
recv2	$L\times(N-3)\times 4$	1,2,3	f_8, f_9, f_{10}
recv3	$L\times M\times 4$	4,5,6,7	f_7, f_8, f_9, f_{10}
recv4	$L\times(N-3)\times 4$	1,2,3	f_{12}, f_{13}, f_{14}
recv5	$L\times M\times 4$	4,5,6,7	$f_{11}, f_{12}, f_{13}, f_{14}$
recv6	$L\times M\times 4$	4	f_5, f_6

$f_1 \sim f_4$ の衝突計算のための描画の際には，$1 \le l \le L-2$，$m = M-2$ の領域をピクセルバッファから読み取り，send1 に格納し，f_2 のみをノード 2 へ送信する．また，同時に $l = L-2$，$1 \le m \le M-2$ の領域を読み取り，send2 に格納し，f_1 のみをノード 1 へ送信する．これを $1 \le n \le N-2$ で繰り返す．

$f_7 \sim f_{10}$ に関しては，$1 \le n \le N-3$ の場合には $1 \le l \le L-2$，$m = M-2$

の領域を，send3 に格納し，f_7, f_8 をノード 2 へ送信し，$l = L-2$ の f_7 をノード 3 へ送信する．同時に $l = L-2$，$1 \leq m \leq M-2$ の領域を send4 に格納し，f_7, f_{10} をノード 1 へ送信する．$n = N-2$ の場合には全ての領域を send5 に格納し，全てをノード 4 へ，$1 \leq l \leq L-2$，$m = M-2$ の領域の f_7, f_8 をノード 6 へ，$l = L-2$，$1 \leq m \leq M-2$ の領域の f_7, f_{10} をノード 5 へ，$l = L-2$，$m = M-2$ の f_7 をノード 7 へ送信する．

f_{11}, f_{14} に関しては，$2 \leq n \leq N-2$ の場合には $1 \leq l \leq L-2$，$m = M-2$ の領域を，send6 に格納し，f_{11}, f_{12} をノード 2 へ送信し，$l = L-2$ の f_{12} をノード 3 へ送信する．同時に $l = L-2$，$1 \leq m \leq M-2$ の領域を send7 に格納し，f_{11}, f_{14} をノード 1 へ送信する．$n = 1$ の場合には全ての領域を send8 に格納し，全てをノード 4 へ，$1 \leq l \leq L-2$，$m = M-2$ の領域の f_{11}, f_{12} をノード 6 へ，$l = L-2$，$1 \leq m \leq M-2$ の領域の f_{11}, f_{14} をノード 5 へ，$l = L-2$，$m = M-2$ の f_{11} をノード 7 へ送信する．

f_5, f_6, f_0 に関しては，$n = 1, N-2$ の場合にのみ $1 \leq l \leq L-2$，$1 \leq m \leq M-2$ の領域を読み取り，$n = N-2$ の場合は send9 に，$n = 1$ の場合は send10 に格納し，send9 から f_5 を，send10 から f_6 を取り出し，ノード 4 へ送信する．

受信用バッファでは，テクスチャとしてグラフィクスメモリに転送できる形式で受信する．ここで受信する情報は，ノードが担当する領域が接する外部ノードの格子点から並進してくる分布関数の情報のみである．$f_1 \sim f_4$ の並進の際には recv1 によって受信するが，受信する分布関数は f_3, f_4 の 2 種類のみである．$f_7 \sim f_{10}$ の並進の際には $2 \leq n \leq N-1$ については recv2，$n = 1$ については recv3，$f_{11} \sim f_{14}$ の並進の際には $1 \leq n \leq N-3$ については recv4，$n = N-2$ については recv5，$f_5 \sim f_6$ の並進の際には $n = 1, N-2$ のとき recv6 をそれぞれテクスチャとして参照し，空間の断面に接する格子点へ向かう分布関数の情報を得る．

6.4 評価

1 台で実行した場合と，本章のとおり 8 台で実行した場合での 100 タイムステップの実行時間の比較を図 15 に示す．問題サイズは $64 \times 64 \times N$ とし，1 台のグラフィクスメモリにデータが十分入りきる範囲 (5.3 節) で計測した．ただし，1 台で実行する場合，周囲に 1 格子分のマージンをとっているため，実際の計算領域は $62 \times 62 \times (N-2)$ となる．また，8 台で実行する場合，各ノード

で周囲にマージンをとっているため，実際の計算領域は $60 \times 60 \times (N-4)$ となる．8 台で実行する場合，1 台で実行した場合の計算領域を計算した場合に相当する時間を表すように修正を加えた．

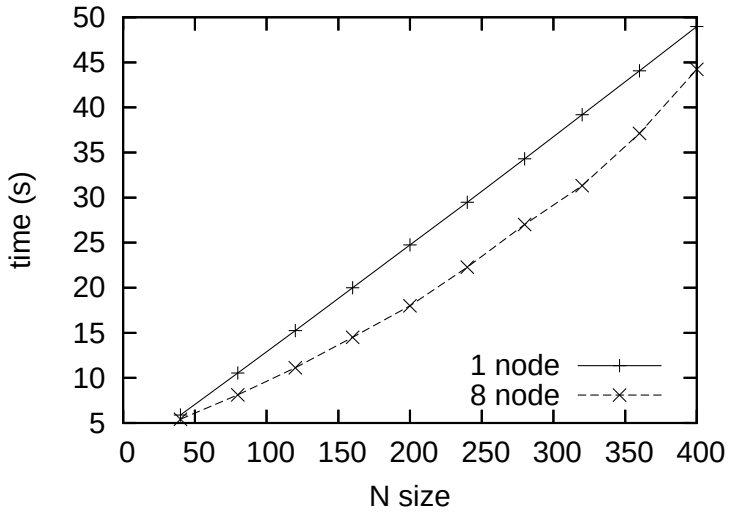

図 15: 1 台と 8 台での実行時間 ($64 \times 64 \times N$,100 time step)

1 台のグラフィクスメモリに入りきる問題サイズでは，並列化による顕著な高速化が実現できたとはいえない．この原因として，通信の最適化が十分に行えていないことが考えられる．send1∼send4，send6，send7 について，描画の度に送信用バッファへの読み込みを行ったが，本実装では，読み込みの度に読み込みに続けてノード間通信を行った．このため，通信頻度が増大し，高速実行を妨げる要因となっているものと思われる．問題サイズを $128 \times 128 \times N$ とした場合の 1 台と 8 台での 100 タイムステップの実行時間を図 16 に示す．$64 \times 64 \times N$ の場合に比べ，並列化による実行速度向上の割合が高くなっている．同じ方法で送受信したものであるが，スライス数が減少しているため，通信頻度は少なくなっている．通信頻度を抑えるため，描画の度に (送信バッファへの読み込みの度に) 送信するのではなく，各スライスのデータをまとめて送るようにすることで，実行速度の向上が期待できる．

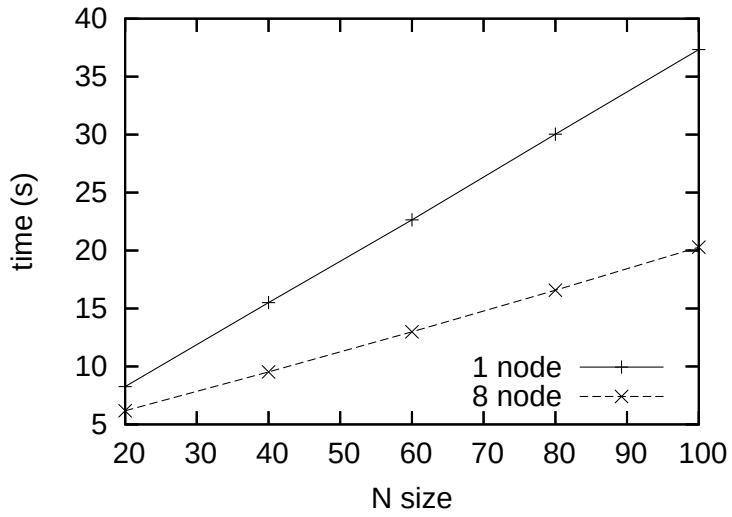

図 16: 1 台と 8 台での実行時間 ($128 \times 128 \times N$,100 time step)

6.5 本章のまとめ

大規模なインタラクティブシミュレーションの効果的な実行を目指し，並列計算の適用手法を説明し，実装報告を行った．これにより，1 台では事実上不可能であった大規模なシミュレーションの実現の可能性が示唆されたが，主にノード間通信において更なる改善の余地があると考えられる．これについては今後の課題とする．

第7章　おわりに

本稿では，インタラクティブな流体シミュレーションを実現するための手法について説明し，実装報告を行った．計算を効果的にGPU上に実装できることを示し，GPUを用いることによってCPUによる計算よりもはるかに高速な計算を実現できることを確認した．さらに，触覚デバイスを用いて流体シミュレーションに操作を加え，結果を視覚と力覚として提示させるシステムの実装によって，シミュレーションによってオペレータがより現実に近い仮想現実を体験できる可能性を示した．また，並列計算を適用する手法を示し，実装報告を行うことで，大規模なインタラクティブシミュレーションシステムに流体計算を組み込める可能性を示唆した．

その中で，注意すべき問題や，未解決の課題も見出された．まず，本研究では計算にGPUを用いることでシミュレーションの高速実行を実現したが，現状でサポートされているGPUの計算精度の限界から，数値シミュレーションに用いるには注意が必要であるといえる．また，本研究では3次元シミュレーション結果の視覚による提示手法については詳細な検討は行わなかったが，インタラクティブシミュレーションではオペレータに対する実時間での効果的なフィードバックが必要であり，効果的で高速な可視化処理についての検討も必要である．

我々の周りには，常に流体が存在しており，常に流体に関わりながら生活している．現実により近い仮想体験を実現するためのインタラクティブシミュレーションに流体シミュレーションを取り入れることで，シミュレーションの適用範囲が広がり，結果をより現実に近いものにすることが可能となるであろう．

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，嶋田創助手，三輪忍助手に深く感謝致します．さらに，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] 富田 眞治, 超高速体感型シミュレーションシステムの研究, 科学研究費補助金, 平成 16 年度基盤研究 (S) 新規採択課題の概要について, 日本学術振興会, p.10 (2004)

[2] G. McNamara and G. Zanetti, Use of the Bolzmann equation to simulate lattice-gas automata, *Phys. Rev. Lett.* 61, Issue 11, pp.2332-2335 (1988)

[3] S. Chen and G. D. Doolen, Lattice Boltzmann Method for Fluid Flows, *Annu. Rev. Fluid Mech.* 30, pp.329-364 (1998)

[4] 蔦原 道久, 高田 尚樹, 片岡 武, 格子気体法・格子ボルツマン法, コロナ社 (1999)

[5] T. Ertl, D. Weiskopf, M. Kraus, K. Engel, M. Weiler, M. Hopf, S. Rottger and C. Rezk-Salama, Programmable Graphics Hardware for Interactive Visualization, Eurographics2002 Tutorial Note(T4) (2002)

[6] 森 眞一郎, 篠本 雄基, 五島 正裕, 中島 康彦, 富田 眞治, 汎用グラフィクスカード上での簡易シミュレーションと可視化, 電子情報通信学会信学技報, CPSY2004-24, pp.25-30 (2004)

[7] General Purpose Computation on Graphics Processor, http://www.gpgpu.org/

[8] 高橋 隆, 室井 克信 編, 人工現実感手術室, IPSJ Magazine Vol.43 No.5 (2002)

[9] 牛島 省ほか 編, 計算力学の最前線, 土木学会誌 Vol.88 No.8 (2003)

[10] 小沢 拓, 棚橋 隆彦, 二相系格子ボルツマン法の非構造格子への適用, 計算工学会論文集, No. 20050006 (2005)

[11] M. Woo, J. Neither and T. Davis, OpenGL プログラミングガイド, ピアソンエデュケーション (1997)

[12] NVIDIA Corporation, Cg Toolkit User's Manual Release 1.2 (2004)

[13] M. J. Kilgard, NVIDIA OpenGL Extension Specifications, NVIDIA Corporation (2004)

[14] K. E. Hillesland, A. Lastra, GPU floating-point Paranoia, GPGPU Workshop (2004)

[15] 小松原 誠，森 眞一郎，中島 康彦，富田 眞治，汎用グラフィクスカード

を用いた格子ボルツマン法による流体シミュレーション，情報処理学会研究報告 2005-ARC-163, pp.37-42 (2005)

[16] Y. H. Qian, D. d'Humières, and P. Lallemand, Lattice BGK models for the Navier-Stokes equation *Europhys. Lett.* 17, pp.479-484 (1992)

[17] U. Frisch, D. d'Humières, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. P. Rivet, Lattice gas hydrodynamics in two and three dimensions, *Complex Syst.* 1, pp.649-707 (1987)

[18] P. A. Skordos, Initial and boundary conditions for the lattice Boltzmann method, *Phys. Rev. E* 48, pp.4823-4842 (1993)

[19] I. Ginzbourg and D. d'Humières, Local second-order boundary methods for lattice Boltzmann models, *J. Stat. Phys.* 84, pp.927-971 (1996)

[20] M. Bouzidi, M. Firdaouss and P. Lallemand, Momentum transfer of a Boltzmann-lattice fluid with boundaries, *Phys. Fluids* 13, pp.3452-3459 (2001)

[21] P. Lallemand and L. S. Luo, Lattice Boltzmann method for moving boundaries, *J. Comput. Phys.* 184, pp.406-421 (2003)

[22] R. Mei, D. Yu, W. Shyy and L. S. Luo, Force evaluation in the lattice Boltzmann method involving curved geometry, *Phys. Rev. E* 65, 041203 (2002)

[23] D. Yu, R. Mei, L. S. Luo and W. Shyy, Viscous flow computations with the method of lattice Boltzmann equation, *Progr. Aero. Sci.* 39, pp.329-367 (2003)

[24] SensAble Technologies
http://www.sensable.com/

付録

以下は第 3 章での実装で用いた計算プログラムである.

```
//fp2
struct fpIN{
   float4 texCoord :TEX0;
};
struct fpOUT{
   float4 col :COLOR;
};
fpOUT main(   fpIN IN,
              uniform samplerRECT tex2,
              uniform samplerRECT tex3,
              uniform samplerRECT tex4,
              uniform float FAI,
              uniform float RHO){
   fpOUT OUT;
   float4 u=f4texRECT(tex3,IN.texCoord.xy);
   float kabe=f4texRECT(tex4,IN.texCoord.xy).x;
   float4 f=f4texRECT(tex2,IN.texCoord.xy);
   float4 fe,fee;
   int4 e;
   float u2;
   if(kabe<0.5f){
      u2=u.x*u.x+u.y*u.y;
      e=int4(1,-1,-1,1);
      fee=e*u.x+(e.xxzz)*u.y;
      fe=(u.w-RHO+u.w*(3.0f*fee+4.5f*fee*fee-1.5f*u2))/36.0f;
      OUT.col=f-(f-fe)/FAI;
   }
   else OUT.col.xyzw=f.zwxy;
   return OUT;
}
```

図 A.1: フラグメントプログラム fp2

```
//fp3
struct fpIN{
   float4 texCoord :TEX0;
};
struct fpOUT{
   float4 col :COLOR;
};
fpOUT main(   fpIN IN,
              uniform samplerRECT tex1,
              uniform samplerRECT tex4,
              uniform float sizeY)
{
   fpOUT OUT;
   float kabe=f4texRECT(tex4,IN.texCoord.xy).x;
   float2 uv=IN.texCoord.xy;
   float4 f;

   uv.x-=1.0f;
   f.x=f4texRECT(tex1,uv).x;
   uv.x+=2.0f;
   f.z=f4texRECT(tex1,uv).z;
   uv=IN.texCoord.xy;
   if(uv.y>sizeY-1.0f)uv.y=0.5f;
   else uv.y+=1.0f;
   f.w=f4texRECT(tex1,uv).w;
   uv=IN.texCoord.xy;
   if(uv.y<1.0f)uv.y=sizeY-0.5f;
   else uv.y-=1.0f;
   f.y=f4texRECT(tex1,uv).y;
   OUT.col=f;
   return OUT;
}
```

図 A.2: フラグメントプログラム fp3

```
//fp4
struct fpIN{
   float4 texCoord :TEX0;
};
struct fpOUT{
   float4 col :COLOR;
};
fpOUT main(   fpIN IN,
              uniform samplerRECT tex2,
              uniform samplerRECT tex4,
              uniform float sizeY)
{
   fpOUT OUT;
   float kabe=f4texRECT(tex4,IN.texCoord.xy).x;
   float2 uv=IN.texCoord.xy;
   float4 f;

   uv.x-=1.0f;
   if(uv.y<1.0f)uv.y=sizeY-0.5f;
   else uv.y-=1.0f;
   f.x=f4texRECT(tex2,uv).x;
   uv.x+=2.0f;
   f.y=f4texRECT(tex2,uv).y;
   uv=IN.texCoord.xy;
   uv.x+=1.0f;
   if(uv.y>sizeY-1.0f)uv.y=0.5f;
   else uv.y+=1.0f;
   f.z=f4texRECT(tex2,uv).z;
   uv.x-=2.0f;
   f.w=f4texRECT(tex2,uv).w;
   OUT.col=f;
   return OUT;
}
```

図 A.3: フラグメントプログラム fp4

```
//fp5
struct fpIN{
   float4 texCoord :TEX0;
};
struct fpOUT{
   float4 col :COLOR;
};
fpOUT main(   fpIN IN,
              uniform samplerRECT tex1,
              uniform samplerRECT tex2,
              uniform samplerRECT tex3,
              uniform samplerRECT tex4,
              uniform float FAI,
              uniform float RHO)
{
   fpOUT OUT;
   float4 f1to4,f5to8,f1to8;
   float fe0;
   float kabe=f4texRECT(tex4,IN.texCoord.xy).x;

   OUT.col=f4texRECT(tex3,IN.texCoord.xy);
   if(kabe<0.5f){
      f1to4=f4texRECT(tex1,IN.texCoord.xy);
      f5to8=f4texRECT(tex2,IN.texCoord.xy);
      f1to8=f1to4+f5to8;
      fe0=(OUT.col.w-RHO+OUT.col.w*(-1.5f*
         (OUT.col.x*OUT.col.x+OUT.col.y*OUT.col.y)))*4.0f/9.0f;
      OUT.col.z=OUT.col.z-(OUT.col.z-fe0)/FAI;
      OUT.col.w=f1to8.x+f1to8.y+f1to8.z+f1to8.w+OUT.col.z+RHO;
      OUT.col.x=(f1to4.x+f5to8.x+f5to8.w-f1to4.z-f5to8.y-f5to8.z)/OUT.col.w;
      OUT.col.y=(f1to4.y+f5to8.x+f5to8.y-f1to4.w-f5to8.z-f5to8.w)/OUT.col.w;
   }
   return OUT;
}
```

図 A.4: フラグメントプログラム fp5

```
//fp6
struct fpIN {
   float4 texCoord :TEX0;
};
struct fpOUT{
   float4 col :COLOR;
};
fpOUT main(   fpIN IN,
              uniform samplerRECT tex3,
              uniform samplerRECT tex4)
{
   fpOUT OUT;
   float2 u;
   float h;
   int n;
   float kabe;
   float fl;

   u=f4texRECT(tex3,IN.texCoord.xy).xy;
   kabe=f4texRECT(tex4,IN.texCoord.xy).x;
   h=240.0f-12000.0f*sqrt(u.x*u.x+u.y*u.y);
   if(h<0.0f)h=0.0f;
   n=(int)floor(h/60);
   fl=h/60.0f-n;

   if(n==0){OUT.col.x=1.0f;OUT.col.y=fl;}
   if(n==1){OUT.col.x=1.0f-fl;OUT.col.y=1.0f;}
   if(n==2){OUT.col.y=1.0f;OUT.col.z=fl;}
   if(n==3){OUT.col.y=1-fl;OUT.col.z=1.0f;}
   if(n==4){OUT.col.x=fl;OUT.col.z=1.0f;}
   if(n==5){OUT.col.x=1.0f;OUT.col.z=1-fl;}

   if(kabe>0.5f)OUT.col=float4(1.0,1.0,1.0,1.0);
   return OUT;
}
```

図 A.5: フラグメントプログラム fp6