芸術科学会論文誌 Vol. 7 No. 3 pp. 113 – 123

PC クラスタ環境のための 3 次元モデル軽量化手法の分散化

吉田 安男 *1　今野 晃市 *2　徳山 喜政 *3

*1(株) プレミアムエージェンシー　*2 岩手大学工学部　*3 東京工芸大学工学部

A Distributed Simplification Method with PC Cluster

Yasuo Yoshida*1　Kouichi Konno*2　Yoshimasa Tokuyama*3

*1Premium Agency Inc.　*2Iwate University　*3Tokyo Polytechnic University

E-mail: konno@cis.iwate-u.ac.jp

アブストラクト

3 次元モデルの軽量化手法は，肥大化する 3 次元形状の冗長性を削減し，様々なアプリケーションに適用可能な，基本的で重要な技術である．3 次元モデルの特徴を保存しながら，データを軽量化するためのアルゴリズムとして，QEM 手法があげられる．QEM 手法は，頂点を縮退した後の詳細さと軽量化速度の間で，最も良いバランスを持つアルゴリズムのひとつである．しかし，QEM 手法は，一度にひとつの稜線しか削除できない逐次的なアルゴリズムであるため，膨大な 3 次元モデルの軽量化には時間がかかる．PC クラスタは，メモリ分散型の計算環境として一般に用いられており，複数の PC を束ねることで，計算資源を拡大することができる．本論文では，QEM 手法を PC クラスタ環境で動作させるための分散化手法について提案する．PC クラスタを利用することで，安価なシステムにより 3 次元モデルの特徴を維持しながら，高速に軽量化することが可能となる．

Abstract

Simplification algorithm of 3D model is basic, important technology that can reduce the redundancy of the expanding 3D shape data, and that can apply to various applications. The QEM method is given as an algorithm to simplify data while preserving the feature of 3D model. The QEM method is one of the algorithms with the best balance between details and the simplification speed after the vertex is degenerated. However, because the QEM method is a successive algorithm that can delete only one edge at a time, it takes time for simplification of the huge 3D models. PC cluster is one of the environment that can enlarge computational resources with uniting many PCs. It is constructed easily and used generally. In this paper, it proposes the method made parallel by extending the QEM method, and using the PC cluster. With the PC cluster constructed by typical windows PCs, simplification high-speed by this method becomes possible while maintaining the feature of 3D model.

キーワード：3 次元モデル，PC クラスタ，データ軽量化，QEM 手法

Keywords：3D Model, PC Cluster, Simplification, QEM Method

1　はじめに

デジタルエンジニアリング技術の発展により，3 次元モデルは，設計，製造だけでなく企画，営業，広報などの多くの部署で活用されている．しかし，元になる 3 次元モデルのデータ量は，年を追うごとに巨大化し，モデリングや解析などのための計算量やメモリ使用量なども増加の一途をたどっている．このような背景で，3 次元モデルの軽量化手法は，肥大化する 3 次元形状の冗長性を削減し，様々なアプリケーションに適用可能な，基本的で重要な技術となっている．

3 次元モデルの特徴を保存しながら，データを軽量化するためのアルゴリズムとして，QEM 手法 [1] があげられる．QEM 手法は，頂点を縮退した後の詳細さと軽量化速度の間で，最も良いバランスを持つアルゴリズムのひとつである [2] ．しかし，QEM 手法は，一度にひとつの稜線しか削除できない逐次的なアルゴリズムであるため，膨大な 3 次元モデルの軽量化には時間がかかる．3 次元モデルを有効に活用するためには，膨大なデータ量の 3 次元モデルに対する処理をストレスなく実行することが必要であり，このようなニーズに対応するためのひとつの解決策として，PC クラスタを利用することがあげられる [3] ．

PC クラスタは，複数の PC をネットワークで結合したシステムである．膨大な処理を，各 PC へ分散化することによって，各 PC の負荷を低減し処理を高速化することが可能である．例えば Hashimoto らは，3 次元モデルを描画するために，PC クラスタを利用し，高解像度な画像を生成する手法を提案している [4] ．また，Wagner らは，巨大なモデルをレンダリングするために，PC クラスタを利用し，高解像度な画像をレンダリングするシステムを提案している [5] ．また，著者らは，3 次元モデル間の衝突判定を PC クラスタを利用して高速に実現する手法について提案している [6] ．このように PC クラスタを利用した画像処理や 3 次元形状処理に関する提案はいくつかなされているが，3 次元モデルを高速に軽量化するための手法はほとんどない．

El–Sana らは，3 次元モデルの軽量化を並列処理するための手法について提案している [7] ．しかし彼らの手法は，複数 CPU を持つ共有メモリ型の計算機で実行することを想定している．このような計算機は高価であり，CPU 数などの計算資源を自由に拡張することは難しい．

本論文では，QEM 手法を PC クラスタ環境で動作させるための分散化手法について提案する．本手法では，処理コストの高い軽量化処理を，複数台のサーバに分散することにより，処理時間を短縮することが可能となる．また本研究で用いる PC クラスタは，通常のネットワーク環境に接続された安価な Windows PC の集合である．既存の計算機環境をそのままクラスタ化することができるため，計算問題の複雑さと 3 次元モデルの規模，あるいは PC の遊休度に応じて動的に PC を結合して，クラスタ化することができる利点がある．以降では，PC クラスタを構成する PC をサーバと呼ぶ．

2 関連研究

2.1 QEM 手法

Garland と Heckbert が考案した QEM 手法 [1] は，頂点縮退アルゴリズムの一つである．QEM 手法は，軽量化を実行した後の詳細さと軽量化スピードの間で最も良いバランスを持つアルゴリズムの一つである [2] ．

QEM 手法の概要は以下のようになる．

- QEM 手法を適用したい，稜線の両端点を頂点ペアとして登録
- 登録した頂点を共有するポリゴンから 4×4 行列 Q を計算
- 行列 Q を利用して，全てのペアのコストを求め，コストの小さい順にソート
- 頂点ペア ($\mathbf{V}1$，$\mathbf{V}2$) の最適な縮退目標 $\mathbf{V}$ を計算
- 最小コストの頂点ペアを削除し，1 点に縮退した頂点で置換
- 置換された縮退頂点の周りのすべてのペアのコストを更新

QEM 手法では，上記の手順に従い頂点の縮退を繰り返し行い，削除するポリゴン数が一定の割合に達したところで，軽量化処理を終了する．

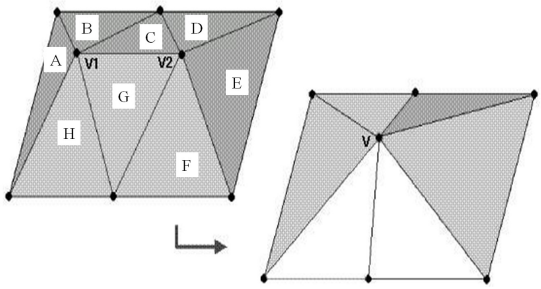

図 1　QEM 手法アルゴリズム

行列 Q の要素は，頂点の周りのすべての平面の方程式から計算する．平面の方程式を，

$$ax + by + cz + d = 0 \quad (a^2 + b^2 + c^2 = 1)$$

と定義すると，行列 Q は，式 (1) のように表される．

$$Q = \sum_{P \in planes(v)} K_p \tag{1}$$

ただし，

$$K_p = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

である．また $planes(v)$ は，頂点 v の周りのすべての平面を表す．

図 1 に示すように，注目する頂点を $\mathbf{V}1$ とすると，頂点 $\mathbf{V}1$ の周りの平面は A，B，C，G，H である．この 5 つの平面の方程式の係数から得られた値を，頂点 $\mathbf{V}1$ における行列 Q の要素とする．

また，縮退目標 $\mathbf{V}$ とは，ペア ($\mathbf{V}1$，$\mathbf{V}2$) が縮退したときに移動する頂点のことで，($\mathbf{V}1$，$\mathbf{V}2$)→$\mathbf{V}$ と表わされる．縮退目標 $\mathbf{V}$ における誤差は，誤差評価式を次のように定義して計算される．

$$\Delta(\mathbf{V}) = \mathbf{V}^T Q \mathbf{V} \tag{2}$$

ただし，

$$\mathbf{V} = [\ V_x \quad V_y \quad V_z \quad 1\ ]^T$$

である．

ここで，Q は頂点ペアの持つ行列 Q_i の和である．例えば，($\mathbf{V}1$, $\mathbf{V}2$)→$\mathbf{V}$ では，$Q = Q_1 + Q_2$ となる．誤差

評価式は $\mathbf{V}$ の二次関数であるので，$\Delta(\mathbf{V})$ を偏微分して縮退目標 $\mathbf{V}$ を求める．

また，2 頂点の間の稜線を縮退するかどうかを評価するときのコストとして $\Delta(\mathbf{V})$ の値を利用する．例えば，($\mathbf{V}1$，$\mathbf{V}2$) の誤差評価式 $\Delta(\mathbf{V})$ の値を，頂点ペア ($\mathbf{V}1$,$\mathbf{V}2$) の稜線のコスト値とする．最小コストの頂点ペアを縮退することは，形状の変化が最小で頂点縮退が可能であることを意味する．

QEM 手法を PC クラスタへ適用するとき，各サーバに割り当てられたポリゴンを分散して読み込み，各サーバが個別に QEM 手法を行う方法が考えられる．しかし，サーバで行われる軽量化の終了条件を，どのように決定するかが不明確である，といった問題がある．従来の QEM 手法では，削除するポリゴン数が一定の割合に達したところで処理を終了していた．しかし，分散化を行なった場合，各サーバに同一のポリゴン数削減割合を終了条件として与えると，各サーバから得られたポリゴンを統合後，軽量化された形状の幾何学的整合性が崩れる可能性がある．なぜなら，形状全体からすれば，あるサーバに割り当てられた領域は終了条件として与えられた削減割合よりも少ない削減数とするのが妥当であるかもしれないし，また別のサーバへ割り当てられた領域は，削減割合よりも多くのポリゴンを削減する必要があるかもしれないからである．

また，QEM 手法は，稜線の縮退を行なうとき，形状全体から 1 度に 1 つの稜線しか縮退できないという制約がある．分散化を行ったとき，あるサーバへ割り当てられた領域の中で，1 つの稜線が縮退可能であると判断されても，形状全体からすれば，その稜線が最優先で縮退されるべき稜線であるとは限らない．そのため，どのサーバが持っている領域の稜線を縮退するかを決定しながら，1 本ずつ稜線を縮退していく方法では，PC クラスタによる分散化の効果は，まったく得られない．我々は，この点に着目し，各サーバが独立して稜線を縮退できる基準を新たに設け，各サーバが持つ部分形状を個別に軽量化できるように，QEM 手法を PC クラスタ環境に適用させる．

2.2 並列形状軽量化手法

El–Sana らは，並列環境におけるポリゴンの軽量化手法を提案している [7] ．El–Sana らの手法は，共有メモリ型のマルチプロセッサマシンにおいて，視点パラメータに依存してポリゴンを軽量化する手法である．

El–Sana らの軽量化手法の概要は以下のようになる．

1. ポリゴンをオクトリに従って空間分割する．分割された部分空間をセルと呼ぶ．
2. セルに含まれる頂点数の多い順に，各セルをソートし，セルのリストを作っておく．
3. リストからセルをひとつ取り出し，セル内のポリゴンに関して稜線ヒープを構築し，セル優先度キューに追加する．この処理は, 空いている CPU に次々割り当てられる．
4. セル優先度キューから優先順位の高いセルを取り出し, 空いている CPU へ割り当てる．その CPU では，セルが持つ稜線ヒープに登録されている稜線を対象に軽量化 (稜線削除) を行なう．一定の部分空間の処理が修了したら，その親の部分空間をひとつのセルと見なして，稜線ヒープを再構築し，セル優先度キューへ追加する．このとき，子空間を表すセルは優先度キューから削除する．
5. キューを更新する．

各セルでは，セル内において，稜線の長さが短いものから，稜線削除を行なう．ただし，次の 2 つの条件に合う場合のみ，稜線の削除処理は行なわれる．

1. 削除された 2 つの頂点のどちらかに隣接する三角形において, 削除の前後の三角形の法線の違いが, ユーザに指定された閾値の範囲内である
2. 削除された 2 つの頂点のどちらかに隣接する三角形において，三角形のクォリティが，ユーザが決定した閾値を下回っていない

三角形のクォリティは，三角形の面積 a および，各辺の長さ l_0，l_1，l_2 に基づいた式 (3) で定量化する．

$$Quality = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2} \tag{3}$$

El–Sana らの手法は，複数の CPU を持つメモリ共有型の計算機で実行することを想定している．メモリを共有すると，複数の CPU は，同一のメモリ空間にアクセスすることが可能なので，通信や同期などのオーバーヘッドは少ない．しかし，このような計算機は高価であり，CPU の数などを自由に拡張することは難しい．

彼らの手法を，分散メモリ型の PC クラスタ環境で実現する場合には，共有メモリー上に保存されているセル優先度キューと軽量化対象の形状モデルを，何らかの方法で分散化する必要がある．分散メモリ型の PC クラスタでは，通信量と回数を少なくすることが，並列化効率を上げるための方策のひとつであり，その点を考慮する必要がある．軽量化は PC クラスタで実行するので，各サーバに形状モデルを持たせると仮定する．また，空いているサーバを管理するのはクライアントなので，セル優先度キューをクライアントに持たせ，空いているサーバにどのセルを処理するのかを指示すると仮定する．このようなシステムにおける問題点は次のようになる．

- セルに含まれる形状を軽量化するサーバは，クライアントが選択するので，異なるサーバが実行し

た軽量化結果を，他のサーバに反映して，形状モデルの整合性を取る必要がある．たとえば，子の部分空間を表すセルに含まれる形状を軽量化したサーバと，その子の親の部分空間を表すセルに含まれる形状を軽量化するサーバが異なる場合には，複数のサーバ間で形状モデルの整合性を取る必要がある．PC クラスタを利用するとき，形状モデルの整合性をとりながら，軽量化するのはコストがかかり，効率もよくない．

- セル優先度キューの更新は，各サーバでセルの処理が終了するごとに排他制御しながら行う必要がある．セル数が多くなると，同期や通信コストが多くなり，効率が低下する．

図 2　PC クラスタ

3　3 次元モデル軽量化手法の分散化

3.1　概要

本手法では，一般の Windows PC をギガビットネットワークで結合したメモリ分散型のクラスタシステムを利用する．図 2 に本研究で利用している PC クラスタを示す．この図では，8 台の PC をギガビットネットワークに接続している．

2.1 節で述べたように，QEM 手法は，削減したいポリゴン数あるいは，削減割合を指定して，削除することによる形状変化のもっとも少ない稜線を縮退してポリゴンを削除する手法である．縮退する稜線は，ポリゴンモデルの局所的な幾何形状に基づいて，計算されたコスト値により決定される．したがって，ポリゴンを部分形状に分割し各サーバに分散した場合，部分形状の境界以外は，各稜線のコストは分割前と変わらない．そこで各サーバは，部分形状の境界に接触しているポリゴン以外のポリゴンに属する稜線を対象として，軽量化処理を実行すれば，並列に軽量化することが可能である．各サーバでの軽量化の過程では，部分形状の境界は軽量化対象ではないため，境界稜線は変更されない．すなわち，ほかのクラスタで処理されている部分形状には影響しない． クライアントが各サーバの軽量化結果を統合化した後，部分形状の境界稜線を対象としてさらに軽量化を実行すれば，形状全体の軽量化が可能である．

部分形状に対する軽量化処理は，各サーバで独立に実行できるので，サーバ間で形状モデルの整合性を取る必要はない．また，各サーバで実行される軽量化処理では，毎回クライアントに実行結果を送信する必要はないため，通信回数も少なくてすむ．以上のような手法で軽量化することによって，2.2 節で述べた問題を解決できる．

そのためには，逐次的に縮退稜線を決定する QEM 手法を PC クラスタ環境に適用させる必要がある．そこで本手法では，各サーバが計算した稜線のコストをクライアントで集約し，削減割合を考慮した，コストの閾値を各サーバへ指示する方法を新たに導入する．最小コストでなくともコストが閾値以下の稜線であれば，削除することができることとする．これにより各サーバでは，コストの閾値に基づき，QEM 手法で稜線縮退を行なうことが可能となるため，ポリゴン数に依存せずに独立して軽量化を実行することができるようになる．軽量化の終了条件を閾値で指定することで，各サーバでは並列に QEM 手法を実行することができるため，メモリ分散型の PC クラスタ環境において，分散化手法を実現することが可能となる．また，設定した閾値までなら各サーバで別々に削除することができることから，クライアントとの通信量も削減できる．

本手法では，コストの閾値を軽量化前の形状の全コストとユーザーが指定したポリゴンの削減割合から自動的に決定し，その値を各サーバに指定する．各サーバでは，QEM 手法を適用中に，最小コスト値と与えられたコストの閾値を比較し，最小コスト値が閾値よりも大きくなったところで，軽量化を終了する．本手法は，クライアントが各サーバに分散されている稜線のコストを集約し，適切なコストの閾値を決定する．したがって，クライアントが保存できる稜線数以上の，膨大なモデルは処理が困難である，といった制約は残される．

3.2　分散処理の流れ

図 3 に，本手法による 3 次元形状の軽量化の流れを示す．図中の番号が，以下の処理の番号と対応する． ただし，3 次元形状を分散化するときは，事前に形状を領域分けしておく．領域分けの詳細は 3.3 節で述べる．

(1)　クライアントは，すべてのサーバへ，データファイルとポリゴンの領域を指定し読み込みを命令する．
(2)　サーバは指定された領域についてのデータを読み込む．
(3)　クライアントは，すべてのサーバへ，読み込んだ

領域のすべてのコストの計算を命令する．

(4) サーバは，各頂点の行列 Q 及び頂点ペアのコストを計算し，クライアントの要求により保存しているコスト値を渡す．

(5) クライアントは，ユーザが指定したポリゴン削減割合を基に，サーバから収集した全コスト値を参照して閾値を決定する．

(6) クライアントは，すべてのサーバへ，閾値を終了条件として指定し，QEM 手法を適用するように命令する．

(7) サーバは，閾値と現在の最小コスト値を比較しながら，終了条件に達するまで，繰り返し QEM 手法を適用する．

(8) クライアントは，すべてのサーバへ，QEM 手法を適用した回数として縮退した稜線数を要求する．サーバはクライアントの要求通り，稜線数をクライアントへ渡す．

(9) クライアントは，すべてのサーバへ，QEM 手法を適用した結果を要求する．サーバはクライアントの要求通り，軽量化結果をクライアントへ渡す．

(10) クライアントは，受け取った稜線数から削減されたポリゴン数を計算し，ユーザが指定した削減割合にどの程度近づいたかを算出する．

(11) クライアントは，ユーザが指定した削減割合に対して，どの程度削減すればよいのかを考慮しながら，閾値を再設定する．

(12) 手順 6 から 11 を，削減割合と全体のポリゴン数及び境界上のポリゴン数を考慮した割合になるまで繰り返す．

(13) クライアントは，すべてのサーバの結果を元に軽量化されたデータを統合化する．

(14) 統合化後に，1 台のサーバを使用し，ユーザが指定した削減割合に達するまで，従来の QEM 手法を適用する．

3.3 領域分け

3 次元形状を各サーバへどのように割り当てるのかは，重要な課題である．計算オーバーヘッドを少なくし，高速に処理するためには，各サーバにおける計算量をできるだけ均一にすることが望ましいが，あらゆる形状に対して，計算量を均一にすることは困難な課題である．

本研究では，各サーバが処理するポリゴン数が均一のときに，各サーバで削減されるポリゴン数が，極端にアンバランスにならないような，ポリゴンモデルを前提とする．そこで本手法では，各領域のポリゴン数がほぼ均一になるように，領域成長法の概念を用いた単純な方法で領域分けを行なう．具体的には，基準面をユーザーが指定し，その面に隣接する面を同一のグループとなるように追加する．そのグループに属する面数が一定数に達するまで，処理を継続する．図 4(b) は，図 4(a) のポリゴンモデルをポリゴン数が均一になるように 8 つの領域へ領域わけした例である．

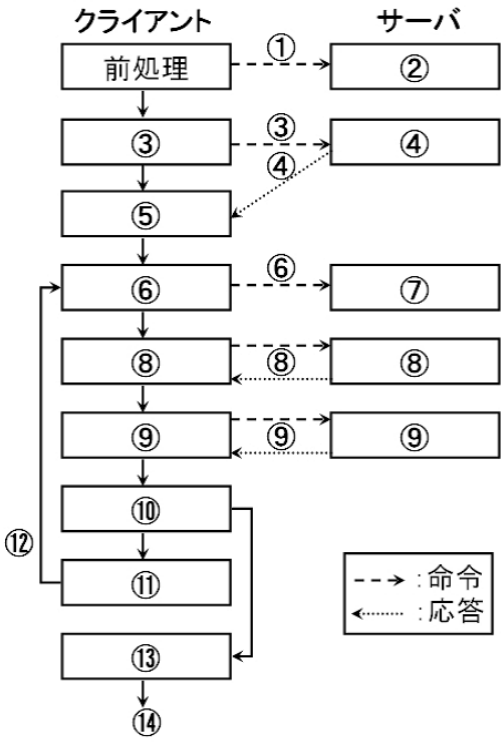

図 3　分散処理の流れ

ポリゴンモデルの一部が複雑な場合や，ポリゴンの詳細さが偏った場合には，よりよい領域分割の手法が必要である．適切な分割手法は，今後の課題である．

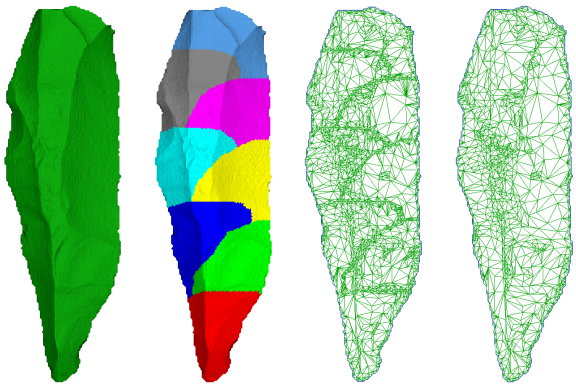

図 4　処理の様子 (石器)　(a) 元のポリゴンモデル (b) 領域分け例　(c) 1 パス目の軽量化後 (d)2 パス目の軽量化後

3.4 QEM 手法の適用

3.3 節で述べたように，3 次元形状を領域分けした後，サーバでは，個別に QEM 手法を適用する．ある閾値に達するまでサーバ個別に QEM 手法を適用することは，メモリ分散型の環境においては，クライアントとサーバ間の通信量を少なくすることができるため，重要である．通信量を削減するため，本手法では，領域分けされた部分ポリゴンの内側の稜線は，各サーバで軽量化を行い，部分ポリゴンの境界線は，クライアントが軽量化結果を統合後に，1 台のサーバで軽量化を行う．

サーバで軽量化された 3 次元モデルを，クライアントが統合化する際に，領域間の境界が一致していないと，領域間を近似的に接合する必要が生じる．領域間を近似接合するためには，各領域を接合するための共通の境界線を近似手法により算出しなければならないため，処理が複雑になる．また，軽量化手法とは別に，形状を近似してしまうため，結果の形状が乱れる可能性がある．部分ポリゴンの境界線を別途処理することで，上記の問題点も解決できる．

部分ポリゴンの境界線と接触するポリゴンは，各サーバでの軽量化結果となるポリゴンを統合化後に，1 台のサーバで再度 QEM 手法を実行する．このときには，本論文で提案する，修了条件となる閾値を与えるのではなくて，従来の QEM 手法 [1] を用いて，削減割合に達するまでポリゴンを削減する．これによって，領域の境界曲線に対しても，軽量化処理が適用されることになる．統合化の詳細は，3.6 節で述べる．

本手法では，サーバごとの軽量化を 1 パス目，統合化後に 1 台のサーバでもう一度軽量化する処理を 2 パス目とする．図 4(c) は，(b) に示す 3 次元形状を軽量化した例である．(c) を見るとわかるように，1 パス目では，領域の境界付近のポリゴンは軽量化されていない．(d) に示すように，2 パス目では，領域の境界付近のポリゴンについても軽量化されるため，全体的に 3 次元モデルが軽量化されていることがわかる．

3.5 コストの閾値の決定

本手法で導入するコストの閾値とは，QEM 手法を適用するときの終了条件である．しかし，コストの閾値は，削除可能な稜線の周りにある平面形状から算出されたコストによって決められるため，稜線のコストを計算するまでは閾値を決定することは難しい．また，形状によってコストやその閾値は全く異なるため，削減割合を考慮し試行錯誤しながら閾値を指定しないと，最適な閾値を得ることは難しい．よって，手動によりコストの閾値を決めることは，困難である．

そこで本手法では，ユーザがポリゴンをどのくらい削減したいかを指定するだけで，自動的にコストから最適な閾値を決定する．図 5 は，閾値決定の概念を示す．まず初めに，各サーバへ割り当てられた領域に含まれる稜線のコストを，各サーバに計算させ，クライアントは形状のすべてのコストを昇順にソートしたリストとして持つ．ユーザが指定したポリゴンの削減割合を $r(0 < r < 1)$ として，コストと削減割合を対応させる．一般に，削減割合 r とコストリストの id_i が線形に対応するわけではないので，あるコスト値を与えたときの実際の削減割合から，最終的な削減割合 r を達成するためのコストを推測する．具体的には，直前とその前の削減割合 R_{i-1}, R_{i-2} と，R_{i-1}, R_{i-2} を達成したときのコストを決定したコストリストのインデックス id_{i-1}, id_{i-2} から削減割合の差分 $(R_{i-1} - R_{i-2})$ とインデックスの差分 $(id_{i-1} - id_{i-2})$ が線形に変化すると仮定して，コスト値の位置 id_i を決定する．

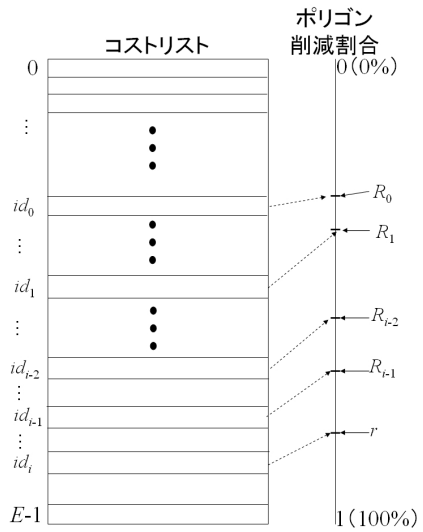

図 5　閾値決定の概念図

リストに登録されている稜線数を E としたときに，式 (4) に示す位置 id_0 にあるコスト値を，コストの閾値の初期値とする．

$$id_0 = E * r/2 \tag{4}$$

次のコスト値を決定するためのリストの位置を決定するためには，直前の 2 回分の削減割合の差分とコストリストを決定した id の比を必要とする．そのため id_1 を，式 (5) から暫定的に決定する．

$$id_1 = id_0 + E * C_1 \tag{5}$$

本研究では，経験的に定数 $C_1 = 0.002$ とする． クライアントは，サーバが $i-2$ 回目と $i-1$ 回目に軽量化した形状の削減割合 R_{i-2}, R_{i-1} と，そのとき使用したコスト値を決定するためのリストの位置 id_{i-2}, id_{i-1} より，$(R_{i-1} - R_{i-2})$ と $(id_{i-1} - id_{i-2})$ の変化が線形であると仮定し，i 回目のコスト値を決定するためのリス

トの位置 $id_i(i \geq 2)$ を，式 (6) から決定する．

$$id_i = id_{i-1} + E * \left(\frac{(R_{i-1} - R_{i-2})}{(id_{i-1} - id_{i-2})} * (r - R_{i-1}) \right) \quad (6)$$

2 パス目の処理で，境界部分の稜線が十分に軽量化されるように，ユーザが指定した削減割合よりも低い削減割合で 1 パス目の処理を終了する．具体的には，id_i におけるコスト値をサーバへ指定したときの実際の 削減割合 R_i が，式 (7) を満たすまで，式 (6) を用いて閾値を再設定し軽量化を続ける．

$$R_i \geq r - \frac{n}{E} * C_2 \quad (7)$$

ただし，n は，サーバ 1 台あたりのポリゴン数とする．2 パス目の処理でどの程度軽量化するかは，定数 C_2 の値で微調整することができる．本研究では，$C_2 = 1.0$ としている．$C_2 > 1$ とすることによって，1 パス目で削減されるポリゴン数は減少し，$C_2 < 1$ のときには，1 パス目で削減されるポリゴン数は増加する．

ユーザは従来と同様に QEM 手法の削減割合を指定し，システム側でコストの閾値に置換することで，PC クラスタで軽量化を並列処理することが可能となる．ユーザが，削減された後のモデルの面数を指定したい場合は，全体の面数と削減された後のモデルの面数から，削減割合を算出すればよい．

3.6 統合化

統合化とは，各サーバで行なった軽量化処理のすべての結果をクライアントへ集約することである．統合化に必要な情報は，各サーバの軽量化後のポリゴンデータである．本手法では，クライアントが結果形状を得る方法として，Hoppe が提案した Progressive Mesh [9] の考え方を用いる．具体的には，各サーバが，軽量化処理を適用する際に使用した，頂点ペア ($\mathbf{V}1$，$\mathbf{V}2$) の頂点番号と頂点 $\mathbf{V}1$ から縮退目標 $\mathbf{V}$ へ向かうベクトルの情報を，形状へ適用した順に並べた変形操作履歴を用いる．クライアントは，変形操作履歴のみを受け取り，それに従って変形操作を形状に適用することで，形状を統合化する．

大量のポリゴンを削除した結果，面数が少なくなれば変形操作履歴よりも結果のポリゴンデータのほうがデータ量が少なくなることも考えられる．その場合には，ポリゴンデータを直接転送してもよい．

4 実験

まず最初に，3.4 節で述べた 2 パス方式での軽量化手法による軽量化結果の妥当性について検証する．1 台で軽量化した軽量化結果と，サーバ台数を 2 台から 8 台まで変化させたときの軽量化結果をそれぞれ比較した．本手法は，サーバが 1 台の場合は，1 パス目をスキップして 2 パス目のみをそのサーバで実行する．すなわち，サーバ台数が 1 台の場合は，従来の QEM 手法による軽量化を実行するので，その結果を基準にして形状を比較する． 比較に利用したツールは，Metro [8] である．図 6 に示す馬モデルと，図 7 に示す船モデルに対して，削減割合を９０，９５，９９%に指定したときの軽量化モデルを，Metro を利用して評価した結果が表 1 である．数値は，形状の境界箱の大きさに対する最大距離の割合である．表を見るとわかるように，各削減率において，サーバ台数の変化が形状の変化に与える割合はほとんど変わらない．すなわち結果形状の品質は，領域分けにほとんど影響されないことがわかる． このことから，提案手法である 2 パスで実行する場合と従来の QEM 手法との違いは非常に少ないことが分かる．

表 1　サーバ 1 台で軽量化した結果に対する最大距離の割合

サーバ数	2	4	8
馬 (90%)	0.0403 %	0.0446 %	0.0448 %
馬 (95%)	0.0678 %	0.0819 %	0.0724 %
馬 (99%)	0.3147 %	0.3121 %	0.3217 %
船 (90%)	0.0830 %	0.0830 %	0.0561 %
船 (95%)	0.0756 %	0.0756 %	0.1117 %
船 (99%)	1.2885 %	0.3907 %	0.4668 %

次に，サーバ台数を増加させたときの，速度を評価する．実験に使用したサーバ PC は，CPU Pentium4 3.0GHz，メモリ 1GB である．まず 3.3 節で述べた方法により，形状に対して，サーバ台数分の領域分けを行なう．次に，部分領域を各サーバへ読み込み，本手法による軽量化を実施する．サーバ台数を 1 台から 8 台まで変化させたときの，軽量化時間と 1 台の実行時間との速度比を示し，PC クラスタにより軽量化速度が向上していることを確認する．ただし，ポリゴンの削減割合は 90% に設定する．

表 2 に，図 4 に示した石器のデータに対する軽量化処理時間を示す．ポリゴン数は 24,609 である．軽量化時間は，サーバが QEM 手法を適用していた時間である．結果取得時間は，クライアントがサーバから変形操作履歴を受け取るために要した時間である．統合化時間は，クライアントによる統合化の時間である．また，軽量化時間，結果取得時間及び統合化時間については，1 パス目と 2 パス目及びその合計となっている．上記 3 合計は，軽量化時間，結果取得時間及び統合化時間の合計である．速度比率はサーバ数が 1 台のときとの比較による，合計時間の倍率である．なお，サーバ台数が 1 台のとき，1 パス目の時間が「−」になっているのは，領域を分割していないため，1 パス目を実行する必要がないか

表 2　石器のデータの軽量化処理時間　(単位は秒)

サーバ数	1 台	2 台	4 台	8 台
1 パス目	——	3.204	1.798	1.046
2 パス目	7.016	0.110	0.235	0.359
軽量化時間合計	7.016	3.314	2.033	1.405
1 パス目	——	0.187	0.186	0.249
2 パス目	0.140	0.015	0.000∗	0.016
結果取得時間合計	0.140	0.202	0.186	0.265
1 パス目	——	0.171	0.172	0.172
2 パス目	0.172	0.016	0.031	0.047
統合化時間合計	0.172	0.187	0.203	0.219
上記 3 合計	7.328	3.953	2.422	1.889
速度比率	1.00	1.85	3.03	3.88

らである．また，表中の 0.000∗ は，計測時間が 1/1000 秒よりも短い時間であることを示す．

表 2 より，サーバ台数が 8 台のときは，1 台のときの約 3.9 倍に速くなっている．これは，サーバ数が増えると軽量化時間は約 5.0 倍高速化できるが，分散処理のオーバーヘッドにより，軽量化処理としては，約 3.9 倍程度になるためと考えられる．また，サーバ数が増えると，徐々に分散化効率が落ちているが，形状を表わすポリゴン数があまり多くないため，軽量化処理時間が頭打ちになってくるためであると考えられる．

図 4(c) は，領域を 8 つに分け PC クラスタに割り当てたときの，1 パス目の軽量化処理後のワイヤーフレーム表示である．同様に図 4(d) は，2 パス目の軽量化処理後のワイヤーフレーム表示である．軽量化前の形状のポリゴン数が 24,609 であるのに対し，1 パス目の軽量化後で 4,031 に，2 パス目の軽量化後で 2,459 に変化している．

図 6(a) に，ポリゴン数 96,966 の馬のデータを示す．また，図 6(b) は，8 つに領域分けをした図である．このデータについても同様の実験を行なった結果が表 3 である．サーバ台数が 8 台の時は，1 台の時の約 5.3 倍高速化することができた．軽量化時間だけに着目すると，1 台のときと比較して 8 台のときは，約 6.8 倍高速化されている．ポリゴン数が多くなると，全体の処理時間に対する結果取得時間と統合化時間の占める割合が少なくなるため，PC クラスタによる分散化効率が高まることがわかる．

図 6(c) は，図 6(a) に示した馬のデータの，軽量化処理前のワイヤーフレーム表示である．また図 6(d) は，領域を 8 つに分け PC クラスタに割り当てたときの，1 パス目の軽量化処理後のワイヤーフレーム表示である．同様に図 6(e) は，2 パス目の軽量化処理後のワイヤーフレーム表示である．軽量化前の形状のポリゴン数が 96,966 であったのに対し，1 パス目の軽量化後で 12,176 に，2 パス目の軽量化後で 9,696 に変化している．また図 6(f) は，図 6(e) に示した結果をポリゴンで表現したものである．図 6(a) と比較してみても，見た目の変化はほとんど無く，軽量化は成功している．

表 3　馬のデータの軽量化処理時間　(単位は秒)

サーバ数	1 台	2 台	4 台	8 台
1 パス目	——	11.688	5.906	3.628
2 パス目	29.859	0.469	0.844	0.781
軽量化時間合計	29.859	12.157	6.750	4.409
1 パス目	——	0.703	0.672	0.748
2 パス目	0.641	0.000∗	0.015	0.031
結果取得時間合計	0.641	0.703	0.687	0.779
1 パス目	——	0.562	0.547	0.547
2 パス目	0.563	0.047	0.079	0.078
統合化時間合計	0.563	0.609	0.626	0.625
上記 3 合計	31.063	13.469	8.063	5.813
速度比率	1.00	2.31	3.85	5.34

表 4　船のデータの軽量化処理時間　(単位は秒)

サーバ数	1 台	2 台	4 台	8 台
1 パス目	——	28.892	12.579	6.750
2 パス目	82.156	1.031	1.672	3.078
軽量化時間合計	82.156	29.923	14.251	9.828
1 パス目	——	1.311	1.328	1.312
2 パス目	1.297	0.016	0.047	0.078
結果取得時間合計	1.297	1.327	1.375	1.390
1 パス目	——	1.015	1.047	1.016
2 パス目	1.094	0.078	0.078	0.110
統合化時間合計	1.094	1.093	1.125	1.126
上記 3 合計	84.547	32.343	16.751	12.344
速度比率	1.00	2.61	5.05	6.85

図 7(a) に，ポリゴン数 194,092 の船のデータを示す．図 7(b) は，船のデータを 8 つに領域分けをした図である．このデータについても同様の実験を行なった結果が表 4 である．また，この船のデータのポリゴンを Loop 細分割 [10] を 1 回実行して，ポリゴン数を 4 倍にしたポリゴン数 776,368 の船 (4 倍) のデータについて，同様の実験を行なった結果が表 5 である．なお，船 (4 倍) のデータの領域分けについては，船のデータのポリゴンを，細分割したものであるため，船のデータと同様となっている．船のデータについては，サーバ台数が 8 台のとき，1 台のときの約 6.9 倍に高速化することができた．船 (4 倍) のデータについては，約 18.2 倍高速化することができ，特に軽量化時間だけに着目すると，サーバ台数が 1 台のときと比較して，約 22.0 倍高速化されている．サー

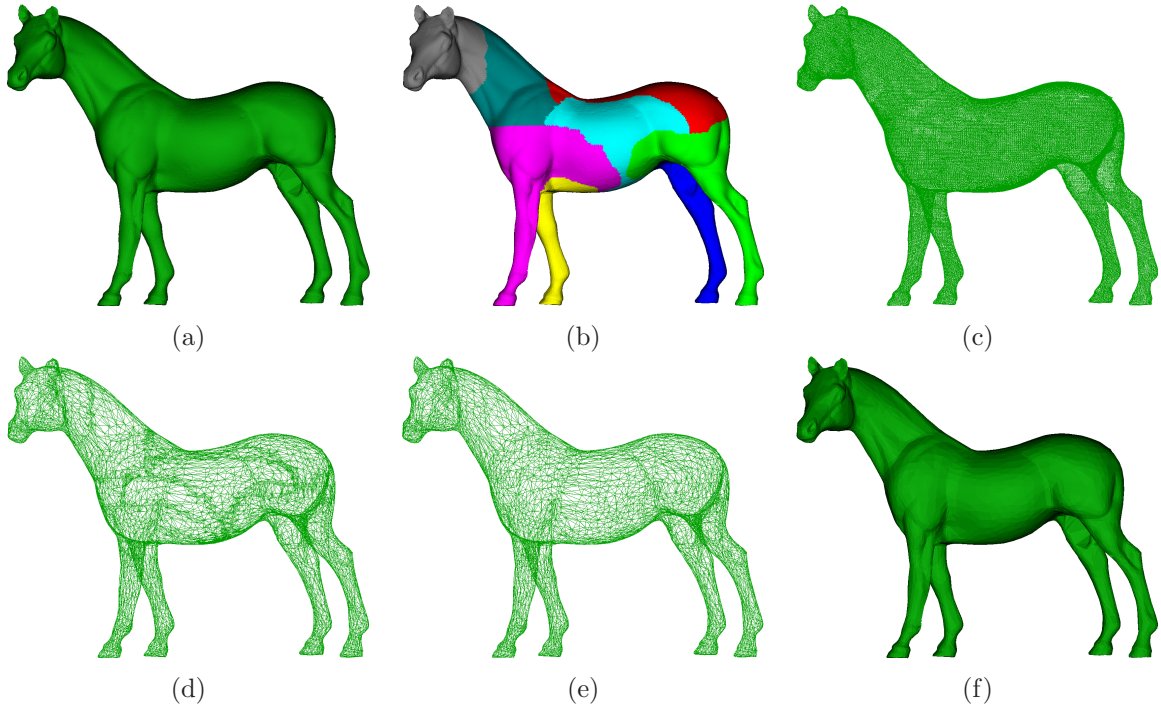

図 6　実験例 (馬)　(a) 元のポリゴンモデル　(b) 領域分け例　(c) 軽量化前の形状　(d)1 パス目の軽量化後の形状　(e)2 パス目の軽量化後の形状　(f) 軽量化後のポリゴンモデル

表 5　船 (4 倍) のデータの軽量化処理時間　(単位は秒)

サーバ数	1 台	2 台	4 台	8 台
1 パス目	——	260.281	85.533	35.813
2 パス目	978.125	3.859	5.922	8.547
軽量化時間合計	978.125	264.140	91.455	44.368
1 パス目	——	5.313	5.342	5.515
2 パス目	5.265	0.047	0.094	0.141
結果取得時間合計	5.265	5.360	5.436	5.656
1 パス目	——	4.156	4.093	4.062
2 パス目	4.250	0.094	0.125	0.172
統合化時間合計	4.250	4.250	4.218	4.234
上記 3 合計	987.640	273.750	101.109	54.258
速度比率	1.00	3.61	9.77	18.20

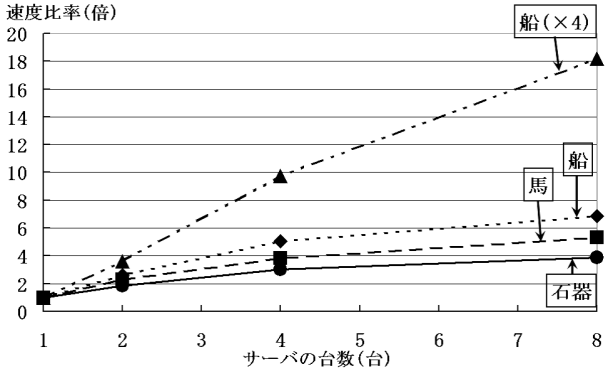

図 8　各実験データの速度比率

バ台数が 4 台から 8 台に変化するときには，速度は約 2 倍に高速化されているが，2 台から 4 台，あるいは 1 台から 2 台の場合には，2 倍以上の高速化が実現されている．これは，ポリゴン数が多い場合には，1 台のサーバで利用できるメモリ量の制限などによってメモリスワップなどのオーバーヘッドが生じるため，処理時間が長くなると考えられる．

船のデータの軽量化処理の様子について，図 7(c),(d),(e) にワイヤーフレーム表示で示す．船のデータについては，軽量化前の形状のポリゴン数が 194,092 であるのに対し，1 パス目の軽量化後で 25,280 に，2 パス目の軽量化後で 19,408 に変化している．また，船 (4 倍) のデータについては，軽量化前の形状のポリゴン数が 776,368 であるのに対し，1 パス目の軽量化後で 103,200 に，2 パス目の軽量化後で 77,636 に変化している．また図 7(f) は，図 7(e) に示した結果をポリゴンで表現したものである．この例に関しても，軽量化結果の形状は良好である．

図 8 に，石器, 馬, 船, 船 (4 倍) のデータについて，サーバ台数と速度比率に関するグラフを示す．ポリゴン数が多い形状ほど，グラフはより線形に推移し，分散化による処理効率が高まることがわかる．また，ポリゴン数が多くなったとき，サーバの台数を増やすことによって，軽量化処理の効率が高まることも予想できる．以上のことから，本軽量化手法の有効性が確認できた．

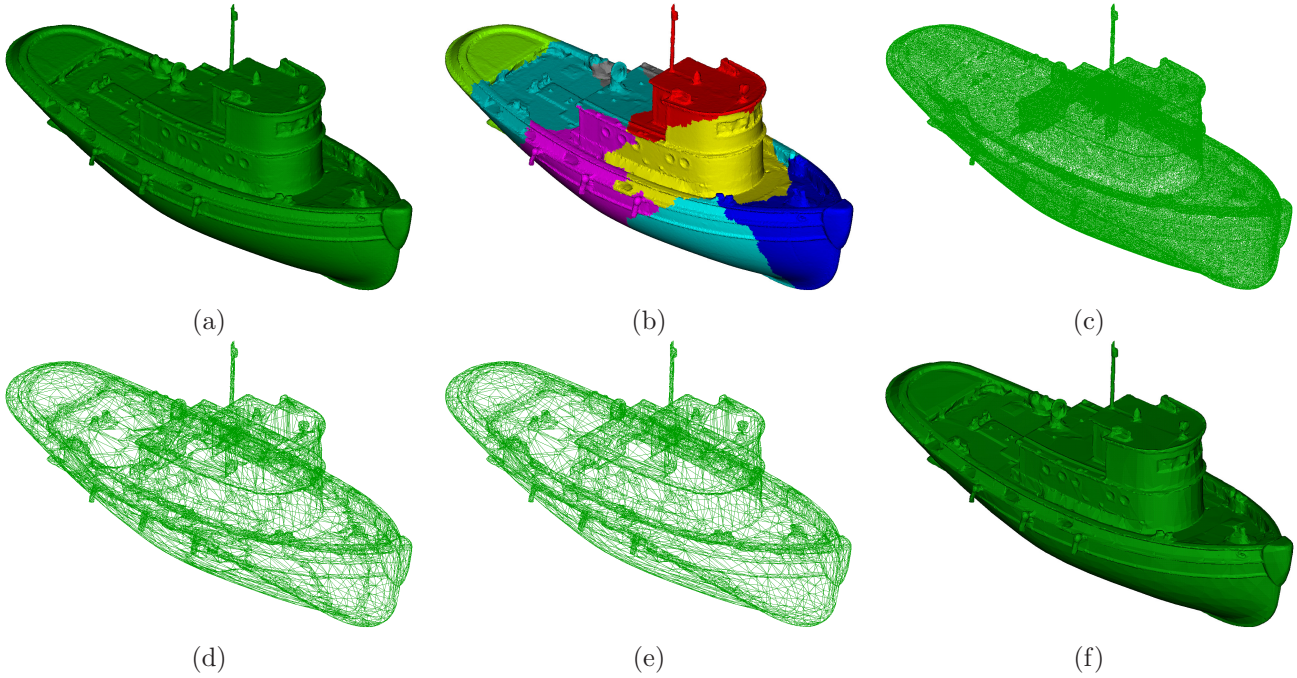

図 7　実験例 (船)　(a) 元のポリゴンモデル　(b) 領域分け例　(c) 軽量化前の形状　(d)1 パス目の軽量化後の形状　(e)2 パス目の軽量化後の形状　(f) 軽量化後のポリゴンモデル

5　まとめ

本論文では，QEM 手法を PC クラスタ環境で動作させるための分散化手法について述べた．本手法は，軽量化終了条件を与える閾値を計算し，それに基づいて各サーバが個別に軽量化できるようなアルゴリズムを導入した．また，クライアント・サーバ間の通信コストを削減するために，軽量化処理を 2 パスにし，1 パスの場合と比較して，軽量化結果に影響がほとんどないことも確認した．本手法により，1 台での処理と比較して，8 台の PC により，形状によっては約 18.2 倍の高速化を達成できた．今後の課題としては，形状に応じた適切な領域分割手法の提案と，巨大なデータによる実験を行なうことである．

参考文献

[1] M. Garland and P. S.Heckbert: “Surface Simplification Using Quadric Error Metrics”, Proc, SIGGRAPH ’ 97, pp.209-216, 1997.

[2] D. P. Luebke: “A Developer’s Survey of Polygonal Simplification Algorithms”, Computer Graphics and Applications, May/June, pp.24-35, 2001.

[3] Y. Yoshida, K. Konno, and Y. Tokuyama “A Distributed Simplification Algorithm with PC Cluster”, IWAIT 2006, pp.121-126, 2006.

[4] N. Hashimoto, Y. Ishida, and M. Sato: “A Self-Adaptive Software Environment for Cluster-Based Display System”, IWAIT 2005, pp.417-422, 2005.

[5] W. T. Correa, J. T. Klosowski, and C. T. Silva: “Out-Of-Core Sort-First Parallel Rendering for Cluster-Based Tiled Displays”, In Proceedings of PGV 2002 (4th Eurographics Workshop on Parallel Graphics and Visualization), pp.89-96, 2002.

[6] 藤原慎也, 今野晃市, 曽根順治, 徳山喜政: “階層化境界球群を用いた正確な衝突面検出法”, 画像電子学会誌, Vol.35, No.1, pp. 20-29, 2006.

[7] J. El–Sana and A. Varshney: “Parallel Construction and Navigation of View-Dependent Virtual Environments”, Proc. SPIE’99 Conference on Visual Data Exploration and Analysis, pp.23-29, 1999.

[8] P.Cignoni, C. Rocchini, and R. Scopigno : “Metro:measuring error on simplified surfaces”, Computer Graphics Forum, Vol.17, No.2, pp.167-174, 1998.

[9] H. Hoppe: “Progressive Meshes”, Proc, SIGGRAPH ’ 96, pp.99-108, 1996.

[10] Charles Teorell Loop: “Smooth Subdivision Sur-

faces Based on Triangles", Master's thesis, University of Utah, 1987.