Clustered Shape Matching法における
再破断までを考慮した高速な破断面生成

筑波大学
図書館情報メディア研究科
2019年3月
辻　和徳

目 次

図 目 次

表 目 次

第 1 章

序論

コンピュータグラフィックス（以下，CG）は近年エンターテイメントなど様々な分野において用いられている．特に，水の流れや布の動きのような我々の身近に存在する物理現象の再現は CG アニメーションを製作する上で最も重要なことの一つである．従来，CG アニメーションでこれらの物理現象を再現するためにはクリエイタに多くの技術が求められ，またアニメーションの 1 フレームごとに細かく確認をしていかなければならないため，かかる負担も大きかった．しかし，近年は物理法則に基づくシミュレーション（以下，物理シミュレーション）によって自然現象を含むアニメーションをコンピュータが自動的に作成できるようになったことで，クリエイタの負担を大きく減らすことが可能になってきている．

エンターテイメント分野においてよく用いられている自然現象として，物体の破壊があげられる．物体の破壊は，一か所に応力が集中することで物体の変形の限界を超え、その箇所から破断が発生することによって生じる．従来，CG アニメーションにおいて破壊表現のために物理シミュレーションを用いる際には，有限要素法と呼ばれる手法が用いられていた．この手法は，物体を有限範囲の領域に分割し，それぞれの領域にかかる力から運動方程式を考慮して線形システムによる解析を行い，物体の運動をシミュレーションするものである．しかし，有限要素法は 1 ステップ毎に物体の運動の複雑な解析を行うため，計算コストが非常に高く，ゲーム等のようなリアルタイム性を求められるアプリケーションでの使用は難しい．そのため，リアルタイムアプリケーションではボロノイ図形に基づき物体の分割パターンを事前に生成し，物体に力がかかった際にそのパターンに応じて物体を分離させることで破壊を表現するという手法も用いられている．この手法は非常に高速に破壊表現を行うことができる一方で，幾何学的なパターン分割に基づくものであるため物理法則に基づいておらず，更に弾性体のように変形する物体には破断中に変形が起こることから分割パターンを当てはめることができないため，剛体によるシミュレーションでしか採用されていない．

そこで近年，物体の動きを高速に計算できる位置ベース法が注目されている．位置ベース法は，物体を構成する計算点に対して位置制約をかけることで物体の動きをシミュレーションすることができる手法である．この手法は高速に計算できるだけでなく，次ステップの位置を現ステップの位置と制約を用いて計算しているため，現ステップまでの力や速度の積み重ねにより次ステップの位置を決める力学ベース法よりも数値的に安定していることが特徴である．位置ベース法の中でも，Shape Matching 法は弾性体の変形をシミュレーションするため有限要素法のように局所的な部位の変形や破壊に対応できない．そこで，局所部位の変形に対応すべく Shape Matching 法を拡張したものとして，Clustered Shape Matching 法がある．Clustered Shape Matching 法は，複数の計算点から成るクラスタと呼ばれる小領域を物体内に多数設定し，それぞれのクラスタごとに変形計算を行うことで，物体の局所的な変形を扱えるようにしたものである．連なった複数のクラスタをそれぞれ分離していくことによって，物体の局所的な部位の破壊にも対応することができる．

破壊表現を Clustered Shape Matching 法を用いて行った手法として，Jones らの研究 [4] がある．Jones らは各クラスタの変形度を解析によって求め，その値が閾値を越えた場合にクラスタが分離するように設定することで破壊の表現を行っていた．しかし，破断は計算点で構成されたクラスタの分離のみで表現されており，破断後の各計算点の位置から破断面を含めたメッシュを生成するということは考慮されていなかった．破断面を含めた手法として，辻ら [5] の研究が挙げられる．辻らは，あらかじめ物体を多数の四面体で構成されたものとしてメッシュを生成し，クラスタと四面体を対応付け，破断したクラスタに合わせて四面体を分離させ新たな立体を生成することによって，破断面を表現した．しかし，四面体を破断して生じた立体が再度破断された場合のことを考慮されておらず，また四面体の分離にも大きな計算コストがかかっていた．

本論文では，Clustered Shape Matching 法を用いた破壊における破断面の生成手法の改良とその高速化について提案する．本論文では辻らのように物体を多数の四面体で構成し，破断によって生成される立体も動的に四面体分割することで再破断にも対応する．また，グラフィックス用のプロセッサである GPU を用いて，一部の計算を並列化し，CPU と GPU を両方用いたハイブリッド方式により，全体の計算を高速化できることを示す．

第2章

関連研究

2.1 破壊現象の物理シミュレーションを用いた表現

物体の破壊を表現するためには，応力が集中して物体が形を維持できる限度を超えた箇所を検出する必要がある．これを実現するために O'Brien ら [1] は有限要素法の使用に加えて連続体力学に基づき応力のテンソルを固有値解析を行うことで物体各部位の変形の度合いを調べ，破断する箇所を検出する方法を提案した．また，前計算によって物体が四面体で構成し，破断面も四面体の分割によって表現していた．Busaryev ら [3] は紙や金属膜のような薄い物体の破断を考慮するために，O'Brien らの方法を三次元空間中で定義した三角形メッシュの集合に適用した．これらの手法は，現実に起きうる破壊と変わりない高品質な破断を実現できているが、計算コストが高いため，リアルタイムで破壊計算を行うには向いていない．また，物体の表面のみを考慮して破断面を含むメッシュを効率的に扱う手法である境界要素法を用いて破壊表現を行う手法が Hahn ら [6] によって提案された．Hahn らは破壊表現を行うために，内部にかかる力と外部にかかる力を推定し，それによって起きる破断の位置，方向を計算した．また，破断後の物体の様子を大きさによって 4 つの種類に分け，ごく小さな欠片は破断計算を行わないようにした．これにより従来とほぼ変わらないクオリティを維持しつつ破壊によって生じる要素数が増えても計算コストが増加することを抑えることができた．しかし，境界要素法も有限要素法と同じ力から解析を行う手法であるため，リアルタイムアプリケーションに適用するには計算コストが大きかった．

2.2 ShapeMatching 法による破壊現象の表現

弾性体の破壊を扱うために，Müller ら [2] が提案した位置ベース法の一つである Shape Matching 法を用いた変形計算によって品質を維持しつつ，リアルタイムで破壊を表現できる実行できる手法が提案されている． その中でも複雑な変形を扱うものとして，物体を複数のクラスタと呼ばれる小領域に分割して各部位ごとに変形計算を行う Clustered Shape Matching 法を用いる方法 [2] と，物体を格子状の領域に分けて各格子ごとに変形計算を行う Lattice Shape Matching 法を用いるもの [7] がある．

Clustered Shape Matching 法を用いた破壊表現のための手法として，Jones ら [4] はクラスタ毎に変形計算で得られた変形の度合いを調べ，その値が閾値を越えた際にクラスタを破断方向に沿って二つに分離することで，延性破壊をシミュレートしていた．ただし，破断によって得られる新しい面の生成手法については考えられていなかった．これを解決するために，辻ら [5] は物体を四面体で構成し，クラスタが破断した際に対応する四面体も合わせて分離して新しい立体を生成することによって破断面の表現を実現していた．しかし，破断によって生じた新しい立体が更に破断した場合について考慮されておらず，クラスタ

数が少ない場合はリアルタイム実行可能である一方で，クラスタ数が増加した場合の多大な計算時間が問題であった．

Ohta ら [8] は Lattice Shape Matching 法を用いた破壊表現が提案した．この論文では，格子状の集合の中にある複数の計算点について，それぞれの近隣にある計算点の間の距離を計算し，互いの距離が近いもの同士によって構成された二つの格子に分離する．ただし，このままだと破断面が格子状になってしまうため，パーリンノイズをかけて面の座標を修正していた．しかし，破断面から生じる更なる破断については考慮されておらず，生成された破断面もノイズに基づくためリアリティに欠けるものであった．

本論文は，Jones ら，および辻らの手法をベースにし，GPU による高速化の実装および破断面を構成する立体のさらなる破断を考慮することで，リアルタイム性を向上しつつよりリアリスティックな破壊表現を可能とする手法を提案する．

第3章

提案手法の流れ

提案手法は下記に示した手順で実行する．

1. Clustered Shape Matching 法による変形計算 (4.1 節，4.2 節)
2. クラスタの破断判定・破断方向の計算 (4.3 節)
3. クラスタの破断処理 (4.3 節)
4. 破断面を構成する立体の分離と再構成 (5.1 節，5.2 節)

1 では，Clustered Shape Matching 法と呼ばれる手法を用いて物体内のクラスタと呼ばれる小領域ごとに弾性体の変形を計算する．このとき，内部の構造についても考慮するため物体を多数の四面体で構成し，変形計算を行う．四面体構造はクラスタを構成するために用い，また破断面形成にも用いる．

2 では，1 で得られたクラスタの変形行列から固有値解析に基づき変形度を求め，破断が起きるかどうかを判定するとともに，固有値解析で得られた固有ベクトルを用いて破断方向を決定する．このとき，破断が起きるクラスタはすべてのクラスタの中で最も変形度が大きいものであるとする．

3 では，2 で破断すると選択したクラスタについて実際に破断処理を行う．

4 では，破断面を形成するために，クラスタ内で定義された立体 (四面体) の分離処理を行う．このステップで新しい計算点も生成される．

全体の変形計算のフローチャートは図 3.1 の通りである．このとき，1 と 2 に該当する部分は GPU 側で行い (6.1 節)，3 と 4 に該当する部分は CPU 側で行い，更に，破断処理の後に CPU と GPU の両側で保持されているデータを同期させる．(6.2 節)

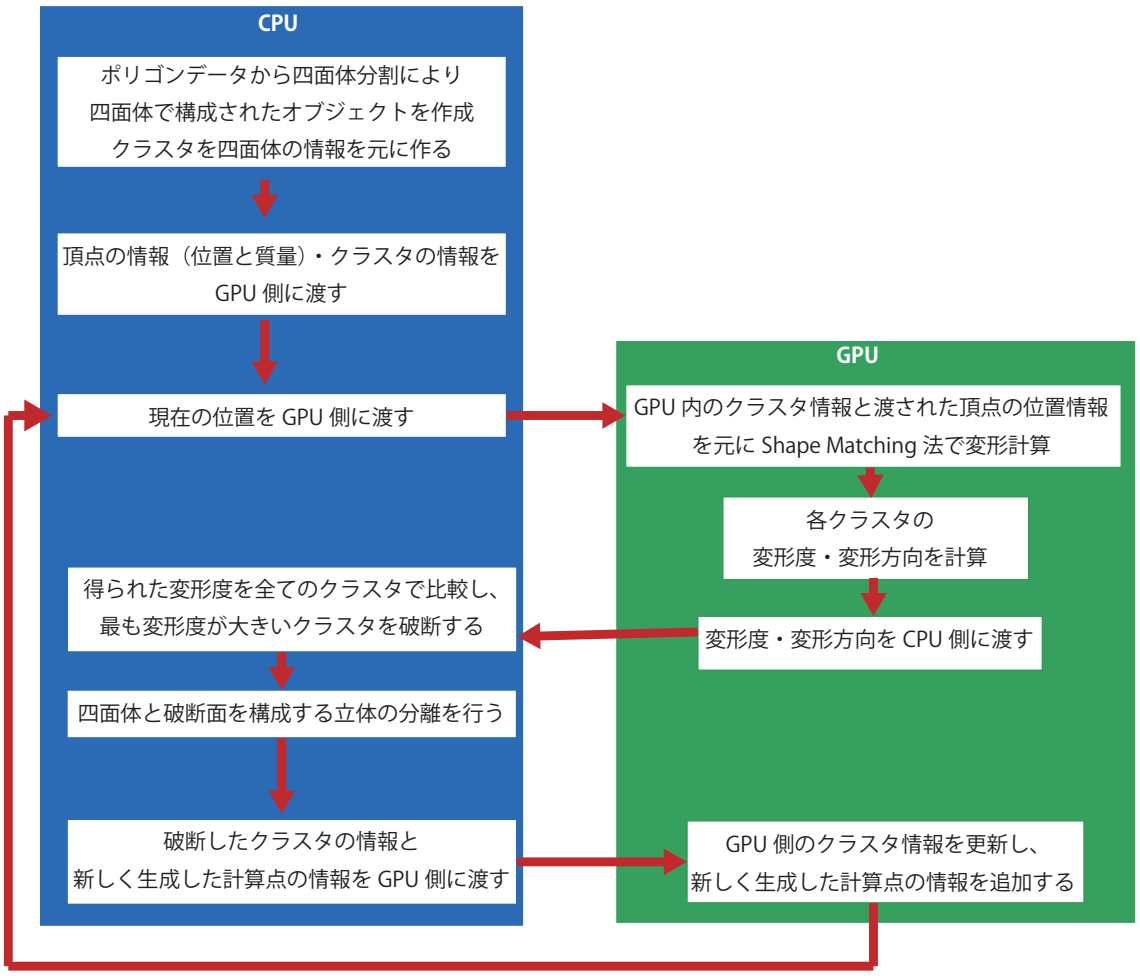

CPU
ポリゴンデータから四面体分割により
四面体で構成されたオブジェクトを作成
クラスタを四面体の情報を元に作る
頂点の情報（位置と質量）・クラスタの情報を
GPU 側に渡す
現在の位置を GPU 側に渡す
得られた変形度を全てのクラスタで比較し、
最も変形度が大きいクラスタを破断する
四面体と破断面を構成する立体の分離を行う
破断したクラスタの情報と
新しく生成した計算点の情報を GPU 側に渡す
GPU
GPU 内のクラスタ情報と渡された頂点の位置情報
を元に Shape Matching 法で変形計算
各クラスタの
変形度・変形方向を計算
変形度・変形方向を CPU 側に渡す
GPU 側のクラスタ情報を更新し、
新しく生成した計算点の情報を追加する

図 3.1: 処理の流れ

第4章

Shape Matching法とそれを用いた破壊表現

ゴムのような弾性体の物理シミュレーションを実行するため，提案手法では Shape Mathing 法 [2] を用いる． Shape Matching 法は物体を多数の計算点によって構成されたものと見なし，シミュレーション初期の計算点の位置関係と現ステップの位置関係を使って，弾性体の変形した形状が元に戻る性質をシミュレーションする手法である．弾性体では形状は元に戻ろうとするが，全体の回転と平行移動については保存されるべきである． そのために，現在の位置関係から回転成分と平行移動成分を残して元の位置関係に戻るように各計算点の位置を直接計算する．位置を直接計算できるということは有限要素法のような力学ベース法で用いられているような力と速度の時間積分を考慮する必要がなくそれ故に誤差の蓄積が起きえないため，安定したシミュレーションを実現できる．また，変形計算にかかるコストが非常に少ないため，リアルタイム性を要求されるアプリケーションにも応用できる．以下，Shape Matching 法とそれを用いた破壊表現のための計算手順について説明する．

4.1 Shape Matching法の説明

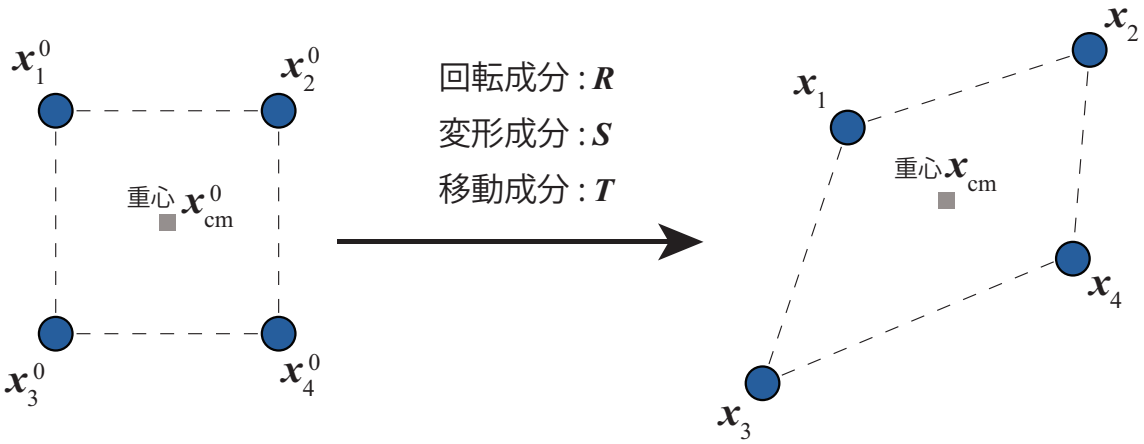


図 4.1: 初期形状からの変形

ある物体を構成する N 個の計算点 $\boldsymbol{x}_k(1 \leq k \leq N)$ について考える．図 4.1 は 4 個の計算点からなる二次元上にある物体の変形を示している．シミュレーション開始時，物体は図 4.1 左のような形状であるとする．この形状を元にシミュレーションを行う．このとき，各計算点の位置を $\boldsymbol{x}_k^0$(図 4.1 左の青い点) として，$\boldsymbol{x}_k$(図 4.1 右の青い点) とは別途記録してお

く．シミュレーションが進むにつれて，物体は図 4.1 右のように外力などによる変形を受ける．物体が受ける変形は，回転成分を表す行列 $\boldsymbol{R}$・変形成分を表す行列 $\boldsymbol{S}$・平行移動成分を表すベクトル $\boldsymbol{T}$ の 3 つに分解することができ，各計算点の現ステップでの位置 $\boldsymbol{x}_k$ は以下の式 (4.1) で表すことができる．

$$\boldsymbol{x}_k = \boldsymbol{R}\boldsymbol{S}(\boldsymbol{x}_k^0 - \boldsymbol{x}_{cm}^0) + \boldsymbol{T} \tag{4.1}$$

ここで，$\boldsymbol{x}_{cm}^0$ は計算点 $\boldsymbol{x}_k^0$ を全て用いて計算された重心座標を表し，式 (4.1) 右辺の括弧内に記された式は，シミュレーション初期状態における各計算点の物体の重心からの相対座標を示している．

Shape Matching 法は図 4.1 右のように変形した後に，元の形状に戻ろうとする性質をシミュレーションする．この原理を図 4.2 に示す．シミュレーション初期の形状に回転成分と平行移動成分を加えたものを図 4.2 左のような目標形状とし，その形状に合わせるように各計算点を移動させることで，全体の回転と平行移動を残したまま，弾性体の元の形に戻る性質をシミュレーションできる．各計算点の目標座標 (図 4.2 の赤点) を $\boldsymbol{g}_k$ とすると，その求め方は式 (4.2) のようになる．

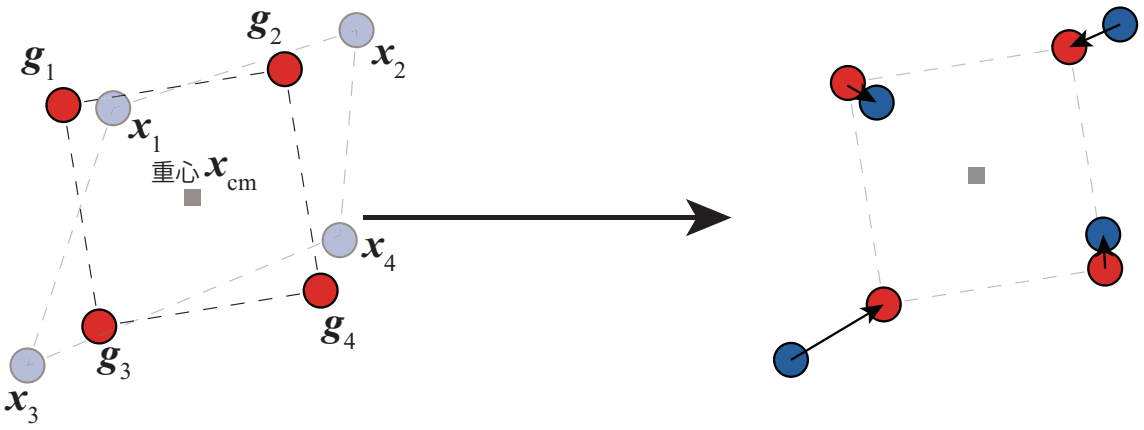


図 4.2: 目標座標への移動

$$\boldsymbol{g}_k = \boldsymbol{R}(\boldsymbol{x}_k^0 - \boldsymbol{x}_{cm}^0) + \boldsymbol{T} \tag{4.2}$$

目標座標を計算した後，図 4.2 右の青い点のように，計算点を $\boldsymbol{g}_k$ に向けて移動させることにより，弾性体の変形を実現することができる．以上が Shape Matching の大まかな計算の流れである．

目標座標を計算する上で必要な回転成分 $\boldsymbol{R}$ と平行移動成分 $\boldsymbol{T}$ は，各計算点の質量 m_k とシミュレーション初期における物体の重心座標 $\boldsymbol{x}_{cm}^0$ を用いて以下の最小化問題を示す式 (4.3) により求められる．

$$\arg\min_{\boldsymbol{R},\boldsymbol{T}} \sum_{k=1}^{N} m_k |\boldsymbol{g}_k - \boldsymbol{x}_k|^2 \tag{4.3}$$

式 (4.3) は目標座標 $\boldsymbol{g}_k$ と現在座標 $\boldsymbol{x}_k$ の差分に関する最小化問題となっている．この式を最小にする $\boldsymbol{R}$ と $\boldsymbol{T}$ を求めることにより，適切な $\boldsymbol{g}_k$ を設定することができる．このとき，平行移動成分 $\boldsymbol{T}$ はシミュレーション空間上の原点から物体重心へのベクトルで表され，以下

の式 (4.4) により求めることができる．

$$\boldsymbol{T} = \boldsymbol{x}_{cm} = \frac{\sum_{k=1}^{N} m_k \boldsymbol{x}_k}{\sum_{k=1}^{N} m_k} \tag{4.4}$$

式 (4.4) を式 (4.3) に代入し，式を簡略化するために変数 $\boldsymbol{s}_k = \boldsymbol{x}_k^0 - \boldsymbol{x}_{cm}^0$，$\boldsymbol{p}_k = \boldsymbol{x}_k - \boldsymbol{x}_{cm}$ を定義することで，最小化問題を式 (4.5) のように簡略化し，回転行列 $\boldsymbol{R}$ 一つだけに関する問題に変換することができる．

$$\arg\min_{\boldsymbol{R}} \sum_{k=1}^{N} m_k |\boldsymbol{R}\boldsymbol{s}_k - \boldsymbol{p}_k|^2 \tag{4.5}$$

ここで，変形成分 $\boldsymbol{S}$ に対応するために式 (4.5) を拡張する．線形変換行列 $\boldsymbol{A} = \boldsymbol{R}\boldsymbol{S}$ の最小化問題を以下の式 (4.6) で定義する．

$$\arg\min_{\boldsymbol{A}} \sum_{k=1}^{N} m_k |\boldsymbol{A}\boldsymbol{s}_k - \boldsymbol{p}_k|^2 \tag{4.6}$$

式 (4.6) を最小にする $\boldsymbol{A}$ を求めるためには，極小値を取るように，すなわち式 (4.6) の行列 $\boldsymbol{A}$ の要素 a_{ij} で微分したとき 0 になるようにして解けばよい．新たに
$\boldsymbol{E} = \sum_{k=1}^{N} m_k |\boldsymbol{A}\boldsymbol{s}_k - \boldsymbol{p}_k|^2$ を定義して，式 (4.6) を微分すると式 (4.7) のようになる．

$$\frac{\partial \boldsymbol{E}}{\partial a_{ij}} = \sum_{k=1}^{N} 2m_k (\boldsymbol{A}\boldsymbol{s}_k - \boldsymbol{p}_k)\boldsymbol{s}_k^T = 0 \tag{4.7}$$

式 (4.7) を $\boldsymbol{A}$ について解くと，以下の式 (4.8) が得られる．

$$\boldsymbol{A} = (\sum_{k=1}^{N} m_k \boldsymbol{p}_k \boldsymbol{s}_k^T)(\sum_{k=1}^{N} m_k \boldsymbol{s}_k \boldsymbol{s}_k^T)^{-1} = \boldsymbol{A}_{ps}\boldsymbol{A}_{ss}^{-1} \tag{4.8}$$

ここで，$\boldsymbol{A}_{ps} = \sum_{k=1}^{N} m_k \boldsymbol{p}_k \boldsymbol{s}_k^T$，$\boldsymbol{A}_{ss} = \sum_{k=1}^{N} m_k \boldsymbol{s}_k \boldsymbol{s}_k^T$ とした．$\boldsymbol{A}_{ss}$ は対称行列であり，拡大縮小による変形成分を含んでいる．$\boldsymbol{A}_{ps}$ は回転成分 $\boldsymbol{R}$ と変形成分 $\boldsymbol{S}$ を含んだ連続体における変形勾配テンソルを示す行列である．従来，回転成分と変形成分を分離するために極分解と呼ばれる手法が使用されていた [2] が，その結果を出すためのアルゴリズムでは薄い破片のように計算点が一つの平面上に集まったときに，行列のランクが小さくなり，計算が不安定になるケースがあった．そこで，本論文では変形計算の際に Müller ら [9] が提案した $\boldsymbol{A}_{ps}$ を変換した四元数 $\boldsymbol{q}$ を使った方法を用いる．$\boldsymbol{A}_{ps}$ と計算開始時に単位行列で初期化されている行列 $\boldsymbol{B}$ の各々の列ベクトルを利用して $\boldsymbol{q}$ を反復計算により修正し，各反復での $\boldsymbol{q}$ の結果を $\boldsymbol{B}$ に反映させ，反復計算の誤差が無くなった際に $\boldsymbol{B}$ を回転成分 $\boldsymbol{R}$ とし，その逆行列を $\boldsymbol{A}_{ps}$ の右からかけることで変形成分 $\boldsymbol{S}$ に分解する．以下その過程について説明する．

この手法の目的は $\boldsymbol{A}_{ps}$ を二つの成分に分解した際に，最小限の回転エネルギーを持つ回転成分 $\boldsymbol{R}$ を探すことである．$\boldsymbol{A}_{ps}$ の列ベクトルを $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$ とし，$\boldsymbol{R}$ の列ベクトルを $\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3$，$\boldsymbol{B}$ の列ベクトルを $\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3$ と定義し，回転成分に関するフロベニウスノルム $F(\boldsymbol{R}) = ||\boldsymbol{A}_{ps} - \boldsymbol{R}||^2$ を定義する．$F(\boldsymbol{R})$ について，そのエネルギーを最小化するような $\boldsymbol{R}$ が求められるため，$F(\boldsymbol{R})$ のエネルギーを E_F と定義すると，以下のエネルギー最小化問題式 (4.9) が定義される．

$$\arg\min_{\boldsymbol{R}} E_F = \arg\min_{\boldsymbol{R}} \frac{1}{2} F(\boldsymbol{R}) = \frac{1}{2}(\sum_{i=1}^{3} (\boldsymbol{r}_i - \boldsymbol{a}_i)^2) \tag{4.9}$$

$\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$ は反復計算中も定数であるため，式 (4.9) は $\boldsymbol{R}$ のみについての最小化問題に帰結する．

また，$\boldsymbol{R}$ を剛体の回転と考え，その回転を与えるように各軸周りにかかる力を $\boldsymbol{f}_i$ とすると，$\boldsymbol{f}_i$ は E_F を各列ベクトル $\boldsymbol{r}_i$ について微分することで求めることができるので，式 (4.10) のようになる．

$$\boldsymbol{f}_i = -\frac{\delta E_F}{\delta \boldsymbol{r}_i} = \boldsymbol{a}_i - \boldsymbol{r}_i \tag{4.10}$$

式 (4.10) より，$\boldsymbol{f}_i$ によってかかるトルクを τ とすると，式 (4.11) のようになる．

$$\tau = \Sigma_i \boldsymbol{r}_i \times \boldsymbol{f}_i = \Sigma_i \boldsymbol{r}_i \times \boldsymbol{a}_i \tag{4.11}$$

Müller ら [9] により，$\tau = 0$ となることが式 (4.11) の最小化を達成するための必要十分条件となっていることが示されている．式 (4.11) を元に，τ による角速度ベクトルを ω と定義し，$\omega = \boldsymbol{a}_i \times \boldsymbol{r}_i$ とする．ω のノルム $|\omega|$ は反復内における前後二つの ω の差を示す角度 ϕ を定義すると $|\omega| = |\boldsymbol{a}_i||\boldsymbol{r}_i| \sin\phi$ となる．ϕ を剛体の回転角と考えると，$|\omega| = \phi$ となるように ω を設計することで，$\tau = 0$ となる [9]．そこで，ω を正規化し再定義する．

$$\omega = \frac{\boldsymbol{a}_i \times \boldsymbol{r}_i}{|\boldsymbol{a}_i \cdot \boldsymbol{r}_i|} \tag{4.12}$$

式 (4.12) により，ϕ が十分小さい場合，以下の式 (4.13) が成り立つ．

$$|\omega| = \frac{|\boldsymbol{a}_i||\boldsymbol{r}_i| \sin\phi}{|\boldsymbol{a}_i||\boldsymbol{r}_i| \cos\phi} = \tan\phi \approx \phi \tag{4.13}$$

式 (4.13) より式 (4.12) の ω は $\tau = 0$ となる，つまり式 (4.9) を最小化するような回転を表す．そのため，反復計算において $\boldsymbol{B}$ を四元数に変換したものを $\boldsymbol{q}$ とすると，式 (4.14) に示した計算 (姿勢 q の ω による回転) の繰り返しによって式 (4.8) を最小化する $\boldsymbol{R}$ に $\boldsymbol{q}$ を近づけることができる．

$$\boldsymbol{q} \leftarrow \frac{\sum_{i=1}^{3}(\boldsymbol{a}_i \times \boldsymbol{r}_i)}{\sum_{i=1}^{3}(\boldsymbol{a}_i \cdot \boldsymbol{r}_i) + \epsilon}\boldsymbol{q} \tag{4.14}$$

ϵ は分母が 0 になることを回避するための数であり，本論文では Müller ら [9] に従い $\epsilon = 1.0 \times 10^{-9}$ と定義する．

反復計算により $|\omega|$ がやがて閾値より下回る．本論文では実装言語に C++を用いているため，その閾値は float 型の精度限度 (有効数字 7 桁) に基づき 1.0×10^{-6} とする．閾値を下回る，または反復計算回数を突破した際，反復計算で最終的に得られた $\boldsymbol{B}$ を $\boldsymbol{R}$ とし，式 (4.15) で $\boldsymbol{S}$ を計算する．

$$\boldsymbol{S} = \boldsymbol{R}^{-1}\boldsymbol{A}_{ps} \tag{4.15}$$

以上の手順により，$\boldsymbol{A}_{ps}$ から回転成分 $\boldsymbol{R}$，歪み成分 $\boldsymbol{S}$ を取り出すことができた．$\boldsymbol{A}_{ss}$ は，変形の計算には使われないが，物体の体積保存を考えた変形を行うためには，$\det\boldsymbol{A}_{ss} = 1$ を満たす必要があるため，変形計算の前に計算して，チェックする．

n ステップ目に働く外力 $\boldsymbol{f}^n_{external}$，タイムステップ幅 Δt を考えると，速度と位置の更新式は以下の式 (4.16)，式 (4.17) となる．

$$\boldsymbol{v}_k^{n+1} = \boldsymbol{v}_k^n + \alpha\frac{\boldsymbol{g}_k^n - \boldsymbol{x}_k^n}{\Delta t} + \frac{\Delta t \boldsymbol{f}^n_{external}}{m_k} \tag{4.16}$$

$$\boldsymbol{x}_k^{n+1} = \boldsymbol{x}_k^n + \Delta t \boldsymbol{v}_k^{n+1} \tag{4.17}$$

α は物体の剛性に関するパラメータである．$0 < \alpha < 1$ のとき，式 (4.12) より物体は目標形状へ徐々に近づくようにふるまうため，弾性体のシミュレーションができる．一方，$\alpha = 1$ のとき，物体の形状は常に変形を考えない目標形状になるため，剛体のような物体をシミュレーションできる．

4.2 Clustered Shape Matching 法

4.1 節で説明した Shape Matching 法は，物体全体で 3×3 の 1 つの変形行列を用いて計算するため，物体の局所的な変形には対応できない．そこで，物体をクラスタと呼ばれる小領域に分割し，各クラスタで Shape Matching 法の計算を行うことで局所的な変形に対応させる．このとき，図 4.3 に示したように，クラスタを重複して定義し，局所的な変形を段階的に物体全体に伝播させることでより複雑な変形が可能となる．この手法は Clustered Shape Matching 法と呼ばれる．

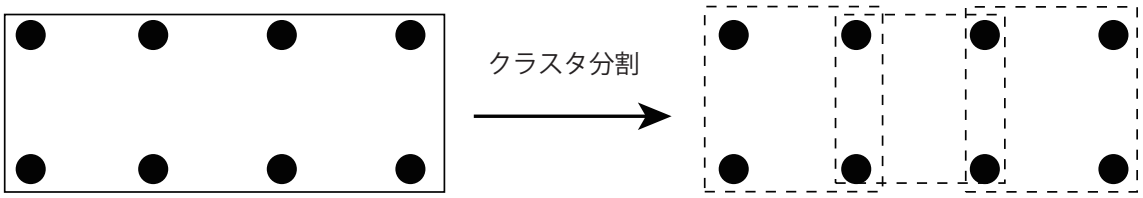


図 4.3: 物体のクラスタ分割 （破線で囲まれた領域がクラスタ内部

Clustered Shape Matching 法は，目標形状をクラスタごとに決定する．クラスタの回転成分を表す行列を $\boldsymbol{R}_c$，クラスタの平行移動ベクトルを $\boldsymbol{T}_c$，クラスタの初期重心位置を $\boldsymbol{c}_{cm}^0$ とすると，式 (4.2) から目標座標 $\boldsymbol{g}_k$ に関する式 (4.16) が導かれる．

$$\boldsymbol{g}_k = \boldsymbol{R}_c(\boldsymbol{x}_k^0 - \boldsymbol{c}_{cm}^0) + \boldsymbol{T}_c \tag{4.18}$$

ここで，Shape Matching 法では，$\boldsymbol{A}_{ps}$ の元となるベクトル $\boldsymbol{p}_k$ を $\boldsymbol{p}_k = \boldsymbol{x}_k - \boldsymbol{c}_{cm}$ と定義していた．Clustered Shape Matching 法においても，クラスタの現在における重心を $\boldsymbol{c}_{cm}$ として $\boldsymbol{p}_k = \boldsymbol{x}_k - \boldsymbol{c}_{cm}$ と定義すればよい．ただし，このような $\boldsymbol{p}_k$ を定義しても各クラスタを独立に計算するだけでは，他のクラスタからの影響を受けず，変形を伝播させることができない．そこで，既に目標座標 $\boldsymbol{g}_k$ を得た計算点については，$\boldsymbol{p}_k = \boldsymbol{g}_k - \boldsymbol{c}_{cm}$ として $\boldsymbol{p}_k$ を更新し，更新された $\boldsymbol{p}_k$ を用いて，その点を含む別のクラスタで変形計算を行うことで，他のクラスタへ変形を伝播させることができる．

また，Clustered Shape Matching 法では，反復によって物体の硬さを調整できる [11]．図 4.4 は反復による硬さの調整について説明したものである．図 4.4 右での赤い点は変形により移動した計算点を表している．物体のある計算点 (図 4.4 左の白丸) が引っ張られ，そこから変形が始まるとする．このとき，反復を 1 回行う（図 4.4 右上）と，引っ張られた点が属するクラスタが変形し，その変形は隣接するクラスタまで伝播する．図 4.4 右上を見て分かるように，このままではこの隣接クラスタは大きく変形している．このような局所的な変形が起こるのは柔らかい物体の場合である．一方，反復を 2 回行う（図 4.4 右下）と，隣接したクラスタの変形がその隣のクラスタまで伝わり，またクラスタごとの変形も小さくなる．このように，反復計算によって局所的な運動が物体全体に伝わり変形が収束することで，より硬い物体を計算することが可能となる．

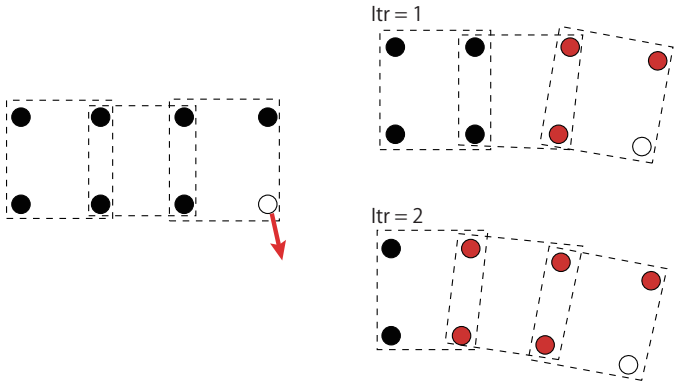


図 4.4: 反復処理による物体の硬さの調整

本論文では内部点についても考えるために，Hang ら [10] の方法を用いて物体を四面体分割して各頂点を計算点とし，その頂点（図 4.5 の中心点）とエッジで直接接続された近傍頂点（図 4.5 周りの点．以下，1-ring neighbor）でクラスタを構成する [11]．

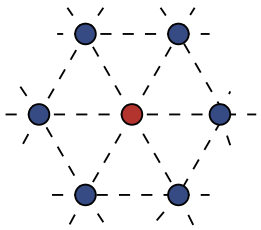


図 4.5: 接続関係による近傍頂点の探索

4.3 Shape Matching 法による破壊発生箇所検出

物理シミュレーションにおいて破壊を取り扱うためには，応力が集中して変形の限度を超えて破断が始まる箇所を検出しなければならない．連続体力学に基づいた手法では，応力を解析するために応力分布を表すテンソル（以下，応力テンソル）の固有値解析を行う．ここでは O'Brien らの手法 [1] について説明する．まず，シミュレーション空間が 3 次元であるとして，ベクトル $\boldsymbol{a} \in \mathbf{R}^3$ について 3×3 の正方対称行列 $\mathbf{m}(\boldsymbol{a})$ を式 (4.19) のように定義する．

$$\mathbf{m}(\boldsymbol{a}) = \frac{\boldsymbol{a}\boldsymbol{a}^T}{|\boldsymbol{a}|} \tag{4.19}$$

このとき，微小領域に働く応力の分布を表す応力テンソル $\boldsymbol{\sigma}$ は上記の式 (4.19) で定義した対称行列 $\mathbf{m}(\boldsymbol{a})$，応力テンソルの固有値 $e^i(\boldsymbol{\sigma})$ $(1 \leq i \leq 3)$，それに対応する固有ベクトル $\hat{\boldsymbol{n}}^i(\boldsymbol{\sigma})$ を用いて，式 (4.20)，(4.21) のように引張成分 $\boldsymbol{\sigma}^+$ と圧縮成分 $\boldsymbol{\sigma}^-$ に分解できる．

$$\boldsymbol{\sigma}^+ = \sum_{i=1}^{d} \max(0, e^i(\boldsymbol{\sigma}))\mathbf{m}(\hat{\boldsymbol{n}}^i(\boldsymbol{\sigma})) \tag{4.20}$$

$$\boldsymbol{\sigma}^- = \sum_{i=1}^{d} \min(0, e^i(\boldsymbol{\sigma}))\mathbf{m}(\hat{\boldsymbol{n}}^i(\boldsymbol{\sigma})) \tag{4.21}$$

ここで，計算点の重心座標系における位置 $\boldsymbol{b}$ の各成分 $\boldsymbol{b}_i(1 \leq i \leq 4)$ を使って，次のような行列 $\boldsymbol{\beta}$ を定義する．

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 & \boldsymbol{b}_3 & \boldsymbol{b}_4 \\ 1 & 1 & 1 & 1 \end{pmatrix}^{-1} \tag{4.22}$$

式 (4.22) で得られた $\boldsymbol{\beta}$ と $\boldsymbol{\sigma}^+$，物体の体積 vol，そして $\boldsymbol{b}_i$ を使うことで，引張力 $\boldsymbol{f}^+_{[i]}$ を以下の式のように得ることができる．

$$\boldsymbol{f}^+_{[i]} = \frac{vol}{2} \sum_{j=1}^{4} \boldsymbol{b}_j \sum_{k=1}^{3} \sum_{l=1}^{3} \boldsymbol{\beta}_{jl}\boldsymbol{\beta}_{ik}\boldsymbol{\sigma}^+_{kl} \tag{4.23}$$

圧縮力 $\boldsymbol{f}^-_{[i]}$ についても，$\boldsymbol{\sigma}^-$ を使うことで式 (4.23) と同様に求めることができる．その後，全体の分離テンソル $\boldsymbol{\varsigma}$ を以下の式 (4.24) のように計算する．

$$\boldsymbol{\varsigma} = \frac{1}{2}(-\mathbf{m}(\boldsymbol{f}^+) + \sum_{\boldsymbol{f} \in \{\boldsymbol{f}^+\}} \mathbf{m}(\boldsymbol{f}) + \mathbf{m}(\boldsymbol{f}^-) - \sum_{\boldsymbol{f} \in \{\boldsymbol{f}^-\}} \mathbf{m}(\boldsymbol{a})) \tag{4.24}$$

$\{\boldsymbol{f}^+\}$，$\{\boldsymbol{f}^-\}$ をそれぞれ各計算点に作用する引張力，圧縮力をまとめた集合とした．$\boldsymbol{\varsigma}$ の最大固有値が閾値を超えた場合，最大固有値に対応する計算点から破断が始まる．このとき，最大固有値に対応する固有ベクトルと垂直の方向に破断が進展する．

一方，Shape Matching 法では応力の解析はできないが，応力テンソル $\boldsymbol{\sigma}$ は Shape Matching 法の変形成分 $\boldsymbol{S}$ に対応しており，$\boldsymbol{S}$ を固有値解析することでその引張成分，圧縮成分を得ることができる．式 (4.20)，式 (4.21) より引張成分・圧縮成分の最も大きい方向はそれぞれ最大固有値，最小固有値のそれぞれに対応する固有ベクトルである．また，局所的な計算を可能とする Clustered Shape Matching 法を用いれば，全体の分離テンソル $\boldsymbol{\varsigma}$ の局所的な成分を $\boldsymbol{S}$ で表すことができ，従って $\boldsymbol{S}$ の最大固有値が閾値を超えた場合にクラスタが破断し，その破断方向は最大固有ベクトルと垂直の方向であると定義できる．

本論文では，Jones ら [4] の手法をベースとして弾性体における破壊発生箇所の検出を行う．全体的な手順としては以下の通りとなる．

1. 各クラスタについて運動計算を行い，最後の反復計算で S を固有値分解を行うことで最大固有値 λ_{max} を取得する
2. λ_{max} が変形限界の閾値 γ_{max} を超えたクラスタまたはそれ以外で λ_{max} が最も大きいクラスタを 1 タイムステップにおいて破断計算を行うクラスタとし，このクラスタを 2 つに分離するために新しいクラスタを生成する
3. 各計算点に対してクラスタの最初の要素とのエッジを考え，そのエッジが破断方向と交差する場合，計算点を新しいクラスタに所属させる
4. 分離したクラスタが以後数ステップまで破断しないように設定する
5. 2 で選ばれたクラスタの近傍クラスタすべてについて 2 〜 4 を行う
6. 5 で計算を行ったクラスタの更に近傍にあるクラスタのうち，破断方向の先にあるものを優先破断リストに加える

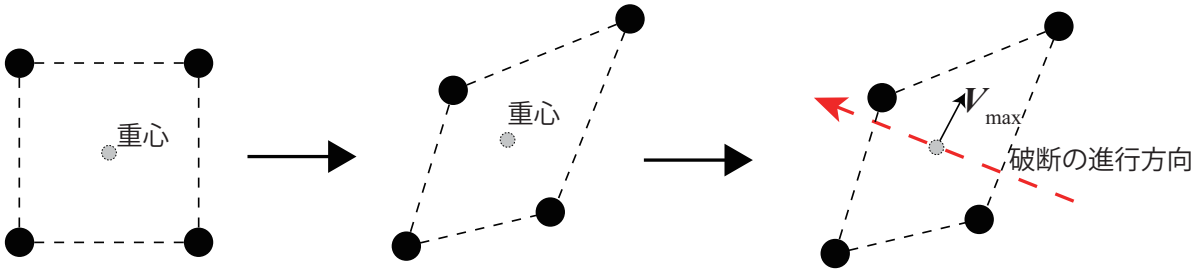


図 4.6: クラスタの破断

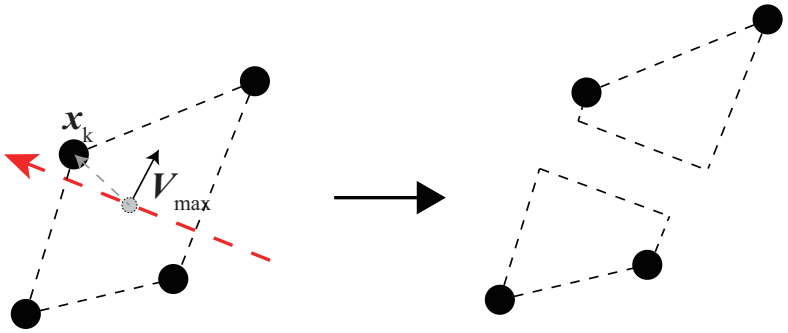


図 4.7: クラスタの分離

以下，これらの手順について詳細な説明を行う．

破壊が発生した場所を特定するためには，局所的な変形の検出が必要である．Shape Matching 法では全体の変形を 1 つの行列で表すため，局所的な変形の検出はできない．そのため，4.2 節で説明した Clustered Shape Matching 法を用いる．節の始めに説明したように，Clustered Shape Matching 法においてはクラスタの変形成分 $\boldsymbol{S}$ の最大固有値 λ_{max} がユーザの設定した閾値 γ を超えた場合に破断が発生すると定義できる．λ_{max} は反復による変形計算の最後のステップで固有値分解することで得られる．ここで，多数の亀裂が同時に発生するのを防ぐため，本論文では全てのクラスタについて最も大きな λ_{max} を持ちかつその λ_{max} が γ を超えるクラスタをそのタイムステップで破断するクラスタとする．

破断するクラスタが決まったら，図 4.6 のようにそのクラスタの変形成分 $\boldsymbol{S}$ の最大固有ベクトル $\boldsymbol{V}_{max}$ を使って破断方向を決める．最大固有ベクトルはクラスタにかかる引張力が最も大きい方向を示し，破断によって作られる新しい面（以下，破断面）は $\boldsymbol{V}_{max}$ と垂直な面となる．方向だけでは面の位置が定まらないため，ここでは図 4.6 のように破断面はクラスタの重心を通ると仮定する．

破断によって，クラスタは破断面に沿って 2 つに分離すると仮定すると，図 4.7 のようになる．ここで，図 4.7 右において分離に新しく生じる 2 つのクラスタをクラスタ A，クラスタ B とする．このとき，まず図 4.7 左に示すように，クラスタ内の計算点の位置 $\boldsymbol{x}_k$ と重心 $\boldsymbol{c}_{cm}$ の差のベクトルと $\boldsymbol{V}_{max}$ との内積 $\boldsymbol{d}_k = (\boldsymbol{x}_k - \boldsymbol{c}_{cm}) \cdot \boldsymbol{V}_{max}$ を計算する．内積の正負によってクラスタ A もしくはクラスタ B のいずれかに所属させる．

ここで 4.2 節で記述したクラスタ生成の方法から，計算点は 1 つのクラスタに所属しているとは限らないため，図 4.8 左に示すように 2 つの計算点が所属するクラスタが破断によって分けられても，どちらの計算点もそのクラスタとは別のクラスタに存在し破断が完全に進まない場合がある．そのため，図 4.8 中央および右に示すように破断が起きたクラ

スタと隣接するクラスタについても破断計算を行う．破断方向は元のクラスタのそれと同じものであると仮定し，同様の計算を行う．

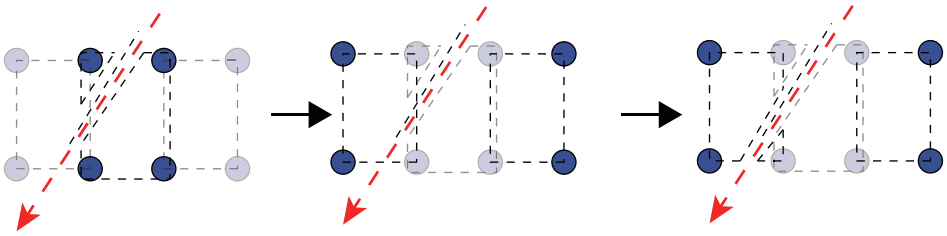


図 4.8: 近傍クラスタの破断

また，クラスタの破断を更に伝播させるために，破断計算を行ったすべてのクラスタの隣接クラスタのうち，破断面を通るものを優先破断リストに入れる．リストに加えられたクラスタのうち λ_{max} が最大であったものは，全クラスタ中で最大の λ_{max} ではなかったとしても，そのタイムステップで破断するクラスタとして選択される．

破壊の計算が終わった後は，所属する計算点が 3 個以下のクラスタを削除する． 3 個以下のクラスタは立体構造を成さず，運動が不安定になってしまうためである．また，どのクラスタにも所属しない計算点は，Shape Matching 法を使った計算ができなくなるため削除する．

図 4.9 に示すように破壊が終わった後も，隣接クラスタの影響により破壊が起きるまでの変形が残り，同じクラスタが破断し続けるというケースが起きてしまう．辻ら [5] の研究では破断後に計算点ではない頂点を使っていたことから問題にはならなかったが，提案手法では再破断を考慮するために新しい計算点を生成する必要があるため，シミュレーションを安定に実行できるようにするために対策を考える必要がある．そこで，クラスタの破断後数ステップまでは図 4.10 のように破断が起きないように設定し破断前の変形の影響を受けないようにすることで同じクラスタが破断し続ける状態を防ぐ．

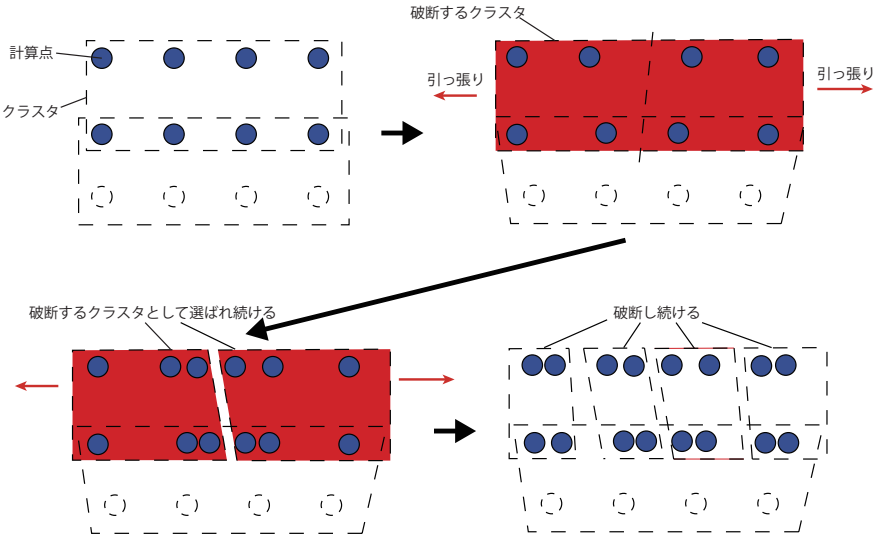


図 4.9: 同じクラスタで破断が起きたときのイメージ

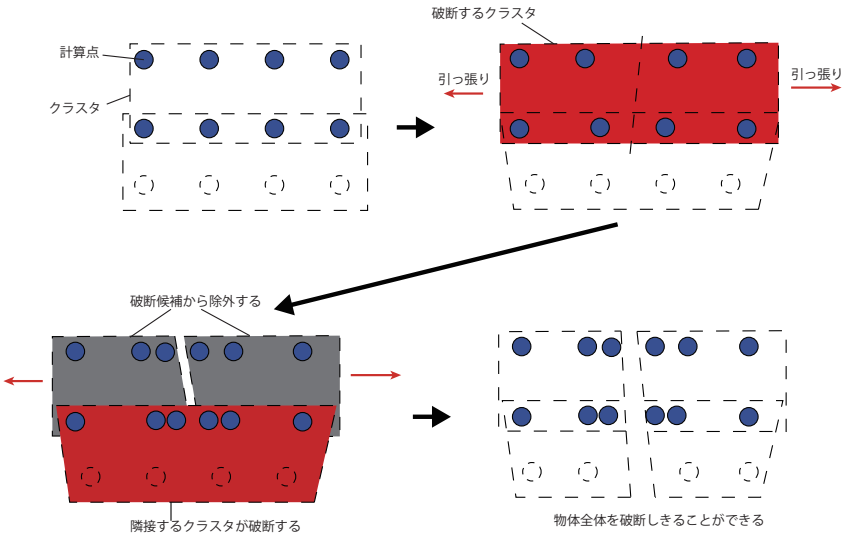


図 4.10: 同じクラスタで破断が起きないようにしたときのイメージ

第 5 章

破断面表現手法とその改良

第 2 章では Clustered Shape Matching 法を用いた破壊計算について説明したが，破壊計算で行われたのはクラスタの分離であり，視覚的な物体の分離を行うことができていない．破壊シミュレーションにおいては，破断面の生成についても考えなければならない．

この章では，辻ら [5] の手法をベースに，再破断まで考慮した破断面の生成方法について説明する．

5.1 四面体の分離による破断面生成

4.2 節で四面体分割によるクラスタ生成，4.3 節で破断面によるクラスタ分離を説明した．クラスタの分離は破壊の計算のために必要であるが，それだけでは新しい破断面を作ることはできない．そこで，4.2 節で作成した四面体を利用し，四面体とクラスタの対応関係を考えて破断の際に四面体の分割を行い，破断面を生成する．本節では，四面体構造とその分割による破断面生成手法について述べる．

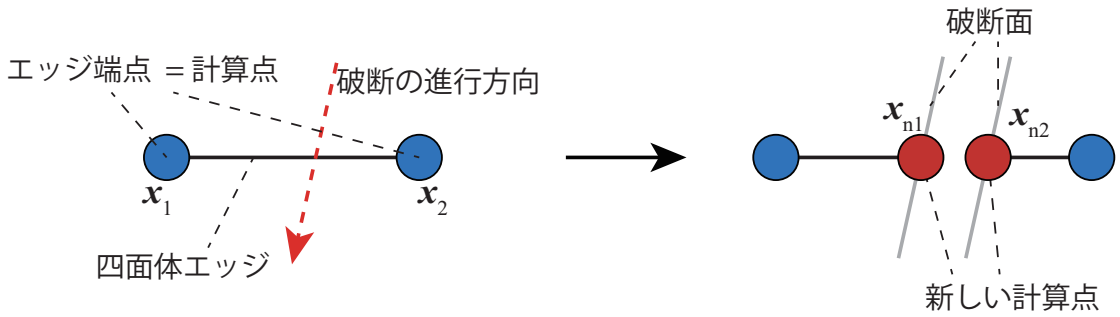


図 5.1: 新たなる計算点の生成

まずクラスタと四面体の対応関係が分かるように，クラスタ生成時にその基になった四面体をそのクラスタに所属するとして記録する．クラスタが破断によって分離したとき，クラスタに所属する四面体のエッジ中から破断方向と交差するものを探す（図 5.1 左）．エッジの両端のそれぞれの現在座標を $\boldsymbol{x}_1$，$\boldsymbol{x}_2$ とすると，エッジの交差判定式は以下のようになる．

$$((\boldsymbol{x}_1 - \boldsymbol{c}_{cm}) \cdot \boldsymbol{V}_{max})\,((\boldsymbol{x}_2 - \boldsymbol{c}_{cm}) \cdot \boldsymbol{V}_{max}) < 0.0 \tag{5.1}$$

式 (5.1) が成り立つとき，エッジと破断の進行方向との交点を求めることができる．ここで，得られた交点の座標に新たな計算点を生成する．ここで，エッジの両端の現在座標を

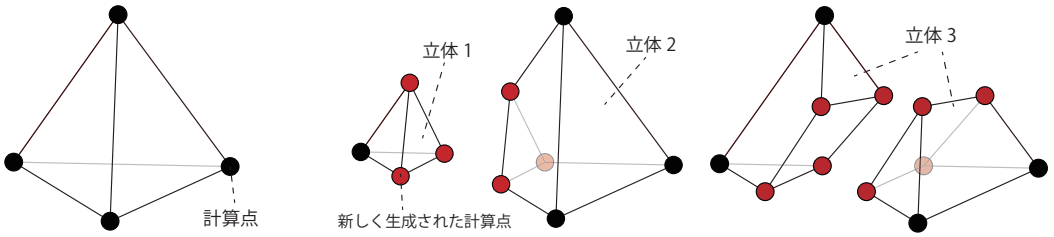
$\boldsymbol{x}_1^0$，$\boldsymbol{x}_2^0$，初期座標を $\boldsymbol{x}_1^0$，$\boldsymbol{x}_2^0$ とし，$\boldsymbol{V}_{max}$ を近傍計算点が所属しているクラスタにかかる回転行列の逆行列 $\boldsymbol{R}^{-1}$ だけ回転させてクラスタの初期状態に対応させたものを $\boldsymbol{V}_{max}^0$ と定義した上で，更に位置ベクトル $\boldsymbol{x}$ を引数としてスカラー値を返す 2 つの関数
$f(\boldsymbol{x}) = |(\boldsymbol{x} - \boldsymbol{c}_{cm}^0) \cdot \boldsymbol{V}_{max}^0|$，$g(\boldsymbol{x}) = |(\boldsymbol{x} - \boldsymbol{c}_{cm}) \cdot \boldsymbol{V}_{max}|$ を定義すると，生成された計算点の初期座標 $\boldsymbol{p}^0$，現在の座標 $\boldsymbol{p}$ は，以下の式 (5.2)，式 (5.3) で求まる．

$$\boldsymbol{p}^0 = \boldsymbol{x}_1^0 + \frac{(\boldsymbol{x}_2^0 - \boldsymbol{x}_1^0) f(\boldsymbol{x}_1^0)}{f(\boldsymbol{x}_1^0) + f(\boldsymbol{x}_2^0)} = \boldsymbol{x}_2^0 + \frac{(\boldsymbol{x}_2^0 - \boldsymbol{x}_1^0) f(\boldsymbol{x}_2^0)}{f(\boldsymbol{x}_1^0) + f(\boldsymbol{x}_2^0)} \tag{5.2}$$

$$\boldsymbol{p} = \boldsymbol{x}_1 + \frac{(\boldsymbol{x}_2 - \boldsymbol{x}_1) g(\boldsymbol{x}_1)}{g(\boldsymbol{x}_1) + g(\boldsymbol{x}_2)} = \boldsymbol{x}_2 + \frac{(\boldsymbol{x}_2 - \boldsymbol{x}_1) g(\boldsymbol{x}_2)}{g(\boldsymbol{x}_1) + g(\boldsymbol{x}_2)} \tag{5.3}$$

提案手法では，1 つのエッジが破断するごとに 2 つの計算点が生成されると定義する． 2 つの計算点の現在座標，および初期座標を $\boldsymbol{x}_{n1}$ と $\boldsymbol{x}_{n2}$，$\boldsymbol{x}_{n1}^0$ と $\boldsymbol{x}_{n2}^0$ と定義する (図 5.1 右) と，$\boldsymbol{x}_{n1} = \boldsymbol{x}_{n2} = \boldsymbol{p}$，$\boldsymbol{x}_{n1}^0 = \boldsymbol{x}_{n2}^0 = \boldsymbol{p}^0$ によって初期化される．なお，本論文では辻ら [5] と異なり，一度破断した立体の再破断を考慮するために，破断面頂点ではなく，計算点を新たに定義している．

以上の交差判定式 (5.1)，および新たに追加される計算点の位置計算式 (5.2)，式 (5.3) を各四面体の 6 本のエッジについて行った後，交差するエッジを含む四面体を無効化し，図 5.2 のように新たな立体構造を 2 つ生成する．四面体が 2 つの立体構造に分離するとき，図 5.2(b) と図 5.2(c) に示した 2 つのパターンが考えられ，このとき図 5.2(b) の立体 1 と立体 2，図 5.2(c) の立体 3，計 3 種類の立体構造に分類分けできる．



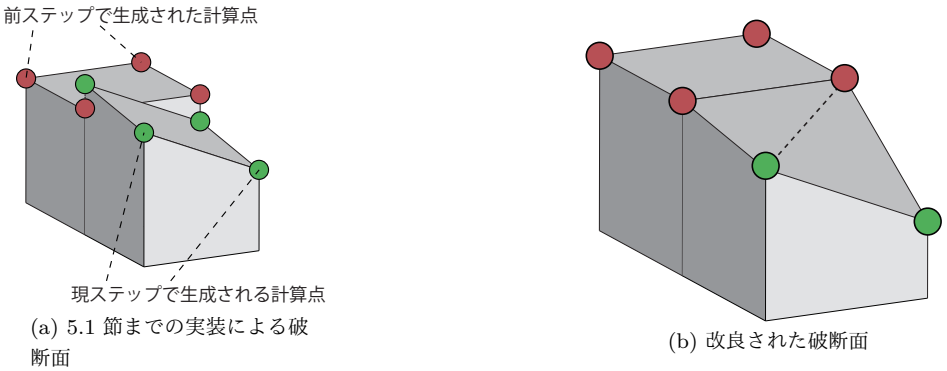
(a) 元の四面体　(b) 四面体の破断パターン 1　(c) 四面体の破断パターン 2

図 5.2: 破断によってできる新たな立体構造

5.2 破断面の連続化

5.1 節では四面体の破断の計算による新たな立体構造の生成について述べたが，四面体単位で独立して破断を考えており，かつ破断面位置が重心を通るように決めているため，破断面が一部連続しない箇所がある．実際の破壊では，破断が進む面 (crack front) に応力が集中することで，材質によって多少の凹凸は生じるものの，全体的に連続した破断面となる．そこで本節では各立体構造の面が連続するようにし，破断面の質を向上させる方法について述べる．

5.1 節までの手法を実装すると，各破断面がエッジでつながらず，図 5.3(a) のようにずれが残ってしまい，また同じエッジ上に含まれる複数の計算点がクラスタに所属してしまい計算点を不安定にさせてしまう恐れがある．そこで，既に破断が起きたエッジを含む四面体で破断が起きた場合，図 5.3(b) に示すようにエッジが破断したときに生成された計算点



(a) 5.1 節までの実装による破断面

(b) 改良された破断面

図 5.3: 破断によってできる新たな立体構造

を用いて破断面をの位置を修正し，新たな計算点は生成しない．これにより，計算点の過剰な増加によるシミュレーションの不安定化を抑え，よりリアルスティックな破断面を実現できる (処理のより詳しい手順は文献 [5] を参照)．

5.3 分離した後の破断面の構成

破断した後の破断面メッシュ生成について，辻ら [5] は破断面を四面体の分離によって表現したが，破断面の更なる破断については破断後の立体を更に破断させるとより複雑な形状となってしまうことから，更なる破断については考慮せず，破断後の立体で破断が起きたときはその立体を消すようにしている．しかし，この手法では自然現象として起きる破壊をリアリスティックに表現することができない．そこで，本論文では更なる破断を考慮した立体の構成方法およびその破断方法について提案する．

手法としては単純で，破断面を構成する立体について，更なる破断を考慮しやすくするために，図 5.2 で示した全ての立体を四面体で構成するように再分割する．図 5.2 に示した破断後に作られる立体の種類は 3 種類だが，これらの立体を四面体に分割した際の図形を図 5.4 に示す．

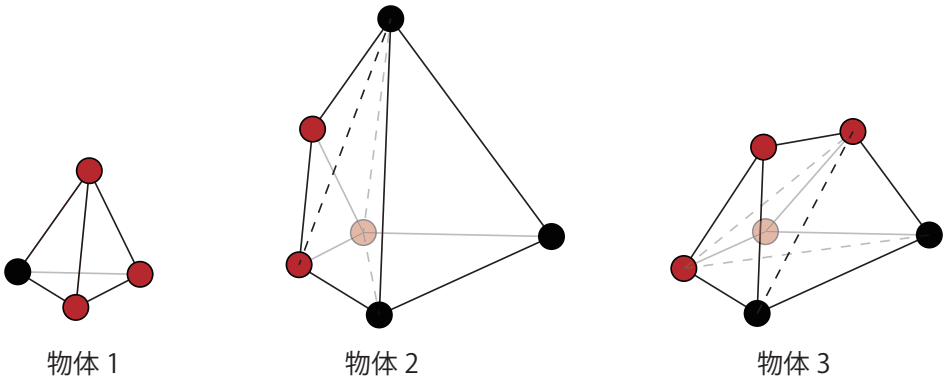


図 5.4: 新たな立体構造の四面体による構成

破断後に構成される立体を四面体によって再構成することで，5.1 節で説明した四面体を分離する手法と同じ要領で破断面の更なる破断を行うことができるようになる．

5.4　破断面生成の高速化

辻ら [5] の手法では，破断によって分離した四面体・破断によって無効となった破断面を構成する立体は図 5.5 のようにそれらのデータを削除することによって対応していた．しかし，配列に格納されたデータの一部を削除する処理は大きな計算コストがかかり，リアルタイムシミュレーションを実現する上でボトルネックとなる．

そこで，提案手法では分離した四面体および破断面を構成する立体は，図 5.6 のようにそれらのデータを削除せず，無効なデータとして参照をしないように設定し，削除までは行わないようにする．これによりデータの削除より計算コストを低く済ませることができる．ただし，必要なメモリ量が多くなるため，一定のタイミングで配列のパッキング処理を行う必要はある．また，このような方法は，6 章で述べる GPU を用いた並列計算においても，データ転送量を減らすという点で有効である．

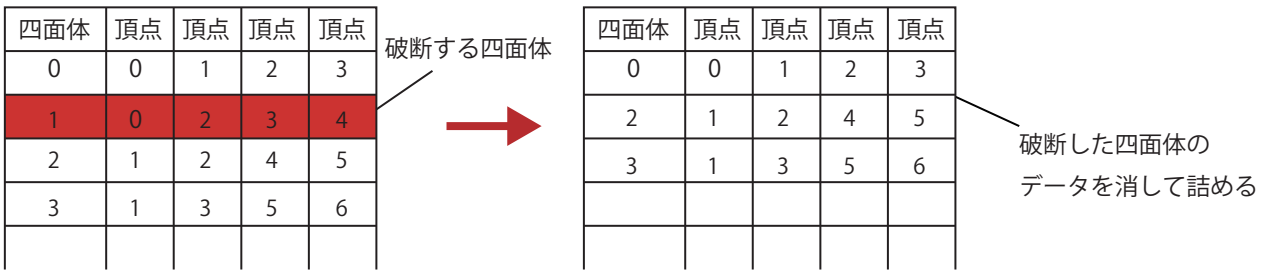


図 5.5: 四面体情報の削除

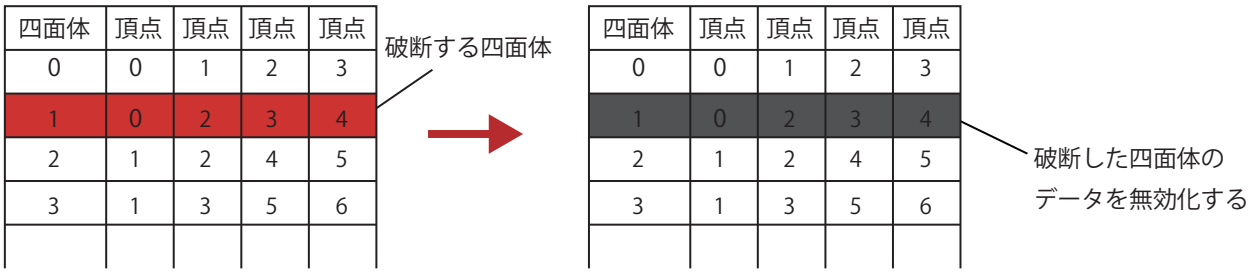


図 5.6: 四面体情報の無効化

第6章

GPUによる高速化

GPU はもともとグラフィックス処理専門のプロセッサとして開発され，多くのコンピュータに搭載されている．GPU は千を超える個数のコアが搭載されており，大量のデータの並列処理に特化している [12]．GPU は近年，一般的な計算にも使われており，そのための専用言語も多数開発されている．Clustered Shape Matching 法による変形計算も並列処理を行うことが可能であり，GPU を用いることで大幅な高速化を見込める．

本節では，GPU による Clustered Shape Matching 法による変形計算の高速化の手法，および GPU と CPU の間におけるデータの管理方法について説明する．

6.1 変形計算の高速化

GPU を用いて Clustered Shape Matching 法による変形計算を行う際には，4.2 節の図 4.5 で示したように，1 つ 1 つのクラスタ変形を順番に計算するのではなく，クラスタの変形を並列に行うことができる反復計算手法が必要となる．その一つとして，Jacobi ソルバー [13] が挙げられる．

Clustered Shape Matching 法における Jacobi ソルバーは，図 6.2 に示すように 1 つの計算点の位置について所属するクラスタにおける変形計算の結果から得られた位置を平均して求めるというものである．このとき，計算点 k の目標位置 $\boldsymbol{g}_k$ について，計算点 k が n 個のクラスタに含まれているとすると，Jacobi ソルバーを用いた位置の計算式は以下の式 (6.1) となる．

$$\boldsymbol{g}_k = \frac{1}{n}\Sigma_{m \in C_k}(\boldsymbol{R}(\boldsymbol{x}_k^0 - \boldsymbol{x}_{m,cm}^0) + \boldsymbol{x}_{m,cm}) \tag{6.1}$$

ここで，C_k は計算点 k が所属するクラスタの集合，$\boldsymbol{x}_{m,cm}^0$，$\boldsymbol{x}_{m,cm}$ は C_k 内の m 番目のクラスタのシミュレーション開始時における重心と現在のステップにおける重心である．

この計算を行った後，式 (4.16)，式 (4.17) を用いて計算点の位置と速度を更新，次の反復処理に使う $\boldsymbol{x}_k$ とする．

Jacobi ソルバーは物体内の各クラスタ内における変形を個々に独立して計算するができるため，並列処理に適した計算法である．しかし，変形の伝播を考慮していないため物体全体が柔らかくなってしまう恐れもある．この問題に対して，加速緩和係数を導入する方法 [14]，疎なクラスタ分布を用いる方法 [13] も提案されているが，提案手法では弾性体の破壊を考慮するためこの問題への対処は考えない．

6.2 ハイブリッド計算のためのデータ管理

GPU は，その設計から非常に多くのクラスタを一斉に計算することに向いているが，少数のクラスタ上での計算となる破壊計算においては並列に特化した性能を活かすことがで

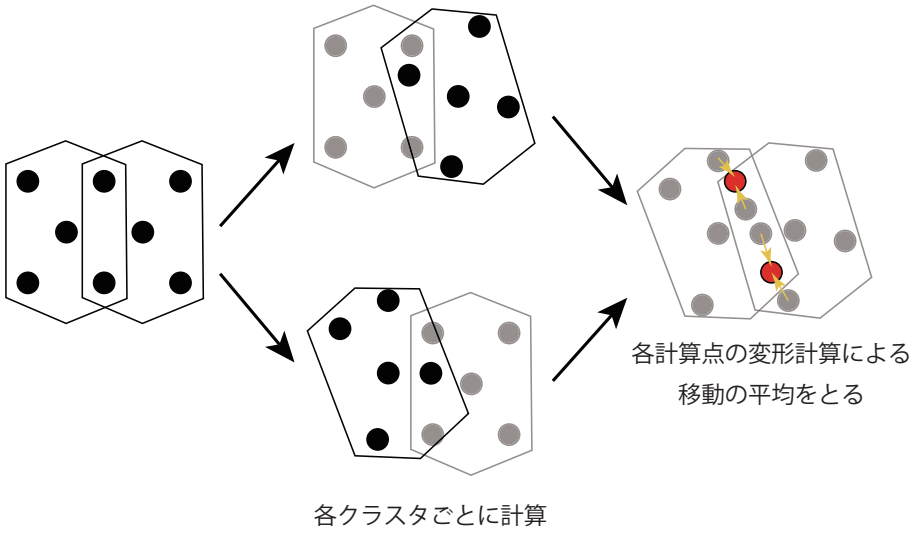


図 6.1: Jacobi 法によるクラスタの変形計算のイメージ

きず，むしろ一つ一つのコアの処理能力が CPU のコアよりも大幅に低いため低速化してしまう．実際にクラスタの破断計算を GPU 実装してみたが，CPU 実装よりも遅くなるケースが確認された．そこで，変形計算のみを GPU 側で行い，計算で得られた変形度・変形方向を CPU 側に移して破壊計算を行うハイブリッド計算を提案する．その後，破壊計算で更新されたクラスタの情報および計算点の情報をコピー・転送する必要がある．以下で詳細に説明する．

GPU 側で Shape Matching 法による計算を行うために，GPU のメモリ上に CPU 側で記録されている各計算点の位置と質量の情報，およびクラスタの情報をシミュレーション前に転送する必要がある．このデータは，4.3 節で説明したように，破断が起こったときにその構造を変化させるためにクラスタを二つに分け，更に計算点を新たに生成する必要があることから，その内容を更新する必要がある．クラスタの更新・新しい計算点の情報追加のためには破断する度にデータを転送する必要があるが，GPU-CPU 間のデータ転送は GPU 内での処理速度よりも遥かに遅く，大量のデータ転送はリアルタイムシミュレーションを実現する上で大きな障害となる．そこで，この節ではデータの転送量をできるかぎり最小限にする方法について説明する．

シミュレーション前にデータ格納のためにメモリを確保する必要があるが，変形計算のみを考慮するにあたってはその計算点数・クラスタ数の分のデータ容量を確保できれば十分である．一方で，シミュレーション中にこれらの情報が追加されることを考慮すると，そのままでは破壊が起きるたびにメモリ領域を拡張したり，内部のデータを削除する必要がある．しかし，メモリ領域を拡張したり削除すると，既存のデータを全て初期化した上で新しい計算点・クラスタを含めたデータを全て転送する必要もあり．先述の通り，転送には大きな計算コストがかかり，リアルタイムシミュレーションを実現することができない．また，このような場合一般的にはリスト構造を用いるが，メモリ上でデータが連続していないこの構造は GPU での処理に適していない．

そこで，計算点とクラスタについて図 6.2 のようにあらかじめ初期頂点・クラスタ数の数倍のメモリ領域をシミュレーション前に確保する．そして，新しく追加された計算点とクラスタは配列後方に追加していく．また，破断によってクラスタがシミュレーション中から消えてしまうケースでも，図 6.3 に示すようにメモリ上のクラスタのデータを削除する

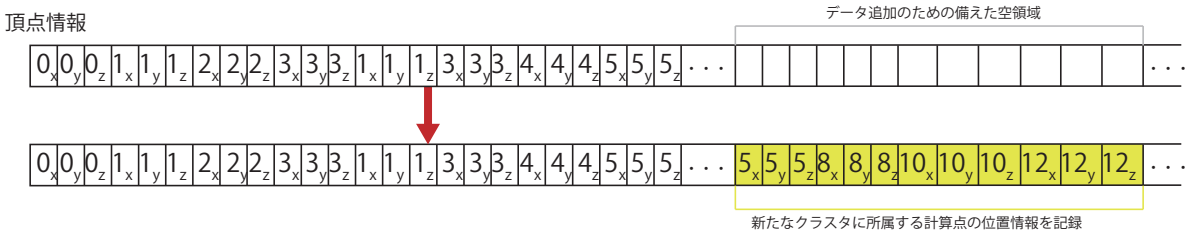


図 6.2: 新たに生成された計算点の管理

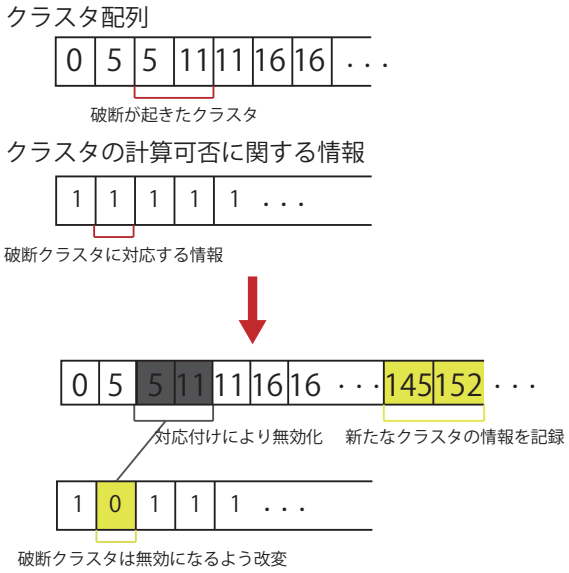


図 6.3: 追加されたクラスタのデータ管理

のではなく，データは残しクラスタを無効にするか否かを判別できるようなデータを追加する．このデータを変形計算時に参照して計算するクラスタを選択することにより，GPU 上で破断後の物体変形シミュレーションが可能となる．これにより転送するデータは追加された計算点・クラスタの分のみでよくなり，計算コストを大幅に下げてリアルタイムシミュレーションを実現できるようになる．

第 7 章

実験と考察

本章では，提案手法によって物体を破壊するシミュレーションを行った結果を示し，その後提案手法全体について考察を行う．実行環境は表 7.1 の通りである．

表 7.1: 実行環境

CPU	Intel Core i5-3470 3.20GHz
Main Memory	8GB
GPU	NVIDIA GTX770 1.163GHz，4GB
OS	Microsoft Windows 7 (64bit)
開発言語	C++,CUDA
グラフィック API	OpenGL

7.1 再破断シミュレーション

本節では，提案手法により破断面から生じる物体のさらなる破断をシミュレーションできることを証明する．実験では，どちらも物体上部を固定しながら物体底部を下方向に引っ張り，更に二つに分離した物体の下の方を右方向に引っ張ることによって破断を行うシミュレーションを行った．

7.1.1 実験 1：単一の四面体を破断した場合

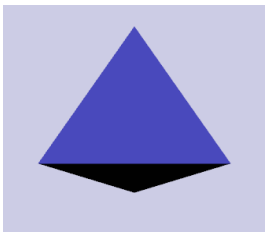


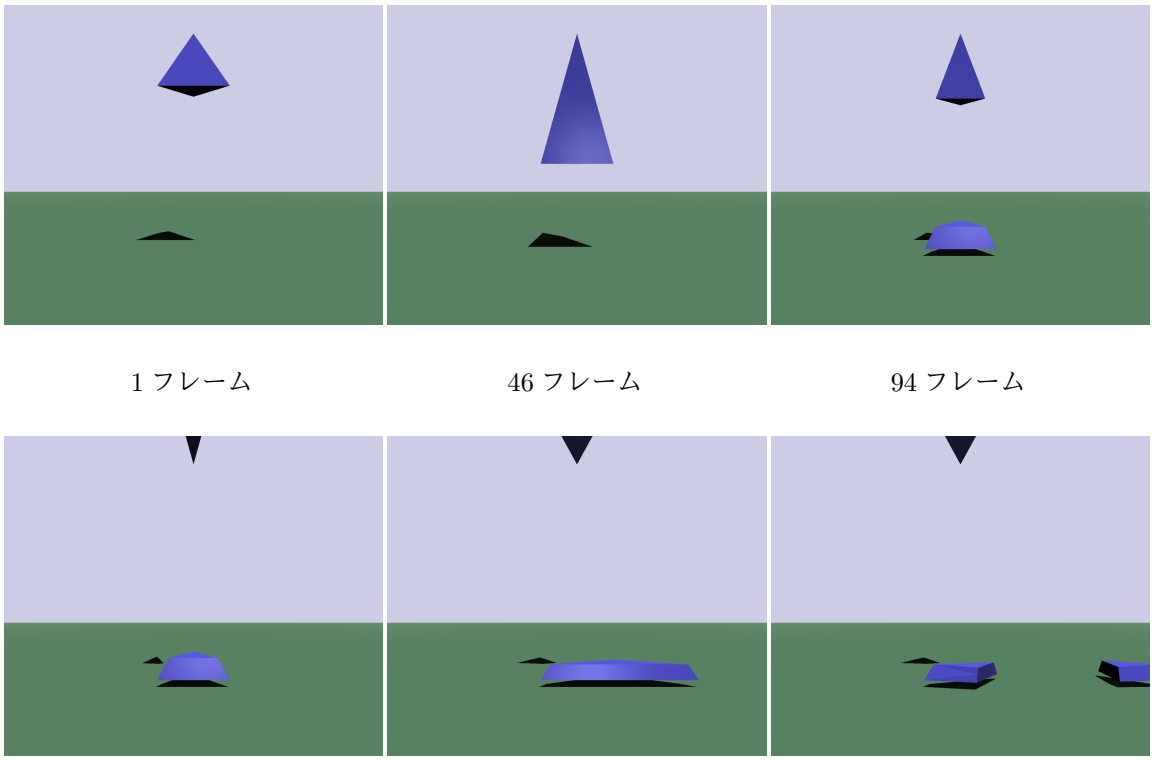
図 7.1: 四面体形状

図 7.1 に示した四面体を破断した実験結果について述べる．パラメータを 7.2 のように設定した．シミュレーションでは，この四面体は 4 個の計算点を持ち，また 4.2 節で説明した Hang ら [10] の方法による四面体分割を行わずにクラスタを構成した．すなわち，この物体は初期状態で 1 個のクラスタのみを持つ．

表 7.2: 実験 1 のデフォルト値

パラメータ名	デフォルト値
閾値	1.271939
物体の剛性度（式 (4.13) の α）	0.9
反復回数	5

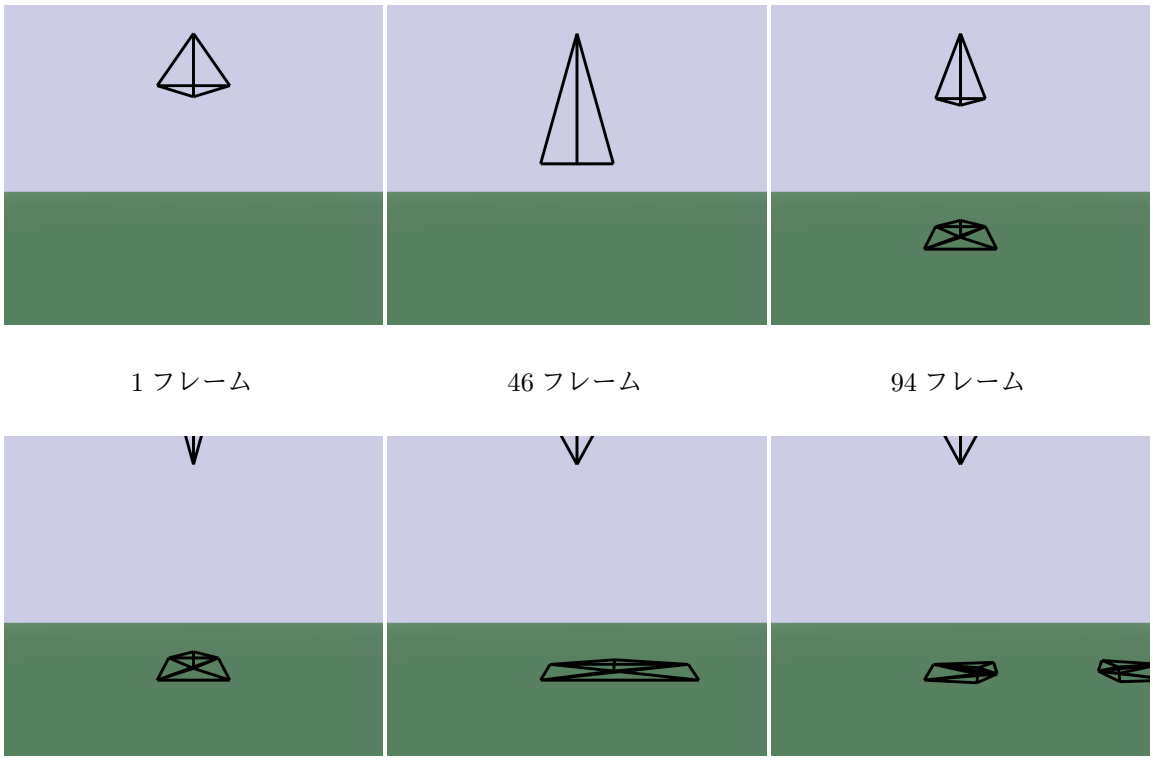
四面体の破壊シミュレーションを実行した結果を図 7.2 に示す．また，図 7.2 のワイヤーフレーム表示を図 7.3 に示す．ワイヤーフレーム表示によるシミュレーションの結果より，破断面を構成する立体がすべて四面体で構成され，なおかつ破断面を通る破断についても対応できていることが示されている．1 フレームあたりのシミュレーション時間は平均 2.7 ミリ秒、最大で 25.462 ミリ秒であった．ただし，最も計算コストが高かったタイムステップは最初のステップであった．これは，4.1 節で紹介した四面体の初期形状において対応する行列を整備する必要があったからであると考えられる．破断の計算時間は破断面の生成を含めて 1 ミリ秒未満であった．



1 フレーム　46 フレーム　94 フレーム

95 フレーム　145 フレーム　195 フレーム

図 7.2: 四面体における破断

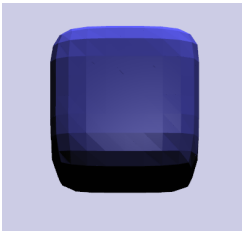


1 フレーム　46 フレーム　94 フレーム

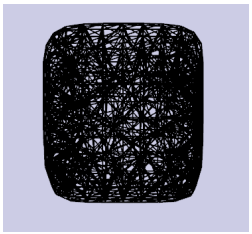
95 フレーム　145 フレーム　195 フレーム

図 7.3: 図 7.2 のワイヤフレーム表示

7.1.2　実験 2：立方体における破断の様子



(a) 立方体



(b) 立方体のエッジ構成

図 7.4: 使用する立方体

図 7.4 に示した多数の四面体によって構成された立方体を破断した実験結果を用いる．この物体は初期状態で 494 個の計算点と 494 個のクラスタのみを持つ．パラメータは表 7.3 のように設定した．

表 7.3: 実験 1 のデフォルト値

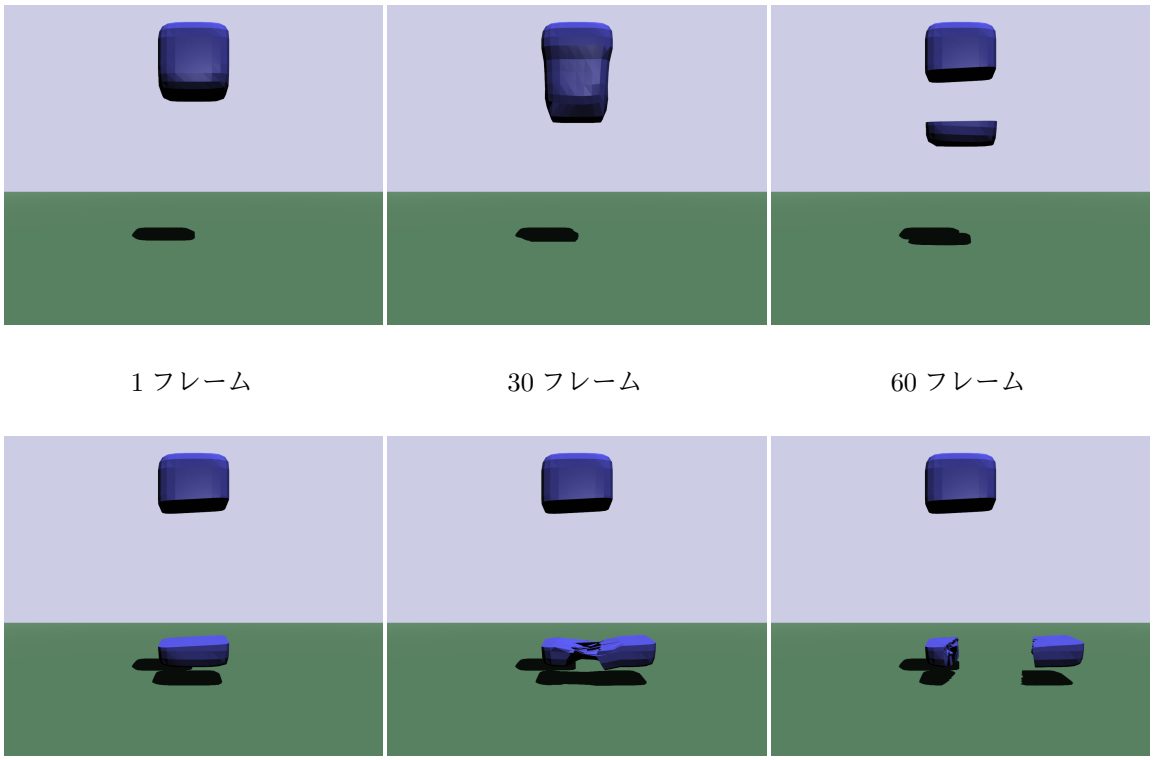
パラメータ名	デフォルト値
閾値	1.5
物体の剛性度（式 (4.13) の α）	0.9
反復回数	100

シミュレーションの実行結果を図 7.5 に示す．このとき，破断の最中で物体の一部が伸びてしまっていることが分かる（209 フレーム目）．これは，4.3 節で説明したように破断面の方向に合わせて前ステップで破断したクラスタの近隣クラスタを一つ選択し，最も変形が大きいが近隣ではないクラスタを破断するクラスタの候補から外してその選択したクラスタを破断している状態が続いて発生し，その結果破断するべきクラスタが破断せず伸びたままになっていると考えられる．また，二回目の破断時に生じた破断面の形が不自然なものとなっている．これは，1 タイムステップにつき 1 つのクラスタのみを破断するという制約により，大きく変形したが破断されないようなクラスタが現れ，その影響により破断方向が大きく変化したためと考えられる．1 フレームあたりのシミュレーション時間は平均 34.55 ミリ秒、最大で 137.729 ミリ秒であった．また，破断が起きる前までの 1 フレームあたりのシミュレーション時間は平均 26.38 ミリ秒であった．最も計算コストが高かったタイムステップは二回目の破断の最中であり，その原因としては破断するクラスタに対応付けされている破断面を構成する立体の数が多かったためであると考えられる．破断面を構成する立体の分離は CPU を用いて行っているため，その計算と立体・計算点の追加を逐次的に行うことで計算コストが大幅に増えたと考えられる．

同条件で CPU を実行させたところ，1 フレームあたりのシミュレーション時間は平均 562.184 ミリ秒，最大で 13159.5 ミリ秒となった．また，破断が起きる前までの 1 フレームあたりのシミュレーション時間は平均 67.46 ミリ秒であった，すなわち，GPU を用いた変形計算の高速化で 2.5 倍高速化されただけでなく，5.4 節，6.2 節によるデータ管理によって 16 倍もの高速化に成功した．

更に，辻ら [5] の手法を用いて同条件でシミュレーションしたところ 1 フレームあたりのシミュレーション時間は平均 101.85 ミリ秒，最大で 147.773 ミリ秒であった．この検証か

ら，本論文の手法は辻らに比べて複雑な手法であるが最大でも僅かではあるが高速で，平均では 2.95 倍高速なシミュレーションを行うことができるということを確認できた．



1 フレーム　30 フレーム　60 フレーム

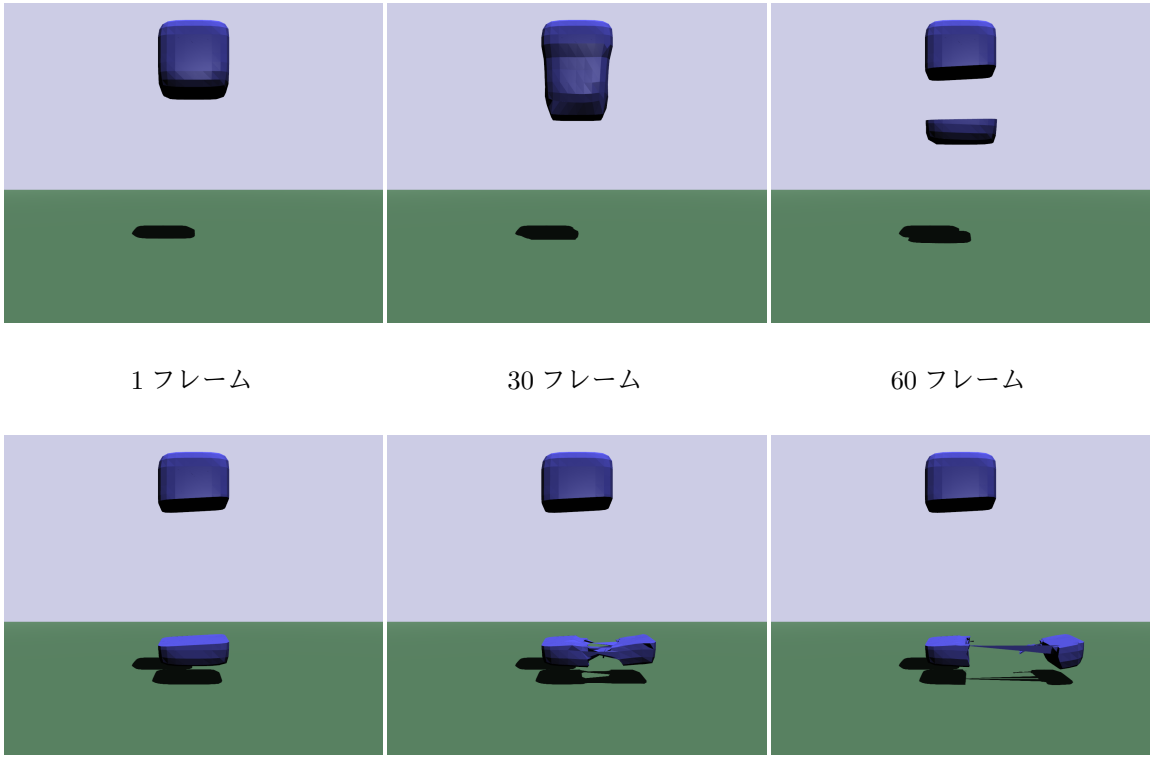
184 フレーム　209 フレーム　234 フレーム

図 7.5: 立方体の破断

7.1.3 実験失敗例：立方体における破断を別の箇所からの引っ張りによって起こさせた場合

提案手法を用いた際に，破断が失敗した例を取り上げる．シミュレーションに使う物体とそのパラメータは 7.1.2 節に準ずるが，引っ張る箇所を変更することでシミュレーション結果を変更している．シミュレーションの実行結果を図 7.6 に示す．

一回目の破断では 7.1.2 節と同じく平らな破断面が生成されたが，二回目の破断においては，未だに完全に破断が終わっておらず，メッシュで繋がったままであることが分かる．具体的には，既に二つに分離している物体の間を結ぶクラスタが残っているということである．この現象は，プログラム上での不具合であり，提案手法そのものの欠陥ではない．



1 フレーム　30 フレーム　60 フレーム

184 フレーム　209 フレーム　234 フレーム

図 7.6: 立方体の破断における不具合

7.2 考察

提案手法では，1 フレームあたりに分離するクラスタの数を 1 つに限定し，分離したクラスタは一定時間破断しないようにした．また，前ステップで破断したクラスタが存在した場合，そのクラスタの近傍クラスタを優先して破断させるように設定した．そのため，図 7.5 に示したように破断途中で変形が大きいクラスタの破断を後回しにされ続け，結果不自然なまでに延びてしまうという現象が見られた．この現象を回避するための手法として，辻ら [5] が提案した非常に大きな変形をしたクラスタは必ず破断を行うという方法が挙げられるが，1 ステップあたりに GPU に転送するデータの量が多くなり，計算コストがさらに高くなる恐れがある．更に，7.1.2 節より，破断面を構成する立体を破断しつづけることによって新しい計算点が密に生成されシミュレーションが不安定になってしまうという問題も明らかになった．この問題を解決するための手法として，密集した計算点を一つの計算点に統合するという手法が挙げられる．

第8章

結論

本論文では，Clustered Shape Matching 法による破壊計算結果に対して，物体を四面体分割してできた構造を元に破断面を生成し，破断面から更に破断が進展した場合についても考慮した手法を提案した．物体を構成する四面体の一部が破断した場合，破断の元となった位置と破断の方向から破断面とエッジの交点を求めるという従来の手法に加え，更なる破断に対応するために交点に新しく計算点を生成した．また，四面体が破断した後の立体は四面体構造で表すようにし，従来の四面体の構造を考慮した破断をと同様の方法が使える手法を提案した．更に，CPU と GPU のハイブリッド演算により，シミュレーション全体の速度を大幅に上げ，リアルタイムシミュレーションが実現できるようになった．

今後の研究として，7.2 節の考察で述べた破断面を構成する立体を破断しつづけることによって計算点が密な部分が生じてシミュレーションが不安定になってしまうという問題の解決が挙げられる．この問題を解決するための手法として，密集した計算点を一つの計算点に統合するという手法が挙げられる．この場合，GPU 上のクラスタ情報を更新する必要があるため，それを低コストで行うことができる手法について考えていきたい．

最後に，今回は計算点とクラスタの数に比べて余裕をもってメモリ領域を確保して，それらの情報を記録し，破断したクラスタや新しく生成された計算点も後方から追加するという形式でデータ転送量を少なくしていた．しかし，クラスタ数が多い物体をシミュレーションする場合非常に大きなメモリ領域が必要となり，一般的な PC で実行することが難しくなる．そのため，必要な時に必要な分だけ追加・削除し，なおかつ顕著な低速化が起きない手法を提案することも課題の一つである．

謝辞

本論文を締めくくるにあたり，様々なご指導と助言をいただきました藤澤誠助教に心から感謝の意を表します。また，合同ゼミにおいて数多くの助言をいただきました三河正彦准教授に深く感謝いたします．最後に，研究活動において数多くのご協力をいただきました物理ベースコンピュータグラフィックス研究室の方々，ソーシャルロボット研究室の方々，そして各研究室出身の先輩方に深く感謝いたします．

参考文献

[1] James F. O'Brien, Jessica K. Hodgins, Graphical modeling and animation of brittle fracture, In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 137–146, 1999.

[2] Matthias Müller, Bruno Heidelberger, Matthias Teschner, Markus Gross, Meshless deformations based on shape matching, *ACM Transactions on Graphics (TOG)*, Vol. 24, No. 3, pp. 471–478, 2005.

[3] Oleksiy Busaryev, Tamal K.Dey, Huamin Wang, Adaptive fracture simulation of multi-layered thin plates, *ACM Transactions on Graphics (TOG)*, Vol. 32, No.4, Article No. 52, 2013.

[4] Ben Jones, April Martin, Joshua A. Levine, Tamar Shinar, Adam W. Bargteily, Ductile fracture for shape matching, In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D' 16)*, pp. 65–70, 2016.

[5] 辻 和徳，藤澤誠，三河正彦, Clustered Shape Matching 法における四面体分割を用いた破断面生成に関する研究，Visual Computing / グラフィクスと CAD 合同シンポジウム 2017, 2017.

[6] David Hahn, Chris Wojtan，Fast approximations for boundary element based brittle fracture simulation, *ACM Transactions on Graphics (TOG)*，Vol. 35, No. 3, Article No.104, 2016.

[7] Alec R. Rivers, Doug L. James, FastLSM: fast lattice shape matching for robust real-time deformation, *ACM Transactions on Graphics (TOG)*, Vol. 26, No. 3, pp. 82:1–82:6, 2007.

[8] Makoto Ohta, Yoshihiro Kanamori, Tomoyuki Nishita, Deformation and fracturing using adaptive shape matching with stiffness adjustment, In *Proceedings of Computer Animation and Social Agents 2009*, Vol. 20, No. 2, pp. 365–373, 2009.

[9] Matthias Müller, Jan Bender, Nuttapong Chentanez, Miles Macklin, A robust method to extract the rotational part of deformations, In *Proceedings of the 9th International Conference on Motion in Games*, pp. 55–60, 2016.

[10] Hang Si, TetGen：A Delaunay-based quality tetrahedral mesh generator, *ACM Transactions on Mathematical Software (TOMS)*, Vol. 41, No. 2, pp. 11:1–11:36, 2015.

[11] Takashi Ijiri，Kenshi Takayama，Hideo Yokota，Takeo Igarashi, ProcDef: Local-to-global deformation for skeleton-free character animation，*Computer Graphics Forum*, Vol. 28, No. 7, pp. 1821–1828，2009.

[12] 乾正知， GPU 並列図形処理入門 -CUDA・OpenGL の導入と活用，技術評論社，2014.

[13] 仲宗根良，藤澤誠，三河正彦，位置ベース粒子法を用いた高速な融解シミュレーション，筑波大学修士 (情報学) 学位論文， 2016.

[14] Miles Macklin，Matthias Müller， Nuttapong Chentanez， Taeyong Kim，Unified particle physics for real-time applications， *ACM Transactions on Graphics (TOG)*, Vol. 33，No.4， Article No.153，2014.