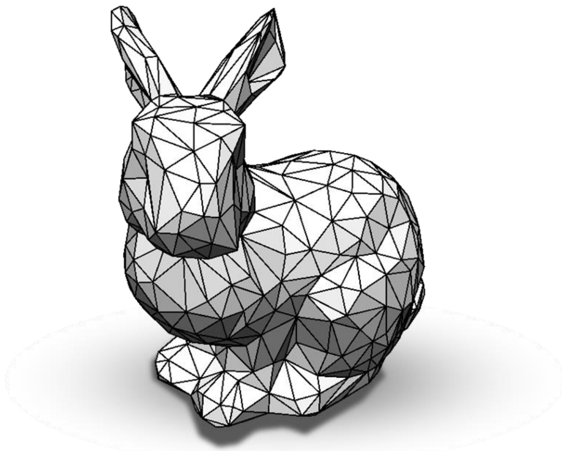
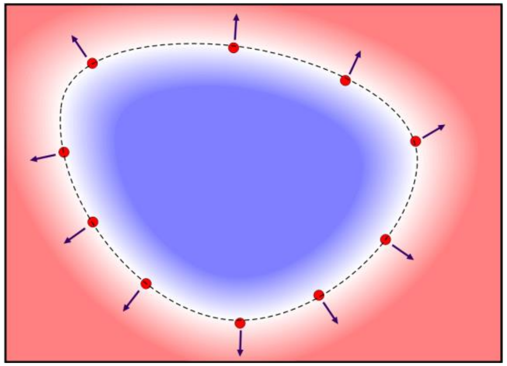
Modelling 1

SUMMER TERM 2020





Lecture 7

Representing Geometry

Michael Wand · Institut für Informatik · michael.wand@uni-mainz.de

Motivation

Modeling

Available tools

- Vectors (in linear spaces)
- Functions (high-res vectors)

Goal

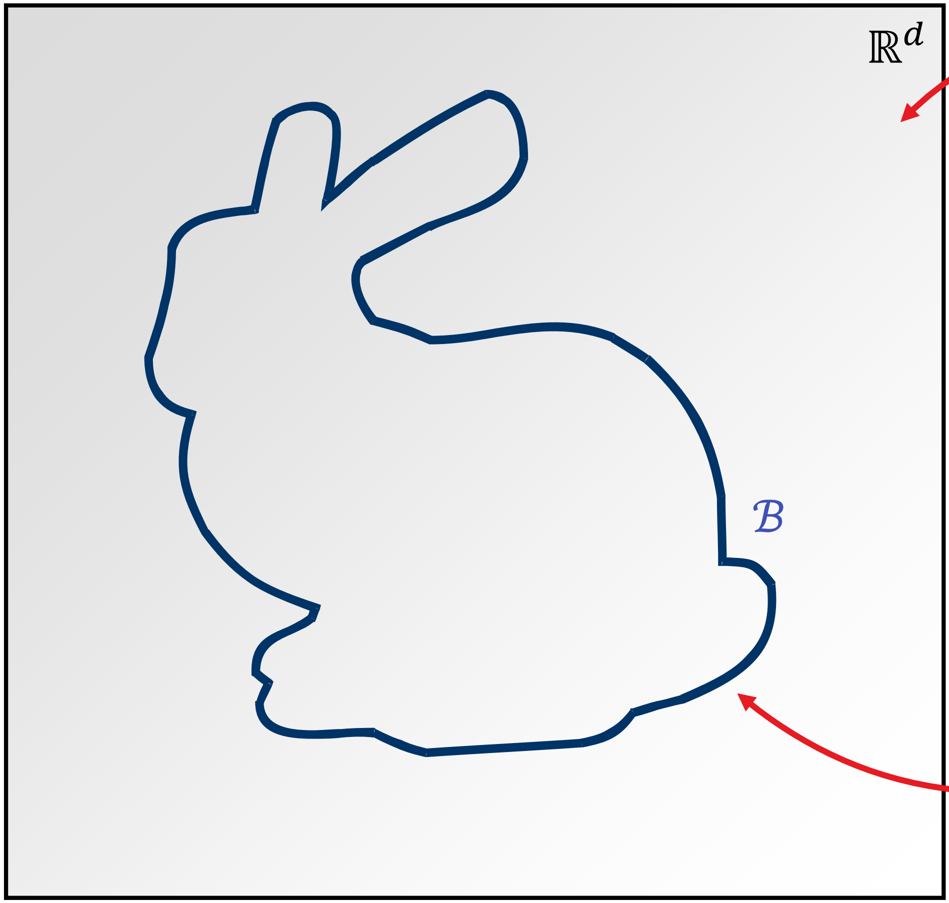
- Model more complex phenomena
- Map to linear representation

Example

- Geometric objects
- Many phenomena have a geometric interpretation

Geometric Modeling

What do we want to do?

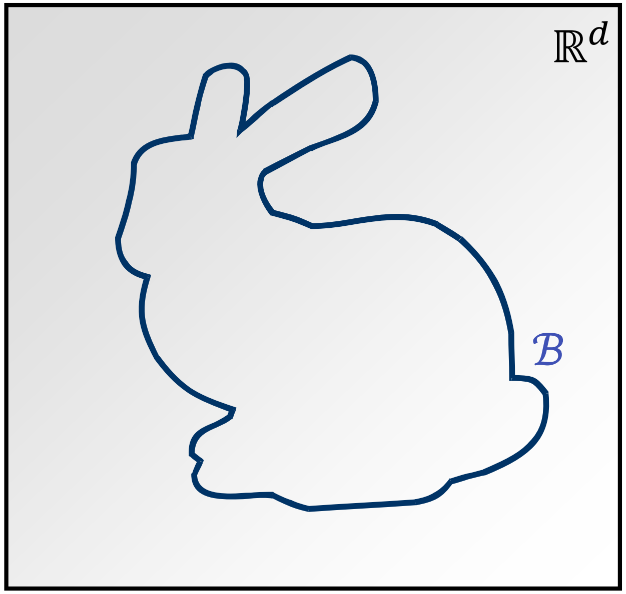


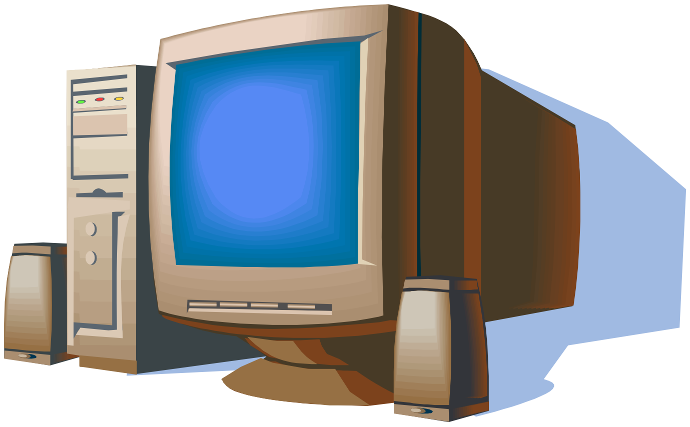
empty space
(typically $\mathbb{R}^3$)

geometric object
$\mathcal{B} \subset \mathbb{R}^d$

Fundamental Problem

Problem





infinite number of points

my computer: 32GB of memory

Encode continuous model with finite information

Modeling Approaches

Two Basic Approaches

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

Fixed Bins (Voxels, Pixels)

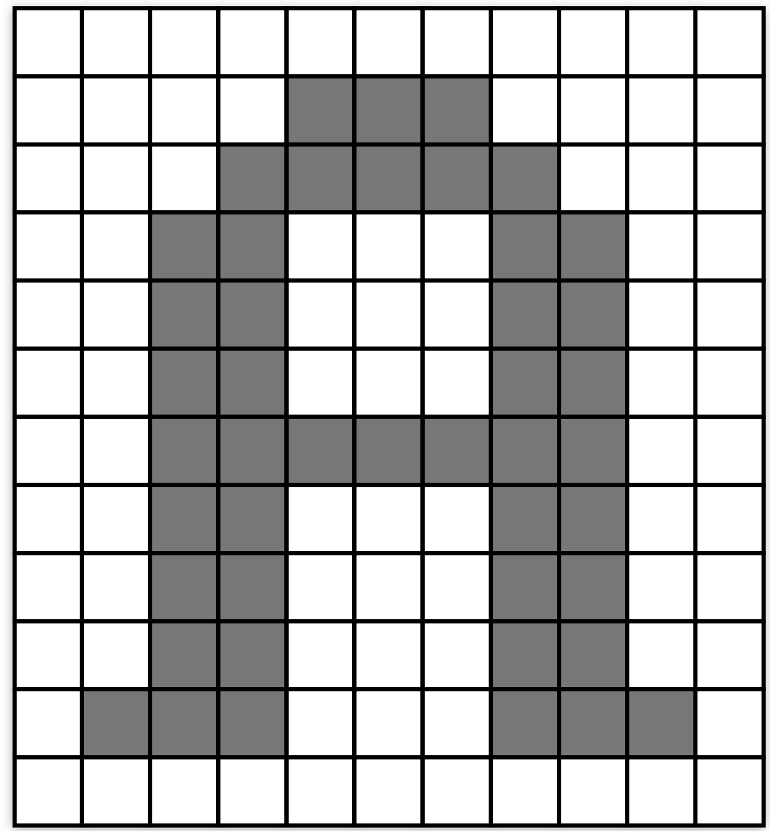
Discrete Representations

You know this...

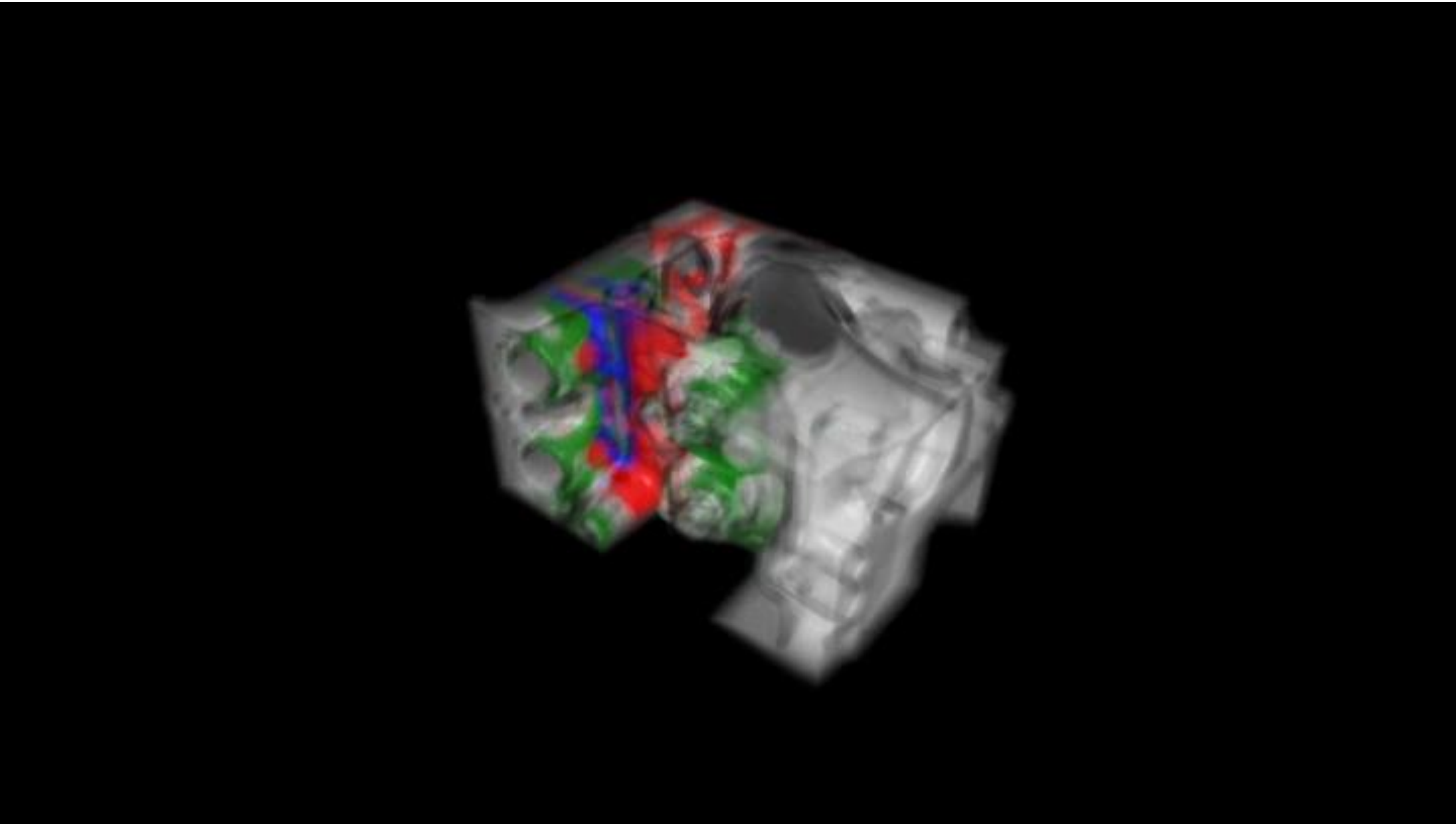
- Fixed Grid of values:

$$[\{1,\ldots,n_1\} \times \cdots \times \{1,\ldots,n_{d_s}\}] \to \mathbb{R}^{d_t}$$

- Typical
  - $d_s = 2, d_t = 3$: Bitmap images
  - $d_s = 3, d_t = 1$: Volume data (scalar fields)
  - $d_s = 2, d_t = 1$: Depth maps (range images)

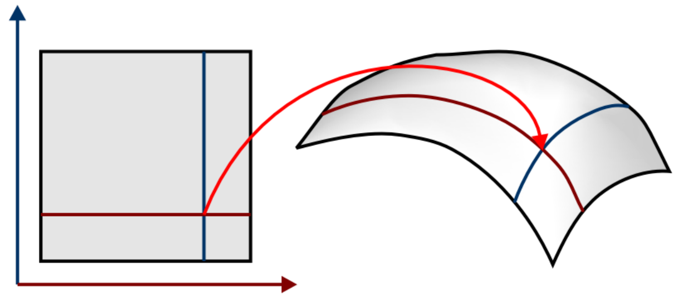


Volume Data (RGBα)

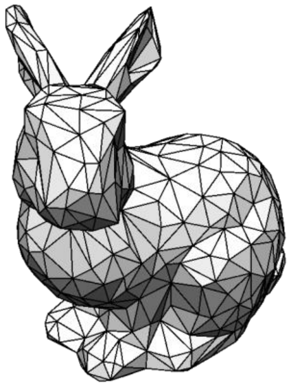


Continuous Modeling Zoo

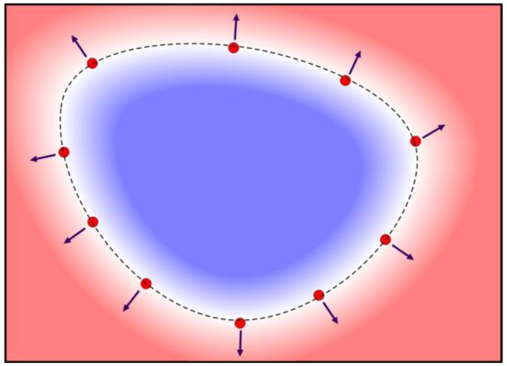
Modeling Zoo



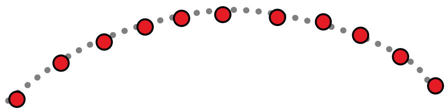
Parametric Models



Primitive Meshes

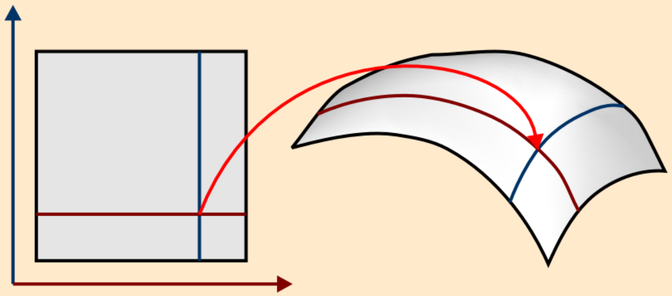


Implicit Models

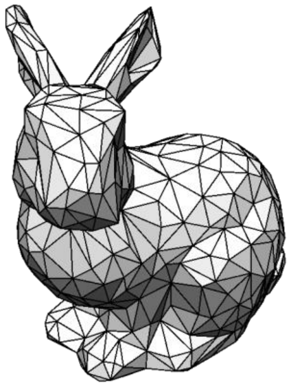


Point-Based Models

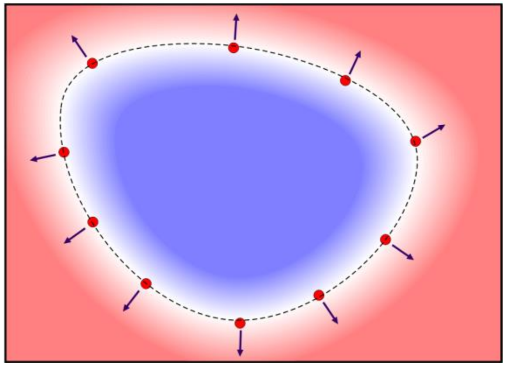
Modeling Zoo



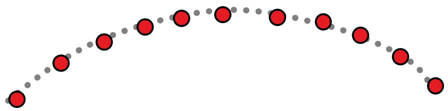
Parametric Models



Primitive Meshes

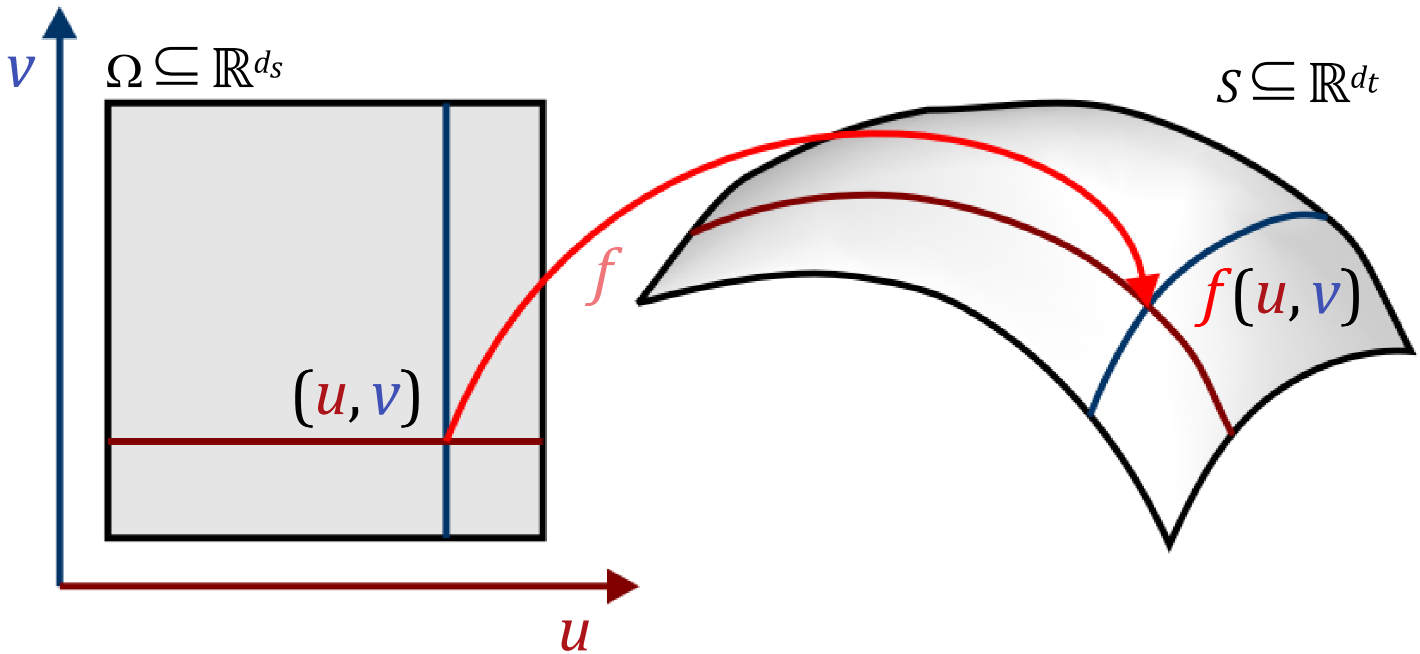


Implicit Models



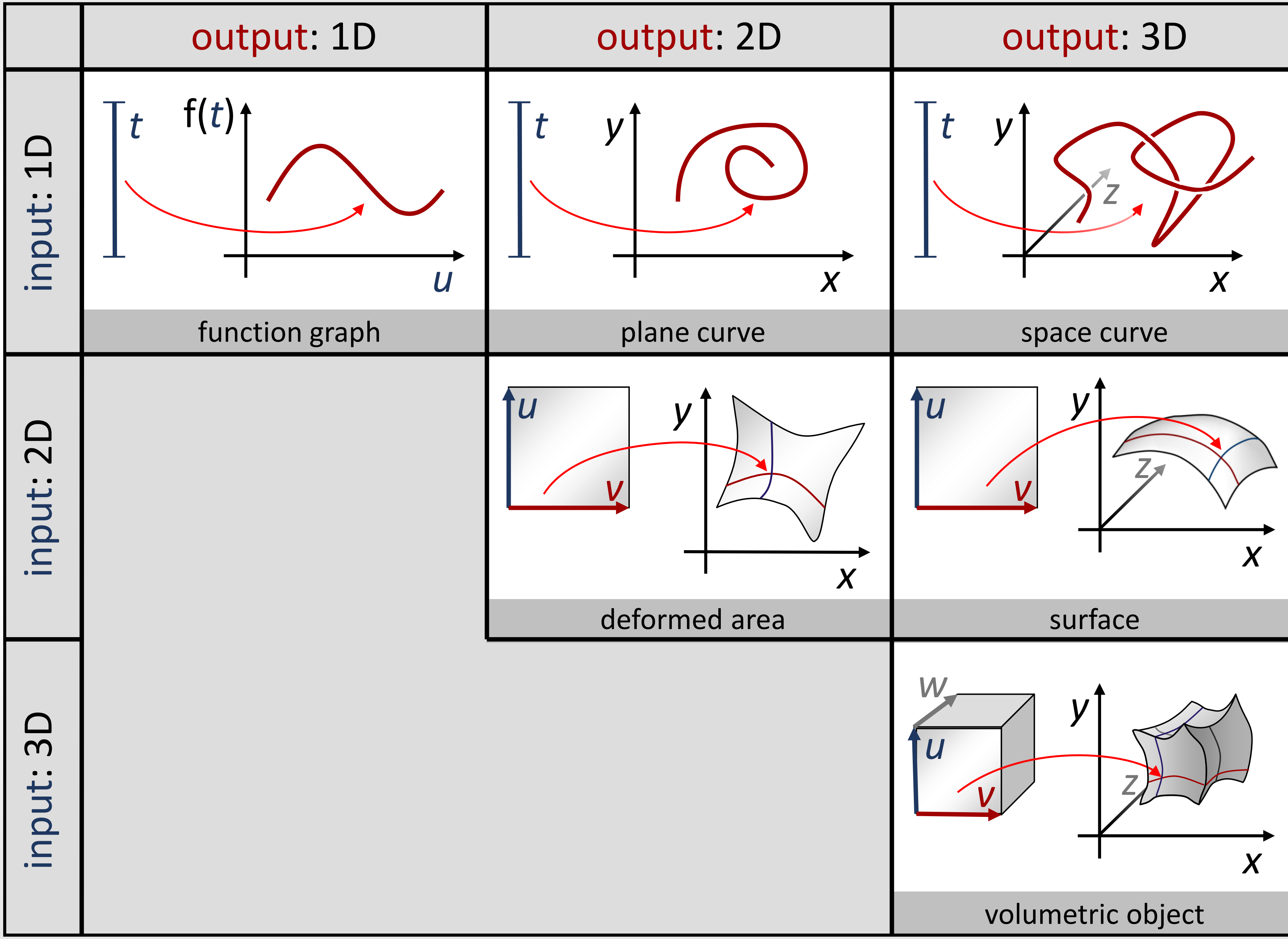
Point-Based Models

Parametric Models



Parametric Models

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

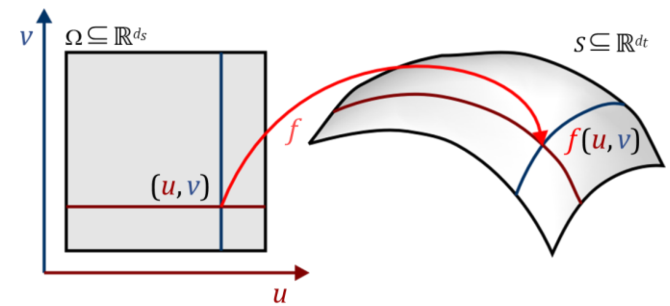


output: 1D
output: 2D
output: 3D
input: 1D
input: 2D
input: 3D
t
f(t)
u
function graph
t
y
x
plane curve
t
y
z
x
space curve
u
v
y
x
deformed area
u
v
y
z
x
surface
w
u
v
y
z
x
volumetric object

Linear Modeling?

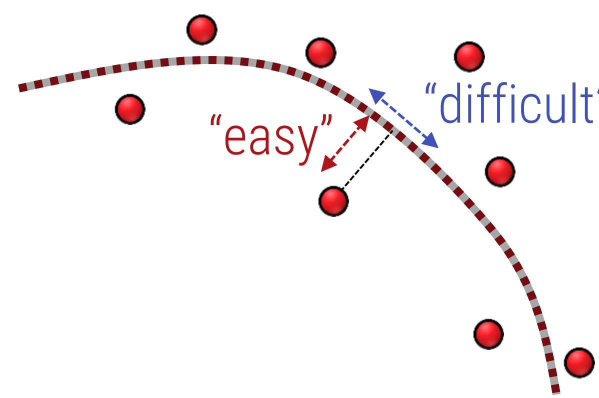
Linear representation

- f is a linear object
- Linear ansatz:



$$f(\mathbf{x}) = \sum_{i=1}^{d} \lambda_i b_i(\mathbf{x})$$

Non-linear (read: difficult)



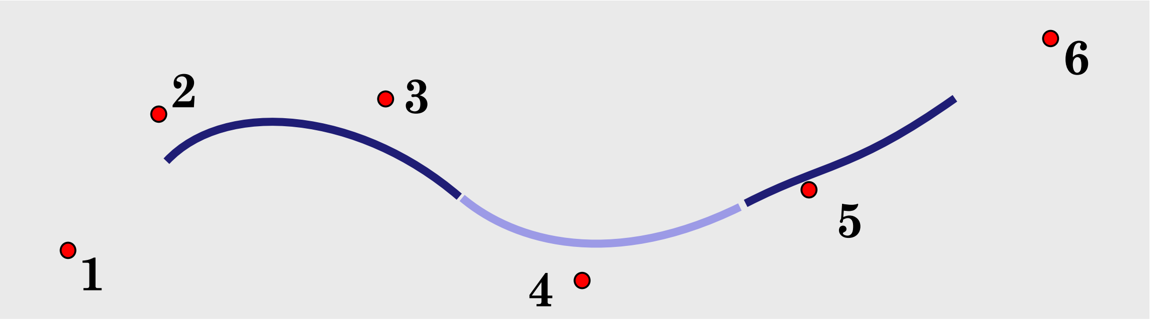
→ total least squares

- Reshaping Ω (domain)
  - Changing the topology
- Reparametrization (warping within Ω)
  - Example: associating points $\mathbf{x} \in \Omega$ with data
  - "Correspondence problem"

Example

Building Smooth Functions with B-Splines

Goal: Smooth Curves/Surfaces



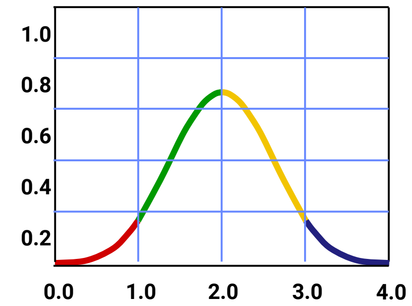
Splines in Computer Graphics (& Numerics)

- Curve roughly follows control points
- Curve should be smooth (C^2) everywhere
- Curve should bend minimally
  - In a certain sense, more later

Build smooth function, linear combine from it!

Cubic Uniform B-Splines

Cubic uniform B-Splines



- Piecewise cubic functions
  - C^2 continuous
  - Popular choice

Ansatz

- Design one basis function $b(t)$
  - $b(t)$ is C^2 continuous.
  - $b(t)$ is piecewise polynomial, degree 3 (cubic).
  - $b(t)$ has local support.
  - Overlaying shifted $b(t + i)$ forms a partition of unity.
  - $b(t) \geq \mathbf{0}$ for all t

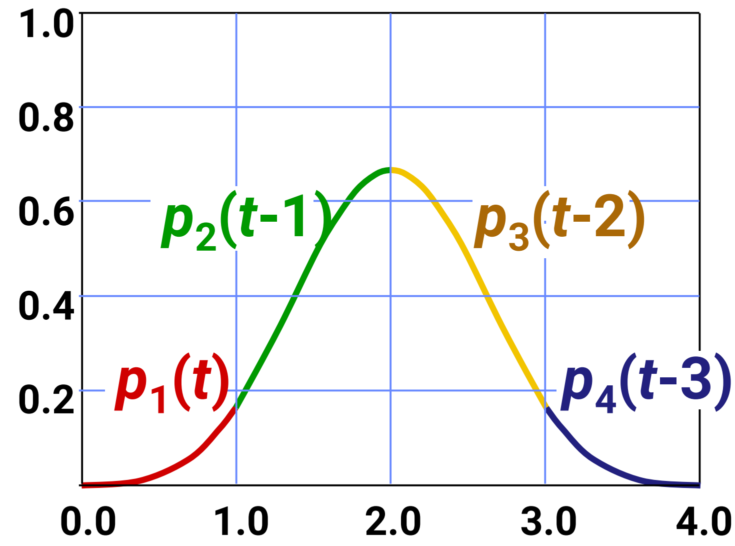
Basis Function

$$p_1(t) = \frac{1}{6}t^3$$

$$p_2(t) = \frac{1}{6}\left(1 + 3t + 3t^2 - 3t^3\right)$$

$$p_3(t) = \frac{1}{6}\left(4 - 6t^2 + 3t^3\right)$$

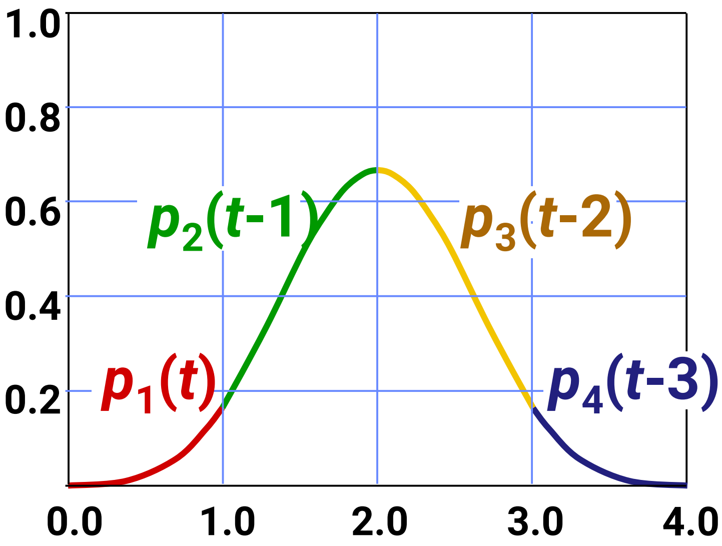
$$p_4(t) = \frac{1}{6}\left(1 - 3t + 3t^2 - t^3\right)$$



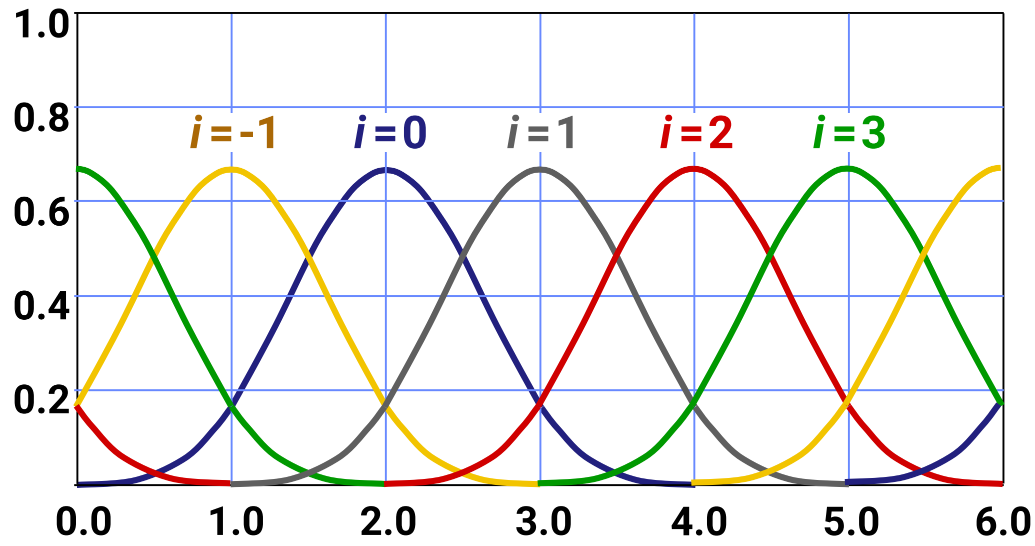
basis function $b(t)$

$$b(t) = \begin{cases} 0 & \text{if } t < 0 \\ p_1(t) & \text{if } 0 < t \le 1 \\ p_2(t-1) & \text{if } 1 < t \le 2 \\ p_3(t-2) & \text{if } 2 < t \le 3 \\ p_4(t-3) & \text{if } 3 < t \le 4 \\ 0 & \text{if } t > 4 \end{cases} = \begin{cases} 0 & \text{if } t < 0 \\ \frac{1}{6}t^3 & \text{if } 0 < t \le 1 \\ \frac{1}{6}\left(1 + 3(t-1) + 3(t-1)^2 - 3(t-1)^3\right) & \text{if } 1 < t \le 2 \\ \frac{1}{6}\left(4 - 6(t-2)^2 + 3(t-2)^3\right) & \text{if } 2 < t \le 3 \\ \frac{1}{6}\left(1 - 3(t-3) + 3(t-3)^2 - (t-3)^3\right) & \text{if } 3 < t \le 4 \\ 0 & \text{if } t > 4 \end{cases}$$

Shifted Basis Functions



basis function $b(t)$

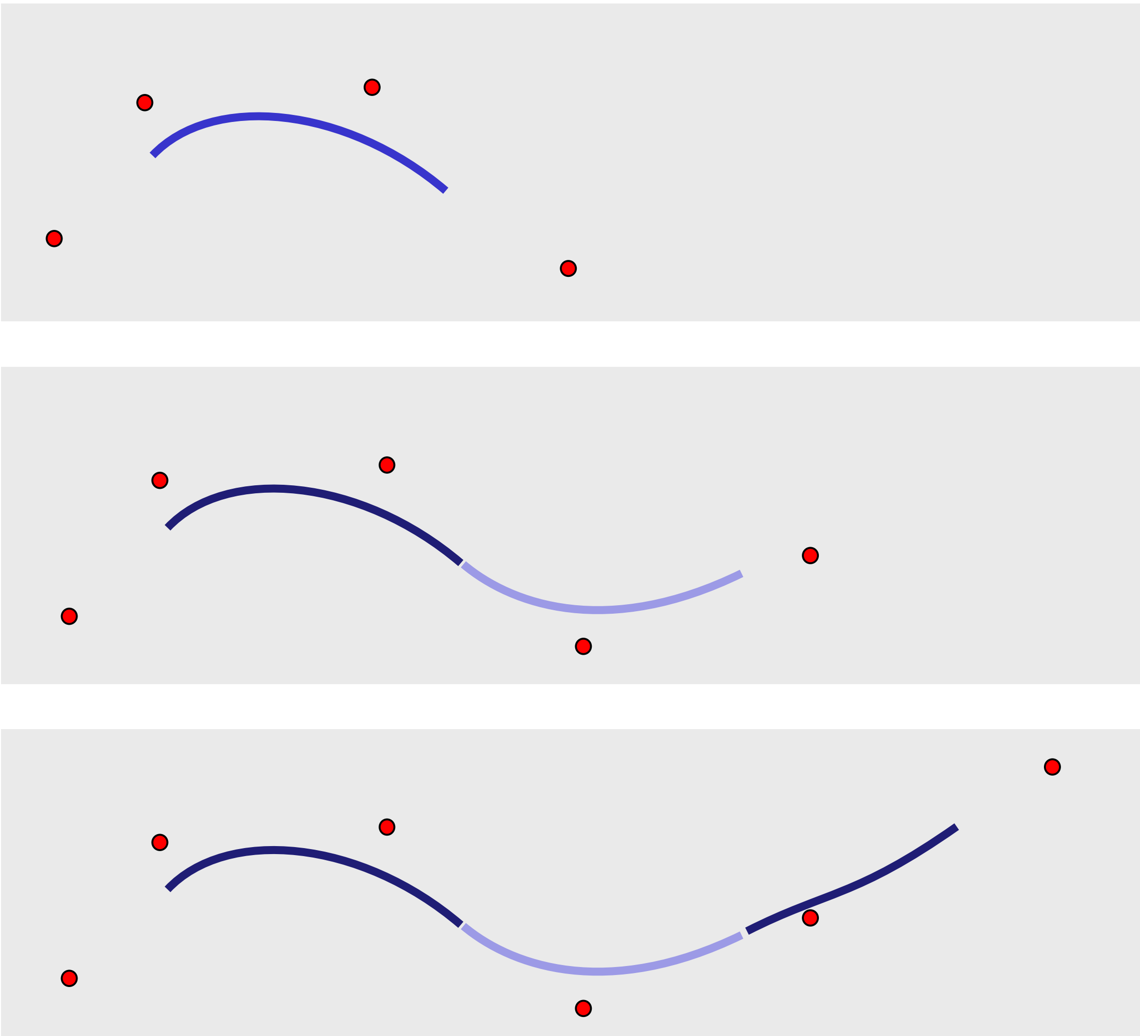


shifted basis functions $b(t-i)$ for [0..6]

Basis function:

- Consists of four polynomial parts $p_1 \ldots p_4$.
- Shifted basis $b(t-i)$: spacing of 1.
- Each interval to be used must be overlapped by 4 different b_i.

Example: Uniform B-Spline Curves



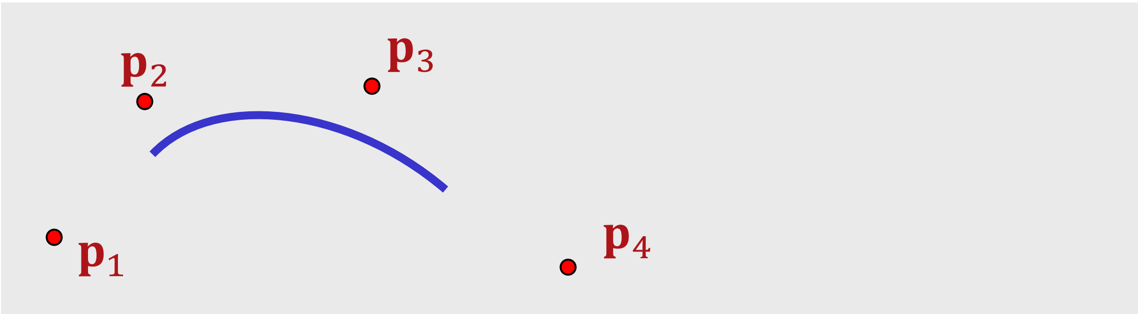
one segment

two segments

three segments

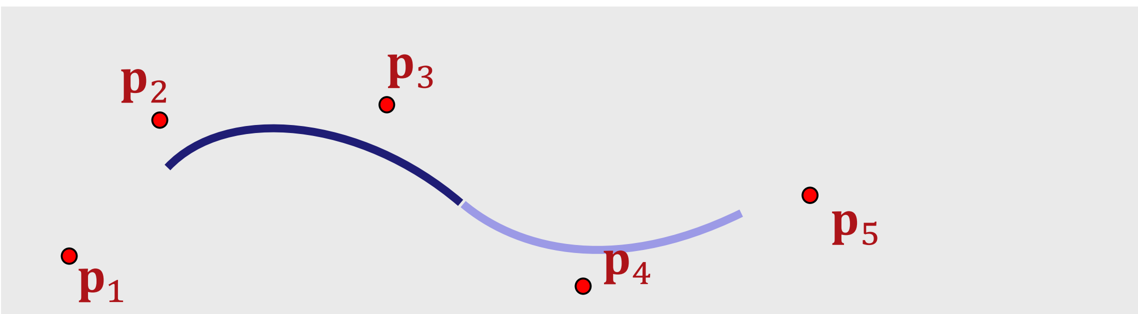
x and y coordinate: linear combination of B-Spline basis
n+3 coefficients (4 for each segment)

Example: Uniform B-Spline Curves



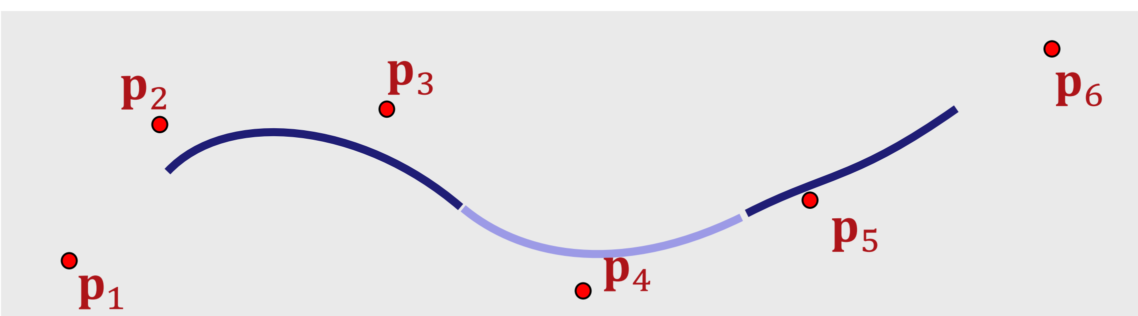
one segment

$$f = \sum_{i=1}^{4} \mathbf{p}_i b_i$$



two segments

$$f = \sum_{i=1}^{5} \mathbf{p}_i b_i$$



three segments

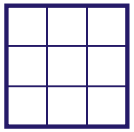
$$f = \sum_{i=1}^{6} \mathbf{p}_i b_i$$

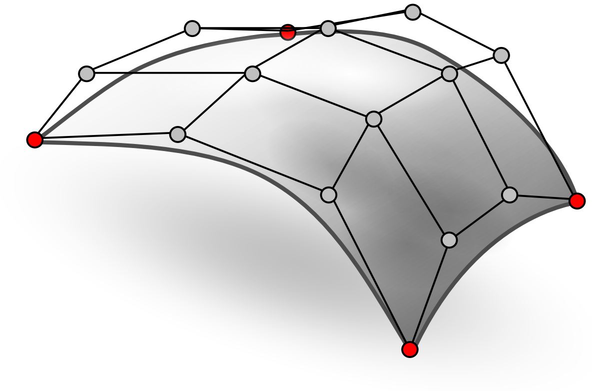
x and y coordinate: linear combination of B-Spline basis
n+3 coefficients (4 for each segment)

Spline Surfaces

Tensor product surfaces

- Simple construction
- Grid of control points
- Rectangular patches





Tensor Product Surfaces

Simple Idea:

- Basis for a one dimensional function space

$$B^{(curv)} = \{b_1(t), b_2(t), \dots, b_n(t)\},$$
$$b_i: (a, b) \to \mathbb{R}$$

- Two parameter basis from all possible products:

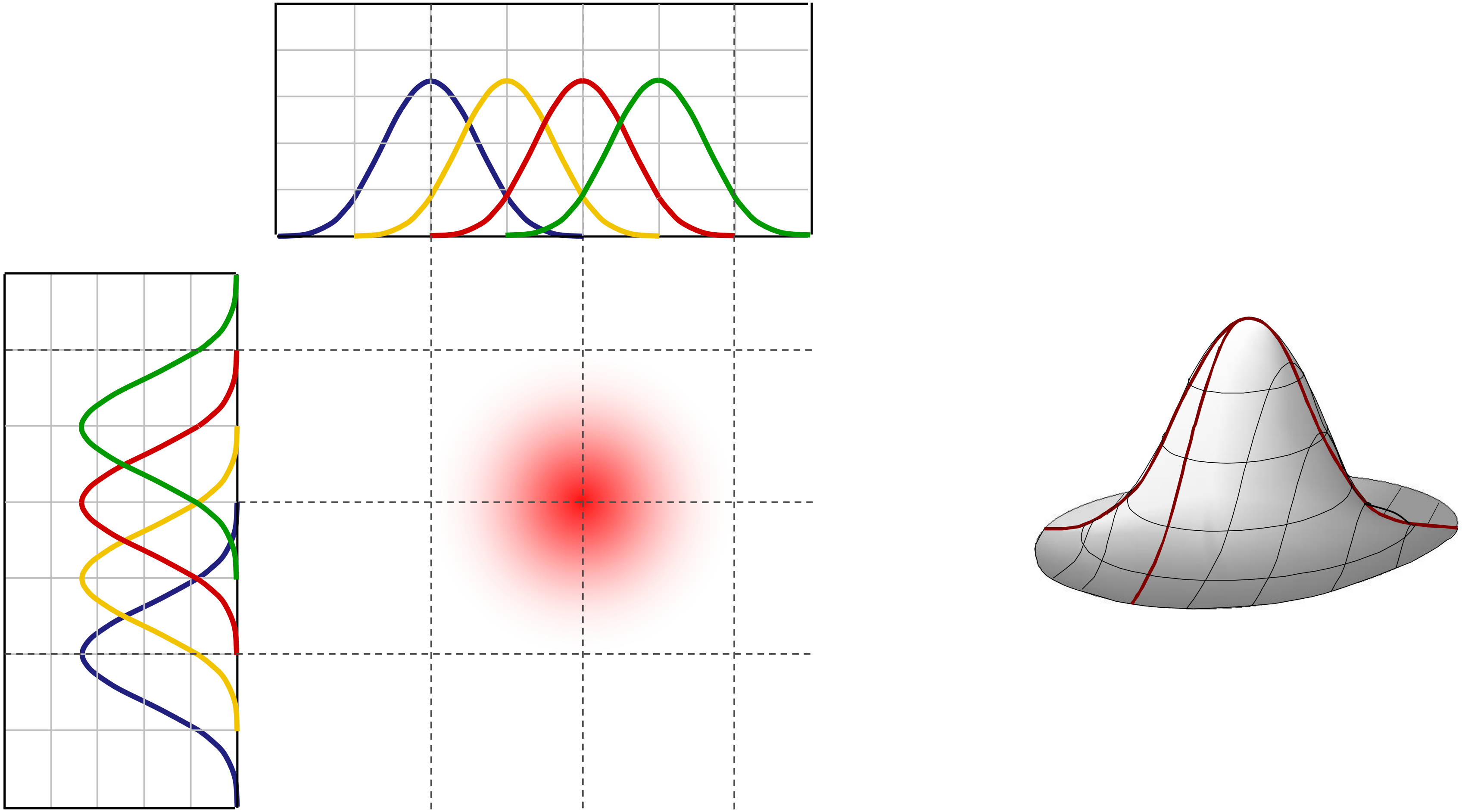
$$B^{(surf)} = \{b_1(u)b_1(v), b_1(u)b_2(v), \dots, b_n(u)b_n(v)\}$$
$$b_i \cdot b_j: (a, b)^2 \to \mathbb{R}$$

Tensor Product Surfaces

Tensor product basis

	$b_1(u)$	$b_2(u)$	$b_3(u)$	$b_4(u)$
$b_1(v)$	$b_1(v)b_1(u)$	$b_1(v)b_2(u)$	$b_1(v)b_3(u)$	$b_1(v)b_4(u)$
$b_2(v)$	$b_2(v)b_1(u)$	$b_2(v)b_2(u)$	$b_2(v)b_3(u)$	$b_2(v)b_4(u)$
$b_3(v)$	$b_3(v)b_1(u)$	$b_3(v)b_2(u)$	$b_3(v)b_3(u)$	$b_3(v)b_4(u)$
$b_4(v)$	$b_4(v)b_1(u)$	$b_4(v)b_2(u)$	$b_4(v)b_3(u)$	$b_4(v)b_4(u)$

2D Spline Basis



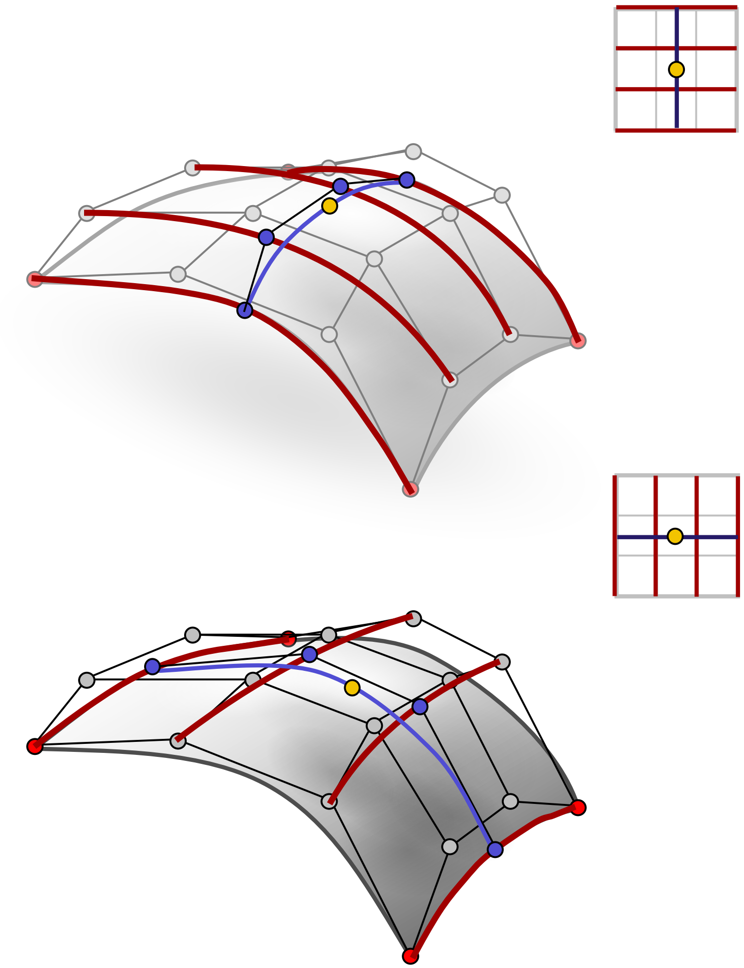
Tensor Product Surfaces

Tensor Product Surfaces:

$$f(u,v) = \sum_{i=1}^{n} \sum_{j=1}^{n} b_i(u)\, b_j(v) \cdot \mathbf{p}_{i,j}$$

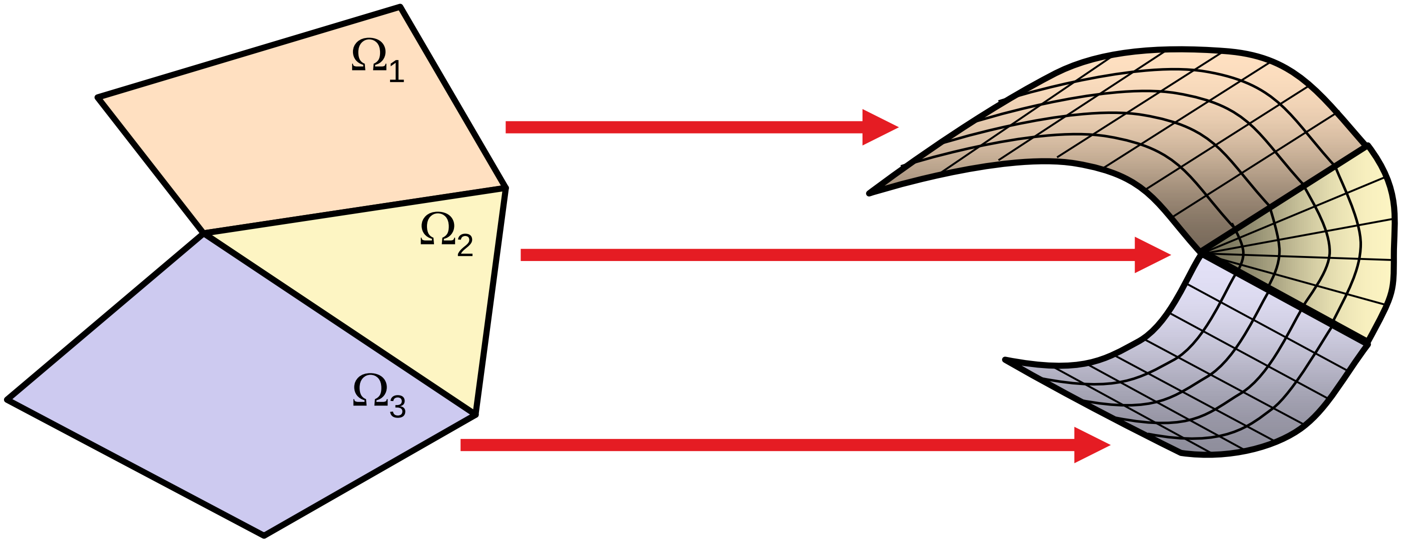
$$= \sum_{i=1}^{n} b_i(u) \sum_{j=1}^{n} b_j(v) \cdot \mathbf{p}_{i,j}$$

$$= \sum_{j=1}^{n} b_j(v) \sum_{i=1}^{n} b_i(u) \cdot \mathbf{p}_{i,j}$$



- "Curves of Curves"
- Order does not matter

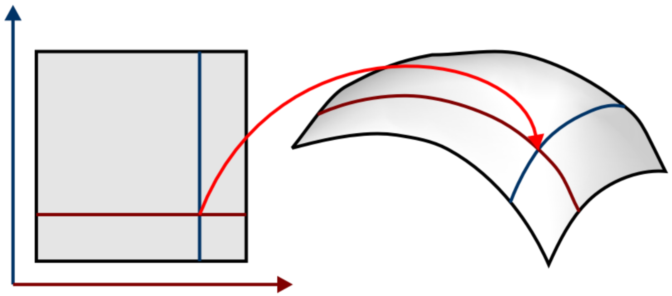
Meshes of Spline Patches



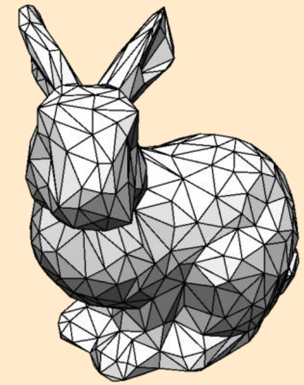
Spline mesh

- Quad-mesh
- Continuity conditions etc. (lecture of its own)
- Popular choice: "Trimmed NURBS"
  - Rational, non-uniform B-Splines; higher degrees
  - Trimming curves in parameter domain

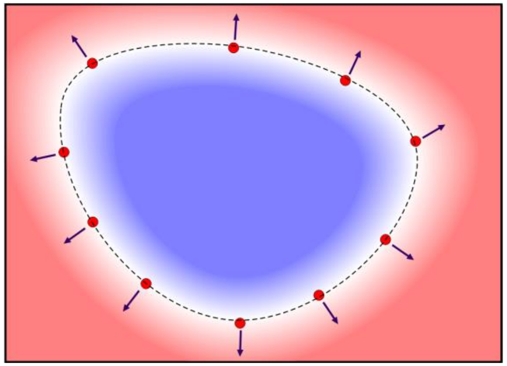
Modeling Zoo



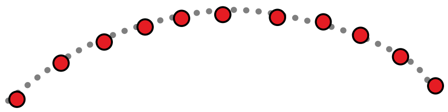
Parametric Models



Primitive Meshes



Implicit Models

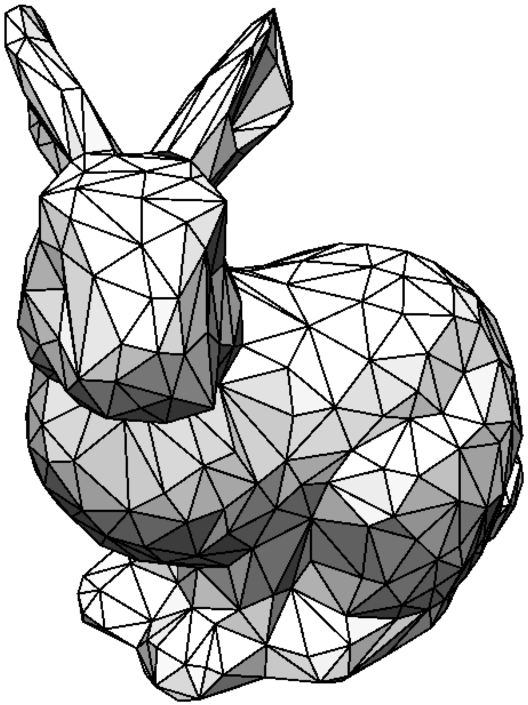


Point-Based Models

Primitive Meshes

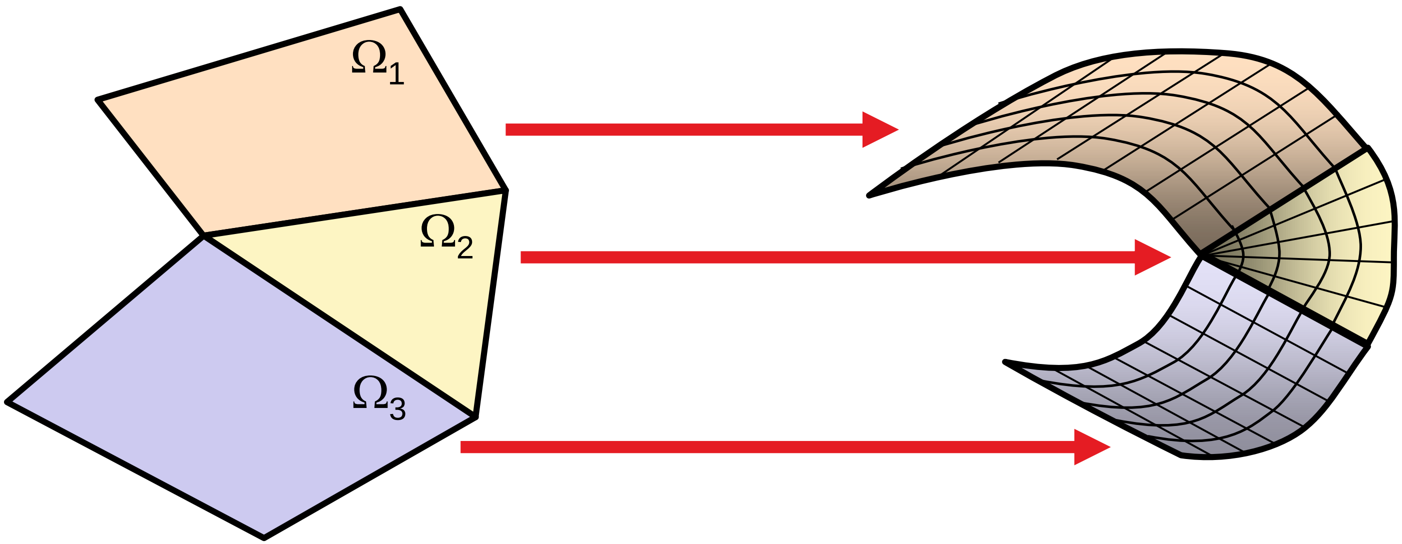
Primitive Meshes

- Collection of geometric primitives
  - Triangles, Quadrilaterals
  - Spline patches etc.
- Primitives are typically parametric surfaces



Triangle meshes rule the world (“triangle soup”)

Primitive Meshes



Composite model

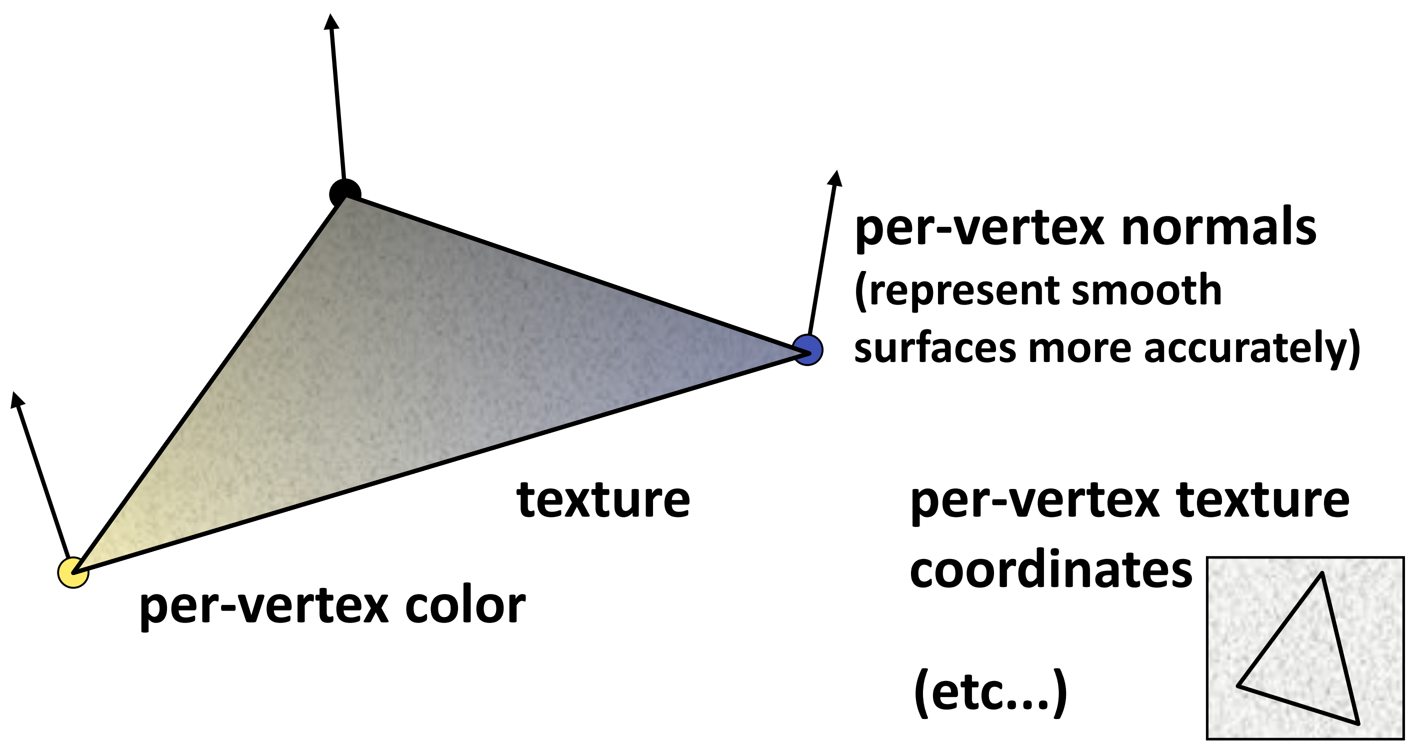
- Mesh encodes topology, rough shape
- Primitive parameter encode local geometry

Triangle Meshes

Attributes

How to define a triangle?

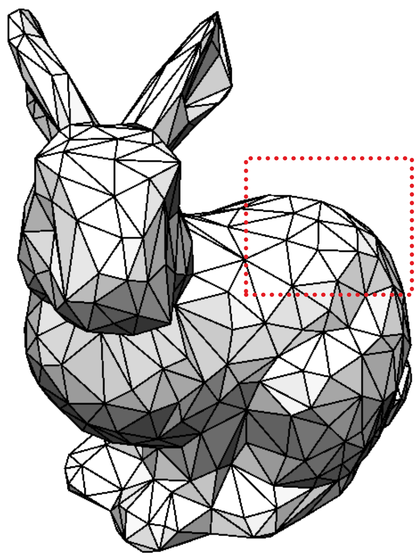
- We need three points in $\mathbb{R}^3$(obviously).
- But we can have more:

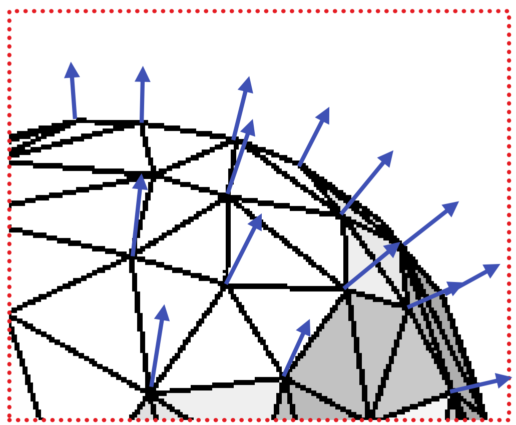


Shared Attributes in Meshes

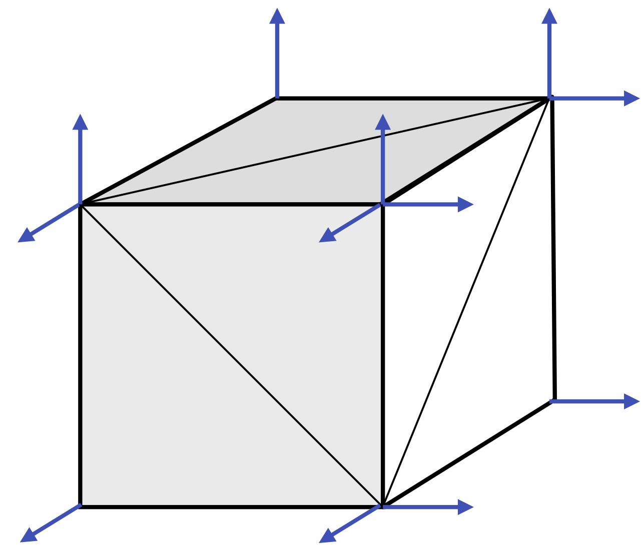
In Triangle Meshes:

- Attributes might be shared or separated:





adjacent triangles share normals



adjacent triangles have separated normals

Attributes

In general:

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

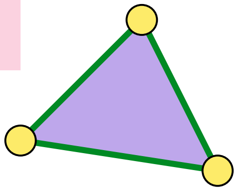
Data Structures

Simplest: List of vertices, edges, triangles

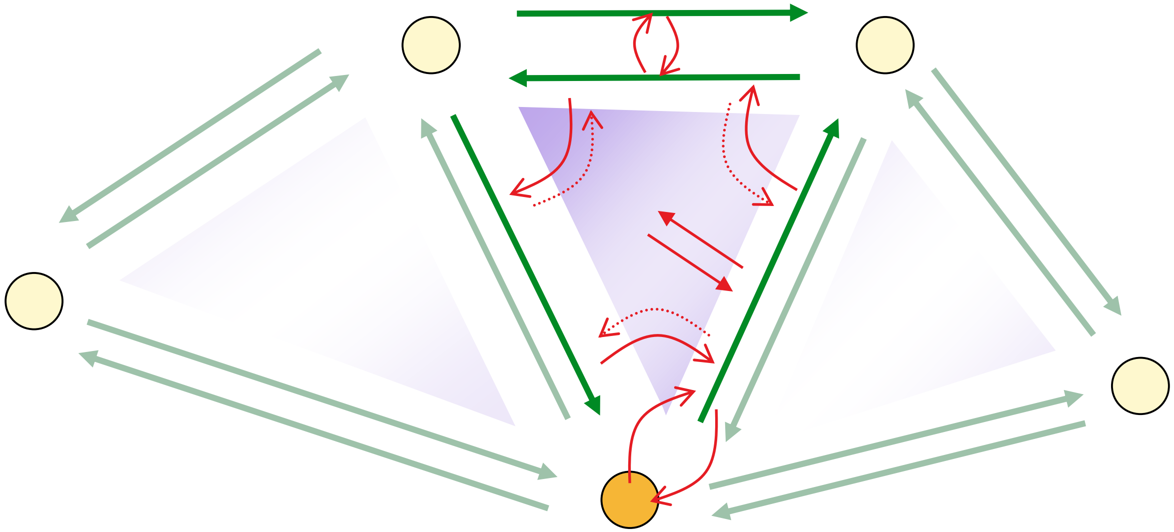
v_1: (posx posy posy), attrib_1, ..., attrib_{nav}
...
v_{nv}: (posx posy posy), attrib_1, ..., attrib_{nav}

e_1: (index_1 index_2), attrib_1, ..., attrib_{nae}
...
e_{ne}: (index_1 index_2), attrib_1, ..., attrib_{nae}

t_1: (idx_1 idx_2 idx_3), attrib_1, ..., attrib_{nat}
...
t_{nt}: (idx_1 idx_2 idx_3), attrib_1, ..., attrib_{nat}



Adjacency Data Structure



Half edge data structure:

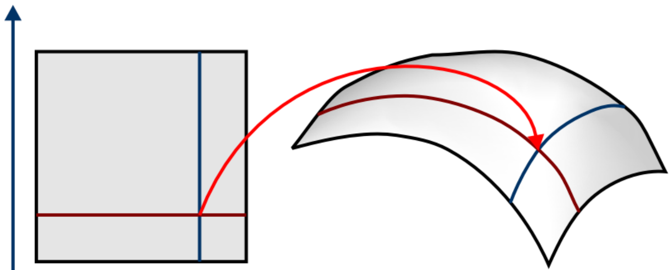
- Half edges, connected by clockwise / ccw pointers
- Pointers to opposite half edge
- Pointers to/from start vertex of each edge
- Pointers to/from left face of each edge

Implementation

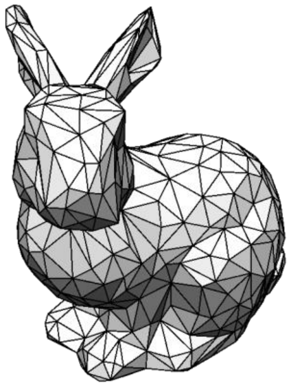
```
// a half edge
struct HalfEdge {
   HalfEdge* next;
   HalfEdge* previous;
   HalfEdge* opposite;

   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 */
};
```

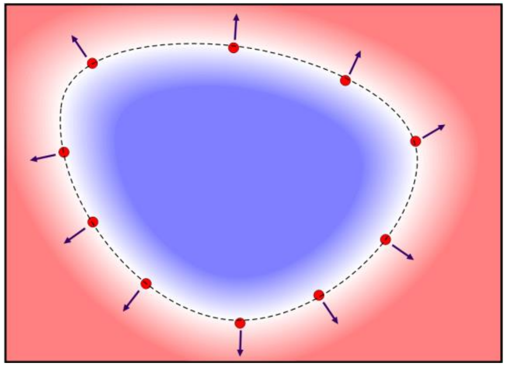
Modeling Zoo



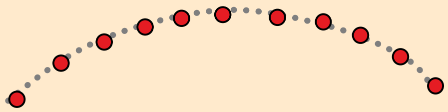
Parametric Models



Primitive Meshes



Implicit Models

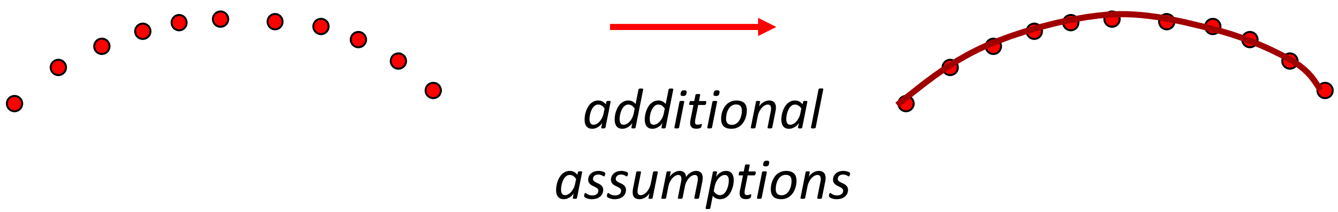


Point-Based Models

Particle Representations

Point-based Representations

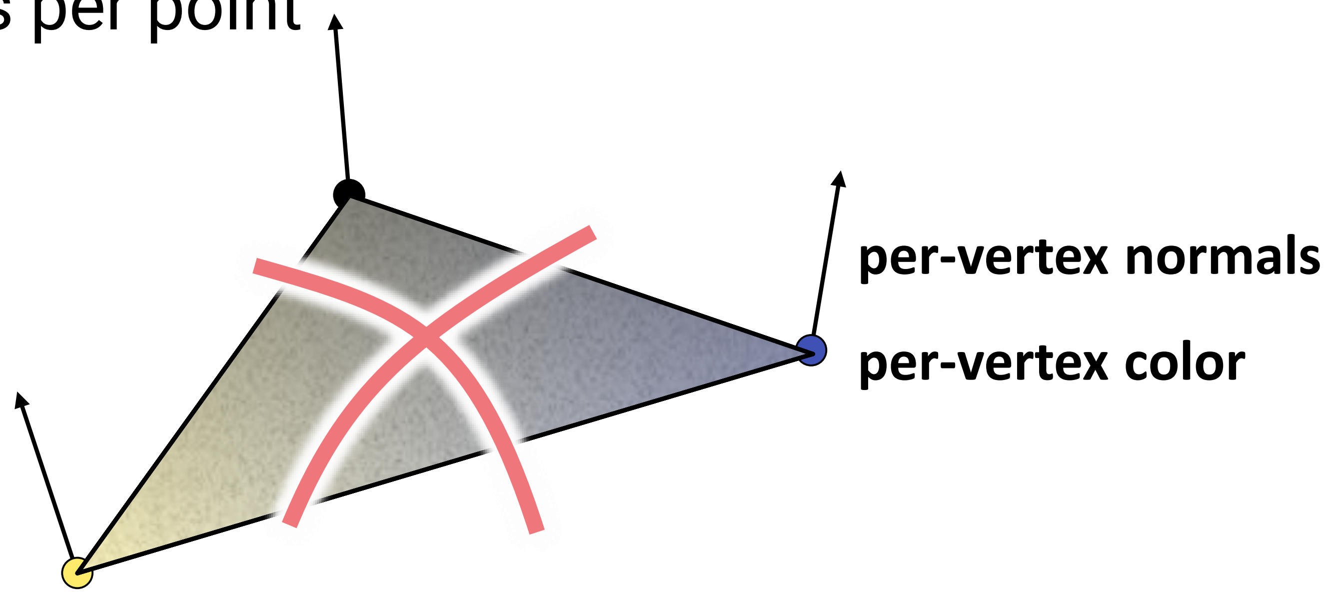
- Set of points
- Points are (irregular) *sample* of the object
- Need additional information to deal with “the empty space around the particles”



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

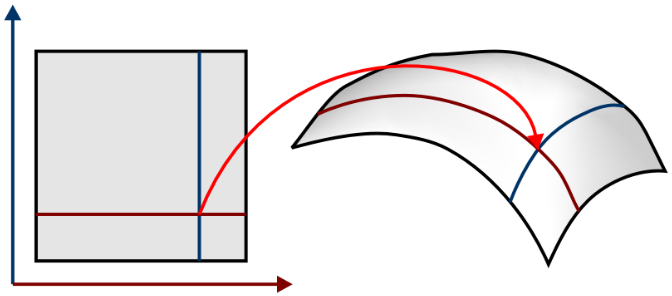
Geometric Modeling

3D Scanners

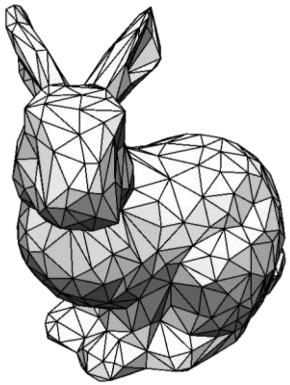
- 3D scanner yield point clouds
  - Have to deal with points anyway
- Algorithms that directly work on "point clouds"

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

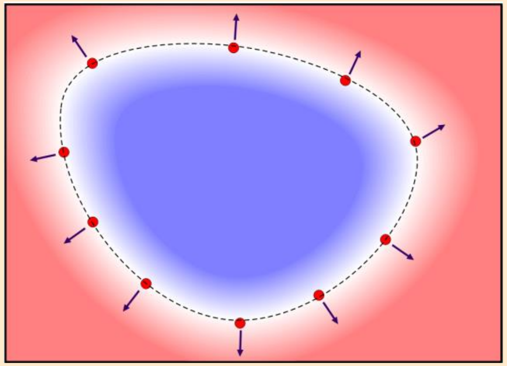
Modeling Zoo



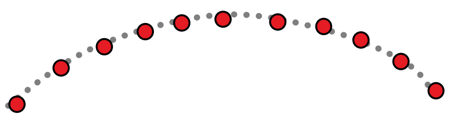
Parametric Models



Primitive Meshes



Implicit Models



Point-Based Models

Implicit Modeling

General Formulation:

- Curve / Surface

$$S = \{\mathbf{x} | f(\mathbf{x}) = 0\}$$

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

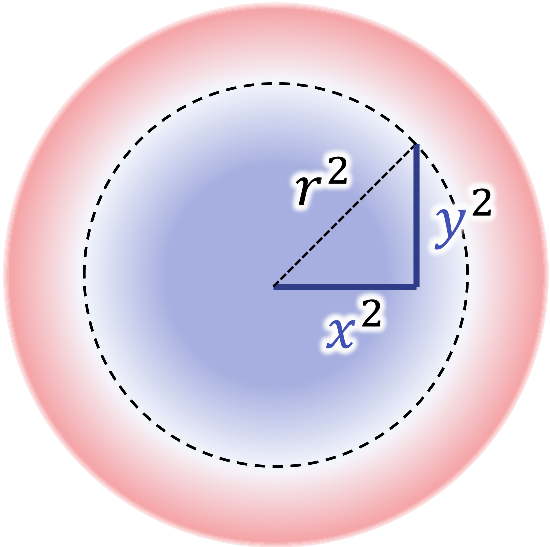
Surface described implicitly

- Set of points where f vanishes: $f(\mathbf{x}) = 0$
- Alternative notation: $S = f^{-1}(0)$
- Aka.: “Level set method”

Implicit Modeling

Example:

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



Special Case:

- Signed distance field
  - Function value is signed distance to surface
  - Negative means inside, positive means outside
  - Circle: $f_r(x,y) = \text{sign}(x^2 + y^2 - r^2)\sqrt{|x^2 + y^2 - r^2|}$

Implicit Modeling: Pros & Cons

Advantages:

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

Implicit Modeling: Pros & Cons

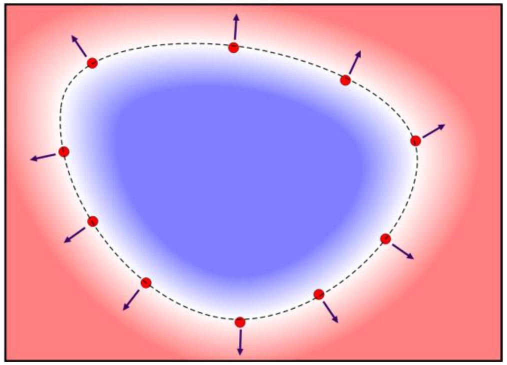
Disadvantages:

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

Implicit Function Types

Use depends on application:

- Signed implicit function
  - Solid modeling
  - Interior well defined
- Signed distance function
  - Frequently used
  - Constant gradient = stable surface definition
  - Distance values useful
- Squared distance function
  - Least-squares (Gaussian cross-section)
  - Modeling of noise
  - Surface extraction less stable (*gradient vanishes!*).



signed distance

Linear Representations

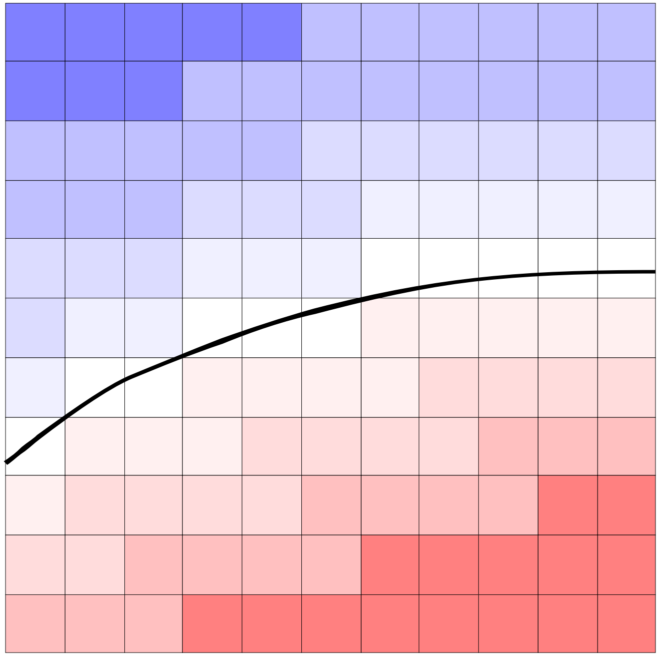
Two basic techniques

- Simple grids ("finite differences")
- Full linear ansatz ("finite elements")
  - Grids of basis functions
  - Hierarchical / adaptive grids
  - Radial basis functions / particles

Regular Grids

Discretization:

- Regular grid of values $f_{i,j}$
- Grid spacing h
- Often: Finite difference approximation



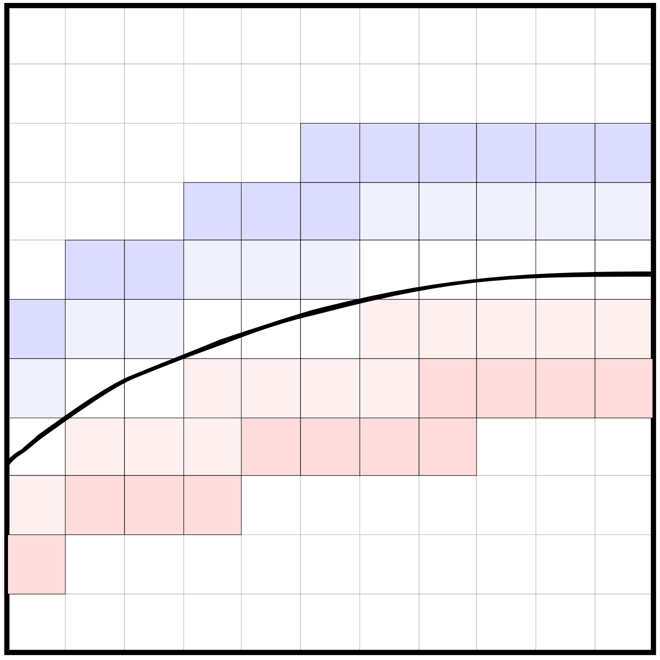
$$\frac{\partial}{\partial_x} f(\mathbf{x}) = \frac{1}{h}\left(f_{i(\mathbf{x}),j(\mathbf{x})} - f_{i(\mathbf{x})-1,j(\mathbf{x})}\right) + O(h)$$

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

Regular Grids

Variant:

- Use only cells near the surface
- Saves storage
  & computation time

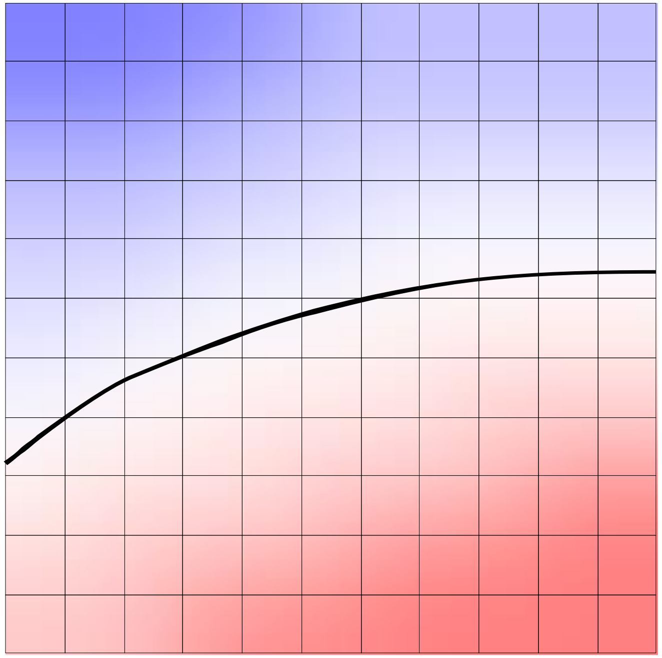


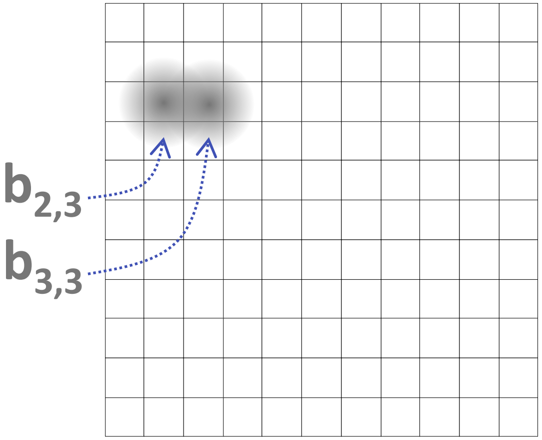
Regular Grids of Basis Functions

Discretization (2D):

- Basis function in each grid cell: $b_{i,j} = b(x - i, y - j)$
- Then

$$f(\mathbf{x}) = \sum_{i=1}^{d} \sum_{j=1}^{d} \lambda_{i,j} b_{i,j}(\mathbf{x})$$

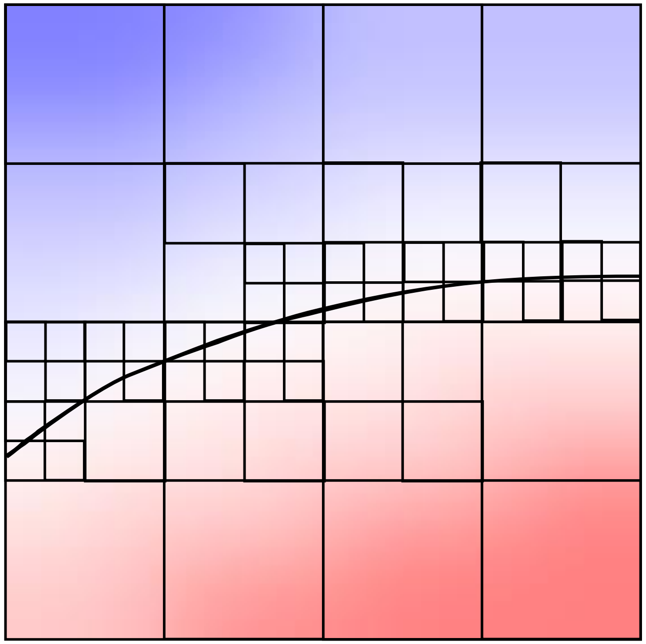




Adaptive Grids

Adaptive / hierarchical grid:

- Quadtree /octree tessellation of the domain
- Refine where more precision is necessary
- Basis functions in each cell
  - Size proportional to cell size

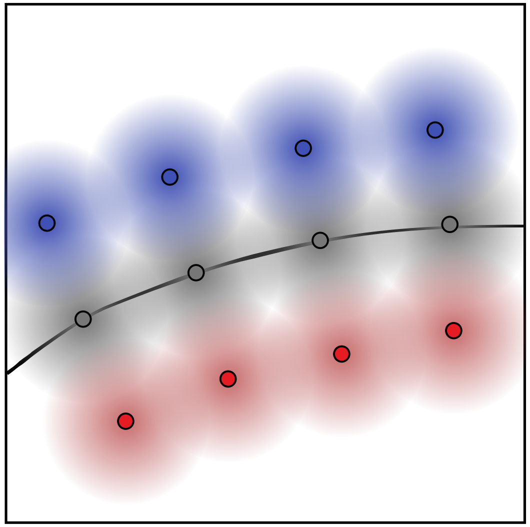


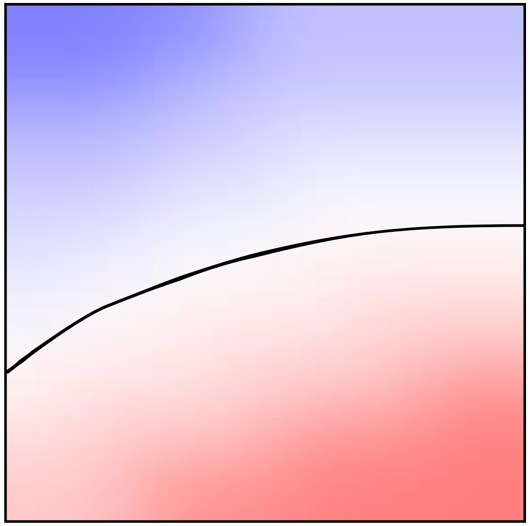
Particle Methods

Particle methods / radial basis functions:

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

$$f(\mathbf{x}) = \sum_{i=1}^{d} \lambda_i b(\mathbf{x} - \mathbf{x}_i)$$





Implicit Surfaces

Level Set Extraction

Algorithms

Converting: Implicit → Meshes

- Standard Algorithm: *Marching Cubes*

Marching Cubes

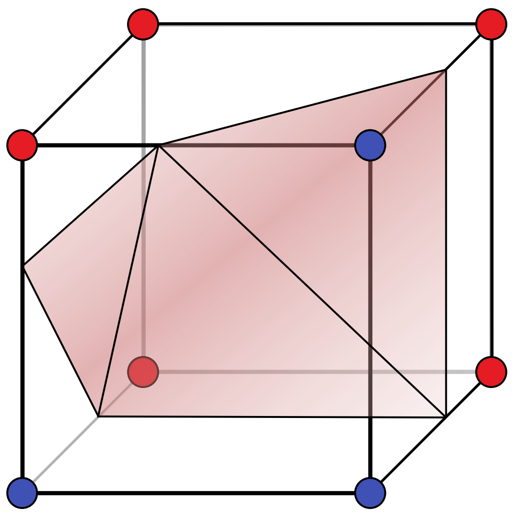
Marching Cubes:

- Simple idea
  - Define and solve a fixed complexity, local problem.
  - Compute a full solution by solving many such local problems incrementally.

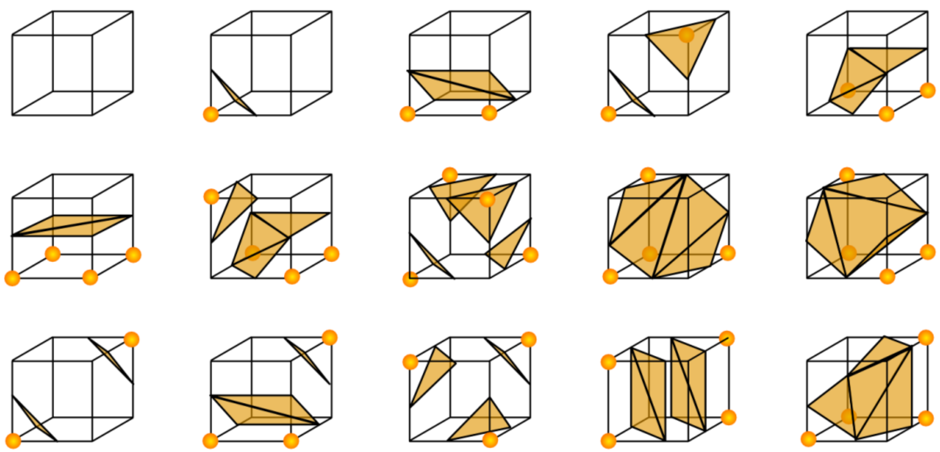
Marching Cubes

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

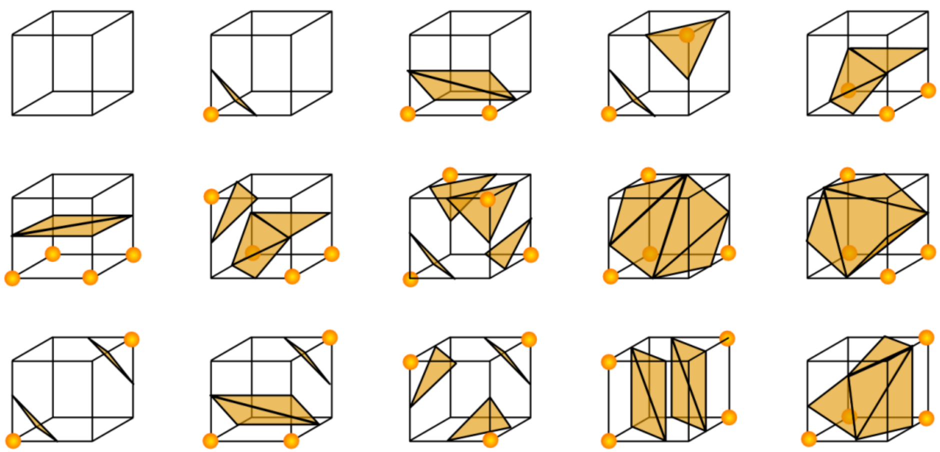


[source: Wikipedia]

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



[source: Wikipedia]

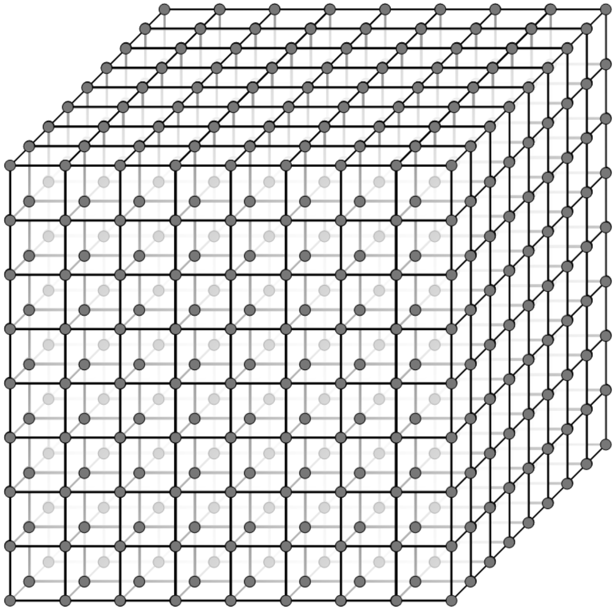
How to place the vertices?

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

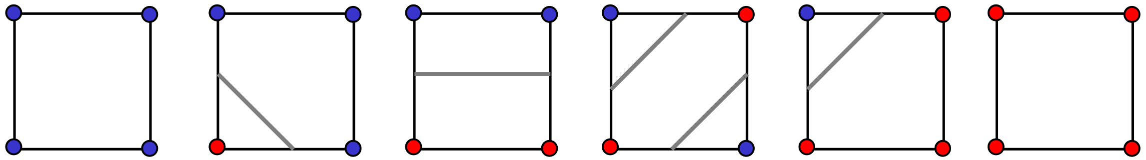
Outer Loop

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

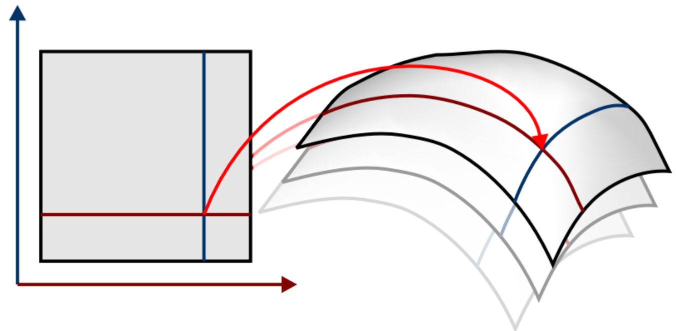


Marching Squares:

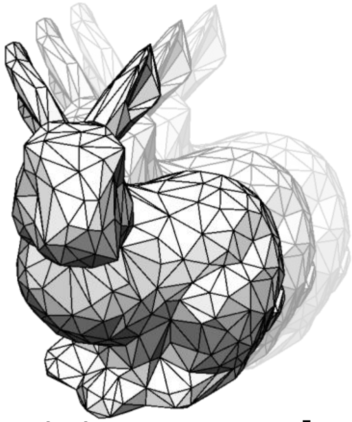
- 2D version of the algorithm
- Same idea, fewer cases

Dynamic Processes

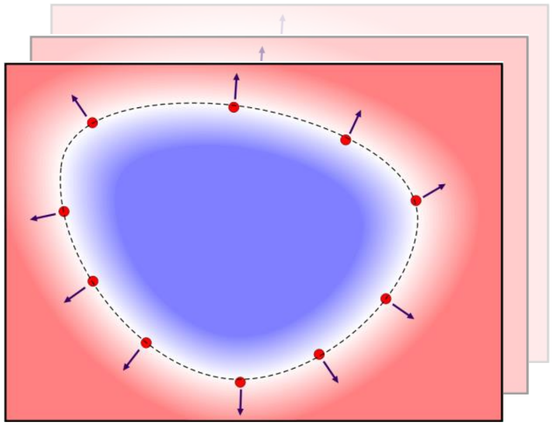
Time Dependent Parameters



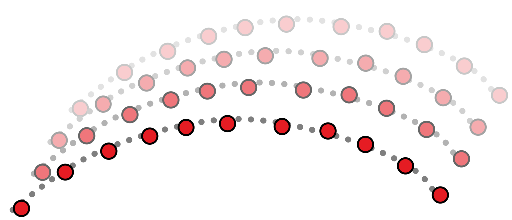
Parametric Models



Primitive Meshes



Implicit Models



Point-Based Models

Simple Idea

Make Parameters Functions of time t

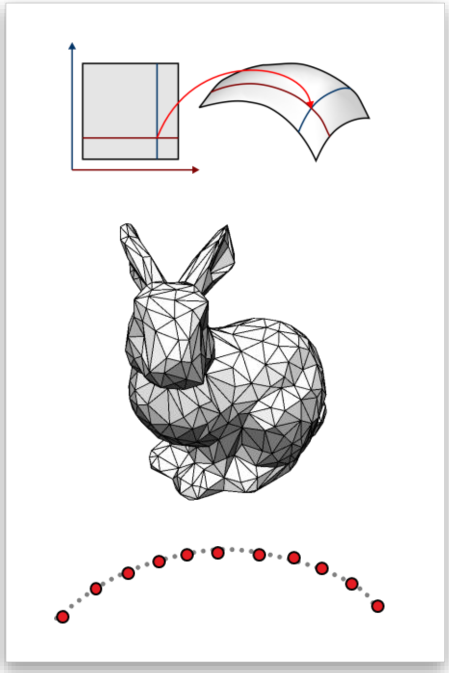
- Point-Based: Moving points
- Meshes: Moving triangles
  - Moving spline-patch-control points
  - Moving tetraheda (volumetric "tet-meshes")
- Implicit functions $f(\mathbf{x}, t)$

Time-discretization

- "Finite-differences" $\rightarrow$ Array over time
- Temporal basis functions

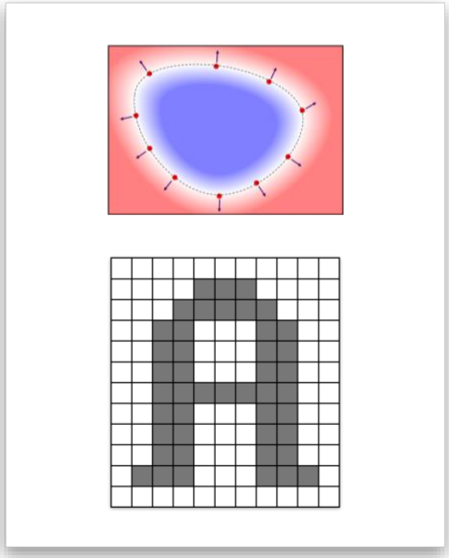
Geometric Representations Summary

Classification



“Lagrangian” Discretization

- Parametric surfaces, meshes, particles
- Parametrization of the geometry
- Variables move with geometry



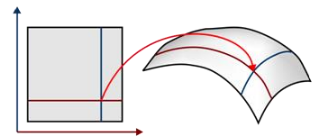
“Eularian” Discretization

- Bitmaps, voxels, level-set methods
- Parametrization of space
- Geometry moves through space
  - Variables remain fixed to spatial location

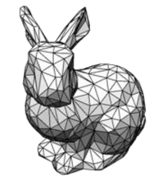
Summary

Summary

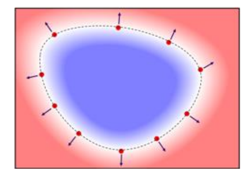
- Different representations
  - No silver bullet
  - All representations work
    in principle for all problems
  - Approximate conversion possible
- Effort application dependent
  - Conceptual effort
  - Computational effort



Parametric Models



Primitive Meshes



Implicit Models

Particle Models

Summary

Summary

- Linear ansatz is our friend

$$f(\mathbf{x}) = \sum_{i=1}^{d} \lambda_i b_i(\mathbf{x})$$

  - Controls shape
  - Linear part easy to control