階層的ボイドアルゴリズムによる大規模な魚群のシミュレーション

日大生産工（院）　○石橋　佳明
日大生産工　　吉田　典正

1. はじめに

海で見られる魚群は，時に数十万体以上から成る大規模なものある．ディスプレイの高解像度化からも，スクリーン上に大規模な群れを表示することが望まれる．群れを形成するアルゴリズムとして，ボイドアルゴリズム [1]が有名であるが，ボイドアルゴリズムを用いて現在のコンピュータを用いてリアルタイムでシミュレーションできる個体数は，我々の実験結果では数千体程度である．本研究では，動作生成処理を高速化し，従来のボイドアルゴリズムよりも効率的にシミュレーションが可能な階層的ボイドアルゴリズムを提案する．また，数万体程度の群れの動作生成がリアルタイムで計算可能なことを示す．

2. ボイドアルゴリズム

ボイドアルゴリズム [1]は Craig Reynolds によって提案された，群れを形成するためのアルゴリズムである．このアルゴリズムは“整列”，“結合”，“分離”という 3 つの単純なルールを与えることによって，各ルールが個体間で相互作用することにより，群れを形成するアルゴリズムである．
図 1 は，黒色の個体をあるステップでの処理対象とし，灰色の個体を同じ群れの個体としたときに，各ルールによって得られる 3 つのベクトル $V_{alignment}$，$V_{cohesion}$，$V_{separation}$ を示している．各ルールによって得られた 3 つのベクトルの和を基に個体の移動ベクトルを決定する．

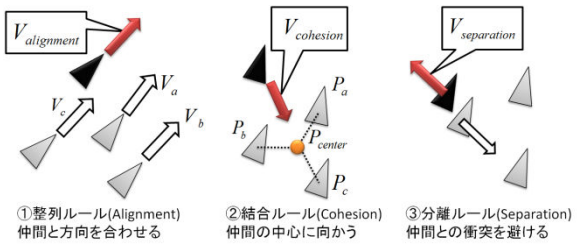


図 1　ボイドアルゴリズムの概要

整列ルールでは仲間と進む方向を合わせる処理を行う．結合ルールでは群れからはぐれないように群れの中心へ向かう処理を行い，分離ルールでは一定距離以上仲間に近付かないように処理をする．ボイドアルゴリズムは，個体数を n としたとき，一番近い個体を探す処理を，線形探索を用いて探す．すべての個体に対して実行するため，計算コストは，$O(n^2)$ である．

3. 本研究の概要

大規模な魚群をリアルタイムでシミュレーションするためには，動作生成処理および描画処理を可能な限り効率的に行う必要がある．本研究では動作生成処理にのみ着目し，探索処理を高速化することを行う．具体的には個体数が少なくなるように小さな群れを複数作成し，この小さな群れによって階層構造を構築する．最終的には，この小さな群れを階層構造によって連結し，１つの群れを作成する．本手法は，障害物回避が発生した場合でも O'Hara らによる手法 [2]と異なり，著しく処理速度が低下する恐れは少ない．また，再帰的ボイドアルゴリズム [3]では，触れられていなかった障害物回避についても触れる．

4. 階層的ボイドアルゴリズム

図 2(a)に示すように，従来のボイドアルゴリズムは，作成された群れ内にいるすべての個体に対して，一番近い仲間を探索するため，線形探索を行う．この線形探索によって処理に非常に時間がかかる．本研究では，図 2(b)のように小さな群れへと分割する，以降この小さな群れをユニットと呼ぶことにする．群れを分割することにより，探索範囲を狭め，探索処理を効率的に行う．
本研究で提案する階層的ボイドアルゴリズムでは，まずユニットを作成することから始め，階層ごとにユニットを作成していき，階層ごとのユニット，または個体の位置の更新処理を行っていく．

Simulation of a Large School of Fish using a Hierarchical Boid Algorithm
Yoshiaki ISHIBASHI and Norimasa YOSHIDA

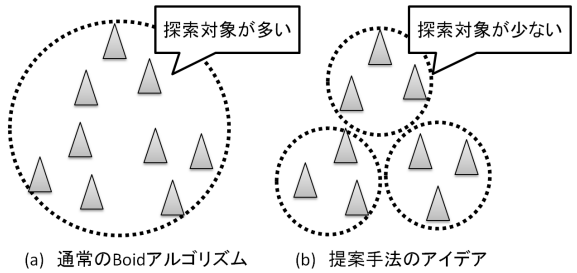


図 2 個体数と探索処理

具体的には，親ユニット，子ユニット，孫ユニットというようにトップダウン的に階層を作成し，位置の更新処理は最下位層のユニットから更新を開始する．孫ユニット，子ユニット，親ユニットというようにボトムアップ的に位置の更新処理は行う．本研究で提案する階層的ボイドアルゴリズムでは，まずユニットを作成することから始め，階層ごとにユニットを作成していき，階層ごとのユニット，または個体の位置の更新処理を行っていく．具体的には，親ユニット，子ユニット，孫ユニットというようにトップダウン的に階層を作成し，位置の更新処理は最下位層のユニットから更新を開始する．孫ユニット，子ユニット，親ユニットというようにボトムアップ的に位置の更新処理は行う．

我々の手法において，個体数をn，階層数をmとし，各階層の分割数および，各ユニット内の個体数を$\sqrt[m]{n}$とすると，計算コストは$O\left(n^{m+1/m}\right)$となる．また，$m \to \infty$の極限では計算コストは$O(n)$に収束する[4]．

4.2 各階層における結合処理

階層的ボイドアルゴリズムでは最下位層のユニットから順に更新処理を行っていく．各階層ではユニットごとに，別々のボイドアルゴリズムが実行される．そのため，別々に動くユニットをまとめる必要がある．このユニットをまとめる処理は，結合ルールの適用の仕方を変更することで可能である．具体的には図 3 のように，通常のボイドアルゴリズムでは，群れの中心に向かうように結合ルールを適用するが，階層的ボイドアルゴリズムでは，親ユニット以外の各ユニットに対して，上位階層の位置へ向かうように結合ルールの適用の仕方を変更する．

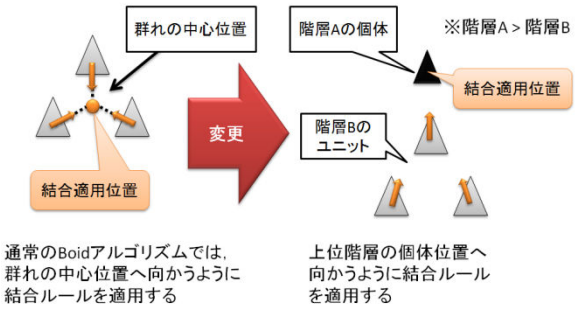


図 3 結合ルールの変更

4.3 各階層における分離処理

結合ルールによってユニット同士が集まるが，ユニット同士が集まることにより，衝突が発生する可能性がある．これを避けるために通常のボイドアルゴリズムと同じように，ユニットに対しても分離処理を行うことによって衝突を回避する．このユニット間の衝突を考慮するために，ユニット内のすべての個体を包括するようなバウンディングスフィアを算出し，各ユニットに対して位置の更新処理後に，算出したバウンディングスフィアを用いて，親ユニットの分離ルールの適用距離を変更する．具体的には，図 4 のように分離ルールを適用する距離に，ユニットごとに算出したバウンディングスフィアの半径を加算し，分離ルールの適用距離を長くするということを行う．

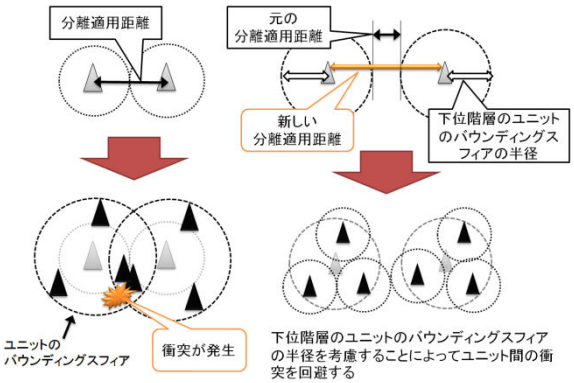


図 4 ユニット間の衝突の回避

4.4 各階層における整列処理

結合ルールを変更することにより群れが統合されるが，前述したように，各階層内の各ユニットごとに異なるボイドアルゴリズムが実行されている．そのため，群れ全体の向きと各ユニットの向きが一致しない場合がある．向きが一致していないことにより，群れ全体の統一性がないため，一つの群れに見えない場合がある．これに対処するため，整列ルールの適用の仕方を変更する．通常

整列ルールは，ユニットの平均速度ベクトルを一致するようにするが，階層的ボイドアルゴリズムでは親以外のユニットに対して，最上位層のユニットの平均速度ベクトルとユニットごとの平均速度ベクトルを線形補間する．このようにすることで，群れ全体の向きをある程度一致させながら，ユニット内で向きが一致するベクトルを作成することができる．

4.5　障害物回避

階層的ボイドアルゴリズムでは，ユニット内で働くボイドアルゴリズムに，第 4 のルールとして，回避ルールを追加することによって，障害物を回避することが可能である．回避ルールを単純に追加することによって，障害物回避が可能なのは，階層的ボイドアルゴリズムが通常のボイドアルゴリズムを基に階層化を行っていることと，4.4 節で説明した各階層における整列処理によるためである．

図 5 に示すように，回避ルールを追加することによって，各階層のユニット内で異なるボイドアルゴリズムが実行されるため，ある個体が障害物との距離が一定距離内になったとき，障害物を回避するベクトルが作成される．次に，4.4 節で述べた整列処理により，回避ベクトルが作成されることによって群れ全体の向きも変化するため，障害物の回避距離内にいない個体も，ある程度全体の向きと一致させる方向に向きが変化する．つまり，間接的に回避するベクトルが作成される．

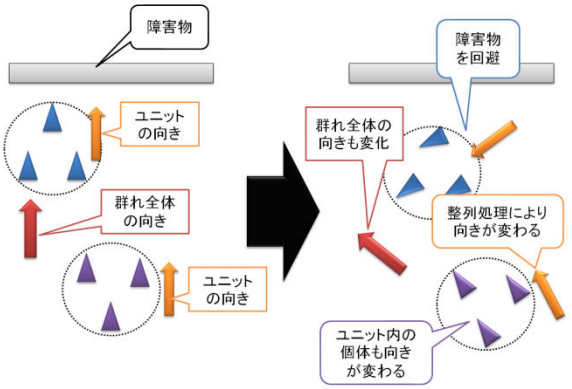
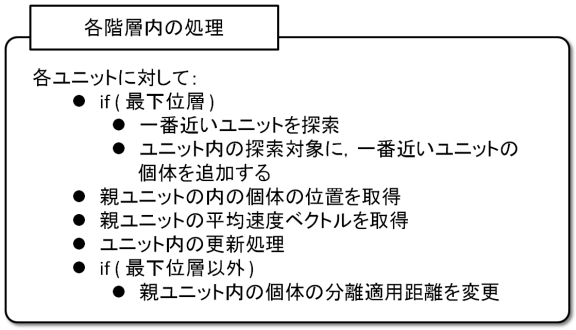


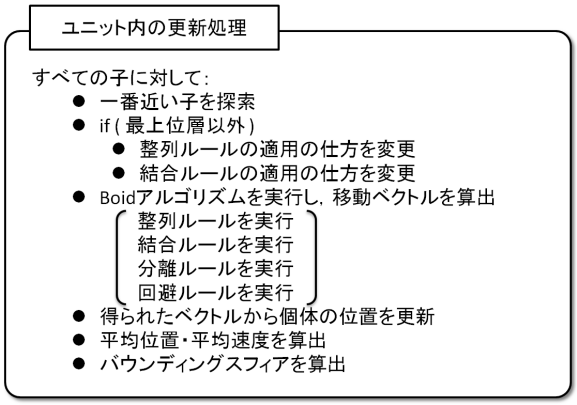
図 5　障害物の回避

4.5　階層的ボイドアルゴリズムの更新処理

階層的ボイドアルゴリズムでは，更新処理は孫ユニットから子ユニットへ，子ユニットから親ユニットへというように最下位層から順にボトムアップ的に行う．ボトムアップ的に更新処理を行うのは 4.3 節で説明した，バウンディングスフィアの半径を渡す必要があるためである．具体的は，図 6(a)に示すように各階層ごとの更新処理を行っていき，図 6(b)に示すようにユニットごとの更新処理を行う．



(a)　各階層内の処理



(b)　ユニット内の更新処理

図 6　更新処理の概要

5. 実行結果

図 7 は，通常のボイドアルゴリズム，および階層的ボイドアルゴリズムにおいて，階層数が 1, 2, 3, 4, 5 の場合の 10 ステップの平均動作生成処理時間の結果である．横軸は個体数を示し，縦軸は 1 ステップあたりの動作生成処理時間(s)を示している．ボイドアルゴリズムと階層的ボイドアルゴリズムを比較してわかるように，処理時間が大幅に改善され，階層数が増えるほどの個体数に対する動作生成時間のグラフが直線に近づいている様子がわかる．

階層数を 3，個体数が 27,000 体のときの実行画面をキャプチャした様子を図 8，図 9 に示す．プログラムの実装には C++言語を用いており，描画に用いた魚 1 体あたりの三角形ポリゴンの数は 635 面である．図 7 は，被食者から捕食者が逃げようとする様子を示しており，図 8 は，その際の群れ全体の様子を示している．このときの描画時間を含めたシミュレーションの 1 ステップの処理時間は約 1.3 秒である．

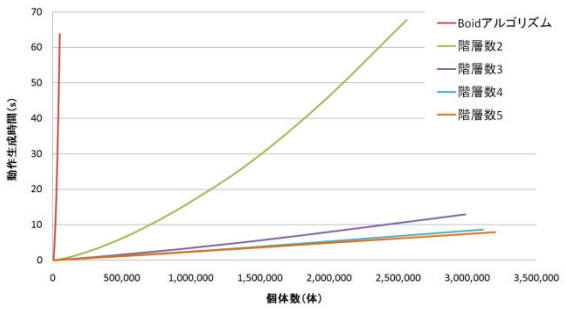


図 7　1 ステップあたりの動作生成時間

6. まとめと今後の展望

階層的ボイドアルゴリズムによって，探索処理を効率化し，動作生成処理を従来のボイドアルゴリズムよりも高速化した．また，階層的ボイドアルゴリズムにおいて，階層数が深くなっていくにつれて計算コストが線形に近づくことをグラフで実例として示した．本手法は，階層構造を用いるために，従来手法に比べて，動作生成を高速化することできるが，群れを制御するパラメータの数とメモリ使用量が増加する．

今後の課題としては，より扱いやすいように制御パラメータ数を減らすことや，激しい動きをする群れにも応用できるように更なる動作の改善，階層構造を用いた場合の魚の渦の形成などがあげられる．また，本研究では，動作生成処理のみに焦点を当てたが，今後は大規模な魚群を高速に表示できるように描画処理の高速化についても取り組む必要性がある．

参考文献

1) Craig W. Reynolds, “Flock, Herds, and Schools: A Distributed Behavioral Model”, Computer Graphics (Proc. SIGGRAPH), 21 (4), Jul. (1987), pp.25-34

2) Noel O'Hara, “Hierarchical Imposters for Flocking Algorithm in 3D”, Computer Graphics forum, vol.21, No.4 (2002), pp.723-731

3) Yoshiaki Ishibashi, Norimasa Yoshida, “View-dependent animation of a large school of fish”, Image Electronics and Visual Computing Workshop, 1p-11, 2007.

4) 石橋佳明，吉田典正，“大規模な魚群シミュレーションのための階層的 Boid アルゴリズム”，情報処理学会　グラフィクスと CAD 研究会第 133 回研究発表会, 2008.

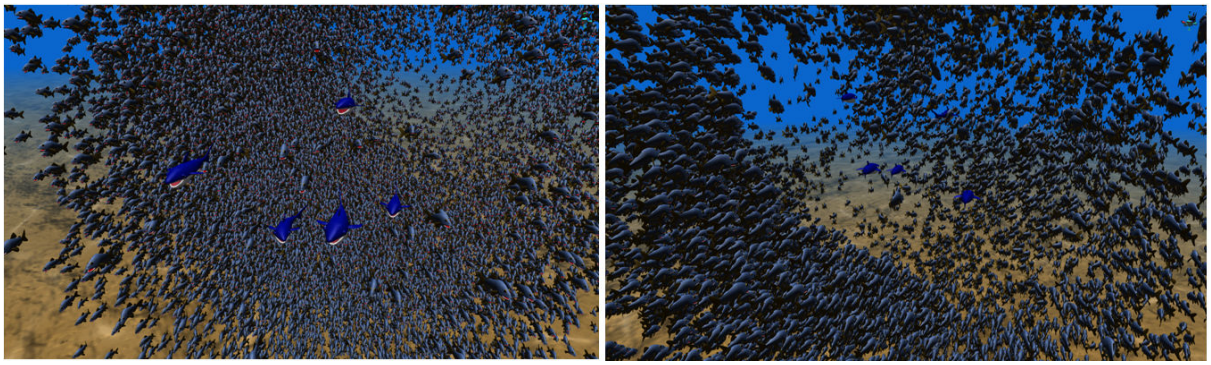


図 8　被食者が捕食者から逃げようとする様子（階層数 3，個体数 27,000）

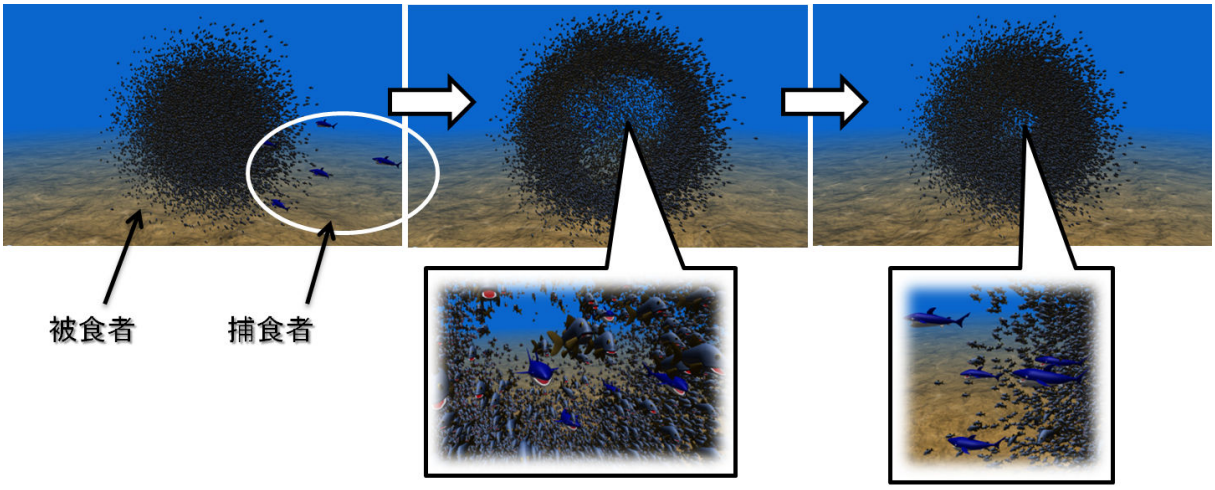


図 9　被食者が捕食者から逃げる時の群れ全体の様子（階層数 3，個体数 27,000）