粒子を用いた火炎の
対話的なビジュアルシミュレーション

間淵 聡

2012年3月

慶應義塾大学大学院

理工学研究科

開放環境科学専攻

博士（工学）学位論文

要旨

近年の対話的コンピュータグラフィックス (CG) は，GPU (Graphics Processing Unit) の進歩と共通処理のミドルウェア化によって，急速な普及を遂げている．この技術は，従来のゲームグラフィックスのみならず，映画制作や都市設計など，様々な分野へ適用が拡げられている．

対話的 CG において，物理シミュレーションは特に重要な位置を占める．現在，固体や弾性体などの多くの物理シミュレーションは対話的な速度で実行可能となっているが，気体状の現象の物理シミュレーションは未だ充分な研究が行われていない．本論文では，手作業での表現が特に難しい，火炎の対話的なビジュアルシミュレーション法を提案した．このビジュアルシミュレーション法では，物理的妥当性を考慮しながらも，視覚的妥当性を優先した．そのため，物理シミュレーション法だけでなく，画像を高精度にするためのレンダリング法も開発した．また，本論文では，一貫して粒子を用いた．これにより，格子を用いた場合と違い，ユーザに空間制限を課す必要がなくなり，しかも対話性を向上させることができた．

具体的には，物理シミュレーション法として，広く用いられている粒子法である，SPH (Smoothed Particle Hydrodynamics) を拡張した，火炎シミュレーション法を提案した．また，高精度なレンダリング法として，ボリュームレンダリング法の 1 つである粒子ベースボリュームレンダリングを，火炎のビジュアルシミュレーション向けに簡易化した，火炎レンダリング法を提示した．さらに，GPU を用いることによって，これらの物理シミュレーション法とレンダリング法をともに高速化した．

本論文での成果により，空間制限のない，対話的な速度で動く，火炎のビジュアルシミュレーションが可能となり，ユーザが自在に火炎を扱えるようになった．

Abstract

In recent years, interactive Computer Graphics (CG) techniques have been getting more available with advances in GPU (Graphics Processing Unit) and functions-integrated middleware. Applications of these techniques have been expanding into various fields, such as conventional game graphics as well as movie production and city design.

Physics simulations serve as an essential factor of interactive CG. At present, many physics simulations, including solid simulations and elastic simulations, can be executed at an interactive speed, whereas simulations of gaseous phenomena are not well established yet. In this thesis, methods for interactive visual simulation of flame, which is very hard to represent in a manual fashion, are introduced. Main focus of the methods is placed on visual plausibility while taking physical validity into consideration. Thus, physics simulation algorithms as well as high-quality rendering algorithms are developed. Consistent use of particles can liberate the users from spatial restriction imposed by grids and lead to improved interactivity.

Specifically, what were proposed in this thesis include a method for flame simulation by extending a widely-used particle method called SPH (Smoothed Particle Hydrodynamics) and a high-quality flame rendering method, which is a simplified version of an existing volume rendering method called Particle-Based Volume Rendering. In addition, both of these simulation and rendering methods were accelerated using a modern GPU.

As a consequence of this thesis, the proposed interactive and space restriction-free visual simulation methods succeeded to open the door for the users to render flames freely.

目 次

表 目 次

図 目 次

第1章　序論

本章では，本論文の前提となる対話的コンピュータグラフィックス（CG）について解説する．CG は，実用と研究とが互いに刺激されて進んできた分野であるため，本章前半で実用的な側面を説明し，後半で技術的な側面にふれる．

1.1　対話的な CG と物理シミュレーション

CG は元々映画産業において 1980 年代から大きく発達した．当時の CG はハイエンドなワークステーションを用いて制作されていた．対話的 CG は従来，CAD やフライトシミュレータ等に用いられていたが，同じくハイエンドなワークステーションが必要であり，一般には普及していなかった．

対話的 CG を一般向けにしたのはコンピュータゲーム機であった．1990 年代後半に，Play Station（ソニー・コンピュータエンタテインメント），Nintendo 64（任天堂）などが登場し，対話的な 3DCG が一般に広まった．これらは CG 処理に関して，当時の PC を凌いでいた．2000 年前後から，CPU と GPU（Graphics Processing Unit）の性能向上に伴い，一般的な PC 上でも対話的な 3DCG が利用可能となった．さらに近年では，プログラマブル GPU が登場したことで，より高度な CG が利用可能となっている．

特に最近では，CPU の性能が頭打ちになっている反面，GPU の性能向上がひじょうに大きく，コンピュータゲーム機よりも PC の方が高度な CG 処理を可能としている．また，GPGPU（General Purpose computing on GPU）とよばれる GPU を用いた汎用計算技法も発達してきた．このように，近年では，ハードウェアの多様化，ソフトウェアの複雑化が発生したため，共通となる処理が CG エンジンとして作成されるようになった．これにより，ハードウェアや OS レベルの違いを吸収することができるようになった．これらのエンジンは，各社が独自に開発していたが，高品質なエンジンは，他社への提供を行っている．Unreal Engine (Epic Games, 1998)，Cry Engine (Crytek, 2004)，Unity (Unity Technologies, 2005) などが代表的である．これらでは，対話的 CG が従来苦手としていた，自然物や肌の表現など，多くの表現はプリレンダリングと大差がなくなっている．旧来のオフラインで培われた技術は近年では対話的グラフィックスへ適用が進んでおり，そのため，映画，都

市設計，教育など，ゲームグラフィックス以外の分野にも進出している．

CG エンジンの構成要素の 1 つに，物理エンジンが挙げられる．物理エンジンはさらに特殊であり，専門の物理エンジンメーカから購入するケースが多い．代表的なものとしては，NVIDIA PhysX (NVIDIA, 2008)， Havok Physics (Havok, 2000) などが挙げられる．これらは，剛体，弾性体，軟体などのシミュレーションを可能としている．物理エンジンでは，厳密性よりも視覚的な妥当性に重点がおかれる．

もちろん物理エンジンにおいても，困難となっているシミュレーションはある．これは，コンピュータでシミュレーションする以上，対象を何らかの構成要素に分解する必要があるためである．剛体であれば，面単位で，古典力学を用いることによって比較的容易に計算が可能である．弾性体や軟体であれば，それに加えて頂点同士の相互作用を計算する必要が出てくるため，より難しくなる．流体は形状が大きく変化するため，構成要素を充分に小さくとらなくてはならなず，さらに困難となる．現在のところ，一部の物理エンジンは液体シミュレーションをサポートするが，気体に関してのサポートは不充分である．

このような，視覚的妥当性を重点においた物理シミュレーションやレンダリングは，一般にビジュアルシミュレーションとよばれる．また，近年のビジュアルシミュレーションでは，GPU によるアクセラレーションも重要となっている．次節で GPU の進歩をまとめ，1.3 節で，本論文で注目する，火炎のビジュアルシミュレーションをまとめる．

1.2 GPU の進歩

上述のとおり，近年の CG の発展には，GPU の進歩が大きな役割を果たしている．本論文でも，GPU を用いてアクセラレーションを行うため，本節において解説を行う．

従来はのレンダリングでは，固定 API（Application Programming Interface）が用いられてきた．これは実装が容易であった反面，用意された API の範囲の表現しかできなかった．近年では，要求される表現の多様化に伴い，各頂点や各画素単位の制御を，自在に行うことが望ましくなった．そこで登場したのがプログラマブル GPU であった．一般に，このプログラミング言語は，シェーダもしくはプログラマブルシェーダとよばれる．

シェーダで記述されたプログラムは，GPU 内で並列に処理される．通常，CG の描画に用いられる頂点や，画素は膨大な個数に上るが，基本的には独立した処理であるため，GPU による並列処理に向いている．もちろん，頂点を定義しなければ投影される画素が判明しないため，バーテックスシェーダステージ，ピクセルシェーダステージの処理として，明示的に割り振る必要があり，元々は並列処理といっても限定されたものであった．

しかしながら，CG で多用されるテクスチャを，読み書き可能なメモリとみなして活用することで，汎用的な並列処理が可能であることが示されはじめ，大規模行列演算の高速化や，物理演算の高速化などが数十倍のオーダで実現された．これが GPGPU とよばれるようになった．当初はシェーダで記述する他なかったため，再現性や移植性の面で非実用的であったが，NVIDIA CUDA (NVIDIA, 2006) が C 言語に似た形式の，汎用並列演算用のフレームワークを提供したことから，再現性や移植性が大幅に向上した．

一方，近年では，本来のプログラマブル GPU という点からも，機能追加が図られている．上述のとおり，バーテックスシェーダステージとピクセルシェーダステージとが存在していたが，新たなシェーダステージも追加されている．このステージによって，GPU 内において，頂点の増加が可能となり，対話的 CG において頻繁に用いられる，詳細度制御が，高速にかつ少メモリで実現されるようになった．

以上をまとめると，GPU によるアクセラレーションは，

- シェーダによるレンダリングの高速化
- GPGPU による汎用計算の高速化

の 2 点に分けて考えられる．本論文では，GPU によるアクセラレーションとして，両方を用いる．

1.3 火炎のビジュアルシミュレーション

気体を用いて行われる表現としては，主に煙と火炎が挙げられる．しかしながら，特に火炎シミュレーションは CG で頻繁に用いられる表現であるにもかかわらず，充分な研究が行われていない．そのため，乱数による粒子法を用いる，ノイズを用いる，といった古典的な手法が用いられている．

オフラインシミュレーションも含めて，火炎の関連するシミュレーションを振り返ると，主には，格子法 (Nguyen et al., 2002) と，粒子法 (Müller et al., 2003) とに分けて考えられる．格子法では，シミュレーション空間を格子セルに離散化し，隣接格子間での相互関係を計算する．粒子法では，シミュレーション対象を粒子に離散化し，近傍粒子間の相互作用を計算する．格子法は計算精度に優れるが，シミュレーション空間を事前に定義しておかなければならない．また，速度を重視する場合には，格子セル数を少なくしなければならない．粒子法は計算精度の面でやや劣るが，シミュレーション空間を事前に定義しておく必要がない．ダム崩壊問題のような，空間が定まった現象には格子法が有利であるが，気体

のシミュレーションにおいては，空間が固定されない粒子法が有利である．そのうえ，格子法では，圧力のポアソン方程式を解く必要があるため，計算速度が低下しやすい．対して，現在広く用いられている粒子法 (Müller et al., 2003) では，圧力のポアソン方程式を解いていないため，計算速度が低下しにくい．気体の場合は，厳密な質量保存を追求しなくても，視覚的に妥当な結果が得られることは既に示されている (Crane et al., 2007)．したがって，速度面からも，粒子法が有利である．

オフラインレンダリングも含めて，火炎のレンダリング法を振り返ると，主には，格子データに対するレンダリング法と，粒子データに対するレンダリング法がそれぞれ存在する．格子データに対するレンダリング法として一般的なものは，レイマーチング（Ray Marching (Kaufman, 1991)）である．レイマーチングは，視点から光線（レイ）を生成し，少しずつ進めながら格子セル上の物理量を補間し，最終的にその画素に書き込まれる色を計算する方法である．レイマーチングは GPU 上での実装により高速化が進んでいるが，格子空間外は扱うことができない．粒子データに対するレンダリング法として一般的なものは，ビルボード（Billboard (Schaufler, 1995)）である．ビルボードでは，各粒子座標に，スクリーンに対して平行な四角形を用意し，その四角形にテクスチャを書き込んでおき，重ね合わせを行うことでレンダリングを行う．ビルボードは，格子空間を事前に定義しておく必要がないが，イメージベースレンダリングに近い手法であり，テクスチャを用意すれば多様な表現が可能であるという利点があると同時に，多くの試行錯誤を重ねなければならないという欠点がある．

1.4 本論文の目的と寄与

以上をまとめると，現状では対話的な火炎表現はシミュレーションとレンダリング両方について不充分である．ハードウェアが進展しても，現在提案されているプリレンダリングの手法を対話的に実行することは困難である．

本論文では，対話的なアプリケーション一般に応用可能であり，物理エンジンに搭載可能な，火炎のビジュアルシミュレーション法を提示する．

一般的な物理エンジンに要求される要求は，主に次のとおりである．

- 標準的な PC で 1 ステップあたり 0.03[sec] 以下の計算時間に抑えられること．これは人間が違和感を感じにくい，30[fps] 以上のフレームレートを確保できるため
- モジュールが分離可能であること．物理エンジンはあくまで CG エンジンの一部であるため

- ハードウェア構成に応じて詳細度制御が可能であること．マルチプラットフォームに対応するため

これらの要求を参考に，本研究では具体的に次の要求が必要であると判断した．

- 現在の標準的な PC で 1 ステップあたり 0.03-0.05[sec] 程度の計算時間に抑えられること．今後のハードウェアの向上により 30[fps] 以上のフレームレートの確保が期待できるため
- シミュレーション法とそれを効果的に描画するレンダリング法を区別すること．モジュールの分離に対応するため
- 既存技術の拡張であること．モジュールの分離を効果的に利用するため
- CPU だけでも一定の質が確保できるが，GPU を用いることで高い質が確保できること．ハードウェア構成の違いに対応するため

本研究では，火炎を扱うことから，さらに次の要求を追加した．

- 拡張性が高いこと．特にバーチャルリアリティへの応用が考えられるため
- 大規模空間を扱うことができること．火炎は事前にシミュレーション範囲を定義しにくいため
- パラメータ調整が直感的であること．流体に加えて化学反応を扱うことになり，パラメータが複雑化しやすいため

これらを本研究の基本要求とした．

応用範囲は，前述の通り，コンピュータゲーム，映画，都市設計，防災訓練などであり，様々な分野に適用が可能である．ただし，本手法はシミュレーション法，レンダリング法ともに，正確性を追求したものではなく，視覚的妥当性を追求した研究である．防災訓練システムなど，人命に関わるアプリケーションに適用する際は，ユーザに周知させるか，より高精度なシミュレーション法と組み合わせて使わなければならない．

1.5 本論文の構成

本論文の章構成と，発表文献の概要を**表 1.1** に示す．本論文では，後半になるにつれて火炎のビジュアルシミュレーションの完成度を上げていく．しかしハードウェア要件も次

第に高くなる．次章で関連研究について述べた後，第 3 章で，提案する火炎のシミュレーション法について述べる．第 4 章で，そのシミュレーション結果を効果的に活用する，新たなレンダリング法を述べる．GPGPU が利用可能なハードウェア向けに，第 5 章で，シミュレーションの GPU を用いた高速化を提示する．さらに，プログラマブルシェーダが利用可能なハードウェア向けに，第 6 章で，レンダリングの GPU を用いた高速化について述べ，さまざまな評価を第 7 章で行い，最後に，第 8 章で結論を述べ，今後の展望を示す．

表 1.1: 本論文の構成と発表文献

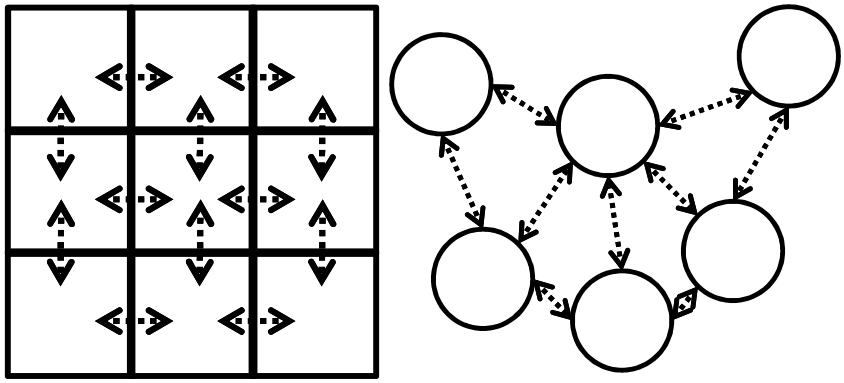
	シミュレーション	レンダリング
CPU	第 3 章 (間淵 他, 2011a)	第 4 章 (間淵 他, 2011b)
GPU	第 5 章 (Mabuchi et al., 2011)	第 6 章

第2章　関連研究

本章では火炎のシミュレーション法と，火炎のレンダリング法についての関連研究を述べる．

2.1　シミュレーション

CG における流体に関する研究は，格子法と粒子法とに大別して考えることができる．格子法でも，一部に粒子を用いたり，粒子法でも，一部に格子を用いたりすることがあり，必ずしも厳密に区別できるわけではないが，計算主体を考慮して大別した．格子法と粒子法の概略図を**図 2.1** に示す．



(a) 格子法　　(b) 粒子法

図 2.1: 格子法と粒子法

格子法では，シミュレーション空間を格子に離散化し，隣接格子間での相互関係を計算することによって，流体の動きをシミュレーションする．粒子法では，シミュレーション対象を粒子に離散化し，近傍粒子間の相互作用を計算することによって，流体の動きをシミュレーションする．

.

2.1.1 格子法

CG における，格子法を用いたシミュレーションでは，液体と気体との区別をあまりつけずに研究が行われている．火炎シミュレーションを初めて格子ベースで導入したのは，Inakage (Inakage, 1990) である．この研究では化学反応を扱っているが，かなり限定的なシーンを対象としている．Chiba (Chiba et al., 1994) は渦場を，Stam (Stam and Fiume, 1995) は拡散モデルを用いた 2 次元の火炎のシミュレーションをそれぞれ行った．

ハードウェアの発展に伴って，物理モデルに厳密に従うようになり，Foster (Foster and Metaxas, 1996) は液体のシミュレーションを，Stam (Stam, 1999) は気体のシミュレーションを行った．特に Stam (Stam, 1999) の手法は Stable Fluids として定着し，Fedkiw (Fedkiw et al., 2001) が煙のシミュレーションに応用した．液体の表現には，体積保存の面から問題があったが，Foster (Foster and Fedkiw, 2001) がレベルセット法を導入し，この問題に対処し，充分な写実性を提供できるようになった．さらに Enright (Enright et al., 2002) により，レベルセット法を改良した，粒子レベルセット法が導入され，さらなる写実性を提供した．

これらを活用することにより，Nguyen (Nguyen et al., 2002) が，極めて写実性の高い火炎シミュレーションを実現した．また，Feldman (Feldman et al., 2003) は，格子と粒子を効果的に組み合わせることで，爆発火炎のシミュレーションを可能とした．Kawada (Kawada and Kanai, 2011) は，これをもとに，爆発火炎の演出と物理シミュレーションとの両立を図った．

さらに，これらの手法は，物理シミュレーション一般として統一化が可能であることを，Losasso (Losasso et al., 2006) が，固体や布の燃焼で示した．化学反応をより正確に扱ったモデルとしては，文献 (Ihm et al., 2004)， (Kang et al., 2007) が挙げられる．しかし，化学的に厳密なモデルを用いなくとも，流体力学に従うことで高精度な表現が可能であることは上述の研究結果から示されている．これらの研究では，高精度なシミュレーションが可能であることを示したが，圧力のポアソン方程式を解かなければならないため，極めて限られた空間を対象として数秒分のアニメーションを作成するだけで数日かかることが報告されている．特に火炎に関するシミュレーションの可能性と限界は文献 (Nguyen et al., 2003) にまとめられている．

したがって，計算効率化のために，格子セル未満のデータを扱うことが必要となった．具体的には，オクトツリーを用いることで計算が効率的になることを，Losasso (Losasso et al., 2004) は示した．また，Selle (Selle et al., 2005) は，格子解像度未満の渦のシミュレーションを行った．さらに，Hong (Hong et al., 2007) は，DSD （Detonation Shock Dynamics）を用いることで火炎のディテールの表現を追求した．

これらの手法を用いても，計算コストの大幅な削減には至らなかったが，近年では，GPGPUの台頭により，対話性をもたせることが可能となった．Crane (Crane et al., 2007) は，火炎に関しては，圧力のポアソン方程式を解く際に，適当回数で収束を打ち切っても，視覚的に問題にならないことを示し，対話的な火炎のシミュレーションを実現した．また，少数の粒子によるシミュレーションを CPU 上で行い，詳細なシミュレーションとレンダリングを複数の GPU 上で行う手法が Horvath (Horvath and Geiger, 2009) によって提案された．また，文献 (Melek, 2007)， (Zhu et al., 2010) では，同じく GPGPU を活用し，固体の燃焼を対話的に扱ったと報告している．Pfaff (Pfaff et al., 2010) は，煙の渦に特化し，対話的な速度で，少数の格子セルから複雑な煙を表現した．さらに，Chentanez (Chentanez and Müller, 2011) は，シミュレーション空間を流体表面付近に限定することで，擬似的に大規模空間のシミュレーションを可能にした．

以上のように，液体・気体を問わず，対話的な流体シミュレーションを実現できるようになっているが，根本的にシミュレーション空間を事前に定めておかなければならない点，そしてその空間が極めて狭くなってしまう点は未だに解決されていない．

2.1.2 粒子法

CG における流体アニメーションを最初に発表したのは Reeves (Reeves, 1983) である．この研究では，炎の生成から消滅までを単純化し，80 万程度の粒子を用いてそれらしく見せることに成功している．その後は，上述の Chiba (Chiba et al., 1994)，Stam (Stam and Fiume, 1995) のように，格子シミュレーションの結果を表現するために粒子を用いられることが多く，物理法則を限定的に考慮した，アニメーションとしての研究が進められた．

Koshizuka (Koshizuka and Oka, 1996) は，数値流体力学分野で，MPS（Moving-Particle Semi-Implicit）を開発した．格子法で計算が難しかったダム崩壊問題のような，飛沫のシミュレーションも可能とし，精度の高い数値解析を可能とした．この MPS は Premoze (Premoze et al., 2003) によって CG に導入され，高品質な画像を生成できることが示された．MPS は飛沫表現が格子法よりも容易であるが，圧力のポアソン方程式を解く必要があり，計算量のかかる手法であった．竹下 (Takeshita et al., 2003)， (竹下 他, 2004) は，MPS を参考にし，粒子間の相互作用を計算することで爆発火炎のシミュレーションを行った．

一方，宇宙物理学分野では，Lucy (Lucy, 1977), Gingold (Gingold and Monaghan, 1977) によって，SPH （Smoothed Particle Hydrodynamics）が開発された．Desbrun (Desbrun and Gascuel, 1995) は SPH を CG に導入し，分子間力を模擬することで液体らしい表現が可能であることを示した．Müller (Müller et al., 2003) は SPH をナビエ・ストークス方程式に適

用し，対話的な流体シミュレーションを実現した. この方法では，粒子間距離が一定になれば非圧縮性が自動的に実現されるとして，圧力のポアソン方程式を解かない．これにより，体積保存は完全でないが，視覚的には妥当なシミュレーションが，高速に実現された．このSPH はコップ程度の小規模なシミュレーションであったが，Kipfer (Kipfer and Westermann, 2006) は，GPU を部分的に活用し，滝や川のシミュレーションが対話的に実行可能であることを示した.

近傍探索の問題から，SPH を完全に GPU 内で完結させることは困難であったが，近傍に存在する粒子数の上限を設定しても視覚的な問題は起こらないことを Harada (Harada et al., 2007b) が示し，速度を CPU の数十倍に引き上げた．これにより，数万単位の粒子を対話的にシミュレーションすることが可能となったが，CPU の方が柔軟な計算が可能であるため，CPU による SPH の研究も続けられてきた．Adams (Adams et al., 2007) は，異なるサイズの粒子を用いて，粒子をリサンプリングすることで，計算を効率化できることを示した．また，Solenthaler (Solenthaler et al., 2007) は，固体や弾性体を含む，統一的な物理シミュレーションが可能であること示した．SPH は上述の通り，圧力のポアソン方程式を解かないため，体積保存が一時的に不十分になるという欠点があったが，Becker (Becker and Teschner, 2007) は，多少の計算量増加と引き換えに，この問題を緩和することが可能であることを示し，Solenthaler (Solenthaler and Pajarola, 2009) は，非圧縮性の再現が可能であることを示した.

しかしながら，近傍が最初から定義される格子法と違い，粒子法では，近傍の粒子を高速に探索する必要がある．元々は，空間を均一に区切り，その近傍空間内の格子を探索する手法が用いられてきたが，これでは，格子法と同様に，一定範囲の空間しか扱えなかった．しかし，ハッシュを用いることで，大規模空間においても近傍探索が行えることを，Teschner (Teschner et al., 2003) が示し，また，ソートを用いることでも同様に大規模空間を扱えることを，Green (Green, 2008) が示した．また，GPU ではデータ構造に自由度が少なかったが，Harada (Harada et al., 2007a) は，GPU 上での大規模空間の近傍探索に適したデータ構造を提案した．これらにより，粒子法においては，極めて大規模な空間を扱えることとなった.

MPS に関してのシミュレーションは文献 (越塚, 2005) に，SPH に関してのシミュレーションは文献 (越塚, 2008) に包括的にまとめられている.

このように近年では，粒子法の発展が目覚ましいが，液体シミュレーションが主流となっており，気体や火炎への適用例は極めて少ない.

2.2 レンダリング

本節では，火炎のレンダリングに焦点をあてて関連研究を述べる．主には，レイマーチングとビルボードに分けられる．これらに加え，本論文で重要となる，粒子ベースボリュームレンダリングについても述べる．

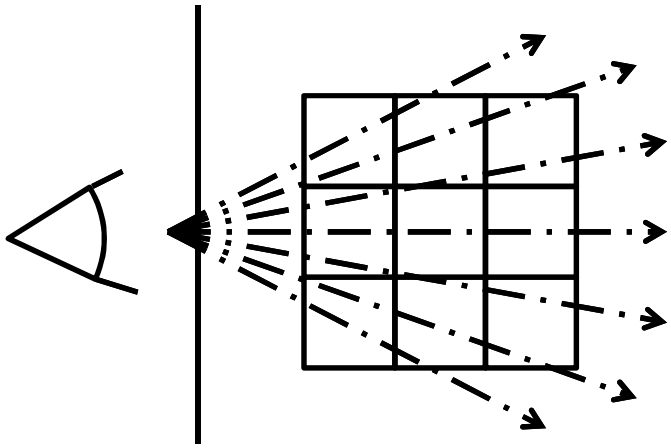


図 2.2: レイマーチング

視点から光線を発生させ，各サンプリング点ごとに評価された色を加算して，各画素の色を求める．

2.2.1 レイマーチング

格子で定義されたボリュームデータのレンダリングは，Blinn と Kajiya によって始められた．Blinn (Blinn, 1982) は，放射理論を単純化し，Kajiya (Kajiya and Von Herzen, 1984) は，媒体の散乱の度合いを加味することによって，主に雲の表現を行った．ただし，これらは単純化の過程で，表現対象に大きく制限が加えられていた．

汎用性を高めるために，解析的なアプローチでなく，数値解析的なアプローチとして，レイマーチング法が広く用いられる．これは，視点から光線を発生させ，サンプリング点ごとに評価された色を加算して，各画素の色を求める方法である．レイマーチングの概要を**図 2.2** に示す．Fedkiw (Fedkiw et al., 2001) は，煙のシミュレーションデータに対してレイマーチングを利用している．Nguyen (Nguyen et al., 2002) では，火炎のレンダリングに再帰的なレイマーチングを用いている．煙も同様に扱うことができ，現在提案されている手法では，最も精度も高いが，計算コストも高い．Crane (Crane et al., 2007) は，通常のレイマーチングで，視覚的には充分な火炎表現が高速に行えることを示した．しかし，再帰的な計算を行わないため，煙の表現を除外している．

レイマーチングは，画素ごとに，光線を発生させることができ，さらに格子セル上の値を補間するため，高精度な画像を得ることができる．しかし，定義されていない格子の外をレンダリングすることはできない．

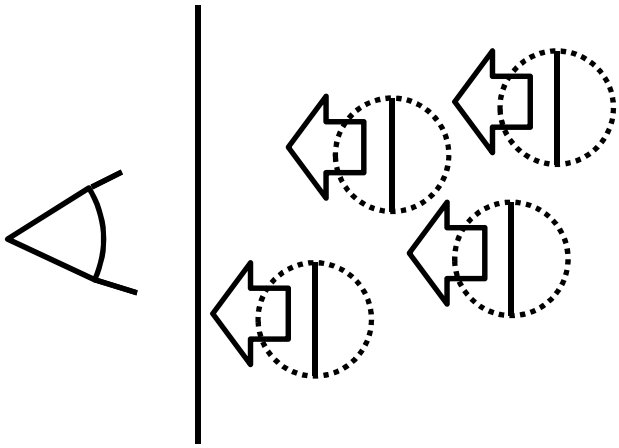


図 2.3: ビルボードレンダリング

描画対象の構成要素を，スクリーンに対して平行な四角形とし，それを重ね合わせることで画像を得る．

2.2.2 ビルボードレンダリング

粒子で定義されたボリュームデータの表現には，ビルボードレンダリングが広く用いられている．ビルボードレンダリングとは，スクリーンに対して平行な，テクスチャを貼付した四角形（ビルボード）を用意し，それを重ね合わせてレンダリングする手法である．ビルボードレンダリングの概要を**図 2.3** に示す．

このビルボードレンダリングは，厳密な手法ではなく，テクスチャの質に大きく依存するが，比較的高品質な画像を高速に作り出せることが示されている．Dobashi (Dobashi et al., 2000) は，ビルボードを用い，簡易的な散乱を考慮した，高速な雲のレンダリング法を提案した．Dobashi (Dobashi et al., 2004) は，これを煙に適用し，火炎を含むシーン全体のリアリティの向上を実現しているが，火炎自体の表現を目的としたものではない．井村 (井村他, 2010) はビルボードに屈折を適用し，火炎表現を行っているが，これも火炎自体のレンダリングを目的としたものではない．火炎自体のレンダリングは， (Nguyen, 2004) において効果的にデモンストレーションされているが，これもテクスチャの質に大きく依存する．

火炎のシミュレーション結果のレンダリングには，スライスデータを用いることもある．これは，10 枚程度のテクスチャを深度順に直接描画していく方法である．これもビルボー

ド応用の一種と考えることができる．Horvath (Horvath and Geiger, 2009) は，シミュレーション結果を描画解像度に合わせることでこれを実現している．Fuller (Fuller et al., 2007) も同様のレンダリング手法をとっている．レンダリングの精度も速度も上がるが，多くのビデオメモリが必要となる．

ビルボードレンダリングは，描画対象ごとに，スクリーンに投影していくため，高速なレンダリングが可能である．画素毎に描画する手法ではないため，レイマーチングのように高精度な画像を得ることが難しいが，格子を必要としないので，レンダリング対象の空間に制限がない．

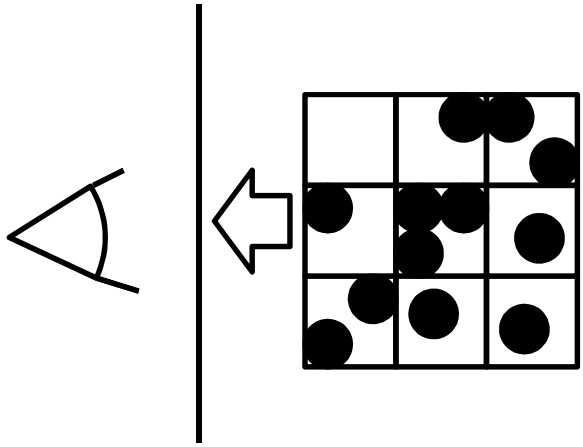


図 2.4: 粒子ベースボリュームレンダリング

各格子から描画用粒子を発生させ，それを繰り返し投影し，平均をとることで各画素の色を求める．

2.2.3 粒子ベースボリュームレンダリング

近年，可視化分野において，Sakamoto (Sakamoto et al., 2007) によって粒子ベースボリュームレンダリングが開発された．この粒子ベースボリュームレンダリングは，不透明度に応じて，確率的に粒子をばらまき，投影を繰り返すという方法をとっている．粒子ベースボリュームレンダリングの概要を**図 2.4** に示す．

Zhongming (Zhongming et al., 2010) は，GPGPU を用いて，より大規模なデータを取り扱えることを示した．包括的な資料としては，文献 (小山田, 坂本, 2010) が挙げられる．これらの研究では，特に精度が重要であり，ノイズを抑えるために，大量の粒子を事前生成しておくことを前提としている．現在のところ，粒子ベースボリュームレンダリングは，大規模データの可視化を主としており，対話的な CG を目的とした粒子ボリュームレンダリングは報告されていない．

粒子ベースボリュームレンダリングは，ビルボードレンダリングのように，スクリーンに投影していくため，高速なレンダリングが可能であり，原理的には格子を用いる必要がない．また，投影の繰り返し回数によってレンダリングの質を容易に調整することができる．しかし，相当数の繰り返しを行わないと，ノイズが発生してしまう．通常の可視化においては，ノイズは好ましくないが，火炎のビジュアルシミュレーションの点からとらえると，火炎は人間の視覚系がはっきりととらえることは困難であるため，多少のノイズが発生することはむしろ好都合であると考えられる．

第3章　火炎のシミュレーション

前章で述べたとおり，これまでの火炎の対話的シミュレーションには，格子法，粒子法ともに問題がある．本章では，SPH をもとにした，新たな火炎シミュレーション法を提案する．なお，本章の内容は文献 (間淵 他, 2011a) をもとにしている．

3.1　従来手法の問題点

従来手法の問題点をまとめると，火炎の物理ベースシミュレーションは格子法で発達しているが，想定するシミュレーション空間を事前に定義しておく必要がある．また，対話性を要求される場合には，格子セル数を大きく減らす必要があり，空間サイズにも強い制限が加わる．粒子法においては，火炎の物理ベースシミュレーション法がそもそも発達していない．しかし，シミュレーション空間を事前に想定する必要がなく，また空間サイズへの制限もほとんどない．

3.2　アプローチ

火炎の性質上，形状が動的に，頻繁に変化するため，シミュレーション空間を事前に想定したり，空間への制限がないことが望ましい．従って，本研究では物理ベースの粒子法である SPH を用いる．これによる長所は，次の通りである．

3.2.1　物理的妥当性

前章で述べたとおり，火炎らしさは流体力学に従うか否かで大きく左右される．本研究では，流体力学のシミュレーションを SPH に担わせる．そのため，SPH が保証するものと同等の力学的妥当性を提供することができる．これは同時に，SPH プログラムを保持していれば，拡張するだけで本手法を再現できる，というソフトウェア管理上の利点にもつながる．

ただし，化学的な妥当性を保証することは困難である．これは，燃焼の化学反応が極めて複雑なプロセスであり，ある程度忠実なシミュレーションを行うと，非直感的な調整パラメータが増大してしまうためである．

3.2.2　対話性

火炎のシミュレーションにおいて，対話性は，次の 2 つの側面に分けて考えることができる．

- 計算速度が高速であること
- 扱える空間が大規模であること

これらは，前章で述べたとおり，SPH の特性と一致する．

3.3　火炎に関する基礎知識

火炎は，気体中で，燃焼反応が起こることによって生成される．したがって，流体力学と，化学反応の面から，それぞれ本論文の理解に必要な点を説明する．

3.3.1　流体力学の面から

流体の運動はナビエ・ストークス方程式によって定められる．運動量保存則は，

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{v} + \mathbf{F} \tag{3.1}$$

であり，ここで $\mathbf{v}$ は速度ベクトル，t は時間，ρ は密度，ν は粘性係数，$\mathbf{F}$ は外力ベクトルである．この方程式の左辺第 1 項は時間微分項，第 2 項は対流項であり，右辺第 1 項は圧力項，第 2 項は粘性項，第 3 項は外力項である．粒子法では，粒子の動きそのものが対流項に相当するため，対流項が省略され，

$$\frac{\partial \mathbf{v}}{\partial t} = \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{v} + \mathbf{F} \tag{3.2}$$

と簡略化される．これは，圧力項と粘性項を解けば流体の動きが再現されることを示している．

また，非圧縮性流体の質量保存則は，

$$\nabla \cdot \mathbf{v} = 0 \tag{3.3}$$

であるが，これは，各粒子の質量が一定とみなした場合，各粒子間距離が一定になることと同じである．SPH では，一時的な多少の体積変動を受け入れることで，この質量保存則を解く必要がなくなる．

3.3.2 化学反応の面から

燃焼には，

- 気体燃料の燃焼．代表的にはバーナー火炎
- 液体燃料の燃焼．身近にはないが，工業的に頻繁に利用
- 固体燃料の燃焼．代表的にはろうそくの燃焼

の 3 種類が主に存在する．このうち，液体燃料の燃焼は身近に観察されることが少ない．固体燃料の燃焼は，一般に燃料が気化して反応する．これらのことから，本論文では気体燃焼を前提とする．

気体燃焼はさらに，

- 拡散燃焼．燃料と酸化剤が別々に供給される燃焼．空気を供給しないプレゼンバーナーが代表的
- 予混合燃焼．燃料と酸化剤が予め混合されて供給される燃焼．空気を供給するプレゼンバーナーが代表的

とに分けられる．このうち，拡散燃焼は燃料と酸化剤の混合を考慮しなければならないのに比べ，予混合燃焼は扱いが容易である．また，視覚的には予混合燃焼と拡散燃焼の区別はつきにくい．以上より，本論文では，予混合燃焼を前提とする．

3.4 SPH

SPH における離散化の基本式を解説する. 任意の座標 $\mathbf{r}$ における物理量 A は，

$$A(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) \tag{3.4}$$

と計算される．ここで，A_j は粒子物理量，m_j は粒子質量，ρ_j は粒子密度，$\mathbf{r}_j$ は粒子座標，h は粒子間影響距離，$W(\mathbf{r}, h)$ はスムーシングカーネルである．特に，微分してもスムーシ

ングカーネルを微分すればよいという性質があるため，勾配やラプラシアンも容易に定められる．すなわち，勾配は，

$$\nabla A(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla W(\mathbf{r} - \mathbf{r}_j, h) \tag{3.5}$$

となり，ラプラシアンは，

$$\nabla^2 A(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r} - \mathbf{r}_j, h) \tag{3.6}$$

となる．

本研究では，文献 (Müller et al., 2003) と同様のスムーシングカーネルを用いた．すなわち，

$$W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & (0 \le r \le h) \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

$$\nabla W_{spiky}(r, h) = \frac{45}{\pi h^6} \begin{cases} (h - r)^2 \frac{\mathbf{r}}{r} & (0 \le r \le h) \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{3.8}$$

$$\nabla^2 W_{viscosity}(r, h) = \frac{45}{\pi h^6} \begin{cases} (h - r) & (0 \le r \le h) \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

である．

これらを用いて，式（3.2) を解いていく．

まず粒子 i の座標 $\mathbf{r}_i$ における密度を計算する．

$$\rho(\mathbf{r}_i) = \sum_j m_j W_{poly6}(\mathbf{r}_j - \mathbf{r}_i, h) \tag{3.10}$$

求められた密度と安定密度の差から圧力 p_i が求められる．

$$p_i = k_{stiffness}(\rho_i - \rho_{rest}) \tag{3.11}$$

ここで，$k_{stiffness}$ は圧力係数，ρ_{rest} は安定密度である．

圧力項は式（3.5) から導かれ，

$$\mathbf{f}_{pressure} = \frac{m_j}{\rho_j}(p_j - p_i)\nabla W_{spiky}(r_j - r_i) \tag{3.12}$$

となる．

粘性項も式（3.4) から導かれ，

$$\mathbf{f}_{viscosity} = k_{viscosity} \frac{m_j}{\rho_j}(\mathbf{v}_j - \mathbf{v}_i)\nabla^2 W_{viscosity}(r_j - r_i) \tag{3.13}$$

となる．ここで，$k_{viscosity}$ は粘性項係数である．

以上より得られた力を時間積分することで，粒子の動きが得られる．数値積分法には様々なものが存在するが，Harada (Harada et al., 2007b) によると，最も単純なオイラー法で，視覚的に充分な精度が得られると報告されている．本研究でもそれに従う．すなわち，

$$\mathbf{v}_{new} = \frac{\Delta t \mathbf{f}}{\rho} + \mathbf{v}_{old} \tag{3.14}$$

$$\mathbf{r}_{new} = \Delta t \mathbf{v} + \mathbf{r}_{old} \tag{3.15}$$

である．

SPH の擬似コードを**アルゴリズム 3.1** に示す．実際には，近傍粒子の探索が計算上ボトルネックとなるため，最初に近傍粒子ペアを作っておくことが望ましい．その他は，上述の各式のとおりに計算を行えばよい．

3.5　SPH による火炎のシミュレーション

提案する，SPH をベースとして用いた，火炎シミュレーションの概要を**図 3.1** に示す．化学反応による熱の発生を計算し，そこから発生した熱を SPH 相互作用により生じた力に，浮力項として加えることとした．すなわち，この浮力項をゼロとすれば,（表面張力を除いた）流体シミュレーションと同一になる．

3.5.1　仮定

前述のとおり，本研究では従来の SPH (Müller et al., 2003) を拡張した形で扱うが，燃焼は複雑な物理・化学現象であり，これを厳密にシミュレーションする場合は，調整パラメータが増大してしまう．そこで，本研究では次の仮定をおいた．

- 生成された熱エネルギーは直接温度に反映されるものとする．これは，比熱や熱容量を考慮すると，非直感的なパラメータが増大してしまうため
- 酸素は常に供給されるものとする．これは，燃料の多寡のみで火炎の規模を調整できるようにするため
- 燃料は気体燃料とする．実際には，固体燃料であっても，気化されて，気体燃料となって反応が起こるため

本章では，上記の仮定を離散化し，実現するモデルについて述べる．なお，k_* と表記しているパラメータは，実際には一度定めてしまえば変更の必要がほとんどないものである．

アルゴリズム 3.1: SPH による流体シミュレーション

```
for all particles do
    createParticlePairs()
end for
for all particlePairs do
    calculateDensity()   // 式 (3.10)
end for
for all particles do
    calculatePressure()   // 式 (3.11)
end for
for all particlePairs do
    calculatePressureForce()   // 式 (3.12)
    calculateViscosityForce()   // 式 (3.13)
end for
for all particles do
    updateVelocity()   //   式 (3.14)
    updatePosition()   //   式 (3.15)
end for
```

3.5.2 浮力項の導入

ナビエ・ストークス方程式の離散化には，SPH を用いる. また，拡張部の離散化も SPH に沿って行う.

原理的には，温度に比例して周囲より密度が下がり，浮力が発生するのであるが，極端な密度差に対応することは粒子法，格子法ともに難しい. そのため，本研究では外力として温度に応じた浮力項を，ナビエ・ストークス方程式に加えることとした. すなわち，

$$\mathbf{f}_{buo} = k_{buo}\,(T - T_{amb})\,\mathbf{z} \tag{3.16}$$

とする．ここで，k_{buo} は浮力係数であり，$\mathbf{z}$ は上方向単位ベクトル，$\mathbf{f}_{buo}$ は浮力項ベクトルである. また，T_{amb} はシミュレーション環境の温度であり，通常は 300[K] 程度の定数である.

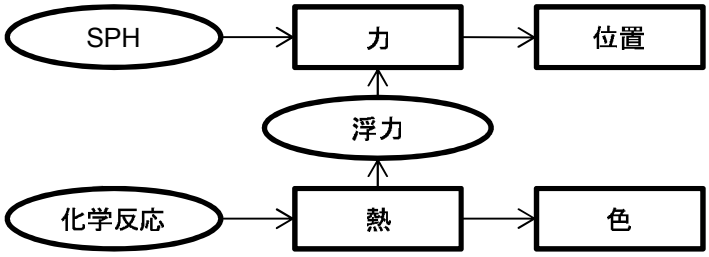


図 3.1: シミュレーション概要

従来の粒子法にシンプルな化学反応モデルを追加し，浮力と熱を生成する．

3.5.3 火炎粒子と化学反応

本研究では扱う対象が火炎を含む気体であるため，各粒子に温度 T，燃料 F を追加する．次の条件を満たした粒子を火炎粒子とする．

- 温度が一定値 T_{ign} より高い
- 燃料がゼロでない

以上で定義された火炎粒子は，化学反応を行う．本研究では，火炎粒子が行う化学反応を次のようにモデル化する．化学反応速度 S を，

$$S = k_{rs} T F \tag{3.17}$$

で定義する. ここで k_{rs} は化学反応速度係数を表す. 反応速度に応じた燃料を消費するため,

$$\frac{\partial F}{\partial t} = -k_{cf} S \tag{3.18}$$

とする. ここで， k_{cf} は燃料消費係数を表す. また， 反応速度に応じて熱を発生するため,

$$\frac{\partial T}{\partial t} = k_{pt} S \tag{3.19}$$

とする. ここで， k_{pt} は温度生成係数を表す.

3.5.4 拡散

熱拡散の取り扱いには文献 (Müller et al., 2005) のモデルを用いる．すなわち，k_h を熱伝導係数として,

$$\frac{\partial T}{\partial t} = k_h \nabla^2 T$$

とモデル化し，これを離散化して，

$$\frac{\partial T_i}{\partial t} = k_h \sum_j \frac{m_j}{\rho_j}(T_j - T_i)\nabla^2 W_{viscosity}(\mathbf{r}_i - \mathbf{r}_j, h) \tag{3.20}$$

とする．

温度が高い状態ということは，燃料を近傍に放出しやすい状態と考えることができるため，燃料拡散係数を k_f として，

$$\frac{\partial F}{\partial t} = k_f T \nabla^2 F$$

とモデル化し，これを離散化して，

$$\frac{\partial F_i}{\partial t} = k_f \sum_j \frac{(T_j + T_i)}{2}(F_j - F_i)\frac{m_j}{\rho_j}\nabla^2 W_{viscosity}(\mathbf{r}_i - \mathbf{r}_j, h) \tag{3.21}$$

とする．

3.6 ビルボードレンダリング

本章では，ビルボード上に，アルファテクスチャを貼りつけたものを用いて簡易的なレンダリングを行う（以下，ビルボードレンダリングとよぶ）．このアルファテクスチャは，滑らかに変化するものが望ましい．本論文では，SPH のカーネルを用いて作成した．これを**図 3.2** に示す．

色の決定には，本研究で扱う対象は火炎であるため，主に発光を考慮すればよい．この色は黒体放射モデルを用いた文献 (Nguyen et al., 2002) と同様の方法で定めた．この論文はレイマーチングを用いているため，各光線の色へ適用しているが，本研究では各描画粒子の色へ適用する．各粒子はこの色伝達関数のテーブルを用いて温度から RGB 値を決定する．この手法を用いて，3,000[K] を最大温度とした色伝達関数を**図 3.3** に示す．**図 3.4** に全粒子を表示した場合と，火炎粒子を表示した場合，それに火炎粒子にビルボードレンダリングを用いた場合の比較を示す．

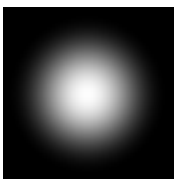
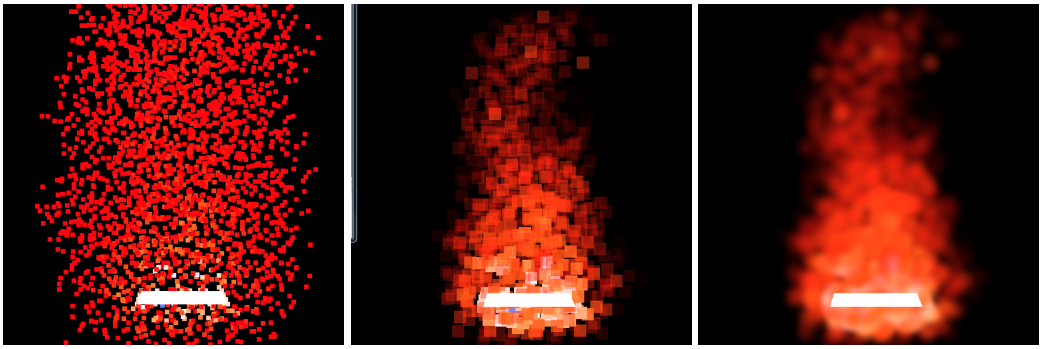


図 3.2: 各ビルボードに貼りつけるアルファテクスチャ

図 3.3: 粒子の温度に対する色を決定する伝達関数

3,000[K] を白色と仮定しており，低温になるにしたがって赤成分が多くなる．



(a) 全粒子　(b) 火炎粒子　(c) ビルボードレンダリング

図 3.4: 全粒子を表示した場合と，火炎粒子を表示した場合，それに火炎粒子にビルボードレンダリングを用いた場合のレンダリング比較

3.7 CPU 実装

本研究では計算コストにも注意を払うため，効率的かつ効果的な実装を提示する．

シミュレーションの擬似コードを**アルゴリズム 3.2** に示す．最初に従来の SPH と同様の計算を行ったのち，熱拡散，燃料拡散計算を行う．データ構造には，従来の SPH 粒子データ構造に対して，温度と燃料の追加が必要となる．本手法は，従来の SPH のプログラムに追加するだけで済む．

3.7.1 近傍探索

近傍粒子探索においては，空間を均一セルに分割する手法 (Teschner et al., 2003) か，k-d 木を用いた手法 (Adams et al., 2007) が主に用いられる．

本研究の性質上，対話性を保持するために，比較的少ない粒子で，広大な空間を扱うことが必要となる．このため，Green (Green, 2008) が提案した，空間把握の役割を粒子に転嫁した，近傍探索手法を用いることにした．この手法は特に対話的な速度を求められる場合に適している．

3.7.2　リサンプリング

液体と違い，火炎のシミュレーションでは火炎の付近に粒子が配置されていなければ相互作用計算ができない．空間に粒子が充填されていれば高い精度での計算が可能となるが，大量の粒子はメモリ消費が激しく，また相互作用計算が増加するため，計算効率が低下してしまう．本研究では粒子の追加と削除によるリサンプリングを開発した．なお，同一ステップ内で粒子が増減すると条件分岐が増えてしまうため，追加，削除される粒子へのポインタを保持しておき，次のステップの先頭部で追加，削除する．リサンプリングは毎ステップ行うより，一定ステップ毎にまとめて行う方が効率的である．本論文では 5 ステップ毎に行った．リサンプリングの仕組みを**図 3.5** に示す．

粒子の追加では，火炎粒子の近傍に粒子を配置する必要があるが，単に周囲に粒子を追加したのでは既に存在している粒子と重複が発生してしまう．火炎粒子の近傍に，重複が発生しないように粒子を配置するために，本研究では法線を用いた．法線は既存 SPH (Müller et al., 2003) の，表面張力を求める際に用いられる法線をそのまま用いることができる．火炎粒子の法線方向に粒子直径分移動させた座標に新しい粒子を配置する．この新しい粒子は燃料をゼロ，温度を環境温度として，300[K] に設定する．内部の火炎粒子には粒子の追加が不要であるため，近傍粒子数が一定未満の粒子の周囲に追加を行う．これにより，衝突判定を行うことなく，高速に火炎近傍に粒子を充填することができる．

粒子の削除では，近傍に火炎粒子が存在しない粒子を削除対象とする．具体的には近傍粒子ペア作成時に近傍火炎粒子をカウントしておき，ゼロであればその粒子を削除することとする．

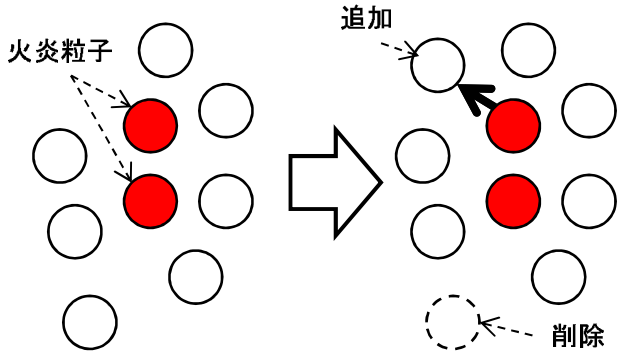


図 3.5: リサンプリングの仕組み

各火炎粒子の法線方向の近傍に新しい粒子を生成する．各粒子の近傍に火炎粒子が存在していなければ，その粒子を削除する．

3.7.3　固体粒子

シミュレーションの開始には火炎の燃焼源となる固体粒子が必要である．固体粒子は他の粒子と同様に計算するが，時間積分を行わないことによって自動的に速度，座標，燃料，温度が一定に保たれる．固体粒子の初期燃料 F_{init} と初期温度 T_{init} を燃焼可能値以上にしておけば，自動的に周囲にリサンプリングが行われ，火炎の生成源として扱うことが可能となる．この固体粒子から周囲の粒子に燃料と温度が拡散されるため，連鎖的に，かつ継続的に化学反応が行われることとなる．また，燃焼可能値未満の固体粒子を用意しておき，その固体粒子の速度をユーザに指定させれば，火炎に固体からの一方向の作用を加えることが可能となる．なお，火炎粒子と固体粒子は，属性であり，粒子データ構造そのものは同一である．

3.7.4　時間積分

新たに追加した，燃料と温度の時間積分には，粒子の動きと同じくオイラー法を用いる．すなわち，

$$T_{new} = \Delta t \Delta T + T_{old} \tag{3.22}$$

$$F_{new} = \Delta t \Delta F + F_{old} \tag{3.23}$$

とする．

3.7.5　パラメータ

本手法のパラメータは，既存 SPH のパラメータ，固定できるパラメータ，対話的に調整するパラメータに分類できる．

実際に用いたシミュレーションのパラメータ一覧を**表 3.1** に示す．既存 SPH のパラメータは物理的な根拠のある数値であり，単位も明確化されている (Müller et al., 2005)．また，本章で追加した固定パラメータはシミュレーション毎に変更する必要がほとんどない．ただし，物理的な根拠のある数値ではないため，現実の火炎と比べて，シミュレーション時間と実時間とが一致することや，シミュレーションスケールが実サイズと一致することは保障しない．実際に対話的に調整する必要のあるパラメータは，温度と燃料の 2 変数を調整するだけで火炎を調整できるが，このうち，燃焼源の温度は日常的に目にする火炎であれば大差がないため，実際には定数とみなしてもよい．つまり燃料の 1 変数を調整するだけで調整が可能である．

パラメータが明記されている関連研究と比較すると，Feldman の手法 (Feldman et al., 2003) では 11 個のパラメータを，竹下の手法 (竹下 他, 2004) では 9 個のパラメータをシミュレーション毎に調整する必要がある．これらと比べると，本手法は調整するパラメータが充分に少ないといえる．

表 3.1: パラメータ一覧

パラメータの種類	表記	単位	用いた値
既存 SPH パラメータ	ΔT	$[sec]$	0.001
	h	$[m]$	0.01125
	k_{stiff}	$[Nm/kg]$	10.0
	$k_{viscosity}$	$[Ns/kg^2]$	0.005
	ρ_{rest}	$[kg/m^3]$	1.0
固定パラメータ	T_{amb}	$[K]$	300.0
	T_{ign}	$[K]$	1000.0
	k_{rs}	-	0.01
	k_{cf}	-	5.0
	k_{pt}	-	500.0
	k_h	-	0.02
	k_f	-	0.001
対話的パラメータ	T_{init}	$[K]$	2000.0
	F_{init}	-	80.0-120.0

3.8 結果

本章の結果は CPU に Intel Core2 2.40GHz，主メモリに 2.0GB，実装に C++，グラフィクス API に OpenGL1.5 を用いて実装したものである．火炎の形成から安定状態への推移までを**図 3.6** に，安定状態を**図 3.7** に示す．これらは，中心部が白色に近く，外側ほど赤みを帯びる点，火炎が上へと向かおうとする点がどちらも表現されている．

外力を加えた例を**図 3.8** に示す．強い外力を受けている際には火炎が小さくなっている点が特徴的である．これは外力により熱が逃げているためと考えられる．また，固体からの作用を受けるシミュレーションを**図 3.9** に示す．図中の白板が固体である．火炎の形状が

固体からの片方向の作用を受けて，変化していく様子が確認される．これらの表現は乱数とライフタイム（粒子が存在する期間）を用いる従来の粒子法 (Reeves, 1983) では困難であり，物理ベースならではのシミュレーションであると考えられる．これらのシミュレーションは粒子数 2,500 前後で安定し，各ステップ 0.015〜0.025[sec] で実行した．

粒子数と計算速度の遷移の例を，**図 3.10** に示す．粒子数は，シミュレーション開始から 200 ステップあたりまで，2,300 個程度まで急激に増加し，以後緩やかに粒子が増えた後，1,700〜2,700 個の間で推移していることが確認できる．計算速度は粒子数にほぼ比例していた．

応用例として，火炎のマージを**図 3.11** に示す．2 つの火炎がマージされ，1 つの大きな火炎となっていることが確認できる．格子法と違い，事前にシミュレーション空間を定義する必要がないため，動的な変化への対応が容易である．また，従来の粒子法と違い，ランダムに動くのではなく，自然に一体化されていく様子が確認できる．このシミュレーションは粒子数 5,000 個前後で安定し，各ステップ約 0.05[sec] で実行した．

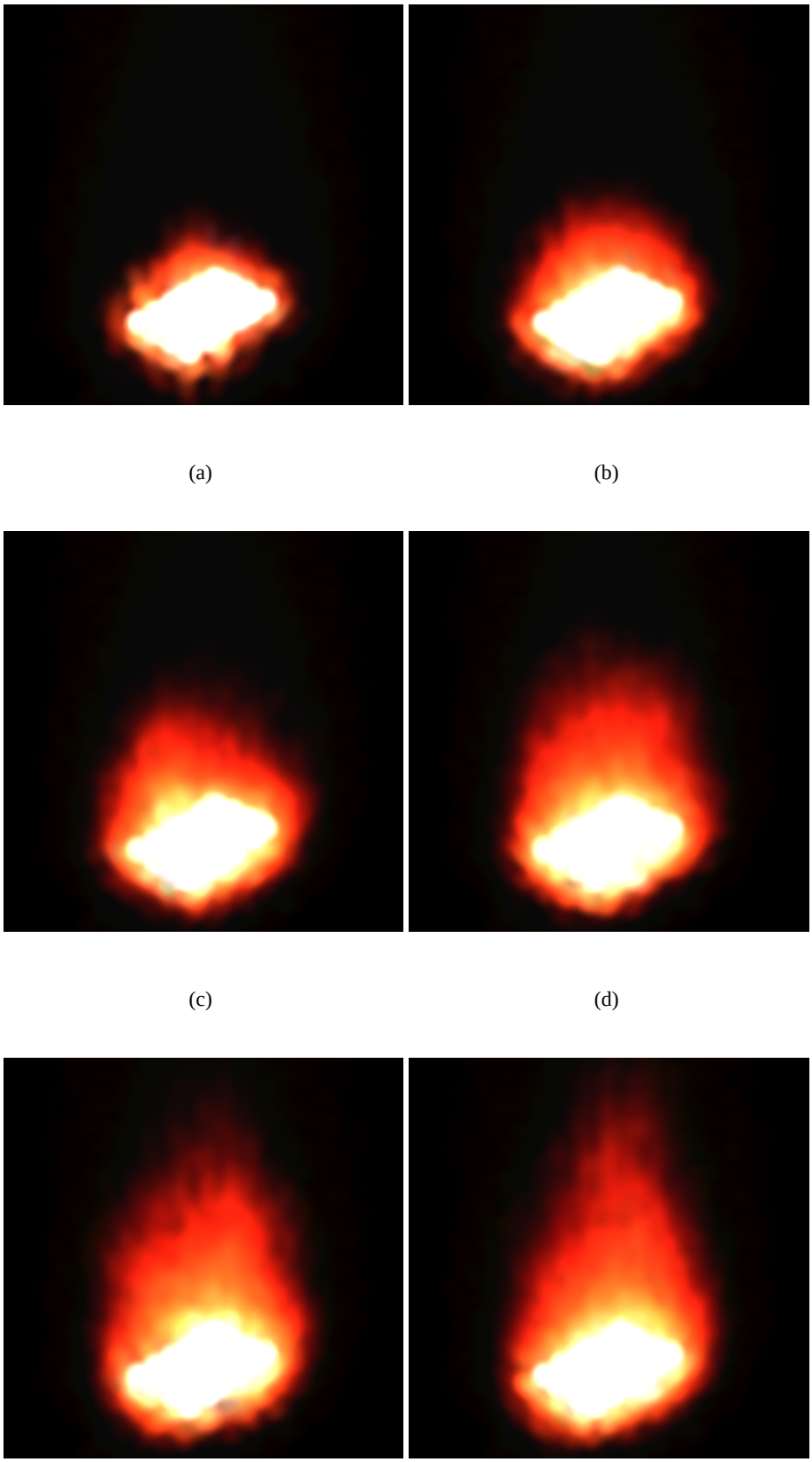
3.9　まとめと考察

本章では，SPH を拡張することによって，対話的な火炎のシミュレーションが可能であることを示した．SPH は本来流体全般に対して用いられる手法であることから，本章の成果により，SPH は火炎に限らず，煙や雲など気体上での物理現象の表現一般に用いることが可能である．また本章の手法では，大半の調整パラメータは固定することができる．実際に調整する必要があるものは，温度と燃料だけであり，直感的である．本章では，単純な化学反応モデルを用いても，火炎らしさの表現は可能であることを示した．

しかしながら，レンダリングの面から判断すると，粒状の印象が発生してしまっている．この原因の 1 つは，粒子数が数千個程度と少ないためである．本手法には特に粒子数の制限がないが，対話性を考えると，数万個程度の粒子を単一の CPU で扱うことは難しい．そこで次章で，少ない粒子からでも，高精度な画像を得られるレンダリング法を提案し，第 5 章では GPGPU を用い，より多くの粒子を対話的に用いる手法を提案する．

アルゴリズム 3.2: SPH ベース火炎シミュレーション

```
// アルゴリズム 3.1 に追加した箇所を斜体とした.
if isResamplingTimeStep then
   doResampling()
end if
for all particles do
   createParticlePairs()
end for
for all particlePairs do
   calculateDensity()  // 式 (3.10)
end for
for all particles do
   calculatePressure()  // 式 (3.11)
end for
for all particles do
   calculatePressureForce()  // 式 (3.12)
   calculateViscosityForce()  // 式 (3.13)
   calculateHeatDiffusion()  // 式 (3.20)
   calculateFuelDiffusion()  // 式 (3.21)
end for
for all particles do
   calculateChemicalReaction()  // 式 (3.18), 式 (3.19)
   updateTemperature()  // 式 (3.22)
   updateFuel()  // 式 (3.23)
   addBuoyancyForce()  // 式 (3.16)
   updateVelocity()  // 式 (3.14)
   updatePosition()  // 式 (3.15)
end for
```



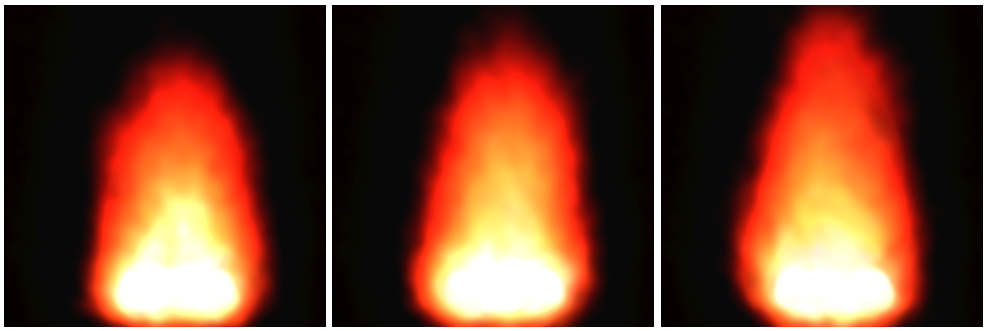
(a) (b)

(c) (d)

(e) (f)

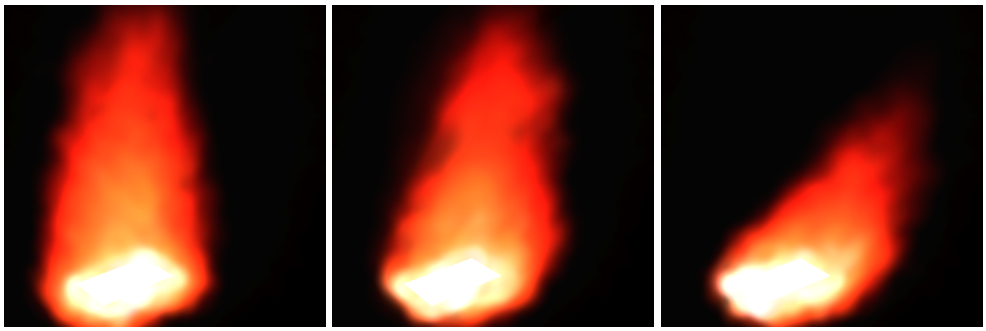
図 3.6: 火炎の形成のシミュレーション結果

火炎の形成から安定状態への推移までが自動的に実現される.



(a) (b) (c)

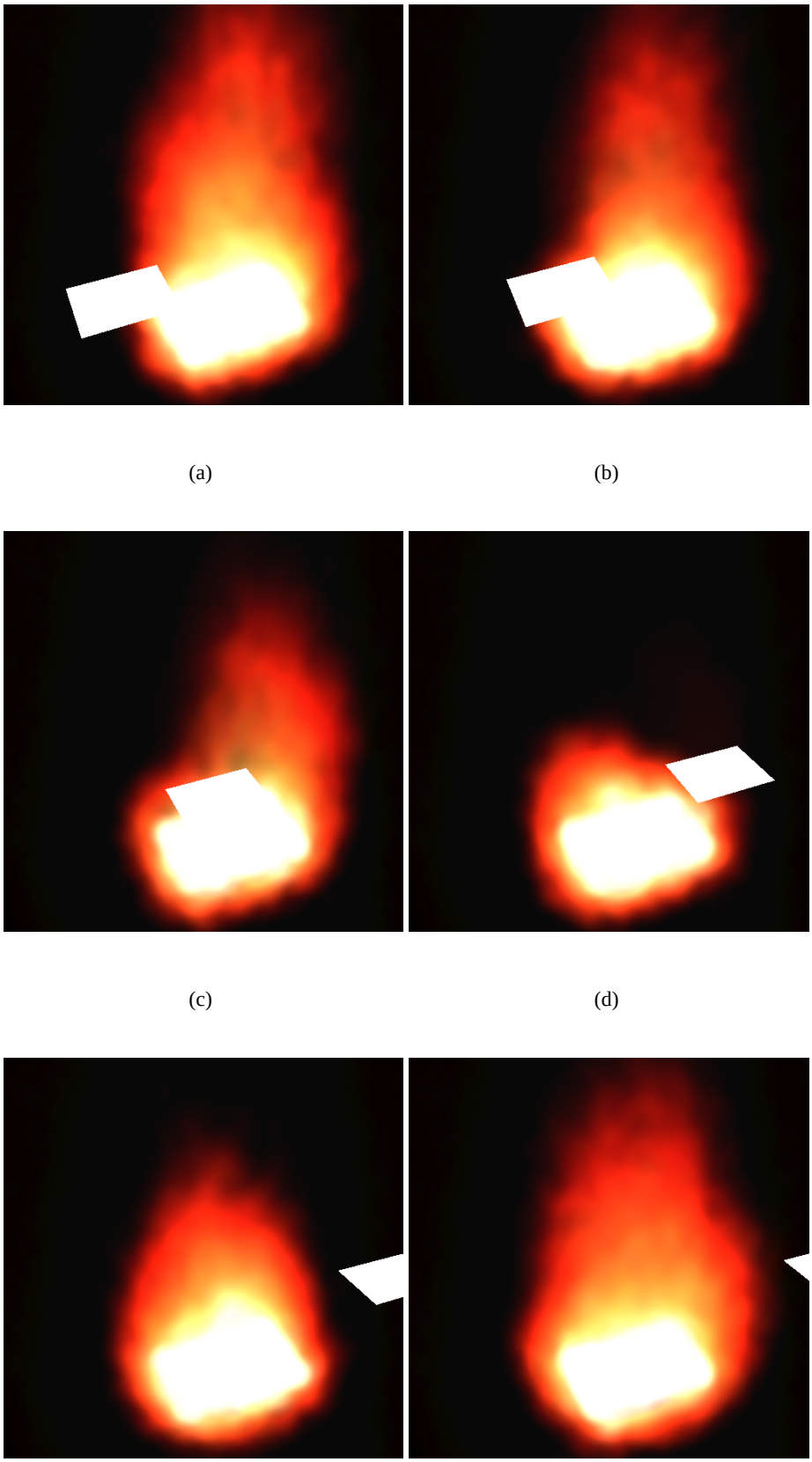
図 3.7: 安定状態の火炎のシミュレーション結果



(a) (b) (c)

図 3.8: 外力による変形のシミュレーション結果

強い外力を受けている間，火炎はなびき，小さくなる．



(a) (b)

(c) (d)

(e) (f)

図 3.9: 固体からの作用のシミュレーション結果

固体が火炎を通過している間，火炎は小さくなり，通過後に再び火炎が大きくなる．

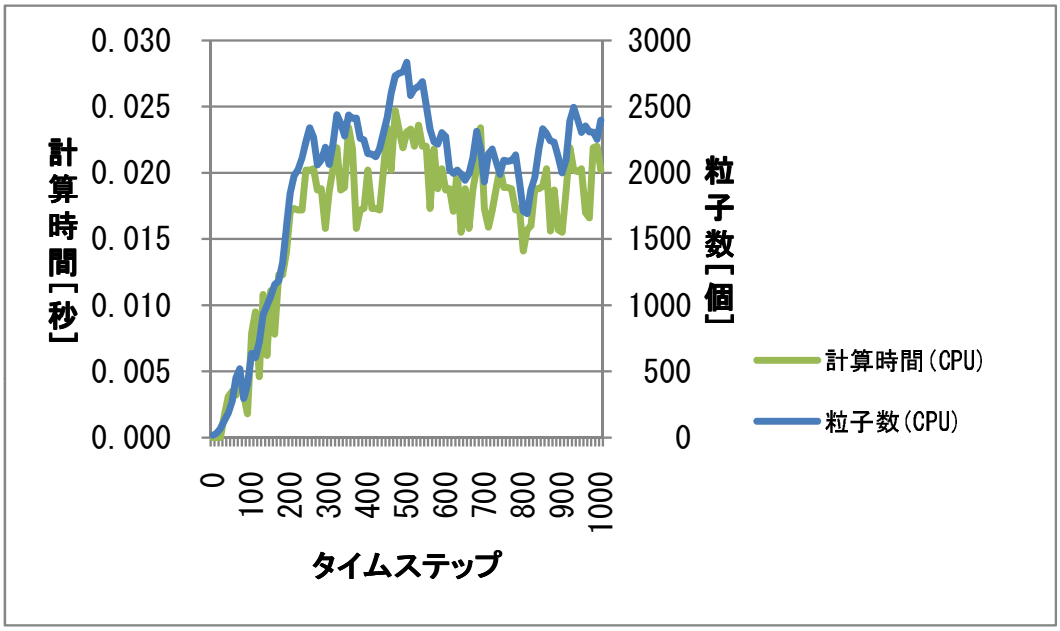


図 3.10: シミュレーションの進行による，粒子数と計算時間の変遷

粒子数は，シミュレーション開始後急激に増加し，以後緩やかに粒子が増えた後，1,700～2,700 個の間で推移していることが確認できる．また，計算時間は粒子数にほぼ比例している．

(a) (b) (c)

図 3.11: 火炎のマージのシミュレーション結果

2 つの火炎が滑らかに一体化し，1 つの大きな火炎となっていく．

第4章　火炎のレンダリング

前章で，火炎のシミュレーション法について述べた．これにより，粒子データの作成は可能となったが，粒子データからの火炎のレンダリングに関しては充分な方法が確立されていない．本章では粒子ベースボリュームレンダリングをもとにした，新たな火炎のレンダリング法を提案する．なお，本章の内容は文献 (間淵 他, 2011b) をもとにしている．

4.1　従来手法の問題点

第 2 章で述べたとおり，ボリュームデータのレンダリングには，主にレイマーチングとビルボードレンダリングが用いられているが，それぞれに長所と短所がある．

レイマーチングは格子データへ適用することが前提とされている．精度の高いレンダリングが可能であるが，粒子データへ適用する場合は，一度格子データに変換する必要があるうえ，定義されていない格子の外をレンダリングすることもできない．

ビルボードレンダリングは，描画対象ごとにスクリーンに投影していくため，高速なレンダリングが可能である．特に，各ビルボードを粒子に対応させることが容易なため，粒子データに対して広く用いられる．レイマーチングのように高精度な画像を得ることが難しいが，格子を必要としないため，レンダリング対象の空間に制限がない．

ビルボードレンダリングに関する問題とその対処は，文献 (Umenhoffer et al., 2006) に詳しく記述されている．この論文では，ビルボードポッピングアーティファクトとビルボードクリッピングアーティファクトが問題とされている．これらは，文献 (Umenhoffer et al., 2006) で示されたように，球状ビルボードを用いることによって，一定の対処が可能である．

しかし，なおもビルボードレンダリングは，本質的に，火炎に関してどのテクスチャが適切か決定できない，という問題点を保持している．本来，ビルボードレンダリングはテクスチャにイメージを描いていくものであり，ボリュームレンダリング法としては擬似的なものである．

4.2 アプローチ

本章では，次の特長を保持する，粒子ベースボリュームレンダリングをベースとする，粒子ベース火炎レンダリングを提案する．

4.2.1 汎用性

プログラマブルシェーダの進歩に伴い，複雑化するレンダリング技法のなかで，既存 API の活用や入力データへの要求が少ないことは実用性の点で重要である．提案手法のレンダリングは，主に Z バッファ法とアルファテクスチャ，アルファテストといった標準的なレンダリング手続きから構成され，これらは API によるサポートがなされるため実装が容易である．また，提案手法は，入力データとして粒子座標と温度を要求し，特定のシミュレーション手法やアニメーション手法に依存しない．

4.2.2 対話性

提案手法は，上述の通り標準的なレンダリング手続きから構成されるため，ハードウェアによる高速化の恩恵を受けやすい．これらから，提案手法は充分な高速性を保持することができる．

火炎は想定する現象やモニタ，静止画，動画圧縮形式，またそれを知覚する人間で大きく印象が異なる．従ってどのようなレンダリング手法を用いても完全にはなり得ないが，そのため調整の容易性は他の自然表現よりも重要である．提案手法は少数のパラメータを調整するだけで，火炎表現の調整が容易である．

4.3 粒子ベースボリュームレンダリング

本節では，既存の粒子ベースボリュームレンダリングの枠組みを紹介する．詳細は，文献 (小山田, 坂本, 2010) に詳しい．粒子ベースボリュームレンダリングでは，入力データとして，粒子データではなく，格子データを対象としている．その格子データから描画粒子を生成することを前提としている．その際の描画粒子は完全に不透明としている．格子データから各セルに格納されているスカラ値に応じて描画粒子を事前生成する．描画粒子生成座標は乱数によって決定される．描画粒子数と描画粒子座標は隣接格子セルのスカラー値を参照することで滑らかになる．生成された微小な描画粒子を，数十から数百回に分けて Z バッファ法により投影し，その画素の輝度平均をとることによって，半透明表現を行う．

この投影回数をリピートレベルとよんでいる．これは確率的なレンダリング手法の一種である．リピートレベルを増やせば，レイマーチングと同等の画質が得られる．この比較は文献 (小山田, 坂本, 2010) に示されている．同報告では最大二百万個程度の四面体格子から，数十万個から数百万個の描画粒子を事前生成し，それをリピートレベル数十から百程度で表示している．

概要図を**図 4.1** に，擬似コードを**アルゴリズム 4.1** に示す．

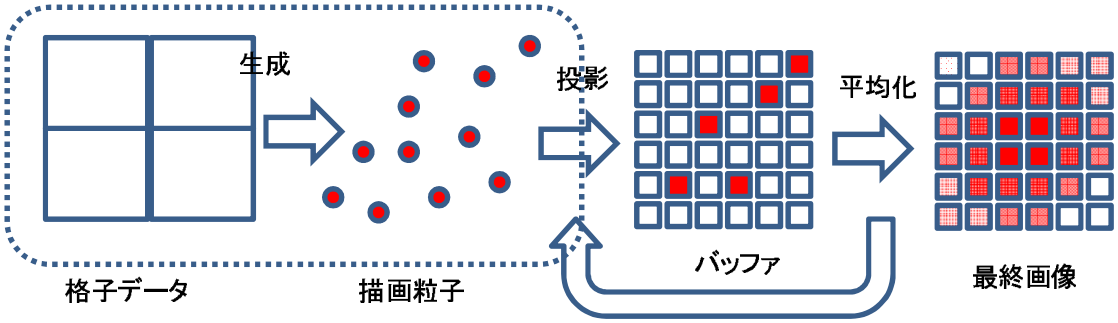


図 4.1: 粒子ベースボリュームレンダリングの概要

枠線部は事前処理を行う箇所である．格子データから事前に粒子を生成しておき，その粒子を順次投影していき，輝度平均をとることで半透明表現を行う．

アルゴリズム 4.1: 粒子ベースボリュームレンダリング

```
for all gridCells do
    generateAllPoints()
end for
for all repetitionLevel do
    getPoints(repetitionLevel)
    for all points do
        renderPoints()
    end for
    renderToBuffer(1.0/repetitionLevel)
end for
returnFromBuffer()
```

4.4　粒子ベースボリュームレンダリングによる火炎のレンダリング

本研究では，粒子ベースボリュームレンダリングをベースとして用い，これを対話的な火炎表現に適用させることとした．なお，以下では混乱を避けるため，もととなる粒子を物理粒子，物理粒子から乱数によって生成される描画用の一時粒子を描画粒子と定義する．

粒子ベースボリュームレンダリングは可視化一般を前提としている．一方本研究では対話的な火炎表現が目的であり，視覚的にそれらしく見ることが可能かどうかという点が重要なため，

- 描画粒子生成を粒子データ向けに簡易化する
- アルファテクスチャとアルファテストにより描画粒子数を擬似的に増幅する
- 色伝達関数に黒体放射モデルを用いる

ことでこれを達成する．提案手法である粒子ベース火炎レンダリングの概要を**図 4.2** に示す．

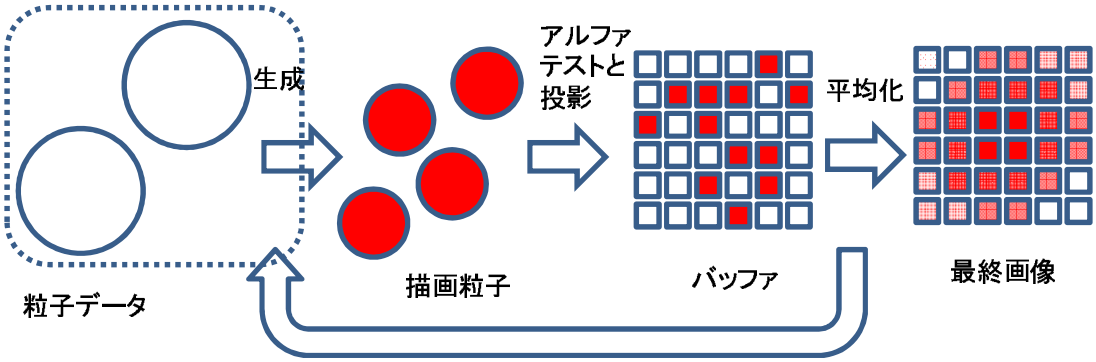


図 4.2: 粒子ベース火炎レンダリングの概要

枠線部は事前処理を行う箇所である．物理粒子データから描画粒子を生成し，その描画粒子を投影していき，輝度平均をとることで半透明表現を行う．

4.4.1　描画粒子生成の簡略化

本研究では接続情報のない物理粒子から描画粒子を生成するため，一様サンプリングを用いることとする．つまり，物理粒子の占めている空間に一様乱数で描画粒子を生成する．各フレームにおいて相対位置を保存しておかないとフレーム間の連続性が途切れてしまうため，最初に乱数テーブルを用意しておき，それを毎フレーム参照していくものとした．各物理粒子が，それぞれいくつの描画粒子を生成したかをカウントしておき，その描画粒子数をインデックスとして乱数テーブルを参照すれば，物理粒子毎に別々の乱数を用いる必

要はない．本研究では，メルセンヌ・ツイスタを用いて 1,000 個の乱数を用意した．なお，高温部ほど不透明になりやすいと考えられるため，生成する描画粒子数は，温度に比例させる．

4.4.2　描画粒子の擬似的な増幅

ただし，このままでは大量の微小な描画粒子が必要となり，粒子の事前生成が必要となってしまう．そこで本研究では，アルファテストとアルファテクスチャを用い，擬似的に描画粒子を増加させる．放射状のアルファ値に Perlin Noise (Perlin, 2002) を乗算して，穴あきアルファテクスチャを生成した．このテクスチャを，グレースケール表示で**図 4.3** に示す．穴あきアルファテクスチャを各描画粒子に貼りつけ，アルファテストで一定値以下の画素を切り捨てることにより，描画粒子のサイズを比較的大きくとることができ，実際の描画粒子数を抑えたまま，擬似的に描画粒子を増幅させることができる．アルファテストによる描画粒子の疑似的な増幅の一例を**図 4.4** に示す．この穴あきアルファテクスチャは，描画粒子のアルファ値に乗算するので，高温の描画粒子ほど合格率が高くなり，低温の描画粒子ほど合格率が低くなる．

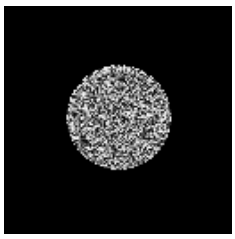
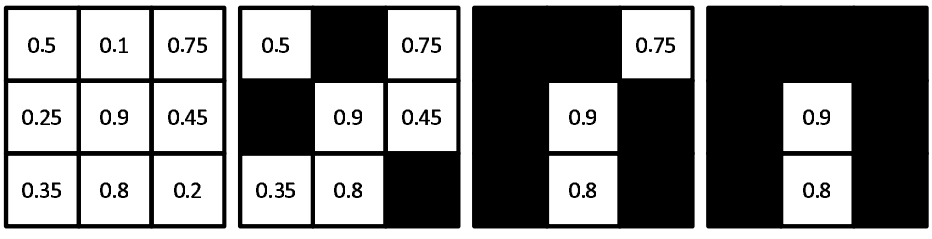


図 4.3: ノイズ乗算アルファテクスチャ（グレースケール表示）



(a) テクスチャ例　(b) 閾値 0.25　(c) 閾値 0.50　(d) 閾値 0.75

図 4.4: アルファテストによる疑似的な描画粒子の増幅

閾値を変化させることで，合格する画素が増減する．

アルゴリズム 4.2: 粒子ベース火炎レンダリング

```
// アルゴリズム 4.1 に変更を加えた箇所を斜体はとした.
applyAlphaTexture()
applyAlphaTest(threshold)
for all repetitionLevel do
  for all physicsParticles do
    generatePoints()
    for all points do
      renderPoints()
    end for
  end for
  renderToBuffer(1.0/repetitionLevel)
end for
returnFromBuffer(intensityScale)
```

4.5 実装

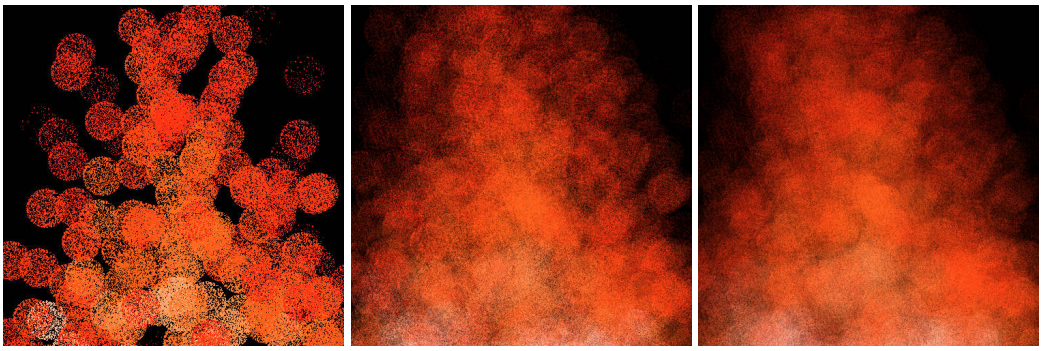
本章のアルゴリズムは，既存 API を活用することによって，容易に実装できる．レンダリングの擬似コードを**アルゴリズム 4.2** に示す．**アルゴリズム 4.1** と違い，事前に粒子生成を必要としない．毎回のレンダリング結果はアキュムレーションバッファに格納する．実際には，レンダリング総和をとると，描画されない画素が発生するアルゴリズムであり，輝度が落ちやすくなるため，このアキュムレーションバッファを利用して，輝度補正を行ったり，ブラーを使用したりすることが望ましい．

4.6 調整と拡張

本研究では，調整容易性を重視しているため，リピートレベルとアルファテスト閾値による比較をそれぞれ示す．本手法は，この 2 つのパラメータで直感的に調整することができる．また，既存 API による実装は拡張が容易であり，拡張例として煙の付加とラインの描画を示す．

4.6.1　リピートレベル

リピートレベルが多いほど描画精度は向上するが，レンダリング時間も増加するため，要求されるフレームレートや使用しているハードウェアに応じて調整する必要がある．観察者自身の残像効果も強く発生するため，比較的少ないリピート回数でも，高速にレンダリングが行われると火炎らしさが生成しやすい．比較画像を**図 4.5** に示す．



(a) リピートレベル 1　　(b) リピートレベル 10　　(c) リピートレベル 20

図 4.5: リピートレベルによる画質比較

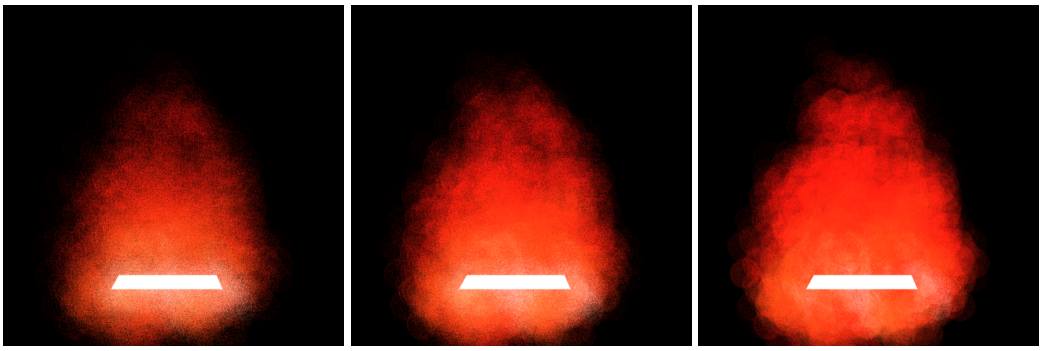
リピートレベルが増えるに従い，粒子から構成されている印象が薄くなる．

4.6.2　アルファテスト閾値

アルファテスト閾値が低いと，多くの画素が合格してしまうため，繰り返し投影，平均化されると，不透明に近づいていく．アルファテスト閾値が高いと，多くの画素が不合格となるため，繰り返し投影，平均化されると，透明に近づいていく．すなわち，アルファテスト閾値の調整は，不透明度の調整と等価な処理となる．比較画像を**図 4.6** に示す．

4.6.3　煙の付加

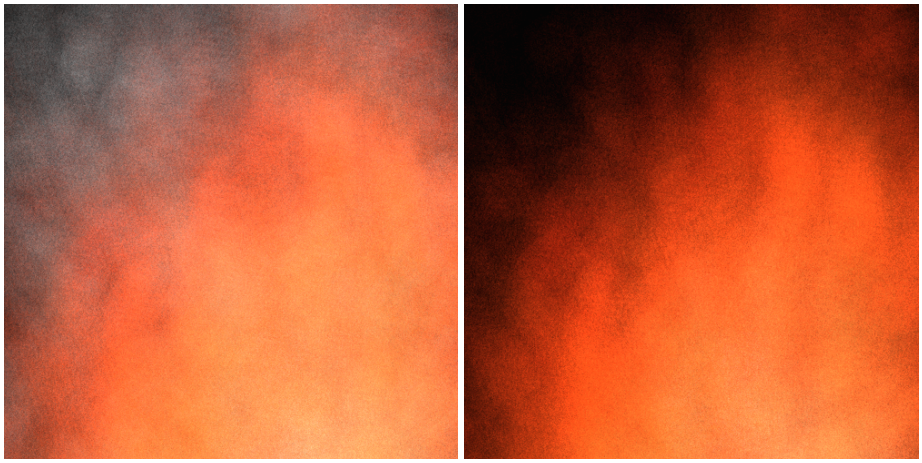
一定温度以下の物理粒子を，同様にレンダリングすることで，簡易的な煙の表現も可能となる．煙の色は一般にグレースケールに近いため，ユーザがこれを指定することとする．この煙は散乱を考慮していないが，火炎境界が滑らかになる．比較画像を**図 4.7** に示す．



(a) 閾値 0.25　　(b) 閾値 0.50　　(c) 閾値 0.75

図 4.6: アルファテスト閾値による不透明度の比較

アルファテスト閾値を上下させることで，不透明度の調整と等価な処理となる．



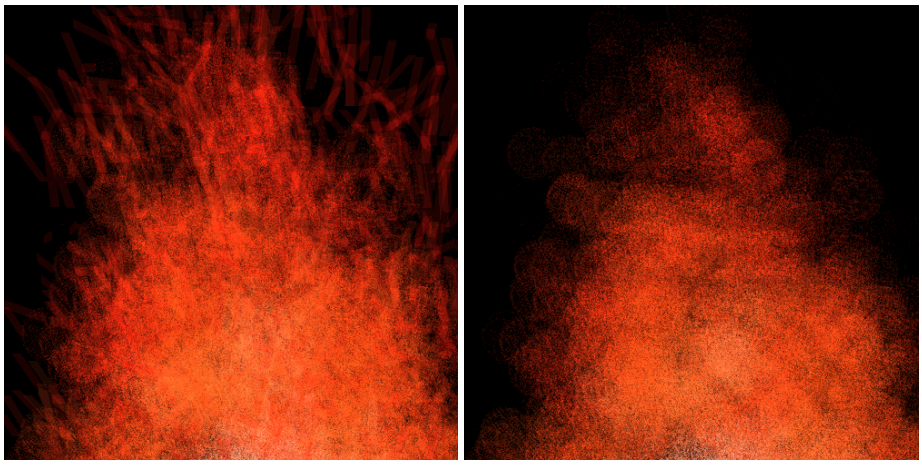
(a) 煙描画あり　　(b) 煙描画なし

図 4.7: 煙の有無による比較

煙を用いることで，火炎境界が滑らかになる．

4.6.4　ラインの描画

入力データとして，速度ベクトルが入手可能である場合は，ラインを描画することで，動的な印象を生成することが可能である．粒子が 1 点を指定するのに対し，ラインは 2 点を指定するだけの違いであるため，同様の枠組みで処理することができる．比較画像を**図 4.8** に示す．



(a) ライン描画あり　　(b) ライン描画なし

図 4.8: ラインの描画による比較
ライン描画のある方が動的な印象を生成する．

4.7　結果

本章の結果は CPU に Intel Core2 2.40GHz，主メモリに 2GB，GPU に NVIDIA GeForce 8800 GTS，実装に C++，グラフィックス API に OpenGL1.5 を用いて，512×512 の解像度で生成したものである．なお，プログラマブルシェーダは用いていない．本節中の結果は，物理粒子として，前章の SPH を用いている．これを**図 4.9** に示す．実用向けの例として，空と地面をテクスチャとして用意してレンダリングを行ったものを**図 4.10** に示す．この際の物理粒子数は 2,500，リピートレベルは 15 であり，各フレームのレンダリングは 0.03[sec] で完了した．以上より，本手法は充分な対話性を保持しているものと考える．

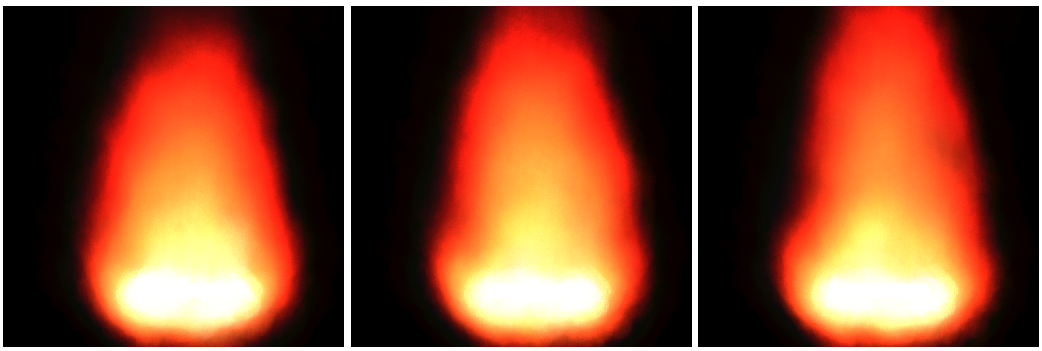
次では，ビルボードレンダリングによるレンダリング手法と本手法を比較する．同一物理粒子における比較図を**図 4.11** に示す．物理粒子数は 2,500 である．アルファテクスチャとして**図 3.2** のテクスチャを用いた．ビルボードレンダリングでは，各粒子のテクスチャが

見えやすくなり，粒子から生成された画像であるという印象が発生してしまう．本手法では，粒子境界が滑らかになり，粒子から形成されている印象が少なくなる．

この際のレンダリング速度は，ビルボードレンダリングが 0.01[sec]，本手法ではリピートレベル 5 で 0.015[sec]，リピートレベル 10 で 0.025[sec]，リピートレベル 20 で 0.05[sec] であった．ビルボードレンダリングより高速ではないが，リピートレベルを操作することにより，レンダリングの速度と品質を容易に調整することが可能である．これに相当する処理はビルボードレンダリングでは困難である．

さらに，ビルボードレンダリングでは，粒子を増減させるたびに画素に書き込まれる輝度が増減するため，各粒子のアルファ値を再調整しなければならない．一方，本手法では，リピートレベルを増減させても画素に書き込まれる輝度の期待値は変わらないため，パラメータの再調整は不要である．

以上より，火炎が背景表現や瞬間的な表現に過ぎないアプリケーションにおいては，ビルボードレンダリングが効果的であるが，火炎表現を主体とするアプリケーションでは本手法が効果的であると考えられる．



(a) (b) (c)

図 4.9: 粒子ベース火炎レンダリングによる安定状態の火炎のレンダリング

4.8 まとめと考察

本章では，既存の粒子ベースボリュームレンダリング法を簡易化し，火炎に特化したレンダリング法について述べた．

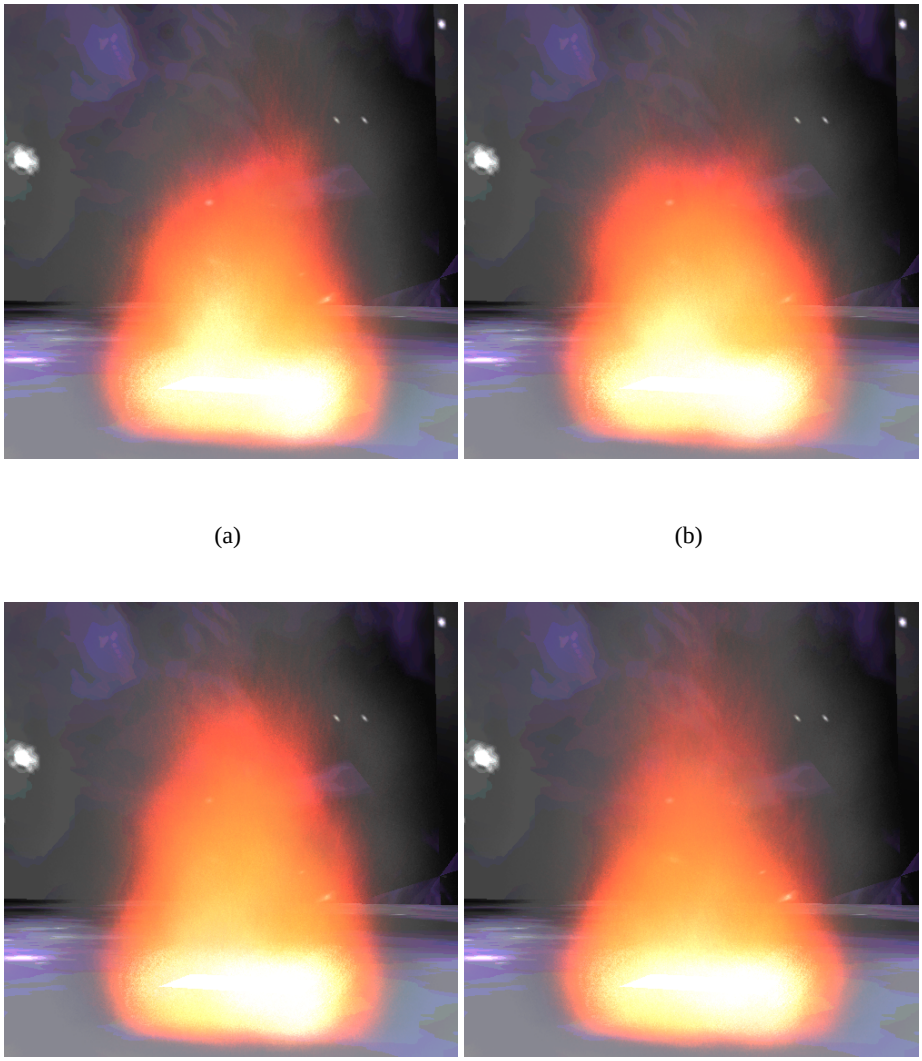
CG においては，粒子データに対するレンダリング法が整っていなかった．本章では，自己発光する火炎に限定することで，粒子データに対する効果的なレンダリングが可能であ

ることを示した．これはシェーダプログラミングを必要とせず，既存 API を用いるだけで再現が可能である．

他の表現への応用として，煙や雲のレンダリングが考えられる．これらは自己発光をせず，表現には内部散乱が重要となる．提案したレンダリング法では，内部散乱の扱いが難しい．負荷はやや高くなるが，同じ粒子で扱うことのできる，フォトンマッピングを簡易化して用いることで，これらの表現が可能であるものと考える．

描画粒子生成がやはりボトルネックとなっているため，ここまでの数千程度の物理粒子のレンダリングは対話的に行えるが，それ以上になると対話性は低下する．これには並列処理が不可欠である．

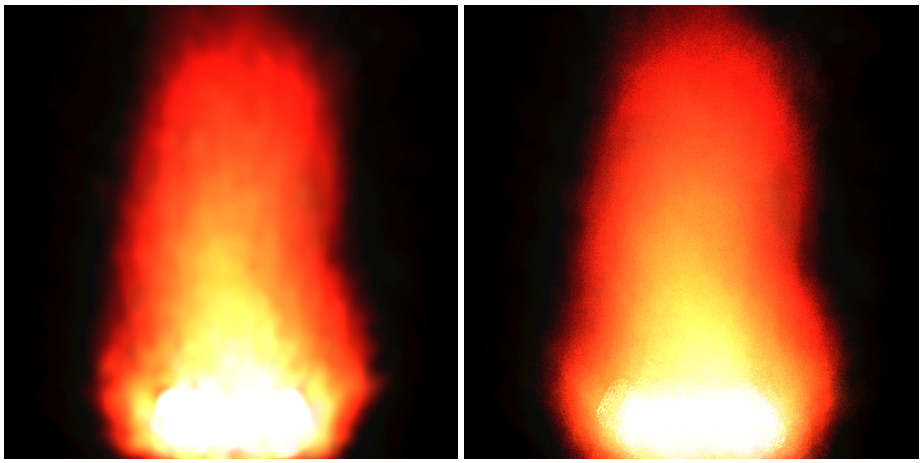
本章までは，単一の CPU とオンボードの GPU という現在のハードウェアとしては，最低限の構成を前提に論じてきた．より高い表現を求めるには，マルチコアの CPU を用いる，プログラマブル GPU を用いる，といった並列処理が必要となる．本論文では，次章以降，プログラマブル GPU を用いる方法を選び，並列処理を行うこととする．



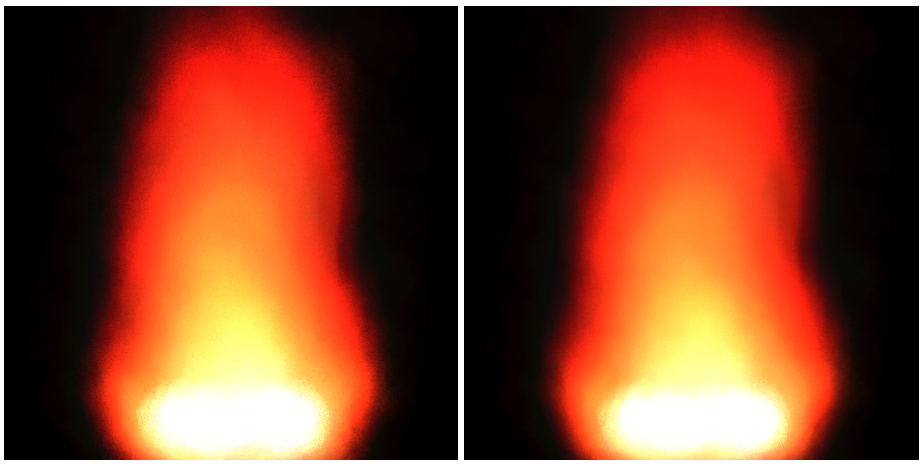
(a) (b)

(c) (d)

図 4.10: 粒子ベース火炎レンダリングに背景を付加したレンダリング例



(a) ビルボードレンダリング　　(b) リピートレベル 5



(c) リピートレベル 10　　(d) リピートレベル 20

図 4.11: ビルボードレンダリングと粒子ベース火炎レンダリングの比較

ビルボードレンダリングはテクスチャが見えやすくなるのに比べ，粒子ベース火炎レンダリングは滑らかである．

第5章　火炎シミュレーションのGPGPU高速化

第3章で述べたとおり，CPUで対話的に扱うことのできる粒子数は数千程度が限界である．本章では，より多くの粒子を扱うことのできる，GPGPUによる高速な火炎シミュレーション法を提案する．本章の内容は文献 (Mabuchi et al., 2011) をもとにしている．

5.1　CPU実装の問題点

第3章のCPUでのシミュレーション法でも火炎表現が可能であり，また前章のレンダリング法を用いることで，より効果的なビジュアルシミュレーションが可能である．しかし，より多くの粒子を用いることで，ビジュアルシミュレーションの精度を向上させることができる．ただし，数万単位の粒子を単独のCPUで扱う場合，計算コストが増え，対話性が低下してしまう．高い対話性を維持したまま，扱うことのできる粒子を増やすには，何らかの並列処理が必要になる．

5.2　アプローチ

並列処理には，マルチコアCPU上でのマルチスレッドプログラミング，GPGPUプログラミングなどがある．本章ではGPGPUを用い，次の要件を満たす並列処理法を提案する．概要を**図5.1**に示す．

- 並列処理を行うことによって，大量の粒子を扱う
- 並列処理実装部を明確に区別する．つまり並列処理を行う，行わないに関わらず，CPUシミュレーションの結果に影響を及ぼさない
- 既存の並列処理機構を用いることによって，実装負荷を最小限にする

なお，マルチコアCPU上でのマルチスレッドプログラミングは5.6節で検討する．

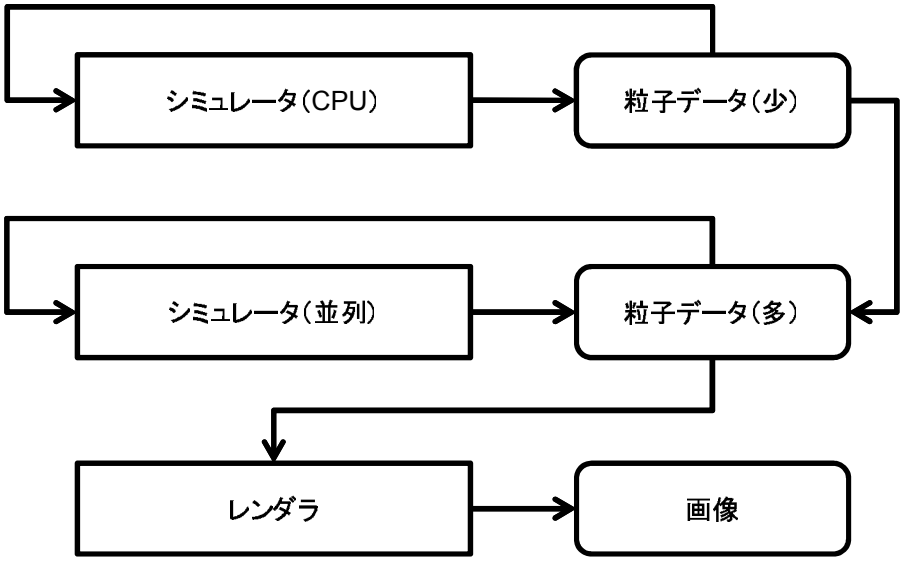


図 5.1: 並列処理シミュレーションの概要

並列処理実装部を明確に区別する．

5.3 GPU を用いた物理シミュレーション

GPU を用いた物理シミュレーションは，シェーダ言語を用いて始められた．特に SPH を GPU 上で実装したものは，Harada の各文献 (Harada et al., 2007b)， (Harada et al., 2007a) に詳しい．シェーダ言語を用いたプログラミングがたいへん困難であったが，第 1 章で述べたとおり，現在では，GPU での汎用計算に NVIDIA CUDA などを用いることができる．さらに，NVIDIA PhysX には，SPH を含む多くの物理シミュレーションが GPU 実装されている．

5.4 GPU を用いた火炎のシミュレーションの高速化

一般に，CPU は柔軟な処理が可能であるが，大量のスレッドを扱うのには適していない．対して，GPU は柔軟な処理が難しいが，大量のスレッドを扱うのに適している．本章では，両方の PU の長所を活かした並列シミュレーション法を提案する．

本章では，以下の定義を行う．

- “CPU 粒子”は CPU メモリ上の粒子データを意味する
- “GPU 粒子”は GPU メモリ上の粒子データを意味する

CPU から GPU へのメモリ転送を**図 5.2** に，メモリ転送を含む並列シミュレーションの概要を**図 5.3** に示す．最初に，それぞれの GPU 粒子は，それぞれの CPU 粒子の周辺のラン

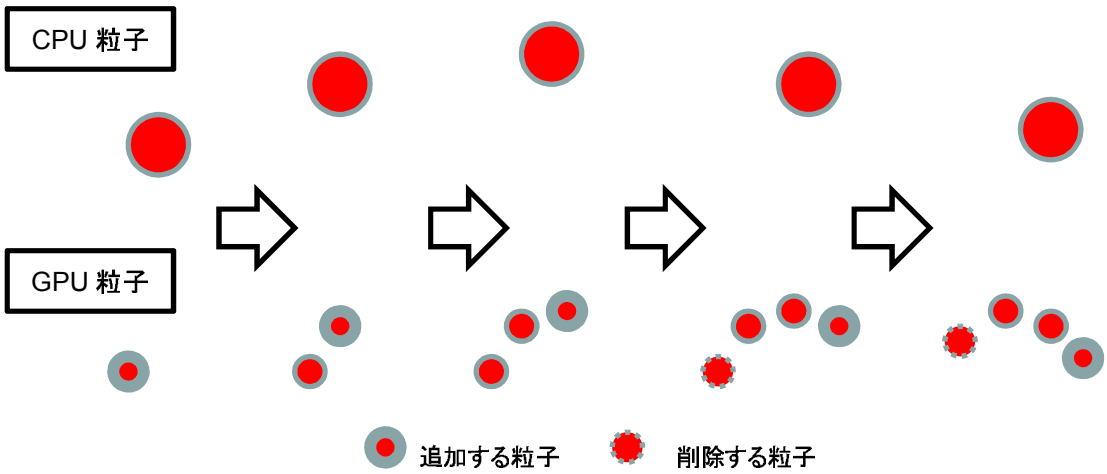


図 5.2: CPU から GPU へのメモリ転送

CPU 粒子のあとに GPU 粒子が次々に追加，削除される．

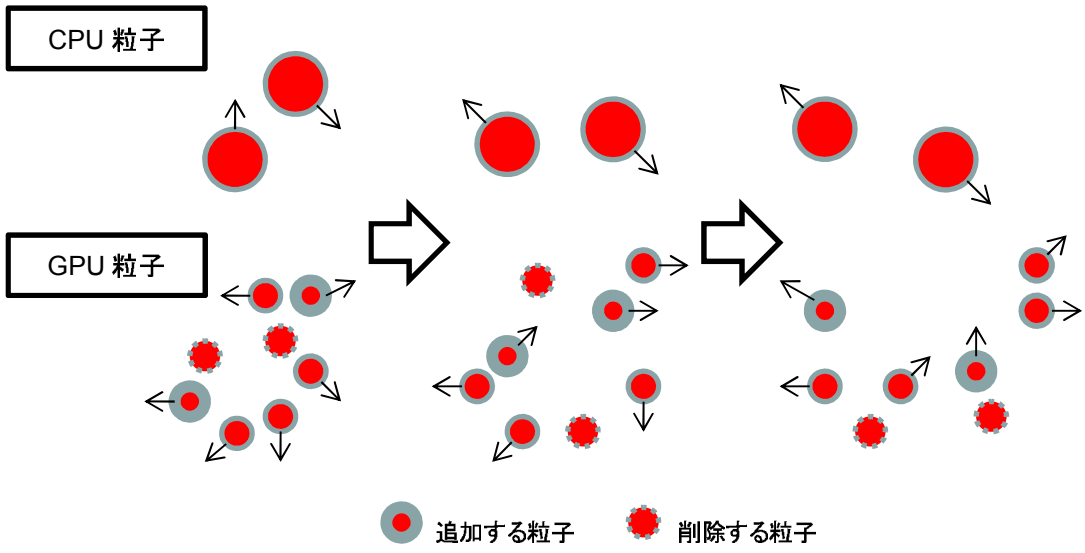


図 5.3: CPU と GPU による並列シミュレーション

GPU 粒子が次々に追加，削除されながら，CPU 粒子は CPU 粒子間で，GPU は GPU 粒子間で，それぞれ相互作用が計算され，シミュレーションが並列に進行する．

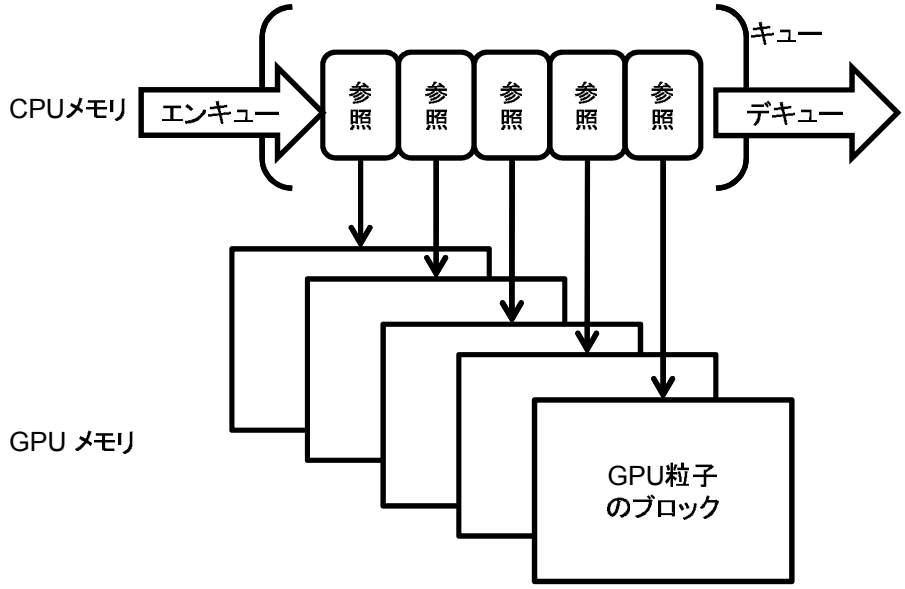


図 5.4: キューによる GPU メモリ管理

GPU メモリブロックを用意し，CPU でキューによってこの GPU メモリブロックへの参照を管理する．

アルゴリズム 5.1: CPU-GPU シミュレーションプログラム

```
deleteOldestGPUParticles()
createNewGPUParticles()
CPUSimulator.simulate()   //  アルゴリズム 3.2
GPUSimulator.simulate()   //  アルゴリズム 3.1
synchronizeCPUandGPU()
```

ダムな位置に生成される．この際の GPU 粒子の速度は CPU 粒子の速度を継承する．次に，GPU 粒子を GPU 上で SPH によりシミュレーションする．このままでは GPU 粒子が増える一方なため，GPU 粒子は，一定タイムステップ後に削除する．これは同時に，CPU シミュレーションと GPU シミュレーションの乖離を防ぐことにもなる．この処理は CPU と GPU の間で行うデータ転送を最小限に済ませることができ，GPU シミュレーションと CPU シミュレーションを同時に実行することが可能になる．

本研究では，上述のとおり，GPU 粒子の生成・破棄が頻繁に起こる．GPU 実装におけるボトルネックは頻繁な動的メモリ再割当てである．一般的に，GPU は些細なメモリ再割り当てには不向きであるが，大規模なメモリ再割当てには比較的向いている．そこで本研究ではキューを用いた GPU メモリの管理を実装した．このキューは CPU で管理され，GPU の各メモリブロックへの参照を保持する．それぞれのタイムステップにおいて，もっとも古い GPU 粒子データセットは破棄され，新しい GPU 粒子データセットが生成される．つまり，キューのサイズが大きくなるに従って，GPU 粒子の個数も大きくなる．本論文では，キューのサイズを 5 とした．**図 5.4** はこのメモリ管理の概要を示す．

前節で述べたとおり，GPU 上での SPH シミュレーション技術は既に提案されており，サポートする物理エンジンも存在している．そのため，追加実装は最小限で済む．**アルゴリズム 5.1** に，この並列シミュレーションプログラムの擬似コードを示す．この方法では，**アルゴリズム 3.2** で示した CPU シミュレーションプログラムに一切変更を加える必要がない．

5.5 結果

本章の結果は CPU に Intel Core2 2.40GHz，主メモリに 2.0GB，GPU に NVIDIA GeForce 8800GTS，実装に C++，物理エンジンに NVIDIA PhysX 2.8.3，グラフィクス API に OpenGL1.5 を用いて実装したものである．シミュレーション自体の効果を明示するため，前章のレンダリング法ではなく，第 3 章と同様に，ビルボードレンダリングを用いた．

第 3 章と同様のシミュレーション結果を，それぞれ示す．火炎の形成から安定状態への推移までを**図 5.5** に示す．**図 3.6** と比べて，火炎のコアが成長していく様子が確認できる．固体からの作用を受けるシミュレーションを**図 5.6** に示す．GPU 上でのシミュレーションでは，固体に関する扱いを行っていないが，CPU 上でのシミュレーション結果をトレースするため，固体からの作用が実現される．また，外力を加えた例を**図 5.7** に示す．**図 3.8** と同様，強い外力により，火炎が小さくなっていく．これらのシミュレーションは GPU 粒子数 12,500 個前後で安定し，各ステップ 0.02〜0.035[sec] で実行された．

粒子数と計算速度の遷移を，**図 5.8** に示す．粒子数は，シミュレーション開始から 200 ステップあたりまで，10,000 個程度まで急激に増加し，以後緩やかに粒子が増えた後，8,000〜13,000 個の間で推移していることが確認できる．本研究の範囲内では，GPU シミュレーションでも，計算速度は GPU 粒子数にほぼ比例していた．また，CPU シミュレーションに対する GPU シミュレーションの計算速度比はおよそ 1.5 倍程度であった．

応用例として，火炎のマージを**図 5.9** に示す．**図 3.11** と同様に，2 つの小さな火炎が 1 つの大きな火炎となっていく．このシミュレーションは GPU 粒子数 25,000 個前後で安定し，各ステップ 0.05[sec] で実行された．

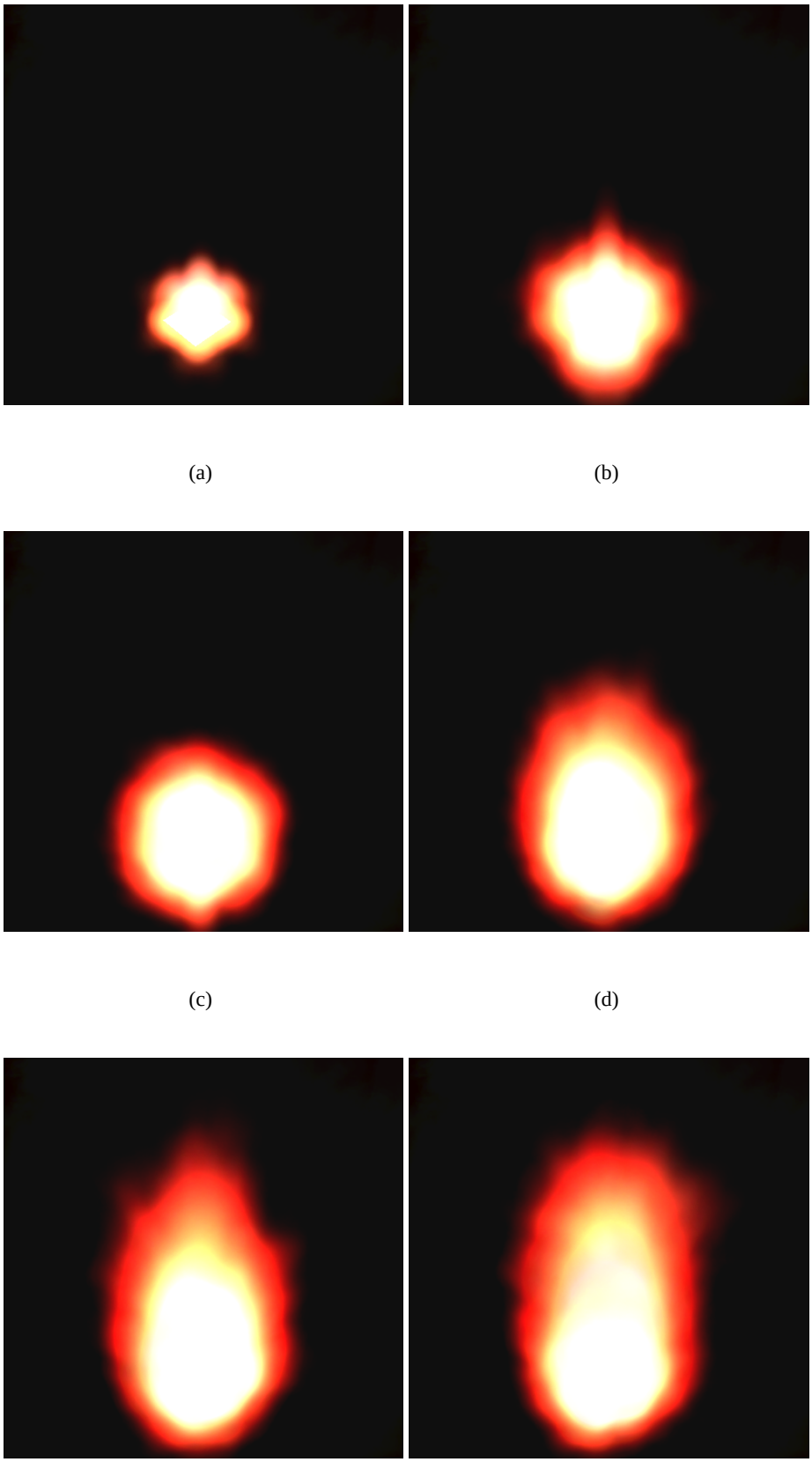
5.6 まとめと考察

本章では，火炎シミュレーションに GPGPU を用い，対話性を低下させることなく，既存の物理エンジンを用いることで，実装難度を抑えてより多くの粒子を用いることが可能であることを示した．

しかし，本手法を実装するには，GPGPU が利用可能でなければならない．これは現在の PC のハードウェア構成からは必ずしも一般的とは言えない．また，本来はレンダリングに用いられるはずの GPU を使用し続けることになる．一方，現在の PC のハードウェア構成ではマルチコア CPU が一般的となっている．前節での結果に示した通り，GPU を用いた手法により，約 1.5 倍の計算量で，5 倍の粒子数を可能としている．しかし，マルチコア CPU を適切に利用すれば，同程度の高速化は可能であると考えられる．各社のマルチコア CPU や，GPU がどれほど普及し，性能向上を行っていくのかを見通すことは困難である．本章では，CPU と GPU による並列処理を示したが，この別の PU への粒子データの受け渡しやメモリ管理は，どのハードウェアを用いても，共通したものとなると考えている．

次に，渦のシミュレーションについて考察する．現在のシミュレーション法では，渦に関して特別な計算を行っていない．小規模な火炎であっても，火炎周囲の渦表現があれば，火炎先端の揺らぎを発生することができ，表現を大きく前進させることができる．粒子法ではないが，格子法においては，少数の格子セルから複雑な渦表現が可能であることは Pfaff (Pfaff et al., 2010) によって示されている．この結果から，粒子法においても，各粒子において渦度を保持，計算することによって，渦の表現が可能であると考えられる．この渦表現用の粒子では，ボトルネックである，粒子間相互作用を計算しなくて済むことが示されているため，総合的な計算速度を落とすことなく，シミュレーションに幅をもたせられる可能性がある．

レンダリングの面から比較すると，第 3 章よりは向上がみられるが，ビルボードレンダリングでは，粒子を多く用いるようになると，各画素に書き込まれる色の調整が困難になっていく．また，やはり粒状の印象が発生している．これより，本質的にはビルボードレンダリングでは，高精度なレンダリングは困難であると判断できる．しかし，前章で導入した火炎レンダリング法は，速度面ではビルボードレンダリングに及ばないため，本章の粒子データに適用すると対話性に影響が生じてしまう．次章では，前章で導入した火炎レンダリング法の，GPU 実装による高速化を示す.



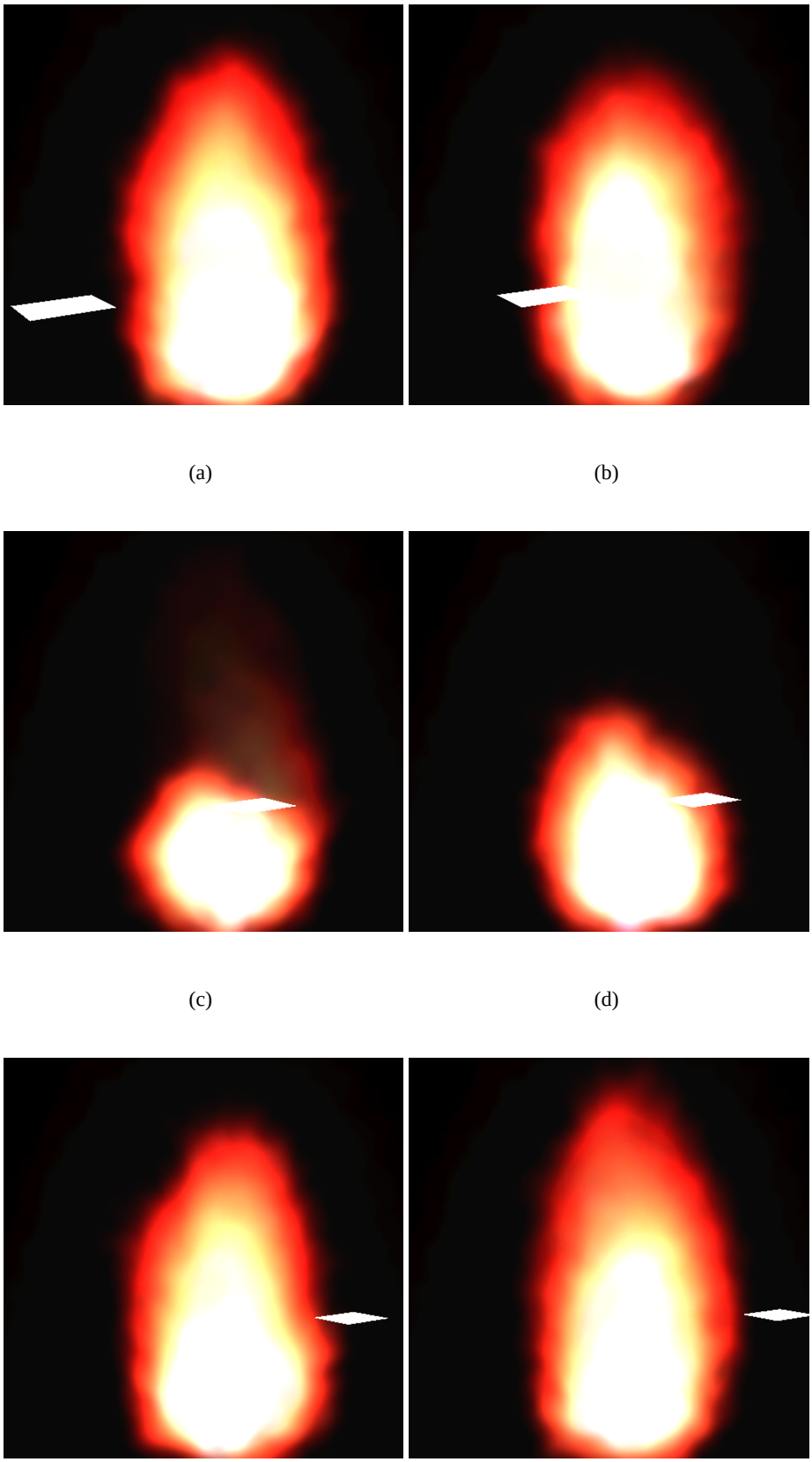
(a) (b)

(c) (d)

(e) (f)

図 5.5: GPGPU による火炎の形成シミュレーション

図 3.6 と比べて，火炎のコアが成長していく様子が確認できる．



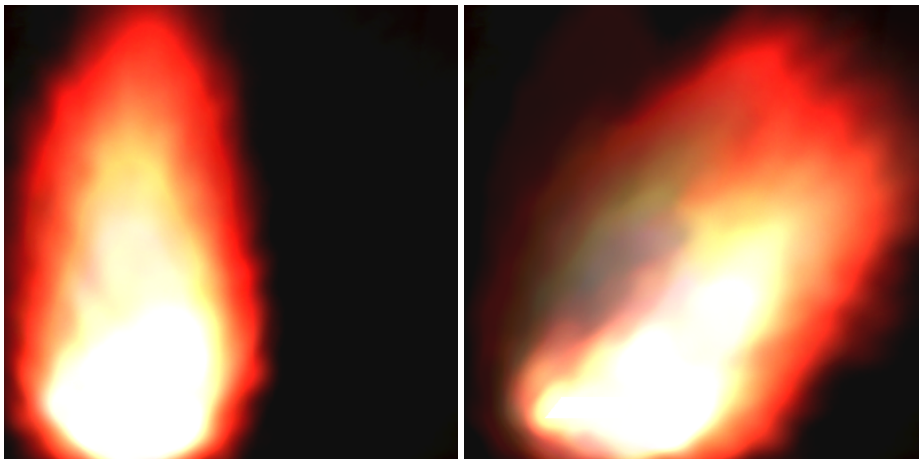
(a) (b)

(c) (d)

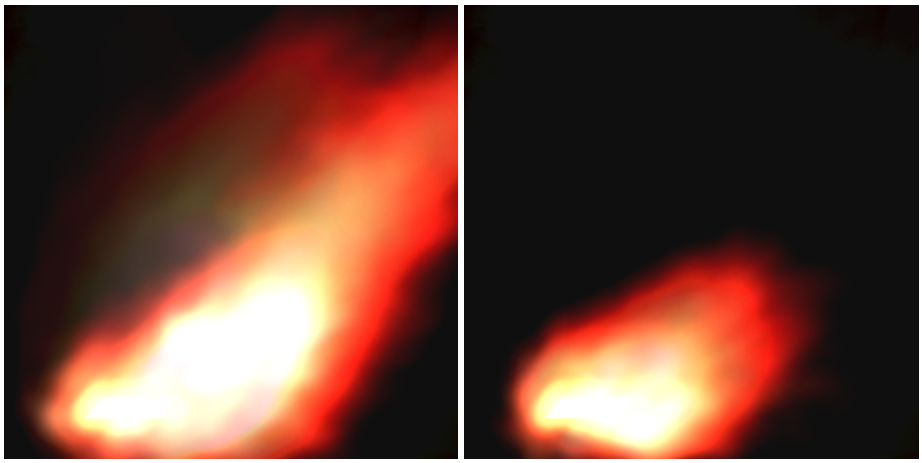
(e) (f)

図 5.6: GPGPU による固体からの作用のシミュレーション

GPU のシミュレーションでも，固体からの作用が実現される．



(a) (b)



(c) (d)

図 5.7: GPGPU による外力による変形シミュレーション

図 3.8 と同様，強い外力により，火炎が小さくなっていく．

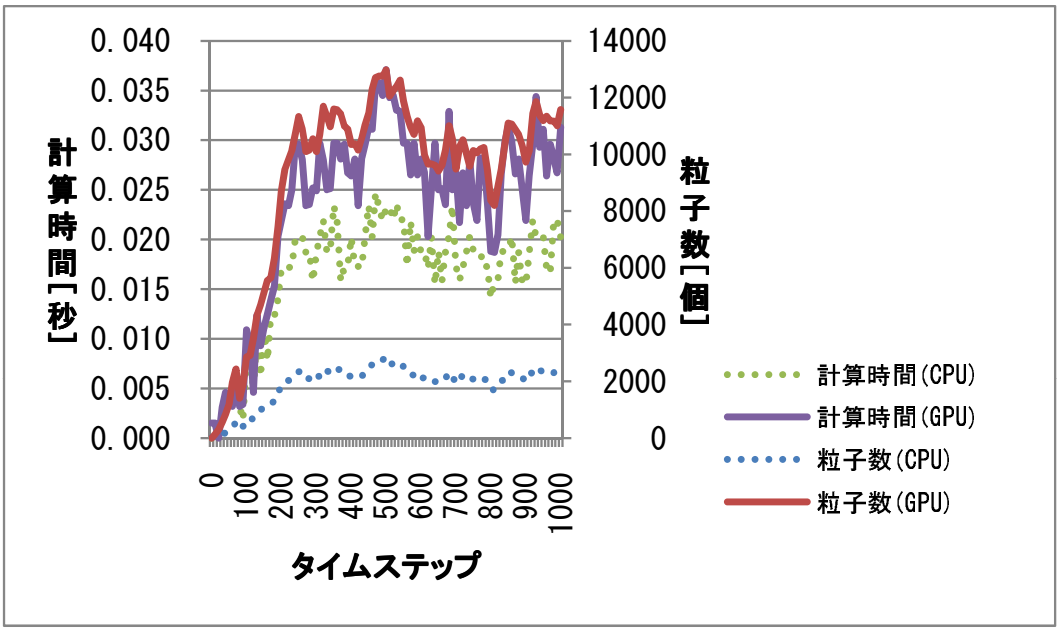
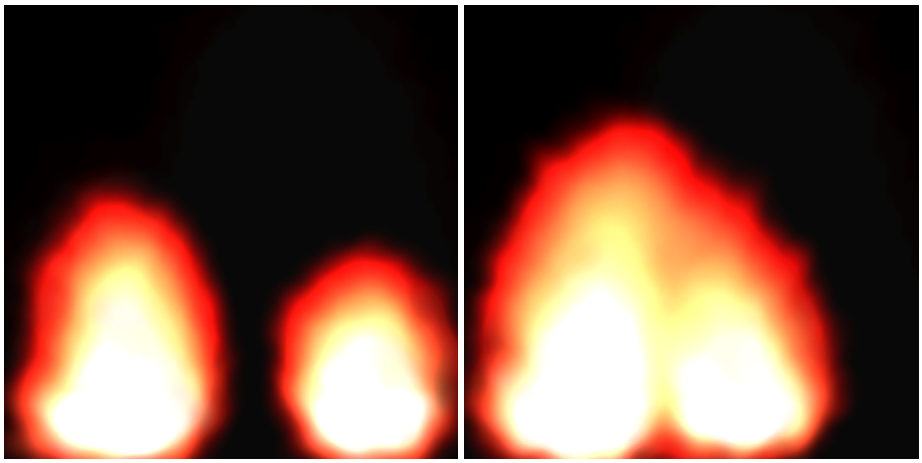
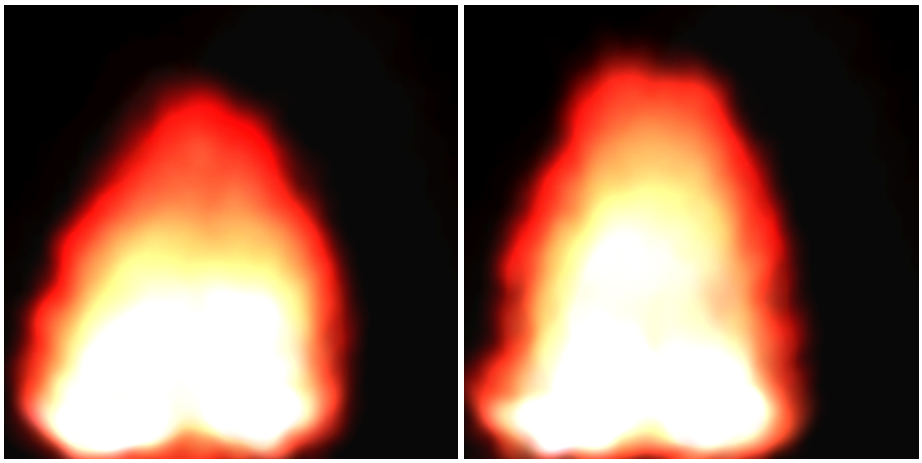


図 5.8: シミュレーションの進行による，GPU 粒子数と計算時間の変遷

GPU 粒子数は，シミュレーション開始後急激に増加し，以後緩やかに粒子が増えた後，8,000～13,000 個の間で推移している．また，計算時間は GPU 粒子数にほぼ比例している．



(a) (b)



(c) (d)

図 5.9: GPGPU による火炎のマージのシミュレーション

図 3.11 と同様，2 つの小さな火炎が 1 つの大きな火炎となっていく.

第6章　火炎レンダリングのGPU高速化

第 4 章では，CPU（固定 API）による火炎のレンダリング法について述べ，前章では GPGPU による火炎のシミュレーション法について述べた．これらを統合するため，火炎のレンダリングの GPU による高速化について述べる．

6.1　CPU 実装の問題点

粒子ベース火炎レンダリングでは，大量の描画粒子生成の負荷が大きく，数万単位の物理粒子へ適用すると対話性が低下してしまう．つまり，前章の GPGPU による火炎のシミュレーションデータのレンダリングには，何らかの高速化処理が必要となる．

6.2　プログラマブルシェーダ

近年の対話的 CG は，固定 API によるレンダリングから，GPU によるレンダリング（以下シェーダ）に移行している．固定 API では柔軟な処理が難しく，提供されていない機能を用いる場合は，CPU で実装する必要があった．一方シェーダでは，C 言語に似た高級言語で GPU 実装できるようになった．

しかしながら，GPGPU と違いレンダリングに特化しているため，主には 4 つのシェーダステージが存在する．具体的には，バーテックスシェーダ，テッセレーションシェーダ，ジオメトリシェーダ，ピクセルシェーダから構成されている．バーテックスシェーダは，CPU から渡された頂点を調整する．モデルビュー投影変換を加えることが代表的な処理である．テッセレーションシェーダは，制御パラメータを通じてプリミティブを再分割することができ，複雑な曲面の詳細度制御処理に適している．ジオメトリシェーダは，各プリミティブを任意に追加することができる．テッセレーションを明示的に行ったり，飛沫を表現したりすることに適している．ピクセルシェーダは，各プリミティブから画面に投影される各画素の色を決定する．陰影計算やテクスチャの適用を行うことが多い．これらのステージの概要を**図 6.1** に示す．

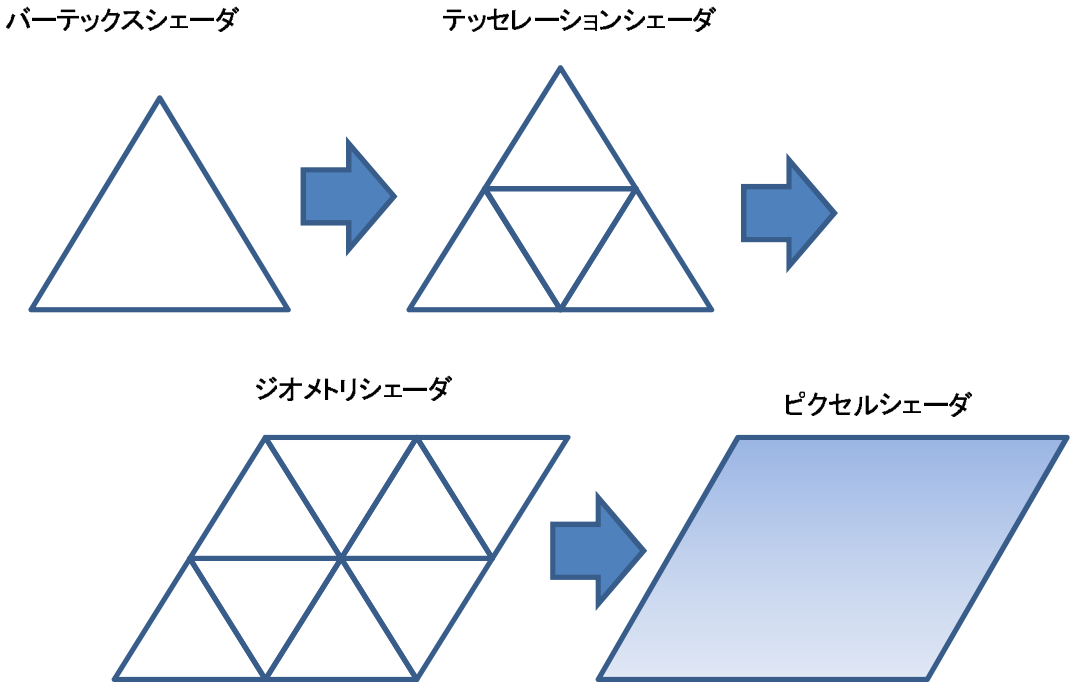


図 6.1: プログラマブルシェーダの各ステージ

バーテックスシェーダは CPU から渡された頂点の調整を，テッセレーションシェーダはプリミティブの再分割を，ジオメトリシェーダはプリミティブの任意追加を，ピクセルシェーダは各画素の色決定を行う．

6.3　GPU を用いたレンダリングの高速化

粒子ベース火炎レンダリングの観点から各ステージを検討すると，バーテックスシェーダはモデルビュー投影変換行列を乗ずるだけである．テッセレーションシェーダはプリミティブが点であるため，使用する必要がない．ピクセルシェーダはアルファテクスチャとアルファテストを適用する．

ここまでは，既存 API による実装に比べて利点がないが，重要となるのはジオメトリシェーダである．このジオメトリシェーダを活用することで，高速な粒子生成が可能となる．この概要を**図 6.2** に，擬似コードを**アルゴリズム 6.1** に示す．

乱数は GPU 内での生成が困難であるが，事前に CPU 側で生成したものをテクスチャとして GPU 側に渡しておく．この際，テクスチャの各テクセルには RGBA の順に乱数をそれぞれ割り当てていく．この RGB を X 座標，Y 座標，Z 座標とみなす．なお，A は実際には使用しない．ここまでは，1 次元テクスチャとして扱うことが適当であるが，GPU 実装では CPU 実装と違い，各物理粒子がそれぞれいくつの描画粒子を生成したかをカウントすることができないため，リピートレベル分だけ乱数テーブルを用意しておく必要がある．ただし，毎回テクスチャを GPU に転送しなおすと無駄が多くなるため，2 次元テクスチャとして GPU に一括転送することで効率化を図る．本研究では，メルセンヌ・ツイスタを用いて 400,000 個の乱数を用意して，乱数テクスチャを生成した．CPU 実装と比べると多くの乱数が必要となるが，実際には，100×1000 の解像度の RGBA テクスチャを転送するだけで済む．**図 6.3** にこの概要を示す．

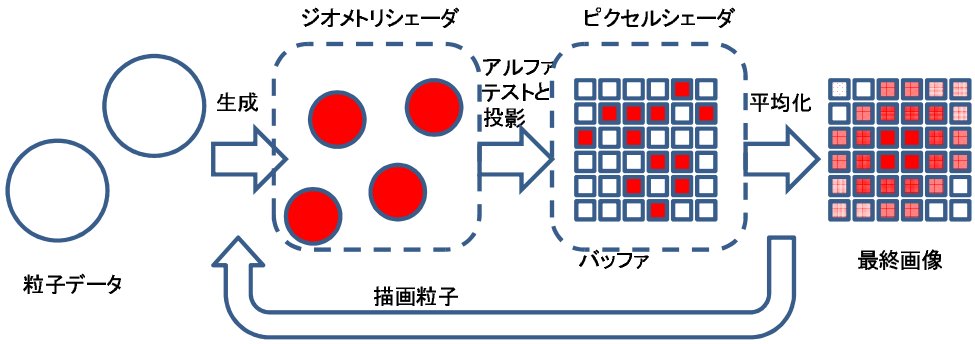


図 6.2: GPU による粒子ベース火炎レンダリング

描画粒子生成をジオメトリシェーダに割り振ることで高速化を達成する．

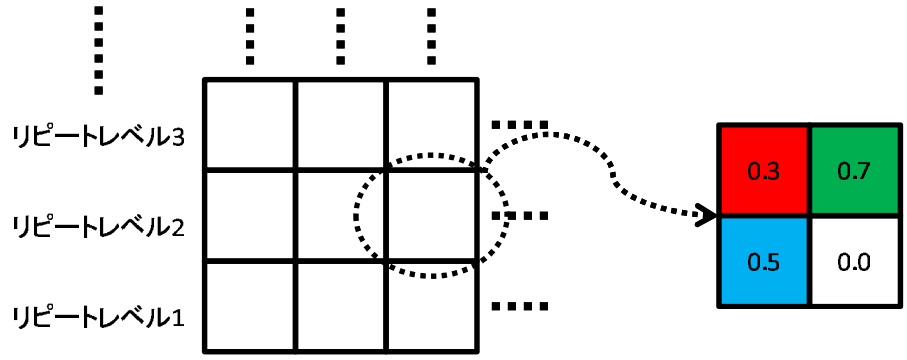


図 6.3: 乱数テクスチャ

リピートレベル毎に乱数を用意する．その乱数は RGB 値として保存する．

6.4 結果

本章の結果は CPU に Intel Core2 2.40GHz，主メモリに 2.0GB，GPU に NVIDIA GeForce 8800 GTS，実装に C++，グラフィックス API に OpenGL3.3 と GLSL3.3 を用いて，512×512 の解像度で生成したものである．これらは，物理粒子として前章の GPGPU による高速化した SPH を用いている．特に粒子が疎になりやすい，外力を加えたシミュレーションデータのレンダリングを**図 6.4** に示す．また，背景に実写背景を用いてレンダリングを行ったものを**図 6.5** に示す．この際の物理粒子数は 12,500 個，リピートレベルは 15 であり，各フレームのレンダリングは 0.03[sec] で完了した．以上より，GPU を用いることで充分な対話性を提供することができる．

第 4 章と同様に，リピートレベルによる比較画像を**図 6.6** に示す．固定 API 実装と同様，リピートレベルが多いほど画質は向上する．なお，本手法ではリピートレベル 5 で 0.01[sec]，リピートレベル 10 で 0.015[sec]，リピートレベル 20 で 0.03[sec] であった．第 4 章の結果と比べると，物理粒子数が増えているにも関わらず，速度が向上していることが判明した．

シミュレーション結果に合わせて，視点を近付けながらシースルーアニメーションを行った画像を**図 6.7** に示す．火炎の遠くから近付き，火炎を通り抜けるまでが，それぞれ滑らかにレンダリングされていることが確認できる．

応用例として，火炎のマージとスプリットのレンダリングを**図 6.8** に示す．最大 35,000 個の物理粒子を用い，0.06[sec] 程度であった．レンダリングでも粒子のみを用いているため，動的な変化が自動的に達成される．

アルゴリズム 6.1: GPU による粒子ベース火炎レンダリング

```
// Vertex Shader
vertex.position = position * modelViewProjectionMatrix
vertex.temperature = temperature
// Geometry Shader
howMany = getHowMany(vertex.temperature)
for all howMany do
   position = texelFetch(randomTexture)
   emiteVertex()
end for
emitPrimitive()
// Pixel Shader
if alpha > threshold then
   discard()
end if
```

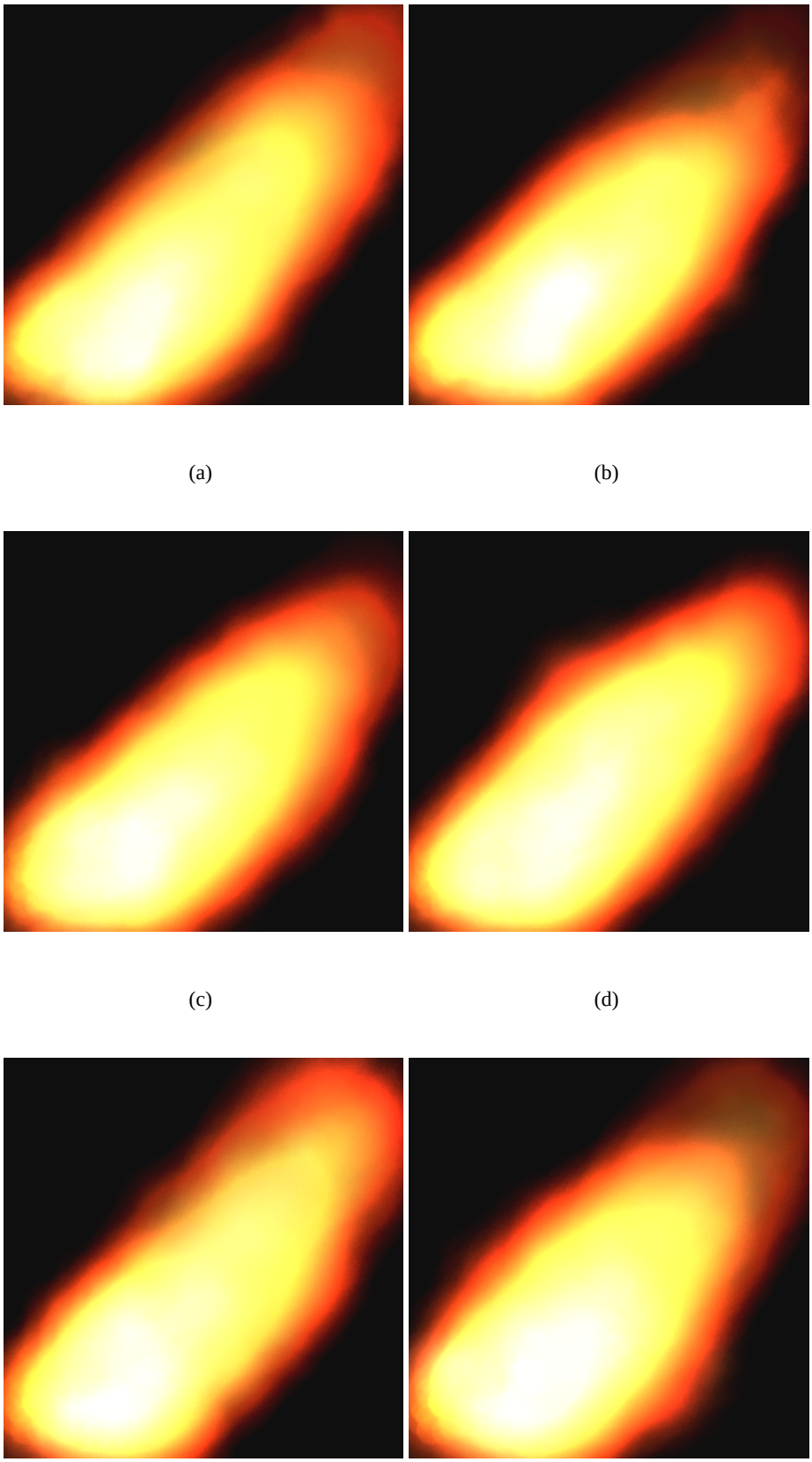
6.5 まとめと考察

本章では，粒子ベース火炎レンダリングの GPU 実装について述べた．ジオメトリシェーダを活用することにより，前章まででは対話性をもたせることが困難であった，数万単位の物理粒子から，レンダリングを行うことを可能とした．

現在のところ，ジオメトリシェーダを扱うことが可能な GPU は必ずしも多くはない．また，対応している GPU でも，ドライバにより性能のばらつきが予想される．現在の対話的 CG において，GPU 内で描画要素を増減させることは主流になりつつある．本章の高速化手法は，粒子データに限らず，格子データに対しても適用できる．これまでのところ，粒子ベースボリュームレンダリングは，シェーダを用いた高速化が充分に行われていない．火炎レンダリングに限らず，ボリュームレンダリングの一手法として，一般化が可能である．

また，実写との合成も試したが，違和感が残っている．これは，屈折の表現を行っていないこと，火炎を光源として処理していないことに原因がある．井村の手法 (井村 他, 2010) では，屈折用にテクスチャを用意し，それを一度バッファに書き込んでおいてからレンダ

リングを行い，屈折表現を行っている．本手法でも，同様の屈折表現は扱える可能性が高い．また，光源として扱うには，第 4 章で議論したとおり，簡易的なフォトンマッピングが必要であり，同時に実写の深度値も必要であり，別途，バーチャルリアリティの面から，研究を進める必要があるものと考えている．



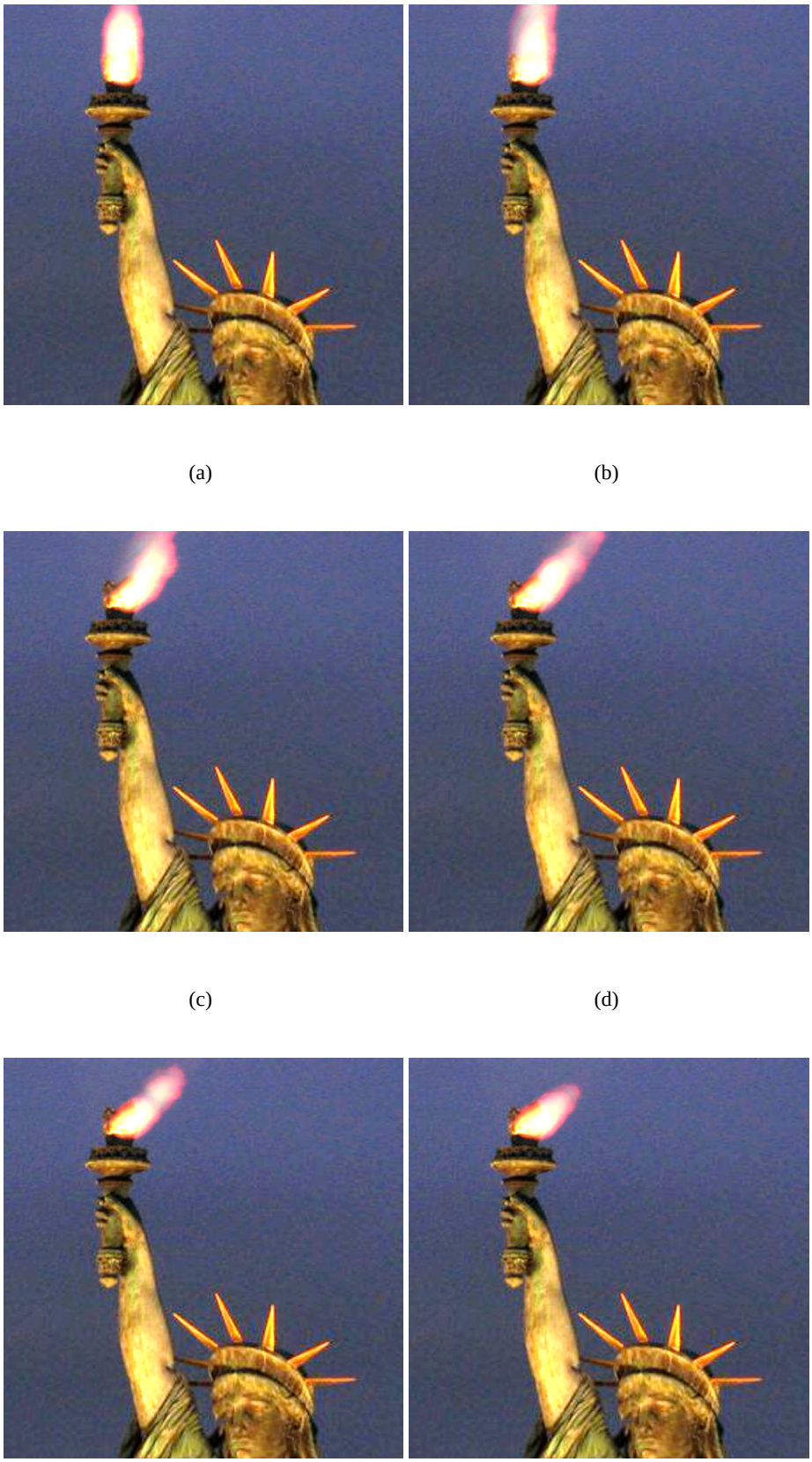
(a) (b)

(c) (d)

(e) (f)

図 6.4: GPU レンダリングの外力を適用したシミュレーションデータへの適用

粒子が疎であっても，本手法では滑らかにレンダリングされる．



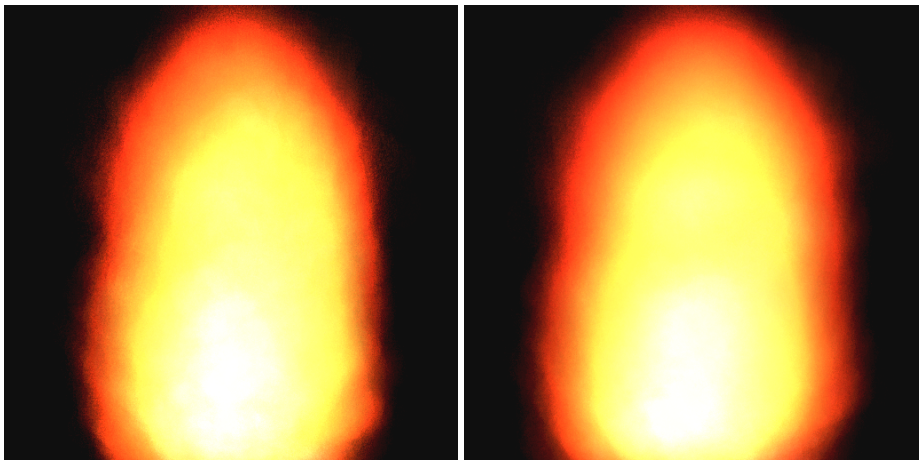
(a) (b)

(c) (d)

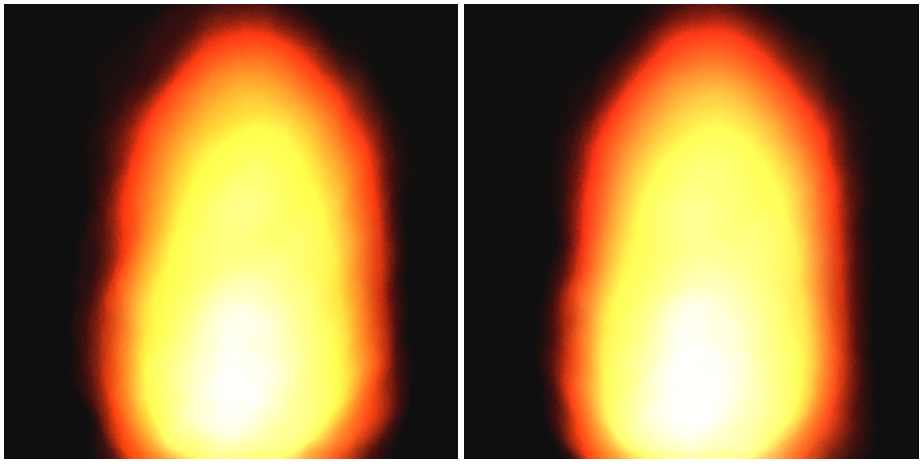
(e) (f)

図 6.5: GPU レンダリングに実写背景を用いた例

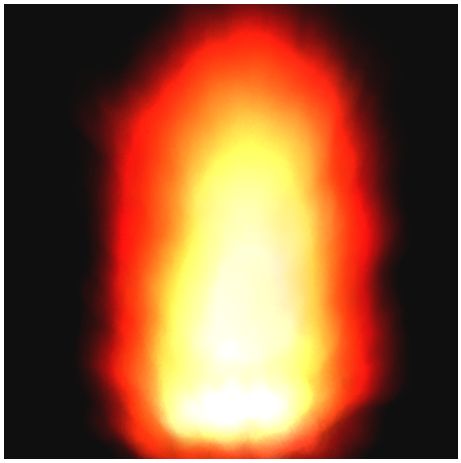
背景の画像は Wikipedia「自由の女神像」より引用.



(a) リピートレベル 5　　(b) リピートレベル 10



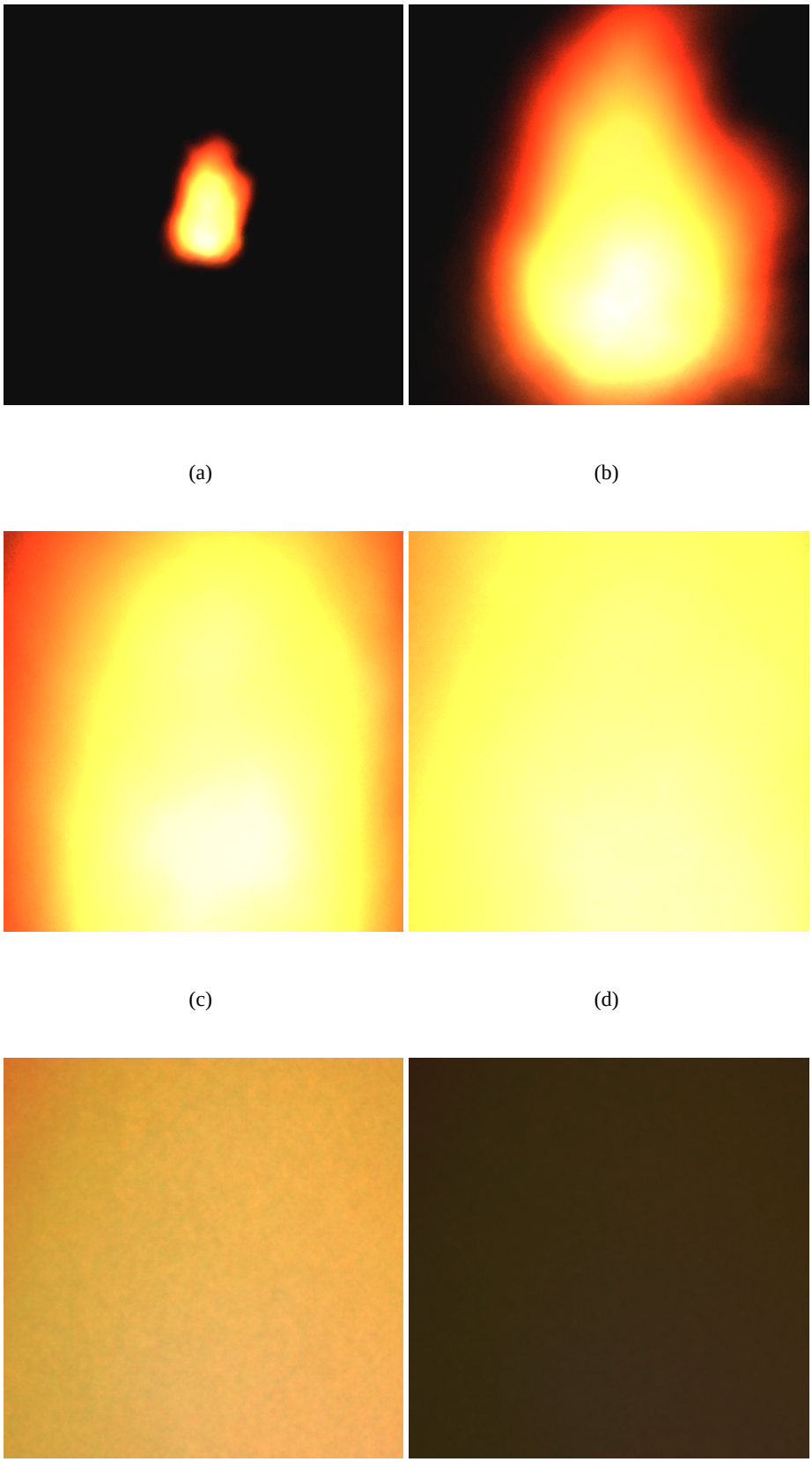
(c) リピートレベル 15　　(d) リピートレベル 20



(e) ビルボード（参考）

図 6.6: GPU レンダリングのリピートレベルによる画質比較

リピートレベルが増えるに従って，滑らかになる．



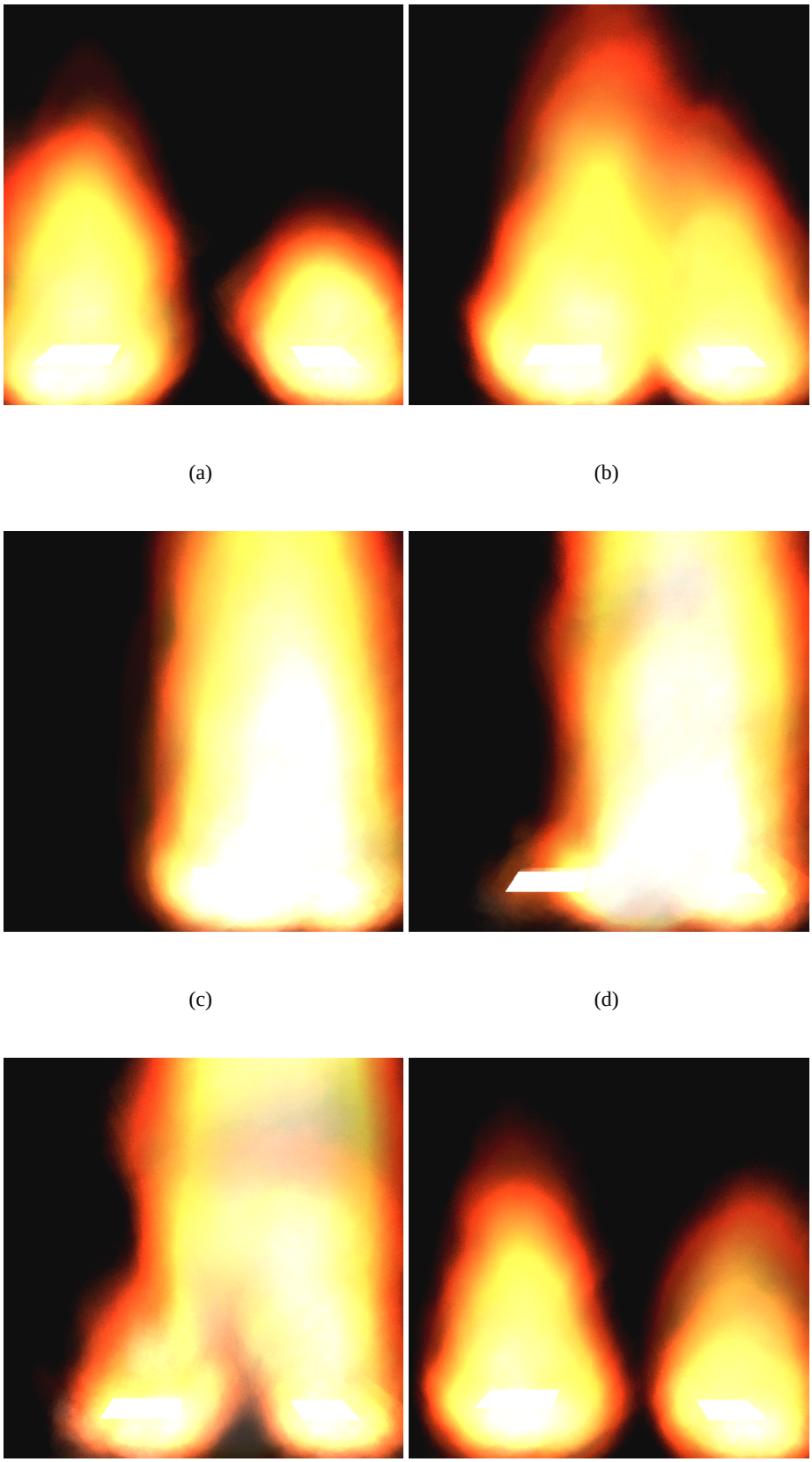
(a) (b)

(c) (d)

(e) (f)

図 6.7: GPU レンダリングによるシースルーアニメーション

火炎の遠くから近付き，火炎を通り抜けるまでが，それぞれ滑らかにレンダリングされている．



(a) (b)

(c) (d)

(e) (f)

図 6.8: GPU レンダリングによる 2 つの火炎のマージとスプリット

火炎の規模に関係なくレンダリングが可能である.

第7章　評価

本章では，本研究の評価として，現実の火炎との比較，高精度なシミュレーションとの比較，CG 作成者の評価の 3 点を論じる．

7.1　現実の火炎との比較

桑名 (桑名 他, 2009) による小規模火炎の写真を**図 7.1** に，大規模火炎の写真を**図 7.2** にそれぞれ引用して示す．カラーの写真は，文献 (Baukal Jr and Shwartz, 2001) が参考になる．現実の火炎との比較を，慶應義塾大学理工学部機械工学科の横森剛専任講師にお願いしたところ，次の評価をいただいた．好ましい点は，

- 全体的な挙動は現実の火炎に近い
- 小規模な火炎と似ている
- 中心部が白く，外側が赤い現象は実際によく見られる
- 火炎上部が千切れていく現象が再現できている

などであり，小規模火炎を前提とすれば，全体的なモデルには大きな問題がないと考えられる．また，好ましくない点は，

- 小規模な火炎にしては，反応面がなく，火炎下部が白過ぎる
- 大規模な火炎には見えにくい
- アニメーションがスローに見え，違和感がある

などであった．これらをまとめると，主には，時間スケール，空間スケールに関する問題が大きい．実写と確認してみても，**図 7.1** の小規模火炎は，本研究の結果に近いが，**図 7.2** の大規模火炎は，本研究の結果から大きく異なることが分かる．ビジュアルシミュレーションでは，あえて誇張した表現をすることがあるため，このスケールの問題は重要になる．時間スケールに関しては，シミュレーションタイムステップ幅を大きくとっても安定するア

ルゴリズムにする，空間スケールに関しては，渦専用のソルバを追加し，大規模な現象に見せること，などが改良案として考えられる．

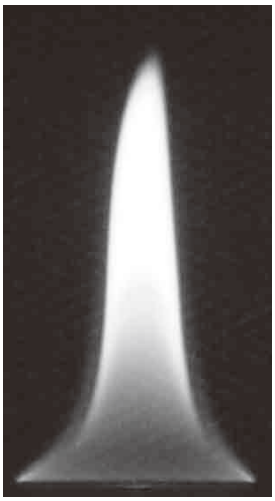


図 7.1: 桑名 (桑名 他, 2009) による小規模火炎（底面 3cm）の写真

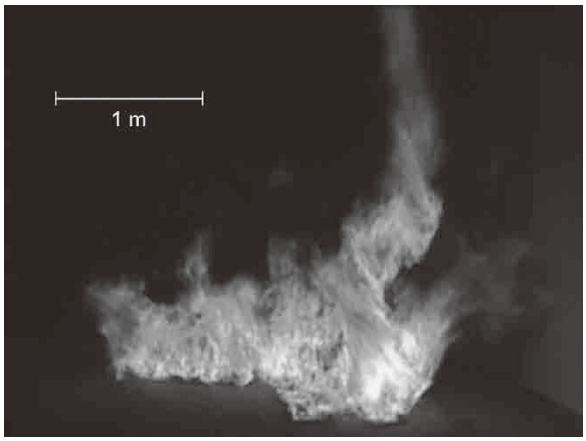


図 7.2: 桑名 (桑名 他, 2009) による大規模火炎の写真

7.2　高精度なシミュレーションとの比較

CG 分野において提案されている手法のうち，最も精度の高いビジュアルシミュレーション法は，格子法による，Nguyen の論文 (Nguyen et al., 2002) であると判断する．本節では，この論文を参照しながら比較を行う．

この論文の，火炎放射を想定した結果を**図 7.3** に，安定状態を想定した結果を**図 7.4** に，2 つの燃焼源から生成された火炎を**図 7.5** にそれぞれ引用して示す．これと，例えば**図 6.7**

を比較すると，中心が白色に近く，外側ほど赤身を帯びる点は同じである．Nguyen の手法では数日単位の計算時間がかかるが，本手法では対話的に作成することができることを考慮すれば，安定状態に関しては，充分な結果であると考える．しかし，**図 7.3** のように高速な初期条件下では，渦が支配的となり，本研究の結果と比べ，ディテールの表現が豊富である．Nguyen の手法は，煙のシミュレーション (Fedkiw et al., 2001) で用いられた渦度計算を用いている．そのため，このように渦が支配的となり，詳細な表現が可能となっている．粒子法では，渦に関する研究が進んでいないため，本研究でも渦に関して特別な計算をしていない．これは格子法と粒子法との差であるのではなく，渦のシミュレーションが粒子法では未完成であることが原因である．

図 7.5 では，2 つの燃焼源から生成された火炎が示されているが，その前後にマージしたりスプリットしたりする結果は報告されていない．格子セル数が膨大になるため，シミュレーション空間に余白がないものと考えられる．本手法では**図 6.8** で示したとおり，大規模空間を対話的に扱うことができる．これはすべてを粒子で取り扱う，本手法の特長である．

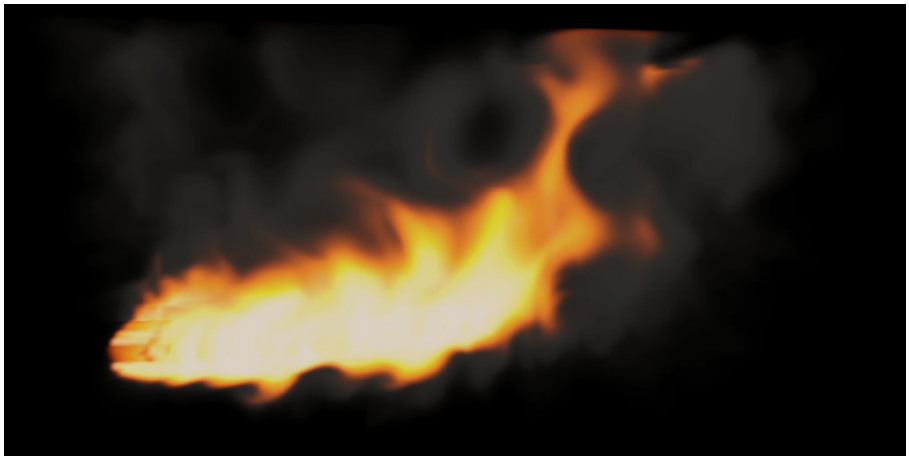


図 7.3: Nguyen (Nguyen et al., 2002) による火炎放射のシミュレーション

7.3 CG 作成者の評価

主にドラマや報道番組用の CG を制作する，株式会社フジテレビジョン 技術開発局開発業務センター設備運用部と，主にアニメーションや映画用の CG を制作する，株式会社オー・エル・エム・デジタル 研究開発部にお願いし，本研究に関して以下の意見をいただいた．

次に関しては，ほぼ同じ意見であった．

演出可能性 誇張した火炎表現が必要．この表現は自由度が高い方が望ましい

図 7.4: Nguyen (Nguyen et al., 2002) による安定状態の火炎のシミュレーション

図 7.5: Nguyen (Nguyen et al., 2002) による 2 つの燃焼源を用いた火炎のシミュレーション

付随表現 煙やすすなどの表現ができるほうがよい

背景への映り込み 実写背景や 3D モデルへの，火炎の映り込みが重要になる

実写素材の利用 シミュレーションよりも実写素材を用いることが多い

演出可能性については，渦専用のソルバを付加することで一定の表現が可能である．また本手法のような物理ベースの研究だけでなく，演出を組み合わせた研究も既に行われており (Horvath and Geiger, 2009) (Kawada and Kanai, 2011)，これらの研究成果の取り込みが重要になるものと考えられる．付随表現や，背景への映り込みに関しては，現在のところ，光源計算を導入していないため実現できていない．簡易的なフォトンマッピングを開発することで，これに対応することができると考える．実写素材の利用は，プリレンダリングにおいては効果が高いが，ゲームやバーチャルリアリティのような，ユーザのアクションによって状況が変化していくアプリケーションにおいては限界があり，シミュレーションの必要性が高いものと考える．

次は意見が分かれた．

制作スピード より早く結果が得られることが必要（フジテレビジョン）．元々時間と手間をかけてエフェクト作成を行う（オー・エル・エム・デジタル）．

手軽さ より手軽に結果が得られることが必要（フジテレビジョン）．元々エフェクト作成を一工程で済ませることはない（オー・エル・エム・デジタル）．

時間スケール 火炎が遅く感じる．シミュレーション速度やフレームレートを操作したい（フジテレビジョン）．コンポジットを何度も行うので，結局は後工程で調整する（オー・エル・エム・デジタル）．

空間スケール 火炎のスケールが分かりづらい（フジテレビジョン）．コンポジットを何度も行うので，結局は後工程で調整する（オー・エル・エム・デジタル）．

本研究は対話性が主題であるため，フジテレビからいただいた意見に重点を置いて考察する．空間スケールと時間スケールは，7.1 節で述べたことと同様の対処が考えられる．制作スピードや手軽さについては，本手法のように調整パラメータが少ない点は重要であるが，現状では実写による撮影ほど手軽ではない．しかし今後，制作スピードはハードウェアの向上により，手軽さはプラットフォームの抽象化により解決されていくものと考えている．

第8章　結論

本論文では，粒子を用いた対話的な火炎のビジュアルシミュレーションを提案した．ビジュアルシミュレーションを，物理シミュレーション法と，効果的に表現するためのレンダリング法に分け，またそれぞれを CPU と GPU 双方に実装した．

本論文では，計 4 種類のビジュアルシミュレーション法を示したが，他の物理シミュレーション法で得られた結果を，本レンダリング法で描画することも可能であり，本シミュレーション法で得られた結果を，別のレンダリング法で描画することも可能である．実際には，物理シミュレーションモジュールとレンダリングモジュールは同様のものを使うとは限らないため，このモジュール化は実用的であると考えている．また，プログラマブルシェーダ対応の GPU が必ずしも一般化していない点も踏まえ，CPU による実装と，GPU によるアクセラレーションとを分離して述べた．現在でも，商用の CG エンジンや物理エンジンは，ハードウェア次第でディテール表現を変化させている．この点からも，実用性が高いと考えられる．

上述の通り，本論文では GPU を用いることによって並列化を図った．しかしながら，近年ではマルチコア CPU が普及しているため，CPU による並列化も考えられる．ハードウェア詳細を意識させない，並列処理仕様も普及しはじめているが，CPU メーカは CPU 並列化に，GPU メーカは GPU 並列化に重点をおいているため，その並列処理仕様だけで，充分な速度を引き出すことができるのは当面先であると予想される．実際には，シミュレーションを CPU で分担させ，レンダリングを GPU に特化させることで最適な結果が得られる．

並列処理を用いた具体的な今後の展望として，シミュレーションに関しては，渦専用のソルバを追加することで，比較的少ない粒子でも詳細な表現を追求すること，レンダリングに関しては，簡易的なフォトンマッピングを追加することで，煙や爆発火炎などの内部散乱を考慮すること，などが挙げられる．

火炎に限らず，自然現象の対話的表現は未完成の分野である．本論文では対象を火炎そのものに絞ったが，煙・雲・爆発などの自然現象一般に応用が可能であると考える．これまでのところ，粒子法は限定的にしか自然現象の表現を可能としていない．新たな計算プリミティブが不要である上，統一的なデータ構造を活用することが可能であることから，粒

子によるビジュアルシミュレーションは，将来性が高いものと考えている．

また，本論文ではアルゴリズムに絞って論じたが，実際に対話性を重視するにはユーザインタフェースが重要となる．ビジュアルシミュレーション一般に，ユーザインタフェースに関する研究が少ない．火炎のシミュレーションに限らず，対話的なパラメータ調整は重要であるため，キーボードやマウスでの直感的なパラメータ調整，また重要なパラメータの保持・管理など，一般化したユーザインタフェースの研究が別途必要である．

第 1 章で述べた通り，対話的 CG は，シミュレーション，レンダリングともに，エンジン化が進んでいる．今後も重要なシミュレーション法，レンダリング法は取り込まれていく可能性が高い．本論文で提案したビジュアルシミュレーション法が様々な応用につながることを期待する．

謝辞

本研究は，著者が慶應義塾大学大学院理工学研究科後期博士課程在学中に，同大学大学院理工学研究科開放環境科学専攻コンピュータサイエンス専修 藤代一成教授の指導のもとに行ったものです．ご指導を賜り，多くの発表機会を与えて下さいました藤代先生に感謝いたします．

本研究の基盤を確立させる間，定年退任されるまでご指導を賜りました，元同大学大学院理工学研究科開放環境科学専攻コンピュータサイエンス専修 大野義夫教授に感謝いたします．

また，同大学大学院理工学研究科開放環境科学専攻コンピュータサイエンス専修の 山本喜一教授，今井倫太准教授，同大学大学院理工学研究科空間・環境デザイン工学専修の高野直樹教授，同大学大学院理工学研究科開放環境科学専攻コンピュータサイエンス専修教員各位より貴重なご意見，ご指導を賜りました．厚く御礼を申し上げます．

評価にあたっては，慶應義塾大学理工学部機械工学科の横森剛専任講師に燃焼の観点の評価をいただき，株式会社フジテレビジョン 技術開発局開発業務センター設備運用部の新井清志氏，遠山健太郎氏，中山陽介氏からテレビ用の CG 制作の点から評価をいただき，株式会社オー・エル・エム・デジタル 研究開発部の安生健一氏，四倉達夫氏，藤堂英樹氏，徐才雨氏には，映画やアニメーションの CG 制作の点から評価をいただきました．お忙しい中，様々なご意見をいただきましたことを，この場を借りてお礼申し上げます．

本研究は文部科学省グローバル COE プログラム（環境共生・安全システムデザインの先導拠点）の支援を受けました．また、研究生活には田村淳記念大学院特別奨学金の援助を頂きました．謹んで感謝の意を表します．

参考文献

[Adams et al., 2007] Adams, B., Pauly, M., Keiser, R. and Guibas, L. J. (2007). Adaptively sampled particle fluids. *Proceedings of ACM SIGGRAPH 2007*, pages 48:1–48:8.

[Baukal Jr and Shwartz, 2001] Baukal Jr. C. E. and Shwartz. R. E. (2001). Combustion handbook. CRC Press.

[Becker and Teschner, 2007] Becker, M. and Teschner, M. (2007). Weakly compressible SPH for free surface flows. *Proceedings of Eurographics Workshop on Computer Animation 2007*, pages 209–217.

[Blinn, 1982] Blinn, J. (1982). Light reflection function for simulation of clouds and dusty surfaces. *Computer Graphics*, **16**(3):21–29.

[Chentanez and Müller, 2011] Chentanez, N. and Müller, M. (2011). Real-time eulerian water simulation using a restricted tall cell grid. *ACM Transactions on Graphics*, **30**(4):82:1–82:10.

[Chiba et al., 1994] Chiba, N., Muraoka, K., Takahashi, H., and Miura, M. (1994). Two dimensional visual simulation of flames. *The Journal of Visualization and Computer Animation*, **5**:37–53.

[Crane et al., 2007] Crane, K., Liamas, I., and Tariq, S. (2007). Real-time simulation and rendering of 3D fluids. *GPU Gems3*, pages 633–675. Addison-Wesley.

[Crytek, 2004] Crytek (2004). Cry Engine. http://mycryengine.com/

[Desbrun and Gascuel, 1995] Desbrun, M. and Gascuel, M. (1995). Smoothed particles: A new paradigm for animating highly deformable bodies. *Proceedings of Eurographics Workshop on Computer Animation and Simulation 95*, pages 61–76.

[Dobashi et al., 2000] Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T., and Nishita, T. (2000). A simple, efficient method for realistic animation of clouds. *Proceedings of ACM SIGGRAPH 2000*, pages 19–28.

[Dobashi et al., 2004] Dobashi, Y., Yamamoto, T., and Nishita, T. (2004). Synthesizing sound from turbulent field using sound textures for interactive fluid simulation. *Computer Graphics Forum*, **23**(3):539–546.

[Epic Games, 1998] Epic Games (1998). Unreal Engine. http://www.unrealengine.com/

[Enright et al., 2002] Enright, D., Marshner, S., and Fedkiw, R. (2002). Animation and rendering of complex water surfaces. *ACM Transactions on Graphics*, **21**(3):736–744.

[Fedkiw et al., 2001] Fedkiw, R., Stam, J., and Jensen, H. (2001). Visual simulation of smoke. *Proceedings of ACM SIGGRAPH 2001*, pages 15–22.

[Feldman et al., 2003] Feldman, B. E., O'Brien, J. F., and Arikan, O. (2003). Animating suspended particle explosions. *ACM Transactions on Graphics*, **22**(3):708–715.

[Foster and Fedkiw, 2001] Foster, N. and Fedkiw, R. (2001). Practical animation of liquids. *Proceedings of ACM SIGGRAPH 2001*, pages 23–30.

[Foster and Metaxas, 1996] Foster, N. and Metaxas, D. (1996). Realistic animation of liquids. *Graphical Models and Image Processing*, **58**(5):471–483.

[Fuller et al., 2007] Fuller, A. R., Karishman, H., Mahrous, K., Hamann, B., and Joy, K. I. (2007). Real-time procedural volmetric fire. *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, pages 175–180.

[Gingold and Monaghan, 1977] Gingold, R. A. and Monaghan, J. J. (1977). Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, **181**:375–398.

[Green, 2008] Green, S. (2008). Particle-based fluid simulation. *Game Developers Conference 2008*.
http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_ParticleFluids.pdf

[Harada et al., 2007a] Harada, T., Koshizuka, S., and Kawaguchi, Y. (2007a). Sliced data structure for particle-based simulations on GPUs. *Proceedings of the 5th International Conference on Computer graphics and Interactive Techniques in Australia and Southeast Asia*, pages 55–62.

[Harada et al., 2007b] Harada, T., Koshizuka, S., and Kawaguchi, Y. (2007b). Smoothed particle hydrodynamics on GPUs. *Proceedings of Computer Graphics International 2007*, pages 63–70.

[Havok, 2000] Havok (2000). Havok Physics. http://www.havok.com/products/physics

[Hong et al., 2007] Hong, J. M., Shinar, T., and Fedkiw, R. (2007). Wrinkled flames and cellular patterns. *ACM Transactions on Graphics*, **26**(3):47:1–47:6.

[Horvath and Geiger, 2009] Horvath, C. and Geiger, W. (2009). Directable, high-resolution simulation of fire on the GPU. *ACM Transactions on Graphics*, **28**(3):41:1–41:8.

[Ihm et al., 2004] Ihm, I., Kang, B., and Cha, D. (2004). Animation of reactive gaseous fluids through chemical kinetics. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium*, pages 203–212.

[Inakage, 1990] Inakage, M. (1990). A simple model of flames. *Proceedings of Computer Graphics International '90*, pages 71–81.

[Kaufman, 1991] Kaufman, A. (1991). Volume Visualization (Tutorial), *IEEE Computer Society Press*.

[Kajiya and Von Herzen, 1984] Kajiya, J. T. and Von Herzen, B. P. (1984). Ray tracing volume densities. *Computer Graphics*, **18**(3):165–174.

[Kang et al., 2007] Kang, B., Jang, Y., and Ihm, I. (2007). Animation of chemically reactive fluids using a hybrid simulation method. *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2007*, pages 199–208.

[Kawada and Kanai, 2011] Kawada, G. and Kanai, T. (2011). Procedural fluid modeling of explosion phenomena based on physical properties. *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2011*, pages 167–175.

[Kipfer and Westermann, 2006] Kipfer, P. and Westermann, R. (2006). Realistic and interactive simulation of rivers. *Proceedings of Graphics Interface 2006*, pages 41–48.

[Koshizuka and Oka, 1996] Koshizuka, S. and Oka, Y. (1996). Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nuclear Science and Enginnering*, **123**:421–434.

[Losasso et al., 2004] Losasso, F., Gibou, F., and Fedkiw, R. (2004). Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics*, **23**(3):457–462.

[Losasso et al., 2006] Losasso, F., Irving, G., Guendelman, E., and Fedkiw, R. (2006). Melting and burning solids into liquids and gases. *IEEE Transactions on Visualization and Computer Graphics*, **12**(3):343–352.

[Lucy, 1977] Lucy, J. B. (1977). A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, **82**:1013–1024.

[Melek, 2007] Melek, Z. (2007). *Interactive Simulation of Fire, Burn and Decomposition*. PhD thesis, Texas A&M University.

[Müller et al., 2003] Müller, M., Charypar, M., and Gross, M. (2003). Particle-based fluid simulation for interactive applications. *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2003*, pages 154–159.

[Müller et al., 2005] Müller, M., Solenthaler, B., Keiser, R., and Gross., M. (2005). Particle-based fluid-fluid interaction. *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2005*, pages 237–244.

[Nguyen et al., 2003] Nguyen, D. Q., Enright, D., and Fedkiw, R. (2003). *Simulation and Animation of Fire and Other Natural Phenomena in the Visual Effects Industry*. http://physbam.stanford.edu/ fedkiw/papers/stanford2003-11.pdf

[Nguyen et al., 2002] Nguyen, D. Q., Fedkiw, R., and Jensen, H. W. (2002). Physically based modeling and animation of fire. *ACM Transactions on Graphics*, **21**(3):721–728.

[Nguyen, 2004] Nguyen, H. (2004). Fire in the "vulcan" demo. In *GPU Gems*, chapter 6. Addison-Wesley.

[NVIDIA, 2006] NVIDIA, NVIDIA CUDA, http://developer.nvidia.com/category/zone/cuda-zone

[NVIDIA, 2008] NVIDIA, NVIDIA PhysX, http://developer.nvidia.com/physx

[Perlin, 2002] Perlin, K. (2002). Improving noise. *ACM Transactions on Graphics*, **21**:681–682.

[Pfaff et al., 2010] Pfaff, T., Thuerey, N., Cohen, J., Tariq, S., and Gross, M. (2010). Scalable fluid simulation using anisotropic turbulence particles. *ACM Transactions on Graphics*, **29**(6):174:1–174:8.

[Premoze et al., 2003] Premoze, S., Tasdizen, T., Bigler, J., Lefohn, A., and Whitaker, R. T. (2003). Particle-based simulation of fluids. *Computer Graphics Forum*, **22**(3):401–410.

[Reeves, 1983] Reeves, W. T. (1983). Particle systems – a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, **2**(2):91–108.

[Sakamoto et al., 2007] Sakamoto, N., Nonaka, J., Koyamada, K., and Tanaka, S. (2007).

Particle-based volume rendering. *Proceedings of Asia-Pacific Symposium on Visualization 2007*, pages 129–132.

[Schaufler, 95] G. Schaufler. (1995). Dynamically generated impostors. *Modeling Virtual Worlds - Distributed Graphics, MVD Workshop*, pages 129–136.

[Selle et al., 2005] Selle, A., Rasmussen, N., and Fedkiw, R. (2005). A vortex particle method for smoke, water and explosions. *ACM Transcations on Graphics*, **24**:910–914.

[Solenthaler and Pajarola, 2009] Solenthaler, B. and Pajarola, R. (2009). Predictive-corrective incompressible SPH. *ACM Transactions on Graphics*, **28**(3):40:1–40:6.

[Solenthaler et al., 2007] Solenthaler, B., Schläfli, J., and Pajarola, R. (2007). A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds*, **18**:69–82.

[Stam, 1999] Stam, J. (1999). Stable fluids. *Proceedings of ACM SIGGRAPH 99*, pages 121–128.

[Stam and Fiume, 1995] Stam, J. and Fiume, E. (1995). Depicting fire and other gaseous phenomena using diffusion processes. *Proceedings of ACM SIGGRAPH 95*, pages 129–136.

[Takeshita et al., 2003] Takeshita, D., Ota, S., Tamura, M., Fujimoto, T., Muraoka, K., and Chiba, N. (2003). Particle-based visual simulation of explosive flames. *Proceedings of Pacific Graphics 2003*, pages 482–486.

[Teschner et al., 2003] Teschner, M., Heidelberger, B., Müller, M., Pomeranets, D., and Gross, M. (2003). Optimized spatial hashing for collision detection of deformable objects. *Proceedings of Vision, Modeling, and Visualization*, pages 47–54.

[Umenhoffer et al., 2006] Umenhoffer, T., Szirmay-Kalos, L., and Szijártó, G. (2006). Spherical billboards and their application to rendering explosions. *Proceedings of Graphics Interface 2006*, pages 57–63.

[Unity Technologies, 2005] Unity Technologies. Unity. http://unity3d.com/

[Zhongming et al., 2010] Zhongming, D., Kawamura, T., Sakamoto, N., and Koyamada, K. (2010). Particle-based multiple irregular volume rendering on CUDA. *Simulation Modelling Practice and Theory*, **18**:1172–1183.

[Zhu et al., 2010] Zhu, J., Liu, Y., Bao, K., Chang, Y., and Wu, E. (2010). Realtime simulation of burning solids on GPU with CUDA. *Proceedings of IEEE International Conference on Computer and Information Technology 2010*, pages 1219–1224.

[井村 他, 2010] 井村 誠孝, 稲垣 智, 池田 聖, 眞鍋 佳嗣, 大城 理, 千原 國宏 (2010). 粒子ベース流体シミュレーションに基づく炎のリアルタイムレンダリング. **第 15 回日本バーチャルリアリティ学会大会論文集**, pages 560–563.

[小山田, 坂本, 2010] 小山田 耕二, 坂本 尚久 (2010). **粒子ボリュームレンダリング**. コロナ社.

[桑名 他, 2009] 桑名 一徳 , 森下 聡, 土橋 律 (2009). 火災旋風発生時の火炎高さについて―実験室規模の軸対称火災旋風の場合―, 日本燃焼学会誌, **51**:56-62.

[越塚, 2005] 越塚 誠一 (2005). **粒子法**. 丸善.

[越塚, 2008] 越塚 誠一 (2008). **粒子法シミュレーション – 物理ベース *CG* 入門**. 培風館.

[竹下 他, 2004] 竹下 大樹, 大田 真, 田村 真智子, 藤本 忠博, 村上 一信, 千葉 則茂 (2004). 爆発火炎の粒子ベースビジュアルシミュレーション法. **芸術科学会論文誌**, **3**(2):159–167.

発表文献

[間淵 他, 2011a] 間淵 聡, 藤代 一成, 大野 義夫 (2011a). SPH ベースリアルタイム火炎シミュレーション. **情報処理学会論文誌**, **52**(10):2965–2972.

[間淵 他, 2011b] 間淵 聡, 藤代 一成, 大野 義夫 (2011b). 粒子ベースリアルタイム火炎レンダリング. **画像電子学会誌**, **40**(4):541–548.

[Mabuchi et al., 2011] Mabuchi, S., Ohno, Y., and Fujishiro, I. (2011). SPH-based method for interactive flame simulation. *Proceedings of ASIAGRAPH 2011*, pages 36–41.