2007－AL－111 (12)
2007／3／9

調和関数を用いたリメッシングの改良

長野真之[†] 今井桂子[††]

概要

本研究では, 3 次元表面メッシュデータに対するリメッシングアルゴリズムを提案する. 近年の形状測定技術の発達により, 3 次元スキャナで取り込まれた 3 次元表面メッシュデータは, 正確に元のモデルの形状を捉えている. そのため, それらのメッシュを様々な分野で応用することが望まれる. しかし, データ量が膨大であることや, メッシュを構成する要素がユーザの要求と異なることから, 応用分野に合わせてメッシュを加工する必要がある. そこで, Dong らが調和関数を用いたリメッシングアルゴリズムを提案した. 本研究では, 彼らのアルゴリズムを改良し, 鞍点周辺において四角形の面を生成できるアルゴリズムを提案する. また, そのアルゴリズムを実装し, 実際のリメッシングに対して有効であることを示す.

Improvement of Remeshing through the Use of Harmonic Functions

Masayuki Nagano[†] and Keiko Imai[††]

Abstract

In this paper, we consider surface remeshing in the three dimensional space and propose a new algorithm for this problem. Recently, we can generate surface meshes from images obtained by a scanner. The data of such a mesh are quite large and all elements in the mesh are not suitable for the problem that we want to solve. Therefore we have to modify the mesh to fit the application. Dong, Kircher and Garland proposed an algorithm using harmonic function for the modification. We improve their algorithm to generate well-shaped quadrangles near saddle points and show experimental results.

1. 序　　論

3 次元曲面を計算機上で表現するために, メッシュが広く用いられている. メッシュにはボリュームメッシュと表面メッシュがあり, ボリュームメッシュとは与えられた 3 次元領域を, 四面体や六面体で分割したものである. 一方, 表面メッシュは与えられた 2 次元平面上の領域や 3 次元曲面を, 三角形や四角形で分割したものをいう. 本研究では, 表面メッシュを研究の対象とする. メッシュの応用としては, 有限要素法 (FEM) を用いた物理シミュレーション, コンピュータグラフィックス (CG), CAD 等, 様々な応用分野が挙げられる.

近年の形状測定技術の発達により, 3 次元スキャナで取り込まれた 3 次元表面メッシュデータは, 正確に元のモデルの形状を捉えている. しかし, データ量が膨大であることや, メッシュを構成する要素がユーザの要求と異なることから, 多くの場合, データをそのまま使用することはできない. よって, 利用目的に合わせてメッシュを加工する必要がある. しかし, 人の手でデータを加工することはデータ量を考えると現実的ではない. そのため, 過去に多くのリメッシングアルゴリズムが研究されてきた ([2] を参照).

メッシュをコンピュータグラフィックスや有限要素法, CAD 等に使用したとき, 面の数が処理速度のボトルネックになる. そのため, 元のモデルの形状をできるだけ維持したまま, 面数を減少させることが多くの応用分野で望まれる. さらに, 四角形メッシュを用いたほうが, より少ない面数で曲面を表現できることから, データ量をより少なくできる.

以上より, 三角形を用いた表面メッシュから四角形メッシュにリメッシングするという考えが生まれる. この考えを元に, [1, 5] が主曲線を用いることで, 三角形メッシュから四角形を主としたメッシュへとリメッシ

† 中央大学大学院 理工学研究科 情報工学専攻
Information and System Engineering Course, Graduate School of Science and Engineering, CHUO University
†† 中央大学 理工学部 情報工学科
Department of Information and System Engineering, Faculty of Science and Engineering, CHUO University

ングするアルゴリズムを提案した．また，[3] では，メッシュ上に調和関数と呼ばれるユーザ定義のスカラー場を構築し，調和関数の等高線と積分曲線を用いて四角形を主としたメッシュへとリメッシングするアルゴリズムが提案された．[3] のアルゴリズムでは，調和関数にしばしば鞍点と呼ばれる頂点が生じる．鞍点の周辺における等高線と積分曲線は四角形を主としたメッシュの生成には適さない．そのため，調和関数から鞍点を除去する必要がある．そこで，本研究ではユーザ定義の調和関数を基に，鞍点の存在しない調和関数を求めるアルゴリズムを提案する．

本論文の構成は以下の通りである．まず 2 節において，調和関数を用いた [3] のアルゴリズムの概略を記す．3 節では，鞍点における [3] のアルゴリズムの振る舞いについて述べる．4 節では，spectacles を用いて調和関数を再計算するアルゴリズムについて述べ，5 節では，それを実装し提案手法の有効性を示す．最後に 6 節で結論と今後の課題について記す．

2. 調和関数を用いたリメッシング

三角形メッシュから四角形を主としたメッシュへとリメッシングするためのアルゴリズムに [3] のアルゴリズムがある．本節ではまず，[3] のアルゴリズムの概要を記す．ここでは，入力として与えられる三角形メッシュを M とし，M の頂点の集合を V，辺の集合を E，面の集合を F とする．また，V の要素はすべて 3 次元ユークリッド空間内にあり，F の要素はすべて三角形であるものとする．[3] のアルゴリズムでは，調和関数 u を用いる．調和関数とは $\Delta u = \nabla^2 u = 0$ を満たす関数 u である．[3] のアルゴリズムは以下のように 4 つのステップに分かれている．

STEP 1　調和関数 $u : V \to \mathbb{R}$ を定義する．

STEP 2　u から 2 つの直交する接ベクトル場 $\boldsymbol{g}_1, \boldsymbol{g}_2 : F \to \mathbb{R}^3$ を求める．

STEP 3　ベクトル場 $\boldsymbol{g}_1, \boldsymbol{g}_2$ をそれぞれ数値積分し，積分曲線と等高線を得る．

STEP 4　積分曲線と等高線を基にメッシュ化する．

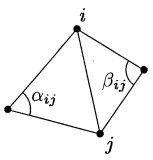

図 1　α_{ij}, β_{ij}．

まず，ステップ 1 では入力メッシュの各頂点にスカラー値を割り当て，スカラー場 u を構築する．ただし，u は次のようにして構築されたものとする．

まず，ユーザが頂点の部分集合である制約点 $C \subset V$ を選択する．そして，すべての $i \in C$ に対して $u(i) = c_i$ となる c_i を与える．次に，辺 $(i,j) \in E$ に隣接する三角形の (i,j) に接しない角を α_{ij}, β_{ij} とし (図 1)，$N_i = \{j \in V | {}^\exists (i,j) \in E\}$ としたき，

$$w_{ij} = -\frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij})$$

$$L_{ij} = \begin{cases} 1 & \text{if } i = j \\ -w_{ij} & \text{if } j \in N_i \\ 0 & \text{otherwise} \end{cases}$$

とし，次の

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$$

を $\boldsymbol{u}$ について解く．ただし，

$$\boldsymbol{u} = \begin{pmatrix} u_1 & u_2 & \cdots & u_n \end{pmatrix}^T$$

$$A_{ij} = \begin{cases} \delta_{ij} & \text{if } i \in C \\ L_{ij} & \text{otherwise} \end{cases}$$

$$b_i = \begin{cases} c_i & \text{if } i \in C \\ 0 & \text{otherwise} \end{cases}$$

とする．

そして，任意の $i \in V$ に対して $u(i) = u_i$ とする．このようにして得られた u において極値が存在するとき，極値を持つ頂点は必ず C の要素である．

次に，ステップ 2 では 2 つのベクトル場 $\boldsymbol{g}_1, \boldsymbol{g}_2$ を求める．$\boldsymbol{g}_1$ は u の勾配ベクトルとし，$\boldsymbol{g}_2$ は $\boldsymbol{g}_1$ に直交するベクトル場とする．まず，$\boldsymbol{g}_1 : F \to \mathbb{R}^3$ を求める．面 $(i,j,k) \in F$ について考える．頂点 i,j,k の座標を $\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_k \in \mathbb{R}^3$ とし，面 (i,j,k) の法線ベクトルを $\boldsymbol{n}$ とする．$\boldsymbol{g}_1$ の値は，次の等式系を解くことで求める．

$$\begin{pmatrix} \boldsymbol{x}_j - \boldsymbol{x}_i \\ \boldsymbol{x}_k - \boldsymbol{x}_j \\ \boldsymbol{n} \end{pmatrix} \begin{pmatrix} \\ \boldsymbol{g}_1 \\ \\ \end{pmatrix} = \begin{pmatrix} u_j - u_i \\ u_k - u_j \\ 0 \end{pmatrix}$$

$\boldsymbol{g}_2$ は外積を用いて $\boldsymbol{g}_2 = \boldsymbol{n} \times \boldsymbol{g}_1$ とする．$\boldsymbol{g}_1$ の例を図 2 に示す．

ステップ 3 では，$\boldsymbol{g}_1, \boldsymbol{g}_2$ から積分曲線，等高線を求める．図 3 に図 2 のベクトル場 $\boldsymbol{g}_1$ から求めた積分曲線を示す．

最後に，ステップ 4 では積分曲線，等高線から，出力となる四角形を主としたメッシュを生成する．まず，積分曲線と等高線の交点を新たなメッシュの頂点とする．次に，新たな頂点間を積分曲線または等高線が接続しているとき，新たな辺を結ぶ．そして，辺で囲まれた領域を新たな面とする．

図 2, 3 の調和関数を基にリメッシングした結果を図 4 に示す．図 4 より，[3] のアルゴリズムによる出力メッシュは，ほぼ全域において四角形の面を主としたメッシュになっていることがわかる．しかし，図 4

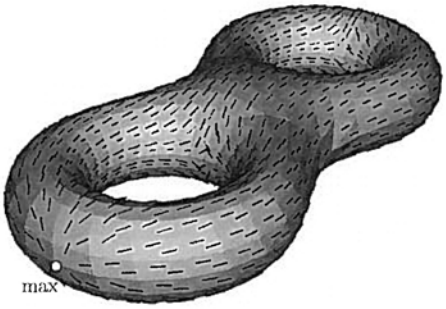

図 2 g_1.

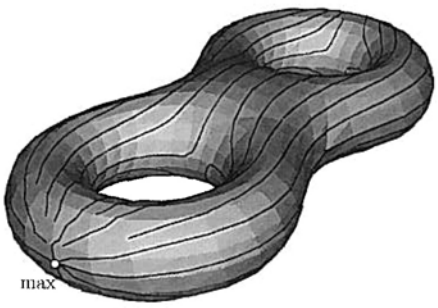

図 3 積分曲線.

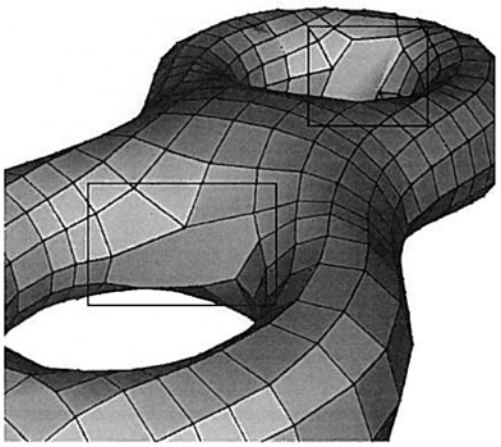

図 4 出力メッシュ.

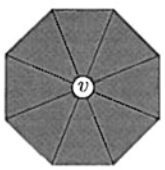

図 5 St v.

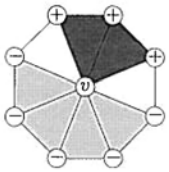

図 6 正則点.

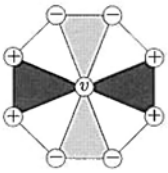

図 7 鞍点.

の黒枠部分を見ると, 一部領域においてはきれいな面が生成できていないことがわかる. [3] のアルゴリズムは鞍点と呼ばれる頂点の周辺において, きれいな四角形メッシュが生成できないという欠点がある. 次節で, 鞍点における [3] のアルゴリズムの振る舞いについて述べる.

3. 鞍点における振る舞い

[3] のアルゴリズムは鞍点と呼ばれる頂点の周辺において, きれいな面を生成できない. 本節ではまず, 鞍点について説明する. 次に, 2 節で述べた, 鞍点における [3] のアルゴリズムの振る舞いについて述べ, なぜきれいな面が生成できないかを示す.

まず, 鞍点を定義するために必要な star, upper star, lower star を定義する. 頂点 v の star $\mathrm{St}\ v$ とは, v を含む面, 辺の集合である (図 5). upper star $\overline{\mathrm{St}}\ v$, lower star $\underline{\mathrm{St}}\ v$ は以下のように定義できる.

$$\overline{\mathrm{St}}\ v = \{\sigma \in \mathrm{St}\ v | u(v) \leq u(w), w \leq \sigma\}$$
$$\underline{\mathrm{St}}\ v = \{\sigma \in \mathrm{St}\ v | u(v) \geq u(w), w \leq \sigma\}$$

ここで, $w \leq \sigma$ は, w が σ のフェイスであることを意味する. upper star, lower star, さらにメッシュ上に定義された関数 $u : V \to \mathbb{R}$ を用いると, メッシュの頂点を以下のように分類できる [4].

極大点, 極小点 頂点 v に隣接する任意の頂点 v' が, $u(v) \leq u(v')$ を満たすとき, 頂点 v を極小点という. また, $u(v) \geq u(v')$ を満たすとき極大点という.

正則点 $\overline{\mathrm{St}}\ v$, $\underline{\mathrm{St}}\ v$ がともにひとつづきのくさび形をしている場合, 頂点 v を正則点という (図 6). 図 6 において, $u(v) \geq u(v')$ となる頂点 v' は "−" で表し, $u(v) \leq u(v')$ となる頂点 v' は "+" で表す.

鞍点 $\overline{\mathrm{St}}\ v$, $\underline{\mathrm{St}}\ v$ がともに 2 組以上のくさび形の部分に分かれている場合, 頂点 v を鞍点という (図 7).

鞍点周辺における積分曲線は, 一般に図 8 のようになる. 図 9 に, [3] のアルゴリズムにより得られた鞍点周辺の等高線を示す. これを基にメッシュ化すると図 10 のように, 八角形の面が生成される. これは, 四角形を主としたメッシュを得たいユーザにとって好ましいことではない. そこで, 本研究では鞍点周辺の調和関数を再計算するためのアルゴリズムを提案する. これにより, 等高線が図 11 のようになる調和関数が得られる.

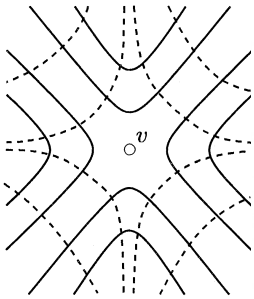

図 8 鞍点 v.

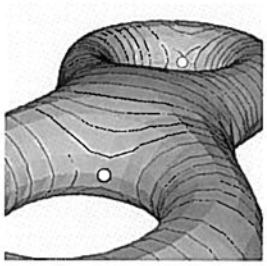

図 9 実際の等高線.

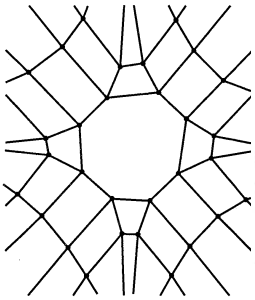

図 10 生成されるメッシュ.

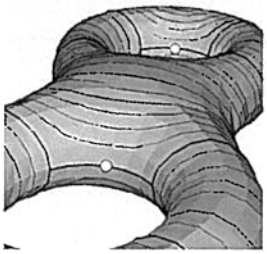

図 11 提案手法.

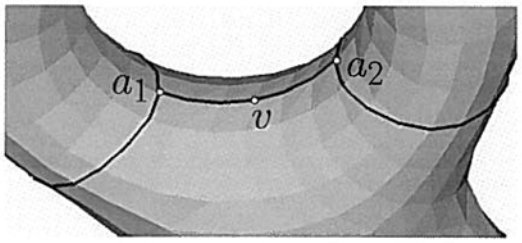

図 12 頂点 v における spectacles.

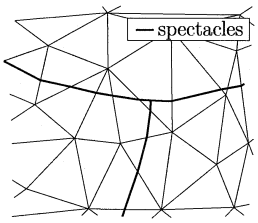

図 13 spectacles.

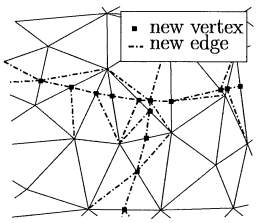

図 14 新たな頂点と辺.

4. Spectacles を用いた調和関数の再計算

本研究では, メッシュ上に定義された調和関数 u から鞍点を除去するために, 鞍点周辺の u を再計算するアルゴリズムを提案する. 提案手法の大まかな流れを以下に記す.

STEP 1 メッシュをグラフに見立て, 極小点から幅優先探索を行なう. 探索中に鞍点を見つけた場合, spectacles [6] を構成する.

STEP 2 spectacles 上の関数値が等しくなるように, 調和関数 u を再計算する.

上記のアルゴリズムによって鞍点を除去したのちの処理は [3] のアルゴリズムと同様に行なう. これ以降本節では, 提案手法の各ステップについて詳述する.

4.1 spectacles の構成

ステップ 1 では, 頂点 v が鞍点であるか否かを判定する必要がある. 頂点 v に隣接する頂点の関数値を調べることで, 頂点 v が鞍点であることは容易にわかる. 頂点 v が鞍点であった場合, $\boldsymbol{g}_1$ を基に v を始点とする 2 本の積分曲線を求める. 図 12 に鞍点 v から a_1, a_2 へと至る積分曲線を示す. この 2 本の積分曲線を bridge と呼ぶ. bridge の長さは出力メッシュの辺長の数倍とする. bridge の 2 つの終点では, それぞれ等高線を求める. この等高線を frame と呼ぶ. bridge と frame を合わせて頂点 v の spectacles と呼ぶ. spectacles は [6] において距離関数を用いて定義された概念である. 本研究では, spectacles を調和関数上で構成し, 調和関数の再計算に用いる. spectacles を用いた調和関数の再計算については次節で詳述する.

4.2 調和関数の再計算

ステップ 2 では spectacles 上の関数値を等しくする. つまり, spectacles がある値の等高線となるようにする. また, それに合わせて spectacles 周辺の頂点における調和関数値も再計算する. 後述するアルゴリズムを用いて再計算する際, spectacles はメッシュの辺上を通過していなければならない. そこで, spectacles とメッシュの辺との交点に新たな頂点を挿入する. そして, これに合わせて面を三角形に分割しなおす. 図 13 のような spectacles が計算された場合, 図 14 のように頂点と辺を挿入する. [3] のアルゴリズムでは, 調和関数を計算する際, 2 節で述べたように連立一次方程式を解いている. この手法は頂点数が増加すると非常に計算時間がかかる. そこで本研究では, 調和関数を再計算する際に以下のアルゴリズムを用いる.

spectacles s 上の頂点の集合を V_s とする. このとき, s の frame 上の頂点における関数値 u_s を用いて, 任意の $i \in V_s$ に対して $u_i = u_s$ とする. 次に, $u_i^0 = u(i)$ とおき, すべての頂点 $i \in V \setminus \{V_s \cup C\}$ に

表 1　実験環境.

OS	Windows XP (SP2)
CPU	Pentium M (2.10 GHz)
メモリ	1.00 GB RAM

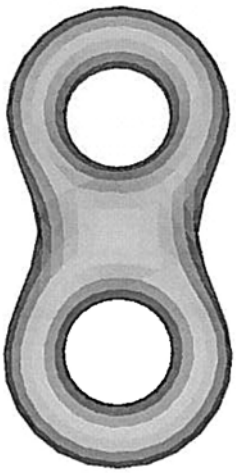

図 15　eight モデル.

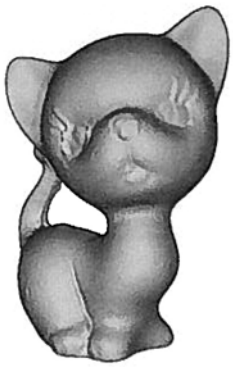

図 16　kitten モデル.

対して以下の式を反復する.

$$u_i^{k+1} = \frac{u_i^k}{2} + \frac{\sum_{j \in N_i} w_{ij} u_j^k}{2 \sum_{j \in N_i} w_{ij}}$$

u_i^k が収束したら, その値を新たな調和関数値とする. 一般に, 収束した値 u_i^k と初期値 u_i^0 の差は大きくない. そのため, この反復式は短時間で収束する.

5. 計算機実験

本節では, 提案したアルゴリズムを実装し, 計算機実験を行なった結果を示す. まず, 実験環境を表 1 に記す.

本実験では, eight モデル (頂点数 766, 面数 1536, 図 15), kitten モデル (頂点数 11039, 面数 22078, 図 16) の 2 つのメッシュを入力として用いた. まず eight モデルに対して極大点と極小点をそれぞれ 1 点ずつ配置し, [3] のアルゴリズムを用いてリメッシングした結果を図 17 に示す. 同様の入力に対して提案したアルゴリズムを用いた結果を図 18 に示す. 鞍点周辺における面を図 17 と図 18 で比較すると, 提案したアルゴリズムを用いた方がより四角形の面を生成しやすいことがわかる.

また, kitten モデルに対しても同様の実験を行なった. kitten モデルに対し, [3] のアルゴリズムを用いてリメッシングした結果を図 19 に示す. 図 19 では白

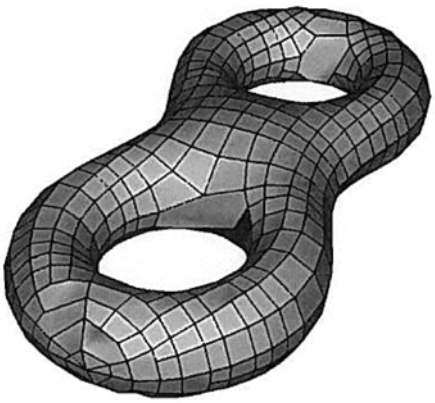

図 17　[3] のアルゴリズムによる出力メッシュ.

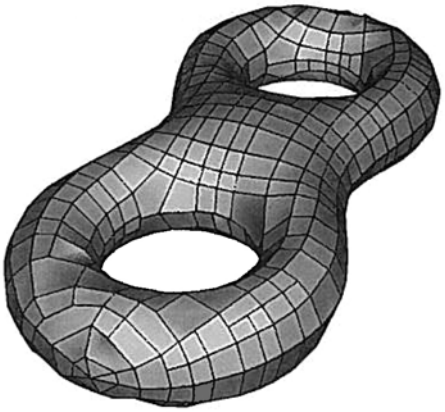

図 18　提案したアルゴリズムを用いた結果.

点の位置に鞍点があり, その付近の面を上手く生成できていない. しかし, 提案したアルゴリズムを用いた場合, 鞍点付近においても概ね四角形の面が生成できていることが分かる (図 20).

次に, 計算時間について述べる. 提案したアルゴリズムを用いて eight モデル, kitten モデルをリメッシングした際にかかった計算時間を表 2 に示す. 表 2 には,

- 調和関数を求めるために要した時間.
- 提案したアルゴリズムを用いて調和関数を再計算するために要した時間.
- 積分曲線と等高線の計算に要した時間.
- メッシュ化に要した時間.

を示す.

表 2 より, 提案したアルゴリズムに要する計算時間はリメッシング全体から見ると無視できる程度の時間であることがわかる.

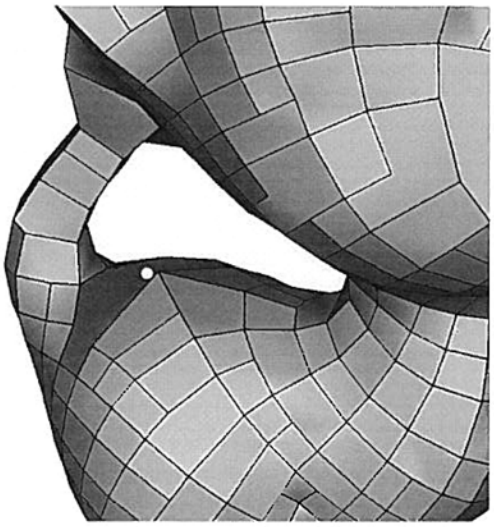

図 19 従来手法による出力メッシュ.

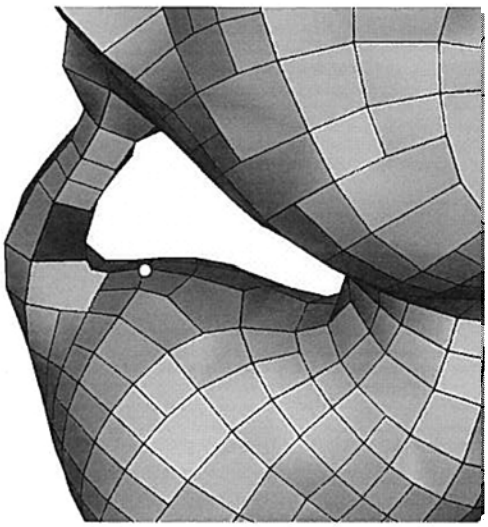

図 20 提案したアルゴリズムを用いた結果.

表 2 計算時間.

	各ステップに要した時間 [秒]				
	調和関数	再計算	積分曲線, 等高線	メッシュ化	合計
eight モデル	0.031	0.030	0.344	0.109	0.514
kitten モデル	146.125	0.531	10.579	0.156	157.391

6. 結　　論

本研究では, 三角形メッシュから四角形を主としたメッシュへとリメッシングする [3] のアルゴリズムを改良した. ユーザ定義の調和関数を再計算することによって調和関数から鞍点を取り除くアルゴリズムを提案した. また, 提案したアルゴリズムを用いた場合, ユーザの望むメッシュが得られることを計算機実験により示した.

今後の課題として, 2 つの鞍点が近い位置にある場合のように, より複雑な調和関数に対してもユーザの望むメッシュを得られるアルゴリズムを考えることが挙げられる.

謝辞　本研究の一部は 21 世紀 COE プログラム中央大学研究拠点「電子社会の信頼性向上と情報セキュリティ」と科学研究費補助金の援助を受けて行なったものである.

参 考 文 献

[1] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Levy and M. Desbrun, "Anisotropic Polygonal Remeshing," *ACM Transactions on Graphics. Special issue for SIGGRAPH conference*, pp. 485–493, 2003.

[2] P. Alliez, G. Ucelli, C. Gotsman, M. Attene, "Recent advances in remeshing of surfaces," *STAR AIM@SHAPE*, 2005.

[3] S. Dong, S. Kircher and M. Garland, "Harmonic Functions for Quadrilateral Remeshing of Arbitrary Manifolds," *Computer Aided Geometric Design*, Vol. 22, No. 5, pp. 392–423, 2005.

[4] H. Edelsbrunner, J. Harer and A. Zomorodian, "Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds," *Discrete and Computational Geometry*, Vol. 30, No. 1, pp. 87–107, 2003.

[5] M. Marinov and L. Kobbelt, "Direct Anisotropic Quad-Dominant Remeshing," In *Pacific Conference on Computer Graphics and Applications*, pp. 207–216, 2004.

[6] O. Sifri, A. Sheffer and C. Gotsman, "Geodesic-based Surface Remeshing," In *Proceedings of 12th International Meshing Roundtable*, pp. 189–199, 2003.