# 陰関数曲面陰関数曲面と流体の同時リアルタイムレンダリング

高石圭人[†1]

**概要**：本研究では、陰関数曲面と流体を前後関係が正しい状態で、同時にリアルタイムレンダリングすることを目的とする．粒子法を用いた流体シミュレーションに対して陰関数曲面を障害物として適応させることがある．このシーンにおいて、陰関数曲面、流体それぞれ単体でレンダリングすることは実現されている．しかし、陰関数曲面と流体をリアルタイムで同時にレンダリングすることは実現されていない．本研究では粒子法による流体シミュレーション結果である流体の座標と流体を内包する容器となる任意の陰関数曲面を入力とし、陰関数曲面と流体を同時にリアルタイムでレンダリングするための GPU 上のアルゴリズムを提案することで、陰関数曲面と流体を前後関係が正しい状態で、同時にリアルタイムでレンダリングする目的を達成した．

## 1. はじめに

流体シミュレーションはゲームや映画ではコンピュータグラフィックスとして、工業解析や津波解析では数値シミュレーションとして使用されている．流体シミュレーションの中でも流体の振る舞いをシミュレーションする手法として粒子法[1-3, 7]という手法が存在する．粒子法は液体、煙などの連続体を離散化された有限個の粒子で表現し、連続体の動きを各粒子の速度、圧力、 密度などの物理量によって計算する方法である．この粒子法によるシミュレーション内での障害物を表現するために陰関数曲面モデルを用いる手法が存在する．陰関数曲面モデルとは f(x) = 0 で定義されている曲面である．従来では、障害物にポリゴンモデルを使用することがほとんどであったが、陰関数曲面を用いることによって、境界面でより自然なシミュレーション結果を得られるようになった．この技術は医療では複雑な形状で構成させる人体内部、工業においては繊細な要素を構成部品として持つ工業部品など、これらのように繊細な領域でのシミュレーションにおいて重要な技術となる．しかし、陰関数曲面を障害物とした場合のシミュレーションに対する写実的かつ、高速な陰関数曲面と流体の同時リアルタイムレンダリングは実現されていない．流体はメッシュとして表現され、障害物として表現される曲面は陰関数で定義されており、それぞれの3次元形状の表現手法が異なる．この点がそれぞれの要素を適切に合成し、写実的かつ高速にリアルタイムレンダリング出来ていないことに通じる．

流体シミュレーションを行うための手法は大きく分けて格子法[3-5]と粒子法の2つが存在する．格子法は空間を有限の格子aに分割し、格子点上、格子領域上でシミュレーションの計算を行う手法である．それに対して粒子法は流体を有限の粒子の運動として離散化する手法である．一般的に格子法の方が粒子法よりも処理速度においては優れている．一方で粒子法は移流項による数値拡散が生じないため、流体表面を特別な手法を用いずとも追跡することが出来ることが利点として上げられる．粒子法には Moving Particle Semi-implicit (以下 MPS)法[1,2]と Smoothed Particle Hydrodynamics(以下 SPH)法[7]がある．MPS 法は非圧縮流体を解くために提案された手法で工業の分野で広く使用されている．SPH 法は MPS 法よりも計算コストは低いが、非圧縮流体を解くために定案された手法ではない．そのため、圧縮性を下げ、非圧縮に近い状態として流体を扱う．粒子法によって流体の状態を解析し、液体表面をレンダリングする手法はすでにいくつか存在する．手法は大きく分けて粒子をメッシュとして表現する方法とメタボールとして表現する 2 つ の方法が存在する．流体をメッシュとして表現する1つ目の方法 は Müller らによる流体表面のレンダリング[6]である．この研究 では粒子法の1種類である SPH 法の解析結果を元に、Point Splatting 法[8]と Marching Cubes 法[9]の二つが使用されている．Point Splatting 法は法線情報を持つポイントの描画に使用 され、各点に対して四角形、楕円、正円を定義し、重複させることで点同士の隙間を補完する方法である．これに対して Marching Cubes 法は空間をセルに分割し、スカラー場の等値 面を形成するように三角形メッシュを生成することで流体をポリ ゴンメッシュとしてレンダリングする方法である．粒子数が少ない 場合、Point Splatting 法では粒子間の隙間が見えてしまう事がある．Marching Cubes 法でのレンダリングでは処理速度は下がってしまうが、この問題を解決することが出来た．しかし、一般的に Marching Cubes 法には空間を分割するグリッドの解像度によってはアーティファクトが発生してしまうことや流体の飛沫などグリッドよりも完全に小さいオブジェクトを評価し損なう可能性があることが問題点としてあげられている．2つ目は van der Laan らによって提案された Screen Space Fluid Rendering(以下 SSFR) 法[10]である．この方法ではレンダリングするカメラから見えている流体表面のみを対象とし、レンダリングを行う．粒子半径を元に流体の厚みを計算し、厚みに応じ

†1 慶應義塾大学大学院 政策・メディア研究科

© 2022 Information Processing Society of Japan

て色を決定することで流体の厚みを表現している. また、流体表面の平滑化には従来使用されていた Gaussian Smoothing を使用するのではなく、screen space curvature flow と呼ばれる曲率を最小化する手法を用いることでより滑らかな表面を生成している. そして流体表面部分のみをポリゴン化なしでレンダリングすることによって、グリッドの解像度によるアーティファクトに悩まされずかつ高速にリアルタイムレンダリングすることを可能とする手法である. 流体をメタボールとして表現する方法には Kanamori らによる方法[11]がある. この手法では流体表面を構築するために必要となる流体の密 度の等値面をレイトレーシングによって抽出する. この手法では、 流体表面のポリゴン化を行わないので、グリッドの解像度によるアーティファクトを発生させずに、流体表面を正確に生成し、写実性の高いレンダリング結果を得ることが出来る. しかしリアルタイムでは粒子数が増えるにつれて計算の負荷が高くリアルタイムでのレンダリングは難しい.

粒子法を用いた流体シミュレーションの課題として障害物付近での流体の振舞いを決定する計算モデルの開発が課題として存在する. 具体的な問題点として、障害物から流体へ働く影響の精度の向上と障害物の表現方法が上げられる. SPH 法や MPS 法[1,2]は障害物を粒子で表現することで、それらのアルゴリズムにそのまま障害物を組み込めるように設計させていた. 粒子を用いて障害物を十分な精度で表現しようとすると粒子の数が増えてしまい、計算時間の増加、メモリコストも増えてしまうことが問題点として上げられていた. この問題を解決するため、障害物を粒子ではなくポリゴンで表現するポリゴンモデル[12]が開発された. これにより計算量を抑え、連続的な障害物を表現することが可能となった. そして、コンピュータグラフィックスに分野ではポリゴンは一般的な物体の表現形式とされているため、シミュレーションにモデルを組み込み易い. しかし、ポリゴンは頂点座標と接続順によって表現がなされているため、なめらかさにおいては問題が残り、衝突判定に誤差が生じる. また、障害物が変形する場合、ポリゴンを再度形成し直さなければならない. これらの問題を解決するために障害物を陰関数で表現する手法[13, 14]が存在する. 陰関数を用いて障害物を表現することによって、ポリゴンモデルよりも滑らかで連続的な障害物を表現することが出来るようになり、流体との境界面においてより精度の高いシミュレーションを行えるようになった. また、時間に依存して変化するような障害物とのシミュレーションも可能となった. Sakamoto らの研究[13]では SPH 法を、Kanetsuki らの研究[14] では MPS 法を拡張することによって陰関数曲面で表現された 障害物との境界面における流体の振舞いを計算するモデルが提案されている. どちらの手法でも境界部分におけるシミュレーションを独立させ、境界部分の粒子から障害物である陰関数曲 面までの最単距離と陰関数曲面上の法線を用いることによって計算を行っている.

陰関数曲面を生成するための手法には様々なアプローチ[15-19]が存在する. Turk らによって Turk-O'Brien 法[15]が提案された. この手法ではレーザスキャナを使用して得た点群の座標と法線を入力とし、連立一次方程式を解くことで、物体の曲面を表現する形状関数を得ることが出来る. しかし、この点群の数が増えるにつれて解くべき連立一次方程式が膨大となり、計算負荷、メモリコストがかかってしまう事が問題としてあげられている. 大量点群に対して形状関数を生成するため、Multilevel Partitional of Unity Implicit(MPU)[16]が提案された. MPU 法は局所的に形状関数を算出することによって大規模な方程式を解くことを回避し、高速性を維持しつつ形状関数を生成することが出来る. しかし問題点として点群密度が密な状態でないと曲面を生成できない問題を抱えている. また、近年では点群特徴量抽出モデル[20]を用いて得た特徴量をもとに陰関数曲面を生成する[18], [19] などの深層学習を用いたモデルも存在している.

生成された陰関数曲面をレンダリングする手法も様々なアプローチが存在する. それらの方法は大きく分けて直接的レンダリングと間接的レンダリングの2種類が存在している. 直接的レンダリングでは、3次元形状計測機から得られた座標とカメラからの仮想的なレイの屈折や反射をもとに写実的かつ滑らかな曲面をレンダリングする手法である. 直接的レンダリングの中にはカメラから各画素へのレイと描画対象との交点を求め、レイの屈折や反射を用いることで写実的で高品質な結果を得られるレイトレーシング法[17]やレイ方向にサンプリング点を移動していき、 陰関数値の正負の値によって陰関数曲面の表面を算出するレイマーチング法[21]がある. これに対して間接的レンダリングでは陰関数曲面をポリゴンに変換してからレンダリングを行う手法である. 先に述べた Marching Cubes 法は間接的レンダリングの代表的な手法である. また、陰関数曲面モデルを高速レンダリングする手法も提案されている. B-スプラインを利用して陰関数曲面を生成し, それをリアルタイムでレンダリングしている Wei らの手法[22]や、スカラー場を区分的多項式で近似しておき, GPU を利用したレイマーチング法によってレンダリングしている Nakata らの手法[23]の手法が存在する.

本研究の目的は陰関数曲面と流体を適切に合成し、リアルタイムでレンダリングを行うことである. 入力として粒子法を用いた 流体のシミュレーション結果である座標と容器となる任意の陰関数曲面を与え、それらを合成したレンダリング結果を出力する. この目的を達成するために解決しなければいけない課題は、流体と陰関数曲面の前後判定を適切に行い、それらの色を適切に合成することである. そしてそれらの処理をリアルタイムで実 行できるような GPU 上で作動するアルゴリズムを提案することである. 本研究の目的である陰関数曲面と流体の同時リアルタイムレンダリングを実現することによって、複雑な形状を持つ臓器、工業製品、芸術品や、時間によって変形する物体を障害物とした流体シミュレーションのシーンが扱われる工業、医療、芸術などの幅広い分野に貢献することができる. また、近年では iPhone に搭載されている LiDAR などの Depth センサーから手軽に点群を入手できるようになり、今後、陰関数曲面がよ

© 2022 Information Processing Society of Japan

り普及することが考えられる. このような実世界から得られた陰関数曲面のモデルに対して流体のエフェクトなどをリアルタイムで適応出来るようになることはゲームや xR によるエンタメコンテンツに対する 体験の向上にもつながると考えられる.

## 2. 提案手法

本研究では、流体のレンダリングには SSFR、陰関数曲面のレンダリングにはレイマーチング法を用いる. そして、陰関数曲面をレンダリングするためのレイマーチング法をどのようにして流体をレンダリングするための SSFR 法に適応させ、適 切な前後関係を考慮し、陰関数曲面と流体を同時リアルタイムレンダリングする方法について述べる.

### 2.1 提案するレンダリングパイプライン

本研究では、図 1 の流れで既存の流体レンダリング手法である SSFR 法に対して、レイマーチング法を用いてレンダリングした陰関数曲面を合成する. これを実現するためには解決すべき問題がある. それはそれぞれの異なるレンダリング手法によってレンダリングされた3次元物体をどのように前後判定を行うかということである.

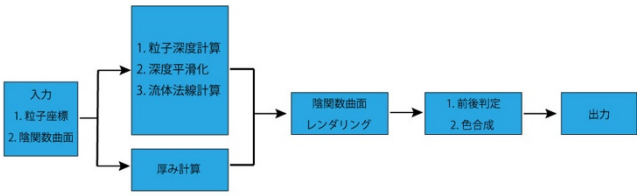


図 1 提案レンダリングパイプライン

### 2.2 GPU 上での前後判定に関する問題点

陰関数曲面と流体を適切に合成するためにはそれぞれの前後判定を行う必要がある. 本項では前後判定を行うにあたり、解決すべき問題点について述べる. GPU 上で Ray Marching 法を用いて陰関数をレンダリングするためは、はじめにレンダリングするスクリーンを覆うような長方形 ポリゴンを用意する必要がある. このポリゴンに対してカメラからそれぞれのピクセルに対してレイを飛ばし Ray Marching 法の処理を行う. このレンダリングを俯瞰した図が図2である. ここで注目しなければいけない点は、陰関数曲面は3次元形状オブジェクトとして存在するのではなく、長方形にカメラから見えている 3 次元形状の陰関数曲面が投影されているということである. それに対して、流体粒子は図3のように3次元空間上に3次元的に存在する. 陰関数曲面の深度情報はカメラから長方形までの距離となってしまうので、図4のようになってしまい正確に前後判定を行うことができない. CPU 上で適切に前後判定を行う場合、陰関数曲面と流体粒子までの深度を互いに 2 次元配列などのバッファに格納し、逐次的に比較することでこの問題を解決することが出来る.

しかし、GPU 上でこの前後判定を行う場合、 VertexShader や FragmentShader などと異なり深度比較部分の処理はプログラマに開放されていない. そのため、適切なタイミングでバッファに深度を書き込み、ハードウェア側に深度比較を任せ、実行させなければいけない. この比較する際に用いる 深度を書き込むタイミングが GPU 上で前後判定を行う上での課題となる.

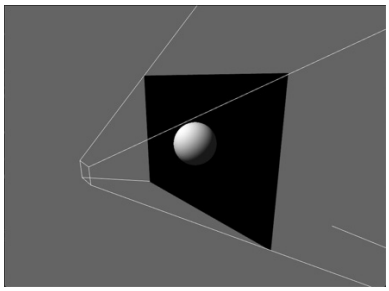

図 2 陰関数曲面のレンダリング俯瞰したイメージ

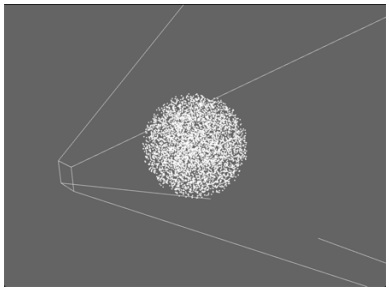

図 3 粒子のレンダリングを俯瞰したイメージ

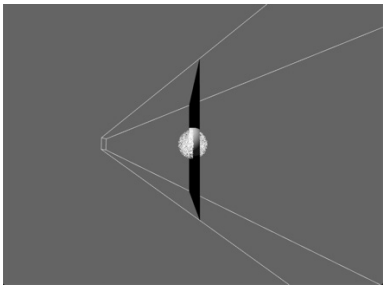

図 4 粒子と陰関数曲面の合成を俯瞰したイメージ

### 2.3 前後判定の実行タイミングに関する問題点

レンダリングパイプラインにおいて、GPU の基本動作を図 5 に 示す. 本来物体が重なって投影される際、すでに投影されている物体よりもさらに奥に投影される物体に関しては画素値を計 算する処理を実行する必要がないため、最適化のために投影 処理の後に早期的な前後判定(以下、早期 Z)が実行される. こ の早期 Z により、すでに描かれている物体よりも手前に投影される場合のみ画素値を決定されるプログラムが実行される. しかし、半透明な流体や陰関数局面で表現された容器の色を合成する場合、すでに投影されている物体の奥に投影される場合でも色合成を行う為に画素値を決定する処理を実行する必要がある.

© 2022 Information Processing Society of Japan


図5 早期的な前後判定を無効にした際の GPU 動作

### 2.4 解決法

先に述べた問題を解決した上で、陰関数曲面と流体を適切に合成し、リアルタイムでレンダリングするという目的を達成するための手法の提案をする. 第1に早期 Z に関しての解決策を述べる. 早期 Z が実行されるためにはフラグメントシェーダー内部で深度の変更処理や、discard(ピクセル破棄)処理が行われないことが条件である. そのため、そのどちらかの処理を意図的に実行することにより早期 Z を機能させないようにすることが出来、深度テストで不合格となった物体のピクセル処理も実行出来るようになる. このような状況になった場合の GPU 内での処理を図 6 に示す. そして、次に流体と陰関数曲面の適切な前後判定に関する問題である. これを解決するにはカメラから陰関数曲面の表面までの深度を算出し、FBO の Depth バッファへ書き込む必用がある. この表面までの深度は Ray Marching 法の処理内で求められているカメラからのレイの長さに相当するため、この値を使用することが出来る. これらの2つの解決法を組み合わせると、フラグメントシェーダー内部で陰関数曲面の表面までの深度を書き込むということになる. これを実行することに2つの問題を解決することが出来る. そして流体と陰関数曲面のカラーブレンドについては式(1)の処理を用いている. **α**は配合率で あり、$\mathbf{0 \le \alpha \le 1}$である.

$$\boldsymbol{\alpha} \cdot \boldsymbol{Color}_{implicit\ surface} + (\mathbf{1.0} - \boldsymbol{\alpha}) \cdot \boldsymbol{Color}_{fluid} = \boldsymbol{Color}_{out} \quad (1)$$


図 6 早期 Z が実行されないパイプライン

## 3. 実験

本章では、前章で提案した手法を利用して、様々な条件の元レンダリングを行った. レンダリングを行った計算機の環境を以下に示す.

| | |
|---|---|
| プロセッサ | Core i7 7700 |
| GPU | GTX 1060 |
| 実装 RAM | 16GB |
| Graphics API | openGL |

### 3.1 レンダリング結果

本節ではレイマーチング法でレンダリングされた陰関数曲面モデルの結果と SSFR 法によってレンダリングされた流体モデル の結果を合成した結果を示す. また、流体を表現する粒子数の 数、流体粒子を表現する粒子の大きさを変更した際の結果を示 す. 粒子数 60 万、80 万、110 万、粒子の大きさ 0.02、0.04、 0.06、0.08 の場合について図 7-図 9 に結果を提示する. ちなみ に粒子の大きさはレンダリングするスクリーンの縦幅を1.0 とした時の大きさである

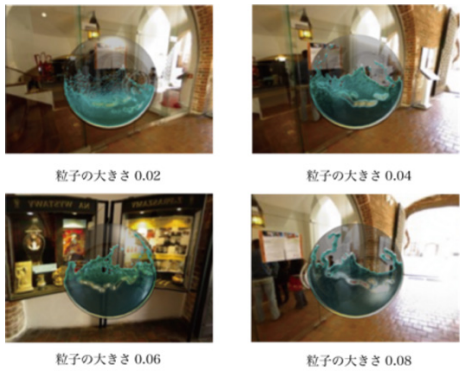


図 7 粒子数 60 万に関するレンダリング結果

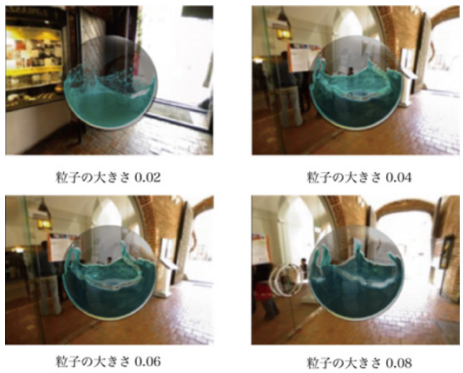


図 8 粒子数 80 万に関するレンダリング結果

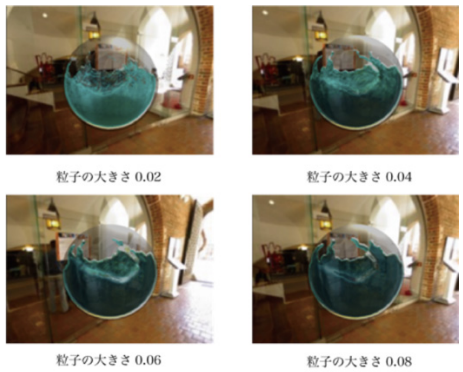


図 9 粒子数 110 万に関するレンダリング結果

### 3.2 レンダリング時間計測

レンダリング時における時間計測を比較する. 粒子数は 80 万 粒子、粒子の大きさも同じく 0.02, 0.04, 0.06, 0.08 のパラメータ時における時間計測を行った. また、陰関数曲面のみをレンダリングした場合、流体のみをレンダリングした場合、陰関数曲面を追加した場合を比較する. これによって陰関数曲面を追加、そしてそれぞれの色を合成することでの負荷を計測する. 計測の方法としては 1 万フレームを描画するのにかかった時間を計測した上で平均を求め、1 フレームを描画する時間を求めている. 結果は図 10 と図 11 に示す.

© 2022 Information Processing Society of Japan

速度計測結果

| | 処理時間(MS) |
|---|---|
| 陰関数曲面のみ | 0.13 |
| 0.02 (流体のみ) | 14.23 |
| 0.02(流体+陰関数曲面) | 14.47 |
| 0.04 (流体のみ) | 26.33 |
| 0.04(流体+陰関数曲面) | 26.52 |
| 0.06 (流体のみ) | 44.83 |
| 0.06(流体+陰関数曲面) | 45.03 |
| 0.08 (流体のみ) | 70.03 |
| 0.08(流体+陰関数曲面) | 70.27 |

図 10 粒子数 80 万に関する速度計測結果

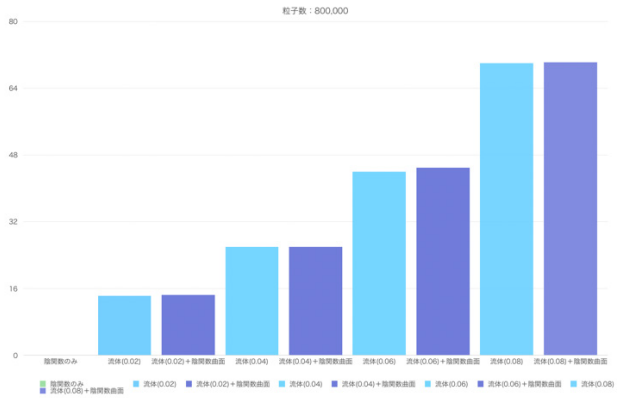


図 11 粒子数 80 万に関する速度計測結果

## 4. 結論

本研究では、陰関数曲面モデルと流体をリアルタイムレンダリングし、それらを適切にリアルタイムで合成することを目的とし、GPU 上のアルゴリズムを提案することによって、目的を達成した. 陰関数曲面のレンダリングにはレイトレーシング法の1種である、レイマーチング法を用いてレンダリングし、流体のレンダリングには SSFR 法を用いてレンダリングを採用した. 適切なタイミングで流体情報を保持している FBO に対して、陰関数モデルの深度値を書き込むことにより陰関数モデルと流体の合成を実現した. 実験結果で示されているように流体のみのレンダリングにかかる処理時間と陰関数曲面のみをレンダリングした処理時間を加えた計測時間と流体と陰関数曲面の同時レンダリング時ではほとんど差がない. そのため、流体単体のレンダリングに対して余分な負荷なしに陰関数曲面のレンダリングを加えることが出来たと言える. また、この際に生じる誤差は合成や前後判定に有する時間と考えられる. また、処理負荷において、処理時間は流体の粒子数よりも粒子の大きさに依存していることがわかる. この理由としては粒子の描画に対するドローコールは深度計算時のみであるため、粒子数が増えても負荷がかかるパスは1部である. それに対して粒子サイズを上げることによって描画されるピクセルの範囲が増大し、全てのパスでの計算量が増える、 つまり Shader 内での計算が実行されるピクセルが増加するということである. レンダリング結果については粒子の大きさは 0.02 では隙間が見えてしまい、流体らしくない. また、隙間については粒子数が少ないほど顕著に現れる. 流体らしいレンダリング 結果になっているのは粒子の大きさが 0.04 以上の結果である. リアルタイムレンダリングにおける最低フレームレートを 24fps とすると、十分に流体らしさを保ちつつ、リアルタイムで陰関数曲面と合成しレンダリングを実現出来ている.

GPU の発達によって、今まで計算負荷が高かったレイトレーシングをリアルタイムで行う手法が可能になり始めている. ゲーム業界でもこのリアルタイムレイトレーシングの技術は採用されている. しかし、カメラに映る全ての物体に関してレイトレーシングを用いたレンダリングはやはり処理負荷が重い. そのため、ガラス、水面、映り込みが起こるような物体など、レイトレーシングを用いることによって顕著に品質が変化するようなマテリアルに対してのみ部分的にレイトレーシングが用いられレンダリングされている. このようにマテリアルによってレイトレーシングとラスタライズを使い分けることによってパフォーマンスを向上させている. ラスタライズによるレンダリングでは正確な光線の経路シミュレーションすることが出来ないため、本研究では光の屈折や反射についての計算が正確ではない. そのため今後の展望としてリアルタイムでのレイトレーシングを用いたレンダリングを行うことによって品質の向上させることがあげられる.

## 参考文献

[1] “粒子法入門”、超塚誠一、柴田和哉、谷浩平、丸善出版、2014
[2] “粒子法”、越塚誠一、丸善出版、2005
[3] “CG のための物理シミュレーションの基礎”、 藤澤誠、マイナビ出版、2013
[4] Stam, Jos. “Stable fluids.” Proceedings of the 26th annual conference on Computer graphics and interactive techniques. 1999.
[5] Müller, Matthias, David Charypar, and Markus Gross. “Particle-based fluid simulation for interactive applications.” Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation. 2003.
[6] Müller, Matthias, David Charypar, and Markus Gross. “Particle-based fluid simulation for interactive applications.” Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation. 2003.
[7] Monaghan, Joe J. “Smoothed particle hydrodynamics.” Annual review of astronomy and astrophysics 30.1 (1992): 543-574.
[8] Zwicker, Matthias, et al. “Surface splatting.” Proceedings of the 28th annual conference on Computer graphics and interactive techniques. 2001.
[9] Lorensen, William E., and Harvey E. Cline. “Marching cubes: A high resolution 3D surface construction algorithm.” ACM siggraph computer graphics 21.4 (1987): 163-169.
[10] van der Laan, Wladimir J., Simon Green, and Miguel

© 2022 Information Processing Society of Japan

Sainz. “Screen space fluid rendering with curvature flow.” Proceedings of the 2009 symposium on Interactive 3D graphics and games. 2009.

[11] Kanamori, Yoshihiro, Zoltan Szego, and Tomoyuki Nishita. “GPU-based fast ray casting for a large number of metaballs.” Computer Graphics Forum. Vol. 27. No. 2. Oxford, UK: Blackwell Publishing Ltd, 2008.

[12] “SPH における壁境界計算手法の 改良”, 原田隆宏, 越塚誠一、情報処理学会論文誌, Vol.48, No. 4, pp. 1836- 1846, 2007

[13] Nakata, Susumu, and Yasuaki Sakamoto. “Particlebased parallel fluid simulation in three-dimensional scene with implicit surfaces.” The Journal of Supercomputing 71.5 (2015): 1766-1775.

[14] Kanetsuki, Yasutomo, and Susumu Nakata. “Moving particle semi-implicit method for fluid simulation with implicitly defined deforming obstacles.” Journal of Advanced Simulation in Science and Engineering 2.1 (2015): 63-75.

[15] Turk, Greg, and James F. O’brien. “Variational implicit surfaces.” Georgia Institute of Technology, 1999.

[16] Ohtake, Yutaka, et al. “Multi-level partition of unity implicits.” Acm Siggraph 2005 Courses. 2005. 173-es.

[17] Cook, Robert L., Thomas Porter, and Loren Carpenter. “Distributed ray tracing.” Proceedings of the 11th annual conference on Computer graphics and interactive techniques. 1984.

[18] Park, Jeong Joon, et al. “Deepsdf: Learning continuous signed distance functions for shape representation.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

[19] Chen, Zhiqin, and Hao Zhang. “Learning implicit fields for generative shape modeling.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

[20] Qi, Charles R., et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2017

[21] Perlin, Ken, and Eric M. Hoffert. “Hypertexture.” Proceedings of the 16th annual conference on Computer graphics and interactive techniques. 1989.

[22] Wei, Feifei, and Jieqing Feng. “Real-time ray casting of algebraic B-spline surfaces.” Computers & Graphics 35.4 (2011): 800-809.

[23] Nakata, Susumu, et al. ″Real-time isosurface rendering of smooth fields.″ Journal of visualization 15.2 (2012): 179-187.

© 2022 Information Processing Society of Japan