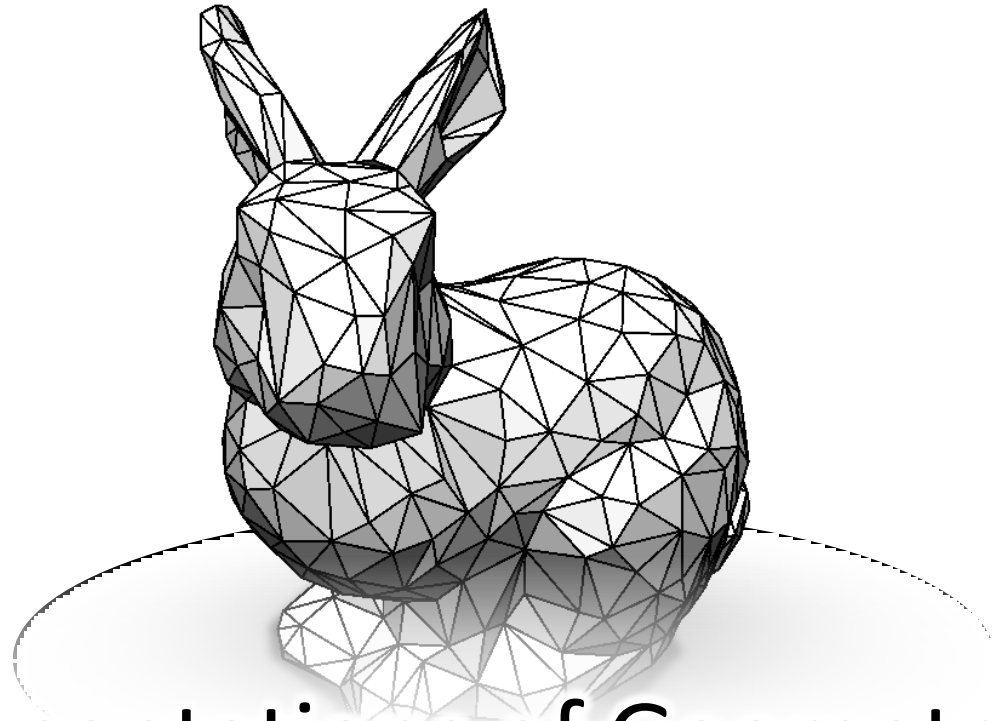


Statistical Geometry Processing

Winter Semester 2011/2012

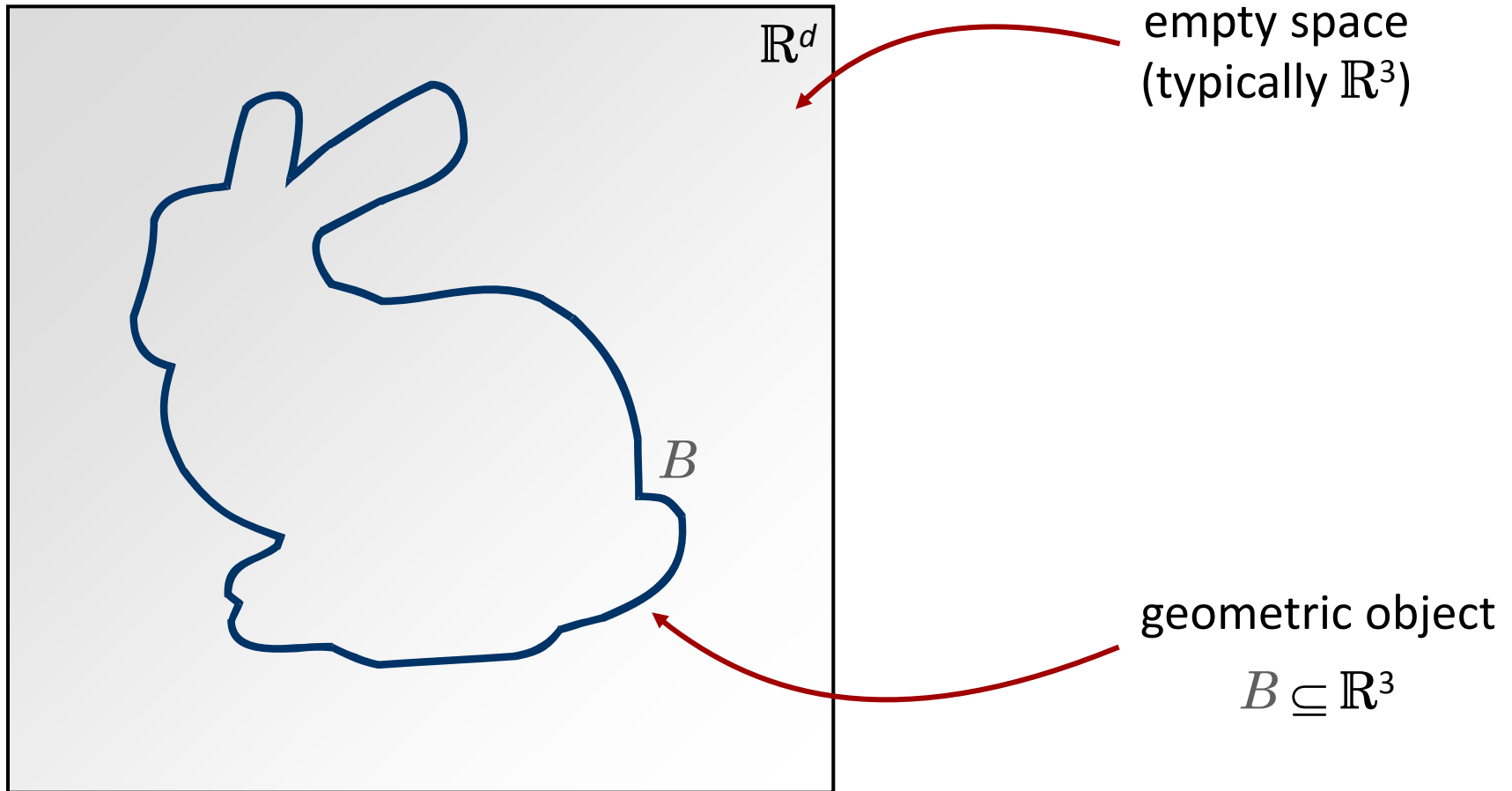


Representations of Geometry

Motivation

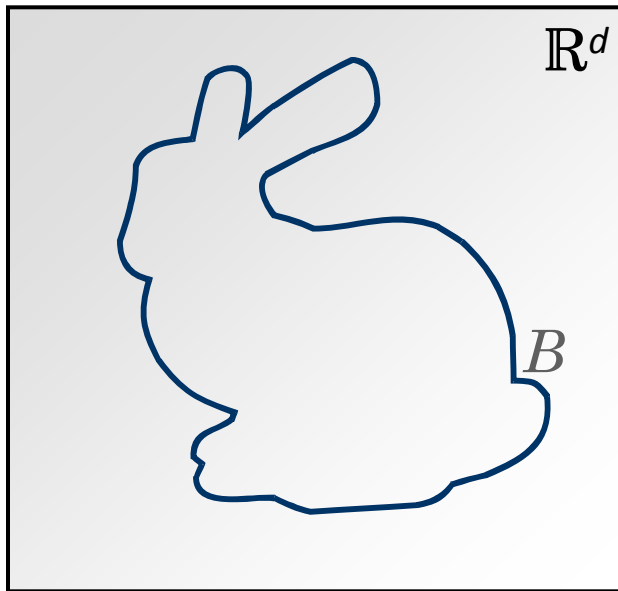
Geometric Modeling

What do we want to do?

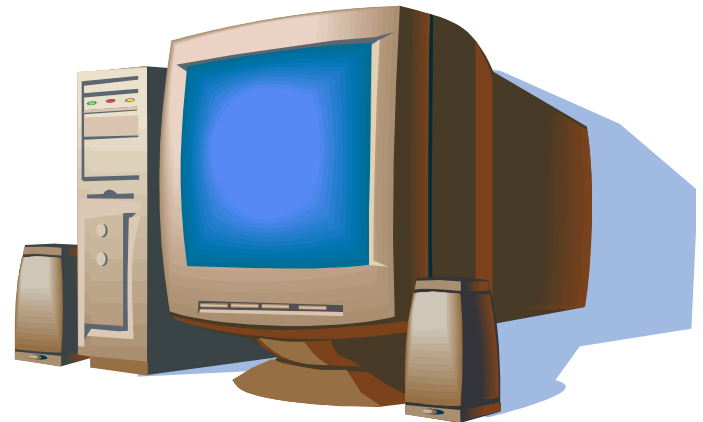


Fundamental Problem

The Problem:



infinite number of points



my computer: 8GB of memory

We need to encode a continuous model with a finite amount of information

Modeling Approaches

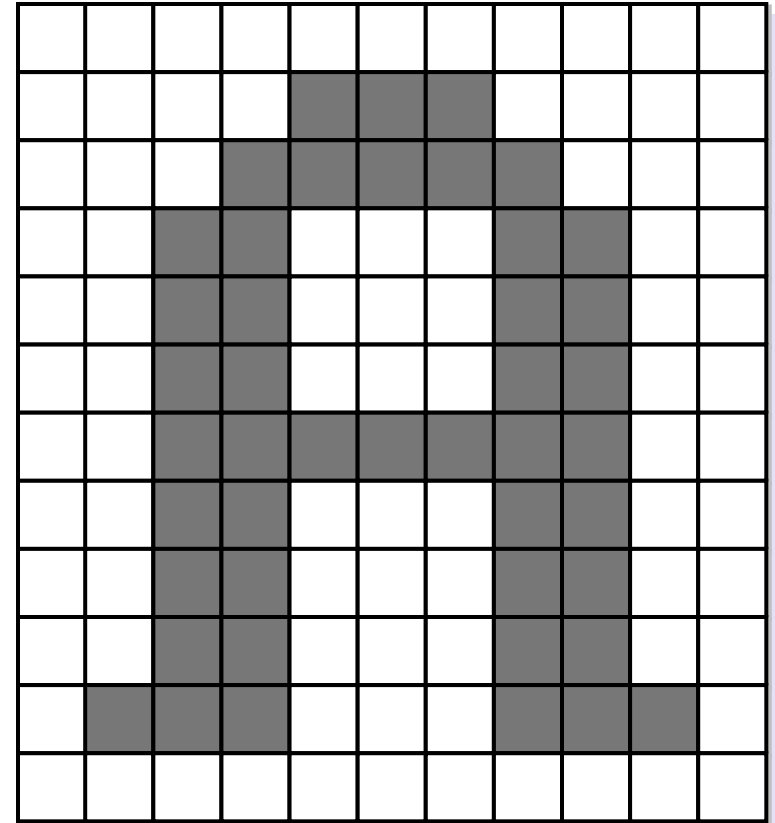
Two Basic Approaches

- Discrete representations
 - Fixed discrete bins
- “Continuous” representations
 - Mathematical description
 - Evaluate continuously

Discrete Representations

You know this...

- Fixed Grid of values:
 $(i_1, \dots, i_{d_s}) \in \mathbb{Z}^{d_s} \rightarrow (x_1, \dots, x_{d_t}) \in \mathbb{R}^{d_t}$
- Typical scenarios:
 - $d_s = 2, d_t = 3$: Bitmap images
 - $d_s = 3, d_t = 1$: Volume data (scalar fields)
 - $d_s = 2, d_t = 1$: Depth maps (range images)
- PDEs: “Finite Differences” models



Modeling Approaches

Two Basic Approaches

- Discrete representations
 - Fixed discrete bins
- “Continuous” representations
 - Mathematical description
 - Evaluate continuously

Classes of Models

Most frequently used models:

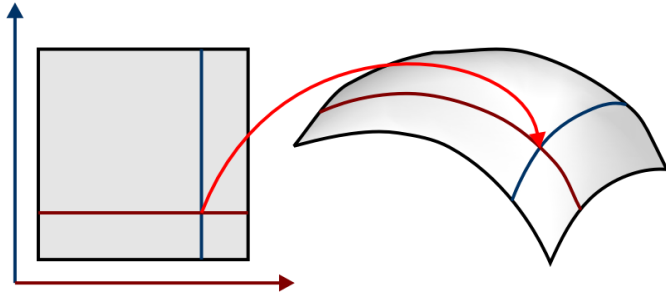
- Primitive meshes
- Parametric models
- Implicit models
- Particle / point-based models

Remarks

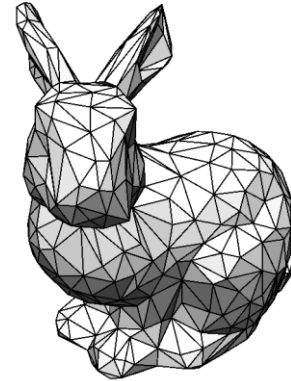
- Often combinations thereof: hybrid models
- Representations can be converted (may be approximate)
- Some questions are much easier to answer for certain representations

Modeling Zoo

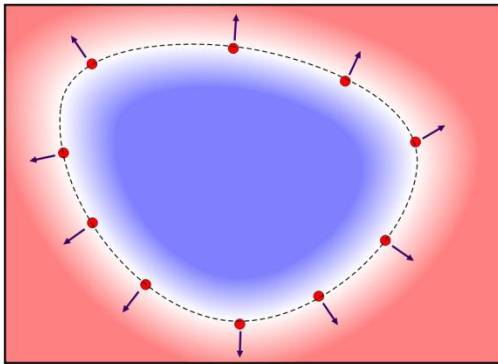
Modeling Zoo



Parametric Models



Primitive Meshes

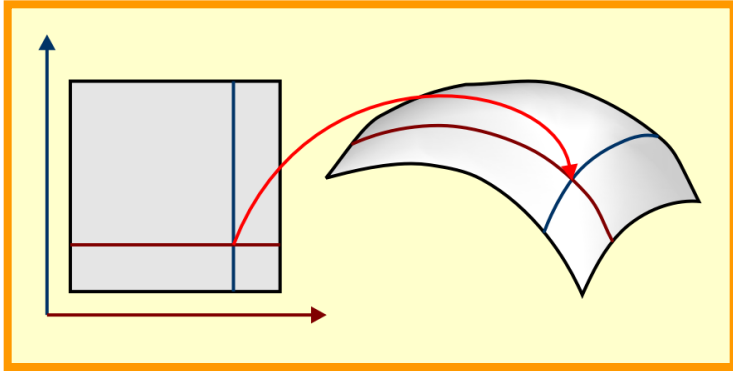


Implicit Models

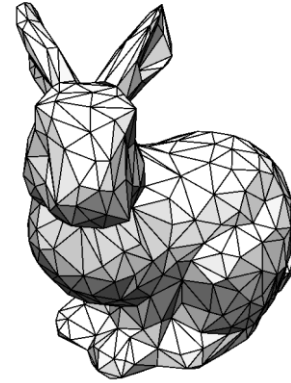


Point-Based Models

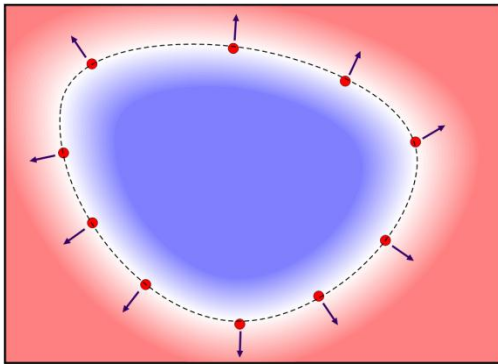
Modeling Zoo



Parametric Models



Primitive Meshes

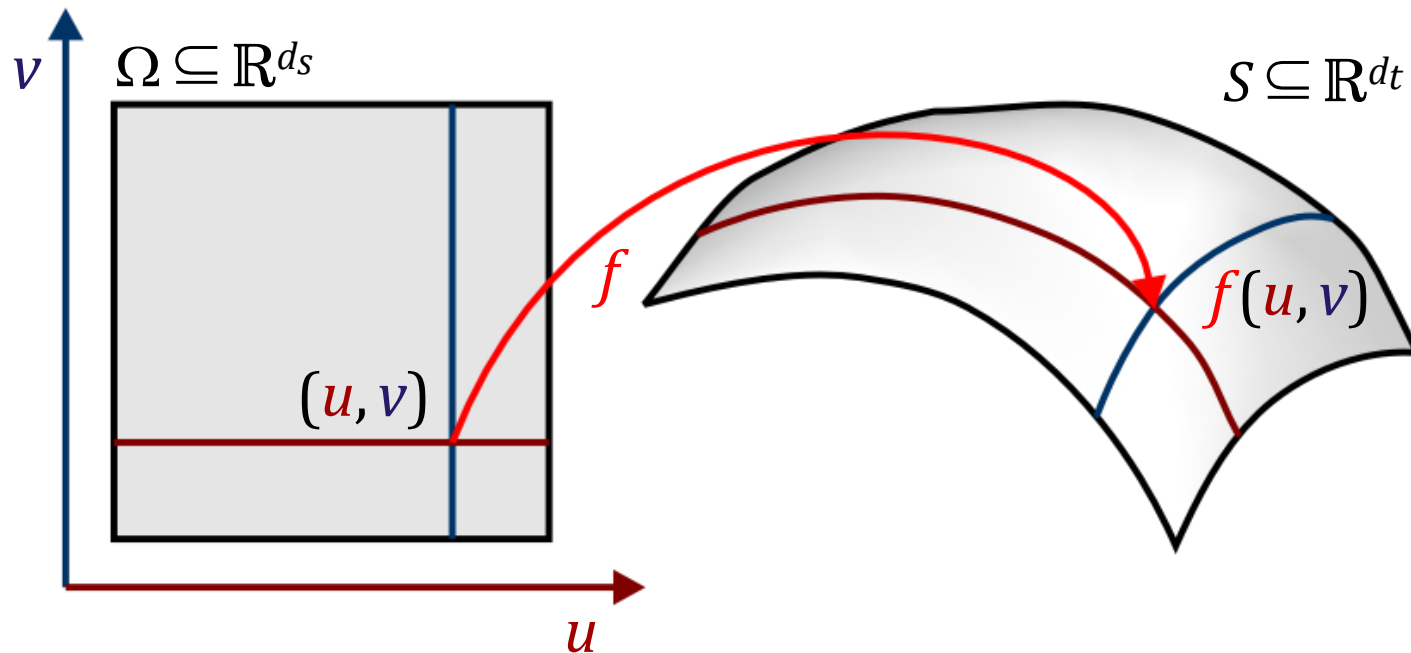


Implicit Models



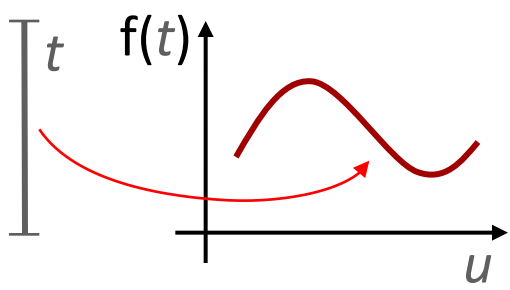
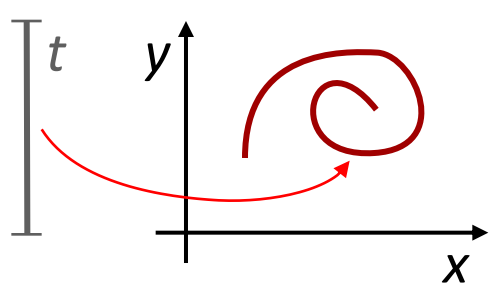
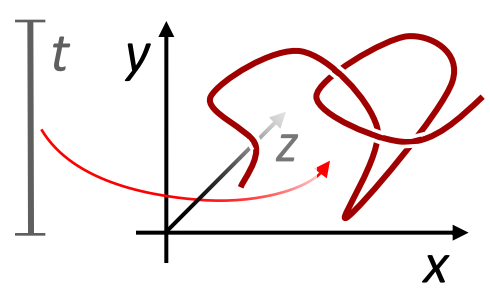
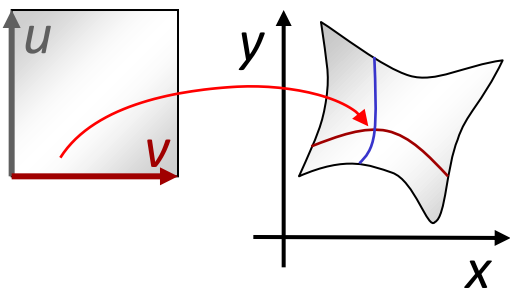
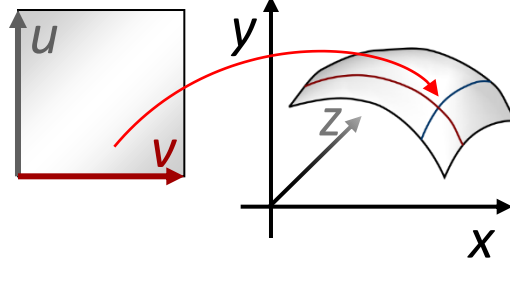
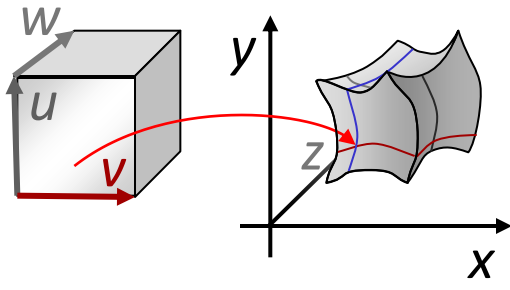
Point-Based Models

Parametric Models

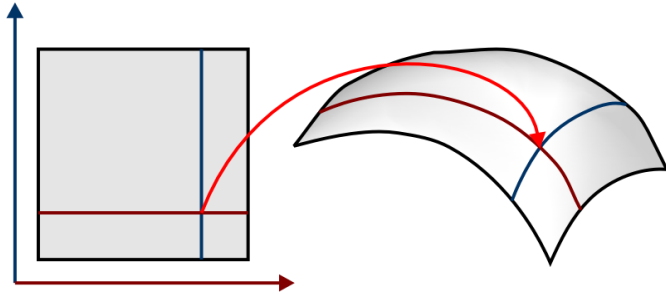


Parametric Models

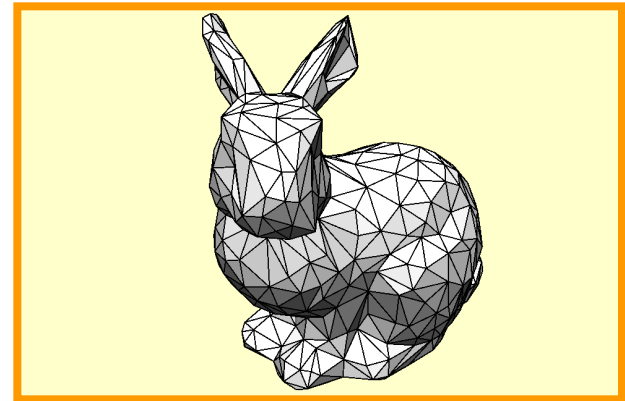
- Function f maps from parameter domain Ω to target space
- Evaluation of f gives one point on the model

	output: 1D	output: 2D	output: 3D
input: 1D	 <p>function graph</p>	 <p>plane curve</p>	 <p>space curve</p>
input: 2D		 <p>plane warp</p>	 <p>surface</p>
input: 3D			 <p>space warp</p>

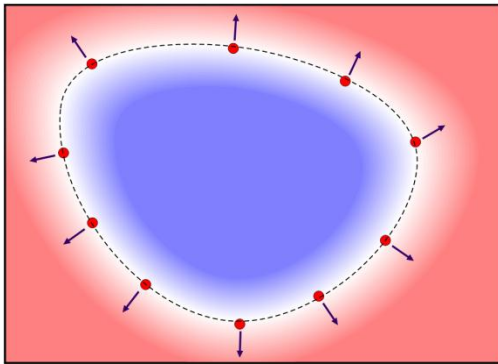
Modeling Zoo



Parametric Models



Primitive Meshes



Implicit Models

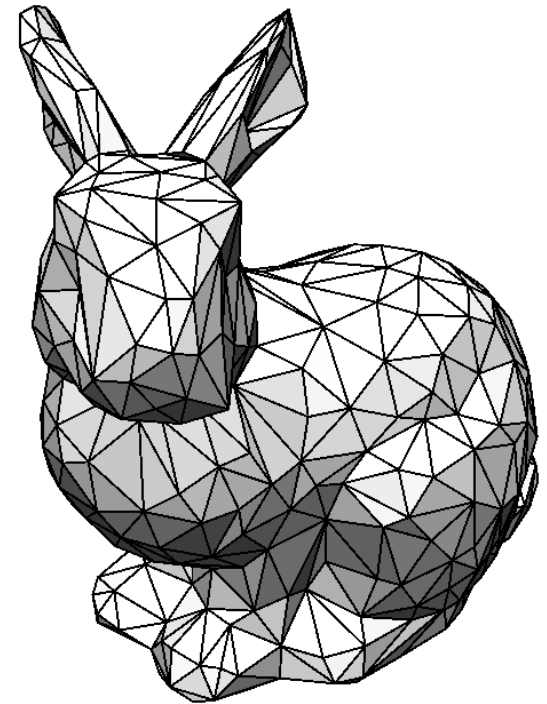


Point-Based Models

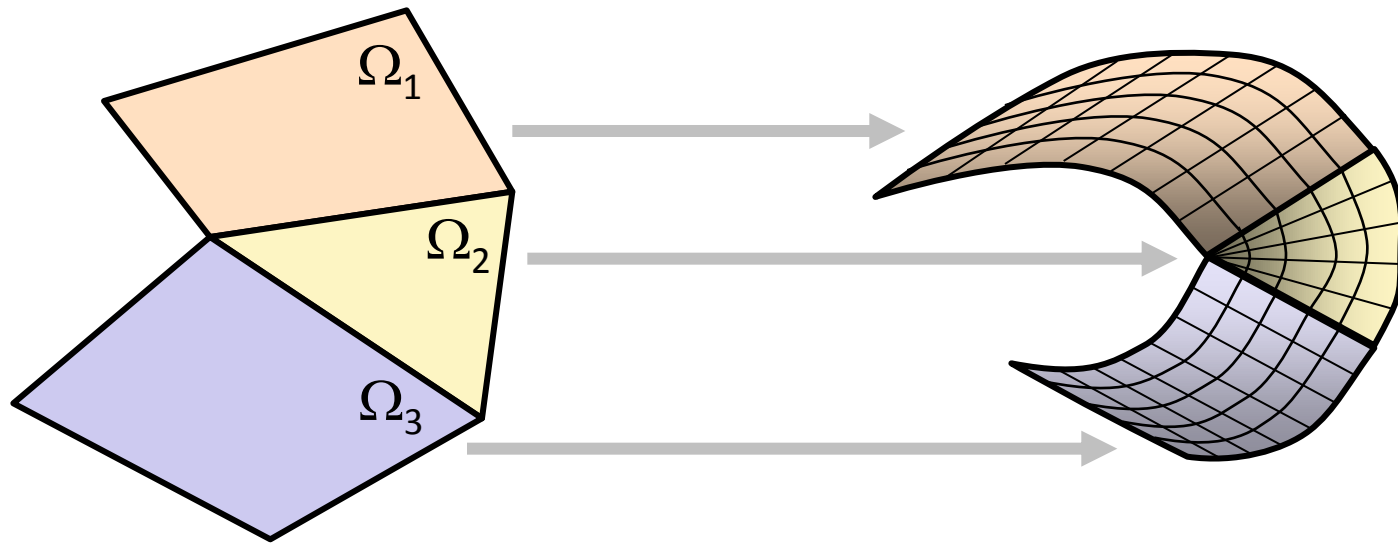
Primitive Meshes

Primitive Meshes

- Collection of geometric primitives
 - Triangles
 - Quadrilaterals
 - More general primitives (e.g. spline patches)
- Typically, primitives are parametric surfaces
- Composite model:
 - Mesh encodes topology, rough shape
 - Primitive parameter encode local geometry
- Triangle meshes rule the world (“triangle soup”)



Primitive Meshes



Complex Topology for Parametric Models

- Mesh of parameter domains attached in a mesh
- Domain can have complex shape (“trimmed patches”)
- Separate mapping function f for each part (typically of the same class)

Meshes are Great

Advantages of mesh-based modeling:

- Compact representation (usually)
- Can represent arbitrary topology

Meshes are not so great

Problem with Meshes:

- Need to specify a mesh first, then edit geometry
- Problems
 - Mesh structure need to be adjusted to fit shape
 - Mesh encodes object topology
⇒ Changing object topology is painful
- Examples
 - Surface reconstruction
 - Fluid simulation (surface of splashing water)

Triangle Meshes

Triangle Meshes

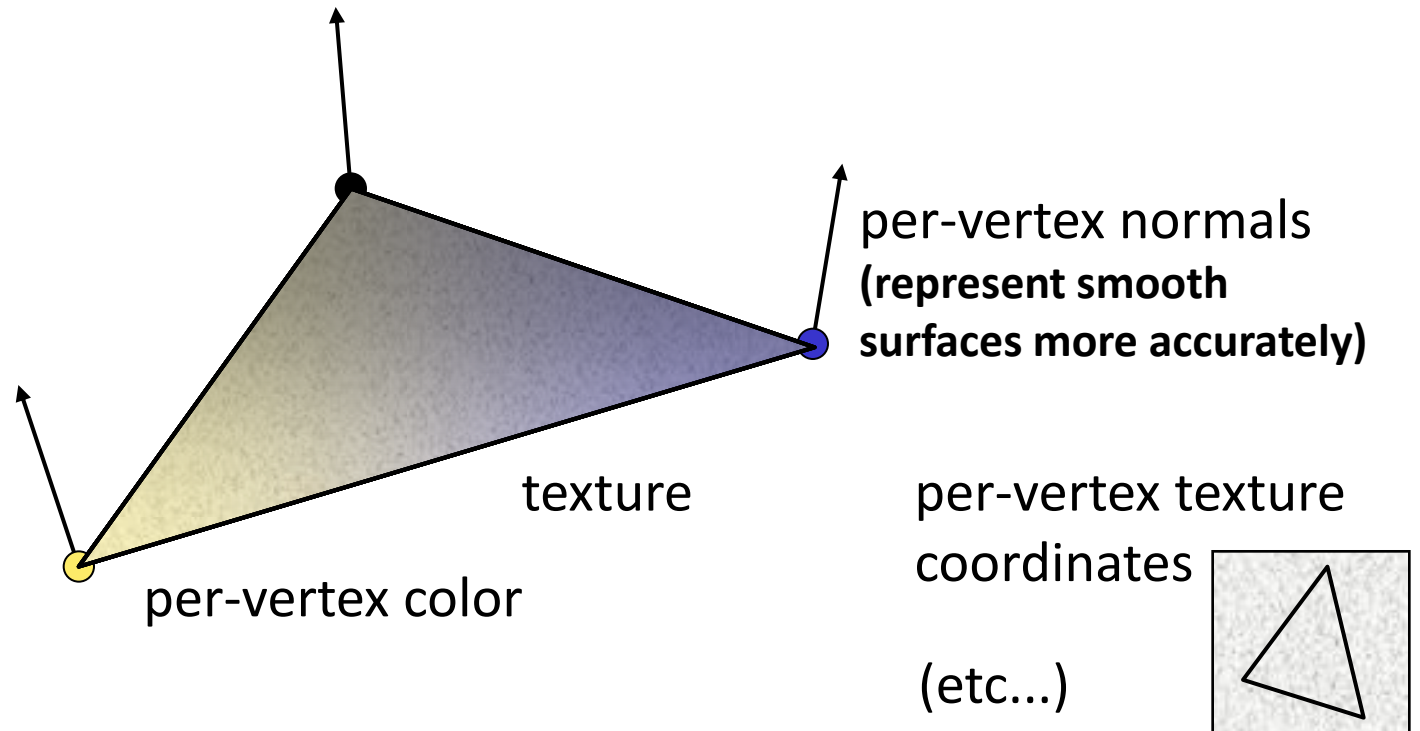
Triangle Meshes:

- Triangle meshes:
(probably) most common representation
- Simplest surface primitive
that can be assembled into meshes
 - Rendering in hardware (z-buffering)
 - Simple algorithms for intersections (raytracing, collisions)

Attributes

How to define a triangle?

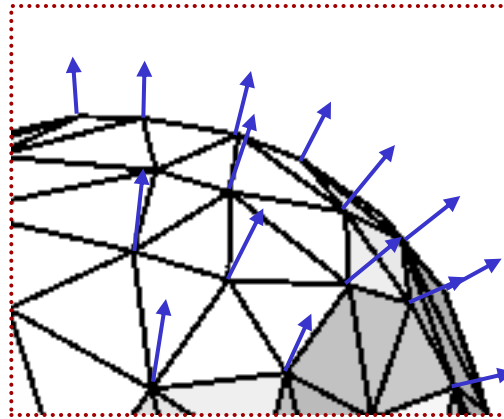
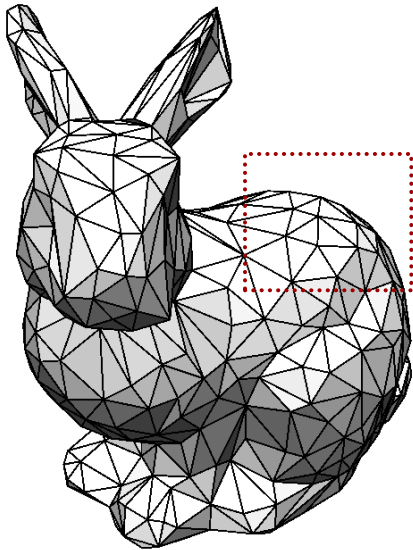
- We need three points in \mathbb{R}^3 (obviously).
- But we can have more:



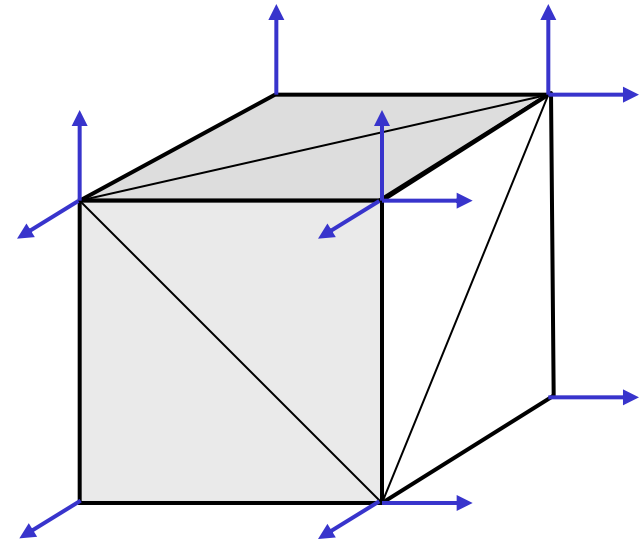
Shared Attributes in Meshes

In Triangle Meshes:

- Attributes might be shared or separated:



adjacent triangles
share normals



adjacent triangles
have separated normals

“Triangle Soup”

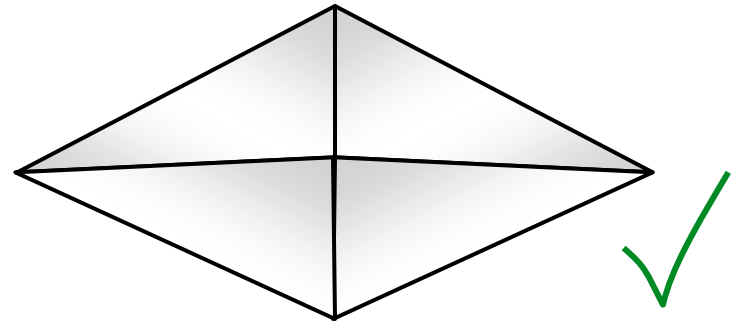
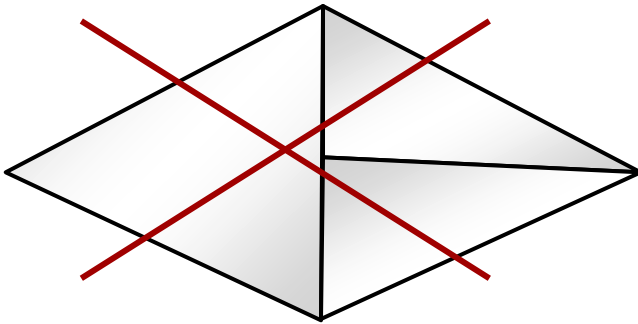
Variants in triangle mesh representations:

- “*Triangle Soup*”
 - A set $S = \{t_1, \dots, t_n\}$ of triangles
 - No further conditions
 - “most common” representation (web downloads and the like)
- *Triangle Meshes*: Additional consistency conditions
 - Conforming meshes: Vertices meet only at vertices
 - Manifold meshes: No intersections, no T-junctions

Conforming Meshes

Conforming Triangulation:

- Vertices of triangles must only meet at vertices, not in the middle of edges:

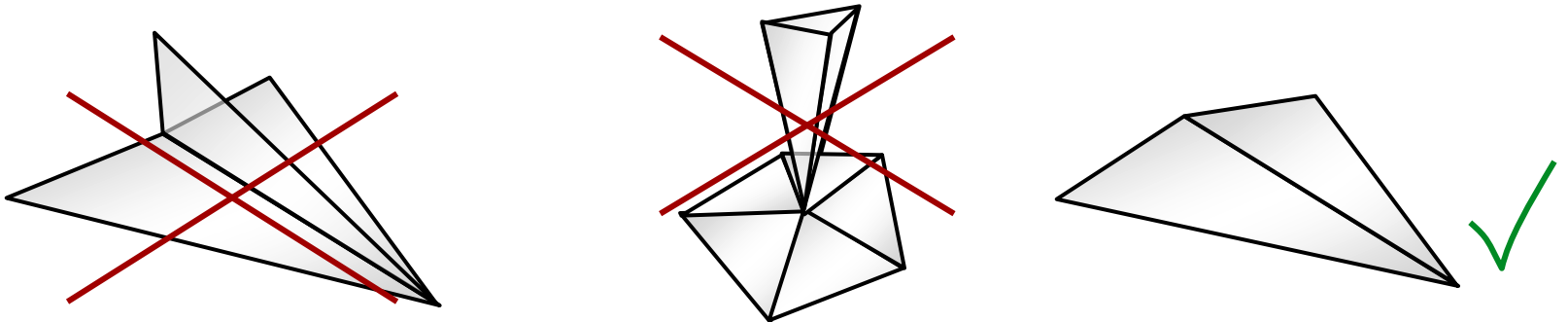


- This makes sure that we can move vertices around arbitrarily without creating holes in the surface

Manifold Meshes

Triangulated two-manifold:

- Every edge is incident to exactly 2 triangles (closed manifold)
- ...or to at most two triangles (manifold with boundary)
- No triangles intersect (other than along common edges or vertices)
- Two triangles that share a vertex must share an edge



Attributes

In general:

- Vertex attributes:
 - Position (mandatory)
 - Normals
 - Color
 - Texture Coordinates
- Face attributes:
 - Color
 - Texture
- Edge attributes (rarely used)
 - E.g.: Visible line

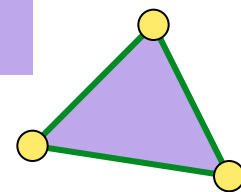
Data Structures

The simple approach: List of vertices, edges, triangles

```
v1: (posx posy posz), attrib1, ..., attribnav  
    ...  
vnv: (posx posy posz), attrib1, ..., attribnav
```

```
e1: (index1 index2), attrib1, ..., attribnae  
    ...  
ene: (index1 index2), attrib1, ..., attribnae
```

```
t1: (idx1 idx2 idx3), attrib1, ..., attribnat  
    ...  
tnt: (idx1 idx2 idx3), attrib1, ..., attribnat
```



Pros & Cons

Advantages:

- Simple to understand and build
- Provides exactly the information necessary for rendering

Disadvantages:

- Dynamic operations are expensive:
 - Removing or inserting a vertex
→ renumber expected edges, triangles
- Adjacency information is one-way
 - Vertices adjacent to triangles, edges → direct access
 - Any other relationship → need to search
 - Can be improved using hash tables (but still not dynamic)

Adjacency Data Structures

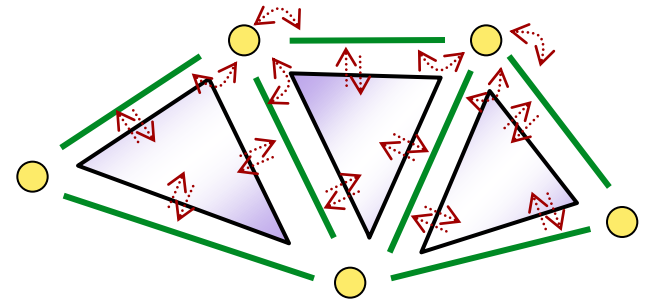
Alternative:

- Some algorithms require extensive neighborhood operations (get adjacent triangles, edges, vertices)
- ...as well as dynamic operations (inserting, deleting triangles, edges, vertices)
- For such algorithms, an *adjacency based* data structure is usually more efficient
 - The data structure encodes the graph of mesh elements
 - Using pointers to neighboring elements

First try...

Straightforward Implementation:

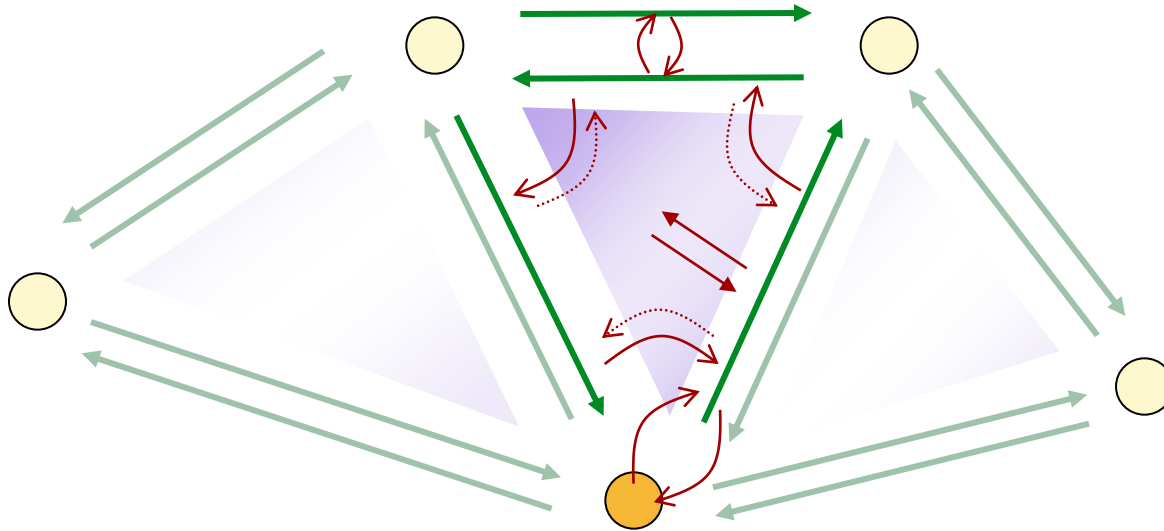
- Use a list of vertices, edges, triangles
- Add a pointer from each element to each of its neighbors
- Global triangle list can be used for rendering



Remaining Problems:

- Lots of redundant information – hard to keep consistent
- Adjacency lists might become very long
 - Need to search again (might become expensive)
 - This is mostly a “theoretical problem” ($O(n)$ search)

Less Redundant Data Structures



Half edge data structure:

- Half edges, connected by clockwise / ccw pointers
- Pointers to opposite half edge
- Pointers to/from start vertex of each edge
- Pointers to/from left face of each edge

Implementation

```
// a half edge  
struct HalfEdge {  
    HalfEdge* next;  
    HalfEdge* previous;  
    HalfEdge* opposite;  
  
    Vertex* origin;  
    Face* leftFace;  
    EdgeData* edge;  
};
```

```
// the data of the edge  
// stored only once  
struct EdgeData {  
    HalfEdge* anEdge;  
    /* attributes */  
};
```

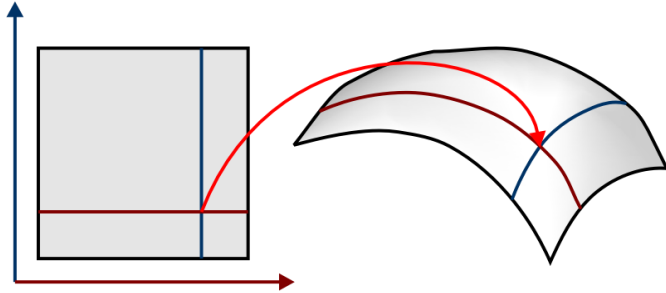
```
// a vertex  
struct Vertex {  
    HalfEdge* someEdge;  
    /* vertex attributes */  
};  
  
// the face (triangle, poly)  
struct Face {  
    HalfEdge* half;  
    /* face attributes */  
};
```


Implementation

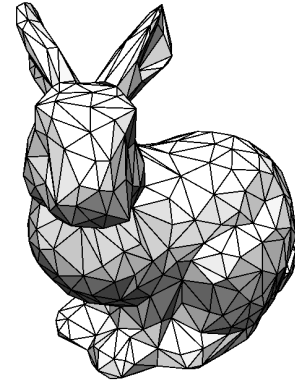
Implementation:

- The data structure should be encapsulated
 - To make sure that updates are consistent
 - Implement abstract data type with more high level operations that guarantee consistency of back and forth pointers
- Free Implementations are available, for example
 - OpenMesh
 - CGAL
- Alternative data structures: for example winged edge (Baumgart 1975)

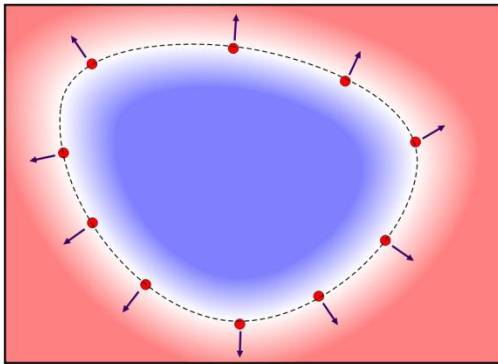
Modeling Zoo



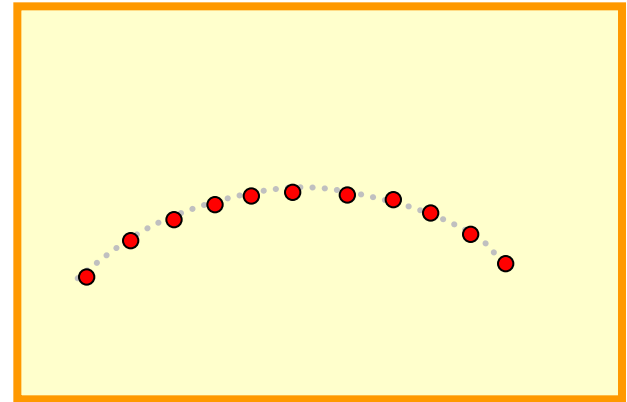
Parametric Models



Primitive Meshes



Implicit Models

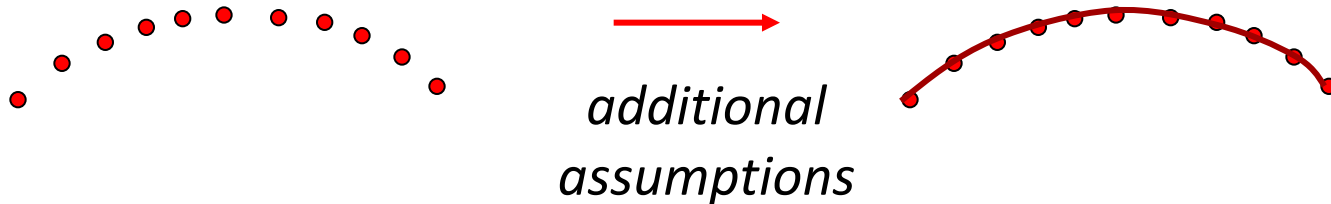


Point-Based Models

Particle Representations

Point-based Representations

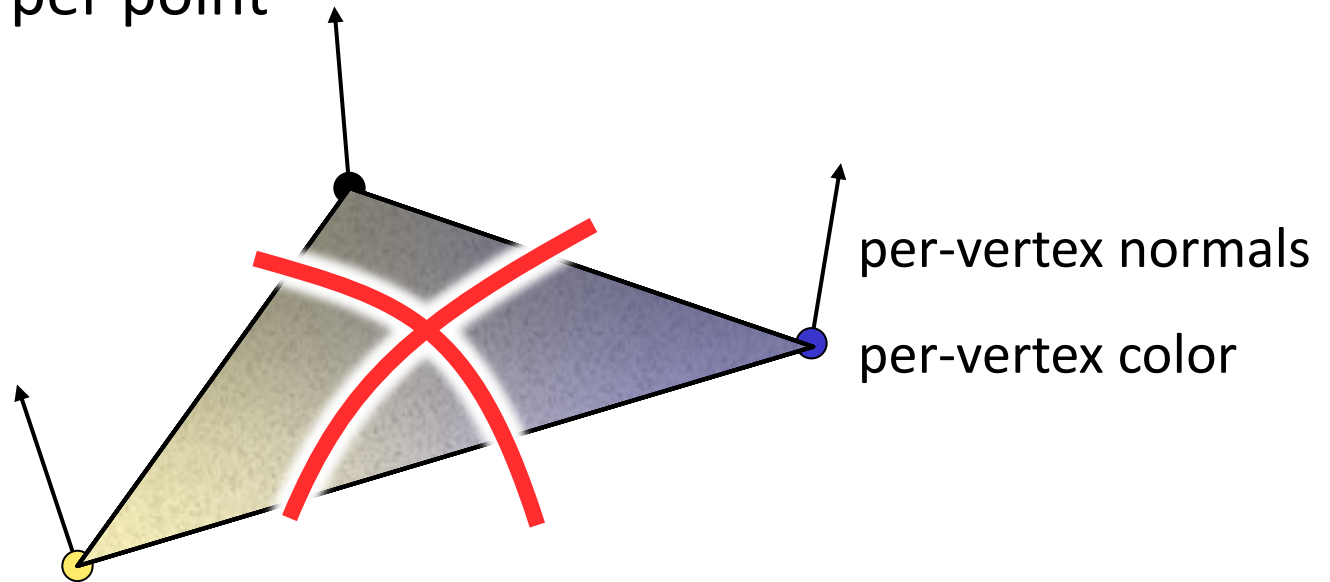
- Set of points
- Points are (irregular) *sample* of the object
- Need additional information to deal with “the empty space around the particles”



Meshless Meshes...

Point Clouds

- Triangle mesh without the triangles
- Only vertices
- Attributes per point



Particle Representations

Helpful Information

- Each particle may carries a set of attributes
 - **Must have:** Its position
 - **Additional geometry:**
Density (sample spacing), surface normals
 - **Additional attributes:**
Color, physical quantities (mass, pressure, temperature), ...
- Addition information helps *reconstructing* the geometric object described by the particles

The Wrath of Khan

Why Star Trek is at fault...

- Particle methods: first used for fuzzy phenomena (fire, clouds, smoke)
- “*Particle Systems—a Technique for Modeling a Class of Fuzzy Objects*” [Reeves 1983]
- Movie: Genesis sequence

Geometric Modeling

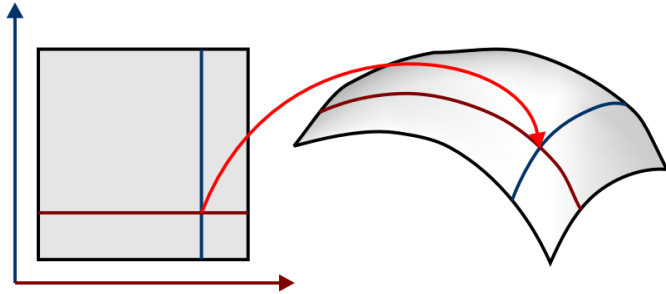
3D Scanners

- 3D scanner yield point clouds
 - Have to deal with points anyway
- Algorithms that directly work on “point clouds”

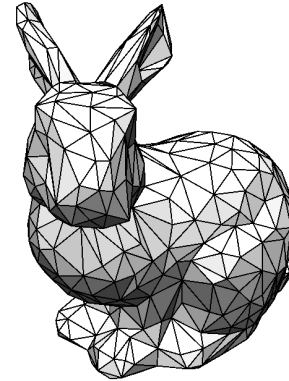


Data: [IKG, University Hannover, C. Brenner]

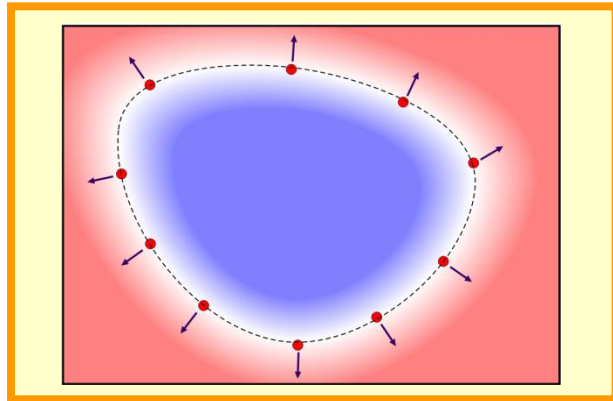
Modeling Zoo



Parametric Models



Primitive Meshes



Implicit Models



Point-Based Models

Implicit Modeling

General Formulation:

- Curve / Surface $S = \{\mathbf{x} \mid f(\mathbf{x}) = 0\}$
- $\mathbf{x} \in \mathbb{R}^d$ ($d = 2, 3$), $f(\mathbf{x}) \in \mathbb{R}$
- S is (usually) a $d-1$ dimensional object

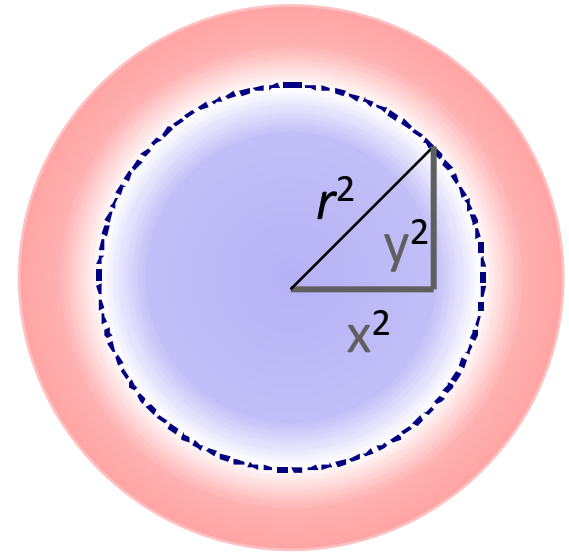
This means....:

- The surface obtained implicitly
- Set of points where f vanishes: $f(\mathbf{x}) = 0$
- Alternative notation: $S = f^{-1}(0)$
("inverse" yields a set)

Implicit Modeling

Example:

- Circle: $\mathbf{x}^2 + \mathbf{y}^2 = r^2$
 $\Leftrightarrow f_r(\mathbf{x}, \mathbf{y}) = \mathbf{x}^2 + \mathbf{y}^2 - r^2 = 0$
- Sphere: $\mathbf{x}^2 + \mathbf{y}^2 + \mathbf{z}^2 = r^2$



Special Case:

- Signed distance field
- Function value is signed distance to surface

$$f(\mathbf{x}, \mathbf{y}) = \text{sign}(\mathbf{x}^2 + \mathbf{y}^2 - \mathbf{r}^2) \sqrt{|\mathbf{x}^2 + \mathbf{y}^2 - \mathbf{r}^2|}$$

- Negative means inside, positive means outside

Implicit Modeling: Pros & Cons

Advantages:

- Topology changes easy (in principle)
- Standard technique for simulations with *free boundaries* (“*level-set methods*”)
 - Example: fluid simulation (evolving water-air interface)
- Other applications:
 - Surface reconstruction
 - “Blobby surfaces”
 - Surface analysis (local)

Implicit Modeling: Pros & Cons

Disadvantages:

- Need to solve inversion problem $S = f^{-1}(0)$
- More complex / slower algorithms
- Usually needs more memory than meshes

Implicit Function – Details

The Implicit Function Theorem

Implicit Function Theorem:

- Given a *differentiable* function

$$f: \mathbb{R}^n \supseteq D \rightarrow \mathbb{R}, \quad f(\mathbf{x}^{(0)}) = 0, \quad \frac{\partial}{\partial x_n} f(\mathbf{x}^{(0)}) = \frac{\partial}{\partial x_n} f(x_1^{(0)}, \dots, x_n^{(0)}) \neq 0$$

- Within an ε -neighborhood of $\mathbf{x}^{(0)}$ we can represent the zero level set of f completely as a heightfield function g

$g: \mathbb{R}^{n-1} \rightarrow \mathbb{R}$ such that for $\mathbf{x} - \mathbf{x}^{(0)} < \varepsilon$ we have:

$$f(x_1, \dots, x_{n-1}, g(x_1, \dots, x_{n-1})) = 0 \text{ and}$$

$$f(x_1, \dots, x_n) \neq 0 \text{ everywhere else.}$$

- The heightfield is a differentiable $(n - 1)$ -manifold and its surface normal is the colinear to the gradient of f .

This means

Surface modeling:

- Use smooth (differentiable) function f in \mathbb{R}^3
- 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

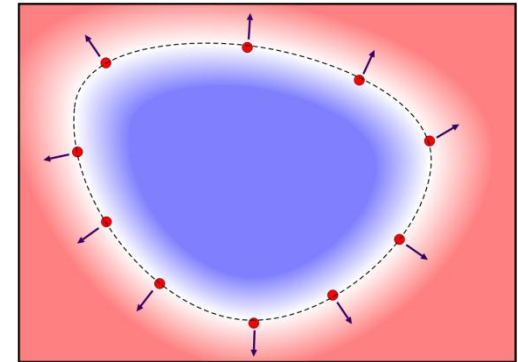
Function types:

- General case
 - Non-zero gradient at zero crossing
 - Otherwise arbitrary
- Signed implicit function:
 - $\text{sign}(f)$: 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
 - $\text{sign}(f)$: negative inside, positive outside
- Squared distance function
 - $f = (\text{distance to the surface})^2$

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!*).

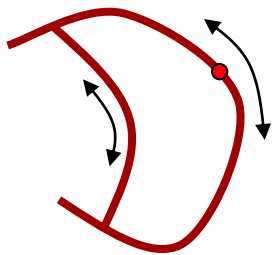


signed distance

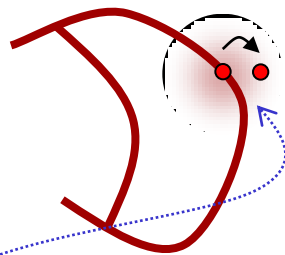
Squared Distance Function

Example: Surface from random samples

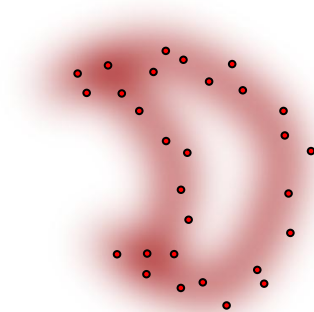
1. Determine sample point (uniform)
2. Add noise (Gaussian)



sampling



Gaussian noise



many samples



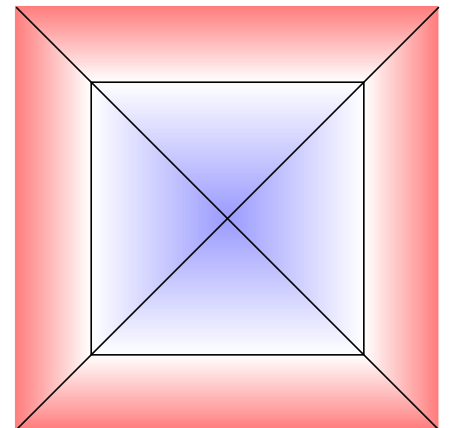
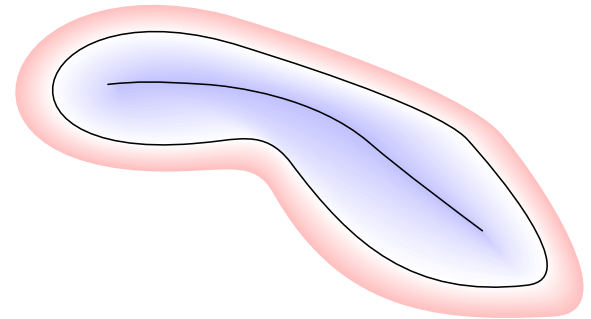
distribution
(in space)

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

Smoothness

Smoothness of signed distance function:

- Any distance function (signed, unsigned, squared) cannot be globally smooth in general cases
- 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



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):

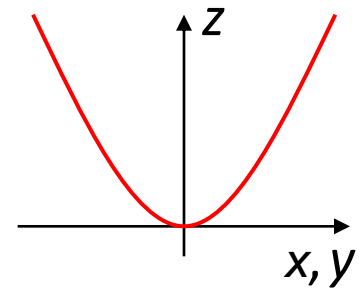
- We assume that the normal is in z -direction, i.e., x, y are tangent to the surface (divergence is invariant under rotation). The surface normal is given by:

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

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

$$= \text{trace} \begin{pmatrix} \frac{\partial^2}{\partial x^2} s(x, y) & \frac{\partial^2}{\partial x \partial y} s(x, y) \\ \frac{\partial^2}{\partial x \partial y} s(x, y) & \frac{\partial^2}{\partial y^2} s(x, y) \end{pmatrix} = -2H(x, y)$$

$$\left[H(\mathbf{x}_0) = \frac{1}{2} \text{tr}(S(\mathbf{x}_0)) \right]$$

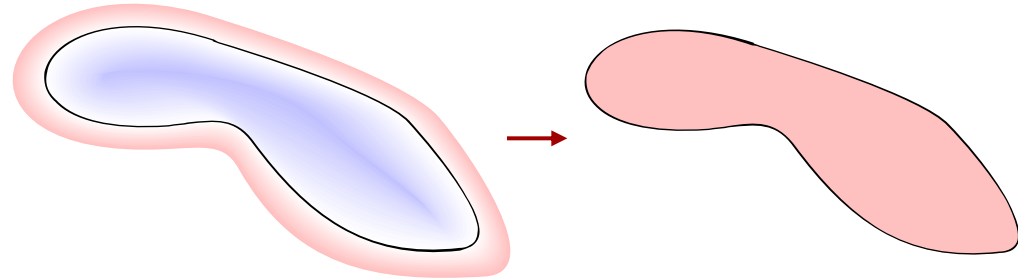


Computing Volume Integrals

Computing volume integrals:

- Heavyside function:

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



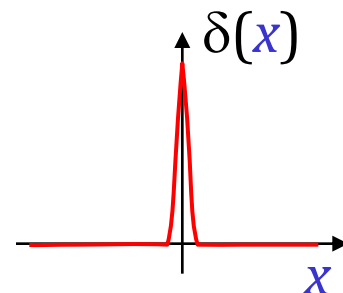
- Volume integral over interior volume Ω_f of some function $g(\mathbf{x})$ (assuming negative interior values):

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

Computing Surface Integrals

Computing surface integrals:

- Dirac delta function:
 - Idealized function (distribution)
 - Zero everywhere ($\delta(\mathbf{x}) = 0$), except at $\mathbf{x} = 0$, where it is positive, infinitely large.
 - The integral of $\delta(\mathbf{x})$ over \mathbf{x} is one.
- Dirac delta function on the surface: directional derivative of $\text{step}(\mathbf{x})$ in normal direction:



$$\begin{aligned}\hat{\delta} &= \nabla[\text{step}(f(\mathbf{x}))] \cdot \mathbf{n}(\mathbf{x}) = [\nabla \text{step}](f(\mathbf{x})) \cdot \nabla f(\mathbf{x}) \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|} \\ &= \delta(f(\mathbf{x})) \cdot |\nabla f(\mathbf{x})|\end{aligned}$$

Surface Integral

Computing surface integrals:

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

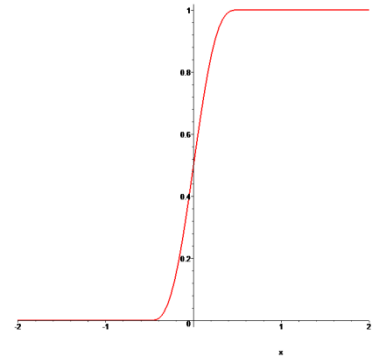
$$\int_{\partial\Omega_f} g(\mathbf{x}) d\mathbf{x} = \int_{\mathbb{R}^3} g(\mathbf{x}) \delta(f(\mathbf{x})) |\nabla f(\mathbf{x})| d\mathbf{x}$$

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

Smoothed Functions

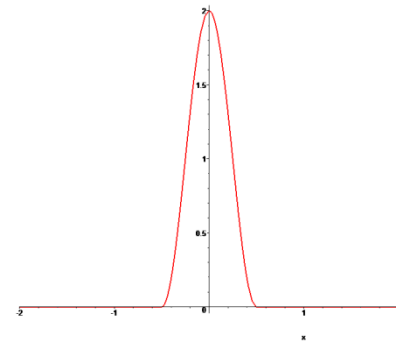
Smooth-step function

$$\text{smooth_step}(x) = \begin{cases} 0 & x < -\varepsilon \\ \frac{1}{2} + \frac{x}{2\varepsilon} + \frac{1}{2\pi} \sin\left(\frac{\pi x}{\varepsilon}\right) & -\varepsilon \leq x \leq \varepsilon \\ 1 & \varepsilon < x \end{cases}$$



Smoothed Dirac delta function

$$\text{smooth_delta}(x) = \begin{cases} 0 & x < -\varepsilon \\ \frac{1}{2\varepsilon} + \frac{1}{2\varepsilon} \cos\left(\frac{\pi x}{\varepsilon}\right) & -\varepsilon \leq x \leq \varepsilon \\ 0 & x > \varepsilon \end{cases}$$



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

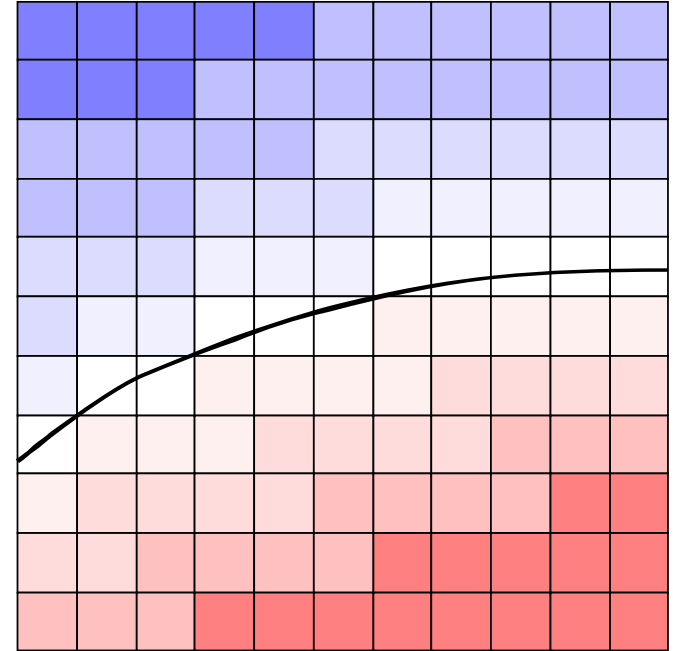
Regular Grids

Discretization:

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

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

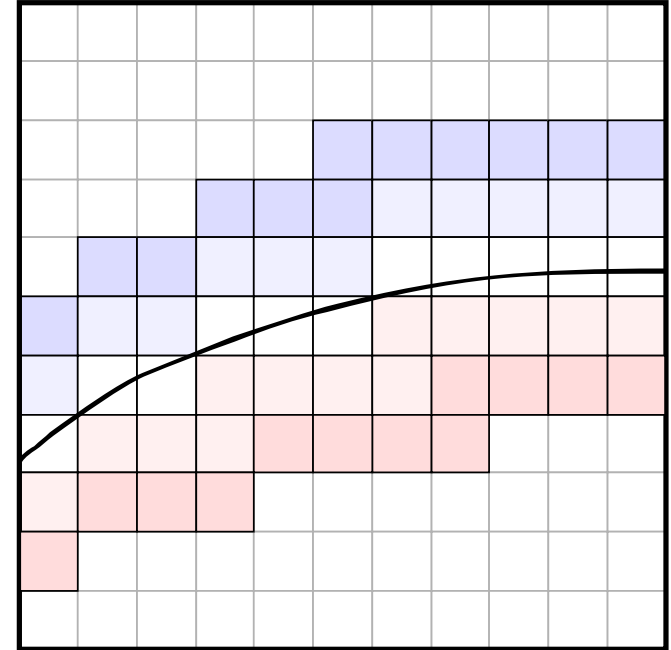
$$\frac{\partial}{\partial_x} f(\mathbf{x}) = \frac{1}{2h} (f_{i(\mathbf{x})+1,j(\mathbf{x})} - f_{i(\mathbf{x})-1,j(\mathbf{x})}) + O(h^2)$$



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.

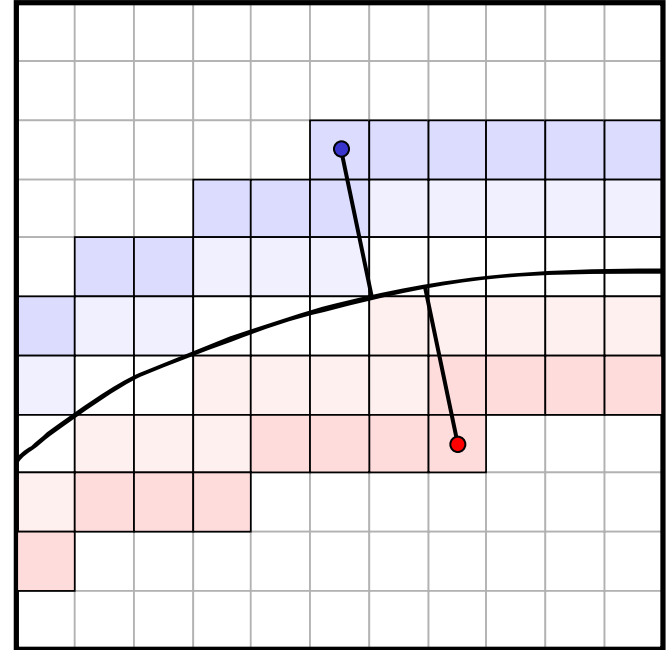
Three solutions:

- 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

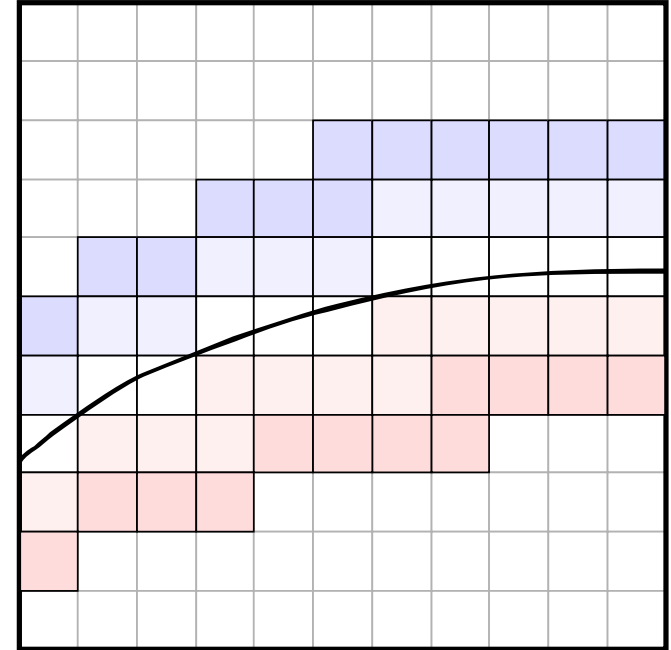
- Place variables in empty cells
- Fixed values in known cells
- Then solve the following PDE:

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

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

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

- 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.

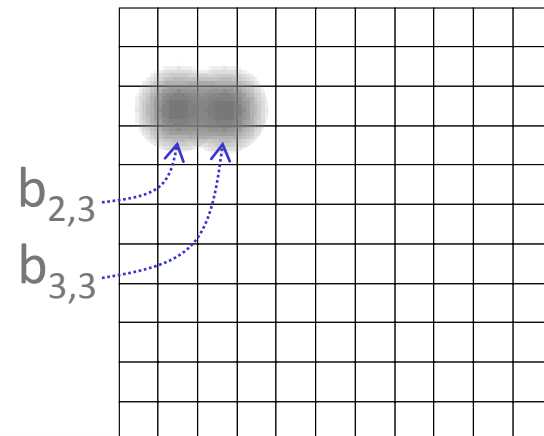
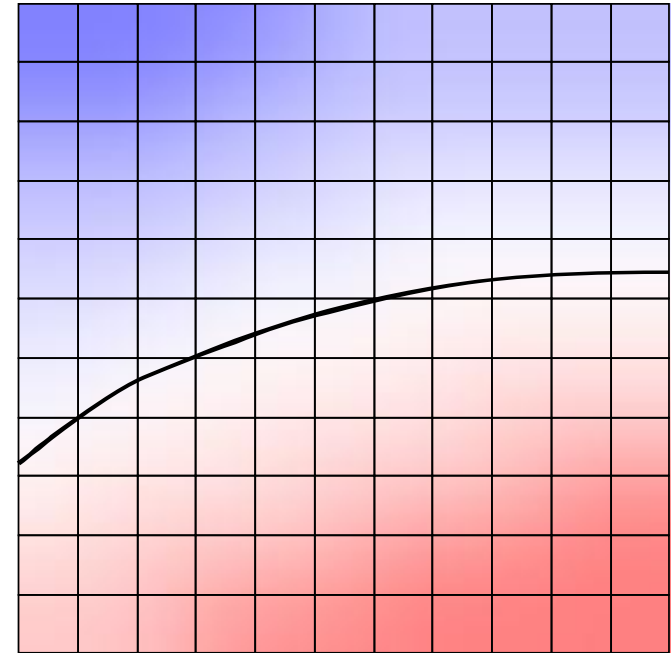
Regular Grids of Basis Functions

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 .



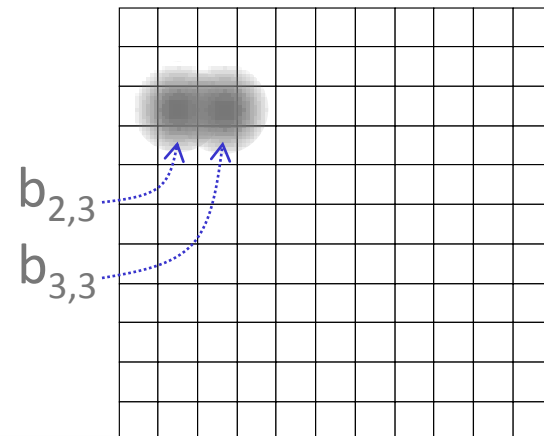
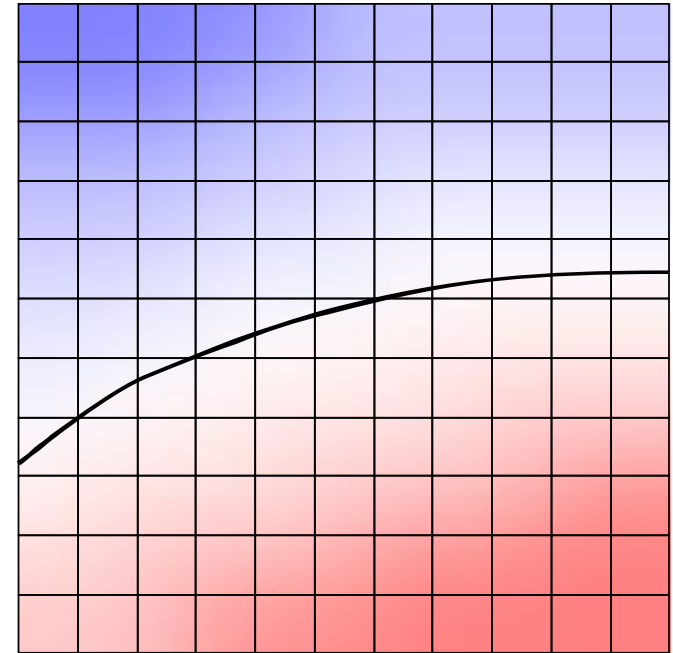
Regular Grids of Basis Functions

Differential Properties:

- Derivatives:

$$\frac{\partial}{\partial x_{k_1} \dots \partial x_{k_m}} f(x, y)$$
$$= \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \lambda_{i,j} \left(\frac{\partial}{\partial x_{k_1} \dots \partial x_{k_m}} b \right) (x, y)$$

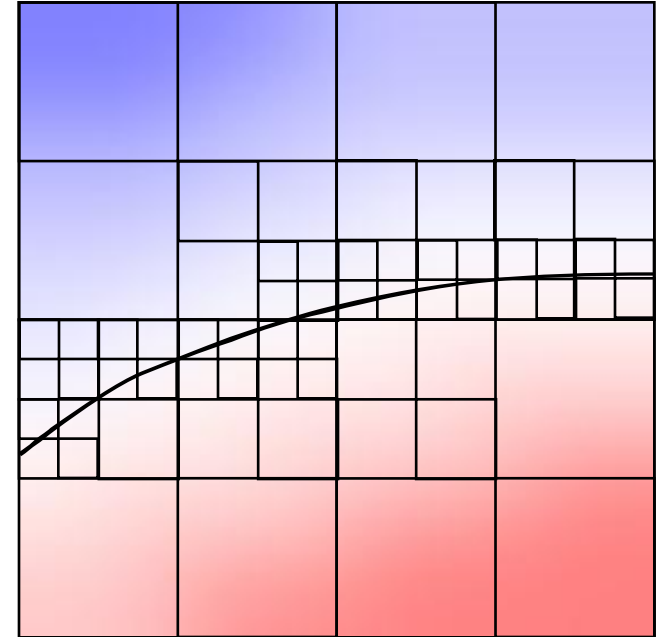
- Derivatives are linear combinations of the derivatives of the basis function.
- In particular: We again get a linear expression in the $\lambda_{i,j}$.



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)



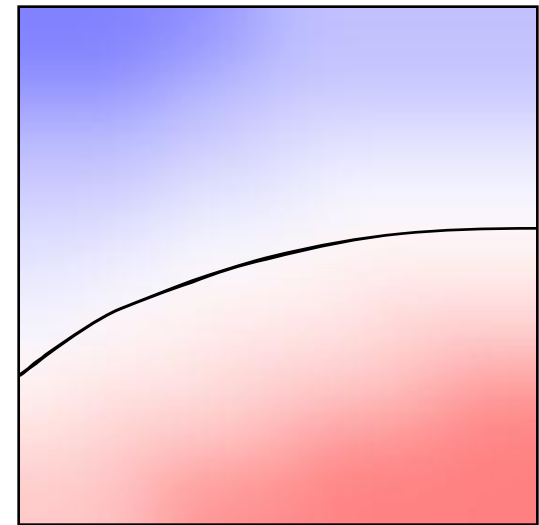
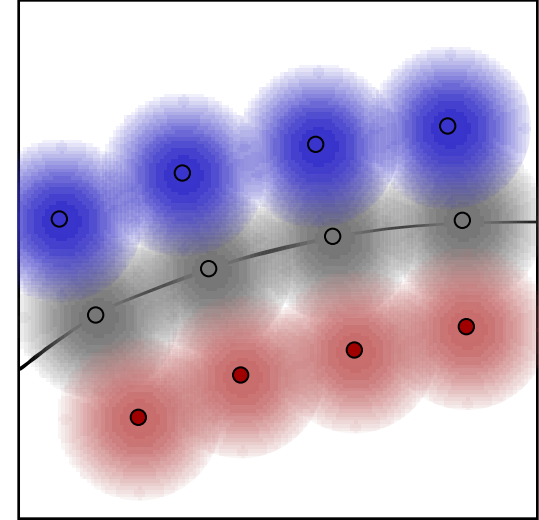
Particle Methods

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 .



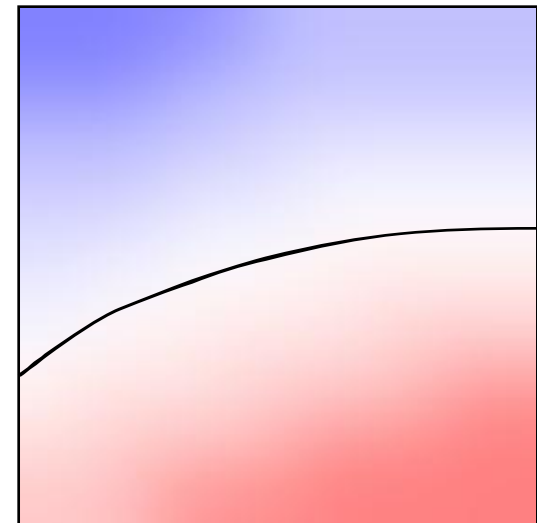
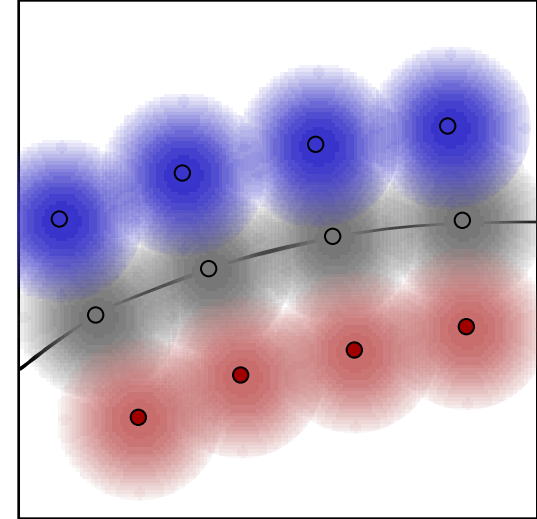
Particle Methods

Particle methods / radial basis functions:

- Obviously, derivatives are again linear in λ_i :

$$\frac{\partial}{\partial x_{k_1} \dots \partial x_{k_m}} f(\mathbf{x}) = \sum_{i=0}^n \lambda_i \frac{\partial}{\partial x_{k_1} \dots \partial x_{k_m}} b(\mathbf{x} - \mathbf{x}_i)$$

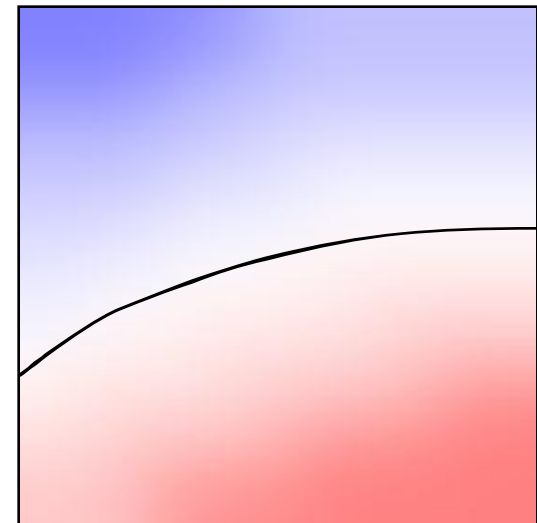
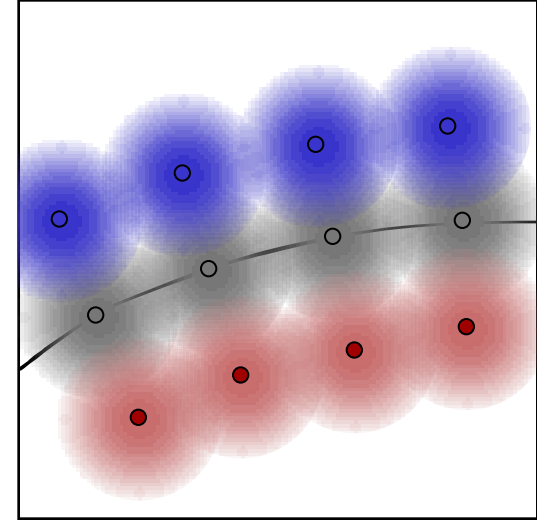
- The radial basis functions can also have different size (support) for adaptive refinement
- Placement: near the expected surface



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.



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.
- There are alternatives (in particular for special cases)

Marching Cubes

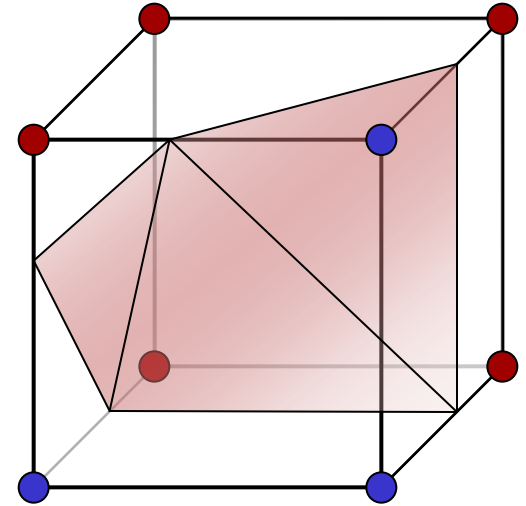
Marching Cubes:

- The most frequently used iso-surface extraction algorithm
 - Triangle mesh from an iso-value surface of a scalar volume
 - Example: Visualization of CT scanner 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:

- Local problem:
 - 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 to triangulate this cube?
 - How to place the vertices?



Triangulation

Triangulation:

- 256 different cases
 - Each of 8 vertices: in or out.
- By symmetry: reduction to 15 cases
 - Reflection, rotation, bit inversion
- Computes the topology of the mesh

Vertex Placement

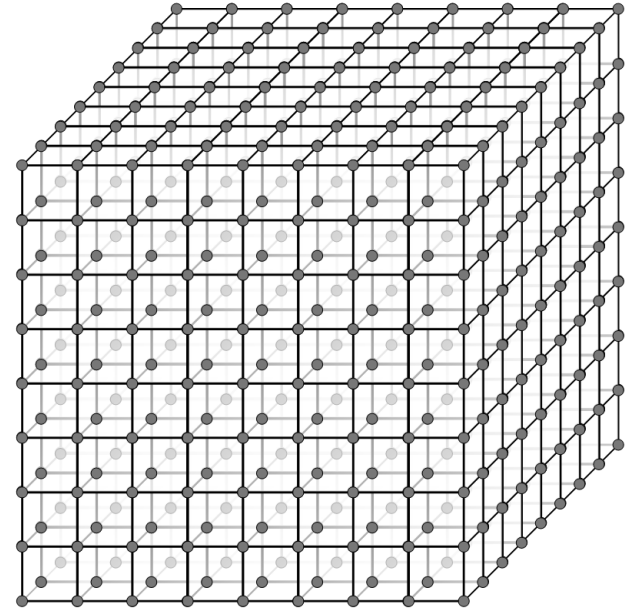
How to place the vertices?

- Zero-th order: Vertices at edge midpoints
- First order: Linearly interpolate vertices along edges.
- Example:
 - $f(x) = -0.1$ and $f(y) = 0.2$
 - Vertex at ratio 1:2 between x and y

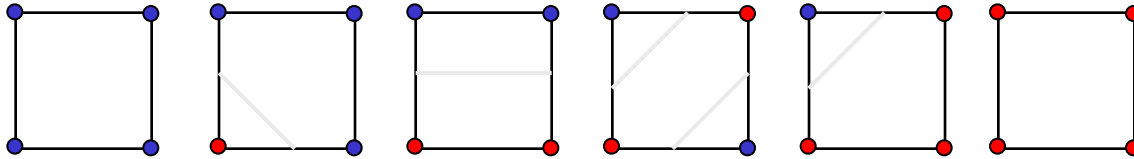
Outer Loop

Outer Loop:

- Start: bounding box
- Divide into cubes (regular grid)
- Execute “marching cube”
in each subcube
- Output: union of all cube results
- Optional:
 - Vertex hash table to make
mesh consistent
 - Removes double vertices



Marching Squares



Marching Squares:

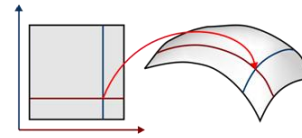
- There is also a 2D version of the algorithm, called marching squares.
- Same idea, but fewer cases.

Representations Summary

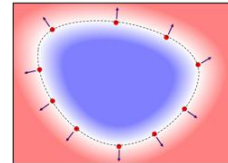
Summary

Summary

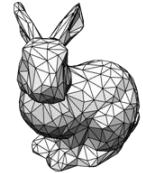
- Many different representations
- No silver bullet
- All representations work in principle for all problems
- Effort application dependent
 - Conceptual effort
 - Computational effort



Parametric Models



Implicit Models



Primitive Meshes



Particle Models