離散境界を用いた距離関数の構築手法

原田隆宏[†] 越塚誠一[†]

距離関数とはある境界条件のもとに計算領域内で求まる解の 1 つであり，境界までの最小距離を表すものである．距離関数は様々な研究分野において用いられているが，その計算コストは高い．本研究ではまず距離関数の計算において離散境界という概念を導入する．これによって境界の次元を下げることができ，複雑な境界条件での計算を単純な計算に変換することができる．次にこの離散境界を用いて高速に距離関数計算を行う．本手法ではポリゴンモデルを離散境界に分解するため，入力モデルのポリゴン数が計算時間に及ぼす影響は既存研究と比べ小さい．そのため高解像度のポリゴンモデルを用いても高速に計算を行うことが可能である．また本手法は境界として用いるポリゴンモデルのトポロジに関して制約条件が必要ないので，完全に閉じていないポリゴンモデルの計算にも用いることができる．本手法は離散境界を求める空間解像度を調整することによって計算時間を制御することも可能であるという特徴も持つ．

Distance Function Computation Using Discrete Boundaries

Takahiro Harada[†] and Seiichi Koshizuka[†]

Distance function which is a solution with a boundary condition in a computational domain, indicate the minimum distance to the boundary. Although the distance function is used in a wide range of research areas, the computational cost is high. In this paper, we introduce discrete boundaries which reduce that the dimension of the boundary. Then, we present a fast distance function computation algorithm based on discrete boundaries. Since the present method decomposes a polygon model into discrete boundaries, the computational time is less sensitive to the number of polygons than that of the past methods. As a result, fast computation of the distance function is possible with a highly tessellated model. Moreover, our method is not restricted to closed polygon model because it does not need any requirement about topology of a polygon model. The proposed method can control computation time with varying the spatial resolution of discrete boundaries.

1. 序　　論

距離関数はある空間において境界条件が与えられたときに，その境界構成要素の最近点までの距離を表す関数である．距離関数の特徴はこれ以外にも存在し，距離関数の空間一階微分は最近点に向かうベクトルであり，また空間二階微分は曲率を表す．このような特徴を持つ距離関数は数値計算やコンピュータグラフィックスをはじめとする様々な分野において用いられている．

たとえば距離関数は流体力学の自由表面流れの界面位置を保持する関数として用いることができる[5]．また剛体や弾性体の力学を考えるときには接触は避けることのできないテーマであり，距離関数を用いることができる[6]．距離関数はある点の物体までの距離を表すだけでなく，接触時に働く力の大きさや方向も同時に得ることができるのでとても有用である．それ以外にもモデルの骨格の抽出[4]，経路設計[7] などにも用いられている．

本手法はまず離散境界という概念を導入し，境界の次元を下げる．そして離散境界を用いた距離関数の計算を行う．既存研究と異なり，本手法ではモデルのポリゴン数が計算コストに与える影響は小さいため，ポリゴン数の多いモデルにおいても少ないモデルと比べ大きく計算時間は増加しないという利点を持つ．また本手法は入力とするモデルのトポロジについて制約条件が必要ないので，どのようなモデルを入力しても距離関数を構築することができる頑強な手法である．既存の研究は多くの複雑形状の構築を行わなければならなかったが，提案手法はそれらの構築を行う必要もないため，実装も容易である．今まで提案されてきた計算手法はどれも計算時間を制御することはできなかったが，提案手法は離散境界の空間解像度を変えるこ

† 東京大学
The University of Tokyo

とによって計算時間の制御も可能である．提案手法はGPU を用いて高速化することができ，高解像度のポリゴンモデルを境界とする問題でも短い計算時間で距離関数を求めることができる．

2. 関連研究

ある境界条件のもと距離関数を求める研究は多く行われている．Chamfer Distance Transform では，ある点での距離関数は距離のテンプレートを用いて近傍の点の値に距離を足していくことで計算する[1]．Vector Distance Transform では，ある点での境界までのベクトルを近傍の点から境界までのベクトルにベクトルテンプレートを用いてベクトルを足すことで計算し，このベクトルの大きさを計算することで距離関数を計算している[2]．これら以外にも距離関数計算手法は存在し，Jones らがレビュー論文で詳細に触れている[8]．

距離関数の算出は計算負荷が高いため，GPU を用いた研究も行われている．本研究も GPU を用いて距離関数を計算するため，これに関連する研究について詳しく述べる．Hoff らは 2 次元でのボロノイ図を 3 次元形状のラスタライズによって求める手法を提案し[7]，ボロノイ領域と距離関数は密接な関係にあることを用いて，距離関数計算を行っている．この手法は 3 次元格子を 2 次元格子の集合と分解し，サイトの形成する 3 次元形状を生成することによって各 2 次元格子でのボロノイ図を求めている．そのため精度の良い距離関数を求めるには十分細かい 3 次元形状を作成しなければならない．

この手法の多量のポリゴンを生成しなければならない問題点を改善する手法を Sud [12] らが開発した．この研究では計算の効率化のためにボロノイ図の隣接情報などを用いてサイトのクラスタリングを導入し計算の高速化を行った．

Mauch は Characteristics / Scan-Conversion（CSC）algorithm を用いてポリゴンの面を押し出すことで多面体を作成し，距離関数を計算している[9]．この手法ではさらにポリゴン間で共有している辺や頂点からも多面体を作成するため，多量の 3 次元形状を作成しなければならなかった．Mauch の実装は CPU を用いた実装であったが Sigg らは GPU を用いてこの手法の高速化を行った[10]．CSC algorithm は多くの形状を作成しなければならなかったため Sigg ら[10]はこれらを減らす手法を提案している．しかしこの手法を用いて作成される多面体内部の距離関数は線形分布をしていないため，GPU を用いた計算ではフラグメントシェーダ内部で複雑な処理を行う必要があった．

このように生成される多面体内部だけでなく，一般的にあるサイトからの平面上の距離関数の分布は非線形である．そこで Sud らはサイトに属する領域においてサイトに向かうベクトルは線形に分布しているということに注目したアルゴリズムを提案している[11]．この手法は Hoff らの手法[7] のように高精度の距離関数を求めるために多くのポリゴンを生成する必要がない．

このように距離関数を求めるには多くの形状の構築を行わなければならず，多大な情報を GPU に送り処理しなければならない．またすべての既存研究では距離関数の構築の計算コストはモデルを形成しているポリゴン数に比例した処理となっており，ポリゴン数が増えれば計算コストは大きく増えてしまうという共通の欠点があった．

3. 手法

提案手法は，3 次元ポリゴンモデルを離散境界に分解し，ポリゴンモデル全体から求まる距離関数を離散境界から求まる距離関数の関数として計算する（**図 1**）．そのため，距離関数を求める計算にポリゴンを用いないので計算コストはポリゴン数に依らないものとなる．本手法の距離関数の計算コストは離散境界の数，すなわちポリゴンモデルの表面積に比例する．本章ではこれを理論的に展開していき，次章で実装方法について述べる．本手法の特徴である計算コストはポリゴンモデルの表面積に比例するということを数値的に示す．

3.1 概念

符号付き距離関数は Eikonal 方程式 $|\nabla u(\boldsymbol{x})| = c$ を $u(\boldsymbol{x})|_S = 0$ の境界条件下で解いた解 $u(\boldsymbol{x})$ である．また距離関数とはこの解の絶対値をとったものであり $d(\boldsymbol{x}) = |u(\boldsymbol{x})|$ と定義される．Eikonal 方程式の右辺が $c = 1$ のとき距離関数は境界 S までのユークリッド距離を表す．ここで計算領域全体において境界 s によって求まる距離関数を $d(s)$ とする．境界 s が 2 つの境界 s_0，s_1 に分解されるとき，ある点 $\boldsymbol{x}$ での距離関数は最も近い境界までの距離となる．つまり距離関数 $d(s)$ は以下のように分解することができる．

$$\begin{aligned} d(s) &= d(s_0 + s_1) \\ &= g(d(s_0), d(s_1)) \end{aligned} \tag{1}$$

ここで関数 g は以下のように定義される．

$$g(f(x), f(y)) = \begin{cases} f(x) & d(x) \leq d(y) \\ f(y) & d(y) < d(x) \end{cases} \tag{2}$$

この性質を用いて 3 次元ポリゴンモデルを境界とした問題を考える．あるポリゴンモデルの境界 p は n

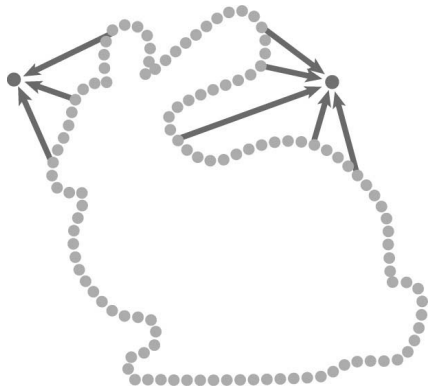

図 1 離散境界を用いた距離関数の計算
Fig. 1 Distance function calculation using discrete boundaries.

個のポリゴン $\{f_0, f_1, f_2, \cdots, f_n\}$ で構成されているとする．すると式 (1) より p を境界条件とする距離関数 $d(p)$ は

$$d(p) = g(d(f_0), d(f_1), d(f_2), \cdots, d(f_n)) \quad (3)$$

と表すことができる．

同様に 1 つのポリゴン f_i はそれを構成する m 個の点要素 $\{p_{i,0}, p_{i,1}, p_{i,2}, \cdots, p_{i,m}\}$ に離散化することができる．この点要素を離散境界と呼ぶ．この f_i を境界条件とする距離関数 $d(f_i)$ は式 (1) を用いて同様に書き換えられる．

$$d(f_i) = g(d(p_{i,0}), d(p_{i,1}), d(p_{i,2}), \cdots, d(p_{i,m})) \quad (4)$$

この式変形はモデル m を構成するすべてのポリゴンに対して用いることができる．

点 $\boldsymbol{x}$ における境界 s までの距離関数は $d(\boldsymbol{x})$ と表した．ここで $d(\boldsymbol{x})$ を空間微分し

$$\boldsymbol{v}(\boldsymbol{x}) = \frac{\partial d(\boldsymbol{x})}{\partial \boldsymbol{x}} \quad (5)$$

と表す．この $\boldsymbol{v}(\boldsymbol{x})$ を勾配ベクトルと呼ぶ．この勾配ベクトルは点 $\boldsymbol{x}$ から最も近い境界までのベクトルとなることが知られている．計算領域全体において境界 s によって求まる距離関数の勾配ベクトルを $\boldsymbol{v}(s)$ とすると $\boldsymbol{v}(s)$ は式 (1) と同様に

$$\begin{aligned}\boldsymbol{v}(s) &= \boldsymbol{v}(s_0 + s_1) \\ &= g(\boldsymbol{v}(s_0), \boldsymbol{v}(s_1))\end{aligned} \quad (6)$$

と分解することができる．ポリゴンモデルの境界 p から求まる勾配ベクトルもその要素であるポリゴンを境界条件として求まる勾配ベクトルを用いて

$$\boldsymbol{v}(m) = g(\boldsymbol{v}(f_0), \boldsymbol{v}(f_1), \boldsymbol{v}(f_2), \cdots, \boldsymbol{v}(f_n)) \quad (7)$$

と表すことができ，ポリゴン f_i の勾配ベクトルも点要素による勾配ベクトルを用いて

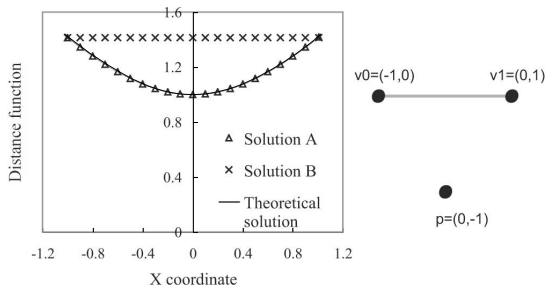

図 2 格子点上以外での距離関数を勾配ベクトルを線形補間して求めた解 Solution A，格子点上の距離関数を線形補間して求めた解 Solution B と理論解
Fig. 2 Solution A is a solution which is calculated by interpolating the gradient vector on the grid points, solution B is a solution which is calculated by interpolating the distance function on the grid points and the theoretical solution.

$$\boldsymbol{v}(f_x) = g(\boldsymbol{v}(p_{i,0}), \boldsymbol{v}(p_{i,1}), \boldsymbol{v}(p_{i,2}), \cdots, \boldsymbol{v}(p_{i,m})) \quad (8)$$

と表すことができる．

これらの変形は距離関数を求める際の境界条件の次元を落とす変形である．式 (3)，(7) は 2 次元曲面の境界 m を 2 次元平面の境界群に分解する変形である．さらに式 (4)，(8) は 2 次元平面の境界を零次元の境界群に分解する変形である．2 次元曲面の境界条件のもと解かなければならなかった問題が零次元の境界条件を持つ問題に変形されたことにより，解を求める計算が容易になる．零次元要素を境界とする問題は，点と点の距離を計算することのみによって距離関数を求めることができる．

今までの議論は空間において連続に距離関数が分布する場合を前提としてきたが，一般的には計算領域を離散化して離散距離関数を用いることが多い．空間上での 1 枚の平面と 1 点が存在する場合を考える．するとその面上では点を境界とする距離関数は非線形な分布になる．格子点での離散距離関数を求めた後に格子点以外の位置での距離関数を求めるときに，格子点での値から線形補間したのでは正しい値は得ることができない．しかし Sud らが述べられているように平面上での勾配ベクトルは線形に分布している[11]．そのため格子点以外の位置での値は勾配ベクトルを線形補間し，それらの大きさを計算することにより距離関数は正確に計算することができる．この事実の詳細な記述は Sud らが論じているため[11]，本論文では例をあげて説明する．**図 2** 右に示す p を境界として v_0，v_1 においてまず勾配ベクトルと距離関数を求める．この 2 点上での勾配ベクトルを線形補間して，線分 v_0v_1 上の距離関数を求めた解は図 2 左に示す Solution A で

あり，距離関数を線形補間した解は Solution B である．そして理論解は線で示したものである．このように距離関数を線形補間すると補間による誤差が生じてしまうのに対し，勾配ベクトルを線形補間して求めた距離関数は理論解と一致する．格子点での距離関数を求めた後，格子以外の位置での距離関数を求めるときに格子点での距離関数を線形補間するか勾配ベクトルを線形補間し距離関数を求めるかは使用事例に応じて使い分けるべきである．本手法でも離散境界からの勾配ベクトルを求め，それを線形補間することで距離関数の精度をあげることができる．なお本論文で示す結果は格子点上の勾配ベクトルを求め，それ以外の部分での距離関数は格子点上の勾配ベクトルを線形補間して求めたものである．

本手法を用いて距離関数を求めるときには離散距離関数を求める．そのため面要素から離散境界を求める際に空間に均一に分布する離散境界を用いるのが，ある計算誤差における分解では最も計算効率が良い．空間的に離散境界の疎密が存在すると全体の計算誤差の最大値は疎な部分におけるものとなり，密な部分を作成しても最大誤差は減少しない．このように空間的に均一な離散境界を求めることによって，距離関数計算のコストは離散境界の数に比例するものとなる．すなわち境界 m の面積に比例する計算である．面の分割の細かさのみ異なる同じ形状を表した 2 つのモデルがあるとすると，それらのモデルから生成される離散境界は同じ数となり，距離関数計算のコストは面の数によらずモデルの表面積に比例する計算とすることができる．

3.2 空間の変換

本手法は距離関数の計算に GPU を用いて高速化することが可能だが，GPU は 3 次元格子の距離関数を出力することができない．GPU での実装の詳細は後述するが，ここでは出力において導入する関数 $t(u,v)$ について説明する．

本手法は距離関数を 3 次元格子上に出力する．この 3 次元格子の各辺の方向をそれぞれ X，Y，Z 軸と定める．GPU は 2 次元格子を出力することが可能なので Z 軸方向に格子を分割し，複数枚の 2 次元格子の集合とする．この次元格子をスライスとよび，1 枚の大きな 2 次元格子に敷き詰めることにより 3 次元格子を 2 次元格子で表現することができる（**図 3**）．ここで 2 次元格子座標 u，v を 3 次元格子 x，y，z に変換する関数 $t(u,v)$ を導入する．x，y，z は $[0,1]$ の値をとるとする．2 次元格子は u，v 軸方向にスライスを l 枚保持し u，v それぞれは $[0,1]$ の値をとるものとすると 3 次元格子座標 x，y，z を出力する関数 $t(u,v)$ は以下のように表される．

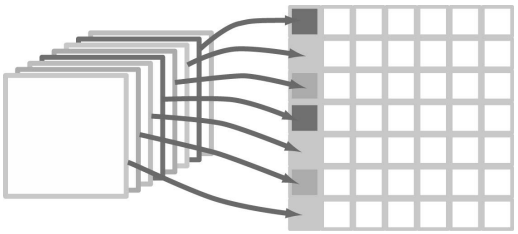

図 3 3 次元格子の 2 次元格子への変換
Fig. 3 Translation from three-dimensional grids to two-dimensional grids.

$$x = \frac{1}{s}(u - s\,[u/s]) \tag{9}$$

$$y = \frac{1}{s}(v - s\,[v/s]) \tag{10}$$

$$z = \frac{1}{l^2}(l\,[v/s] + [u/s]) \tag{11}$$

ここで $s = 1/l$ である．

これらを用いると 3 次元空間内の距離関数は

$$\begin{aligned} d(\boldsymbol{x}) &= d(x,y,z) \\ &= d(t(u,v)) \end{aligned} \tag{12}$$

と変形でき，2 次元空間内での距離関数として求めることができる．

4. 実　　装

GPU は計算を並列で行うことができるようになっており近年その性能の向上は著しく，処理によっては CPU を上回ることが可能である．本手法も GPU を用いることで計算の高速化が可能である．本章ではその実装方法について述べる．

4.1 離散境界への分解

距離関数の計算を行う前に，入力として与えた境界 p を離散境界に分解しなければならない．空間上に均一に離散境界を分布させるため，あらかじめ計算領域を格子状に分割し，それらの格子点上に離散境界を配置する．この処理は 3 次元空間内でのポリゴンモデルの表面のボクセル化にあたる．この処理では Dong らが開発した表面のボクセル化手法を応用して境界の離散境界への分解を行う[3]．

4.1.1 手　　法

GPU はポリゴンモデルをバッファに出力するときに 2 次元格子上での色として出力する．バッファの格子は空間に均一なものとなっている．この手法はこの原理を応用して仮想的な 3 次元格子をバッファ内部に構築する．バッファにモデルがレンダリングされると同時に深度バッファにはその点の深度が保持される．

この深度バッファの情報を取り出すことでポリゴンモデルの表面の座標を 2 次元格子上に離散化したものを得ることができる．しかしこれでは視点に最も近い点の情報しか得ることはできず，視点から見ることのできないポリゴンの情報は得られない．そこで Dong らはバッファの 1 ビットに 1 つの格子点を割り当てることにより 3 次元格子を 2 次元バッファ内に構築している．ポリゴンの深度値はこの 3 次元空間内の位置に変換され，存在位置のビットに値が書き加えられる．ポリゴンの重なりも表現するため，深度テストは無効にし，アルファブレンディングを用いてビットの加算を行うことによりモデルの 3 次元ラスタライズを行うことができる．Dong らはバッファの RGBA の各チャネルに 8 ビット用意して，1 ピクセルで 8×4 ビットすなわち 32 枚の 2 次元格子を表現している．ブレンディングを行うことができるのは 1 チャネル 16 ビットまでのバッファなので本研究ではこのバッファを用いて 1 ピクセルに 16×4 ビットを割り当てた．このようにすることで 1 枚のバッファに格納することのできるデータが 2 倍になり，その結果 Dong の手法では n 枚のバッファを描画しなければならなかった問題が $n/2$ 回のレンダリングで済むようになる．この変更でレンダリング回数が半分になったため，多くの形状をレンダリングしなければならない場合の効率が向上する．たとえば，ポリゴンモデルを構成しているポリゴン数が多い場合などである．

3 次元モデルをバッファにレンダリングするときに視線と平行な面は線として描画される．つまり面の情報が欠落してしまう．そのため，この 3 次元ラスタライズ手法では視線ベクトルと平行な面は正しくラスタライズすることができないので，この手法では 3 方向からラスタライズしている．これにより，欠落した情報をすべて得ることができる．

4.1.2 データの統合

このように得た 3 方向からのビット圧縮されたボクセルデータは 1 つのデータに統合しなければならない．Dong らはビットの参照テーブルを用意し GPU 上でこれを行っていた．しかし 1 チャネル 16 ビットを用いるとこの参照テーブルは $65{,}536 \times 16$ の大きさになってしまう．この大きさのテーブルを保持するとメモリを多く用いなければならず，1 枚のバッファで表現することのできる大きさを超えてしまう．そのため本研究ではビットの参照テーブルを用いずに，CPU にデータを読み戻して処理を行った．この CPU での処理は読み戻した各ピクセルにビット圧縮されていたボクセルデータを 3 次元配列に復元し，XYZ 3 方向からのデータを統合する処理である．**図 4** に処理の疑似コードを示す．初めに X 方向から射影してボクセル化したデータのデータを復元し，その後同様に Y，Z 方向から射影したデータを復元する．この復元処理では各ピクセルのビットをすべて見ていくためにデータのビットを 1 つづつシフトしていき，最下位のビットを見る．そしてそのビットが 1 ならばそのボクセルにデータが存在するので値を書き込んでいく．X，Y，Z 方向から射影したデータはそれぞれ VX，VY，VZ に復元する．すべてのデータを復元した後に，データの統合を行う．この処理はそれぞれのボクセルに対応する VX，VY，VZ の値を足すことで行うことができる．

```
// Bit decompression (projected from X)
FOR ALL SLICES
  FOR ALL PIXELS
    FOR ALL BITS
      IF(LASTBIT==1)
        VX[currentVoxel,j,i]=1
// Bit decompression (projected from Y)
FOR ALL SLICES
  FOR ALL PIXELS
    FOR ALL BITS
      IF(LASTBIT==1)
        VY[i,currentVoxel,j]=1
// Bit decompression (projected from Z)
FOR ALL SLICES
  FOR ALL PIXELS
    FOR ALL BITS
      IF(LASTBIT==1)
        VZ[i,j,currentBit]=1
// Data synthesis
FOR ALL VOXELS
  V[i,j,k]=VX[i,j,k]+VY[i,j,k]+VZ[i,j,k]
```

図 4 CPU でのデータ統合
Fig. 4 Data synthesis on CPU.

このように 16 ビットを用いてデータを圧縮しボクセル化することで 128^3 の 3 次元空間は各方向 128^2 のバッファを 2 枚読み戻すだけで行うことができる．

4.2 距離関数の計算

ポリゴンモデルの境界から離散境界を求めることができたので，これを用いて距離関数を計算する．すべての離散境界において距離関数を求め，式 (4) を用いて境界 m からの距離関数を構築する．距離関数が

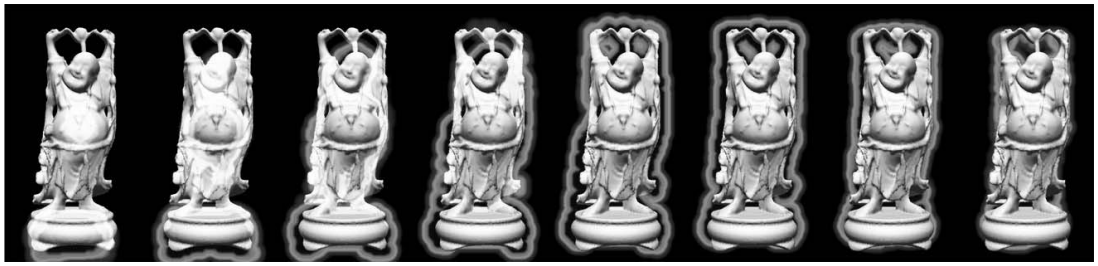

図 5 ポリゴンモデルと距離関数．モデルのポリゴン数は 67,240
Fig. 5 A polygon model with 67,240 polygons and a distance function.

$d(\boldsymbol{x}) = a$ までの領域は，各離散境界から距離 a の球面上に存在している．そのため境界から距離 a 以下の領域において距離関数を求める場合はその球の内部における距離関数を求めればよい．この操作は各スライス上に半径 a の円を描画し，ピクセルシェーダで点と点の距離計算を行えばよい．円を描画するには多くの頂点が必要だが，ここでは 1 辺の長さが $2a$ の点を描画することで各離散境界のあるスライス上での計算は 1 頂点の入力だけで済むようになる．この計算の入力は 2 次元格子上の座標 u，v と離散境界の座標 p_x，p_y，p_z である．まずはバーテックスシェーダで 2 次元座標 u，v を関数 $t(u, v)$ を用いて 3 次元座標 x，y，z に変換し，その座標と離散境界の座標を用いて距離計算を行う．

$$d(\boldsymbol{x}) = \sqrt{(x - p_x)^2 + (y - p_y)^2 + (z - p_z)^2} \tag{13}$$

式 (2) の計算は深度テストもしくはアルファブレンディングを用いて行うことができる．深度テストを用いる場合には $d(\boldsymbol{x})$ を深度値として出力することで最小値を得ることができる．アルファブレンディングでも GL__MIN__EXT を用いることで関数 g の計算を行うことができる．アルファブレンディングを用いると RGBA 各チャネルでの最小値を求めることができる．今までは 2 次元格子状にスライスを 1 枚ずつ敷き詰める方法を述べてきたが，ブレンディングで関数 g の評価をするようにすると 4 枚のスライスを一度に処理することが可能になり，出力データの圧縮を行うことができる．本研究では 1 ピクセルの RGB チャネルに距離ベクトルを出力し，アルファチャネルに距離関数を出力した．

5. 結　　果

本手法を PentiumD 3.2 GHz の CPU，2.0 GB の RAM，GeForce7800GTX を搭載した PC に実装した．GPU のビデオメモリは 256 MB である．プログラムは C++，OpenGL を用いた．シェーダプログラムは Cg を用いて作成した．格子点上の距離関数は 3 次元配列として求まるが，3 次元配列を図示するのは困難であるため，この 3 次元配列を Z 軸に垂直に切った断面での距離関数を値に応じてコンター表示したものを**図 5** に示す．距離関数は 128^3 の 3 次元格子上で求めた．入力としたのは図 5 に示されているブッダのポリゴンモデルである．同様に 4 個のモデルを入力として求めた距離関数とポリゴンモデルを**図 6** に示す．

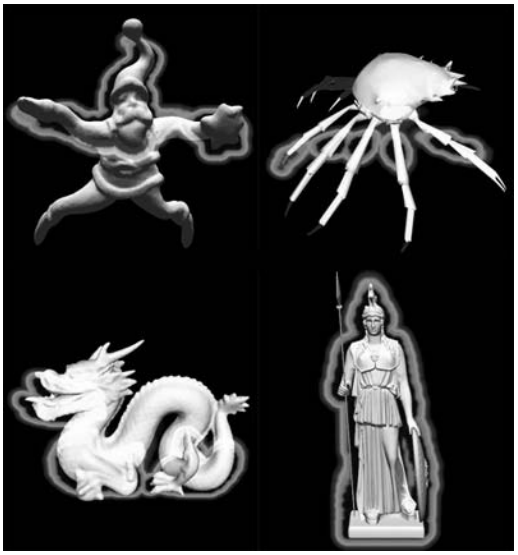

図 6 4 つのポリゴンモデルと距離関数のスライス
Fig. 6 Four polygon models and sliced distance functions.

表 1 128^3 の計算領域における距離関数計算時間の比較（ミリ秒）
Table 1 Comparison of distance function computation in 128^3 resolution grid (in milliseconds).

	Polygons	0.025	0.05	0.1
Buddha(A)	15,536	25.0	34.4	106.2
Buddha(B)	67,240	31.2	39.1	107.8
Santa(A)	37,888	29.7	39.1	104.7
Santa(B)	75,778	35.9	42.2	106.3
Santa(C)	113,668	42.2	50.0	110.9

表 1 に 128^3 の格子解像度において離散境界を構

築し 128^3 の計算格子において距離関数を求めた計算時間を示す．5 種類のポリゴンモデルを用いた．入力のポリゴンモデルの Buddha(A) と Buddha(B) は同じ形状を表したポリゴン数が異なるモデルであり，Santa(A)，Santa(B)，Santa(C) も同様に同じ形状を示したポリゴン数が異なるモデルである．それぞれのモデルに対して，境界から全計算領域の立方体の 1 辺の長さの 2.5，5，10%の距離において距離関数を計算した．つまり $d(\boldsymbol{x}) < ar$ の式を満たす解のみを求めており，r は計算領域の 1 辺の長さであり，a はそれぞれ 0.025，0.05，0.1 である．これらの結果から同じ形状を示したポリゴンモデルを境界として用いた同じ計算範囲での距離関数の計算ならば，ポリゴン数が変化しても計算時間はあまり変化していない．これは本手法での計算コストがモデルの解像度にあまり依存しないことを示している．

表 2 に 128^3 の解像度において離散境界を構築した計算時間の詳細な結果を示す．表 1 で用いた 5 種類のモデルを用いて境界から 5%の距離の計算範囲における計算時間を各段階に分けて測定した．Phase 0 はポリゴンモデルの境界から離散境界の抽出を行う段階であり，Phase 1 は 3 方向から抽出した離散境界を 1 つのデータに統合し，3 次元ボリュームデータから離散境界位置を取り出す段階である．Phase 2 はこれらを用いて距離関数を構築する段階である．表 2 より，表現する形状が同じならば，つまりモデルの表面積が等しければ，Phase 1 と Phase 2 にかかる時間はほぼ同じだということが分かる．すなわち 3 章で述べたように距離関数の構築にかかる時間がモデルの解像度に依存しない処理となっていることが数知的に確認できた．しかしモデルの解像度が異なれば Phase 0 にかかる計算時間は異なる．これはモデルから離散境界を抽出する際に，3 方向からモデルをレンダリングする必要があるため，ポリゴン数と計算時間は線形な関係になるからである．

表 2 境界から 5%の距離内での距離関数計算時間の詳細（ミリ秒）．Phase 0 は離散境界の抽出，Phase 1 はデータの統合，Phase 2 は距離関数の構築

Table 2 Detailed computation time (in milliseconds). Phase 0: the extraction of discrete boundaries, Phase 1: data synthesis, Phase 2: distance function computation.

	Phase 0	Phase 1	Phase 2
Buddha(A)	3.2	14.0	18.8
Buddha(B)	9.4	14.0	18.7
Santa(A)	9.4	15.6	18.7
Santa(B)	15.6	15.7	18.8
Santa(C)	23.5	17.2	18.8

6. 比　　較

本手法の特徴は以下の 3 つである．

(1) ポリゴンモデルのトポロジの制約がない．
(2) 計算時間がポリゴン数に依存しない．
(3) 計算速度を制御できる．

関連研究で述べたように GPU を用いた距離関数の計算手法はこれまで存在するが，本章では提案手法の特徴を既存研究と比較し，有用性の検討を行う．

6.1 トポロジの制約

Sigg らの手法[10] は入力されるデータとして完全に閉じたモデルでなければ適用することができないが，本手法はトポロジに関して制約条件がないため，どのようなモデルでも適用することが可能である（**図 7**，**図 8**）．

6.2 計算時間とポリゴン数

Hoff らの手法[7] はそれぞれのサイトの 2 次元平面上での距離関数の計算を行うときに 3 次元形状を構築している．そのため，その形状を十分細かく分割しなければ正しい距離関数を得ることができないので，誤差の少ない解を求めるためには膨大な量の形状を生成し，座標変換しなければならない．しかし本手法は離散境界の数に比例した数の点を生成するだけで解を求めることが可能であるので，GPU に送らなければならないデータ量も少なくて済む．

Sigg らの手法[10] は Mauch の手法[9] に比べ少ない 3 次元形状の構築で距離関数の計算を行うことができ

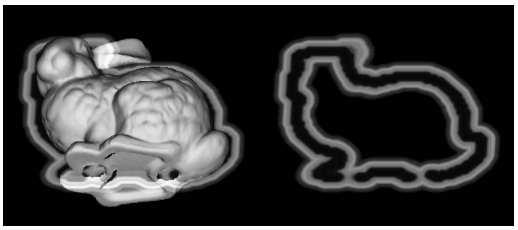

図 7 穴の開いたポリゴンモデルから求めた距離関数
Fig. 7 A distance function calculated from a polygon model with holes.

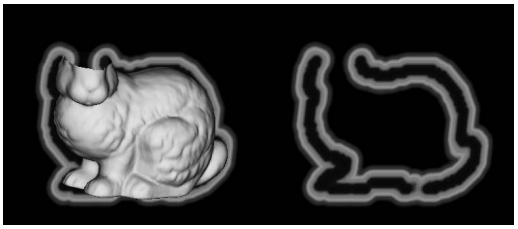

図 8 切断されたポリゴンモデルから求めた距離関数
Fig. 8 A distance function calculated from a cutted polygon model (stanford bunny).

るが，やはり生成される 3 次元形状はポリゴンモデルの面の数に比例している．本手法の計算時間はモデルのポリゴン数にあまり影響を受けないので，本手法との計算時間の差は，モデルの解像度が高くなればなるほど大きくなる．また Sigg らの手法は生成される 3 次元形状内部の非線形分布を持つ距離関数を求めるために複雑なピクセル操作を行わなければならないのに対し，本手法は点と点の距離を求める単純な計算で済むため，ピクセルシェーダの負荷も低い．Sigg らの論文に示されている結果にはポリゴン数と計算時間は線形の関係にあるが，本手法の結果はほぼ一定である．

Sud らの手法[11] は非線形な分布をした距離関数の計算を線形要素の計算で近似することよって単純化し，距離関数計算において生成しなければならない形状も大幅に減らしている．この論文では面の接続情報をもとに高速化を行っている結果のみ提示しているが，面の接続情報を持たないモデルの距離関数の計算時間は大幅に増加すると考えられる．多くの面を持つポリゴンモデルを入力として用いるとこの手法もポリゴン数に比例した計算時間が必要である．論文で述べられているように，高解像度のポリゴンモデルを入力として用いるとそれらの変換がボトルネックとなる．高解像度のポリゴンモデルを用いた場合，本手法の方が効率的に計算できる．

このように既存手法はすべてポリゴン数に比例する計算時間がかかるが，提案手法は境界の次元を落とす変換の後に距離関数の構築を行うため，ポリゴン数が計算時間に及ぼす影響は小さい．そのため特に高解像度のポリゴンモデルを境界として用いた場合に特に有効であると思われる．GPU を用いた境界から一定距離の解を求める手法は Sigg らの CSC method のみであり，この手法と本手法の計算時間の比較を行った結果を**図 9** に示す．256^3 の解像度の計算格子の中で距離関数を計算したものである．図中の Our Method(1), (2) はそれぞれ本手法を用いた計算時間であり離散境界の計算を 256^3 と 128^3 の領域で計算したものである．CSC method を用いた計算時間はモデルのポリゴン数に比例して大きく増加する．しかし本手法での計算時間はポリゴン数が増加してもあまり変化はない．モデルのポリゴン数が変化しても同じ領域内で離散境界を構築した場合は距離関数計算にかかる時間はほぼ一定となることが分かる．

6.3 計算速度の制御

図 9 から本手法は離散境界を抽出する解像度を変えることで，計算時間を制御することが可能であるということも分かる．ただし距離関数を構築する解像度を低くすることで計算時間を短縮することは可能だが，それによって求まる解の精度は落ちる．

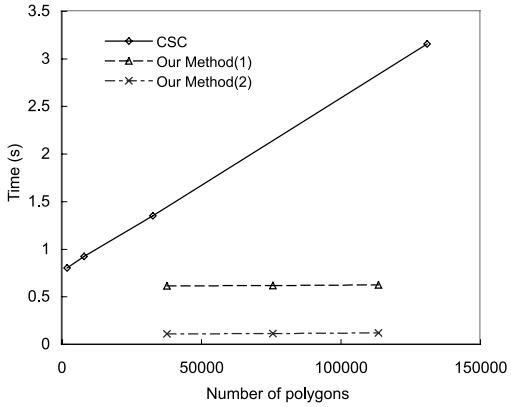

図 9 ポリゴン数と距離関数計算時間．CSC algorithm と本手法の比較．本手法において 256^3 の解像度で離散境界を計算したものが (1) であり，128^3 の解像度で計算したものが (2) である

Fig. 9 Relation between the number of polygons and distance function calculation time. Comparison between the CSC algorithm and our method. Our method (1) is the result with the discrete boundaries calculated in 256^3 resolutions and (2) is that in 128^3 resolutions.

6.4 限　　界

現在の実装では離散境界の構築によって表面の 3 次元ラスタライズを行っているが，これによって求めることのできる離散境界は 3 次元バッファの解像度となってしまう．そのため，1 つのボクセル内部に存在する面はすべてボクセルの中心に位置する離散境界に変換されるため，距離関数の誤差を生む．

7. 結　　論

本論文では距離関数の計算において境界を離散境界に分解してそれらの解の関数として境界から定まる解を求めることができることを示した．これにより複雑な形状の境界を用いた距離関数の計算を，点と点の距離の計算に変換することができ，問題を簡単にすることができた．そしてこの離散境界を用いた距離関数の計算手法を提案した．提案手法を実装した結果，多くのポリゴンを持つモデルを入力として用いても，計算時間が大幅には増加しないことが確認できた．完全に面が閉じていないモデルを用いても閉じたモデルと同様に解を求めることができ，手法の頑強性を示した．距離関数計算の際に用いる離散境界の解像度を変化させることによって計算時間を制御することが可能であることも示した．

提案手法を用いた応用研究は多く考えられる．今

後，提案手法が高速に距離関数を計算することが可能であることを生かし，様々なリアルタイムシミュレーションに応用していく．たとえば本手法を用いることによって高解像度のモデルの変形をともなうリアルタイムシミュレーションが可能になる．また衝突計算においては今までに扱われなかった解像度のポリゴンモデルを用いることも可能になる．

参 考 文 献

1) Borgefors, G.: Distance transformations in arbitrary dimensions, *Computer Vision, Graphics and Image Processing*, Vol.27, pp.321–345 (1984).
2) Danielsson, P.E.: Euclidean distance mapping. *Computer Graphics and Image Processing*, Vol.14, pp.227–248 (1980).
3) Dong, Z., Chen, W., Bao, H., Zhang, H. and Peng, Q.: Real time voxelization for complex polygonal models, *Proc. 12th Pacific Conference on Computer Graphics and Applications*, pp.43–50 (2004).
4) Foskey, M., Lin, C. and Manocha, D.: Efficient computation of a simplified medial axis, *Proc. 8th ACM symposium on Solid modeling and applications*, pp.96–107 (2003).
5) Foster, N. and Fedkiw, R.: Practical animation of liquids, *Proc. Siggraph 01*, pp.15–22 (2001).
6) Fuhrmann, A., Sobottka, G. and Gros, C.: Distance fields for rapid collision detection in physically based modeling, *Proc. Graphicon*, pp.58–65 (2003).
7) Hoff, K.E., Keyser, J., Lin, M., Manocha, D. and Culver, T.: Fast computation of generalized voronoi diagrams using graphics hardware, *Proc. 26th annual conference on Computer graphics and interactive techniques*, pp.277–286 (1999).
8) Jones, M.W., Baerentzen, J.A. and Sramek, M.: 3D distance fields: A survey of techniques and applications, *IEEE Trans. Visualization and Computer Graphics* (2006).
9) Mauch, S.: A fast algorithm for computing the closest point and distance transform, Technical Report, California Institute of Technology (2000).
10) Sigg, C., Peikert, R. and Gross, M.: Signed distance transform using graphics hardware, *Proc. IEEE Visualization*, pp.83–90 (2003).
11) Sud, A., Govindaraju, N., Gayle, R. and Manocha, D.: Interactive 3D distance field computation using linear factorization, *Proc. ACM Symposium on Interactive 3D Graphics and Games* (2006).
12) Sud, A., Otaduy, M.A. and Manocha, D.: Difi: Fast 3D distance field computation using graphics hardware, *Proc. Eurographics*, Vol.23, pp.557–566 (2004).

(平成 18 年 5 月 17 日受付)
(平成 19 年 1 月 9 日採録)

原田 隆宏（正会員）

昭和 56 年生．平成 18 年東京大学大学院工学系研究科システム量子工学専攻修士課程修了．同年東京大学大学院情報学環学際情報学府助手．平成 19 年 4 月助教．計算力学とコンピュータグラフィックスの研究に従事．日本機械学会，ACM 各会員．

越塚 誠一

昭和 37 年生．昭和 61 年東京大学大学院工学系研究科原子力工学専攻修士課程修了．同年東京大学工学部助手．平成 3 年博士（工学）．同年講師．平成 5 年助教授．平成 16 年教授．連続体の力学シミュレーションの研究に従事．特に粒子法の開発を行う．平成 17 年に丸善より『粒子法』を出版．平成 18 年に日本学術振興会賞を受賞．日本原子力学会，日本機械学会，日本流体力学会，日本計算工学会，日本応用数理学会各会員．