2008－CG－133 (7)
2008／11／7

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

石橋佳明* 吉田典正**
*日本大学 生産工学研究科 **日本大学 生産工学部

海にいる魚の群れは数十万体以上の個体から成る場合がある．また，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．しかしながら，群れを形成するのに用いられる Boid アルゴリズムではリアルタイムにシミュレーションできる数は，我々の実装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 Boid アルゴリズムよりも効率的にアニメーション可能な階層的 Boid アルゴリズムを提案する．また数万体程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．

Hierarchical Boid Algorithm for a Simulation of a Large School of Fish

Yoshiaki Ishibashi* and Norimasa Yoshida**
*Graduate School of Industrial Technology, Nihon University
** College of Industrial Technology, Nihon University

A school of fish in the sea is sometimes composed of more than hundreds of thousands of individuals. With the increasing resolution of computer display, a large school of fish is needed to be displayed on the screen. Although the Boid algorithm is usually used for flocking animations, the number of fish simulated in real time is around thousands. In this paper, we propose a hierarchical Boid algorithm for efficient flocking computation of a large school of fish. Our implementation shows that the flocking motion of more than tens of thousands of fish can be computed in real time.

1. はじめに

海で見られる魚の群れは，時に数十万もの個体からなる大規模なものになる場合がある．群れを形成するアルゴリズムとしては Boid アルゴリズム[1]が有名である．Boid アルゴリズムを用いて，現在のコンピュータでリアルタイムにシミュレーションできる数は，我々の実験結果では，数千体程度である．大規模な魚の群れをリアルタイムでシミュレーションする場合，動作生成処理および描画処理を可能な限り高速化する必要がある．本研究では，大規模な魚をシミュレーションするための動作生成を従来の Boid アルゴリズム[1]よりも効率的に行う階層的 Boid アルゴリズムを提案する。本手法では Boid アルゴリズムを基に階層構造を作成し，探索処理の効率化を行い，動作生成処理の時間短縮を図る．また階層的 Boid アルゴリズムにおける障害物回避についても触れる．

2. 関連研究

Boid 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群れをなす個体すべてに “整列(Alignment)”，“結合(Cohesion)”，“分離(Separation)” 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．整列ルールでは仲間と進む方向を合わせる処理を行い，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づかない処理を行う．

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，Boid アルゴリズムの 3 つの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．個体の移動ベクトルは 3 つのルールによって得られたベクトルの和を基にする．

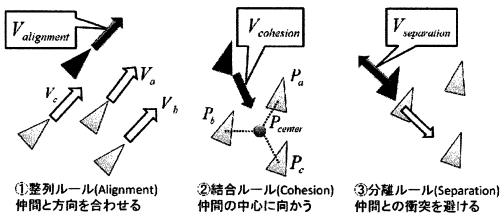


図 1 Boid アルゴリズムの概要

Boid アルゴリズムは，個体数をnとしたとき，一番近い個体を線形探索を用いて探す処理をすべての個体に対して行うため，その計算コストは$O(n^2)$である．

O'Hara らは，階層性を用いて群れをシミュレーションさせる研究[2]を提案している．この手法では群れをグループへと分割し，トップダウン的に階層を形成し，群れの動作生成処理のコストを減らしている．しかし，安定状態のグループが形成されていない場合，あるいは障害物回避が発生したときに処理効率が低下してしまうなど，動的な変化に弱い側面がある．また我々の手法は階層構造を構築する際はトップダウン的に行うが，更新処理はボトムアップ的に行うという点にも相違がある．

Treuille は，群集を流体的にシミュレーションする Continuum Crowd[3]を提案している．この手法は，2 次元のポテンシャル場を構築し，ポテンシャルが減少する方向へと個体を移動させ，群れのシミュレーションを行う．ポテンシャル場を更新する際に Eikonal 方程式を高速マーチング法によって解く処理に格子の数をNとした場合，$O(N \log N)$の計算コストがかかる．魚や鳥といった 3 次元的に動く群れへ適用を考えると大規模な群れをシミュレーションするにはポテンシャル場の構築がネックになる．

佐藤らは，群れの動作生成を効率的に行うフラクタル Boid アルゴリズム[4]を提案している．この手法はフラクタル性を用いることによって，動作生成処理を簡略化し，処理速度を向上するというものである．しかし，上位階層の動作をそのまま下位層にコピーしているため動作に規則性が生じ，不自然に見える場合がある．また，群れ内で他の個体との衝突が発生してしまうなどの問題点がある．また本研究に関連する手法として，石橋らによる再帰的 Boid アルゴリズム[5]がある．このアルゴリズムでは Boid アルゴリズムを再帰的に 2 段階適用することによって処理を高速化するものである．計算は比較的高速だが，多段階での実装，また障害物や群れ内での個体との衝突回避の問題が取り上げられていない．

3. 本研究の概要

本研究では，探索処理を高速化するために，個体数が少なくなるように小さな群れを作成し，この小さな群れによって階層構造を構築する．各階層では，群れ単位で別々の Boid アルゴリズムを実行する．この別々の動きをする各群れを 1 つの大きな群れへと統合させるために，各階層の群れを上位階層の群れと関係を持たせる．

本手法は，Boid アルゴリズムに基づき階層化を行うので，障害物との衝突回避も考慮することが可能である．

4. 階層的 Boid アルゴリズム

図 2 に示すように，通常の Boid アルゴリズムは生成された群れ内のすべての個体に対して，探索処理を行うため，探索処理に非常に時間がかかる．

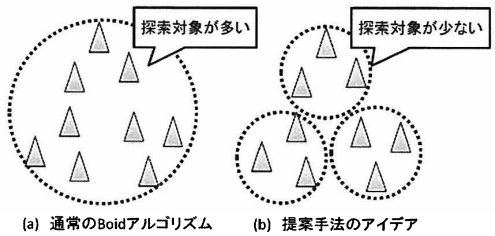


図 2 個体数と探索処理

提案する階層的 Boid アルゴリズムでは，まず図 2(b)のようにいくつかのグループを作成することから始める．以降このグループのことをユニットと呼ぶことにする．階層的 Boid アルゴリズムでは各階層ごとにユニットを作成し，各階層ごとにユニットまたは個体の位置の更新処理を行う．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し，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行っていく．更新処理の詳細について 4.5 節で説明する．すべてのユニットに対して更新処理が終了したら，最下位層のユニットのみを描画し，群れを表示する．

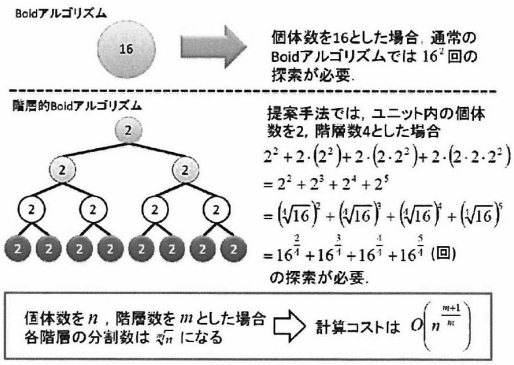


図 3 アルゴリズムの計算コスト

我々の手法において，図3に示すように個体数を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，後述する一番近いユニットの探索を行わない場合，計算コストは$O\left(n^{m+1/m}\right)$となる．また$n\to\infty$の極限では，計算コストは$O(n)$に収束していく．

4.1　階層構造の構築

階層構造の構築はトップダウンで行う．個体を生成する際に，まず基となるユニットを作成する．作成したユニットを親とする子ユニットを作成し，親子関係を構築する．以降この処理を指定した階層数に達するまで再帰的に行う．

4.2　各階層における結合・分離・整列処理

階層的 Boid 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い．各階層の各ユニットごとに別々のBoid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4のように，通常の Boid 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Boid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，群れの中心位置ではなく上位階層の指定した位置へ向かうように結合ルールを適用する．

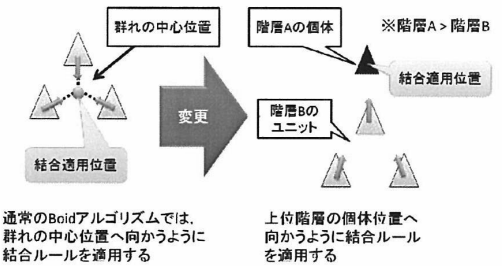


図 4　結合ルールの変更

ユニット間の衝突を考慮するために，バウンディングスフィアを算出する．各ユニットを更新後に，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，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変更し，この変更した分離ルールの適用距離を親ユニットへ渡す（図 5 参照）．分離ルールの適用距離を変更する処理を行わない場合，図6のように隣接するユニット同士が近づきすぎてしまった場合，衝突する個体が発生してしまう．バウンディングスフィアを用いることにより，衝突を考慮することができるが，個々のユニットが独立してしまうという問題が発生する．この問題については，4.3 節で説明する．

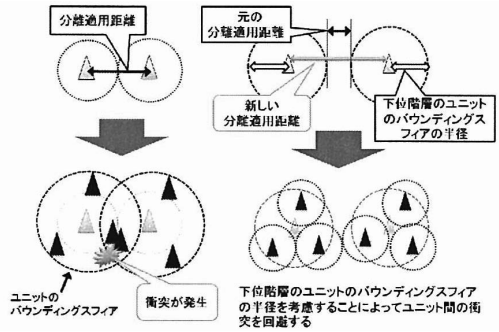


図 5　ユニット間の衝突の回避

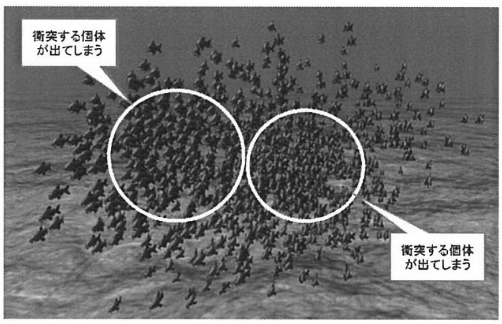


図 6　衝突する個体の発生

結合ルールを変更することにより群れが統合されるが，各ユニット内で異なる Boid アルゴリズムの整列ルールにより群れ全体の向きと各ユニットの向きが一致しないことがある．向きが一致していないことにより，一つの大きな群れに見えないことがある．整列ルールの適用の仕方を変更し，群れ全体の向きと各ユニットの向きをある程度一致させることで，改善させることができる．群れ全体の向きは，最上位層のユニットの動きによって変化するので，ユニットごとの全体向きと最上位層のユニットの全体の向きを考慮することによって，群れ全体の向きと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．本研究では，親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とによって算出している．

4.3　群れ内での衝突回避

隣接ユニットが近づくことで衝突する個体が出てきてしまう問題やアーティファクトが発生してしまう原因は，近くにユニットがいるかどうかを考慮していないためである．そこで，衝突を低減するために，一番近いユニットを探索し，Boid アルゴリズムの探索処理の対象に，一番近いユニット内の個体を加える．ここで，一番近いユニットを探索するのは，探索の対象とする個体の数が増加すると，処理速度が低下するため，一番近いユニットだけに限定している．また，階層的 Boid アルゴリズムでは，最下位層のユニット内の個体のみを表示するので，一番近いユニットを探索する処理は最下位層のユニットのみに限定する．

本研究では，大規模な魚群をシミュレーションすることが目的であるので，一番近いユニットを探索する処理は，できるだけ速いほうが望ましい．そこで，探索の処理速度を向上するために，階層構造を用いて探索範囲を限定するという条件を追加した．具体的には図 7 のように，まず最下位層のユニットに着目し，あるユニットを処理する場合，最初のそのユニットの親を参照する．そして，親は子供のユニットの情報を保持しているので，同じ親を持つユニットに対してのみを探索処理の対象にする．

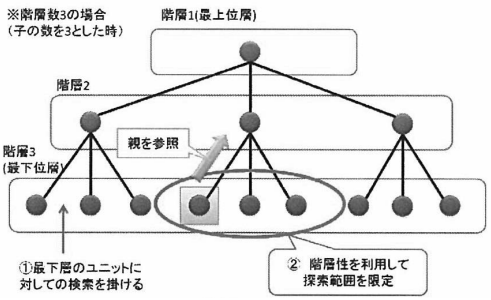


図 7　一番近いユニットの探索

このように探索範囲を限定できるのは結合ルールの変更により同じ親を持つユニットが集まるという階層構造の特性を利用しているためである．

上記の処理は衝突を低減するためのものであり，隣接ユニットが複数密集している場合には完全に衝突を回避できないことがある．この問題に対処するためには，一番近いユニットを探索するのではなく，ある一定距離内にあるユニットを探索対象に追加するなどして対処することが望ましい．ただし，探索処理に追加するユニットの数を多くすればするほど処理効率の低下を招く．

4.4　障害物回避

提案手法は Boid アルゴリズムを基に階層化を行っているので，Boid アルゴリズムの整列・結合・分離の 3 つのルール以外に，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も可能である．

本研究では，次の①と②を満たすようなベクトルを生成する処理を回避ルールとして Boid アルゴリズムに追加した．

①┌ ある一定距離内にある障害物から離れようとする．
② 障害物との距離が近ければ近いほど急激によける．

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 Boid アルゴリズムが実行されるため，障害物との距離がある一定距離内になると回避するベクトルが作成される．この回避する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．ここで前述した整列ルールの変更により，障害物の回避距離内にいない個体も群れ全体と向きをそろえようとするため，間接的に回避するベクトルが作成される（図 8 参照）．また，回避ルールを障害物ではなく，魚に対して適用することで，被食 - 捕食行為を表現することも可能である．

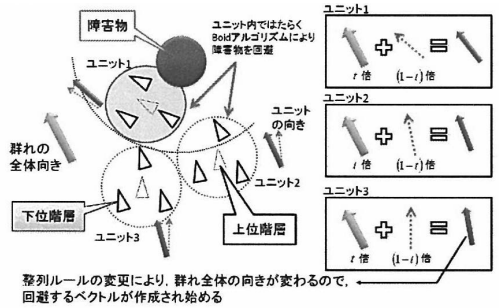


図 8　障害物の回避

4.5　階層的 Boid アルゴリズムの更新処理

階層的 Boid アルゴリズムでは，更新処理は最下位層のユニットから順に更新していく．具体的には，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，というようにボトムアップ的に行う．更新処理を行う際の上位階層と下位階層の関係を図 9 に示す．ボトムアップ的に更新処理を行うのは，4.2 節で述べた，衝突回避のために上位階層の個体に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．各階層の更新処理は図 10 (a)，各ユニットの更新は 図 10 (b)のよう

に示す手順で行う．具体的には，最下位層から更新処理を開始し，最下位層であった場合，一番近いユニットを探索し，各ユニットを更新する際の探索対象に追加しておく．次に各ユニット内の更新処理に移り，親ユニットから取得した平均速度ベクトルとユニット内の平均速度ベクトルを線形補間し，整列ルールを適用する．次に，親ユニットから取得した位置座標を各ユニットの中心位置とみなし，結合ルールを適用する．その後，分離ルールと回避ルールを適用し，各ルールによって得られたベクトルを基に各個体の位置を更新する．各ユニット内の更新終了後，ユニットごとのバウンディングスフィアを算出し，バウンディングスフィアの半径を親ユニットへと渡す．

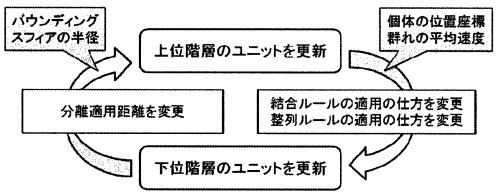
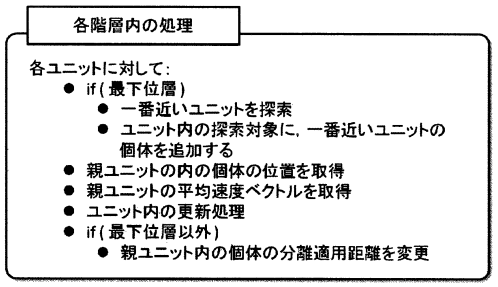
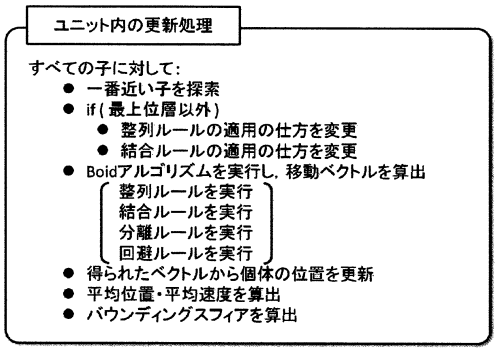


図 9 上位階層と下位階層の更新処理の関係



(a) 各階層内の処理



(b) ユニット内の更新処理

図 10 更新処理の概要

5. 実行結果

図 11 は，通常の Boid アルゴリズム，および階層的 Boid 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．グラフの曲線を見て分かるように，階層的 Boid アルゴリズムでは処理時間が大幅に改善され，階層数が増えるほど個体数に対する動作生成時間のグラフが直線に近づく様子が分かる．4.3 節に述べた一番近いユニットを探索する処理は，計算コストのオーダにあまり影響を与えていないことがわかる．

階層数を 3，表示数が 27000 体のときの実行画面をキャプチャーした様子を図 12，13，14 に示す．このとき描画に用いた魚 1 体あたりの三角形ポリゴンの数は 635 面である．図 12 は，被食者が捕食者から逃げようとする様子を示しており，図 13 は被食者が捕食者から逃げようとする際の群れ全体の様子を示したものである．階層的 Boid アルゴリズムにおいても，きちんと捕食者を避けている様子が分かる．図 14 は個体が集まり，群れが形成される様子を表している．このときの描画を含めたシミュレーションの処理は約 0.74fps である．

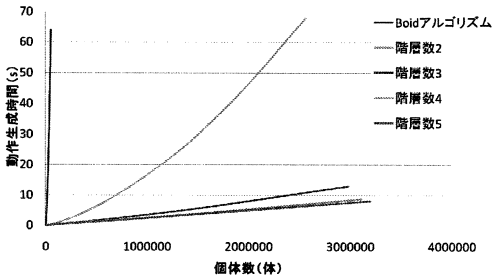


図 11 1 ステップあたりの動作生成時間

6. まとめ

大規模な魚群をシミュレーションするために，階層構造を用いることによって，探索処理を効率化し，動作生成処理を高速に行う階層的 Boid アルゴリズムを提案した．また，回避ルールを追加することにより，階層的 Boid アルゴリズムにおいても，障害物を回避が可能であること，階層的 Boid アルゴリズムにおいて，階層数が深くなればなるほど計算コストが線形に近づくことを実例として示した。

今後の課題としては，階層を用いるため階層ごとにシミュレーションのパラメータの設定を行っており，パラメータ数が多くなってしまう問題点の改善や，激しい動きをする群れに対する，ユニ

ットごとの動きの違いによるアーティファクトの低減，また，階層構造を用いた場合の魚の渦の形成などが挙げられる．本研究では動作生成処理のみに焦点を当てたが，効率的に群れを描画するためのアルゴリズムについても取り組む必要性がある．

参考文献

[1] Craig W. Reynolds, "Flocks, 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] Adrien Treuille, Seth Cooper, Zoran Popovic, "Continuum Crowd", ACM Transaction on Graphics, vol.25, No.3, Jul.(2006), pp.1160-1168.

[4] 佐藤大輔，吉田典正, "魚の群れの捕食—被食シーンにおける動作のリアルな表現", 画像電子学会第 34 回年次大会，2006, pp.71-72.

[5] Yoshiaki Ishibashi, Norimasa Yoshida, "View-dependent animation of a large school of fish", Image Electronics and Visual Computing Workshop, 1p-11, 2007.

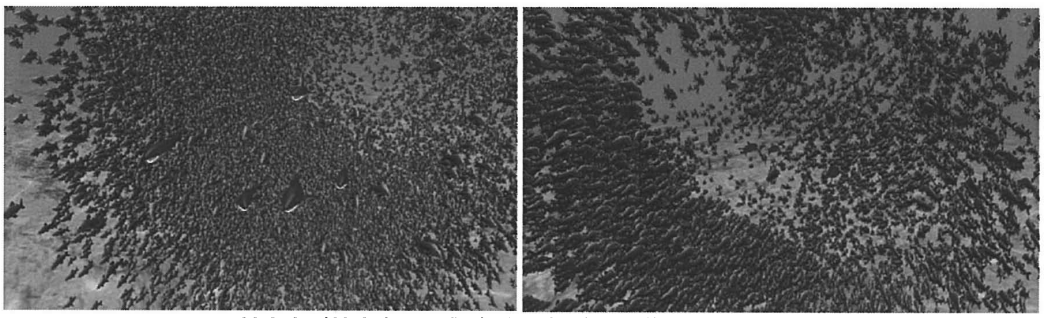


図 12 被食者が捕食者から逃げようとする様子(階層数 3，個体数 27,000)

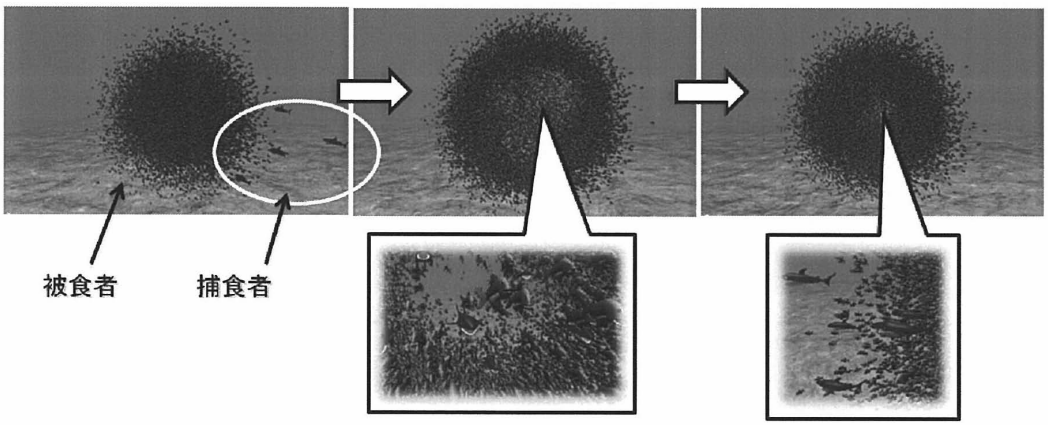


図 13 被食者が捕食者から逃げるときの群れ全体の様子(階層数 3，個体数 27,000)

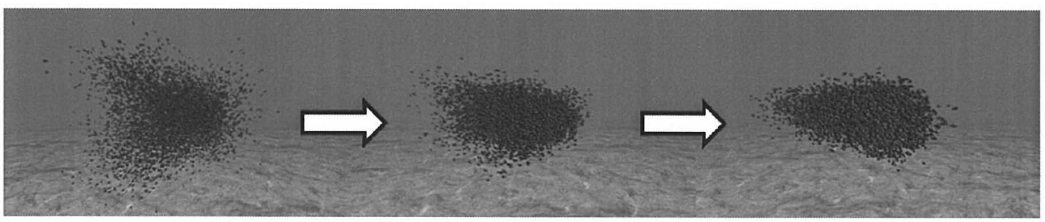


図 14 群れが形成される様子(階層数 3，個体数 27,000)