

PC グラフィックスハードウェアを用いたスカラ場の等値面の高速描画法

山崎俊太郎^{†a)} 加瀬 究^{††} 池内 克史^{†††}

Fast Visualization of Isosurfaces in Scalar Fields Using PC Graphics Hardware

Shuntaro YAMAZAKI^{†a)}, Kiwamu KASE^{††}, and Katsushi IKEUCHI^{†††}

あらまし 3次元空間におけるスカラ場の等値面を表示するには、三角形メッシュ化などの前処理を行うか、計算量の多い ray casting 処理が必要であり、高速表示が困難である。本論文では、PC グラフィックスハードウェアのプログラム可能な照光演算 (programmable shader) を利用して、等値面を高速に表示する手法を提案する。提案法では、曲面を定義するスカラ場関数をグラフィックスハードウェアを用いて描画時に評価し、pre-integration 法を応用した高速 ray casting により曲面を直接描画する。具体的な応用例として、曲面が単純な関数で定義される場合と複雑な関数で定義される場合のそれぞれに関して実装方法を示し、前者については blob を用いた形状モデリング、後者については距離場を用いた形状混合に適用する。また描画速度、表示画質の点から既存手法との比較を行い、提案手法の優位性を示す。

キーワード 等値面, 陰関数曲面, 非多様体, ray casting, PC グラフィックスハードウェア

1. ま え が き

3次元空間におけるスカラ場を考え、その関数値が一定であるような点の集合として曲面を定義する等値面表現は、CSG (Constructive Solid Geometry) モデリング、曲面形状の変形処理 [8] や混合処理 [16], 三角形メッシュモデルの再メッシュ化 [13], 測定点群の統合化処理 [12] や平滑化 [19] など、広い分野で利用されている。等値面表現を用いることにより、曲面の微分連続性を保持したまま形状を変化させたり位相を変更することが容易となり、複雑な形状処理や形状編集を単純な計算で頑健に行うことが可能となる。

等値面表現は形式が単純だが、一般に曲面上の点の座標を直接計算できないため曲面表示の計算コストが

大きい。したがって、高速に描画するためには、表示の画質を犠牲にするか、並列計算機などの特殊なハードウェアを利用する必要があった。これは例えば等値面表現を用いて曲面を対話的に操作する際の大きな問題であり、その適用範囲を制限してきた。

本論文では、入手が容易で安価な PC グラフィックスハードウェアのプログラム可能な照光演算 (programmable shader) を利用して、動的なスカラ場の等値面を、対話的な処理が可能な速度で描画する手法を提案する。提案手法では、pre-integration を応用した高速 ray casting を行うことにより、描画の前処理の解像度に依存しない、高精度な曲面表示が可能である。また、ハードウェアの処理速度に応じて曲面表示の詳細度を変更することで、描画速度を一定に保つ方法について提案する。更に、従来の等値面表現で扱われてきた 2- 多様体曲面のほか、曲面の境界や分岐を含む非多様体曲面にも拡張可能である。最後に、提案手法を用いて blob モデリングや形状モーフィングを可視化した結果を示し、提案法による可視化結果の有効性を実証する。

[†] (独) 産業技術総合研究所, 東京都

The National Institute of Advanced Industrial Science and Technology, Water Front 3F, 2-41-6, Aomi, Koto-ku, Tokyo, 135-0064 Japan

^{††} (独) 理化学研究所, 和光市

RIKEN, 2-1 Hirosawa, Wako-shi, 351-0198 Japan

^{†††} 東京大学, 東京都

The University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo, 153-8505 Japan

a) E-mail: shun-yamazaki@aist.go.jp

2. 関連研究

曲面 $S \subset \mathbf{R}^3$ に対して, スカラ値 $t_{iso} \in \mathbf{R}$ とスカラ値関数 $f: \mathbf{R}^3 \rightarrow \mathbf{R}$ の組で次を満たすものを考える.

$$p \in S \Leftrightarrow f(p) = t_{iso} \quad (1)$$

S は f の値が t_{iso} の点集合, すなわち f で定義されるスカラ場の等値面である. 特に $t_{iso} = 0$ のとき, f は S への距離関数と考えることができる. f が解析的な関数の場合, 式 (1) 右辺による S の定義は陰関数曲面 (implicit surfaces) 表現と呼ばれる. 一般には, 例えば f がスカラ場のサンプル点集合によって定義される場合もあり, 本論文では式 (1) の f をスカラ場関数と呼ぶ.

スカラ場の等値面を表示する手法は大きく, f から直接曲面を描画する直接法と, 表示の前に別の表現形式に変換してから表示する間接法の二つに分類でき, 本論文で提案する手法は前者に属する.

2.1 直接法

等値面をスカラ場から直接可視化する方法の一つとして ray tracing [1] がある. この方法では描画面のサイズに応じた回数光線追跡を行い, 光線と曲面の交点を計算することによって曲面を表示する. 特に光線の2次反射を考慮しない場合には ray casting [9] を用いることができる.

ray casting は, 表示画面の解像度に応じた高画質な描画が可能であるが, 形状が複雑になると光線と曲面の交差判定に時間がかかるため, 高速に描画することが難しい. そこで, 描画対象物の存在領域を限定することで計算を省略する方法や, 曲面形状の複雑さに応じてスカラ場を適応的にサンプルする方法 [18] が提案されているが, 対話的な速度で描画するには不十分である.

描画速度を向上させるために表示を簡略化する手法として, Witkin らは particle を用いて曲面全体を離散的に表示する方法を提案した [20] が, ray tracing の表示結果と比べて画質は低く, 曲面の特徴をとらえることが必ずしも容易ではない.

提案手法では, PC グラフィックスハードウェアを用いて高速に ray casting を行う.

2.2 間接法

曲面 $S \subset \mathbf{R}^3$ に対し, 関数 $g: \mathbf{R}^2 \rightarrow \mathbf{R}^3$ で次を満たすものを考える.

$$p \in S \Leftrightarrow p = g(u, v) \quad (2)$$

式 (2) 右辺による S の定義をパラメトリック曲面 (parametric surfaces) 表現と呼ぶ. ここで (u, v) は曲面上の局所座標系である. 等値面表現された曲面をパラメトリック表現に変換することにより, 描画処理を効率化できる. その中でも特に区分線形 (piecewise linear) 近似の一つである三角形メッシュ表現することで, グラフィックスハードウェアを利用した高速表示が可能となる.

Bloomenthal らは曲面を表すスカラ場の空間を四面体分割し, 曲面と交差する四面体ごとに曲面を近似する三角形の集合を生成し, 曲面全体を三角形メッシュに変換する Continuation 法を提案した [3]. また Lorensen らは医療画像などから得られるポリウム (3. 参照) に対し, 隣接ボクセルで構成される六面体を用いて三角形メッシュ化する Marching cubes 法を提案している [17].

三角形メッシュ化された曲面は高速表示が可能であるが, メッシュ生成に時間がかかるため, 変形や混合などによってスカラ場が逐次変更される場合には, 高速描画が困難である. また, 曲面形状が複雑になるほど生成される三角形の数が増え, メッシュ生成速度や描画速度が遅くなるという欠点がある.

提案手法はスカラ場関数を描画時に直接評価するため, 関数が描画時に変更されても描画速度が大きく変わらない.

3. Ray casting のハードウェア加速

ray casting はポリウムレンダリングの手法として広く用いられており, 高速化の方法が活発に研究されている. 3. では, グラフィックスハードウェアを用いたポリウムレンダリングの手法について紹介し, 4. で一般のスカラ場の等値面の表示法に拡張する.

3.1 スライスによる描画法

ポリウムはデータ点であるボクセルの3次元配列であり, ボクセルがスカラ値をもつとき, 格子点集合を定義域とする関数 $g: \mathbf{Z}^3 \rightarrow \mathbf{R}$ で表現できる. ここで $p \in \mathbf{R}^3$ に対して, p の8近傍格子点における g の値を補間するポリウム関数 $f: \mathbf{R}^3 \rightarrow \mathbf{R}$ を用いると, ポリウムを一つのスカラ場として考えることができる.

通常, ray casing によってポリウム関数 f を描画する際には, 描画視点から描画面の各画素に対し光

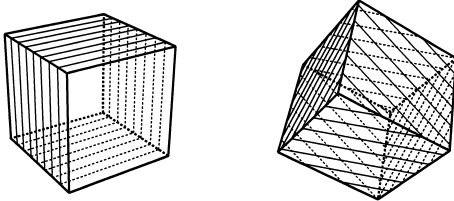


図 1 左：座標軸に垂直なスライス．右：視線に垂直なスライス
Fig. 1 The slices perpendicular to an axis (left) and the viewing direction (right).

線を飛ばし，光線上で f をサンプルして描画色を積分して表示を行う．Lacroute らは f の関数座標系における x, y, z 軸と垂直な 3 組のスライスの集合で関数をサンプルし (図 1 左)，スライスの重ね合わせによって ray casting を実現する shear-warp 因子化法 [15] を提案した．スライスを用いることによって計算機におけるデータ処理効率が上がると同時に，ポリゴンと 2D テクスチャを用いることでグラフィックスハードウェアによる描画加速が利用可能になる [7]．

座標軸に垂直なスライスを利用する場合，同一のスカラー場を表す 3 組のスライス集合を用意する必要があり，データが冗長になる．また視点移動の際にスライスの組を切り換える必要があり，その位置で描画結果が不連続になる．そこで，図 1 右のように，視点移動のたびに視線と垂直で等間隔なスライス集合を発生させ，3D テクスチャを用いてスカラー場関数の値を割り当てることにより高速で高画質な描画を実現できる [6]．

一般に，ハードウェアによる 3D テクスチャマップは，2D テクスチャマップより低速であり，利用できるハードウェアも制限される．本論文で提案する手法はどちらのスライス生成方式でも実現可能であるが，上記の利点により，実験では視線に垂直なスライスを用いた．

3.2 Pre-integration 法

スライスを使った ray casting では，スライスと垂直な方向への f のサンプルが不足することから，結果画像中に本来存在しない穴やしまが観測されるという問題がある．この問題に対し，Engel らは各スライス上の関数値と隣接スライス上の関数値を同時に利用してスライス間の関数値の影響を計算する pre-integration 法を提案した [10]．図 2 左で， S_{front} のスライスを表示する際に，点 p_{front} の位置では関数値 $f(p_{front})$ を利用して描画するが，pre-integration 法では同時に S_{back} 上の点 p_{back} の関数値 $f(p_{back})$ も利用す

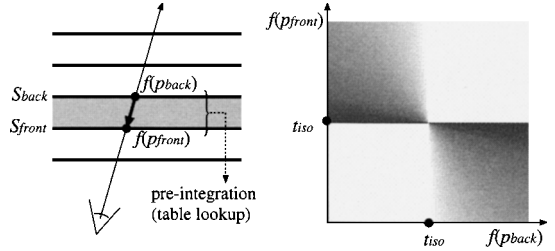


図 2 pre-integration 法によるボリュームレンダリング
Fig. 2 Pre-integrated volume rendering.

る．ここで，スライス間で関数が線形だと仮定すると $f(p_{front})$ と $f(p_{back})$ の組からスライス間の関数の値の，描画結果への寄与を計算できる．また描画の前にあらかじめこの計算を行っておく (pre-integration) ことができる．

pre-integration によるボリュームレンダリングは，密なボリュームに対する半透明表示を含む一般的な可視化の手法であるが，その特殊な場合として，ボリューム内の等値面の表示にも適用可能である．関数の等値面を表示する場合には，2 点 p_{front}, p_{back} に対して線分 $\overline{p_{front}p_{back}}$ と曲面 S の交差判定結果 $\chi = \{\text{true}, \text{false}\}$ を計算すればよい．また，交差する場合にはその交差位置 (内分比)

$$\eta = \frac{t_{iso} - f(p_{back})}{f(p_{front}) - f(p_{back})}, \quad (3)$$

及びその点での曲面の色 C も同時に計算しておく．曲面の法線 \mathbf{n} は p_{front}, p_{back} における関数のこう配を η で線形補間して次のように得る．

$$\mathbf{n} = \eta \nabla f(p_{front}) + (1 - \eta) \nabla f(p_{back}) \quad (4)$$

また，すべての $f(p_{front}), f(p_{back})$ の組合せに対して pre-integration を行い，得られた $\{\chi, \eta, C\}$ をあらかじめ表にしておくと，スライス間の関数の値に対する ray casting 計算を表の参照で高速化できる．図 2 右は式 (3) に従って $f(p) = t_{iso}$ の等値面表示を行うための pre-integration 表である．表内の色の濃さは $\eta \in [0, 1]$ に対応し，空白の領域は $\chi = \text{false}$ であることを示す．

実装する際には， C を RGB 色で表現し， $\eta \in [0, 255]$ と量子化して $\eta = 0$ のとき $\chi = \text{false}$ と考えると，pre-integration 表を RGBA 画像で表現できる．更に $f(p)$ の値を t_{iso} 付近で離散化して $f(p) \in [0, 255]$ と量子化すると，この画像を 256^2 サイズのテクスチャ

画像としてグラフィックスハードウェアに読み込み、描画時に依存テクスチャマッピングを用いて参照することで高速表示が可能になる。

4. スカラ場の等値面の表示手法

ボリュームで表されるスカラ場の等値面は、スカラ場が静的であるため曲面形状の変形が困難であり、形状処理や形状モデルの用途には適していない。等値面を使って形状操作をする場合には、スカラ場関数をパラメータによって変更することによって曲面を変形することが可能であると同時に、描画時に任意の点で関数値を評価できることが望ましい。そこで、描画時にテクスチャを動的に生成して、スカラ場関数の値を評価することにより、3. で述べた ray casting の手法でより一般的なスカラ場の等値面をハードウェア加速を利用して表示する手法を提案する。

4.1 プログラム可能ハードウェアによる実装

近年の PC グラフィックスハードウェアの大きな特色として、描画のための基本的な演算装置がプログラム可能であるという点が挙げられる。例えば、pixel shader をプログラム化することにより一般の関数をテクスチャとして利用可能になる。

ポリゴンを描画する際のハードウェア内部における具体的な処理の流れは次のとおりである：まず頂点単位の処理が行われ、ラスタ化されてフラグメント (fragment) に分解された後、pixel shader で描画色が決定され、最終的にフレームバッファに書き出される。pixel shader にはテクスチャ、定数、及びこれらの値をもとに描画色を決定するフラグメントプログラム (fragment program) を設定することができ、各フラグメントの色や透明度を描画時に決定できる。

提案法では、スカラ場関数 f を、テクスチャの集合 $\{T_f\}$ だけでなく、描画時に画素単位でスカラ場関数を評価するためのフラグメントプログラム P_f 、及び定数の集合 $\{c_f\}$ の組合せとしてグラフィックスハードウェアに読み込み、動的なスカラ場の等値面を描画時に評価する。

表示処理の大まかな流れは以下のとおりである。描画処理はソフトウェアで行う部分とグラフィックスハードウェアで行う部分に分かれる。スカラ場関数 f を表現するための $\{T_f\}$ 、 P_f 、 $\{c_f\}$ の具体的な設定方法、及びスカラ場の領域分割については次節で述べる。

ソフトウェア処理

- ① 照光に必要な光源パラメータを設定。
- ② スカラ場を領域分割し、各領域に以下の処理を行う。
- ③ スカラ場関数を、テクスチャ $\{T_f\}$ 、フラグメントプログラム P_f 、定数 $\{c_f\}$ の組合せとして設定。
- ④ pre-integration 表をテクスチャ T_t として設定。
- ⑤ 描画視点を設定。
- ⑥ 視点から等間隔で視線に垂直なポリゴンを生成。
- ⑦ p_{front} 、 p_{back} をテクスチャ座標 τ_{front} 、 τ_{back} として設定。
- ⑧ ポリゴンを描画 (ハードウェア処理へ)。
- ⑨ 形状が変化する場合は③に戻る。
- ⑩ ⑤に戻る。

ハードウェア処理

- ① ポリゴンをフラグメント化し以下の処理を行う。
- ② $\{T_f\}$ 、 P_f 、 $\{c_f\}$ 、 τ_{front} 、 τ_{back} から $f(p_{front})$ 、 $f(p_{back})$ を取得。
- ③ T_t 、 $f(p_{front})$ 、 $f(p_{back})$ から $\{\chi, \eta, C\}$ を取得。
- ④ $\chi = \text{false}$ のときはこのフラグメントを破棄。
- ⑤ $\chi = \text{true}$ のときは曲面の法線 n を式 (4) で決定。
- ⑥ 光源のパラメータ、 n 、 C から描画色を決定。
- ⑦ フラグメントをフレームバッファに書き出す。

なお、3. で述べた ray casting の高速化手法をボリュームレンダリングに用いる際には、テクスチャの α 値 (不透明度) を利用してポリゴンの α 混合表示をする必要があるため、視点から遠いポリゴンから順に描画しなければならない。これに対し等値面表示の場合には、 α 混合が不要なため、ポリゴンを視点に近い方から順に描画することで、Z バッファを使った表示の高速化が可能である。

4.2 陰関数曲面

スカラ場関数が単純な式で書ける場合には、その式と形状に関するパラメータをフラグメントプログラムと定数の組合せで読み込む。これにより、描画時に描画面の解像度に応じて関数をサンプルすることができ、高精度な描画を行うことができる。また曲面の形状に関するパラメータを描画時に変更することで曲面を対話的に変形することができる。

単純な関数式を用いた陰関数曲面の例として、blob モデル [2] がある。blob モデルでは、少数のパラメータで定義され、影響範囲が有限であるような blob 関

数 $b_i : \mathbf{R}^3 \rightarrow \mathbf{R}$ を考え、スカラ場関数 f を blob 関数の線形和

$$f(p) = \sum_{i=1}^n \lambda_i b_i(p) \quad (5)$$

で定義する．曲面の法線は次の式で決定される．

$$\nabla f(p) = \sum_{i=1}^n \lambda_i \nabla b_i(p) \quad (6)$$

曲面を可視化する場合には、式 (5) と式 (6) を用いてスカラ場関数、及びそのこう配をフラグメントプログラム P_f として与え、フラグメントのテクスチャ座標から描画時に関数値を評価する．各 blob が形状に関する可変パラメータをもつ場合には定数集合 $\{c_f\}$ で与えておき、フラグメントプログラムから参照する．描画時に blob の数や位置、形状を変更するときは P_f を更新する．

blob 関数の影響範囲は有限であることから、blob 中心から離れた空間は等値面表示に影響を与えず、このような空間に対してスライスを生成すると処理するフラグメント数が増えて描画速度が低下する．また、形状を定義する blob の数が増えるとフラグメントプログラムの評価コストが高くなり、描画速度が低下する．これらの問題を避けるために、スカラ場を各 blob の影響範囲を含む bounding box に分割し、bounding box ごとに、その領域に影響を与える blob 集合をもとに描画を行う．blob 関数 b_i が影響を与える空間を $D_i \subset \mathbf{R}^3$ とすると、この blob を描画する際には、 D_i の bounding box 内をスライス集合で表現し、次の式で表される局所的なスカラ場関数 f_i 、及びそのこう配 ∇f_i をフラグメントプログラムとして与えて描画処理を行う．

$$f_i(p) = \sum_{D_j \cap D_i \neq \emptyset} \lambda_j b_j(p), \quad \nabla f_i(p) = \sum_{D_j \cap D_i \neq \emptyset} \lambda_j \nabla b_j(p) \quad (7)$$

ここで各 blob に対する bounding box は互いに重複領域をもつことがあり、これによりスカラ場中の同一の点における関数値が複数回評価される可能性がある．bounding box の重複は、描画効率を低下させる原因となり得るが、描画画質には影響しないため、実験で用いた実装ではこの点を考慮していない．

提案手法により blob モデルを表示した結果を図 3 左に、そのときの描画に利用した bounding box、及

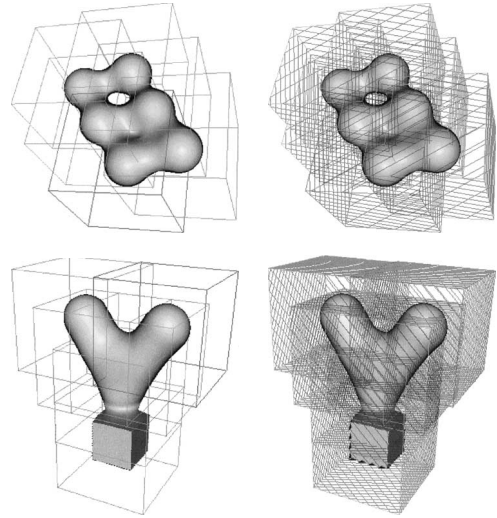


図 3 blob による形状モデリング．左：bounding box 表示．右：スライス表示

Fig.3 Shape modeling using blobby objects. Left: Rendering with bounding boxes. Right: Rendering with slices.

びスライス集合を右に示す．blob 関数は、blob の上の座標を $p = (p_x, p_y, p_z)$ 、中心位置 $c_i = (c_{ix}, c_{iy}, c_{iz})$ 、影響を与える領域の半径 r_i としたとき、球形 blob

$$b_i(p) = \max\left(0, \left(1 - \frac{|p - c_i|^2}{r_i^2}\right)^3\right) \quad (8)$$

及び、立方体 blob

$$b_i(p) = \max\left(0, \left(1 - \frac{\max(|p_x - c_{ix}|, |p_y - c_{iy}|, |p_z - c_{iz}|)^2}{r_i^2}\right)^3\right) \quad (9)$$

の 2 種類を用いた．これらの blob の影響範囲 D_i はそれぞれ、中心が c_i で半径が r_i の球、及び中心が c_i で一辺の長さが $2r_i$ の立方体である．これら 2 種類の blob の組合せで曲面形状を定義し、式 (5) で $\forall i : \lambda_i = 1$ としてスカラ場関数 f を定め、 $t_{iso} = 0.5$ の等値面表示を行った．曲面の法線は式 (8) 及び式 (9) の微分をフラグメントプログラムとして実装した． $\{c_i\}, \{r_i\}$ は pixel shader への定数として与え、フラグメントプログラムから参照した．

4.3 ボリュームサンプルされたスカラ場の等値面
スカラ場関数が三角形メッシュモデルなどのパラメ

トリック曲面から生成された符号付距離^(注1)場の等値面表示を行う場合には、 f が局所的にも単純な式にならないため、4.2 で述べた手法では f の値の評価コストが高くなり、表示速度が低下する。また、前節の blob 描画において、blob 数が増加した場合には、スカラ場関数の評価コストが高くなる可能性がある。この問題を改善するために、スカラ場関数の構成要素のうち描画時に変更されない部分についてあらかじめ一定の解像度でサンプルしてポリウム化しておき、描画時にはこのポリウムから関数値の近似値を得る。これにより形状の詳細度がサンプル時に決定されてしまう代わりに、描画を高速化することができる。

今、スカラ場関数 f が、描画時に変化しない関数 $f_i: \mathbf{R}^3 \rightarrow \mathbf{R}$ ($i = 1, \dots, n$) と p に依存しない汎関数 $\mathcal{F}(\dots)$ を用いて

$$f = \mathcal{F}(f_1, \dots, f_n) \quad (10)$$

と書けるとする。このとき f_i の値の計算量が大きい場合にはこれを一定の解像度の格子点であらかじめサンプルしてポリウム $g_i: \mathbf{Z}^3 \rightarrow \mathbf{R}$ を生成しておき、描画時に g_i を trilinear 補間して f_i の近似値を得る。曲面の法線に関しても同様に、 g_i と同サイズで ∇f_i の値をもつポリウム ∇g_i を trilinear 補間して ∇f_i を得た後、次の式を用いてスカラ場関数のこう配 ∇f の近似値を得る。

$$\nabla f = \mathcal{F}(\nabla f_1, \dots, \nabla f_n) \quad (11)$$

このような処理を実現するためには、 g_i 及び ∇g_i を 3D テクスチャ $\{T_f\}$ としてグラフィックスハードウェアに読み込み、式 (10) 及び式 (11) 中の \mathcal{F} をフラグメントプログラム P_f として与えることで描画時に f 及び ∇f の近似値を得ればよい。なお ∇g_i は、 f_i が微分できる場合には ∇f_i をサンプルして生成し、 ∇f_i が解析的に得られない場合には g_i に 3^3 サイズの微分フィルタをかけて生成する。

4.4 描画速度に応じた詳細度制御

グラフィックスハードウェアによる描画加速を利用する場合、表示速度を低下させる要因は主に二つある。

第1の要因は、ハードウェアによるフラグメント描画速度の低下である。基本的に、フラグメント描画速度はフラグメント数に比例し、フラグメント数はスライスの枚数に強く依存する。提案法では、ポリゴンはスカラ場を等間隔にサンプルするスライスと1対1に対応するため、描画速度に応じてスライス枚数(また

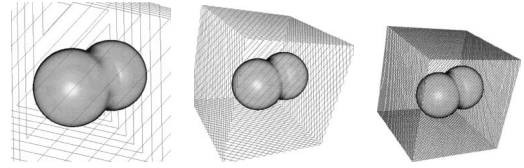


図4 描画速度に応じた詳細度変更
Fig.4 Levels-of-detail control depending on the drawing speed.

はスライス間隔)を変化させることで、描画速度を一定に保つことができる。ある時点におけるスライス枚数を n_0 、そのときの描画速度を F_{n_0} fps (frames per second) とするとき、描画速度を $F_{interactive}$ fps で維持するためには、スライスの枚数 n を

$$n = n_0 \frac{F_{n_0}}{F_{interactive}} \quad (12)$$

とすればよい。この描画速度に応じた詳細度制御 (levels of detail) の例を図4に示す。

第2の要因は、テクスチャが巨大な場合に、利用可能なビデオメモリ容量を超過してテクスチャメモリへのアクセス速度が低下することである。4.3 で述べた描画方式で、高解像度のスカラ場を処理する場合にはメモリ容量が不足する可能性がある。この問題を改善する方法としては、適応的テクスチャマッピング [14] などの階層的な表現利用する方法が考えられる。ポリウム化されたスカラ場では曲面が存在しない領域の関数値は表示には直接関係がなく、このような領域の解像度を下げることでメモリ容量を大幅に節約できる。ただし、記憶容量が足りている状況では階層化によるメモリ利用効率、処理速度の低下の問題の方が大きいため、特に対策をしていない。

5. 非多様体曲面

1. で述べたように、パラメトリック表現で与えられた曲面を等値面表現に変換することによって形状処理が容易になる場合がある。本章では、この変換の際の問題となる非多様体形状を扱う方法について述べる。

5.1 非多様体の等値面表現

$f: \mathbf{R}^3 \rightarrow \mathbf{R}$ に対し、 $\nabla f(p)$ が存在して0でないとき p は正則であるといい、 $f(p) = t_{iso}$ で定義される等値面は正則な p の近傍で2-多様体である。通常、

(注1): 曲面上の最近傍点までの距離に、最近傍点が曲面の表側の場合には正 (+) の符号を、裏側の場合には負 (-) の符号を付けた値。表裏の判定には曲面の法線を用いる。

等値面は $f(p)$ の正則な点 p の集合として定義され、したがって等値面は 2-多様体である [4]。これに対し、多くのパラメトリック表現は面の分岐や境界などの非多様体形状を表現することが容易であるため、このような曲面を距離変換してスカラ場の等値面として扱う場合、もとの形状が正しく表現できなくなるという問題がある。

これに対し Bloomenthal らは、曲面によって分割される空間の領域を考え、逆に領域の境界面で曲面を定義することで、非多様体曲面を等値面表現する手法を提案した [5]。この方法では、分割された空間の領域に番号を付け、その番号を返す関数 $f_r: \mathbf{R}^3 \rightarrow \mathbf{Z}$ と、境界に曲面が存在する領域番号の組

$$\begin{aligned} \mathbf{P} &= \{ \{f_r(p_1), f_r(p_2)\} \} \\ &\exists p_S \in (\overline{p_1 p_2} \cap S), \forall p'_1 \in \overline{p_1 p_S}, \forall p'_2 \in \overline{p_2 p_S}: \\ &p'_1 \neq p_S \wedge f_r(p'_1) = f_r(p_1) \wedge \\ &p'_2 \neq p_S \wedge f_r(p'_2) = f_r(p_2) \} \end{aligned} \quad (13)$$

を考え、式 (1) の代わりに次の式で曲面 S を定義する。

$$\begin{aligned} p &\in S \\ &\Leftrightarrow \exists p_1, p_2 \in \mathbf{R}^3: p \in \overline{p_1 p_2} \wedge \\ &|\overline{p_1 p_2}| < \epsilon \wedge \{f_r(p_1), f_r(p_2)\} \in \mathbf{P} \end{aligned} \quad (14)$$

ただし ϵ は微小な正の実数とし、曲面上の点は隣接するどこかの領域、例えば領域番号が最も大きい領域に含まれるものとする。実際に S の形状を得る際には、 f_r を微小幅 Δp でサンプルし、 $\{f_r(p), f_r(p + \Delta p)\} \in \mathbf{P}$ であれば $\overline{p, p + \Delta p}$ 上に曲面を生成する。ただし $\overline{p, p + \Delta p}$ と S の交点の座標は計算できないため、 $\overline{p, p + \Delta p}$ を繰り返し細分割し、 $|\Delta p| < \epsilon$ となったときの p を曲面との交点、すなわち曲面の位置とする。

この方法では、滑らかな曲面形状を得るためにはサンプル点の間隔を非常に細かくする必要があり、曲面を高速に表示することが困難である。そこで筆者らは、領域番号 f_r と同時に曲面までの符号なし距離 $f_d: \mathbf{R}^3 \rightarrow \mathbf{R}$ を考え、 f_d の補間によって疎なサンプル点からその間の曲面の位置を決定する手法を提案した [21]。提案法では、次の式で曲面 S を定義する。

$$\begin{aligned} p &\in S \\ &\Leftrightarrow \exists p_1, p_2 \in \mathbf{R}^3: f_d(p) = 0 \wedge p \in \overline{p_1 p_2} \wedge \\ &|\overline{p_1 p_2}| < \epsilon \wedge \{f_r(p_1), f_r(p_2)\} \in \mathbf{P} \end{aligned} \quad (15)$$

一般に、曲面が局所的に線形近似可能な場合、曲面付

近の f_d の変化は線形である。このとき、 $f_d(p_1)$ と $f_d(p_2)$ を線形補間することにより式 (15) を満たす p を次の式で推定できる。

$$p = \eta p_1 + (1 - \eta) p_2, \quad \eta = \frac{f_d(p_2)}{f_d(p_1) + f_d(p_2)} \quad (16)$$

f_d で定まる関数場が線形近似できる限り、疎なサンプリング、すなわち式 (15) の ϵ が大きな値の場合にも非多様体の等値面の位置を決定することができる。なお、 f_r と f_d を同時に返すベクトル値関数として領域付距離関数 $f_s: \mathbf{R}^3 \rightarrow \mathbf{Z} \times \mathbf{R}$ を考えると、これは符号付距離の、三つ以上の領域への一般化であると考えられる。

5.2 非多様体の等値面描画

符号付距離関数 $f_s = (f_r, f_d)$ はベクトル値関数であり、 f_s で定義されたスカラ場の等値面をスライスを使った ray casting で表示するためには、pre-integration 表を拡張して 4 次元の表を生成する必要がある。ただし現在の PC グラフィックスハードウェアでは 4D テクスチャが利用できないため、筆者らは f_s を実数空間に埋め込んでスカラ場関数 \tilde{f}_s に変換し、通常の等値面と同様の枠組みで表示する方法を提案している [21]。

一般に、等値面表現では曲面から十分離れた点におけるスカラ場の値は形状に影響を与えない。そこで、ある十分大きい実数 $\xi > 0$ を考え、 ξ で飽和する距離

$$\tilde{f}_d(p) = \min(f_d(p), \xi) \quad (17)$$

を用いて

$$\tilde{f}_s(p) = \tilde{f}_d(p) + \xi f_r(p) \quad (18)$$

という関数を定義する。このとき $\tilde{f}_s(p)$ から $\tilde{f}_d(p)$ 、 $f_r(p)$ を一意に決定できることから、2 点 p_1, p_2 が $\{f_r(p_1), f_r(p_2)\} \in \mathbf{P}$ を満たすとき、次の式により 2 点間に存在する曲面の位置 p を決定できる。

$$\begin{aligned} p &= \eta p_1 + (1 - \eta) p_2, \\ \eta &= \frac{\tilde{f}_s(p_2) - \xi f_r(p_2)}{(\tilde{f}_s(p_2) - \xi f_r(p_2)) - (\tilde{f}_s(p_1) - \xi f_r(p_1))} \end{aligned} \quad (19)$$

図 5 左に非多様体形状をスカラ場の等値面として表現するための領域分割の例を示す。ここでは $f_r(p) \in \{0, 1, 2, 3\}$ とした。pre-integration 表を作成するため $\tilde{f}_s(p) \in [0, 255]$ と量子化すると、式 (18) と $0 \leq f_r(p) \leq 3$ より $\xi \leq 64$ である必

要がある．そこで $\xi = 64$ とおくと，式 (19) に対応する pre-integration 表は図 5 右のとおりである．この表から，例えばこの表内の点 (a) より $\{f_r(p_{front}), f_r(p_{back})\} = \{1, 3\}$ の境界には面が存在せず，点 (b) より $\{f_r(p_{front}), f_r(p_{back})\} = \{1, 2\}$ の境界には曲面が存在することが分かる．また表の色は曲面の色を，濃さは 2 点間における曲面の位置を表す．

ξ の選び方には任意性があり，通常は可能な値のうち最大の数を用いればよい．領域数が増えた場合には，pre-integration 表のサイズを大きくすることで ξ を一定以上の大きさに保つことができる．

領域数が増えても描画速度には影響しないが，pre-integration 表の大きさによって扱える領域数は制限される．例えば，pre-integration 表を 256^2 サイズで実装するとき，256 以上の領域を扱うことはできない．また pre-integration 表のサイズが有限の場合，領域数が増えると ξ を小さくする必要があるが， ξ が小さくなると式 (19) で得られる補間面の位置が不正確となるため，実際には pre-integration 表のサイズは領域数に対して十分大きくとる必要がある． ξ が可能な最小の値をとるとき，提案法は Bloomenthal の手法 [5] と同等であるという特徴をもつ．

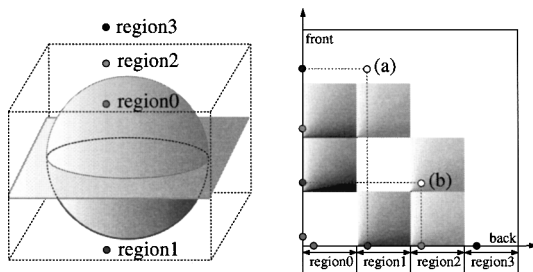


図 5 左：非多様体曲面に対する領域分け．右：対応する pre-integration 表
Fig. 5 Left: An example of segmentation for a non-manifold surfaces. Right: The corresponding pre-integration table.

式 (18) の領域付距離を用いてスカラ場を定義する際には，スカラ場が不連続になる．そのため，4.3 で述べた方法で曲面を表示する場合には，ポリウム関数を表現するテクスチャを線形補間すると，不連続な点の間の補間により本来存在しない関数値が生成され，曲面が誤って表示される不具合が起こる．これを防ぐためには，領域付距離をポリウムで表す場合には最近傍点 (nearest neighbor) 補間を利用する必要がある．

6. 実験結果

実験に利用した計算機は CPU が Pentium4 1.7 GHz，主記憶容量が 1.0 GB の PC である．グラフィックスハードウェアは，特に断りがない限り，GPU が ATI 社製 Radeon 9800 PRO，ビデオメモリがともに 128 MByte である．実装には Open GL グラフィックスライブラリと Nvidia 社の Cg 言語を用いた．表示処理の際の表示画面のサイズは 512^2 である．

図 6 では，4.2 で述べた手法を用い，式 (8) の blob を用いて生成した陰関数曲面を既存手法と提案手法で描画した結果を比較している．左は particle [20]，中央は ray casting，右は提案手法による描画結果である．particle 表示には [11] の手法を用い，描画速度が平均 20 fps になるように particle の発生密度を設定した．ray casting には POV-RAY を用い，描画速度は約 0.1 fps であった．提案手法では 4.4 で述べた方法を用い，描画速度が 20 fps となるようにスライス枚数を決定した．提案手法で表示された曲面は，particle を用いた手法と同等の描画速度で，ray casting による表示の結果と非常に近い結果が得られていることが分かる．

図 7 に，4.3 と 5. で述べた手法を用い，複数の形状モデルの混合処理の様子を示す．三角形メッシュとして与えられた二つの形状モデルを，同一の領域数と ξ で定義されるスカラ値の符号付距離を用いて等値面

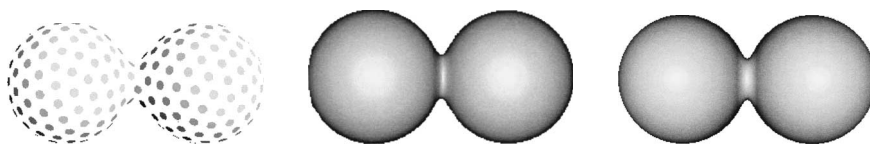


図 6 blob レンダリング．左：particle レンダリング．中央：ray casting．右：提案手法
Fig. 6 Result of rendering blobs. Left: particle rendering. Center: ray casting. Right: ours.

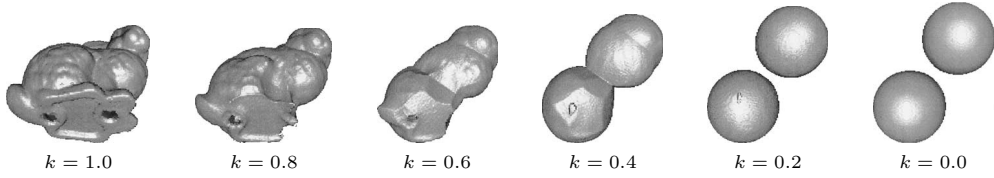


図 7 位相が異なる二つの形状混合. κ は式 (21) における内挿パラメータ

Fig. 7 The result of blending two surface models with different topology. κ is an interpolating parameter used in Eq. (21).

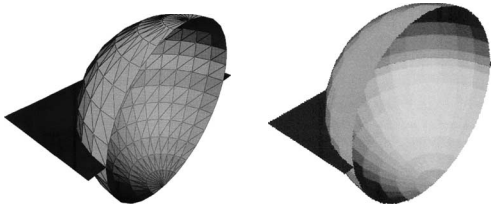


図 8 左: 入力として与えた非多様体メッシュモデル. 右: スカラ場表現に変換後, 提案手法で表示した結果

Fig. 8 Left: A non-manifold mesh model given as an input. Right: The isosurface visualized by our method.

表現し, そのスカラ場関数をそれぞれ

$$f^{(1)} = \tilde{f}_d^{(1)} + \xi f_r^{(1)}, \quad f^{(2)} = \tilde{f}_d^{(2)} + \xi f_r^{(2)} \quad (20)$$

とするとき, 内挿パラメータ κ で混合された曲面のスカラ場関数 f を

$$\begin{aligned} f(p) &= \bar{f}_d(p) + \xi \bar{f}_r(p), \\ \bar{f}_d(p) &= |\kappa \tilde{f}_d^{(1)} - (1 - \kappa) \tilde{f}_d^{(2)}|, \\ \bar{f}_r(p) &= \begin{cases} f_r^{(1)} & \text{if } \kappa \tilde{f}_d^{(1)} > (1 - \kappa) \tilde{f}_d^{(2)}, \\ f_r^{(2)} & \text{otherwise} \end{cases} \end{aligned} \quad (21)$$

と定義した. 描画時には $f^{(1)}, f^{(2)}$ を 3D テクスチャ, κ や $f^{(1)}, f^{(2)}$ の位置に関するパラメータを定数, f をフラグメントプログラムとして pixel shader に与えた. 定数を描画フレーム間で変更することによって, 形状の混合処理を対話的に行うことができる. この表示結果においても描画速度は 20 fps になるようにスライス間隔を設定している. 図中で, 左の形状は曲面上に穴のある非多様体であるが, 提案法ではその特徴を描画することができている.

図 8 に, 5. で述べた手法を用い, 非多様体曲面を領域付距離を用いてスカラ場の等値面表現し, 提案手法で表示した結果を示す. 左が入力として与えた三角形メッシュ, 右が提案手法によって表示された等値面である. 用いた領域数は 4 であり, 領域分割は [21] の

表 1 描画速度の比較

Table 1 Comparison of rendering speed.

	rendering (fps)				modeling (spf)
	#blobs	5	10	20	40
proposed method	33.5	17.5	14.8	10.1	~0
volume rendering	22.1	22.1	22.1	22.1	4.5
ray casting	0.1	0.1	0.09	0.08	~0

方法で行った. 図 8 右はスカラ場を 256^3 サイズのボリュームでサンプルして曲面を表示した結果であり, 描画速度は 20 fps になるようにスライス間隔を設定している. スカラ場をボリュームによって離散化したため, ボクセル幅以下の形状を正確に表現することはできず, 境界付近の形状は多少変化している.

表 1 に, 4.2 で述べた手法を用い, blob モデリングを行った際の描画速度の比較を示す. 具体的には blob の追加, 選択, 移動を対話的に行い, 図 3 のような形状を生成した. この実験では, 速度の評価のためにスライス枚数を 32 に固定し, グラフィックスハードウェアとして NVidia 社製 GeForce FX 5800 Ultra を用いた. 表は, 中央列が視点のみを変更したときの描画速度, 右列が曲面形状を変更したときの付加的な処理時間を示す. 形状変更の際の処理速度は spf (seconds per frame) で表している. 比較した手法は上から順に, 提案法, ボリュームレンダリング, ソフトウェア ray casting である. 提案法では, 各 blob 関数を, 現在編集中の blob を関数の形で, その他の blob を 32^3 サイズのボリュームとして与えた. ボリュームレンダリングを用いる方法では, スカラ場関数の全体を 256^3 サイズのボリュームで表現した. 提案法では blob 数の増加とともにフラグメントプログラムが長くなり, 描画速度が低下する傾向があることが分かる. ただし, 各 blob の描画は近傍の blob のみに影響されるために, その速度の低下率は blob 数の増加とともに減少する傾向がある. ボリュームレンダリングを用いる方法では, 曲面形状が変化しない場合は描画速度を一定に保

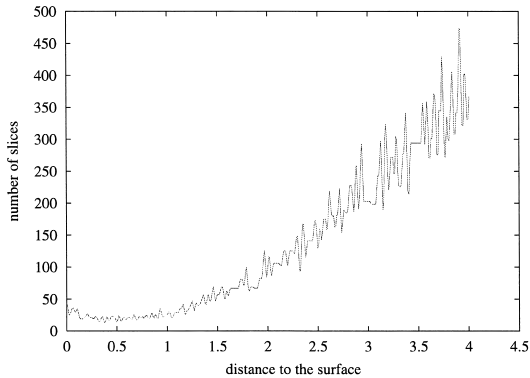


図 9 20 fps 描画時の、スライス数と視点から曲面までの距離の関係

Fig. 9 The relationship between the number of slices and the distance to the surface for the rendering at a speed of 20 fps.

つことができるが、blob の数、位置、大きさなどの変更時にボリューム生成の待ち時間が発生し、モデリング用途の可視化法としては実用的ではなかった。ray casting では、対話的に操作することはできなかった。

図 9 は、4.4 で述べた手法を用い、視点変更に伴いスライス枚数を変化させた結果のグラフである。描画対象は半径 0.3 の球面であり、 $F_{interactive} = 20$ fps とした。グラフの横軸が球の中心と視点の距離、縦軸がスライス枚数を表す。一般に描画面上における対象物の面積は対象までの距離の 2 乗に反比例するため、描画速度を一定に保つときのスライス枚数は対象物と視点の距離の 2 乗にほぼ比例することが分かる。

7. む す び

本論文では、PC グラフィックスハードウェアのテクスチャ合成機能を用いてスカラ場の等値面を高速に描画する手法を提案した。描画結果は ray casting とほぼ同等の画質であり、描画速度は目的に応じて必要な値に保つことが可能である。実験では、blob によるモデリング、位相の異なる形状のモーフィングに対して提案手法を適用し、スカラ場の等値面表現を用いて対話的に形状処理を行うことが可能であることを示した。

文 献

- [1] A. Appel, "Some techniques for shading machine renderings of solids," Proc. AFIPS Joint Computer Conference, vol.32, pp.37-45, 1968.
- [2] J. Blinn, "A generalization of algebraic surface drawing," ACM Trans. Graphics, vol.1, no.3, pp.235-256, 1982.
- [3] J. Bloomenthal, "Polygonization of implicit surfaces," Comput. -Aided Geom. Des., vol.5, pp.341-355, 1988.
- [4] J. Bloomenthal, Introduction to Implicit Surfaces, Morgan Kaufmann Publishers, 1997.
- [5] J. Bloomenthal and K. Ferguson, "Polygonization of non-manifold implicit surfaces," Proc. SIGGRAPH '95, pp.309-316, 1995.
- [6] M.L. Brady, K. Jung, H.T. Nguyen, and T. Nguyen, "Two-phase perspective ray casting for interactive volume navigation," Visualization '97, pp.183-190, 1997.
- [7] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," Proc. ACM Symposium on Volume Visualization, pp.91-98, 1994.
- [8] M. Desbrun and M.-P. Cani, "Animating soft substances with implicit surfaces," Proc. SIGGRAPH '95, pp.287-290, 1995.
- [9] R.A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," Proc. SIGGRAPH '88, pp.65-74, 1988.
- [10] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware, pp.9-16, 2001.
- [11] J.C. Hart, E. Bacht, W. Jarosz, and T. Fleury, "Using particles to sample and control more complex implicit surfaces," Proc. Shape Modeling International, pp.129-136, 2002.
- [12] A. Hilton, A.J. Stoddart, J. Illingworth, and T. Winder, "Reliable surface reconstruction from multiple range images," Proc. European Conference on Computer Vision, pp.117-126, 1996.
- [13] L. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature sensitive surface extraction from volume data," Proc. SIGGRAPH 2001, pp.57-66, 2001.
- [14] M. Kraus and T. Ertl, "Adaptive texture maps," Proc. Eurographics/SIGGRAPH Graphics Hardware 2002, pp.7-15, 2002.
- [15] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," Proc. SIGGRAPH '94, pp.451-458, 1994.
- [16] A. Lierios, C.D. Garfinkle, and M. Levoy, "Feature-based volume metamorphosis," Proc. SIGGRAPH '95, pp.449-456, 1995.
- [17] W. Lorensen and H. Cline, "Marching cubes: A high resolution 3D surface reconstruction algorithm," Proc. SIGGRAPH '87, pp.163-169, 1987.
- [18] R.N. Perry and S.F. Frisken, "Kizamu: A system for sculpting digital characters," Proc. SIGGRAPH 2001, pp.47-56, 2001.
- [19] G. Taubin, "Curve and surface smoothing without shrinkage," Proc. International Conference on Com-

puter Vision '95, pp.852-857, 1995.

- [20] A.P. Witkin and P.S. Heckbert, "Using particles to sample and control implicit surfaces," Proc. SIGGRAPH '94, pp.269-278, 1994.
- [21] S. Yamazaki, K. Kase, and K. Ikeuchi, "Hardware-accelerated visualization of volume-sampled distance fields," Proc. Shape Modeling International 2003, pp.264-271, 2003.

(平成 15 年 8 月 4 日受付, 16 年 2 月 2 日再受付)



山崎俊太郎 (正員)

1999 東大・理・情報科学卒, 2004 同大学院情報理工学系研究科コンピュータ科学専攻博士課程了, 同年より独立行政法人産業技術総合研究所研究員・博士(情報理工学). IEEE, ACM 各会員.



加瀬 究

1989 東大・工・精密機械卒. 1994 同大学院工学系研究科博士課程了, 同年より独立行政法人理化学研究所研究員・博士(工学). 精密工学会賞(1993年). 精密工学会, 情報処理学会, 形の科学会, IEEE, ACM 各会員.



池内 克史 (正員)

1973 京大・機械卒, 1978 東京大学大学院情報工学博士課程了. 博士(工学). MIT 人工知能研究所, 電総研, CMU 計算機科学部を経て, 1996 より東京大学大学院情報学環教授. 論文賞(ICCV-90, CVPR-91, AIJ-92, 日本ロボット学会誌-97, IEEE R&A 誌-98, 日本 VR 学会誌-00) 受賞. 情報処理学会, 人工知能学会, OSA, IEEE (Fellow) 各会員.