情報処理学会論文誌 Vol.54 No.8 2119–2130 (Aug. 2013)

粒子法の近傍粒子探索における連結リスト法への八分木の利用

笠 晃一[1,a)]

受付日 2012年11月6日, 採録日 2013年5月18日

概要：3 次元コンピュータグラフィックスにおける流体の表現では，物理シミュレーションが使用されることが多い．粒子法はそのような物理シミュレーションの一種であり，粒子を用いて流体を表現するアプローチの総称であるが，どの手法であれ，各粒子と相互作用する近傍粒子を探索する必要がある．この近傍粒子探索には，従来より格子を使用する方法が広く用いられてきたが，これは粒子が偏在するときにメモリ効率が悪化するという欠点がある．本研究では近傍粒子探索に八分木を用い，さらに，八分木における隣接セル探索アルゴリズムを効率化することで，探索時間の増加を最小限に抑えつつ，メモリ効率を改善することができた．

キーワード：コンピュータグラフィックス，流体シミュレーション，粒子法，近傍粒子探索，八分木

Utilization of Octree for Linked-list Method Used by Nearest Neighboring Particle Searching in Particle Methods

Koichi Ryu[1,a)]

Received: November 6, 2012, Accepted: May 18, 2013

Abstract: In 3D computer graphics, fluids are often represented by physical simulations. Particle methods are one type of such simulations and include all approaches that represent fluids with particles. All particle methods require searching for the nearest neighboring particles of every particle. For this particle searching, a traditional method which utilizes a grid is used extensively. While this method is convenient, it has the problem that memory efficiency deteriorates when particles are unevenly distributed. In this study, we adopt an octree for the particle searching and improve the efficiency of an algorithm for adjoining cell searching in an octree. The results show that memory efficiency is improved, while the increment of searching time is suppressed to a minimum.

Keywords: computer graphics, fluid simulation, particle method, nearest neighboring particle searching, octree

1. はじめに

近年，3 次元コンピュータグラフィックスが発達し，海の波，流れ落ちる滝，コップに注がれる水など，流体を使用した表現もよく見かけるようになってきた．これらの流体表現では物理シミュレーションも多用されており，写実的な表現を生み出すのに役立っている．このようなシミュレーションでは粒子法 [1] という物理学における数値計算法の一手法が用いられることが多い．これは，流体を粒子を利用して表現する手法であり，主に SPH（Smoothed Particle Hydrodynamics）法 [2] と MPS（Moving Particle Semi-implicit）法 [3] の 2 つが広く普及している．いずれの方法も，1 つの粒子がその近傍に存在する粒子と相互作用をするという仮定がなされているため，すべての粒子について近傍に存在する粒子を探索する必要がある．

近傍粒子の探索法として従来より，全ペア探索法 [4]（all-pair search method），連結リスト探索法 [5], [6], [7]（linked-

[1] 福岡工業大学情報工学部
Faculty of Information Engineering, Fukuoka Institute of Technology, Fukuoka 811–0295, Japan
[a)] ryu@fit.ac.jp

list search method)，木探索法 [8] (tree search method) などが知られている．しかしながら，全ペア探索法の計算量は N を粒子数とするとき $O(N^2)$ となり，決して効率の良い方法ではない．これに対し，連結リスト探索法の計算量は $O(N)$ であるので計算効率は良いが，粒子の分布に偏りが存在する場合に大量のメモリを必要とする．木探索法は全ペア探索法と連結リスト探索法の中間的な存在である．計算量は $O(N \log N)$ であり，連結リスト探索法と比べ計算時間はかかるが，使用するメモリは少なくて済む．この 3 種類の探索法のうち，実用上，最もよく使用されているのは連結リスト探索法であり，他の 2 つはほとんど使用されない．全ペア探索法は探索のための記憶領域をまったく使用しないので連結リスト探索法よりメモリ効率は良いのであるが，粒子数が多くなったときに処理に大量の時間を要する．実用的な規模のデータに対しては処理時間の膨大化から適用が困難なのである．同様に，木探索法も連結リスト探索法と比べメモリ効率は良いのであるが，やはり処理時間に難があり，木探索法の処理時間は連結リスト探索法の 3〜5 倍になるという実験結果 [8] が得られている．全ペア探索法ほどではないにしても，処理時間がここまで増加すると実用に供するのは困難であると思われる．

このように，処理時間の優位性のために連結リスト探索法が最もよく利用されているが，これは粒子数が多くなったときに大量のメモリを消費することが多い．そこで，我々は連結リスト探索法に比べて処理時間をできるだけ増加させずにメモリ消費量を抑える方法はないかと考えた．そこで新しく提案したいのが，連結リスト探索法と木探索法のハイブリッド的な手法である．この手法は，空間分割には八分木構造 [9] を用い，粒子の格納には連結リストを使用する．木探索法の省メモリ特性を生かしつつ，連結リスト探索法と同様の立方体セルを利用することで計算効率を上げている．なお，ハイブリッド的な手法が本研究の主目的であるが，八分木の隣接セルを求める部分が処理速度に大きく関係するので，この部分のアルゴリズムも新規に開発し，従来のもの [10] の 2 倍程度に高速化している．この結果，提案手法の処理時間を連結リスト探索法の 1.1〜1.2 倍程度にすることができた．また，本格的な実験の前に予備的な実験を実施したが，提案手法の消費メモリ量が木探索法に比べ 1/2 以下になるという結果も得られている．

以下，2 章で従来の近傍粒子探索法について概説する．3 章で我々の提案する近傍粒子探索法について述べ，その中で使用されている八分木の隣接セルを求めるアルゴリズムについて 4 章で説明する．5 章は提案手法の有効性を示すために行った実験の報告である．数値流体力学における標準実験であるダム崩壊実験とより実用に近い，乗用車が滝の中を通過するシーンを使用した実験に対する結果について議論する．実験は提案手法と連結リスト探索法の比較という形で行い，木探索法の実験を省いているが，これは上に述べているように木探索法が実用的ではないと考えられているからである．最後の 6 章は総括であり，本論文の内容を簡単にまとめた後，今後の課題を示す．

2. 従来の近傍粒子探索法

ここでは，まず粒子法について簡単に説明した後，全ペア探索法，連結リスト探索法および木探索法について説明する．

2.1 粒子法

粒子法は流体のような連続体を有限個の粒子によって近似し，連続体の動きを粒子の運動によって計算する方法である．したがって，1 個の粒子は連続体の小さな固まりに対応している．粒子法は一般の連続体を扱うことが可能であるが，コンピュータグラフィックスにおいては主として流体の表現に使用され，商用ソフトウェアの成功とも相まって，広く普及している．その特徴は，界面捕獲の簡便さと厳密な体積保存性である．他の差分法などの手法を用いたとき，界面捕獲や体積保存はやっかいな問題となる．

粒子法では，1 個の粒子は近傍にある粒子とのみ相互作用をするという仮定を置く．3 次元において近傍は球で表されるが，この球の半径を MPS 法の術語を用いて影響半径と呼ぶことにする．**図 1** において r_e が影響半径を表している．

2.2 全ペア探索法

これは，近傍粒子探索法の中で最も直接的かつ単純な方法である．すなわち，ある粒子が与えられたとき，これ以外のすべての粒子に対して粒子間距離を計算し，これが影響半径より小さければ近傍粒子の集合に加えるのである．これをすべての粒子に対して実行するので，N を全粒子数とするとき計算量は $O(N^2)$ となる．粒子数が多いと計算時間は非常に長くなるため，この方法は非常に小さいスケールの問題に対してのみ適用される．

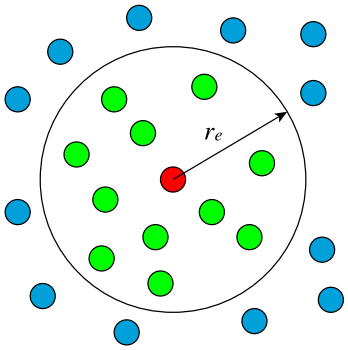

図 1 粒子法における相互作用領域
Fig. 1 Interaction domain in particle methods.

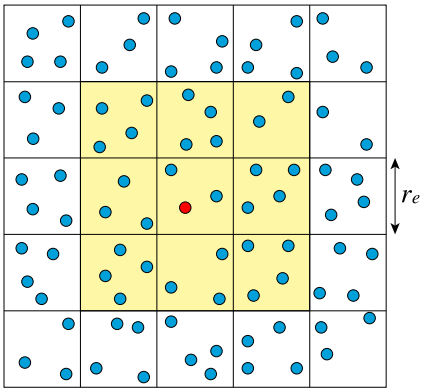

図 2 連結リスト探索法における格子
Fig. 2 A grid used in linked-list search method.

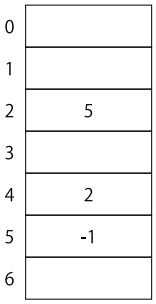

図 3 連結リストの配列による表現
Fig. 3 A representation of linked-list using an array.

2.3 連結リスト探索法

これは影響半径 r_e が一定のときによく使用される方法である．この方法では，問題領域が 3 次元のときは，粒子の存在する直方体領域を求めてから，その空間を格子によって一様に分割する．格子セルは立方体であり，一般に 1 辺の長さは影響半径 r_e に等しくとられる．このようにすれば，ある粒子の影響半径内に存在する粒子を求めるとき，その粒子の入っている格子セルに加え，そのセルに隣接する格子セルのみを探索すればよいからである．2 次元の場合の格子セルは正方形であるが，考え方は同様であり，2 次元における粒子分布を格子によって分割したときの様子を**図 2** に示す．図から明らかなように，2 次元の場合に探索すべき格子セルの数は 9 個であるが，3 次元の場合に探索すべき格子セルの数は 27 個になる．

各格子セルに入っている粒子は連結リストを用いて管理される．連結リストとして 2 進木リストを用いてもよいが，配列を用いて表現することもでき [5]，この方がメモリの節約にもなる．**図 3** は 3 個の粒子のリスト $(4, 2, 5)$ を配列を用いて表したものである．リスト内の数字は粒子番号である．添え字 4 の配列要素から始めて 2，5 の順にたどることができるが，添え字 5 の配列要素に入れられている -1 は終端を意味する．このように，粒子が連結リストによって管理されるので連結リスト探索法の名がある．しかしながら，連結リスト探索法の本質は連結リストを使用することよりも，むしろ空間を格子によって分割するところにある．実際，我々の提案する探索法も粒子の管理に連結リストを使用している．そこで，これ以降本論文では便宜上，連結リスト探索法を格子探索法と呼ぶことにする．

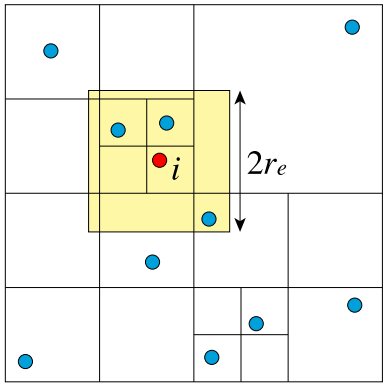

図 4 木探索法における四分木セル
Fig. 4 Quadtree cells used in tree search method.

格子探索法は，格子によって探索範囲が絞られるので計算量は $O(N)$ となり探索効率は良い．しかし，空間内に粒子が一様に分布していない場合や粒子が複数箇所に偏在している場合に，粒子を持たない格子セルが大量に発生することがあり，一般にメモリ効率は良くない．

2.4 木探索法

粒子法では影響半径が可変の場合がある．このような場合には格子探索法の探索効率が悪化し，代わりに木探索法が用いられることが多い．木として，2 次元の場合には四分木構造が，3 次元の場合には八分木構造が使用される．たとえば 2 次元の場合，この方法はまず粒子の存在する領域を 4 個のセルに均等に分割する．そして，この 4 個のセルのうち粒子を含むものに対して，さらに 4 個のセルに細分割する．これを再帰的に続け，最終的に各セルが 0 個または 1 個の粒子を含むようにするのである．2 次元における粒子分布を四分木セルによって分割したときの様子を**図 4** に示す．

木構造を構築したら次は近傍粒子の探索であるが，これは次のようにして実行する．すなわち，2 次元空間内の粒子番号 i の粒子の場合，この粒子を取り囲むように 1 辺の長さが影響半径の 2 倍に等しい正方形を考える．ただし，粒子 i が正方形の中心にくるようにする（図 4）．そして，この正方形に重なる細分割セルを求め，セル内に粒子があれば粒子間距離を計算して，これが影響半径より小さければ近傍粒子として登録するのである．3 次元の場合は正方形が立方体になるだけで，ほぼ同様な手続きとなる．なお，探索の計算量は $O(N \log N)$ となり全ペア探索法よりも効率が良い．また，粒子を含まないセルは細分割されないので格子探索法よりもメモリ効率が良い．ただし，1 章でも述べたように，影響半径 r_e を一定にした場合の実験によ

ると，木探索法は格子探索法よりも 3〜5 倍遅くなることが分かっている [8]．

3. 連結リストを用いた木探索法

我々の近傍粒子探索法も四分木や八分木を使用するが，次の点で従来の木探索法とは異なっている．(1) 葉となるセルはすべて同一の大きさを持ち，1 辺の長さが影響半径 r_e に等しい正方形か立方体である．(2) 葉となるセルには何個でも粒子を入れることができる．セル内の粒子は連結リストを用いて管理する．(3) 各粒子の近傍粒子はその粒子が属するセル自身とそのセルに隣接している葉のセルを探索することで求められる．隣接するセルを求めるアルゴリズムは次章で議論する．2 次元における粒子分布を四分木セルで分割したときの例を**図 5** に示す．

連結リストを用いた木探索法は上述のような特性を持つため，3 次元の場合は八分木における隣接セルを求めるだけでよい．これに対し，従来の木探索法は 1 辺の長さが影響半径の 2 倍に等しい立方体と交差するすべてのセルを見出す必要があり，それは隣接セルに限定されない．そして，このことが従来の木探索法の処理速度を低下させている原因の 1 つであると考えられる．また，我々の木探索法が計算領域を含む立方体を，1 辺の長さが影響半径に等しい立方体になるまで細分割するのに対し，従来の木探索法は各立方体がただ 1 個の粒子を含むようになるまで細分割しなければならない．最近接粒子間の平均距離は一般に影響半径よりも小さいので，従来の木探索法は我々の木探索法よりもセルを細かく分割する必要があり，消費メモリ量の点からも不利であると思われる．ただし，我々の木探索法が連結リスト用配列を必要とするのに対し，従来の木探索法はこれを必要としない．そこで，5 章のダム崩壊実験や乗用車と滝を使用する実験と同様の実験を予備的に両方の木探索法に対して実施してみた．その結果，従来の木探索法は我々の木探索法と比べ 2.0〜3.1 倍のメモリを消費することが明らかになった．

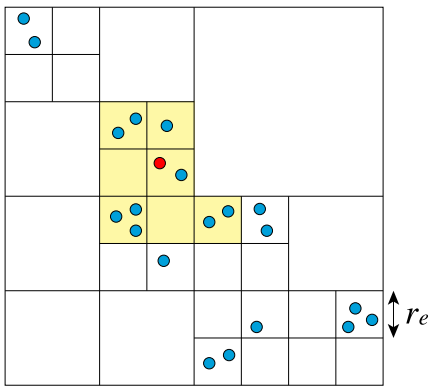

図 5 連結リストを用いた木探索法における四分木セル
Fig. 5 Quadtree cells used in tree search method using linked-list.

さて，次に連結リストを用いた木探索法のアルゴリズムを記述する．ここでは，3 次元空間の八分木に対するアルゴリズムを示すが，2 次元の場合もほぼ同様な手続きである．なお，木構造の構築と近傍粒子の探索を同時並行的に行うアルゴリズムを採用している．

アルゴリズム 1

(1) 粒子の存在する領域を 1 辺の長さが $2^n r_e$ に等しい立方体で覆う．ただし，r_e は影響半径であり，n はできるだけ小さな非負の整数である．
(2) 各粒子 i について以下を実行する．i は粒子番号である．
(3) 木を根の方からたどってゆき，その粒子が入るべき葉のセルがすでに存在するかどうか調べる．存在しないときは，その葉のセルおよび途中に必要な節点のセルを作成する．
(4) 粒子 i が属する葉のセルに対しそのセルに隣接している葉のセルも求め，これらのセルに含まれる粒子のうち影響半径内にある粒子のみを近傍粒子として登録する．また，逆にここで求めた粒子の近傍粒子として粒子 i を登録する．

後ほど，処理に必要なメモリ量が問題になるので，木を構成する節点の実装法についても述べておく．葉でない節点のうち最も下にあるものは大きさ 8 の整数型の配列で表現されている．そして，この配列の各要素が葉を表現している．すなわち，配列要素の中身が −1 であれば対応する葉のセルには粒子が含まれていないことを表し，そうでなければ配列によって実装された連結リストの最初の要素を表す．また，葉でない節点のうち最も下にないものは大きさ 8 のポインタ型の配列で表現されている．この配列要素の中身がヌルであれば対応する節点のセルは粒子を含まないことを表し，そうでなければこの節点の子節点へのポインタを表す．

4. 八分木の隣接セルを求めるアルゴリズム

隣接セルを求める問題は，八分木の研究において比較的早い時期から取り組まれている．たとえば，Gargantini [11] は四分木の隣接セルを求める簡潔なアルゴリズムを提案しているが，これは辺で隣接するセルだけを求めるものであり，頂点で隣接するセルを求めるのは不可能であった．隣接セルを求めるアルゴリズムは，これ以外にもいくつか発表されているが，その中でも八分木の隣接セルを完全な形で求めるアルゴリズム，すなわち面，辺，頂点のすべての方向に対して隣接セルを求めるアルゴリズムを提案したのが Samet [10] である．そして現在に至るまで，このアルゴリズムが主流として使用されているようである．Samet のアルゴリズムは 4 個の表を使用するが，これを実装すると

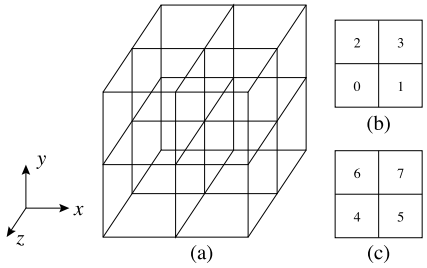

図 6 セルの位置に応じてつけられる番号
Fig. 6 The numbers of cells attached according to their positions in an octree.

4 個の 2 次元配列となる．また，面，辺，頂点のそれぞれに対して異なる手続きが用意されているが，これらはすべて再帰的呼び出しを使用しており，辺に対する手続きは面に関する手続きを，頂点に関する手続きは辺に関する手続きと面に関する手続きをそれぞれ呼び出している．

我々は Samet のアルゴリズムに対し，以下に示すような変更を加えたものを新たに提案したい．まず，Samet のアルゴリズムで使用されている 4 個の表の冗長部分を削減して 2 個の表にする．これは実装すると 2 個の 1 次元配列になるようなものである．すなわち，4 個の 2 次元配列を 2 個の 1 次元配列にする．このように表の方をまとめると，面，辺，頂点のそれぞれに対して用意されている手続きを 1 つにまとめることが可能となる．これにより，より一般化された記述が可能となるし．場合分けの個数も削減することができる．さらに，手続きが一体化されたことにより再帰的呼び出しの代わりにループを使用する手続きが容易に実現可能となる．このように，我々のアルゴリズムでは，配列の 1 次元化と場合分け個数の削減，そしてループの使用により Samet のアルゴリズムの高速化を試みている．なお，実際に Samet のアルゴリズムも実装し，簡単な実験により我々のアルゴリズムの方が 1.9〜2.0 倍速く処理できることを確認した．ここで注意しておきたいのは，我々のアルゴリズムが葉のセルに隣接する葉のセルを求めるものだという点である．そして，本研究で使用している八分木は葉のセルの深さがすべて同一という制約を持っている．しかしながらこれらの制約は本質的でなく，Samet [10] の場合と同様．これらの制約の除去はアルゴリズムのわずかな修正で可能になり処理速度もほぼ変わらないと思われるが，その議論は本研究の主題ではないので割愛する．

4.1 八分木内のセル位置の記述

八分木のセルは 3 次元デカルト座標内に置かれており，各辺が座標軸に平行になっているものと仮定する．このとき，セルを 8 等分してできる 8 個のセルに番号をつける．**図 6** (a) はセルを 8 等分したときの状態を表したものであり，同図 (b) は奥の方（z 軸の負方向）にある 4 個の分割セルにつけられた番号を，同図 (c) は手前の方（z 軸の正方向）にある 4 個の分割セルにつけられた番号をそれぞれ表している．

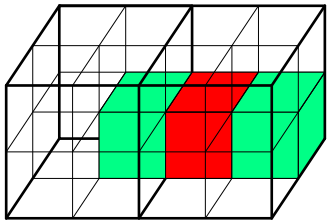

図 7 八分木セルと面で接するセルの例
Fig. 7 Example of cells which possess a face in common with an octree cell.

表 1 各八分木セルの右隣にあるセルの位置
Table 1 Position of the adjacent cell on the right of each octree cell.

セル位置	隣接セルの位置	1 つ上のレベル
0	1	なし
1	0	$+x$
2	3	なし
3	2	$+x$
4	5	なし
5	4	$+x$
6	7	なし
7	6	$+x$

4.2 面で隣接するセル

位置 0 にある八分木セルを考えよう．このセルは任意の深さにあるものとする．同一の大きさで，このセルの右隣（x 軸の正方向）にあるセルは親セルの中の位置 1 にあるセルである．したがって，セルの探索をこれ以上実行する必要はない．一方，同一の大きさで，このセルの左隣（x 軸の負方向）にあるセルはもし存在するならば，親セルの左隣にあるセルの中の位置 1 にあるセルである．ただし，親セルの左隣にあるセルは親セルと同一の大きさであるとする．したがって，このときは 1 つ上のレベルで，同一の大きさで左隣にあるセルを探索すればよいことになる．**図 7** に同一の大きさで右隣にあるセルと同一の大きさで左隣にあるセルを示している．

さて，探索方向を右隣に固定し，八分木セルのセル位置を変化させると，同様にして隣接セルの位置を求めることができ，求めたセル位置を表にすると**表 1** のようになる．ここに，「セル位置」というのは基準となる八分木セルの位置のことであり，「1 つ上のレベル」というのは 1 つ上のレベルでどのような探索をすればよいのかを表すものである．たとえば，$+x$ は x 軸の正方向（図 6 において右隣）を表しており，1 つ上のレベルで右隣にあるセルの探索をすればよいことを意味しているし，「なし」はセルの探索が終了したことを意味している．なお，表 1 は各八分木セルの右側の面に接するセルに関する表であるが，同様にし

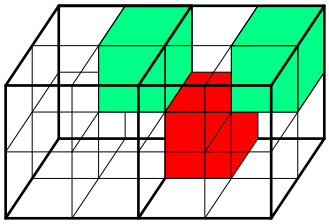

図 8 八分木セルと辺で接するセルの例
Fig. 8 Example of cells which possess an edge in common with an octree cell.

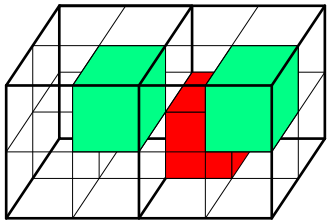

図 9 八分木セルと頂点で接するセルの例
Fig. 9 Example of cells which possess a vertex in common with an octree cell.

表 2 各八分木セルの右上にあるセルの位置
Table 2 Position of the adjacent cell in the upper-right direction of each octree cell.

セル位置	隣接セルの位置	1 つ上のレベル
0	3	なし
1	2	$+x$
2	1	$+y$
3	0	$+x+y$
4	7	なし
5	6	$+x$
6	5	$+y$
7	4	$+x+y$

表 3 各八分木セルの右かつ上かつ前の方向にあるセルの位置
Table 3 Position of the adjacent cell in the front upper-right direction of each octree cell.

セル位置	隣接セルの位置	1 つ上のレベル
0	7	なし
1	6	$+x$
2	5	$+y$
3	4	$+x+y$
4	3	$+z$
5	2	$+x+z$
6	1	$+y+z$
7	0	$+x+y+z$

て他の面に接するセルに関する表も作成することが可能である．

4.3 辺および頂点で隣接するセル

ここでも，位置 0 にある八分木セルを考える．同一の大きさで，このセルのすぐ右上（x 軸の正方向かつ y 軸の正方向）にあるセルは親セルの中の位置 3 にあるセルであり，セルの探索は終了する．また，同一の大きさで，このセルのすぐ左上（x 軸の負方向かつ y 軸の正方向）にあるセルはもし存在するならば，親セルの左隣にあるセルの中の位置 3 にあるセルである．したがって，このときは 1 つ上のレベルで，同一の大きさで左隣にあるセルを探索すればよい．ここで探索方向が左上から左に変化している点に注意されたい．なお，位置 0 にある八分木セルのすぐ右上にあるセルとすぐ左上にあるセルを**図 8** に示している．さらに，探索方向を右上に固定し，八分木セルのセル位置を変化させると，同様にして隣接セルの位置を求めることができ，求めたセル位置を表にすると**表 2** のようになる．ただし，$+x+y$ は探索方向が x 軸の正方向かつ y 軸の正方向であること，すなわち右上を表している．また，探索方向を変え，各八分木セルと他の辺で接するセルに関する表も作成することが可能である．

続けて，位置 0 にある八分木セルと頂点で接するセルを考えよう．同一の大きさで，探索方向として右かつ上かつ前（x 軸の正方向かつ y 軸の正方向かつ z 軸の正方向）にあるセルは親セルの中の位置 7 にあるセルであり，セルの探索はこれで終了する．また，同一の大きさで，探索方向として左かつ上かつ前にあるセルはもし存在するならば，親セルの左隣にあるセルの中の位置 7 にあるセルである．したがって，このときは 1 つ上のレベルで，同一の大きさで左隣にあるセルを探索すればよい．探索方向として右かつ上かつ前にあるセルと左かつ上かつ前にあるセルを**図 9** に示している．さらに，探索方向を右かつ上かつ前に固定し，八分木セルのセル位置を変化させると，同様にして隣接セルの位置を求めることができ，求めたセル位置を表にすると**表 3** のようになる．ただし，$+x+y+z$ は探索方向が右かつ上かつ前であることを表している．また，探索方向を変え，各八分木セルと他の頂点で接するセルに関する表も作成することが可能である．

4.4 表の一体化と隣接セル探索アルゴリズム

面で隣接するセルの位置に関する表は全部で 6 個できるし，辺で隣接するセルの位置に関する表は全部で 12 個できる．また，頂点で隣接するセルの位置に関する表は全部で 8 個できるから，隣接するセルの位置に関する表は全部で 26 個ある．そこで探索効率を向上させるため，これら 26 個の表を連結して 1 つの大きな表にし，さらにそれを元にして 2 個の配列を作成する．1 個目の配列は隣接セルの位置に関するもので，これは元の表の値をそのまま使用する．また，2 個目の配列は 1 つ上のレベルでの探索方向に関するものであるが，ここには探索方向の代わりに，その探索方向に関する情報が記述されている配列部分の最初の添え字を記入する．たとえば，探索方向が $+x+y$ 方向であり，$+x+y$ 方向に関する情報が配列の 200 から 207 の部分に

表 4 隣接セルの探索で使用される 2 個の配列（部分）
Table 4 Two arrays used in searching for adjacent cells (part of them).

添え字	0	1	2	3	4	5	6	7
pos	7	6	5	4	3	2	1	0
upper	0	72	24	96	8	80	32	−1
添え字	8	9	10	11	12	13	14	15
pos	3	2	1	0	7	6	5	4
upper	8	80	32	−1	8	80	32	−1

記述されているとすると，$+x+y$ の代わりに 200 と記入するのである．ただし，「なし」に対しては -1 を記入しておく．なお，表内の「セル位置」の項目は 0 から 7 までの繰返しになるので，これに対応する配列は作成しない．

表 4 に上で述べた 2 個の配列の最初の部分を示している．全体の配列は大きくなるので付録に掲載している．配列 pos が隣接セルの位置に関する配列であり，配列 upper が 1 つ上のレベルでの探索方向に関する配列である．なお，元となる 26 個の表の順番は探索アルゴリズムに影響しないが，ここでは便宜上探索方向に基づいて順番を決定しており，たとえば，表 4 は $-x-y-z$ 方向に関する情報と $-x-y$ 方向に関する情報を表している．そこで次に，これらの配列を用いて隣接セル探索アルゴリズムを記述する．八分木は再帰的な構造であるが，ここでは速度を重視して再帰呼び出しを用いていない．また，使用している配列はすべて 0 から始まる添え字を持っているものとする．

アルゴリズム 2

```
(1) for pp = 0 to 200 step 8（主ループ）
(2)    p = pp
(3)    q = NULL
(4)    for d = 0 to D − 1（上昇ループ）
(5)       k = p + index[d]
(6)       indexA[d] = pos[k]
(7)       p = upper[k]
(8)       if p < 0 then 上昇ループから抜ける end if
(9)    end for
(10)   if 途中で上昇ループから抜けた then
(11)      q = stack[d]
(12)      while q ≠ NULL and d > 0 do（下降ループ）
(13)         q = q が指すセルの child[indexA[d]]
(14)         d = d − 1
(15)      end while
(16)   end if
(17) end for
```

主ループの 1 回の反復で，求めるべき隣接セルの 1 つ上のレベルにあるセルへのポインタが q に求まり，隣接セルの八分木における位置が $indexA[0]$ に求まる．ただし，q が **NULL** のときは隣接セルが存在しないことを意味する．

上記アルゴリズムで D は八分木全体の深さを表している．また，処理対象となる葉のセルに関する経路情報は配列 *stack* および配列 *index* に保存されているものとする．すなわち，*stack* には各レベルにおけるセルへのポインタが，*index* には各レベルにおける八分木セルの位置がそれぞれ保存されているものとする．そして，上記アルゴリズムで各レベルにおける隣接八分木セルの位置を配列 $indexA$ に保存している．ただし，これらの配列の添え字 0 の要素に最も深いレベルの情報が保存され，添え字が増えるにつれ浅いレベルの情報が保存されるようにしている．さらに，3 章でも述べたように，葉でない節点のうち最も下にあるものは整数型の配列で表現され，最も下にないものはポインタ型の配列で表現されるが，これらの配列の名前をそれぞれ *hoc*，*child* としている．また，これらの配列の添え字は図 6 で示されるセル位置に対応している．したがって，2.3 節で述べた連結リストの配列表現における最初の添え字は $hoc[indexA[0]]$ によって求められるが，この値が -1 ならば隣接セル内に粒子が存在しないことを意味する．

5. 比較実験

連結リストを用いた木探索法（ここでは新しい木探索法と呼ぶことにする）を従来の格子探索法（連結リスト探索法）と比較するために実験を行った．使用したコンピュータは 2.8 GHz の Xeon と 4 GB のメモリ搭載した Mac Pro である．すでに，3 次元 CG ソフトウェア Maya（32 ビット）のプラグイン [12] として SPH による流体シミュレータを作成していたので，実験にはこれを使用することにした．すなわち，元々この流体シミュレータの近傍粒子探索アルゴリズムには格子探索法を用いていたのであるが，この部分を新しい木探索法と入れ替えることにより比較実験を実施した．なお，流体シミュレータを Maya のプラグインとして作成したのは，現場の人に実際に使用してもらい，現場の意見を研究に反映させるためである．この流体シミュレータは SPH の粒子を Maya のパーティクルとして表示することもできるし，各粒子に対してメタボールを配置しマーチングキューブによりポリゴンメッシュを生成することも可能である．また，粒子を流入させるだけでなく，任意のポリゴンメッシュ内に初期状態の粒子を配置することや壁粒子 [1] をアニメートさせることも可能になっている．なお，実験は，ダム崩壊実験と乗用車が滝の中を通過するシーンを使用した実験の 2 つを用意した．ダム崩壊実験は数値流体力学における標準的な実験であり，乗用車と滝を使用する実験はより実用に近い実験である．

5.1 ダム崩壊実験

これは水柱崩壊実験とも呼ばれ，水槽内に直方体の形状

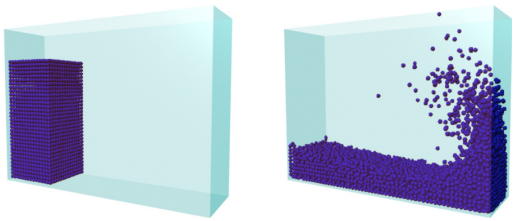

図 10 ダム崩壊実験．(a) 初期状態，(b) 崩壊後の状態
Fig. 10 Dam break experiment. (a) initial state, (b) state after break.

表 5 格子探索法と新しい木探索法における最大使用メモリ量
Table 5 Maximum memory consumption by grid search method and new tree search method.

粒子数	1,382	2,534	4,185	7,194
格子探索法 [kB]	3.96	6.44	10.6	17.0
新しい木探索法 [kB]	1.78	4.09	6.50	8.84
メモリ削減率 [%]	55.0	36.4	38.8	47.9
粒子数	14,644	24,570	47,164	100,080
格子探索法 [kB]	36.5	122	240	486
新しい木探索法 [kB]	17.3	29.7	62.0	115
メモリ削減率 [%]	52.7	75.6	74.2	76.4

をした水を置き，重力のみによってこれを崩壊させる実験である．3 次元 SPH を使用した場合の初期状態と崩壊後の状態を図 **10** に示す．

実験では，粒子数を 1,382 個から 100,080 個まで増加させつつ，格子探索法と新しい木探索法の両方に対し，近傍粒子探索に使用された最大メモリ量と平均処理時間を求めた．ここで，平均メモリ量ではなく最大メモリ量を計測しているのは，実用的な観点から見た場合に最大メモリ量の方が平均メモリ量よりも意味を持つからである．水の崩壊時間は約 0.667 秒間（80 フレーム）であるが，各ステップの時間間隔 Δt は可変であり，クーラン数 [1] がつねに 0.1 になるよう調整した．なお，表面張力はないものとし粘性係数を 0.2 Pa·s とした．粘性係数は実際の値より大きくしているが，これは粒子の運動を安定させるためである．

表 5 は近傍粒子探索に使用された最大メモリ量を表している．さらに，格子探索法に対する新しい木探索法のメモリ削減率も記述しているが，粒子数により約 35%から約 75%までかなり変動し，粒子数が増加するほど削減率も増加する傾向にあることが分かる．また，**表 6** は 1 回の近傍粒子探索にかかる時間の平均値を表している．さらに，格子探索法に対する新しい木探索法の処理時間増加率も計算しているが，最大でも 20%程度に抑制されていることが見て取れる．従来の木探索法の増加率が 200〜400%であることを考慮すると，かなり低く抑えられていることが分かる．なお，粒子数が増加したとき，処理時間の増加率が全体として減少傾向にある原因については 5.3 節で議論する．

表 6 格子探索法と新しい木探索法における平均処理時間
Table 6 Average time consumption by grid search method and new tree search method.

粒子数	1,382	2,534	4,185	7,194
格子探索法 [ms]	1.19	2.38	4.43	8.49
新しい木探索法 [ms]	1.40	2.83	5.29	9.75
処理時間の増加率 [%]	17.2	18.7	19.3	14.7
粒子数	14,644	24,570	47,164	100,080
格子探索法 [ms]	19.5	35.8	78.5	193
新しい木探索法 [ms]	22.1	41.2	88.6	214
処理時間の増加率 [%]	13.5	15.0	12.8	11.4

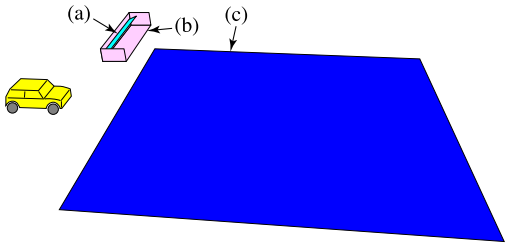

図 11 実験の配置図．(a) 流入口，(b) 容器，(c) 床
Fig. 11 Layout of used objects. (a) inflow gate, (b) container, (c) floor.

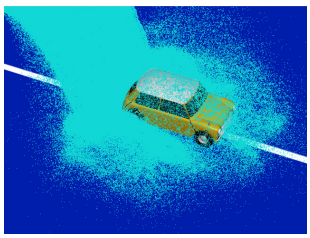

図 12 実験中のレンダリング画像
Fig. 12 Rendered image for an experimental scene.

5.2 乗用車と滝を使用する実験

乗用車が滝の中を通過するシーンに対する実験も実行した．実験における各オブジェクトの位置を図 **11** に示している．フレーム 1 においては壁粒子のみしかないが，時間の経過とともに水粒子が流入口より放出され，これらの水粒子はいったん片面が開いた容器に蓄積された後，滝状になって床の上に落下する．そして，このタイミングで乗用車が滝の中を通過するように乗用車の速度を調整している．図 **12** は滝の中を通過中の乗用車の画像であるが，水粒子はメタボールを使用せず単なる粒子としてレンダリングしているため写実性はない．なお，アニメーションの時間は 800 フレームであり，1 フレームは物理時間で約 0.833 ms に相当する．また，壁粒子の個数は 971,427 個で固定であるが，水粒子の個数は 1 フレーム目で 0 個，800 フレーム目で 963,300 個であった．

近傍粒子探索に必要なメモリ量をフレーム時間に対してプロットしたのが図 **13** である．格子探索法で必要なメモリ量が最大となるのは 800 フレーム目であり，361 MB が

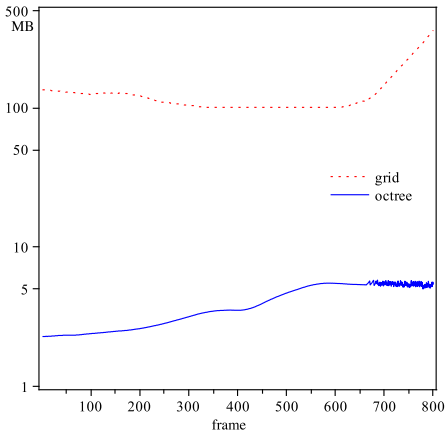

図 13 近傍粒子探索に必要なメモリ量
Fig. 13 Consumed memory for nearest neighboring particle searching.

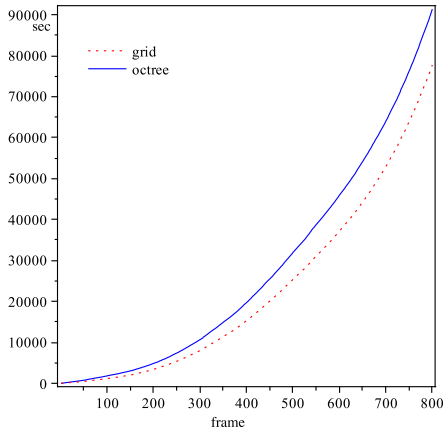

図 14 近傍粒子探索の累積処理時間
Fig. 14 Consumed time for nearest neighboring particle searching.

必要である．このとき，新しい木探索法で必要なメモリ量は 5.38 MB であり，格子探索法のわずか 1.49%で済んでいる．さらに，近傍粒子探索の累積処理時間をフレーム時間に対してプロットしたのが**図 14** であり，格子探索法に対する新しい木探索法の累積処理時間の比率をプロットしたのが**図 15** である．この比率は最初は 1.66 程度で少し大きいが，単調減少傾向にあり，400 フレーム付近で 1.3 を切り，最終的には 1.18 になる．このような，時間の経過とともに累積処理時間の比率が減少する要因については次節で議論する．

5.3 実験に対する考察

実験における代表的な寸法を L とし粒子の影響半径を r_e とすると，格子探索法で必要とされるメモリ量は 3 次元の場合，近似的に次のように表される．

$$M_g = k\left(\frac{L}{r_e}\right)^3 \tag{1}$$

ただし，k は各格子セルに必要な記憶容量である．

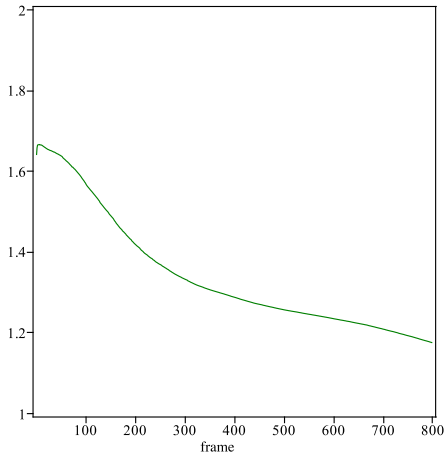

図 15 格子探索法に対する新しい木探索法の累積処理時間の比率
Fig. 15 Ratio of consumed time by new tree search method to that by grid search method.

これに対し，新しい木探索法で必要とされるメモリ量は M_g を使用すると，近似的に次のようになる．

$$M_o = \frac{8}{7}\frac{M_g}{\alpha} \tag{2}$$

ここに，α は計算領域内に粒子がどの程度偏在しているかを表す量で，ここでは偏在度と呼ぶことにする．

粒子が計算領域内で占める体積を V_p とすると，偏在度は次のように定義される．

$$\alpha = \frac{L^3}{V_p} \tag{3}$$

式 (2) より，偏在度が 8/7 よりも大きくなればなるほど M_o が M_g より小さくなることが分かるが，上の 2 つの実験ともに偏在度はこの条件を満たしている．特に，乗用車と滝を使用する実験のような実用的なシーンでは偏在度は非常に大きくなることが多く，したがって必要なメモリ量は格子探索法に比べかなり小さくなる．なお，ダム崩壊実験においては，粒子数が増加するにつれ，水槽の開口部から飛び出す粒子が増えるという現象が観測されており，これが偏在度の増加につながったと考えられる．

次に処理時間であるが，粒子数を N とするとき格子探索法の計算量は $O(N)$ であり，新しい木探索法の計算量が従来の木探索法と同様に $O(N\log(N))$ であると仮定すると，格子探索法に対する新しい木探索法の処理時間増加率は N の単調増加関数でなければならない．しかしながら，ダム崩壊実験においては，粒子数が増加したとき処理時間の増加率が全体として減少傾向にあるし，乗用車と滝を使用する実験においては，時間の経過とともに粒子数が増加するにもかかわらず，累積処理時間の比率は減少している．以下で，この原因について考察するために，まず近傍粒子の探索処理を次の 3 つに分割しよう．

(1) 現在注目している粒子の属するセルを求める．
(2) 求めたセルの隣接セルを計算する．

(3) 求めたセルとその隣接セルの内部にある粒子に対し影響半径 r_e 内にあるかどうか検査する．

格子探索法の場合，処理 (1) と処理 (2) に要する時間は処理 (3) に要する時間に比べ無視することができる．そこで，粒子を流体粒子と壁粒子 [1] に分割し，流体粒子の個数を N_f，処理 (3) の平均的な計算量を C_f とし，壁粒子の個数を N_w，処理 (3) の平均的な計算量を C_w とすれば，全体の計算量 C_g は次のようになる．

$$C_g = N_f C_f + N_w C_w \tag{4}$$

他方，新しい木探索法の場合，処理 (1) と処理 (2) に要する時間は木の深さ D に比例するので，この 2 つの処理に要する計算量を合わせて sD と書くことにする．ただし，s は比例定数である．また，処理 (3) の計算量は格子探索法の場合と同一になるので，全体の計算量 C_o は次のようになる．

$$C_o = (N_f + N_w)sD + C_g \tag{5}$$

よって，格子探索法に対する新しい木探索法の計算量の増加率は次の式で表される．

$$R = \frac{C_o - C_g}{C_g} = \frac{(N_f + N_w)sD}{N_f C_f + N_w C_w} \tag{6}$$

さて，そこでダム崩壊実験の場合を考えよう．粒子の初期配置間隔 [1] を d とし $x \equiv 1/d$ と置けば，流体粒子が 3 次元的に広がっているのに対し，壁粒子は 2 次元的に広がっているから，流体粒子と壁粒子の個数は次のように書ける．

$$N_f = ax^3 \tag{7}$$

$$N_w = bx^2 \tag{8}$$

ただし，a と b は比例定数である．また，粒子の影響半径 r_e は d に比例し，新しい木探索法の木の深さ D は $\log(1/r_e)$ に比例するので，c を比例定数として D は近似的に次のように表される．

$$D = c\log(x) \tag{9}$$

式 (7) から式 (9) までを式 (6) に代入し，さらに $\beta \equiv sc$ と置けば次の式が得られる．

$$R = \frac{(ax + b)\beta\log(x)}{aC_f x + bC_w} \tag{10}$$

そこで，実験で得られた値を基にして $a = 0.0015$，$b = 0.25$ という値を求め，流体粒子と壁粒子の分布状態から $C_f = 125$，$C_w = 20$ という値を求める．さらに実験で得られた計算量の増加率を用い最小二乗法によって $\beta = 2.1$ を求め，これらの値を式 (10) に代入してグラフを描くと**図 16** のようになる．ただし，丸点は最小二乗法で使用した実験値である．単調減少のグラフになるので，これは粒子数が増加したとき処理時間の増加率が全体として減少傾向にあることを説明している．ただし，実験値とのずれが若干大きいのは，木の深さ D を本来なら階段関数で記述すべきところを連続関数にしたことが主な原因と考えられる．なお，壁粒子がないときは式 (10) で $b = 0$ とすればよく，このとき式 (10) は次のようになる．

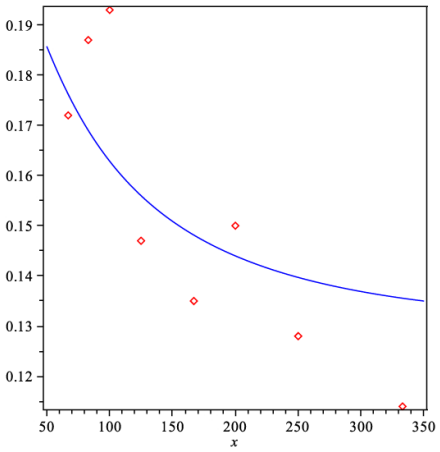

図 16 格子探索法に対する新しい木探索法の計算量の増加率（理論値）

Fig. 16 Incremental ratio of new tree search method to grid search method (theoretical value).

$$R = \frac{\beta\log(x)}{C_f} \tag{11}$$

これは x に関する単調増加関数であるから，処理時間の増加率が全体として減少傾向にあるのは 2 次元状に広がる壁粒子が原因であることが分かる．

次は，乗用車と滝を使用する実験であるが，ここでは壁粒子の個数は一定で流体粒子の個数が時間に比例するから，次の式が得られる．

$$N_f = et \tag{12}$$

$$N_w = N_0 \tag{13}$$

ただし，N_0 と e は定数であり t はフレーム時間を表している．また，この実験では最初から大きな床が広がっており，これに沿って壁粒子が生成されるから，新しい木探索法における木の深さ D は流体粒子が増加してもほぼ一定と考えられる．したがって，式 (6) に式 (12) と式 (13) を代入して，次の式を得る．

$$R = \frac{(et + N_0)sD}{eC_f t + N_0 C_w} \tag{14}$$

これは，格子探索法に対する新しい木探索法の累積処理時間の増加率を表している．この式を t について微分すれば次のようになる．

$$\frac{dR}{dt} = \frac{esDN_0(C_w - C_f)}{(eC_f t + N_0 C_w)^2} \tag{15}$$

C_f は流体粒子における処理 (3) の平均的な計算量であり，

C_w は壁粒子における処理 (3) の平均的な計算量であるが，一般に流体粒子の近傍粒子数は壁粒子の近傍粒子数より多いので $C_w < C_f$ であり，式 (15) より $dR/dt < 0$ であることが分かる．これは，R が t に関する単調減少関数であることを表し，累積処理時間の比率がフレーム時間とともに減少することを説明している．なお，壁粒子が存在しないならば $N_0 = 0$ となり式 (14) は次のようになる．

$$R = \frac{sD}{C_f} \tag{16}$$

これは定数だから，累積処理時間の比率が減少するのは，ここでも壁粒子が原因であることが分かる．

6. おわりに

我々の新しい木探索法を使用してダム崩壊実験を実施した結果，格子探索法に対する計算時間の増加率を 20%以下に抑えたまま，最大使用メモリ量において格子探索法の場合の 35〜75%程度を削減することができた．削減率は粒子数が増えるほど増加する傾向にあった．また，より実用に近い実験として乗用車が滝の中を通過するシーンを作成したが，最大使用メモリ量は格子探索法のわずか 1.49%であった．これはダム崩壊実験よりも粒子の偏在率が高いからで，一般に実用的な流体シーンでは粒子の偏在率が高くなる傾向がある．なお，使用メモリ量が最大となるのは 80 フレーム目であったが，このとき格子探索法に対する累積計算時間の増加率は 18%であった．それから，両方の実験で，粒子数が増加したとき計算時間の増加率が減少するという現象が観測されたが，これは壁粒子の存在が関係していることが明らかになった．

今後は近傍粒子探索以外で使用されるメモリの削減の研究も進めたいと考えている．たとえば，現在のところ，各粒子において近傍粒子が見つかった後，それらの近傍粒子を何らかの方法で記憶しておく必要がある．現在は各粒子に固定長の配列を割り当て，これに近傍粒子の粒子番号を記録しているが，この方法は未使用部分が多く発生するので何らかの対策が必要である．たとえば，近傍粒子が見つかった時点で，その粒子からの影響分のみ計算するなどの工夫をすれば近傍粒子を記憶する必要性をなくせる可能性がある．また，本論文では新しい隣接セル探索アルゴリズムを提案しているが，いくつかの制約を持っている．これについても，制約を除去したより一般的なアルゴリズムを開発したいと考えている．

参考文献

[1] 越塚誠一：粒子法，丸善 (2005).
[2] Monaghan, J.: An introduction to SPH, *Computer Physics Communications*, Vol.48, No.1, pp.89–96 (1988).
[3] Koshizuka, S. and Oka, Y.: Moving-particle semi-implicit method for fragmentation of incompressible fluid, *Nuclear Science and Engineering*, Vol.123, No.3, pp.421–434 (1996).
[4] Liu, G. and Liu, M.: *Smoothed Particle Hydrodynamics – a meshfree particle method*, World Scientific (2003).
[5] Monaghan, J.: Particle methods for hydrodynamics, *Computer Physics Report*, Vol.3, No.2, pp.71–124 (1985).
[6] Hockney, J. and Eastwood, R.: *Computer Simulation Using Particles*, Taylor & Francis (1988).
[7] Simpson, J.C.: Numerical Techniques for Three-dimensional Smoothed Particle Hydrodynamics Simulations: Applications to Accretion Disks, *Astrophysical Journal*, Vol.448, pp.822–831 (1995).
[8] Hernquist, L. and Katz, N.: TreeSPH: A unification of SPH with the hierarchical tree method, *Astrophysical Journal Supplement Series*, Vol.70, pp.419–446 (1989).
[9] Langetepe, E. and Zachmann, G.: *Geometric Data Structures for Computer Graphics*, A K Peters/CRC Press (2006).
[10] Samet, H.: Neighbor Finding in Images Represented by Octrees, *Computer Vision, Graphics, and Image Processing*, Vol.46, pp.367–386 (1989).
[11] Gargantini, I.: An Effective Way to Represent Quadtrees, *Comm. ACM*, Vol.25, No.12, pp.905–910 (1982).
[12] Gould, D.: *Complete Maya Programming: An Extensive Guide to MEL and C++ API*, Morgan Kaufmann (2003).

付　　録

A.1 八分木の隣接セルの探索で使用される 2 個の配列

4.4 節で述べた隣接セル探索アルゴリズムでは 2 個の配列を使用するが，八分木に対する配列全体を**表 A·1** に示す．

表 A·1 隣接セルの探索で使用される 2 個の配列
Table A·1 Arrays used in searching for adjacent cells.

添え字	0	1	2	3	4	5	6	7
pos	7	6	5	4	3	2	1	0
upper	0	72	24	96	8	80	32	−1
添え字	8	9	10	11	12	13	14	15
pos	3	2	1	0	7	6	5	4
upper	8	80	32	−1	8	80	32	−1
添え字	16	17	18	19	20	21	22	23
pos	7	6	5	4	3	2	1	0
upper	8	80	32	−1	16	88	40	104
添え字	24	25	26	27	28	29	30	31
pos	5	4	7	6	1	0	3	2
upper	24	96	24	96	32	−1	32	−1
添え字	32	33	34	35	36	37	38	39
pos	1	0	3	2	5	4	7	6
upper	32	−1	32	−1	32	−1	32	−1
添え字	40	41	42	43	44	45	46	47
pos	5	4	7	6	1	0	3	2
upper	32	−1	32	−1	40	104	40	104
添え字	48	49	50	51	52	53	54	55
pos	7	6	5	4	3	2	1	0
upper	24	96	48	112	32	−1	56	120
添え字	56	57	58	59	60	61	62	63
pos	3	2	1	0	7	6	5	4
upper	32	−1	56	120	32	−1	56	120
添え字	64	65	66	67	68	69	70	71
pos	7	6	5	4	3	2	1	0
upper	32	−1	56	120	40	104	64	128
添え字	72	73	74	75	76	77	78	79
pos	6	7	4	5	2	3	0	1
upper	72	72	96	96	80	80	−1	−1
添え字	80	81	82	83	84	85	86	87
pos	2	3	0	1	6	7	4	5
upper	80	80	−1	−1	80	80	−1	−1
添え字	88	89	90	91	92	93	94	95
pos	6	7	4	5	2	3	0	1
upper	80	80	−1	−1	88	88	104	104
添え字	96	97	98	99	100	101	102	103
pos	4	5	6	7	0	1	2	3
upper	96	96	96	96	−1	−1	−1	−1
添え字	104	105	106	107	108	109	110	111
pos	4	5	6	7	0	1	2	3
upper	−1	−1	−1	−1	104	104	104	104
添え字	112	113	114	115	116	117	118	119
pos	6	7	4	5	2	3	0	1
upper	96	96	112	112	−1	−1	120	120
添え字	120	121	122	123	124	125	126	127
pos	2	3	0	1	6	7	4	5
upper	−1	−1	120	120	−1	−1	120	120
添え字	128	129	130	131	132	133	134	135
pos	6	7	4	5	2	3	0	1
upper	−1	−1	120	120	104	104	128	128
添え字	136	137	138	139	140	141	142	143
pos	7	6	5	4	3	2	1	0
upper	72	136	96	160	80	144	−1	168
添え字	144	145	146	147	148	149	150	151
pos	3	2	1	0	7	6	5	4
upper	80	144	−1	168	80	144	−1	168
添え字	152	153	154	155	156	157	158	159
pos	7	6	5	4	3	2	1	0
upper	80	144	−1	168	88	152	104	176
添え字	160	161	162	163	164	165	166	167
pos	5	4	7	6	1	0	3	2
upper	96	160	96	160	−1	168	−1	168
添え字	168	169	170	171	172	173	174	175
pos	1	0	3	2	5	4	7	6
upper	−1	168	−1	168	−1	168	−1	168
添え字	176	177	178	179	180	181	182	183
pos	5	4	7	6	1	0	3	2
upper	−1	168	−1	168	104	176	104	176
添え字	184	185	186	187	188	189	190	191
pos	7	6	5	4	3	2	1	0
upper	96	160	112	184	−1	168	120	192
添え字	192	193	194	195	196	197	198	199
pos	3	2	1	0	7	6	5	4
upper	−1	168	120	192	−1	168	120	192
添え字	200	201	202	203	204	205	206	207
pos	7	6	5	4	3	2	1	0
upper	−1	168	120	192	104	176	128	200

笠 晃一 （正会員）

1983 年九州大学大学院工学研究科電子工学専攻修士課程修了．1986 年（株）日本データベースネットワーク研究所入社．自然言語によるデータベース検索システムの研究開発に従事．1997 年福岡工業大学情報工学部講師．2003 年同大学情報工学部助教授．2007 年同准教授，現在に至る．計算言語学，3 次元コンピュータグラフィックス等の研究に従事．博士（工学）．電子情報通信学会会員．