Geometric Modeling

Summer Semester 2012

Implicit Surfaces

Mathematical Background · Level Set Extraction · Solid Modeling

UNIVERSITÄT DES SAARLANDES

CLUSTER OF EXCELLENCE

max planck institut informatik

Overview...

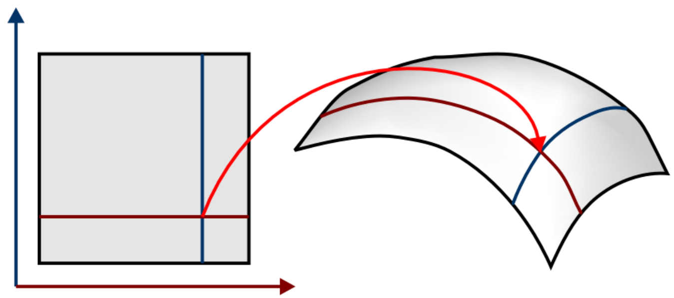
Topics:

- Spline Surfaces
- Implicit Functions
  - Introduction / Mathematical Background
  - Numerical Discretization
  - Level Set Extraction Algorithms
  - Solid Modeling
  - Data Fitting
- Subdivision Surfaces
- Variational Modeling

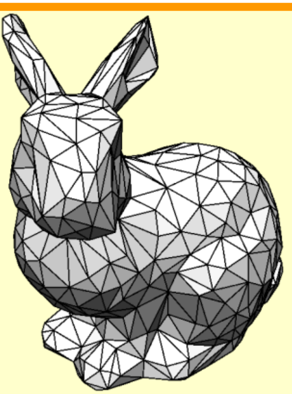
Implicit Surfaces

Introduction

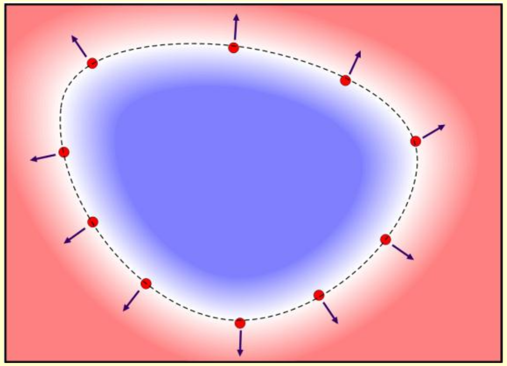
Modeling Zoo



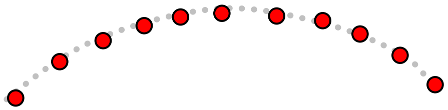
Parametric Models



Primitive Meshes



Implicit Models



Particle Models

Implicit Functions

Basic Idea:

- We describe an object $S \subseteq \mathbb{R}^d$ by an implicit equation:
  - $S = \{\mathbf{x} \in \mathbb{R}^d \mid f(\mathbf{x}) = 0\}$
  - The function f describes the shape of the object.
- Applications:
  - In general, we could describe arbitrary objects
  - Most common case: surfaces in $\mathbb{R}^3$.
  - This means, f is zero on an infinitesimally thin sheet only.

The Implicit Function Theorem

Implicit Function Theorem:

- Given a *differentiable* function
  $f : \mathbb{R}^n \supseteq D \to \mathbb{R},\ f(\mathbf{x}^{(0)}) = 0,\ \frac{\partial}{\partial x_n} f(\mathbf{x}^{(0)}) = \frac{\partial}{\partial x_n} f(x_1^{(0)}, \dots, x_n^{(0)}) \neq 0$
- Within an ε-neighborhood of $\mathbf{x}^{(0)}$ we can represent the zero level set of f completely as a heightfield function g
  $g : \mathbb{R}^{n\text{-}1} \to \mathbb{R}$ such that for $\|\mathbf{x} - \mathbf{x}^{(0)}\| < \varepsilon$ we have:
  $f(x_1, \dots, x_{n\text{-}1}, g(x_1, \dots, x_{n\text{-}1})) = 0$ and
  $f(x_1, \dots, x_n) \neq 0$ everywhere else.
- The heightfield is a differentiable $(n - 1)$-manifold and its surface normal is the colinear to the gradient of f.

This means

If we want to model surfaces, we are on the safe side if:

- We use a smooth (differentiable) function f in $\mathbb{R}^3$.
- The gradient of f does not vanish.

This gives us the following guarantees:

- The zero-level set is actually a surface:
  - We obtained a closed 2-manifold without boundary.
  - We have a well defined interior / exterior.

Sufficient:

- We need smoothness / non-vanishing gradient only close to the zero-crossing.

Implicit Function Types

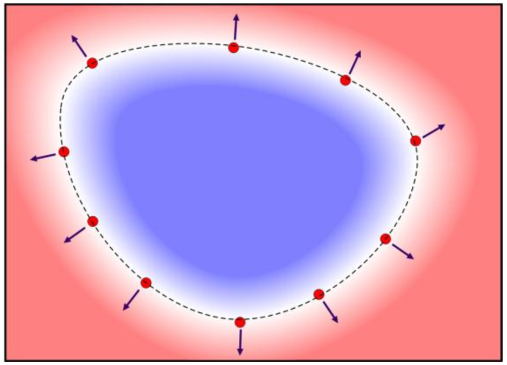
Function types:

- General case
  - Non-zero gradient at zero crossing
  - Otherwise arbitrary
- Signed implicit function:
  - sign(f): negative inside and positive outside the object
    (or the other way round, but we assume this orientation here)
- Signed distance field
  - $|f|$ = distance to the surface
  - sign(f): negative inside, positive outside
- Squared distance function
  - f = (distance to the surface)^2

Implicit Function Types

Use depends on application:

- Signed implicit function
  - Solid modeling
  - Interior well defined
- Signed distance function
  - Most frequently used representation
  - Constant gradient $\rightarrow$ numerically stable surface definition
  - Availability of distance values useful for many applications
- Squared distance function
  - This representation is useful for statistical optimization
  - Minimize sum of squared distances $\rightarrow$ least squares optimization
  - Useful for surfaces defined up to some insecurity / noise.
  - Direct surface extraction more difficult (*gradient vanishes!*).

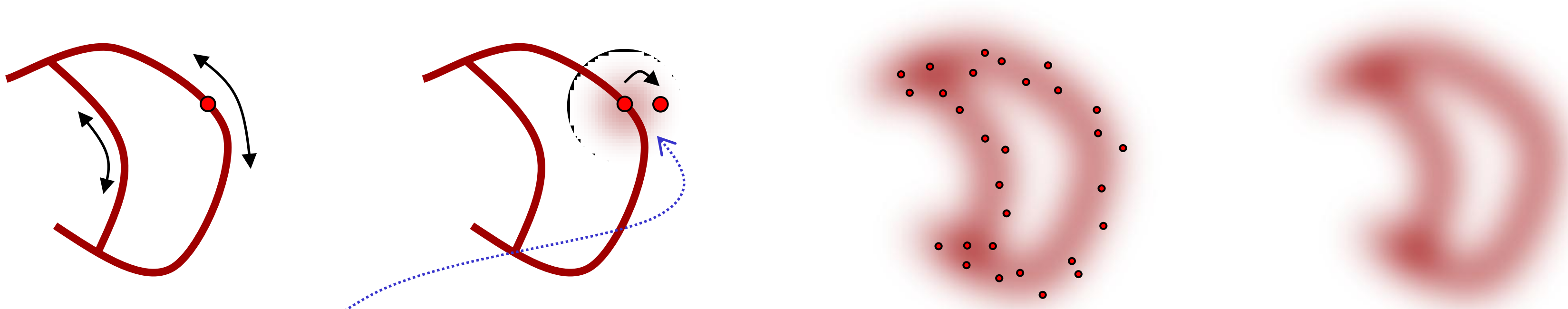


signed distance

Squared Distance Function

Example: Surface from random samples

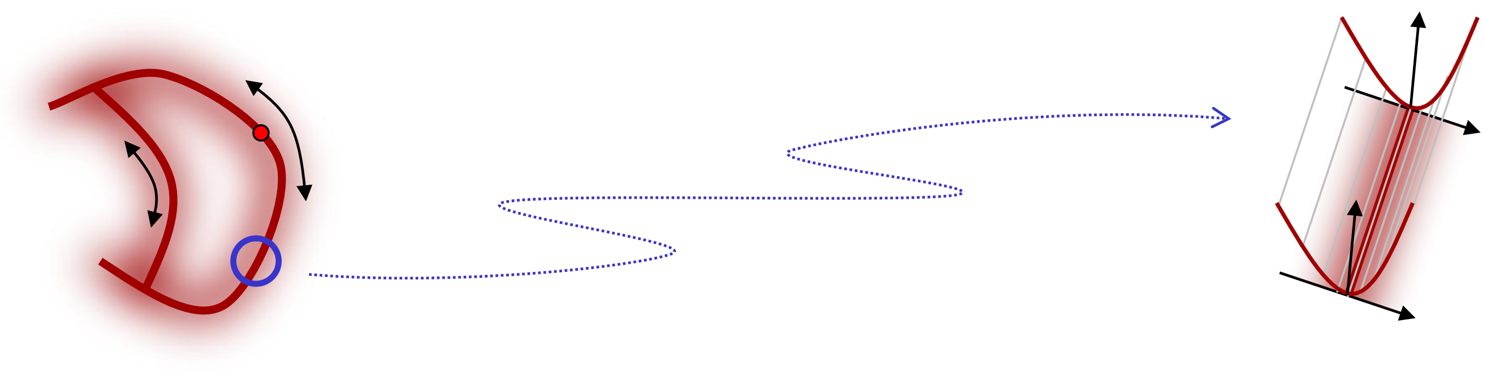
1. Determine sample point (uniform)
2. Add noise (Gaussian)



sampling　　Gaussian noise　　many samples　　distribution (in space)

$$p_{\boldsymbol{\mu},\boldsymbol{\Sigma}}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

Squared Distance Function



Squared Distance Function:

- Sampling a surface with uniform sampling and Gaussian noise:

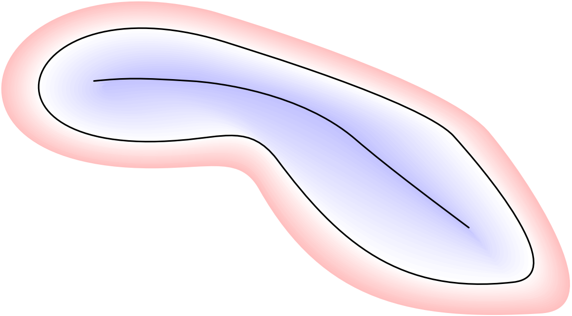
⇒ Probability density is a convolution of the object with a Gaussian kernel

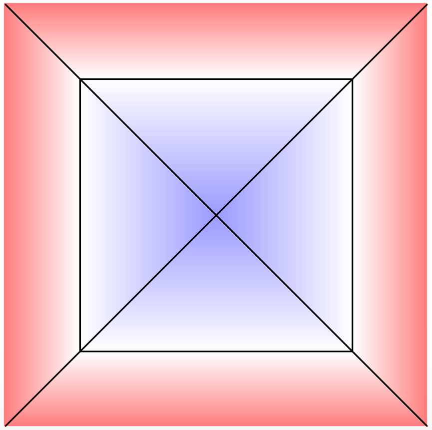
- Smooth surfaces: The log-likelihood can be approximated by a squared distance function

Smoothness

Smoothness of signed distance function:

- Any distance function (signed, unsigned, squared) in general cannot be globally smooth
- The distance function is non-differentiable at the medial axis
  - Medial axis = set of points that have the same distance to two or more different surface points
  - For sharp corners, the medial axis touches the surfaces
  - This means: f non-differentiable on the surface itself
  - Usually, this is no problem in practice.





Differential Properties

Some useful differential properties:

- We look at a surface point $\mathbf{x}$, i.e. $f(\mathbf{x}) = 0$.
- We assume $\nabla f(\mathbf{x}) \neq 0$.
- The unit normal of the implicit surface is given by:

$$\mathbf{n}(\mathbf{x}) = \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$$

  - For signed functions, the normal is pointing outward.
  - For signed distance functions, this simplifies to $\mathbf{n}(\mathbf{x}) = \nabla f(\mathbf{x})$.

Differential Properties

Some useful differential properties:

- The mean curvature of the surface is proportional to the divergence of the unit normal:

$$
\begin{aligned}
-2H(\mathbf{x}) &= \nabla \cdot \mathbf{n}(\mathbf{x}) \\
&= \frac{\partial}{\partial x} n_x(\mathbf{x}) + \frac{\partial}{\partial y} n_y(\mathbf{x}) + \frac{\partial}{\partial z} n_z(\mathbf{x}) \\
&= \nabla \cdot \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}
\end{aligned}
$$

- For a signed distance function, the formula simplifies to:

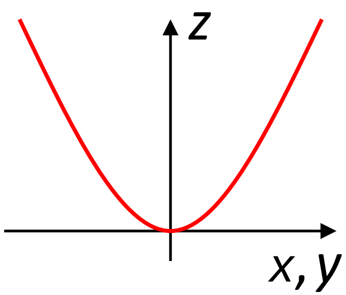
$$
\begin{aligned}
-2H(\mathbf{x}) &= \nabla \cdot \nabla f(\mathbf{x}) = \frac{\partial^2}{\partial x^2} f(\mathbf{x}) + \frac{\partial^2}{\partial y^2} f(\mathbf{x}) + \frac{\partial^2}{\partial z^2} f(\mathbf{x}) \\
&= \Delta f(\mathbf{x})
\end{aligned}
$$

Mean Curvature Formula

Proof (sketch):

- We assume that the normal is in *z*-direction, i.e., *x*, *y* are tangent to the surface (divergence is invariant under rotation). The surface normal is given by:

$$\mathbf{n}(x,y) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -\partial_x s(x,y) \\ -\partial_y s(x,y) \\ 1 \end{pmatrix}$$



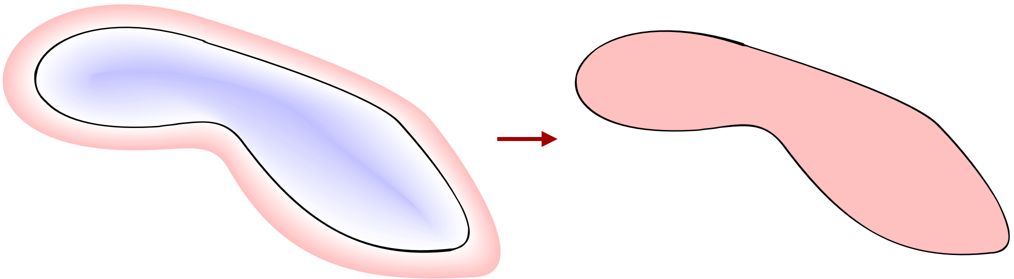
$$\nabla \cdot \mathbf{n}(x,y) = -\frac{\partial^2}{\partial x^2} s(x,y) - \frac{\partial^2}{\partial y^2} s(x,y) + \frac{\partial}{\partial z} 1$$

$$= trace \begin{pmatrix} \frac{\partial^2}{\partial x^2} s(x,y) & \frac{\partial^2}{\partial x \partial y} s(x,y) \\ \frac{\partial^2}{\partial x \partial y} s(x,y) & \frac{\partial^2}{\partial y^2} s(x,y) \end{pmatrix} = -2H(x,y) \quad \left[H(\mathbf{x}_0) = \frac{1}{2} \operatorname{tr}\left(S(x_0)\right) \right]$$

Computing Volume Integrals

Computing volume integrals:

- Heavyside function:



$$\mathrm{step}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

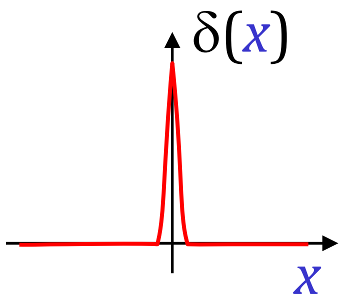
- Volume integral over interior volume Ω_f of some function $g(\mathbf{x})$ (assuming negative interior values):

$$\int_{\Omega_f} g(\mathbf{x}) d\mathbf{x} = \int_{\mathbb{R}^3} g(\mathbf{x})\left(1 - \mathrm{step}(f(x))\right) d\mathbf{x}$$

Computing Surface Integrals

Computing surface integrals:

- Dirac delta function:
  - Idealized function (distribution)
  - Zero everywhere ($\delta(x) = 0$), except at $x = 0$, where it is positive, inifinitely large.
  - The integral of $\delta(x)$ over x is one.



- Dirac delta function on the surface: directional derivative of step(x) in normal direction:

$$\hat{\delta} = \nabla\left[\mathrm{step}(f(\mathbf{x}))\right] \cdot \mathbf{n}(\mathbf{x}) = \left[\nabla \mathrm{step}\right](f(\mathbf{x})) \cdot \nabla f(\mathbf{x}) \frac{\nabla f(\mathbf{x})}{\left\|\nabla f(\mathbf{x})\right\|}$$

$$= \delta(f(\mathbf{x})) \cdot \left|\nabla f(\mathbf{x})\right|$$

Surface Integral

Computing surface integrals:

- Surface integral over the surface $\partial\Omega_f = \{\mathbf{x} \mid f(\mathbf{x}) = 0\}$ of some function $g(\mathbf{x})$:

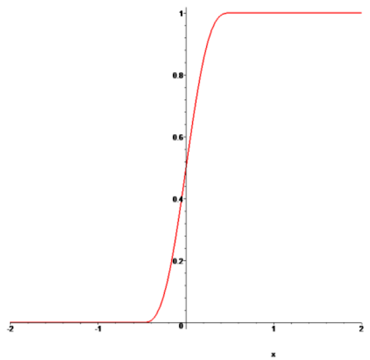
$$\int_{\partial\Omega_f} g(\mathbf{x})d\mathbf{x} = \int_{\mathbb{R}^3} g(\mathbf{x})\delta(f(\mathbf{x}))\,|\nabla f(\mathbf{x})|\,d\mathbf{x}$$

- This looks nice, but is numerically intractable.
- We can fix this using smothed out Dirac/Heavyside functions...

Smoothed Functions

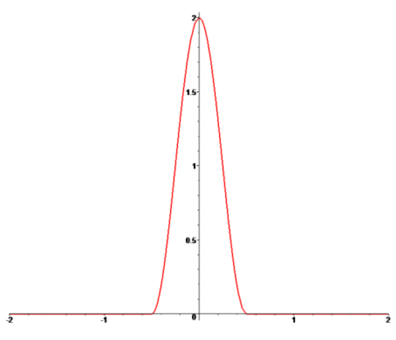
Smooth-step function

$$\mathrm{smooth\_step}(x) = \begin{cases} 0 & x < -\varepsilon \\ \dfrac{1}{2} + \dfrac{x}{2\varepsilon} + \dfrac{1}{2\pi}\sin\left(\dfrac{\pi x}{\varepsilon}\right) & -\varepsilon \le x \le \varepsilon \\ 1 & \varepsilon < x \end{cases}$$



Smoothed Dirac delta function

$$\mathrm{smooth\_delta}(x) = \begin{cases} 0 & x < -\varepsilon \\ \dfrac{1}{2\varepsilon} + \dfrac{1}{2\varepsilon}\cos\left(\dfrac{\pi x}{\varepsilon}\right) & -\varepsilon \le x \le \varepsilon \\ 0 & x > \varepsilon \end{cases}$$



Implicit Surfaces

Numerical Discretization

Representing Implicit Functions

Representation: Two basic techniques

- Discretization on grids
  - Simple finite differencing (FD) grids
  - Grids of basis functions (finite elements FE)
  - Hierarchical / adaptive grids (FE)
- Discretization with radial basis functions (particle FE methods)

Discretization

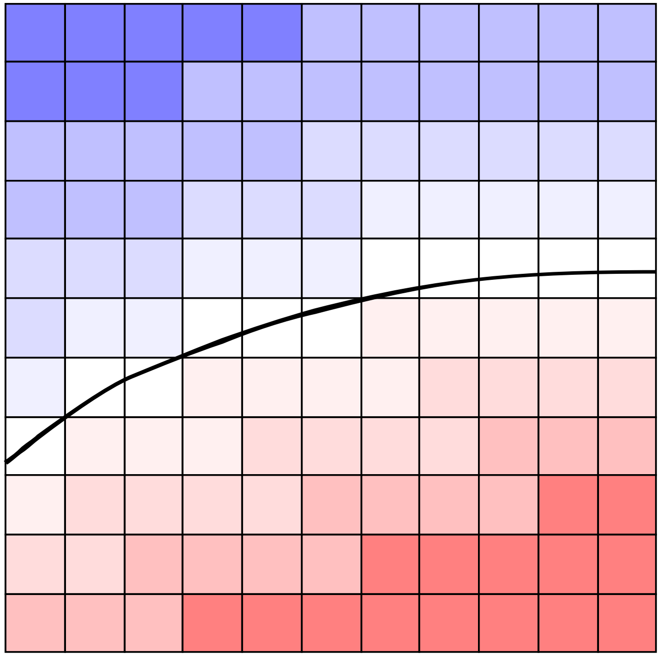
Discretization examples

- In the following, we will look at 2D examples
- The 3D (*d*-dimensional) case is similar

Regular Grids

Discretization:

- Regular grid of values $f_{i,j}$
- Grid spacing h
- Differential properties can be approximated by finite differences:
  - For example:



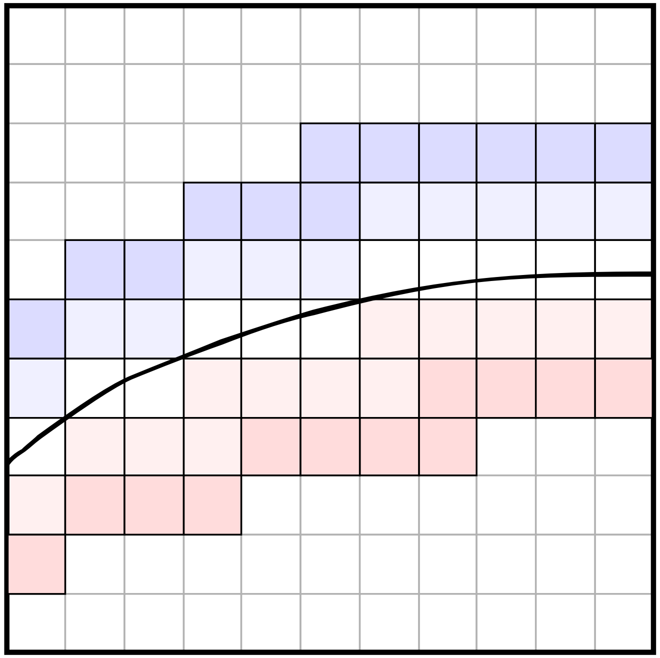
$$\frac{\partial}{\partial_x} f(\mathbf{x}) = \frac{1}{h}\left(f_{i(\mathbf{x}),j(\mathbf{x})} - f_{i(\mathbf{x})-1,j(\mathbf{x})}\right) + O(h)$$

$$\frac{\partial}{\partial_x} f(\mathbf{x}) = \frac{1}{2h}\left(f_{i(\mathbf{x})+1,j(\mathbf{x})} - f_{i(\mathbf{x})-1,j(\mathbf{x})}\right) + O(h^2)$$

Regular Grids

Variant:

- Use only cells near the surface
- Saves storage & computation time
- However: We need to know an estimate on where the surface is located to setup the representation
- Propagate to the rest of the volume (if necessary): *fast marching method*



Fast Marching Method

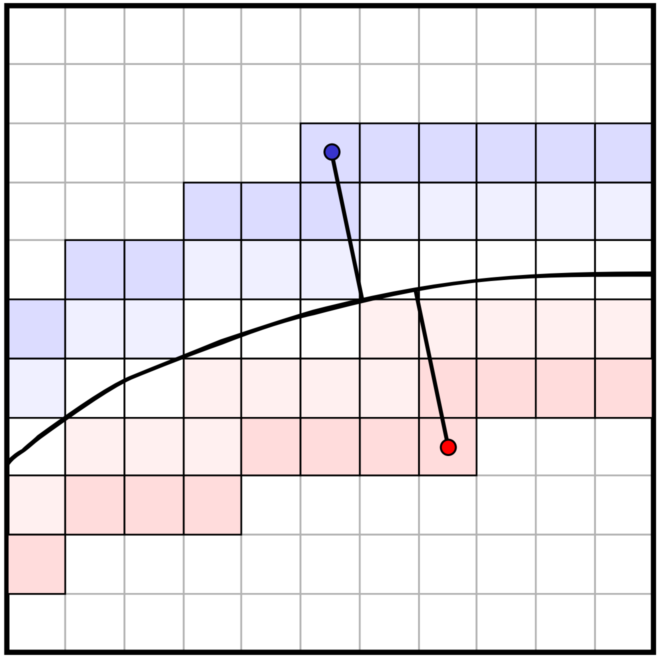
Problem statement:

- Assume we are given the surface and signed distance value in a narrow band.
- Now we want to compute distance values everywhere on the grid.

Three solutions:

- Nearest neighbor queries
- Eikonal equation
- Fast marching

Nearest Neighbors



Algorithm:

- For each grid cell:
  - Compute nearest point on the surface
  - Enter distance
- Approximate nearest neighbor computation:
  - Look for nearest grid cell with zero crossing first
  - Then compute distance curve $\leftrightarrow$ zero level set using a Newton-like algorithm (repeated point-to-plane distance)
- Costs: O(n) kNN queries (n empty cells)

Eikonal Equation

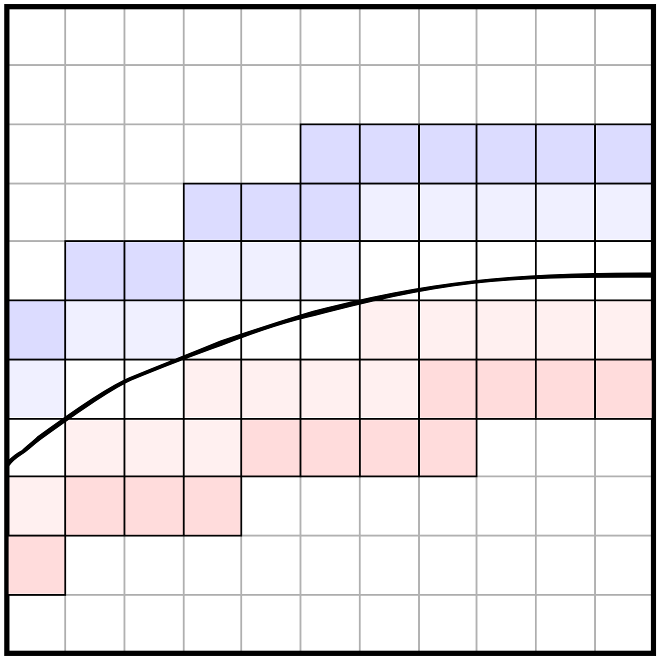
Eikonal Equation

- Place variables in empty cells
- Fixed values in known cells
- Then solve the following PDE:

$$\|\nabla f\| = 1$$

$$\text{subject to } f(\mathbf{x}) = f_{known}(\mathbf{x})$$

$$\text{on the known area } \mathbf{x} \in A_{known}$$



- This is a (non-linear) boundary value problem.

Fast Marching

Solving the Equation:

- The Eikonal equation can be solved efficiently by a region growing algorithm:
  - Start with the initial known values
  - Compute new distances at immediate neighbors solving a local Eikonal equation (*)
  - The smallest of these values must be correct (similar to Dijkstra's algorithm)
  - Fix this value and update the neighbors again
  - Growing front, O(*n* log *n*) time.

(*) for details see: J.A. Sethian, Level Set Methods and Fast Marching Methods, Cambridge University Press 1996.

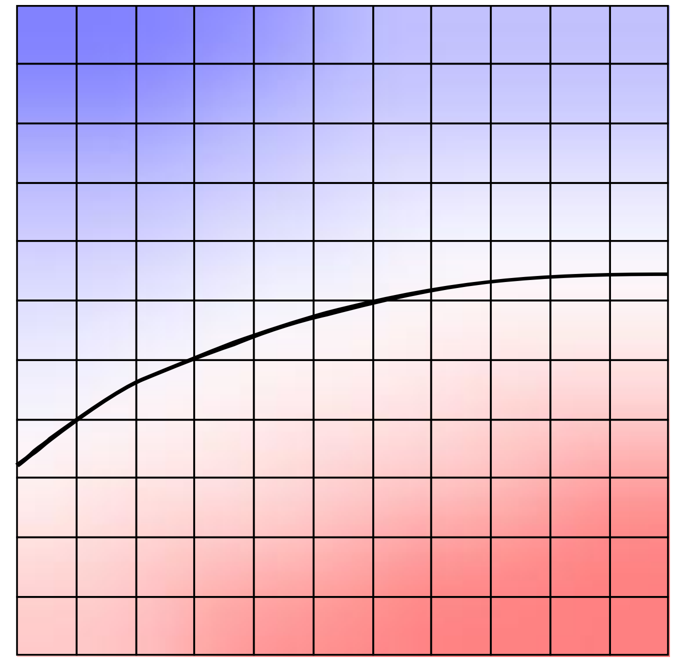
Regular Grids of Basis Functions

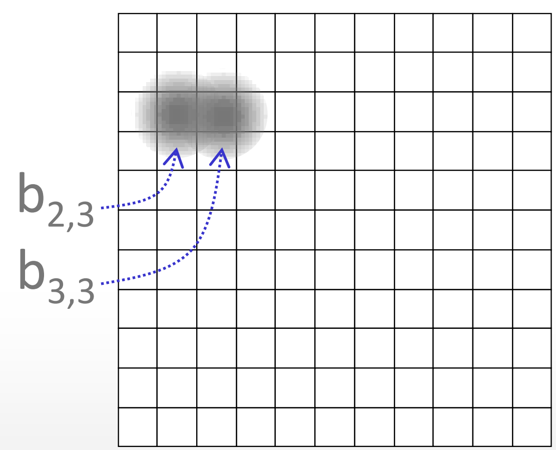
Discretization (2D):

- Place a basis function in each grid cell: $b_{i,j} = b(x - i, y - j)$
- Typical choices:
  - Bivariate uniform cubic B-splines (tensor product)
  - $b(x,y) = \exp[-\lambda(x^2 + y^2)]$
- The implicit function is then represented as:

$$f(x,y) = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \lambda_{i,j} b_{i,j}(x,y)$$

- The $\lambda_{i,j}$ describe different f.





Regular Grids of Basis Functions

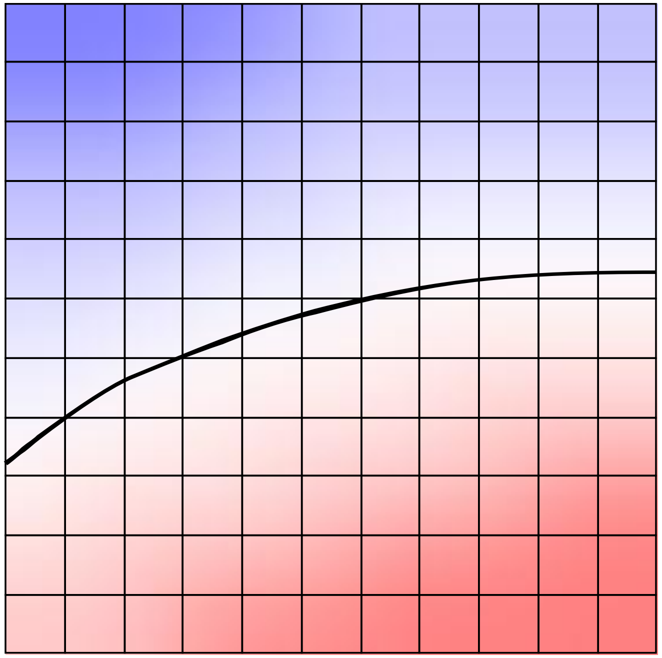
Differential Properties:

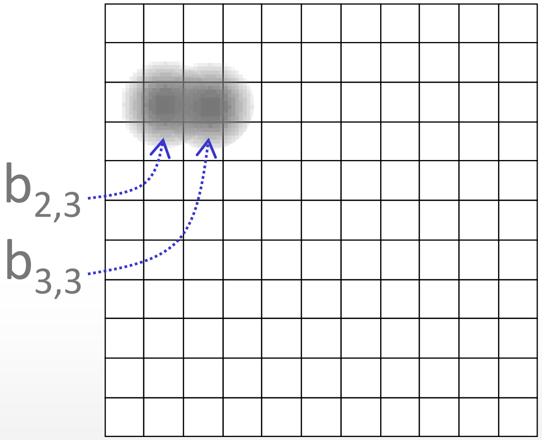
- Derivatives:

$$\frac{\partial}{\partial x_{k1} \ldots \partial x_{km}} f(x,y)$$

$$= \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \lambda_{i,j} \left(\frac{\partial}{\partial x_{k1} \ldots \partial x_{km}} b \right)(x,y)$$

- Derivatives are linear combinations of the derivatives of the basis function.
- In particular: We again get a linear expression in the $\lambda_{i,j}$.

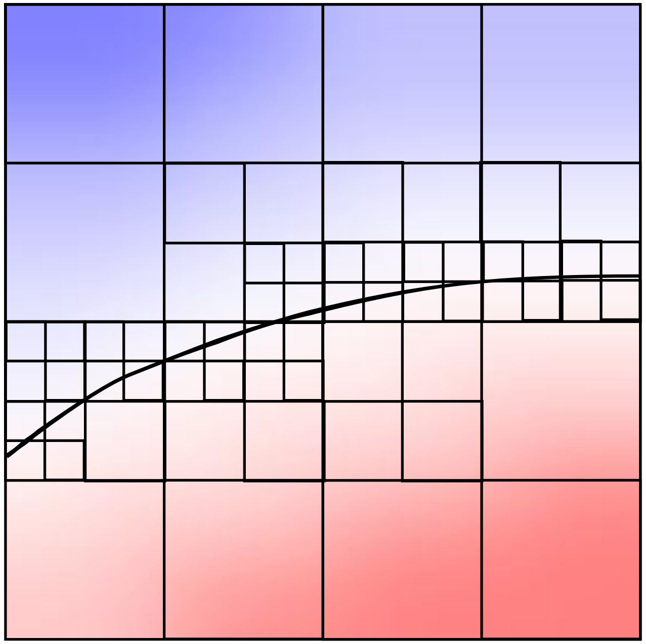




Adaptive Grids

Adaptive / hierarchical grid:

- Perform a quadtree /octree tessellation of the domain (or any other partition into elements)
- Refine where more precision is necessary (near surface, maybe curvature dependent)
- Associate basis functions with each cell (constant or higher order)



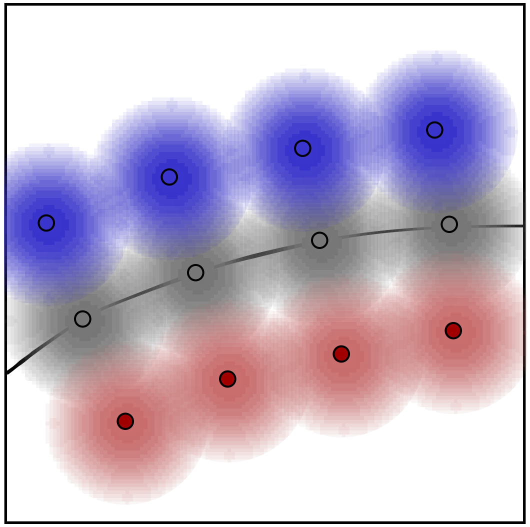
Particle Methods

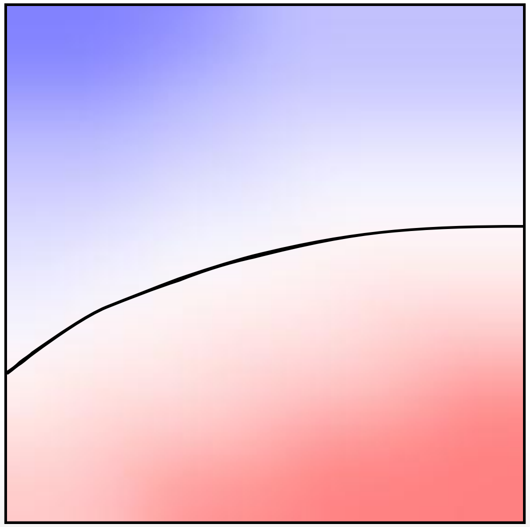
Particle methods / radial basis functions:

- Place a set of "particles" in space at positions $\mathbf{x}_i$.
- Associate each with a radial basis function $b(\mathbf{x}-\mathbf{x}_i)$.
- The discretization is then given by:

$$f(\mathbf{x}) = \sum_{i=0}^{n} \lambda_i b(\mathbf{x}-\mathbf{x}_i)$$

- The λ_i encode f.





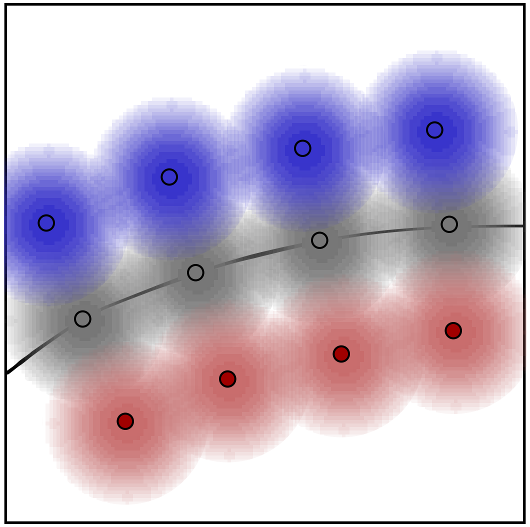
Particle Methods

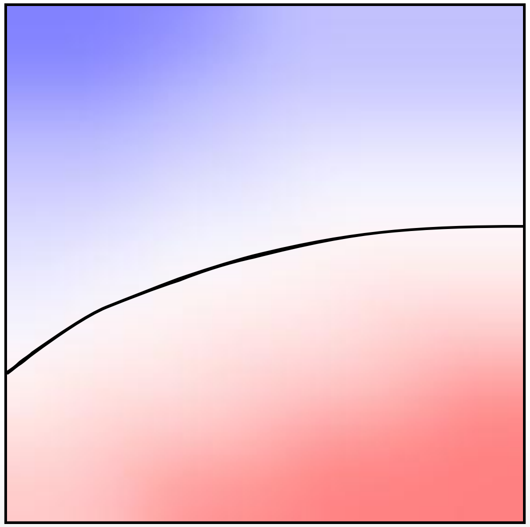
Particle methods / radial basis functions:

- Obviously, derivatives are again linear in λ_i:

$$\frac{\partial}{\partial x_{k1} \ldots \partial x_{km}} f(\mathbf{x}) = \sum_{i=0}^{n} \lambda_i \frac{\partial}{\partial x_{k1} \ldots \partial x_{km}} b(\mathbf{x} - \mathbf{x}_i)$$

- The radial basis functions can also have different size (support) for adaptive refinement
- Placement: near the expected surface

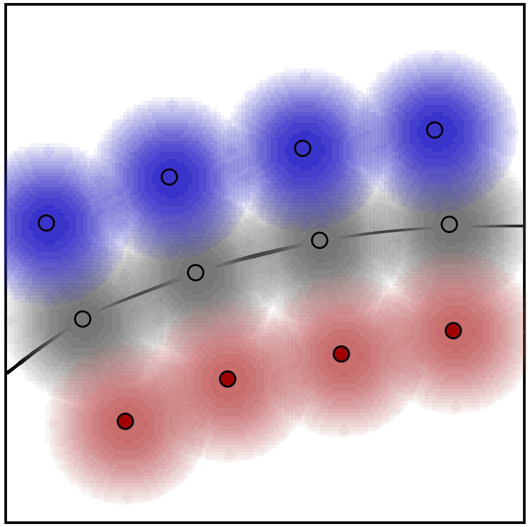


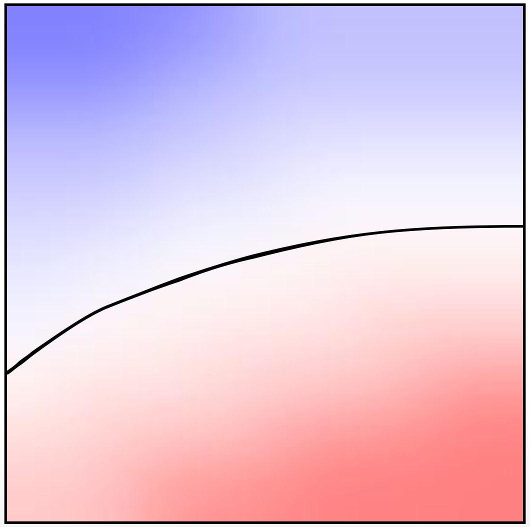


Particle Methods

Particle methods /
radial basis functions:

- Where should we place the radial basis functions?
  - If we have an initial guess for the surface shape:
    - put some on the surface
    - and some in +/- normal direction.
  - Otherwise:
    - Uniform placement in lowres
    - Solve for surface
    - Refine near lowres-surface, iterate.





Types of Radial Basis Functions

Typical choices for radial basis functions:

- (Quasi-) compactly supported functions:
  - Exponentials / normal distribution densities: exp($-\lambda \mathbf{x}^2$)
  - Uniform (cubic) tensor product B-Splines
  - Moving-least squares finite element basis functions (will be discussed later)
- Globally supported functions:
  - Thin plate spline basis functions:
    $\|x - x_0\|^2 \ln\|x - x_0\|$ (2D), $\|x - x_0\|^3$ (3D).
  - These functions guarantee minimal integral second derivatives.

Pros & Cons

Why use globally supported basis functions?

- They come with smoothness guarantees (details in the next lecture)
- However: Computations might become expensive (we will see later how to device efficient algorithms for globally supported radial basis functions)

Locally supported functions:

- Easy to use
- Additional regularization might become necessary to compute a “nice” surface.

Implicit Surfaces

Level Set Extraction

Iso-Surface Extraction

New task:

- Assume we have defined an implicit function
- Now we want to extract the surface.
- I.e. convert it to an explicit, piecewise parametric representation, typically a triangle mesh.
- For this we need an iso-surface extraction algorithm
    - a.k.a. level set extraction
    - a.k.a. contouring

Algorithms

Algorithms:

- Marching Cubes
  - This is *the* standard technique.
  - We will also discuss some problems / modifications.
- Particle methods
  - Just to show an alternative
  - Not used that frequently in practice

Marching Cubes

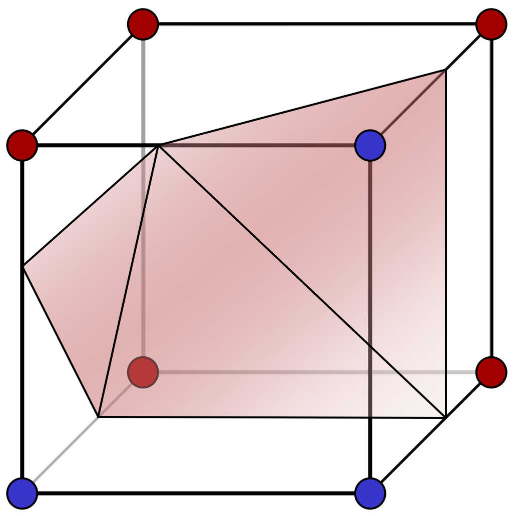
Marching Cubes:

- The most frequently used iso surface extraction algorithm
  - Creates a triangle mesh from an iso-value surface of a scalar volume
  - The algorithm is also used frequently to visualize CT scanner data and other volume data
- Simple idea:
  - Define and solve a fixed complexity, local problem.
  - Compute a full solution by solving many such local problems incrementally.

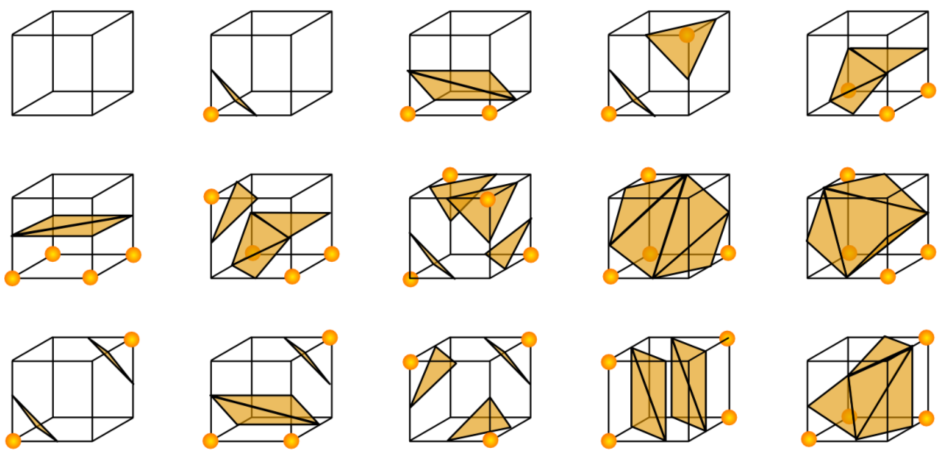
Marching Cubes

Marching Cubes:

- Here is the local problem:
  - We have a cube with 8 vertices
  - Each vertex is either inside or outside the volume (i.e. $f(\mathbf{x}) < 0$ or $f(\mathbf{x}) \geq 0$)
  - How should we triangulate this cube?
  - How should we place the vertices?



Triangulation

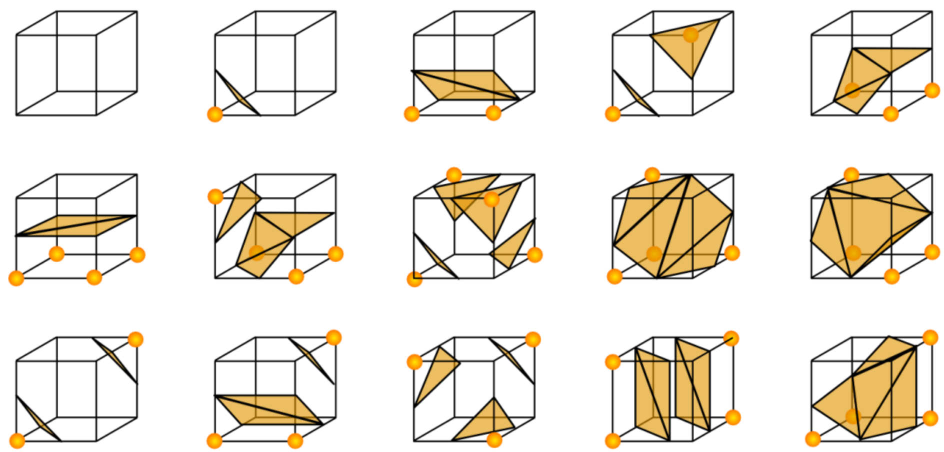


[source: Wikipedia]

Triangulation:

- We have 256 different cases – each of the 8 vertices can be in or out.
- By symmetry, this can be reduced to 15 cases
  - Symmetry: reflection, rotation, and bit inversion
- This means, we can compute the topology of the mesh

Vertex Placement



[source: Wikipedia]

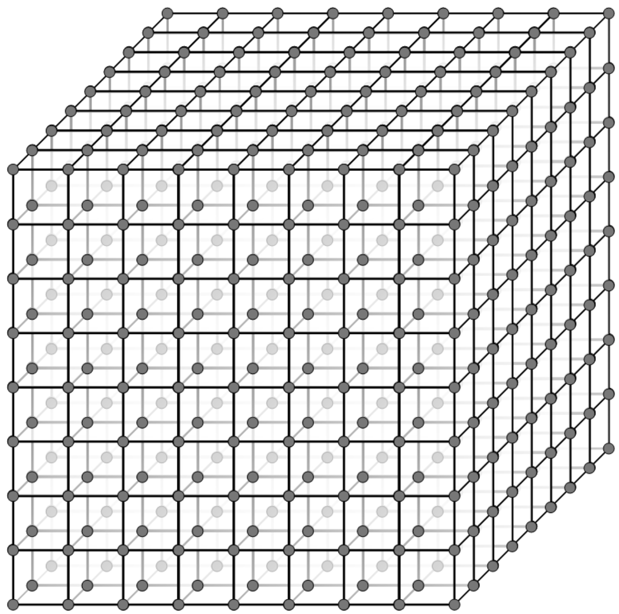
How to place the vertices?

- Zero-th order accuracy: Place vertices at edge midpoints
- First order accuracy: Linearly interpolate vertices along edges.
- Example: for scalar values $f(x) = -0.1$ and $f(y) = 0.2$, place the vertex at ratio 1:2 between x and y.

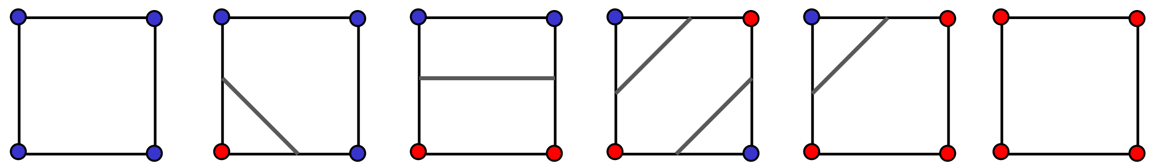
Outer Loop

Outer Loop:

- Compute a bounding box of the domain of the implicit function.
- Divide it into cubes of the same size (regular cube grid)
- Execute "marching cube" algorithm in each subcube
- Output the union of all triangles generated
- Optionally: Use a vertex hash table to make the mesh consistent (remove double vertices)



Marching Squares



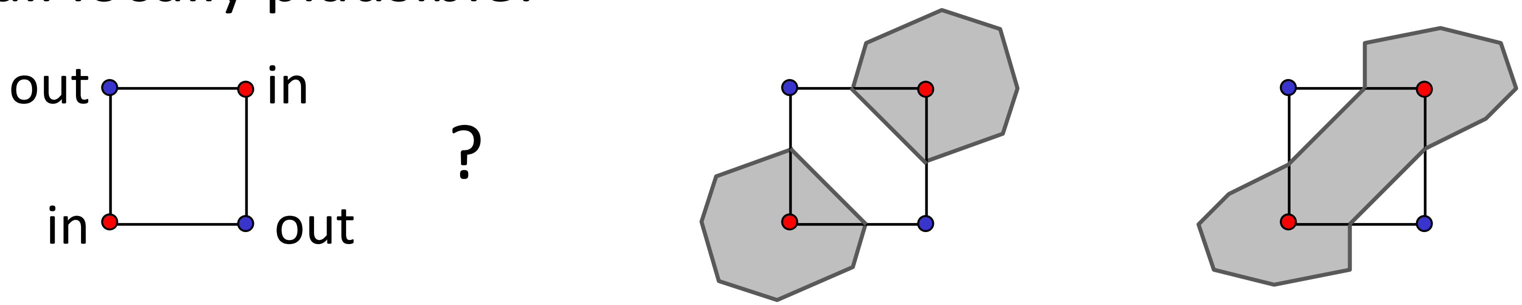
Marching Squares:

- There is also a 2D version of the algorithm, called marching squares.
- Same idea, but fewer cases.

Ambiguities

There is a (minor) technical problem remaining:

- The triangulation can be ambiguous
- In some cases, different topologies are possible which are all locally plausible:



- This is an *undersampling artifact*. At a sufficiently high resolution, this cannot occur.
- Problem: Inconsistent application can lead to holes in the surface (non-manifold solutions)

Ambiguities

Solution:

- Always use the same solution pattern in ambiguous situations
- For example: Always *connect* diagonally.
  - This might yield topologically wrong results.
  - But the surface is guaranteed to be a triangulated 2-manifold without holes and with well defined interior / exterior
- Better solution:
  - Use higher resolution sampling (if possible)
- All of this (problem and solutions) also applies to the 3D case.

MC Variations

Empty space skipping:

- Marching cube uses an n^3 voxel grid, which can become pretty expensive.
- The surface intersects typically only $O(n^2)$ voxels.
- If we roughly know where the surface might appear, we can restrict the execution of the algorithm (and the evaluations of f at the corners) to a narrow band around the surface.
- Example: Particle methods – only extract within the support of the radial basis functions.

MC Variations

Hierarchical marching cubes algorithm:

- One can use a hierarchical version of the marching cubes algorithm using a balanced octree instead of a regular grid
  - We need some refinement criterion to judge on where to subdivide
  - This is application dependent (depends on the definition of f).
- However, we obtain many more cases to consider (which is painful to derive)

Simple solution (common in practice):

- Extract high-resolution triangle mesh
- Then run mesh simplification (slower, but better quality)

Particle-Based Extraction

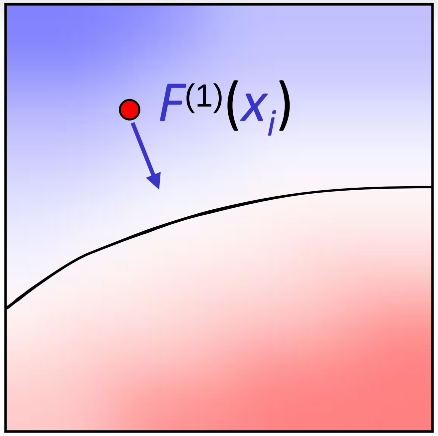
Particle-based method:

- This technique creates a set of points as output, which cover the iso-surface.
- Algorithm:
  - Start with a random point cloud (n points in a bounding volume)
  - Now define forces that attract particles to the zero-level set.
  - Also add some (weak) tangential repulsion to make them distribute uniformly

Forces

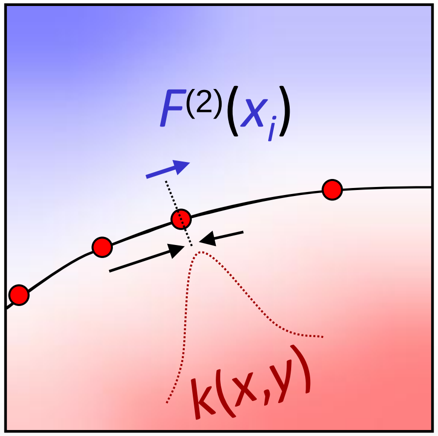
Attraction "force":

$$F^{(1)}(x_i) = m_i \nabla \left\| f(\mathbf{x}_i) \right\|^2$$



Tangential repulsion force:

$$F^{(2)}(\mathbf{x}_i) = \left(\sum_{j \neq i} k(\mathbf{x}_i, \mathbf{x}_j) \frac{\mathbf{x}_i - \mathbf{x}_j}{\left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2} \right) \left(\mathbf{I} - \left[\frac{\nabla f(\mathbf{x}_i)}{\left\| \nabla f(\mathbf{x}_i) \right\|} \right] \cdot \left[\frac{\nabla f(\mathbf{x}_i)}{\left\| \nabla f(\mathbf{x}_i) \right\|} \right]^T \right)$$



Solution

Solution:

- We obtain a system of ordinary differential equations
- The ODE can be solved numerically
- Simplest technique: gradient decent (explicit Euler)
  - Move every point by a fraction of the force vector
  - Recalculate forces
  - Iterate
- We have the solution if the system reaches a steady state (nothing moves anymore, numerically)

Implicit Surfaces

Solid Modeling

Solid Modeling

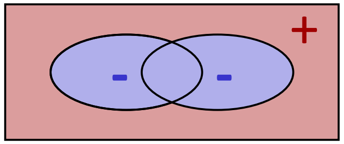
We want to:

- Form basic volumetric primitives (spheres, cubes, cylinders) as implicit functions (this is easy, no details).
- Compute Boolean combinations of these primitives: Intersection, union, etc...
- Derive an implicit function from these operations

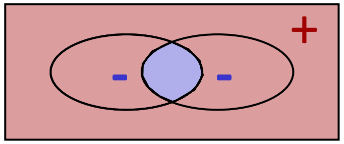
Boolean Operations

Actually, Boolean operations with implicit functions are simple:

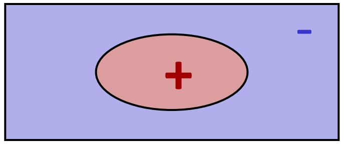
- Given two signed implicit functions (negative inside) f_A, f_B for objects A, B.
- The boolean combinations are given by:
  - Union $A \cup B$: $f_{A \cup B} = \min(f_A, f_B)$



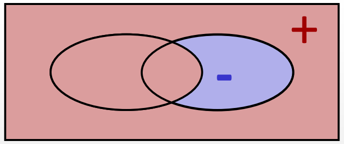
  - Intersection $A \cap B$: $f_{A \cap B} = \max(f_A, f_B)$



  - Complement $\neg A$: $f_{\neg A} = -f_A$



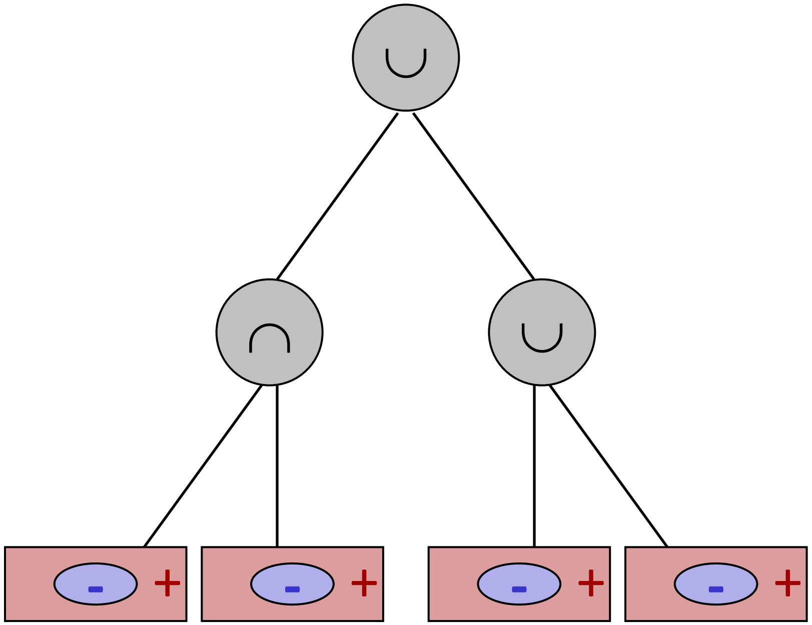
  - Difference $A \setminus B$: $f_{A \setminus B} = \max(f_A, -f_B)$



Hierarchical Modeling

This can be models as a CSG tree (constructive solid geometry):

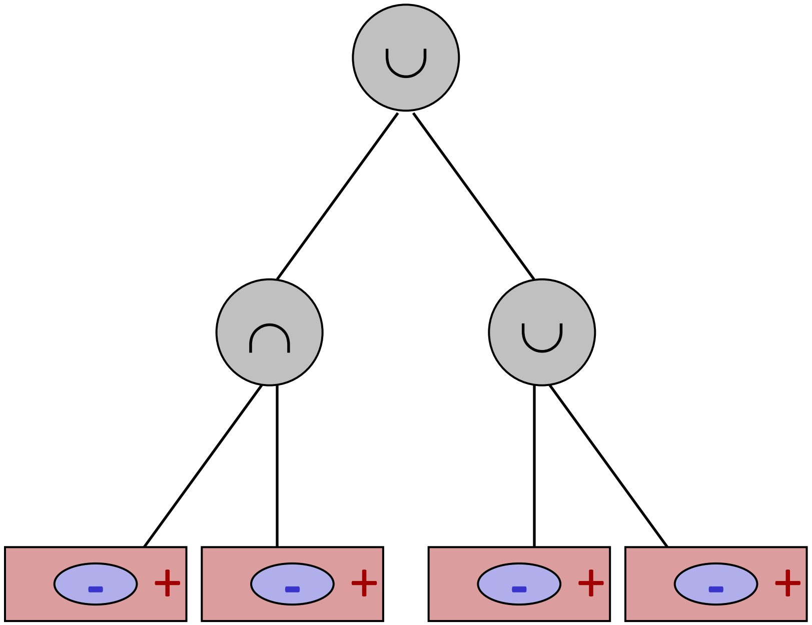
- Leaf nodes are signed distance functions
- Inner nodes are Boolean operations
- Evaluation translates to an arithmetic expression
- Other operations:
  - Deformation (apply vector field)
  - Blending (combine surfaces smoothly)



Hierarchical Modeling

Rendering CSG hierarchies:

- Rendering is simple
- We get one compound signed implicit function
- We can extract the surface using marching cubes
- We can raytrace the surface using a numerical root finding algorithm
  - For example:
    Newton scheme with voxel-based intialization



Implicit Surfaces

Data Fitting

Constructing Implicit Surfaces

Question: How to construct implicit surfaces?

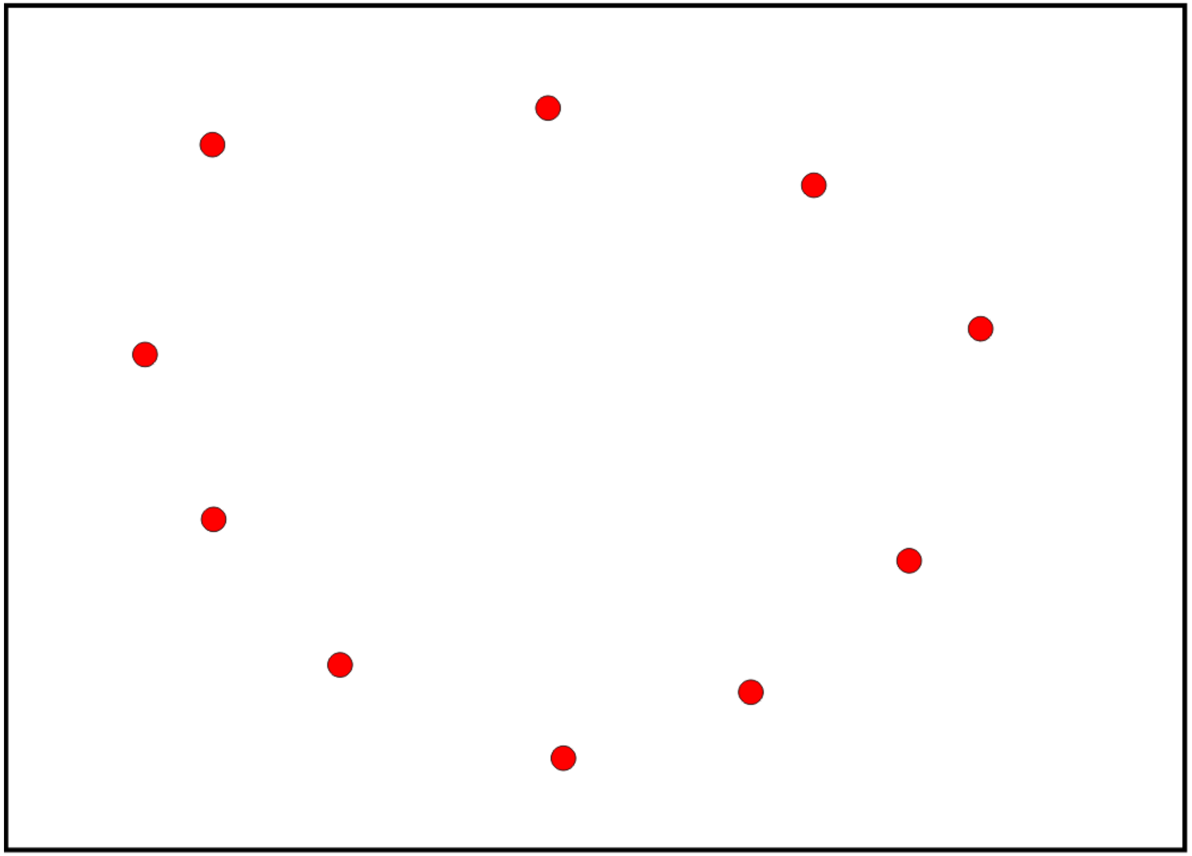
- Basic primitives: Spheres, boxes etc... are (almost) trivial.
- We can construct implicit spline schemes by using 3D tensor product (or tetrahedral) constructions of 3D Bezier or B-Spline functions
- Another option: Variational modeling (next lecture)
- In this chapter of this lecture: Fitting to data

Data Fitting

Data Fitting Problem:

- We are given a set of points
- We want to find an implicit surface that interpolates or approximates these points
- This problem is ill-defined
- We need additional assumptions to make it well-defined
- We will look at three variants:
  - Hoppe’s method / plane blending
  - Thin-plate spline data matching
  - MPU Implicits (multi-level partition of unity implicits)

Plane Blending Method



Initial data

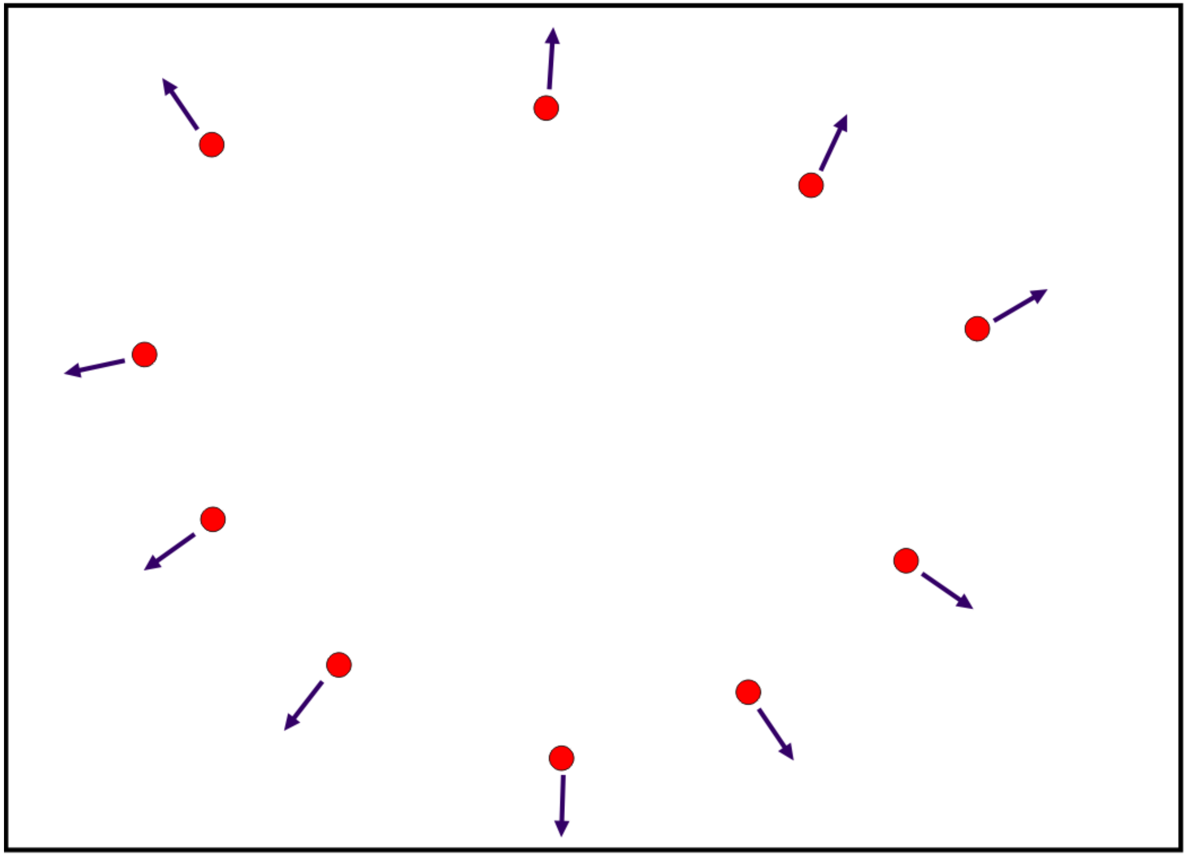
Estimate normals

Signed distance func.

Marching cubes

Final mesh

Plane Blending Method



unoriented normals:
total least squares plane fit (PCA)
in a *k*-nearest neighbors neighborhood

Initial data

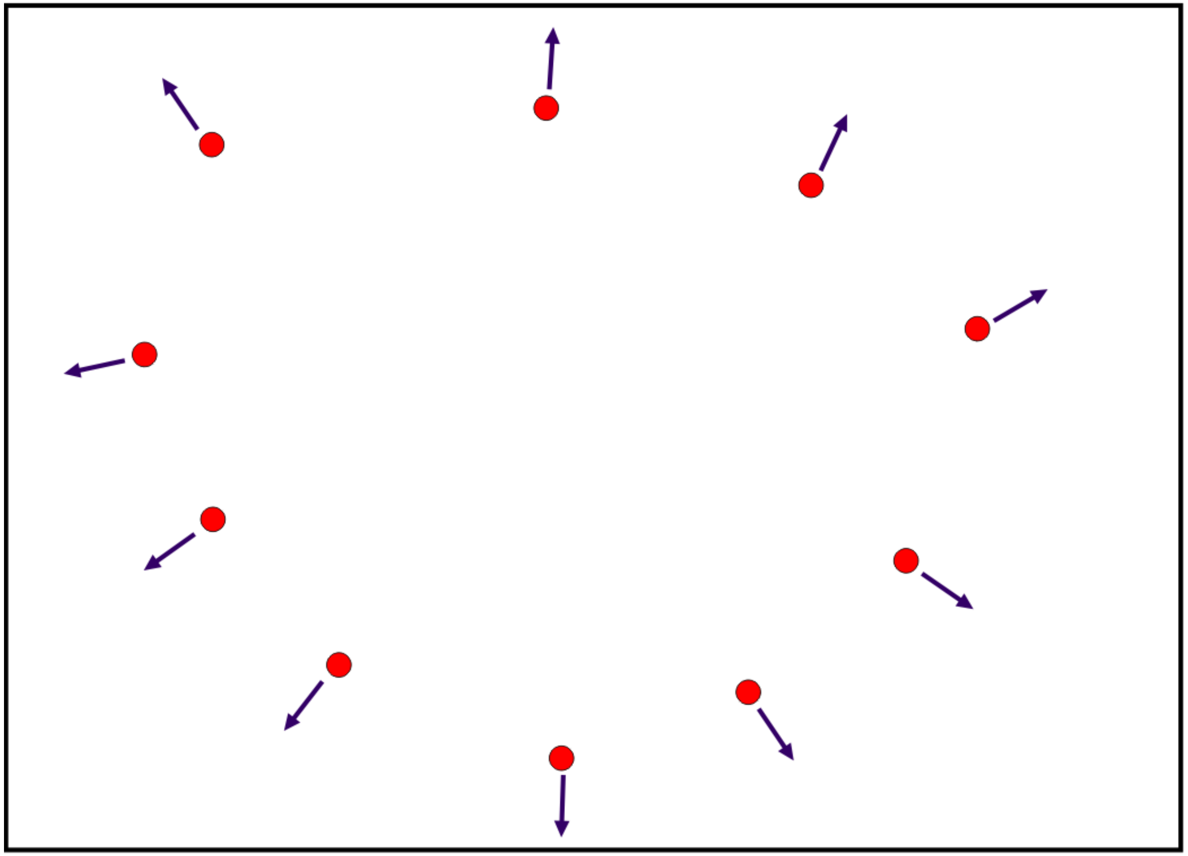
Estimate normals

Signed distance func.

Marching cubes

Final mesh

Plane Blending Method



consistent orientation:
region growing, flip normals if angle > 180°,
pick most similar normal next in each step

Initial data

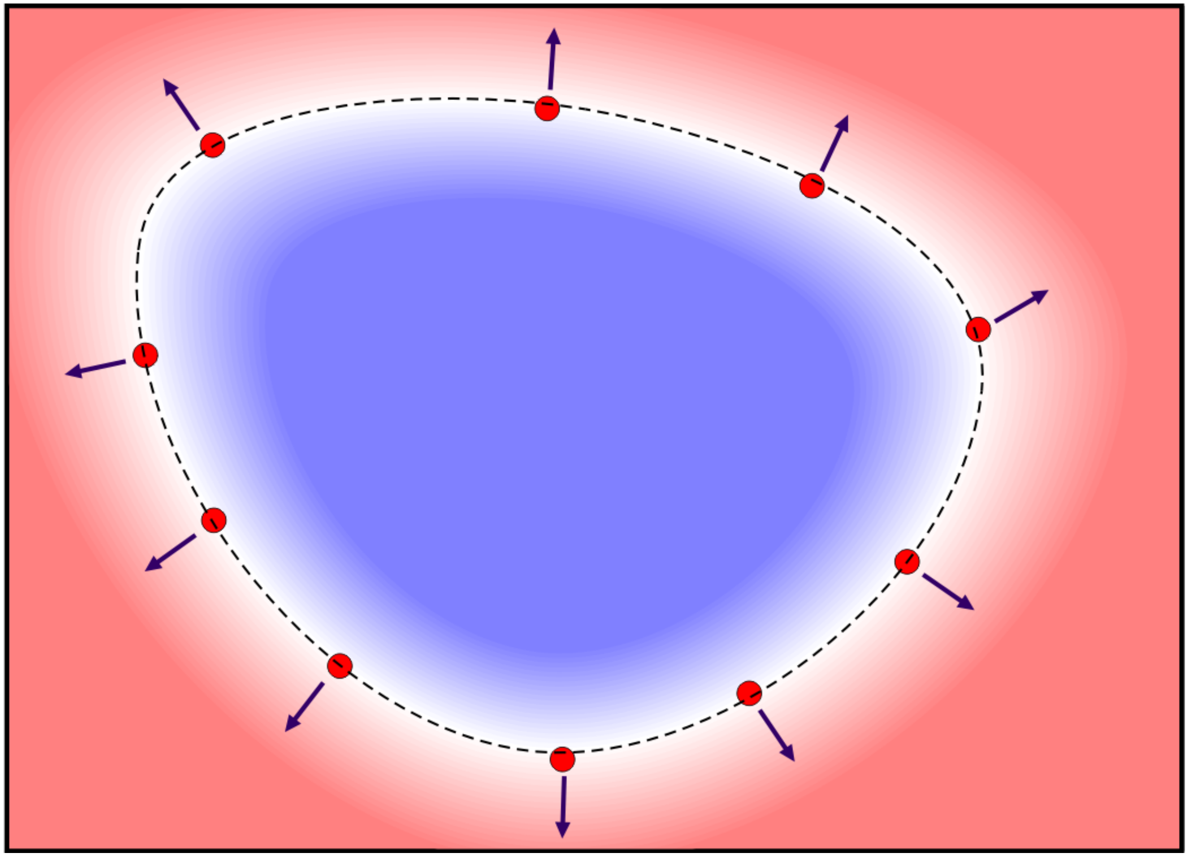
Estimate normals

Signed distance func.

Marching cubes

Final mesh

Plane Blending Method



consistent orientation:
blend between signed distance functions of planes associated with each point

Initial data

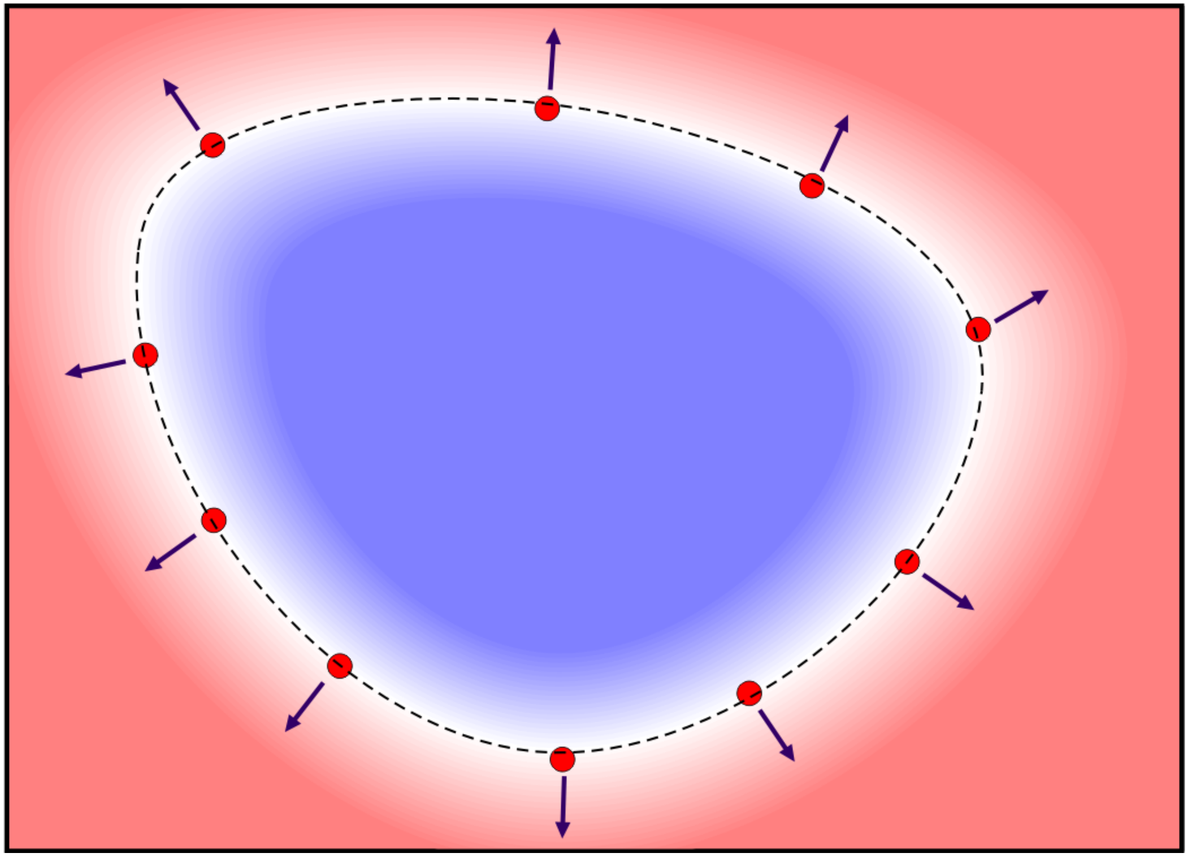
Estimate normals

Signed distance func.

Marching cubes

Final mesh

Plane Blending Method



signed distance function:
plane blending (next slide)

Initial data

Estimate normals

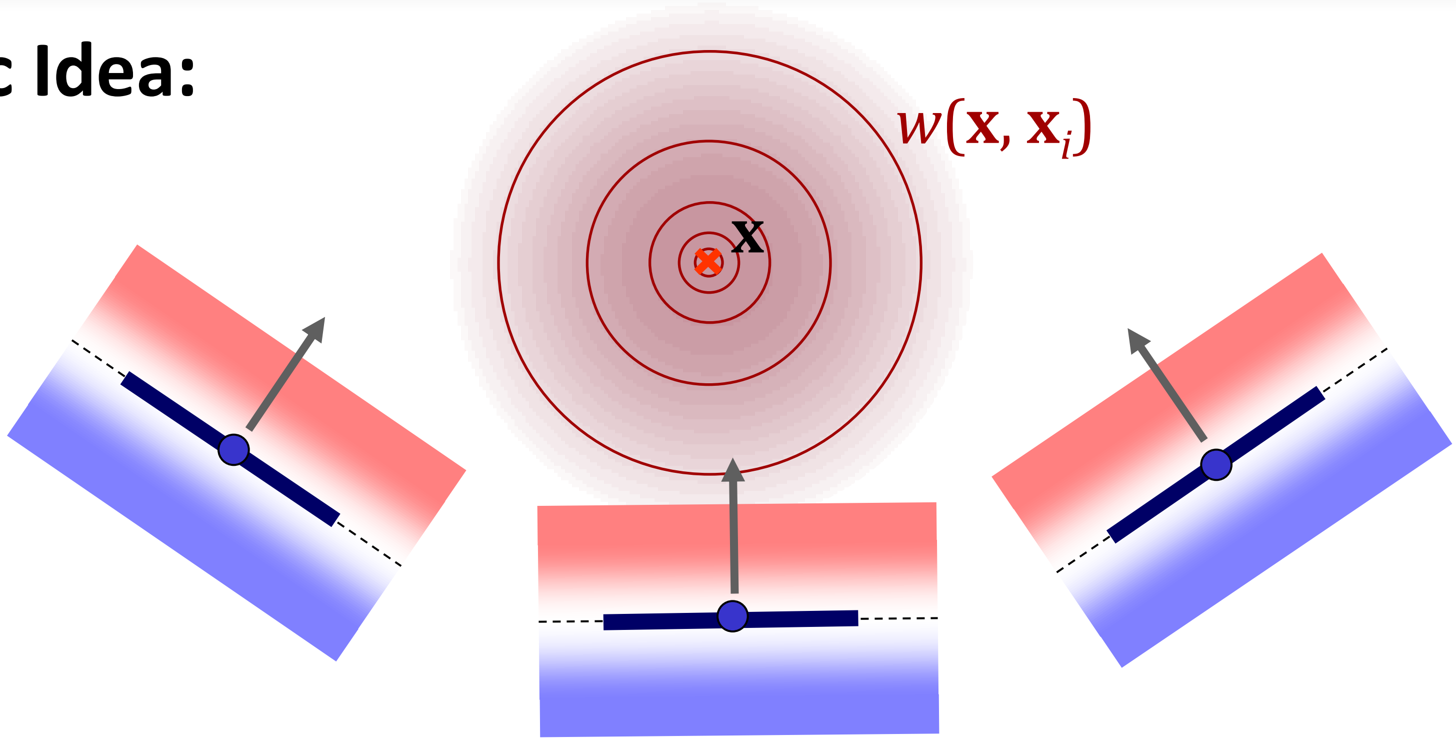
Signed distance func.

Marching cubes

Final mesh

Normal Constraints

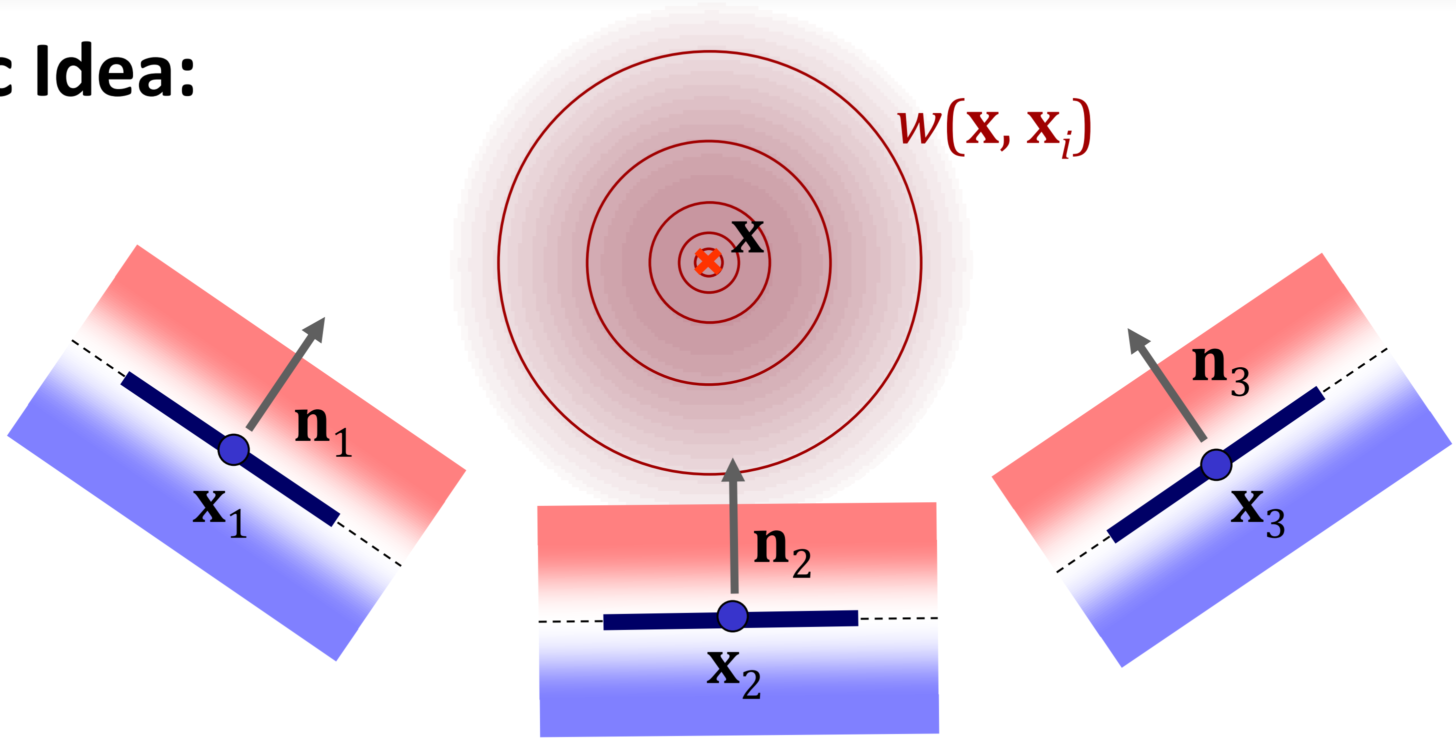
Basic Idea:



- Each point defines an oriented plane and a signed distance function
- To obtain a composite distance field in space:
  Blend these distance functions with weights from a kernel function (Gaussian, or uniform B-Spline)

Normal Constraints

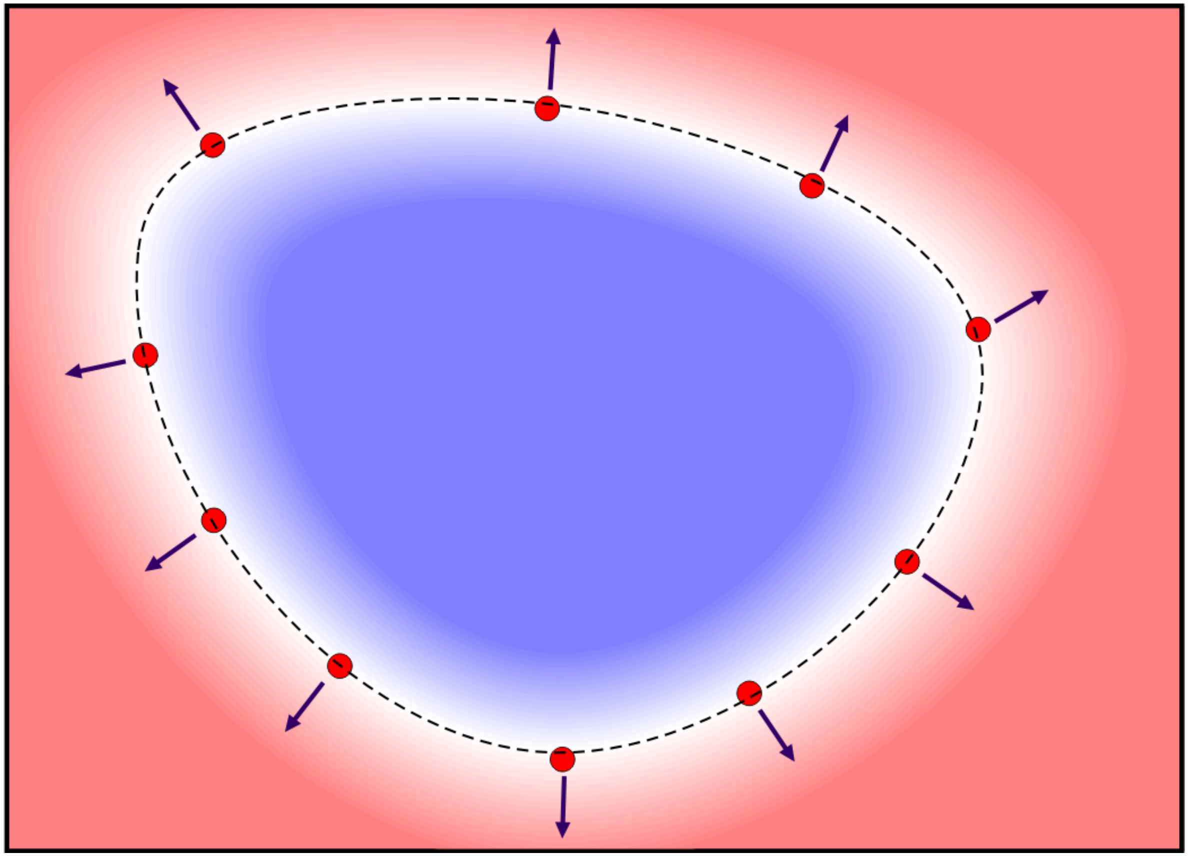
Basic Idea:



$$f(\mathbf{x}) = \frac{\sum_{i=1}^{n} \langle \mathbf{n}_i, \mathbf{x} - \mathbf{x}_i \rangle w(\|\mathbf{x} - \mathbf{x}_i\|)}{\sum_{i=1}^{n} w(\|\mathbf{x} - \mathbf{x}_i\|)}$$

(partition of unity weights)

Plane Blending Method



Initial data

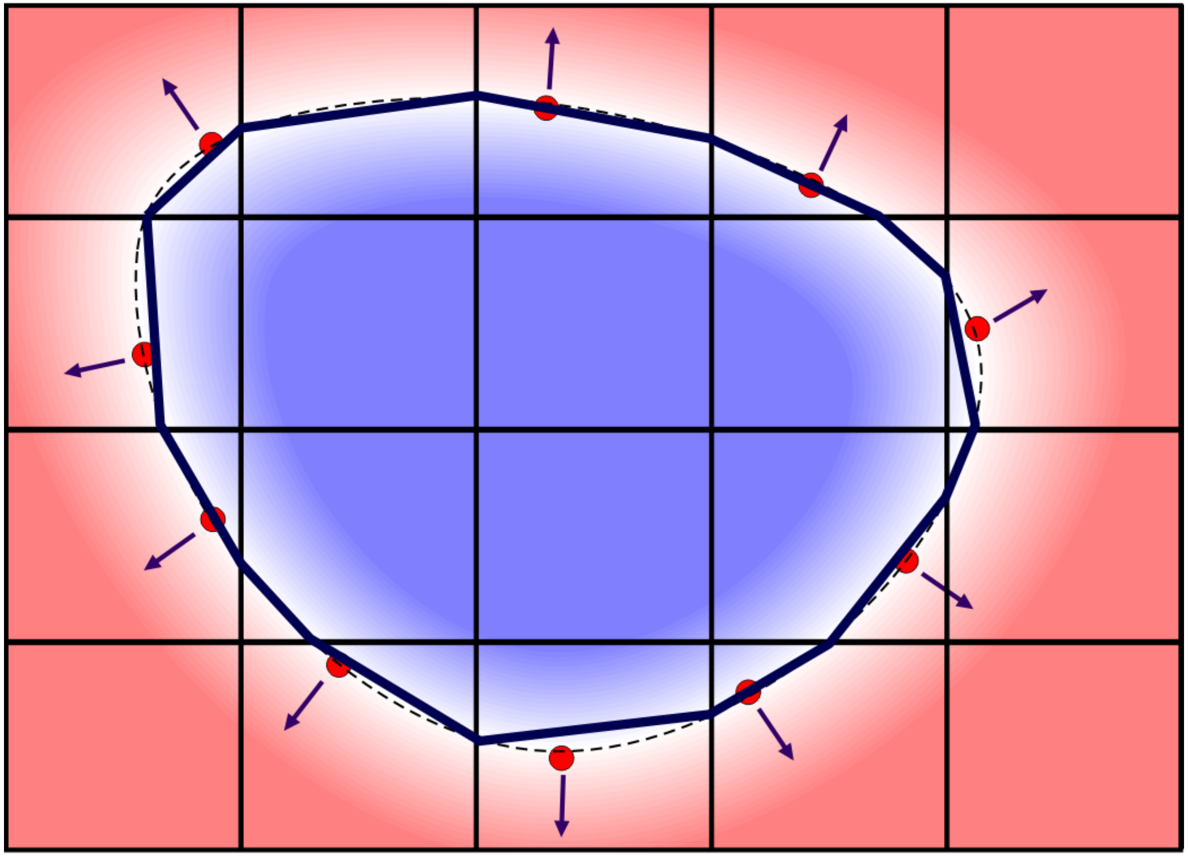
Estimate normals

Signed distance func.

Marching cubes

Final mesh

Plane Blending Method



Initial data

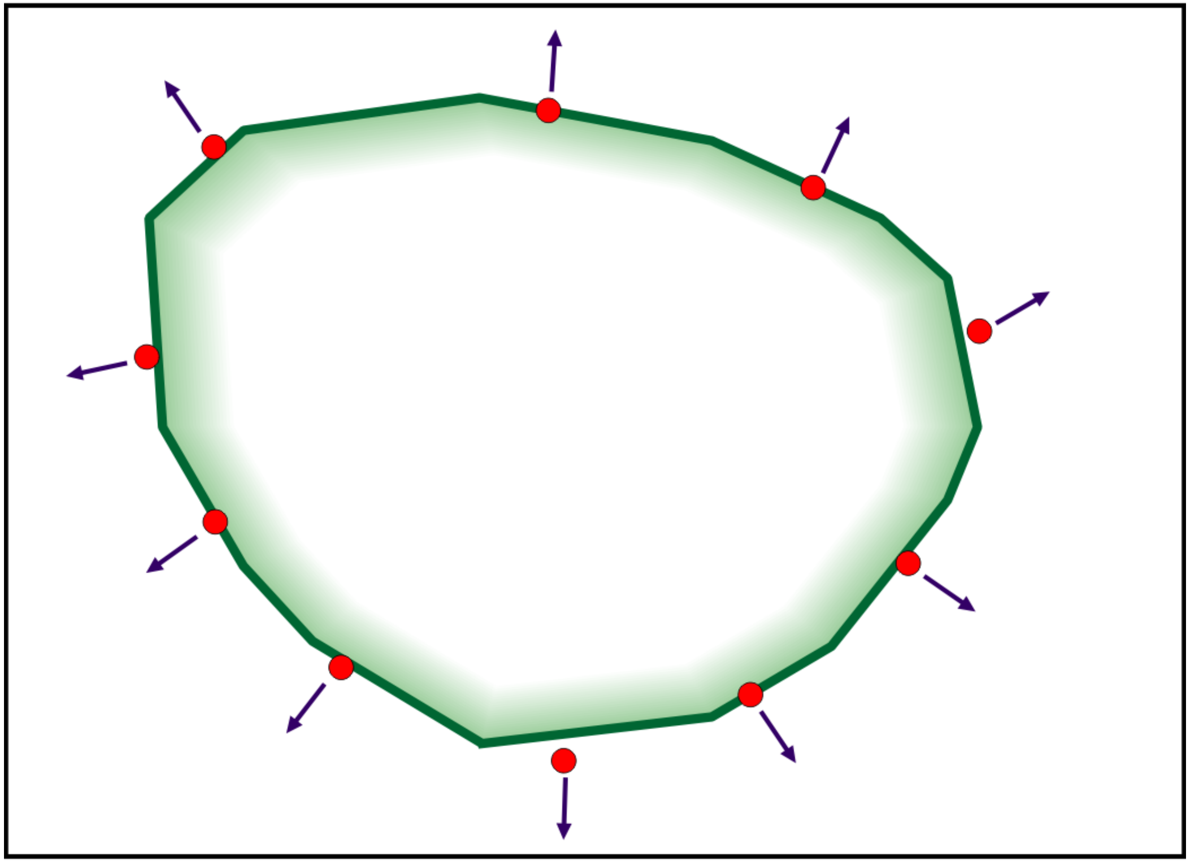
Estimate normals

Signed distance func.

Marching cubes

Final mesh

Plane Blending Method



Initial data

Estimate normals

Signed distance func.

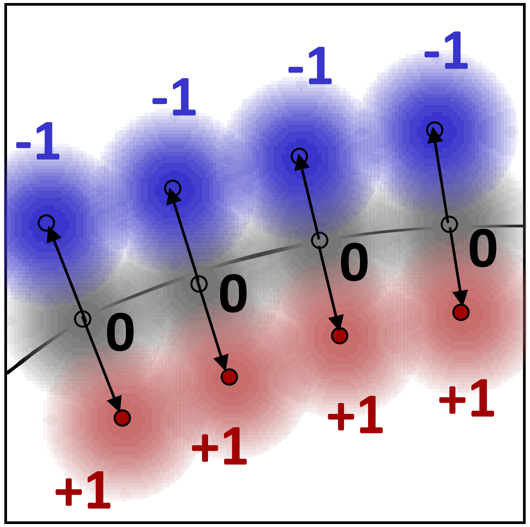
Marching cubes

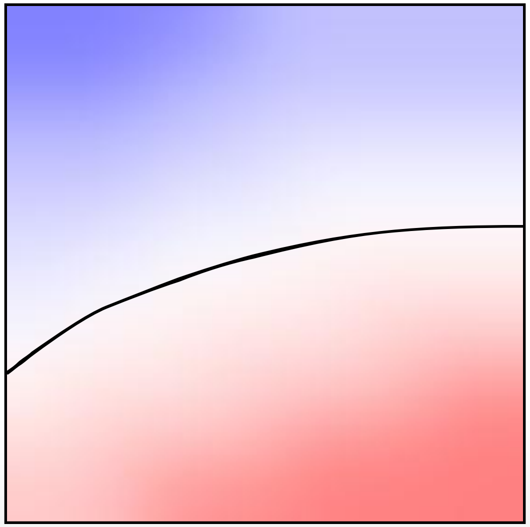
Final mesh

Thin-Plate Spline Data Matching

Agenda:

- Use radial basis functions
- Use a globally supported basis that guarantees smoothness
- Place radial basis functions at the input points
- Place two more in normal and negative normal direction
- Prescribe values +1,0,-1
- Solve a linear system to meet these constraints





Types of Radial Basis Functions

Typical choices for radial basis functions:

- Globally supported functions:
  - Thin splate spline basis functions:
    $\|x - x_0\|^2 \ln\|x - x_0\|$ (2D), $\|x - x_0\|^3$ (3D).
  - These functions guarantee minimal integral second derivatives.
- Problem: evaluation
  - Every basis function interacts with each other one
  - This creates a dense $n \times n$ linear system
  - One can use a fast multi pole method that clusters far away nodes in bigger octree boxes
  - This gives O(log n) interactions per particle, overall O(n log n) interactions.

Alternative

Alternative:

- Use locally supported basis functions (e.g. B-Splines)
- Employ an additional regularization term to make the solution smooth.
- Optimize the energy function

$$E(\boldsymbol{\lambda}) = \sum_{i=1}^{n} f(\mathbf{x}_i)^2 + \mu \int_{\Omega} \left(\left[\frac{\partial^2}{\partial x} + \frac{\partial^2}{\partial y} + \frac{\partial^2}{\partial z} + \frac{2\partial^2}{\partial x \partial y} + \frac{2\partial^2}{\partial y \partial z} + \frac{2\partial^2}{\partial x \partial z} \right] f(\mathbf{x}) \right)^2 d\mathbf{x}$$

with

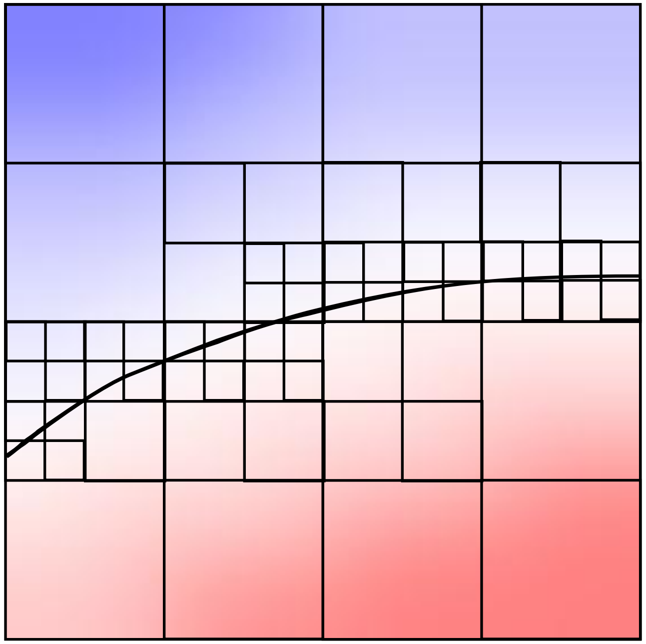
$$f(\mathbf{x}) = \sum_{j=1}^{m} \lambda_j b(\mathbf{x} - \mathbf{x}_j)$$

- The crictical point is the solution to a linear system

MPU Implicits

Multi-level partition of unity implicits:

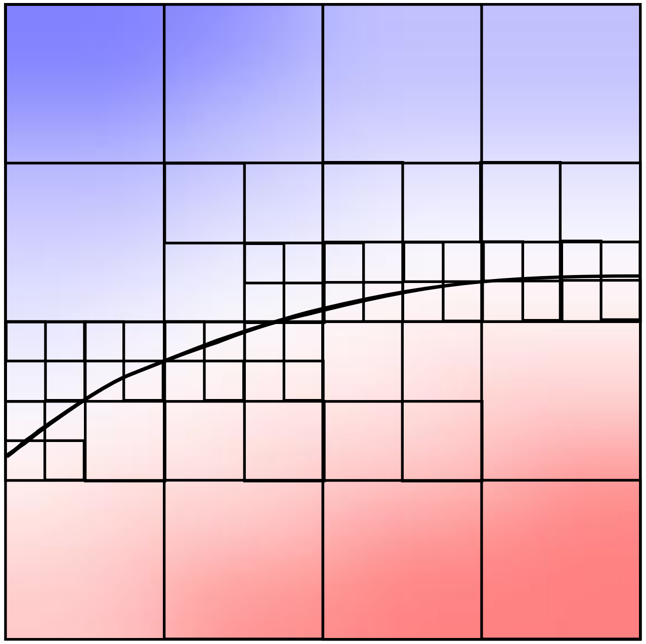
- Hierarchical implicit function approximation
  - Given: data points with normals
  - Computes: hierarchical approximation of the signed distance function



MPU Implicits

Multi-level partition of unity implicits:

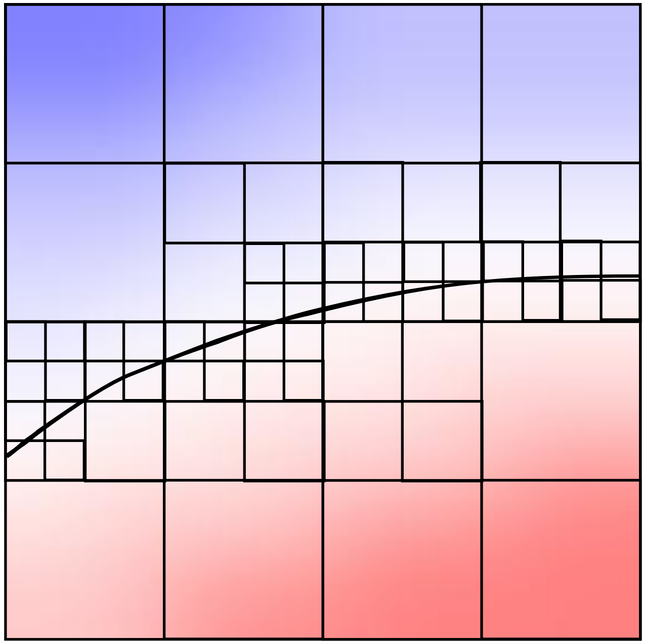
- Octree decomposition of space
- In each octree cell, fit an implicit quadratic function to points
  - $f(x_i) = 0$ at data points
  - Additional normal constraints
- Stopping criterion:
  - Sufficient approximation accuracy (evaluate f at data points to calulate distance)
  - At least 15 points per cell.



MPU Implicits

Multi-level partition of unity implicits:

- This gives an adaptive grid of local implicit function approximations.
- Problem: How to define a global implicit function?
- Idea: Just blend between local approximants using a windowing function

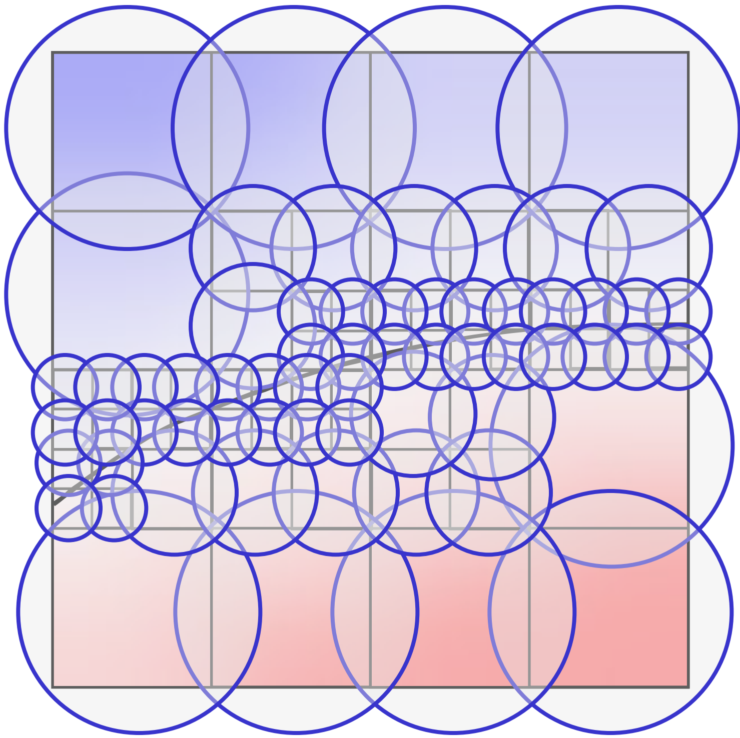


MPU Implicits

Multi-level partition of unity implicits:

- Windowing function:
  - Use smooth windowing function *w*
    - B-splines / normal distribution
    - original formulation: quadratic tensor product B-spline function, support = 1.5× cell diagonal
  - Renormalize to form partition of unity:

$$f(\mathbf{x}) = \frac{\sum_{i=1}^{n} w(\mathbf{x}-\mathbf{x}_i) f_i(\mathbf{x})}{\sum_{i=1}^{n} w(\mathbf{x}-\mathbf{x}_i)}$$



MPU Implicits

Multi-level partition of unity implicits:

- Sharp features:
  - If a leaf cell with a few points has strongly varying normals, this might be a sharp feature.
  - Multiple functions can be fitted to parts of the data
  - Boolean operations to obtain composite distance field