コンピュータグラフィクス論

– モデリング (3) –

2018年5月10日

高山 健志

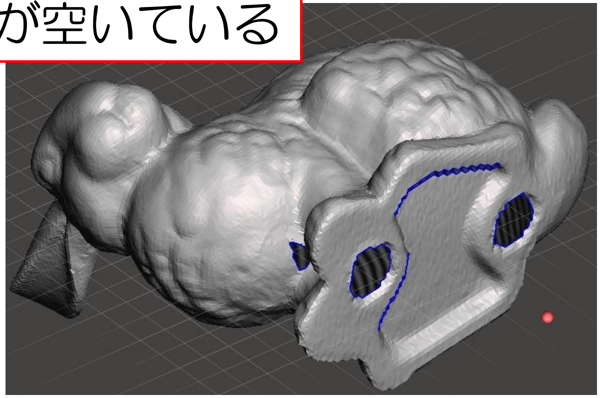
ソリッドモデリング

ソリッドモデルとは

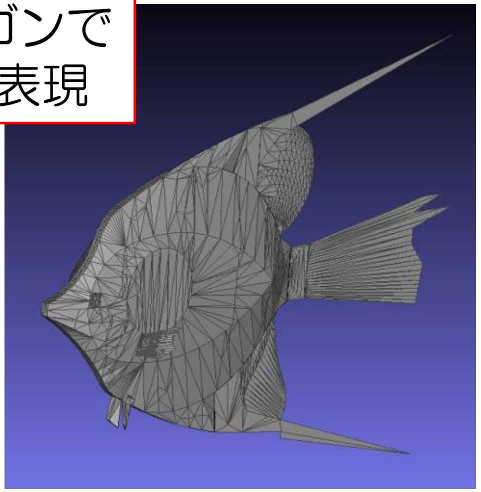
- 3D 空間の任意の位置で、モデルの “内側” と “外側” が定義できるもの

ソリッドでないケース

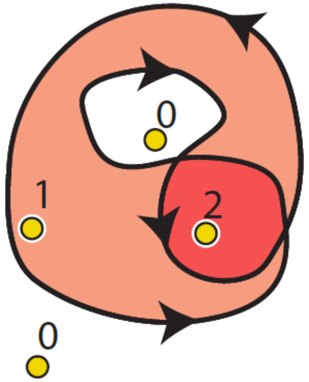
穴が空いている



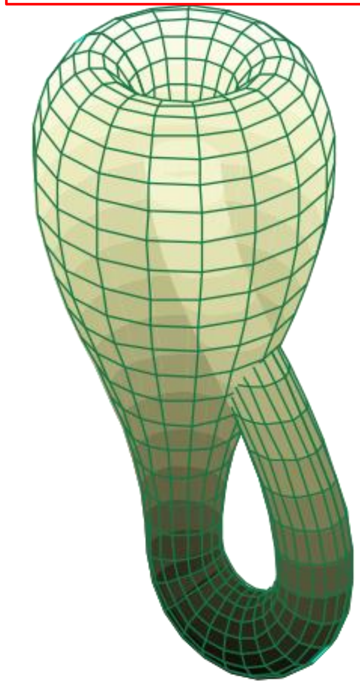
一枚のポリゴンで薄い形状を表現



自己交差している



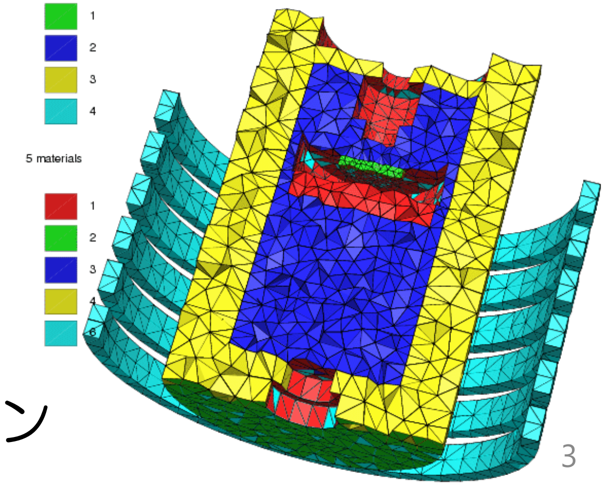
向き付け不可能



Klein bottle

- 主な用途

3D プリント



物理シミュレーション

ソリッドモデルの predicate 関数

- 3D 座標 $\mathbf{p} \in \mathbb{R}^3$ がソリッドモデルの内部であれば true を、そうでなければ false を返す関数：

$$f(\mathbf{p}): \mathbb{R}^3 \mapsto \{\, \text{true}, \text{false} \,\}$$

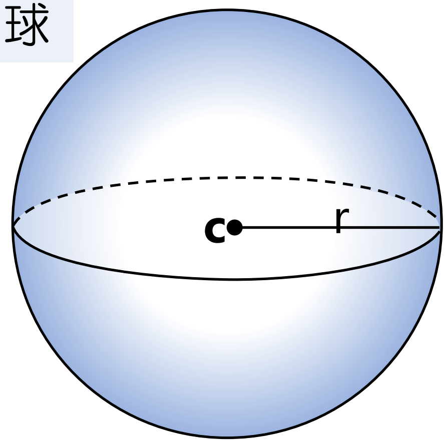
- モデル内部全体を表す集合：

$$\{\, \mathbf{p} \mid f(\mathbf{p}) = \text{true} \,\} \subset \mathbb{R}^3$$

- 例：

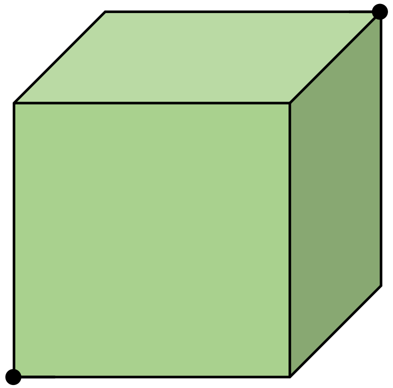
点 $\mathbf{c}$ を中心とした半径 r の球

$$f(\mathbf{p}) \coloneqq \|\mathbf{p} - \mathbf{c}\| < r$$



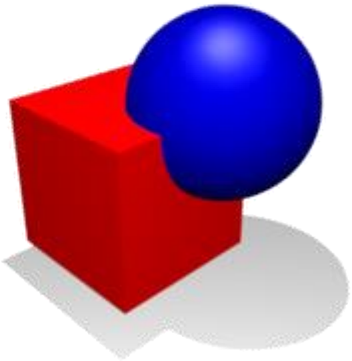
最小と最大の対角コーナーがそれぞれ $(x_{\text{min}}, y_{\text{min}}, z_{\text{min}})$ と $(x_{\text{max}}, y_{\text{max}}, z_{\text{max}})$ であるような直方体

$$\begin{aligned} f(x, y, z) \coloneqq\ & (x_{\text{min}} < x < x_{\text{max}}) \\ & \wedge (y_{\text{min}} < y < y_{\text{max}}) \\ & \wedge (z_{\text{min}} < z < z_{\text{max}}) \end{aligned}$$



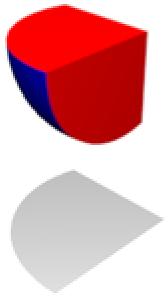
Constructive **S**olid **G**eometry (Boolean演算)

和



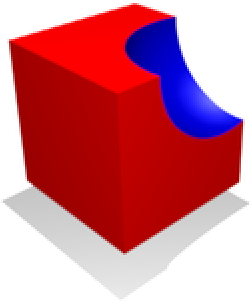
$$f_{A\cup B}(\mathbf{p}) \coloneqq f_A(\mathbf{p}) \lor f_B(\mathbf{p})$$

積



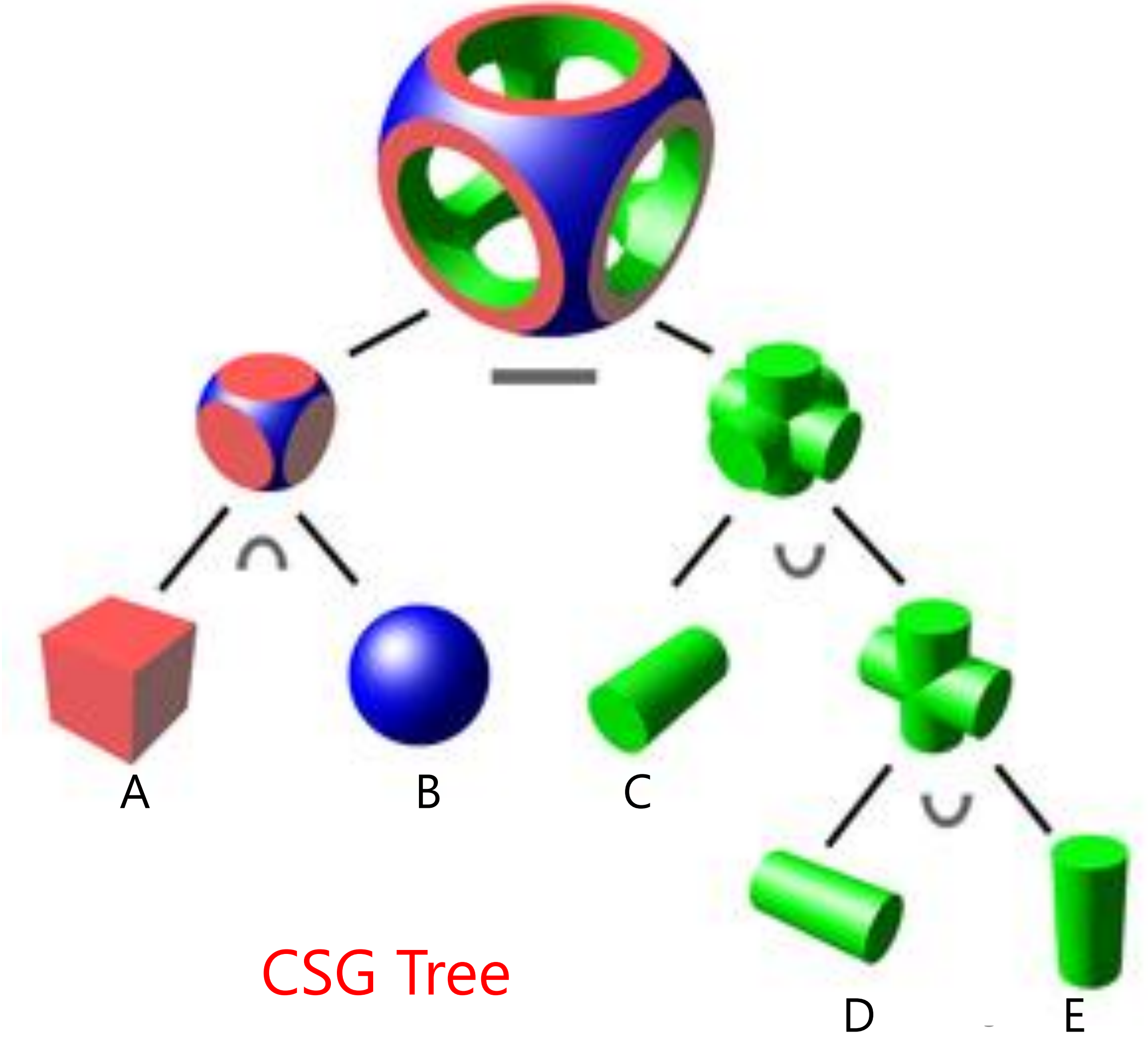
$$f_{A\cap B}(\mathbf{p}) \coloneqq f_A(\mathbf{p}) \land f_B(\mathbf{p})$$

差

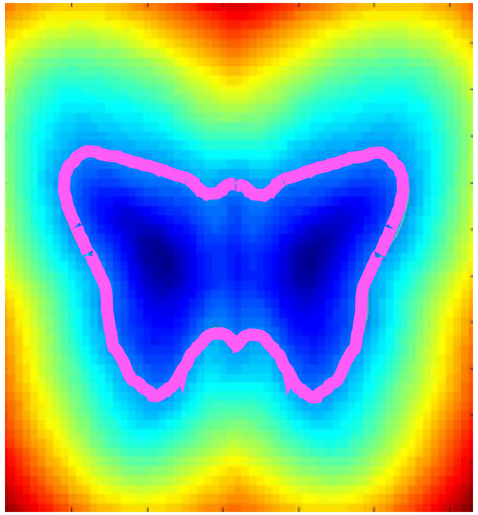


$$f_{A\setminus B}(\mathbf{p}) \coloneqq f_A(\mathbf{p}) \land \neg f_B(\mathbf{p})$$

$$(A \cap B) \setminus (C \cup (D \cup E))$$



符号付き距離場によるソリッドモデル表現



- **S**igned **D**istance **F**unction: 3D座標からモデル表面までの最短距離

$$d(\mathbf{p}): \mathbb{R}^3 \mapsto \mathbb{R}$$

  - 符号付き：内側では負、外側では正

- ソリッドモデルを表すpredicate:

$$f(\mathbf{p}) \coloneqq d(\mathbf{p}) < 0$$

- ゼロ等値面はモデル表面を表す：

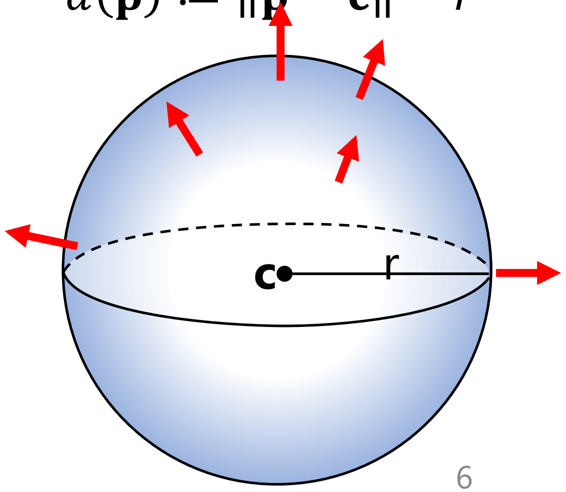
$$\{\mathbf{p} \mid d(\mathbf{p}) = 0\} \subset \mathbb{R}^3$$

- 「陰関数表現」「ボリューム表現」

- 等値面の法線は勾配 $\nabla d(\mathbf{p})$ の方向と一致

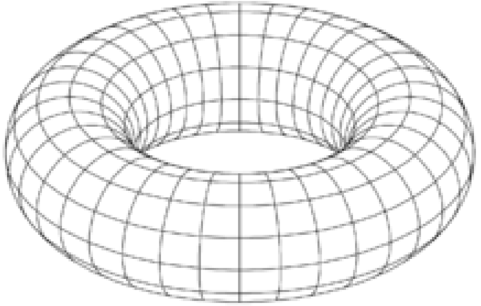
点 $\mathbf{c}$ を中心とした半径 r の球

$$d(\mathbf{p}) \coloneqq \|\mathbf{p} - \mathbf{c}\| - r$$



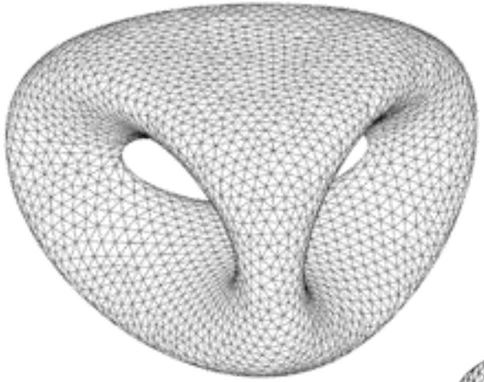
陰関数のデザイン例

必ずしも距離関数とは限らない

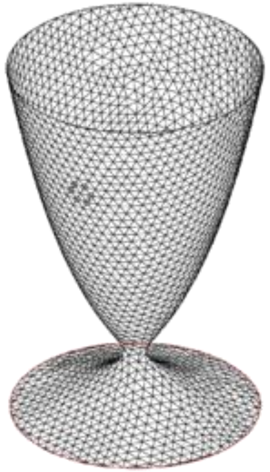


大半径 R, 小半径 a のトーラス

$$(x^2 + y^2 + z^2 + R^2 - a^2)^2 - 4R^2(x^2 + y^2) = 0$$



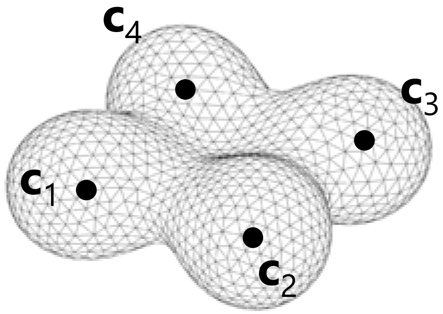
$$2y(y^2 - 3x^2)(1 - z^2) + (x^2 + y^2)^2 - (9z^2 - 1)(1 - z^2) = 0$$



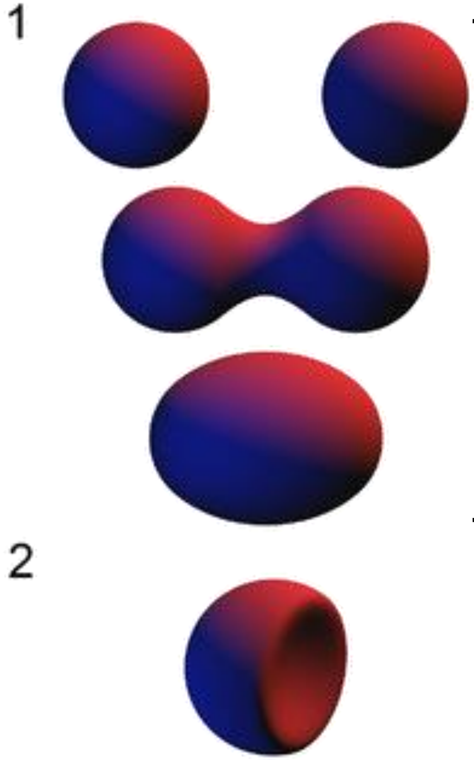
$$x^2 + y^2 - (\ln(z + 3.2))^2 - 0.02 = 0$$

陰関数のデザイン例：等電位面 (Metaball)

$$d_i(\mathbf{p}) = \frac{q_i}{\|\mathbf{p} - \mathbf{c}_i\|} - r_i$$



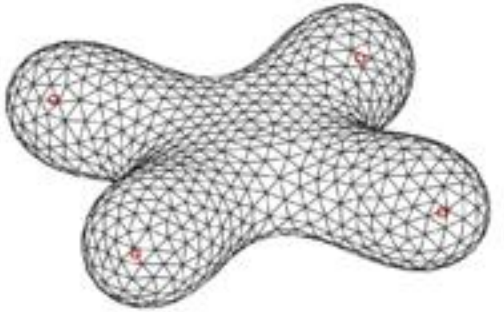
$$d(\mathbf{p}) = d_1(\mathbf{p}) + d_2(\mathbf{p}) + d_3(\mathbf{p}) + d_4(\mathbf{p})$$



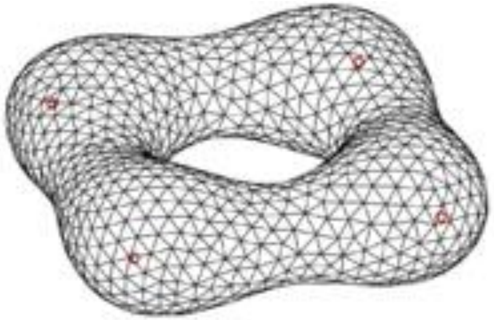
$$d(\mathbf{p}) = d_1(\mathbf{p}) + d_2(\mathbf{p})$$

$$d(\mathbf{p}) = d_1(\mathbf{p}) - d_2(\mathbf{p})$$

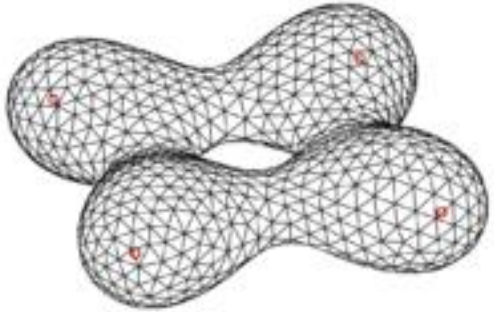
陰関数の線形補間によるモーフィング



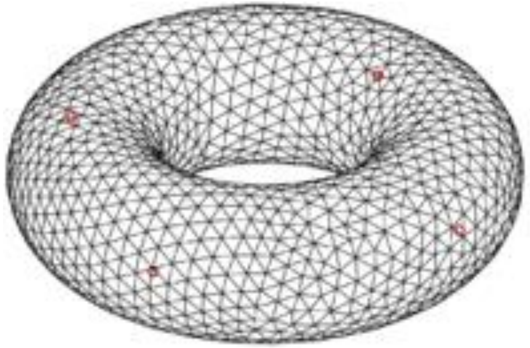
$d_1(\boldsymbol{p}) = 0$



$\frac{1}{3} d_1(\boldsymbol{p}) + \frac{2}{3} d_2(\boldsymbol{p}) = 0$



$\frac{2}{3} d_1(\boldsymbol{p}) + \frac{1}{3} d_2(\boldsymbol{p}) = 0$



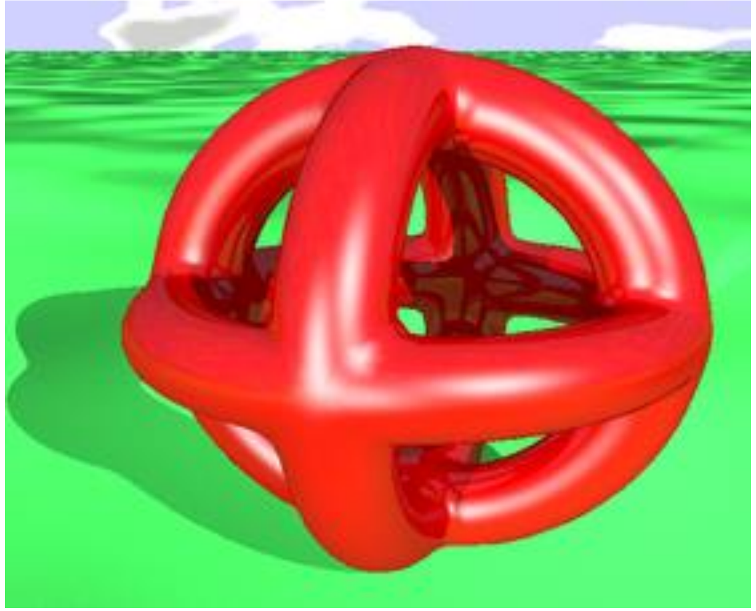
$d_2(\boldsymbol{p}) = 0$

複数の陰関数を組み合わせたモデリング

$$F_1 = (x^2 + y^2 + z^2 + R^2 - a^2)^2 - 4R^2(x^2 + y^2) = 0$$

$$F_2 = (x^2 + y^2 + z^2 + R^2 - a^2)^2 - 4R^2(x^2 + z^2) = 0$$

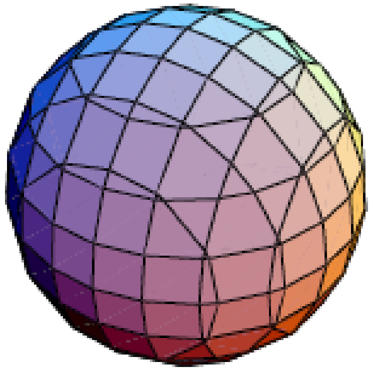
$$F_3 = (x^2 + y^2 + z^2 + R^2 - a^2)^2 - 4R^2(y^2 + z^2) = 0$$

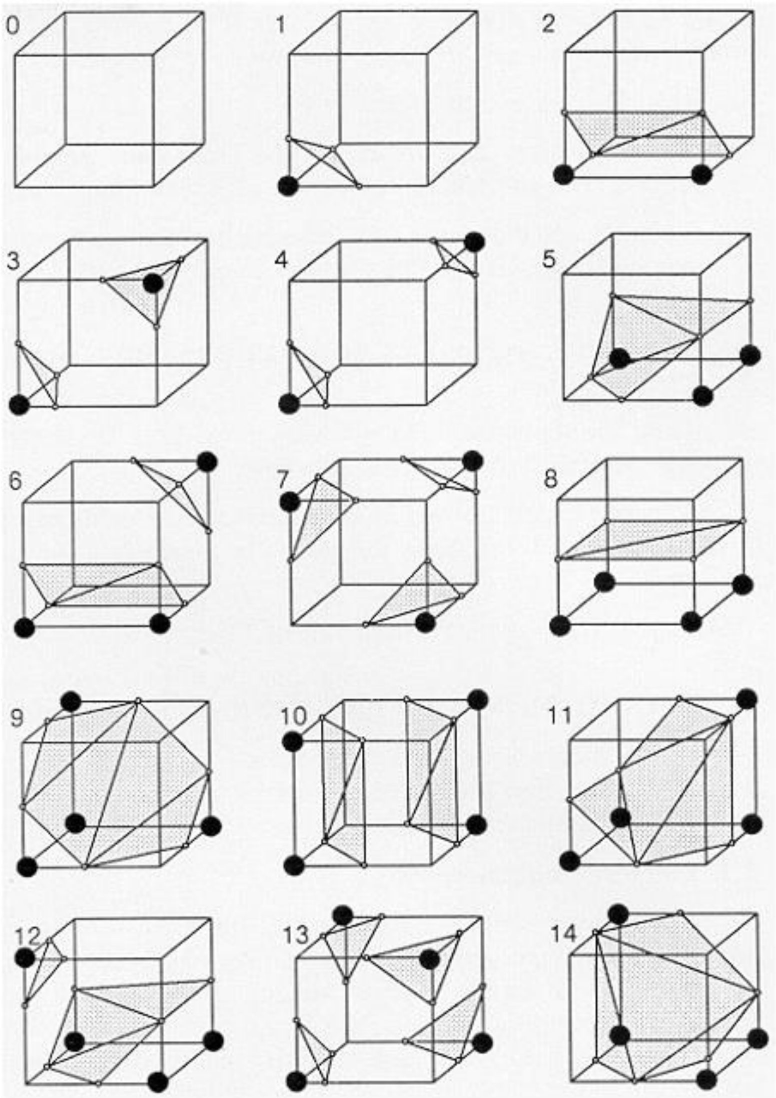


$$F(x, y, z) = F_1(x, y, z) \cdot F_2(x, y, z) \cdot F_3(x, y, z) - c = 0$$

陰関数の表示方法：Marching Cubes

- 等値面を三角形メッシュとして抽出
- 立方体格子の各セルに対し、
  (1) 立方体の 8 頂点で関数値を計算
  (2) その正負のパターンから、
  生成する面のタイプを決定
    - 対称性から 15 通りに分類
  (3) 関数値の線形補間から
  面の位置を決定





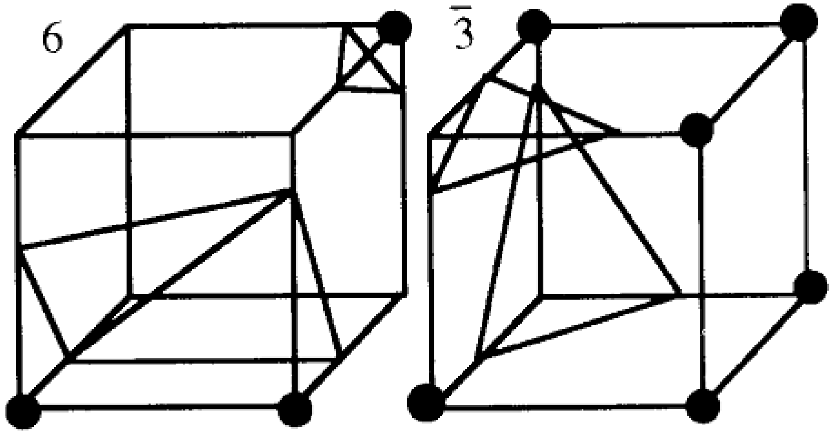
(特許で縛られていたが、期限が切れた)

Marching Cubes: A High Resolution 3D Surface Construction Algorithm [Lorensen SIGGRAPH87]

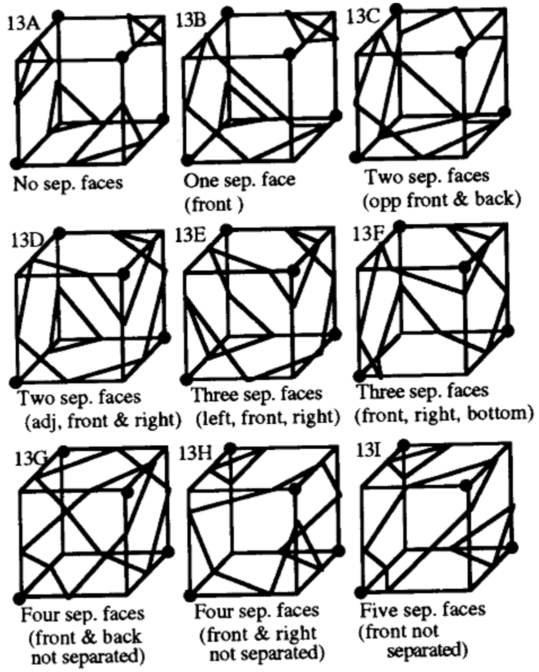
陰関数の性質を活用した3Dモデリングの例

Shapeshop; Sketch-based solid modeling with blobtrees [Schmidt,Wyvill,Sousa,Jorge,SBIM05]

Marching Cubes の曖昧性



隣接するセルの間で面が整合しない

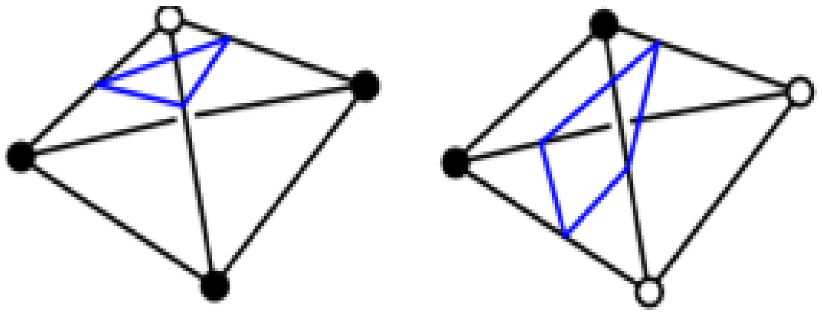


不整合を解決する新たなルール

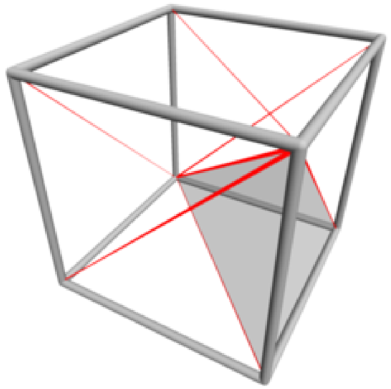
- 根本的な解決法は、格子を細かくすること

The asymptotic decider: resolving the ambiguity in marching cubes [Nielson VIS91]

Marching Tetrahedra

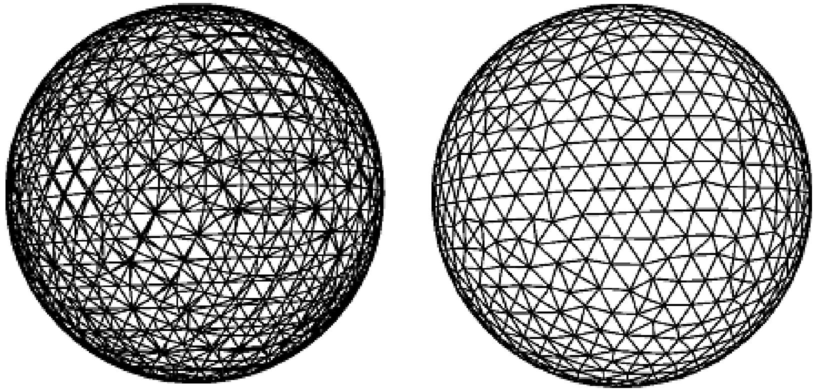


- 立方体の代わりに四面体を使う
  - パターンが少なく、曖昧性が無い
    ➔ 実装が簡単



- 各立方体セルを、6 個の四面体に分割
  - (隣接セル間で分割の向きを合わせることに注意)

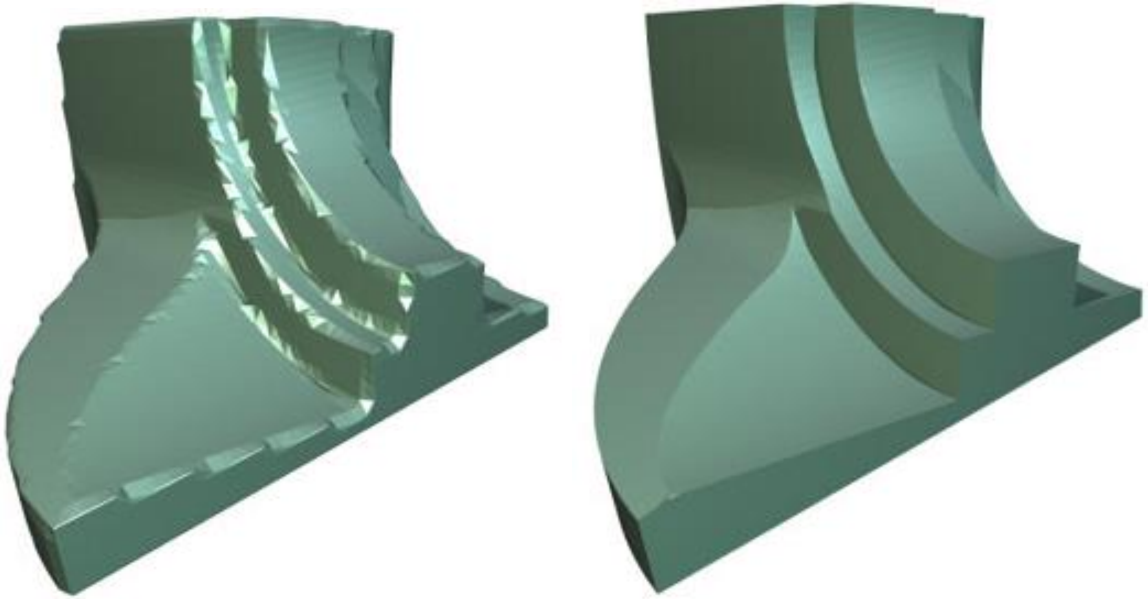
- きれいな三角形メッシュを取り出す工夫



http://paulbourke.net/geometry/polygonise/
Regularised marching tetrahedra: improved iso-surface extraction [Treece C&G99]

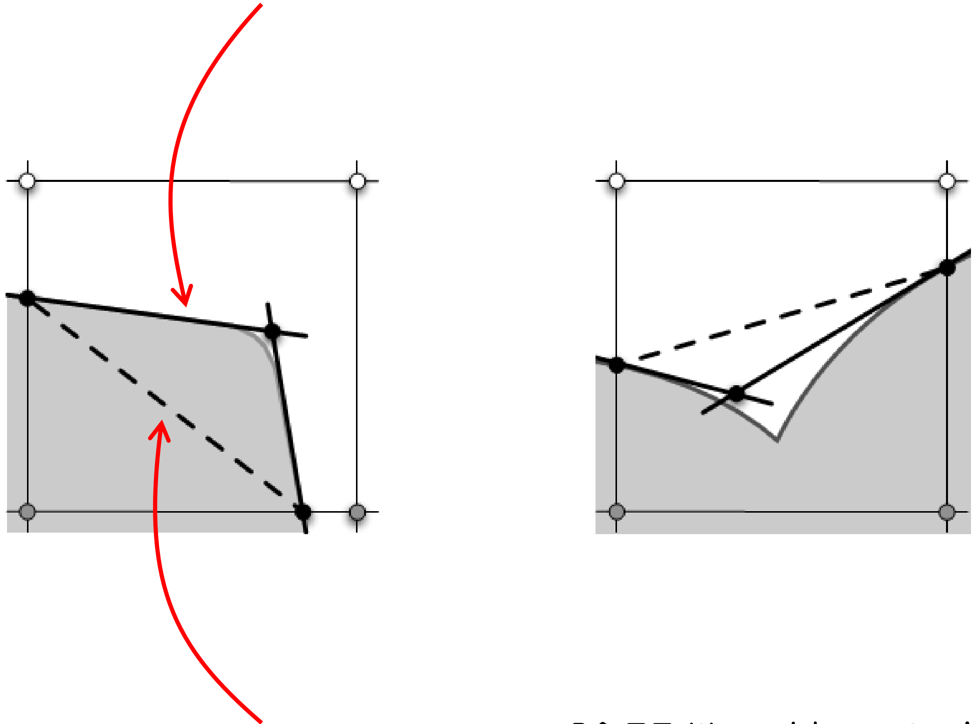
シャープなエッジを保持した等値面抽出

格子サイズ：65×65×65



Marching Cubes　　改善版

改善版 (陰関数の勾配も考慮)

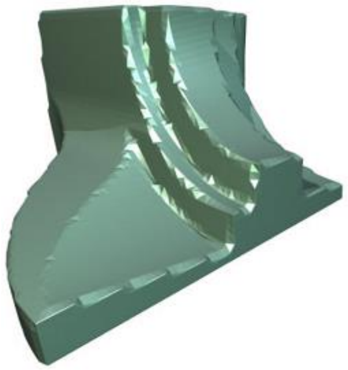


Marching Cubes (陰関数の値のみ考慮)

Feature Sensitive Surface Extraction from Volume Data [Kobbelt SIGGRAPH01]
Dual Contouring of Hermite Data [Ju SIGGRAPH02]
http://www.graphics.rwth-aachen.de/IsoEx/

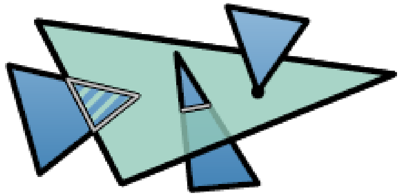
サーフェスメッシュ表現のみに基づく CSG

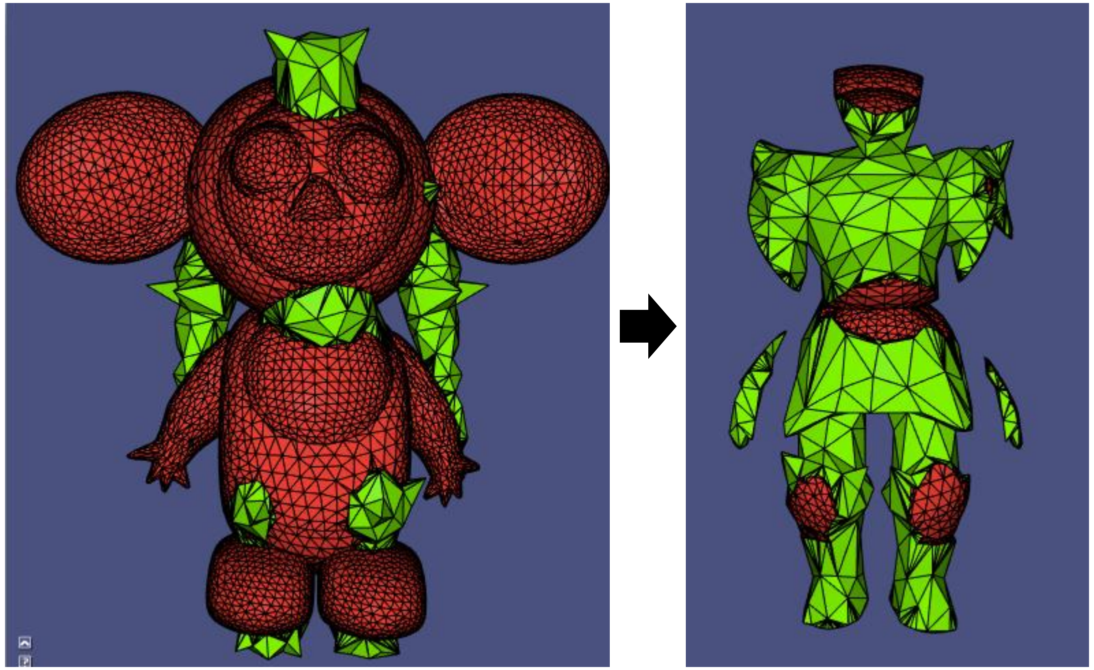
- ボリューム表現 (=Marching Cubesによる等値面抽出)
  ➔ 近似精度が格子の向きや解像度に依存 ☹



- サーフェスメッシュ表現による CSG
  ➔ 元のメッシュの形状を確実に保持 ☺

- ロバストで効率的な実装が難しい ☹
  - 浮動小数の丸め誤差
  - 厳密に同じ位置で重複する複数の三角形



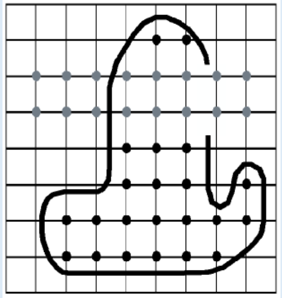


- ここ数年で著しく進化

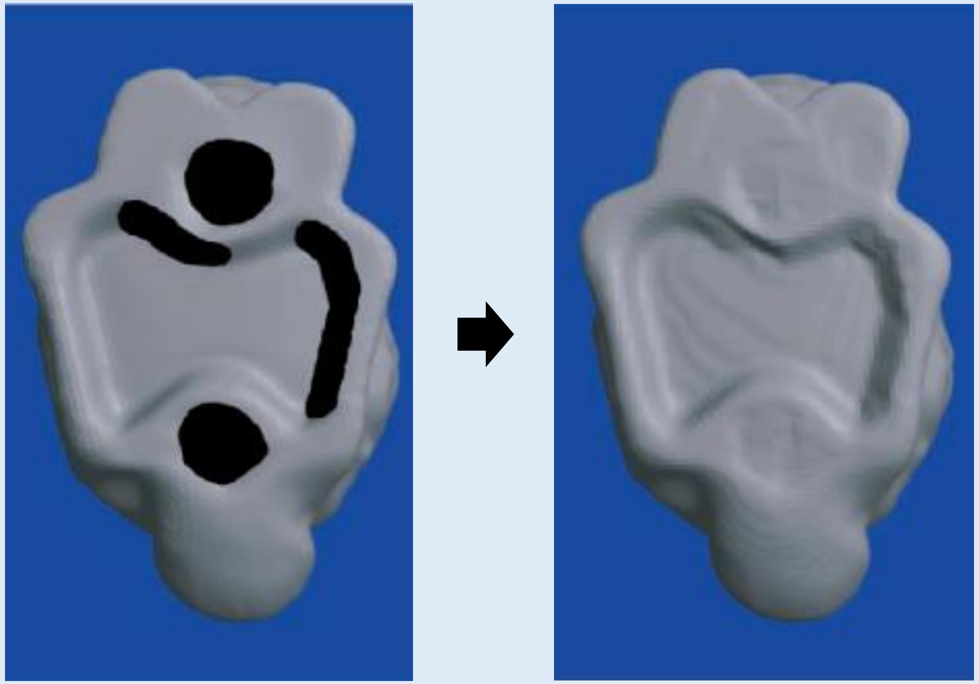
Fast, exact, linear booleans [Bernstein SGP09]
Exact and Robust (Self-)Intersections for Polygonal Meshes [Campen EG10]
Mesh Arrangements for Solid Geometry [Zhou SIGGRAPH16]
https://libigl.github.io/libigl/tutorial/tutorial.html#booleanoperationsonmeshes

メッシュの補修 (mesh repair)

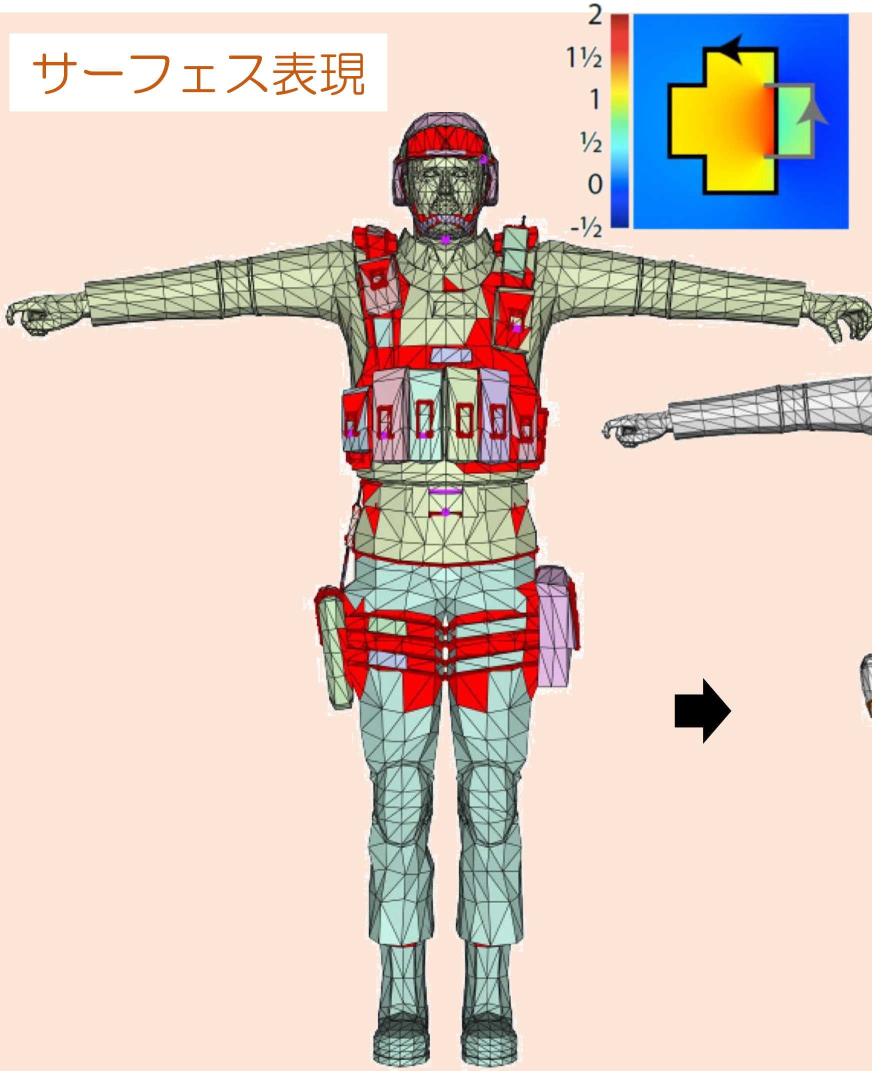
ボリューム表現



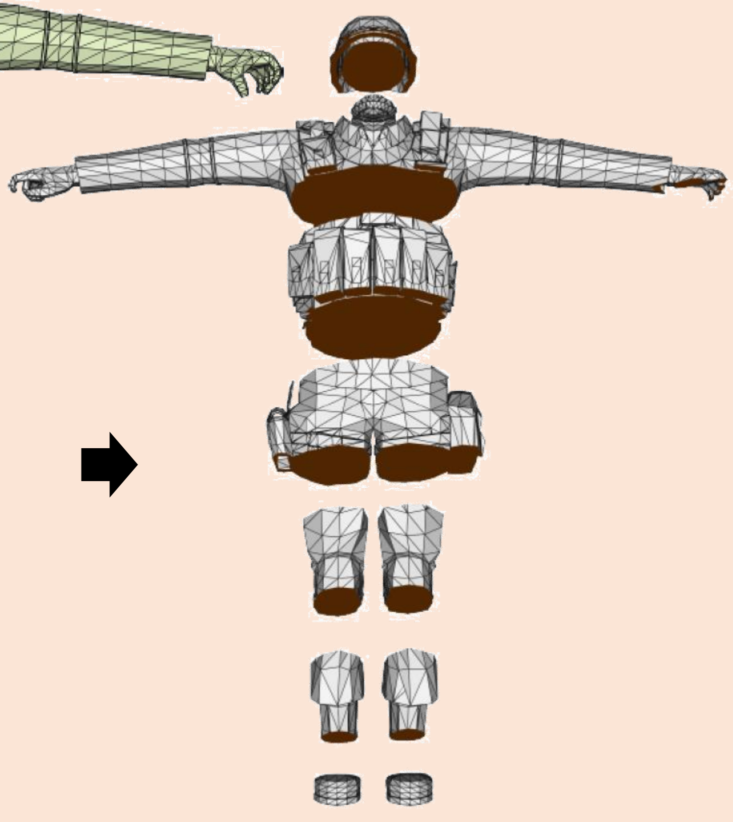
四方八方から飛ばしたレイとの
交差に基づいて内外を判定



サーフェス表現



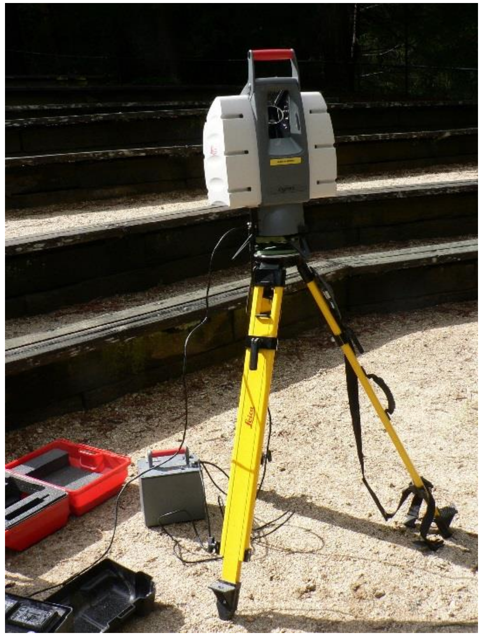
定義を拡張した
winding number に
基づいて内外を判定



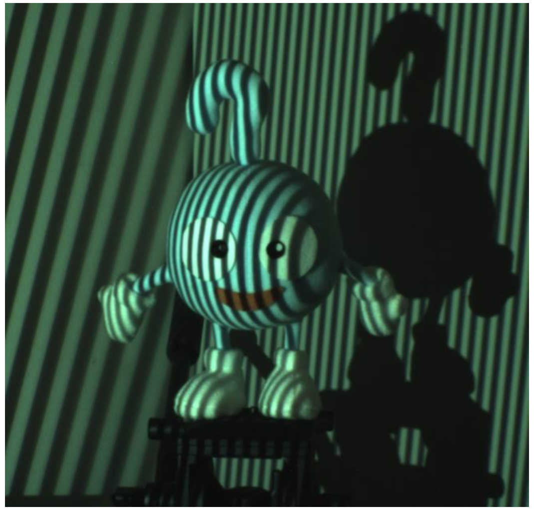
Simplification and Repair of Polygonal Models Using Volumetric Techniques [Nooruddin TVCG03]
Robust Inside-Outside Segmentation using Generalized Winding Numbers [Jacobson SIGGRAPH13]

点群からのサーフェス再構成

3D 形状の計測

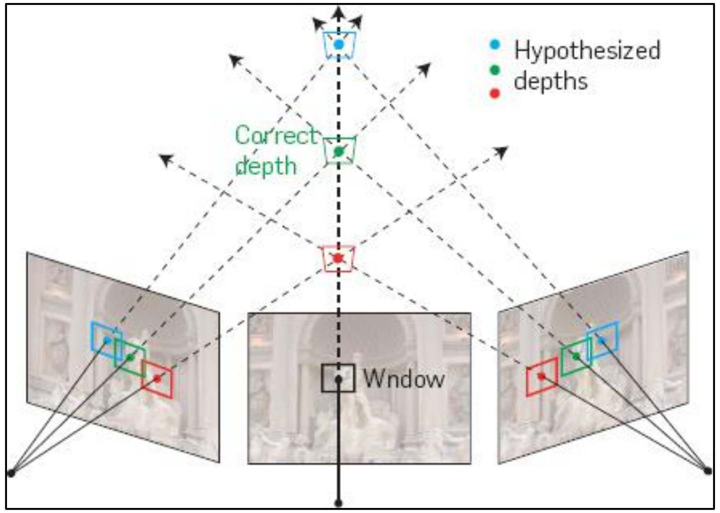


Range Scanner
(LIDAR)



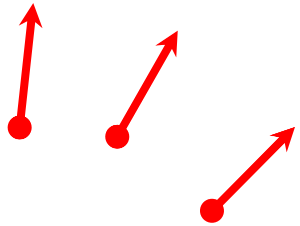
Structured Light

Depth Camera



Multi-View Stereo

- 得られるデータ：点群
  - 3D座標
  - 法線 (面の向き)



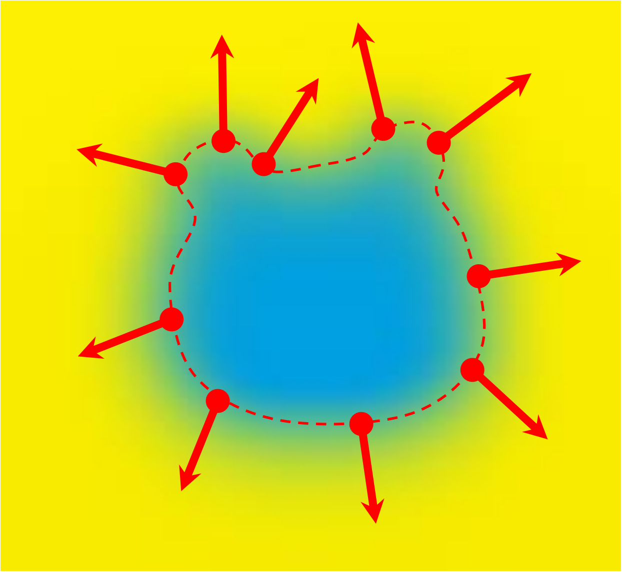
- 法線が得られない場合 ➔ 法線の推定
- ノイズが多すぎる場合 ➔ ノイズの除去

} Computer Vision の代表的なテーマ

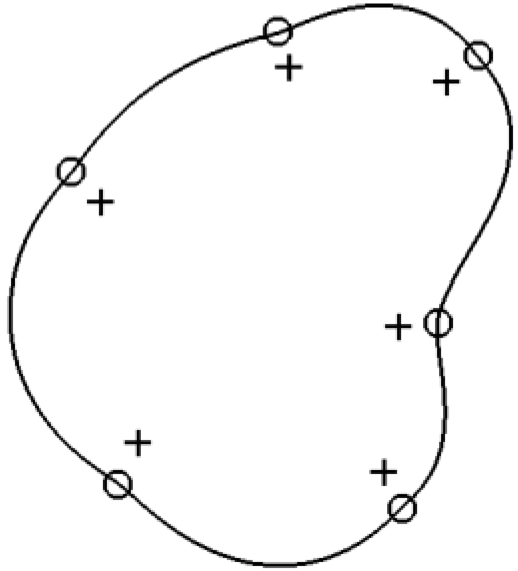
点群からのサーフェス形状再構成

- 入力：N 個の点群データ
  - 座標 $\mathbf{x}_i = (x_i, y_i, z_i)$ と法線 $\mathbf{n}_i = \left(n_i^{\mathrm{x}}, n_i^{\mathrm{y}}, n_i^{\mathrm{z}}\right)$, $i \in \{1, \ldots, N\}$
- 出力：関数 $f(\mathbf{x})$ で、値と勾配の制約を満たすもの
  - $f(\mathbf{x}_i) = f_i$
  - $\nabla f(\mathbf{x}_i) = \mathbf{n}_i$
  - 等値面 $f(\mathbf{x}) = 0$ が出力サーフェス形状
- “Scattered Data Interpolation” と呼ばれる問題
  - **M**oving **L**east **S**quares
  - **R**adial **B**asis **F**unction

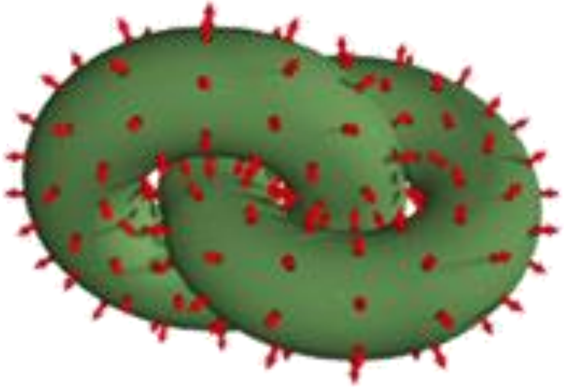
CG以外の分野 (e.g. 機械学習) でも重要



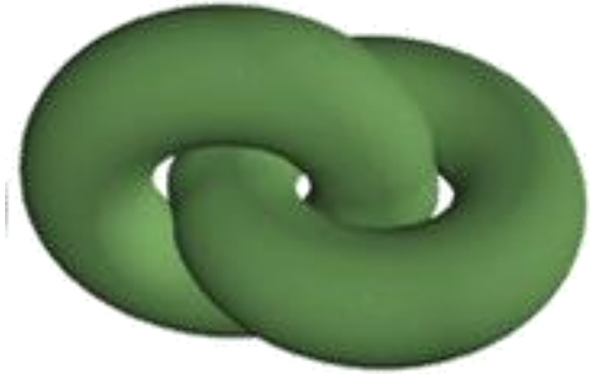
勾配を制約する二通りの方法



- 法線方向にオフセットした位置に値の制約を追加
  - 簡単
- 数学表現そのものに勾配制約を取り入れる (エルミート補間)
  - 高品質



値と勾配の制約



エルミート補間

オフセット法

Modelling with implicit surfaces that interpolate [Turk TOG02]
Hermite Radial Basis Functions Implicits [Macedo CGF10]

Moving **L**east **S**quares による補間 (移動最小二乗)

出発点：**Least SQuares** (最小二乗)

- 求めたい関数が線形だと仮定する：$f(\mathbf{x}) = ax + by + cz + d$
  - a, b, c, d が未知係数

$\mathbf{x} := (x, y, z)$

- データ点における値の制約

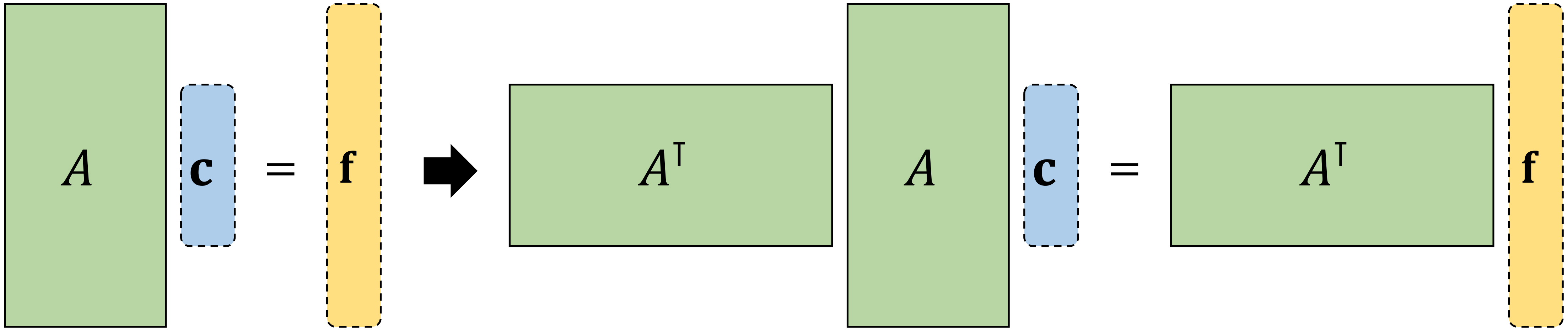
$$
\begin{aligned}
f(\mathbf{x}_1) &= ax_1 + by_1 + cz_1 + d = f_1 \\
f(\mathbf{x}_2) &= ax_2 + by_2 + cz_2 + d = f_2 \\
&\vdots \\
f(\mathbf{x}_N) &= ax_N + by_N + cz_N + d = f_N
\end{aligned}
$$

$$
\underbrace{\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ & \vdots & & \\ x_N & y_N & z_N & 1 \end{bmatrix}}_{A}
\underbrace{\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}}_{\mathbf{c}}
=
\underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}}_{\mathbf{f}}
$$

- (勾配制約は今は考えない)

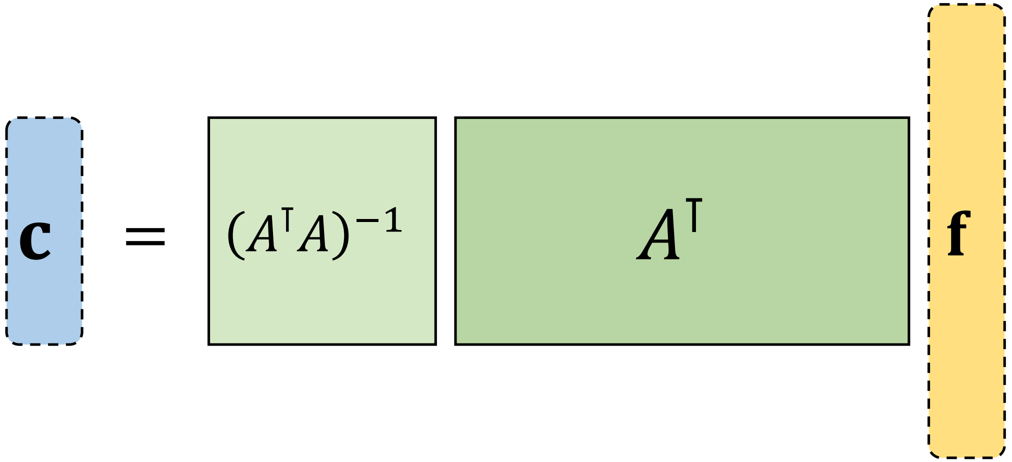
Overconstrained System

- #未知数 < #制約 (i.e. 縦長の行列) ➔ 全ての制約を同時に満たせない



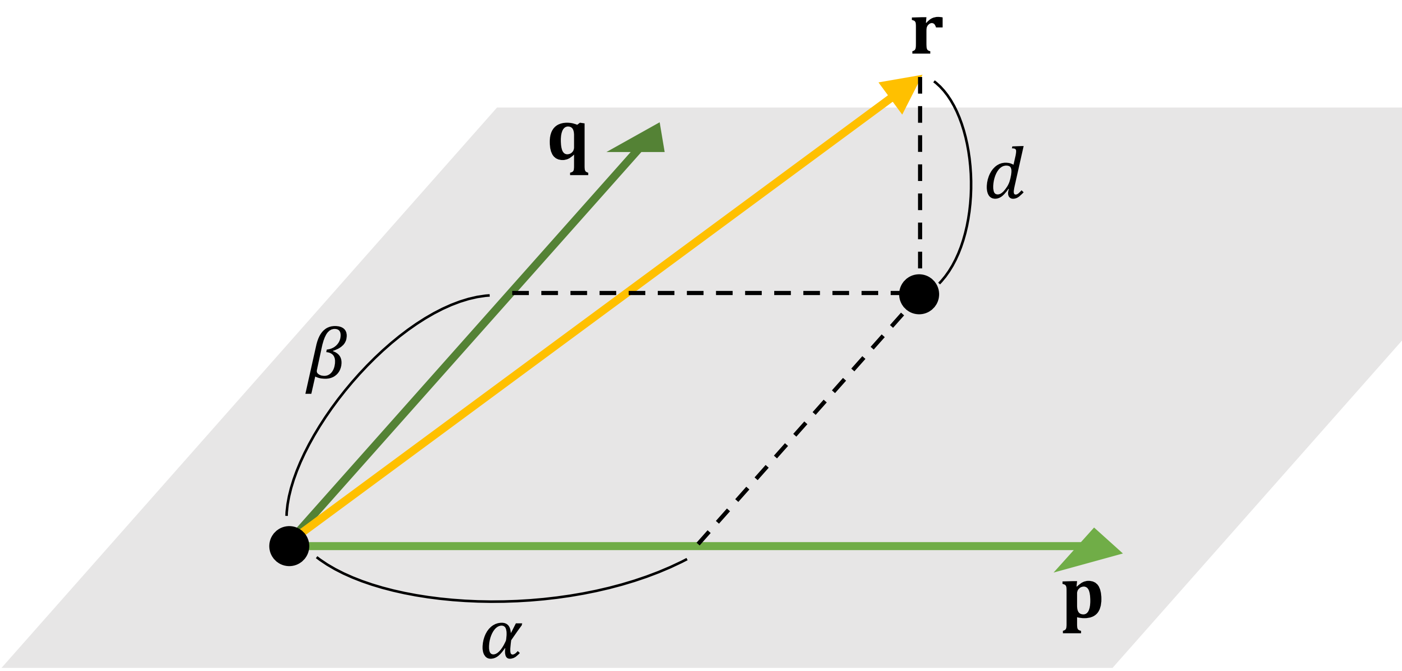
- fitting の誤差を最小化：

$$\|\boldsymbol{A}\,\mathbf{c}-\mathbf{f}\|^2=\sum_{i=1}^{N}\|f(\mathbf{x}_i)-f_i\|^2$$



LSQ の幾何的な解釈

$$\begin{bmatrix} p_{\mathrm{x}} & q_{\mathrm{x}} \\ p_{\mathrm{y}} & q_{\mathrm{y}} \\ p_{\mathrm{z}} & q_{\mathrm{z}} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} r_{\mathrm{x}} \\ r_{\mathrm{y}} \\ r_{\mathrm{z}} \end{bmatrix}$$



- $\mathbf{p}$ と $\mathbf{q}$ が張る空間中で $\mathbf{r}$ に最も近い点を求める (投影する) ことに相当
  - fitting 誤差は投影距離に相当：

$$d^2 = \|\alpha\mathbf{p} + \beta\mathbf{q} - \mathbf{r}\|^2$$

Weighted Least Squares (重み付き最小二乗)

- 各データ点ごとの誤差に、重み w_i をつける
  - 重要度、確信度

- 以下の誤差を最小化：

$$\sum_{i=1}^{N} \| w_i (f(\mathbf{x}_i) - f_i) \|^2$$

$$\underbrace{\begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_N \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ & \vdots & & \\ x_N & y_N & z_N & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}}_{\mathbf{c}} = \underbrace{\begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_N \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}}_{\mathbf{f}}$$

Weighted **L**east **S**quares (重み付き最小二乗)

$$W A \mathbf{c} = W \mathbf{f}$$

$$\Rightarrow \quad \mathbf{c} = (A^\intercal W^2 A)^{-1} A^\intercal W^2 \mathbf{f}$$

Moving Least Squares (移動最小二乗)

- 重み w_i が、評価位置 $\mathbf{x}$ に依存：

$$w_i(\mathbf{x}) = w(\|\mathbf{x} - \mathbf{x}_i\|)$$

- よく使われる関数 (Kernel)：
  - $w(r) = e^{-r^2/\sigma^2}$
  - $w(r) = \frac{1}{r^2+\epsilon^2}$

評価位置に近いほど
大きな重み

- 重み行列 W が $\mathbf{x}$ に依存
  ➔ 係数 a, b, c, d が $\mathbf{x}$ に依存

$$\begin{bmatrix} a(\mathbf{x}) \\ b(\mathbf{x}) \\ c(\mathbf{x}) \\ d(\mathbf{x}) \end{bmatrix}$$

$$f(\mathbf{x}) = [\, x \ \ y \ \ z \ \ 1 \,] \, (A^\mathsf{T} W(\mathbf{x})^2 A)^{-1} \; A^\mathsf{T} W(\mathbf{x})^2 \; \mathbf{f}$$

- 評価点ごとに
  方程式を解き直す

法線制約の導入

- 各データ点が表す 1 次式を考える：

$$g_i(\mathbf{x}) = f_i + (\mathbf{x} - \mathbf{x}_i)^\top \mathbf{n}_i$$

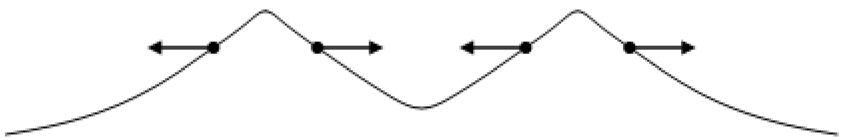
- 各 g_i を現在位置で評価したときの誤差を最小化：

$$\sum_{i=1}^{N} \left\| w_i(\mathbf{x}) \big(f(\mathbf{x}) - g_i(\mathbf{x}) \big) \right\|^2$$

$$\begin{bmatrix} w_1(\mathbf{x}) & & & \\ & w_2(\mathbf{x}) & & \\ & & \ddots & \\ & & & w_N(\mathbf{x}) \end{bmatrix} \begin{bmatrix} x & y & z & 1 \\ x & y & z & 1 \\ & \vdots & & \\ x & y & z & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} w_1(\mathbf{x}) & & & \\ & w_2(\mathbf{x}) & & \\ & & \ddots & \\ & & & w_N(\mathbf{x}) \end{bmatrix} \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_N(\mathbf{x}) \end{bmatrix}$$

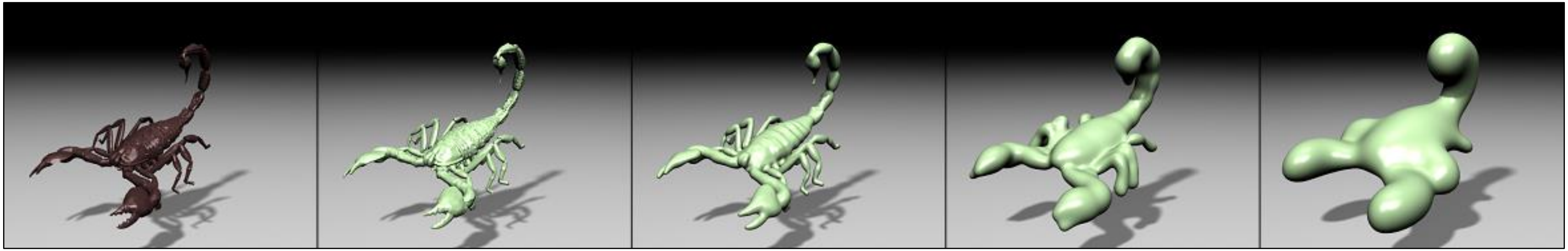
Interpolating and Approximating Implicit Surfaces from Polygon Soup [Shen SIGGRAPH04]

法線制約の導入



法線制約を利用

法線方向にオフセットして値を制約



Input : Polygon Soup　Interpolation　Approximation 1　Approximation 2　Approximation 3

Interpolating and Approximating Implicit Surfaces from Polygon Soup [Shen SIGGRAPH04]

Radial **B**asis **F**unction による補間
(放射基底関数)

基本的な考え方

- 関数 $f(\mathbf{x})$ を、基底関数 $\phi(\mathbf{x})$ の重み付き和として定義：

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(\mathbf{x} - \mathbf{x}_i)$$

未知数

基底関数をデータ位置 $\mathbf{x}_i$ に平行移動

- 放射基底関数 $\phi(\mathbf{x})$：$\mathbf{x}$ の長さのみに依存
  - $\phi(\mathbf{x}) = e^{-\|\mathbf{x}\|^2/\sigma^2}$ (Gaussian)
  - $\phi(\mathbf{x}) = \frac{1}{\sqrt{\|\mathbf{x}\|^2 + c^2}}$ (Inverse Multiquadric)

- 各データ点における制約 $f(\mathbf{x}_i) = f_i$ から、重み係数 w_i を求める

基本的な考え方

$\phi_{i,j} = \phi\left(\mathbf{x}_i - \mathbf{x}_j\right)$ と表記する

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(\mathbf{x} - \mathbf{x}_i)$$

$$f(\mathbf{x}_1) = w_1\phi_{1,1} + w_2\phi_{1,2} + \cdots + w_N\phi_{1,N} = f_1$$

$$f(\mathbf{x}_2) = w_1\phi_{2,1} + w_2\phi_{2,2} + \cdots + w_N\phi_{2,N} = f_2$$

$$\vdots$$

$$f(\mathbf{x}_N) = w_1\phi_{N,1} + w_2\phi_{N,2} + \cdots + w_N\phi_{N,N} = f_N$$

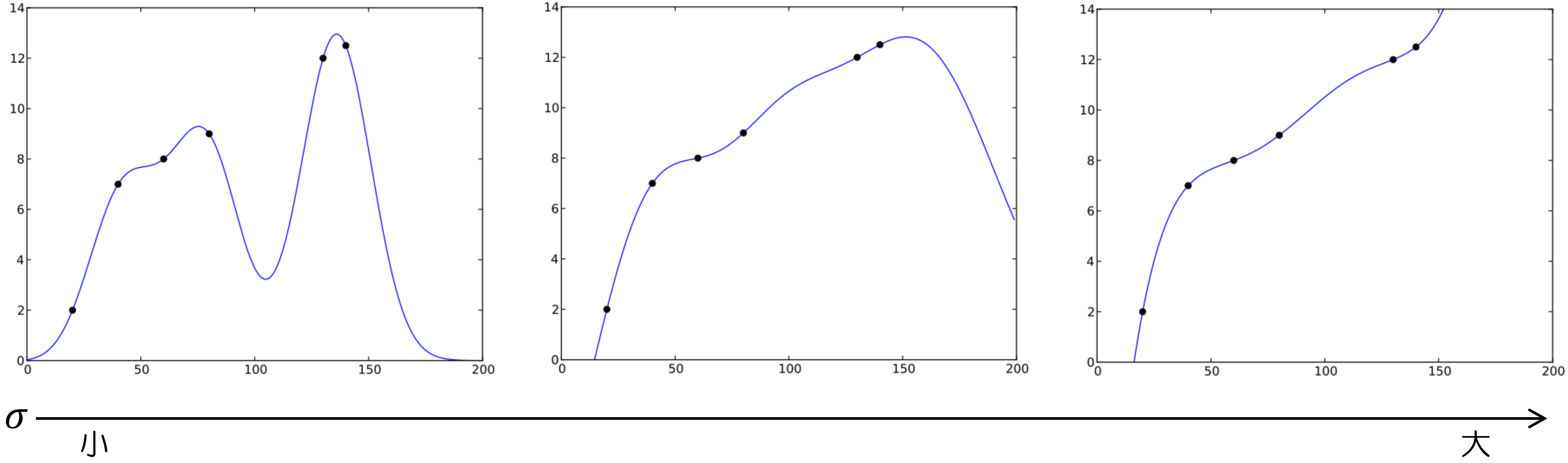
$$\underbrace{\begin{bmatrix} \phi_{1,1} & \phi_{1,2} & & \phi_{1,N} \\ \phi_{2,1} & \phi_{2,2} & & \phi_{2,N} \\ & & \ddots & \\ \phi_{N,1} & \phi_{N,2} & & \phi_{N,N} \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}}_{\mathbf{w}} = \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}}_{\mathbf{f}}$$

これを解けば良い

Gaussian 基底関数を使う場合

$$\phi(\mathbf{x}) = e^{-\|\mathbf{x}\|^2/\sigma^2}$$

- パラメタ σ の選び方によって、結果が大きく変わる！



- なるべく滑らかな結果を得るには？

関数の “曲がり具合” の尺度 (Thin-Plate Energy)

- 2 階微分 (≒曲率) の大きさを空間全体で積分したもの：

$$E_2[f] = \int_{\mathbf{x}\in\mathbb{R}^d} \|\Delta f(\mathbf{x})\|^2 d\mathbf{x}$$

ラプラシアン (Laplacian) 演算子

  - 1 次元空間の場合：

$$E_2[f] = \int_{x\in\mathbb{R}} f''(x)^2 dx$$

  - 2 次元空間の場合：

$$E_2[f] = \int_{\mathbf{x}\in\mathbb{R}^2} \left(f_{\mathrm{xx}}(\mathbf{x})^2 + 2f_{\mathrm{xy}}(\mathbf{x})^2 + f_{\mathrm{yy}}(\mathbf{x})^2\right) d\mathbf{x}$$

  - 3 次元空間の場合：

$$E_2[f] = \int_{\mathbf{x}\in\mathbb{R}^3} \left(f_{\mathrm{xx}}(\mathbf{x})^2 + f_{\mathrm{yy}}(\mathbf{x})^2 + f_{\mathrm{zz}}(\mathbf{x})^2 + 2f_{\mathrm{xy}}(\mathbf{x})^2 + 2f_{\mathrm{yz}}(\mathbf{x})^2 + 2f_{\mathrm{zx}}(\mathbf{x})^2\right) d\mathbf{x}$$

数学分野の知見

- 制約 $\{ f(\mathbf{x}_i) = f_i \}$ を満たす関数全体のうち、E_2 を最小化する関数は以下の基底を使った RBF として表せる：
  - 1 次元空間の場合：$\phi(x) = |x|^3$
  - 2 次元空間の場合：$\phi(\mathbf{x}) = \|\mathbf{x}\|^2 \log \|\mathbf{x}\|$
  - 3 次元空間の場合：$\phi(\mathbf{x}) = \|\mathbf{x}\|$
- 参考
  - 有限要素法の場合：離散化した領域上で E_2 を最小化する f を近似的に求める
  - RBF の場合：グリーン関数を使って E_2 を最小化する f を解析的に求める

Scattered Data Interpolation for Computer Graphics [Anjyo SIGGRAPH14 Course]

線形項の追加

- $E_2[f]$ は 2 階微分を使って定義される
  ➔ 任意の線形項 $p(\mathbf{x}) = ax + by + cz + d$ を加えても不変：

$$E_2[f + p] = E_2[f] \quad \forall f$$

- 線形項を未知数に含めることで、関数を一意に定める：

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \, \phi(\mathbf{x} - \mathbf{x}_i) + ax + by + cz + d$$

線形項の追加

$$f(\mathbf{x}_1) = w_1\phi_{1,1} + w_2\phi_{1,2} + \cdots + w_N\phi_{1,N} + ax_1 + by_1 + cz_1 + d = f_1$$

$$f(\mathbf{x}_2) = w_1\phi_{2,1} + w_2\phi_{2,2} + \cdots + w_N\phi_{2,N} + ax_2 + by_2 + cz_2 + d = f_2$$

$$\vdots$$

$$f(\mathbf{x}_N) = w_1\phi_{N,1} + w_2\phi_{N,2} + \cdots + w_N\phi_{N,N} + ax_N + by_N + cz_N + d = f_N$$

$$\left[\begin{array}{ccc|cccc} \phi_{1,1} & \phi_{1,2} & \phi_{1,N} & x_1 & y_1 & z_1 & 1 \\ \phi_{2,1} & \phi_{2,2} & \phi_{2,N} & x_2 & y_2 & z_2 & 1 \\ & \Phi \ddots & & & \mathrm{P} \vdots & & \\ \phi_{N,1} & \phi_{N,2} & \phi_{N,N} & x_N & y_N & z_N & 1 \end{array}\right] \begin{bmatrix} w_1 \\ w_2 \\ \mathbf{w} \vdots \\ w_N \\ \hline a \\ b \\ \mathbf{c} \; c \\ d \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \mathbf{f} \vdots \\ f_N \end{bmatrix}$$

4 個の未知数 a, b, c, d が追加されたので、4 個の制約を追加する必要がある

追加の制約条件：線形関数の再現性

- 「全てのデータ点の制約 ($\mathbf{x}_i$, f_i) がある線形関数からのサンプリングであるとき、RBF による補間結果はその線形関数と一致する」
- これを満たすための条件：
  - $\sum_{i=1}^{N} w_i = 0$
  - $\sum_{i=1}^{N} x_i w_i = 0$
  - $\sum_{i=1}^{N} y_i w_i = 0$
  - $\sum_{i=1}^{N} z_i w_i = 0$
- 行列がちょうど対称になる

$$\begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,N} & x_1 & y_1 & z_1 & 1 \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,N} & x_2 & y_2 & z_2 & 1 \\ \vdots & & \ddots & & \vdots & & & \\ \phi_{N,1} & \phi_{N,2} & \cdots & \phi_{N,N} & x_N & y_N & z_N & 1 \\ x_1 & x_2 & \cdots & x_N & 0 & 0 & 0 & 0 \\ y_1 & y_2 & \cdots & y_N & 0 & 0 & 0 & 0 \\ z_1 & z_2 & & y_N & 0 & 0 & 0 & 0 \\ 1 & 1 & & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \\ a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} \Phi & \mathrm{P} \\ \mathrm{P}^\intercal & \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \; \end{bmatrix}$$

勾配制約の導入

- 基底関数の勾配 $\nabla\phi$ の重み付き和を導入：

$$f(\mathbf{x}) = \sum_{i=1}^{N} \left\{ w_i \phi(\mathbf{x} - \mathbf{x}_i) + \mathbf{v}_i^{\top} \nabla\phi(\mathbf{x} - \mathbf{x}_i) \right\} + ax + by + cz + d$$

未知数の3Dベクトル

- f の勾配：

$$\nabla f(\mathbf{x}) = \sum_{i=1}^{N} \left\{ w_i \nabla\phi(\mathbf{x} - \mathbf{x}_i) + \mathrm{H}_{\phi}(\mathbf{x} - \mathbf{x}_i)\mathbf{v}_i \right\} + \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

- 勾配の制約 $\nabla f(\mathbf{x}_i) = \mathbf{n}_i$ を追加

$$\mathrm{H}_{\phi}(\mathbf{x}) = \begin{pmatrix} \phi_{\mathrm{xx}} & \phi_{\mathrm{xy}} & \phi_{\mathrm{xz}} \\ \phi_{\mathrm{yx}} & \phi_{\mathrm{yy}} & \phi_{\mathrm{yz}} \\ \phi_{\mathrm{zx}} & \phi_{\mathrm{zy}} & \phi_{\mathrm{zz}} \end{pmatrix}$$

Hessian 行列 (関数)

Hermite Radial Basis Functions Implicits [Macedo CGF10]

勾配制約の導入

- 1 番目のデータ点について：

値の制約：

$$f(\mathbf{x}_1) = w_1\phi_{1,1} + \mathbf{v}_1^\top \nabla\phi_{1,1} + w_2\phi_{1,2} + \mathbf{v}_2^\top \nabla\phi_{1,2} + \cdots + w_N\phi_{1,N} + \mathbf{v}_N^\top \nabla\phi_{1,N} \quad by_1 + cz_1 + d = f_1$$

勾配の制約：

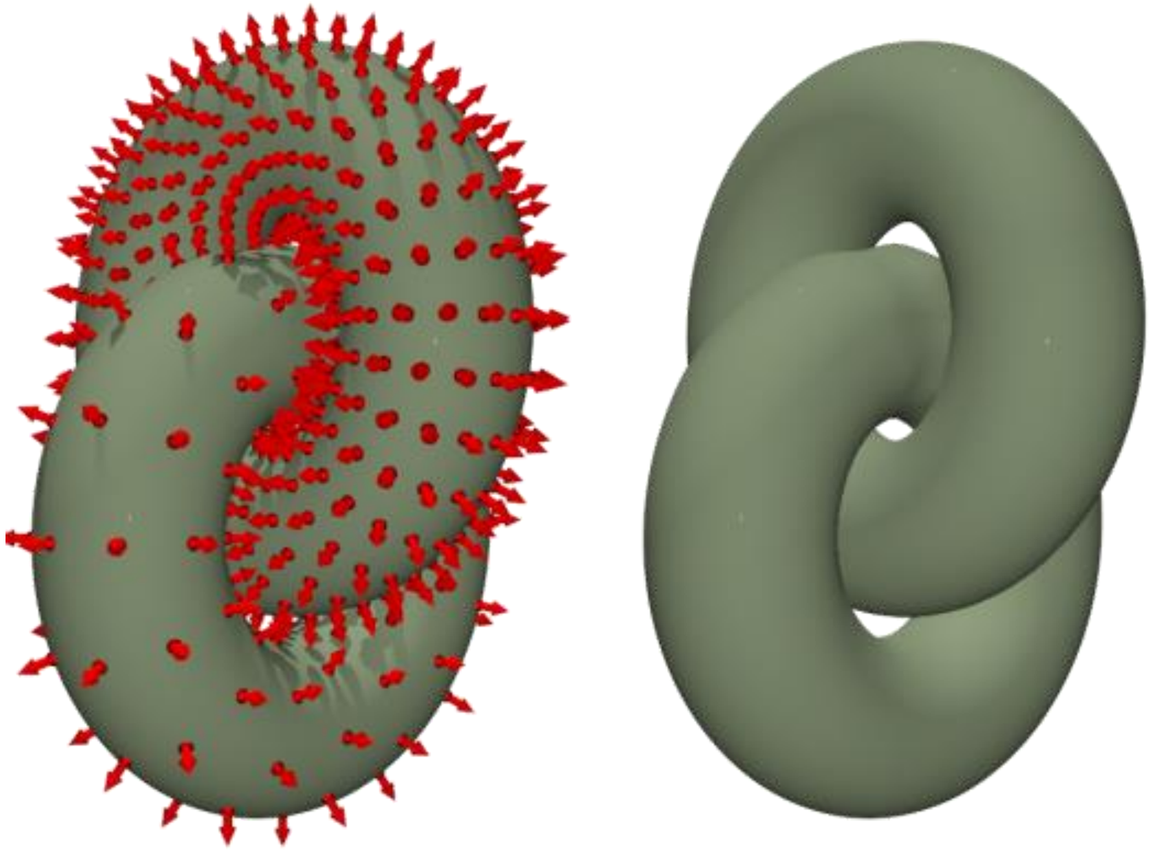
$$\nabla f(\mathbf{x}_1) = w_1\nabla\phi_{1,1} + \mathrm{H}_\phi^{1,1}\mathbf{v}_1 + w_2\nabla\phi_{1,2} + \mathrm{H}_\phi^{1,2}\mathbf{v}_2 + \cdots + w_N\nabla\phi_{1,N} + \mathrm{H}_\phi^{1,N} \quad \mathbf{n}_1$$

$$\begin{bmatrix} \Phi_{1,1} & \Phi_{1,2} & \cdots & \Phi_{1,N} & \mathrm{P}_1 \end{bmatrix} \begin{bmatrix} w_1 \\ \mathbf{v}_1 \\ w_2 \\ \mathbf{v}_2 \\ \vdots \\ w_N \\ \mathbf{v}_N \\ a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} f_1 \\ \mathbf{n}_1 \end{bmatrix}$$

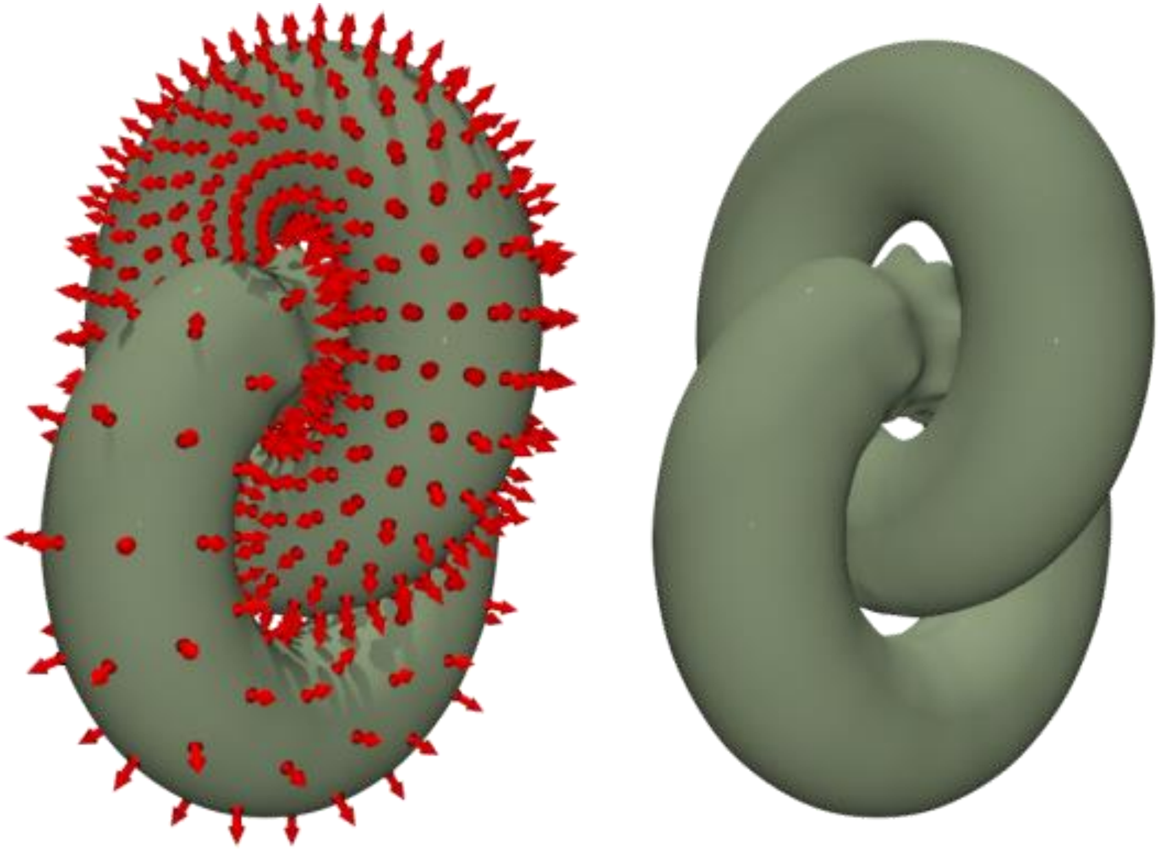
$$\Phi_{1,1} = \begin{bmatrix} \phi_{1,1} & (\nabla\phi_{1,1})^\top \\ \nabla\phi_{1,1} & \mathrm{H}_\phi^{1,1} \end{bmatrix}, \quad \Phi_{1,2} = \begin{bmatrix} \phi_{1,2} & (\nabla\phi_{1,2})^\top \\ \nabla\phi_{1,2} & \mathrm{H}_\phi^{1,2} \end{bmatrix}, \quad \Phi_{1,N} = \begin{bmatrix} \phi_{1,N} & (\nabla\phi_{1,N})^\top \\ \nabla\phi_{1,N} & \mathrm{H}_\phi^{1,N} \end{bmatrix}, \quad \mathrm{P}_1 = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \Phi_{1,1} & \Phi_{1,2} & \cdots & \Phi_{1,N} & \mathrm{P}_1 \\ \Phi_{2,1} & \Phi_{2,2} & & \Phi_{2,N} & \mathrm{P}_2 \\ & \vdots & \ddots & \vdots & \\ \Phi_{N,1} & \Phi_{N,2} & & \Phi_{N,N} & \mathrm{P}_N \\ P_1^\intercal & P_2^\intercal & \cdots & P_N^\intercal & \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix} \end{bmatrix} \begin{bmatrix} w_1 \\ \mathbf{v}_1 \\ w_2 \\ \mathbf{v}_2 \\ \vdots \\ w_N \\ \mathbf{v}_N \\ a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} f_1 \\ \mathbf{n}_1 \\ f_2 \\ \mathbf{n}_2 \\ \vdots \\ f_N \\ \mathbf{n}_N \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

比較



勾配を制約



オフセットして値を制約

参考サーベイ等

- State of the Art in Surface Reconstruction from Point Clouds [Berger EG14 STAR]
- A survey of methods for moving least squares surfaces [Cheng PBG08]
- Scattered Data Interpolation for Computer Graphics [Anjyo SIGGRAPH14 Course]
- An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares for scattered data approximation and interpolation [Nealen TechRep04]

参考ページ

- http://en.wikipedia.org/wiki/Implicit_surface
- http://en.wikipedia.org/wiki/Radial_basis_function
- http://en.wikipedia.org/wiki/Thin_plate_spline
- http://en.wikipedia.org/wiki/Polyharmonic_spline