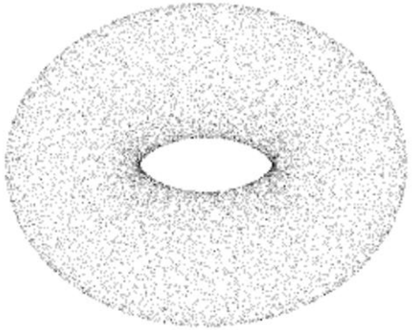
3D Scanning & Motion Capture

Surface Representations

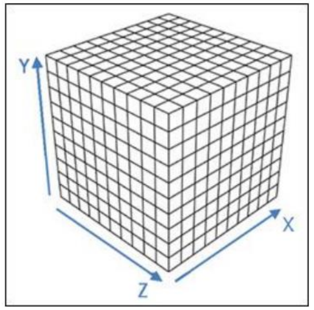
Prof. Matthias Nießner

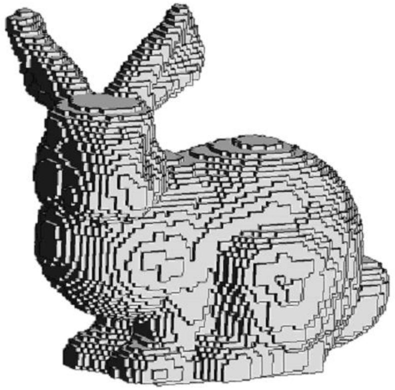
Last Lecture: What is “3D”?

- Point Clouds

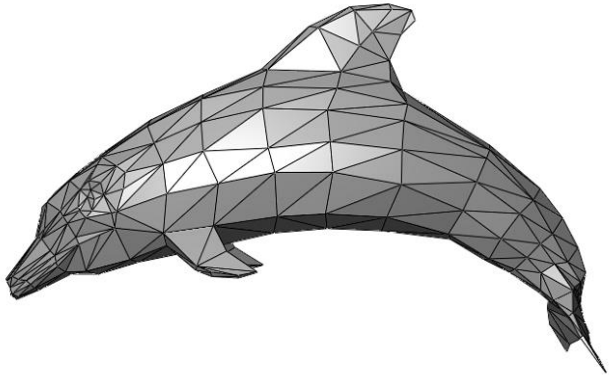


- Voxels

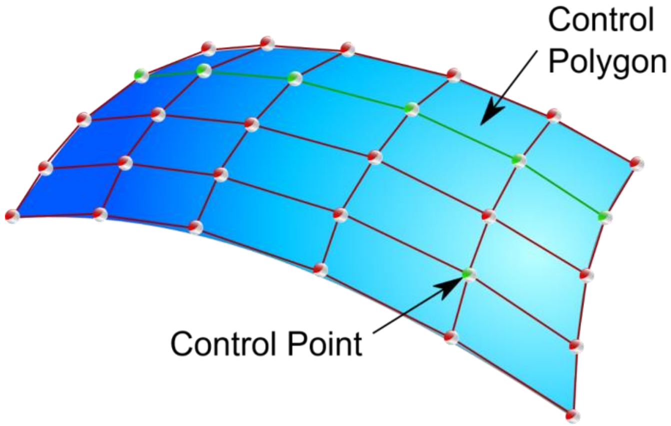




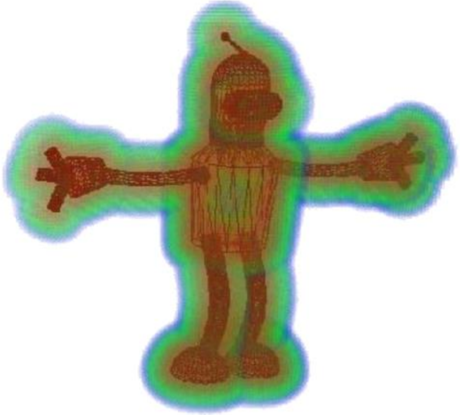
- Polygonal Meshes

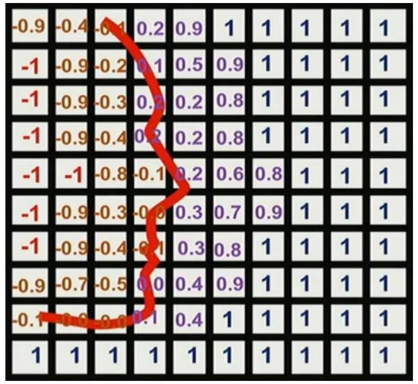


- Parametric Surfaces



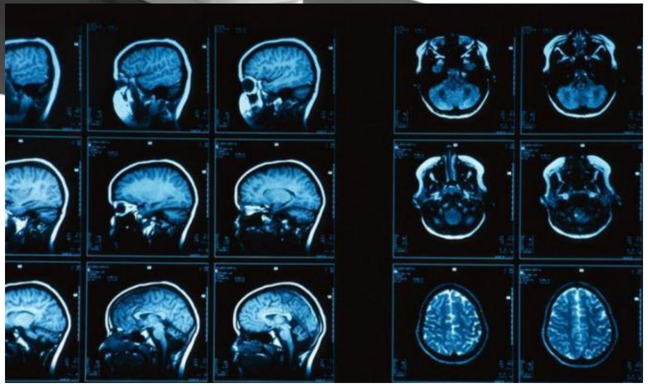
- Implicit Surfaces

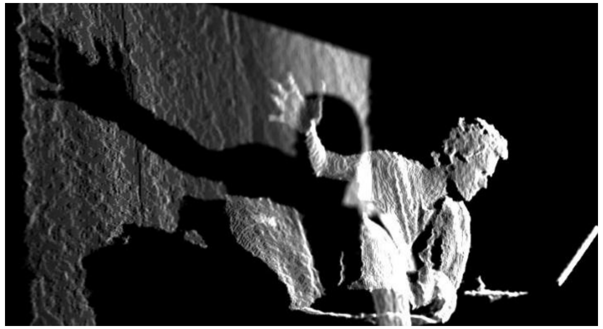


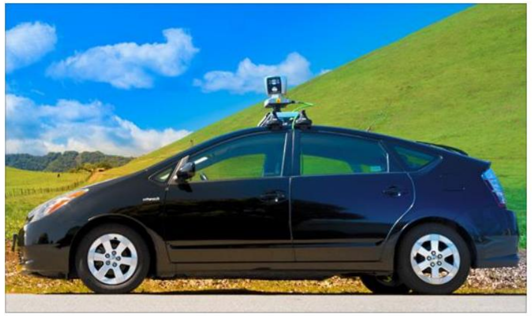


Truncated Signed Distance Field (TSDF)

Last Lecture: How to obtain “3D”?



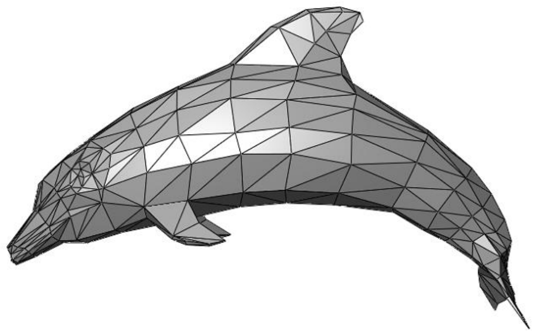




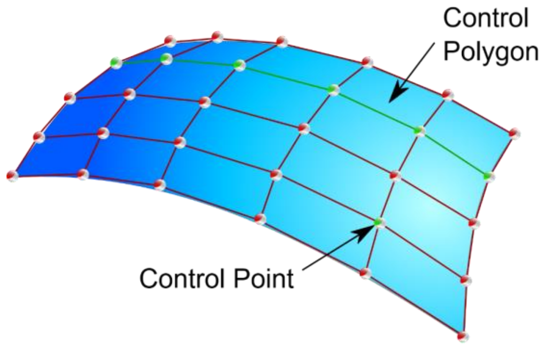
Today: Surface Representations

Overview

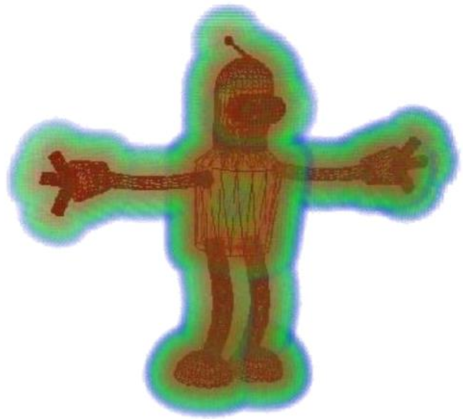
- Polygonal Meshes



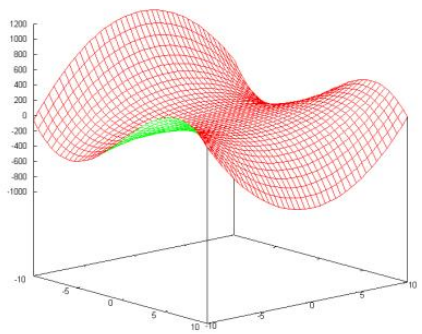
- Parametric Surfaces



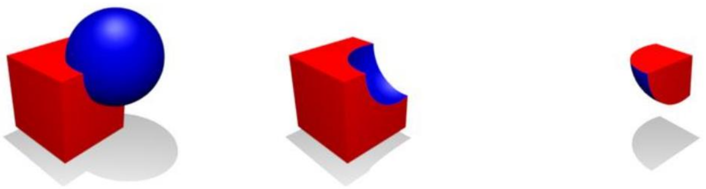
- Implicit Surfaces



- Explicit Surfaces

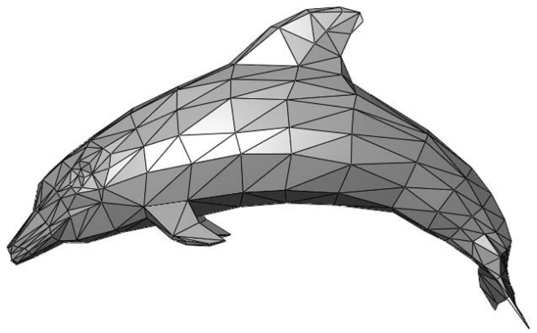


- Constructive Solid Geometry

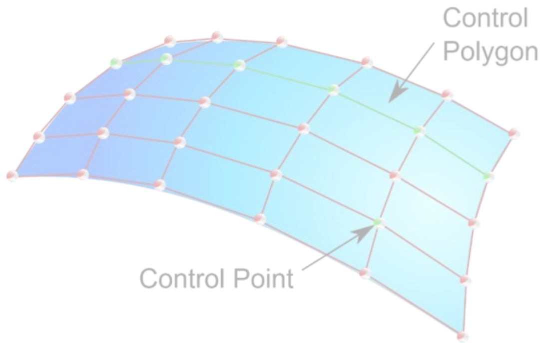


Overview

- Polygonal Meshes

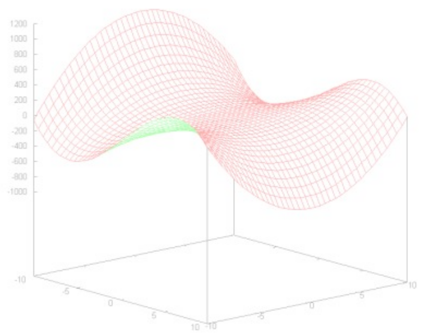


- Parametric Surfaces



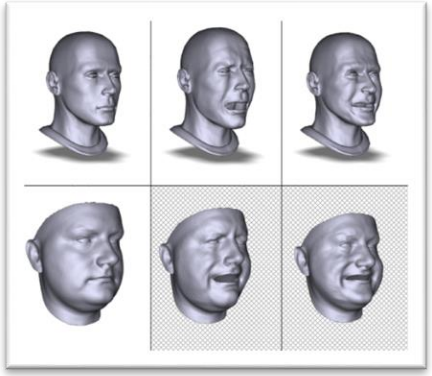
- Implicit Surfaces

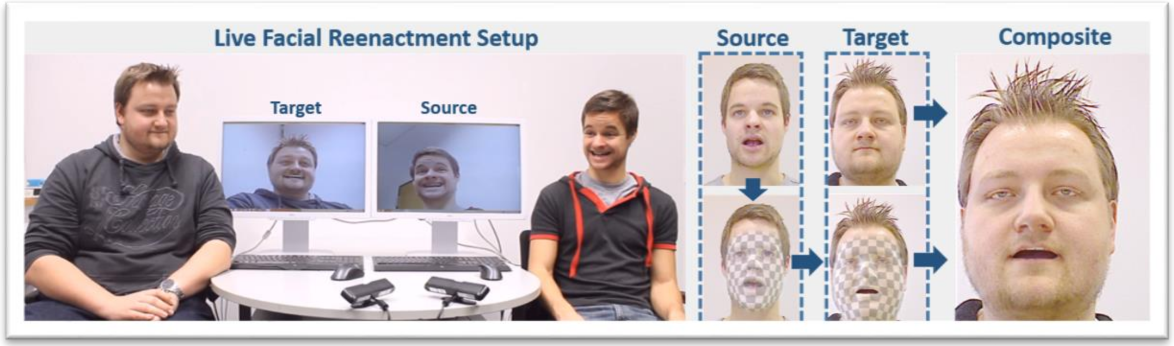
- Explicit Surfaces

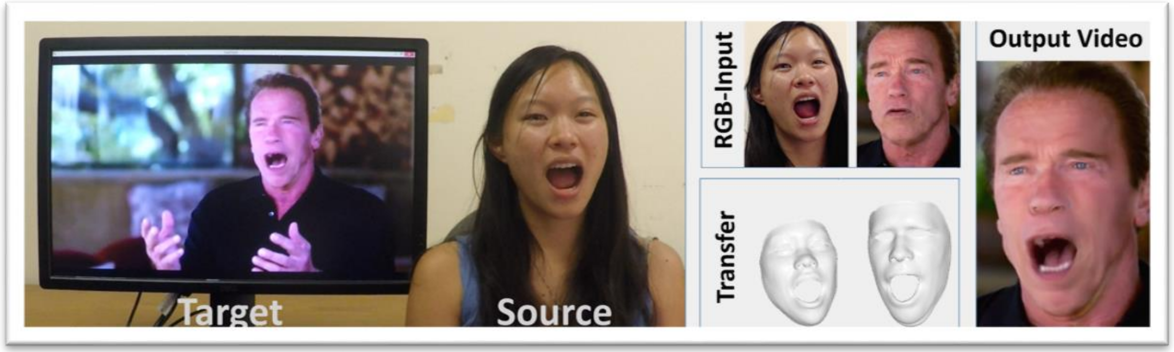


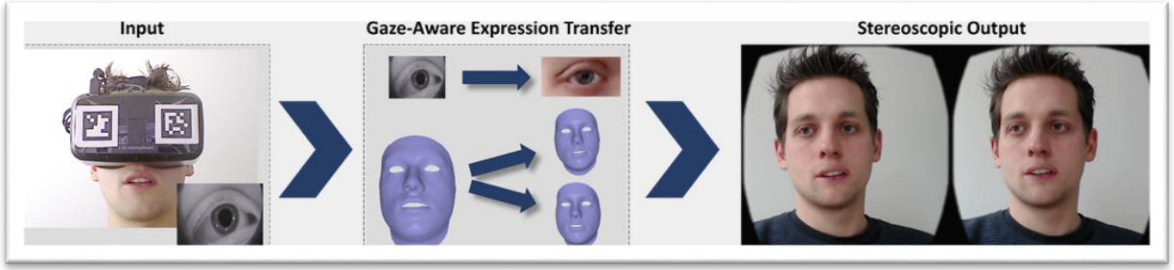
- Constructive Solid Geometry

Polygonal Meshes



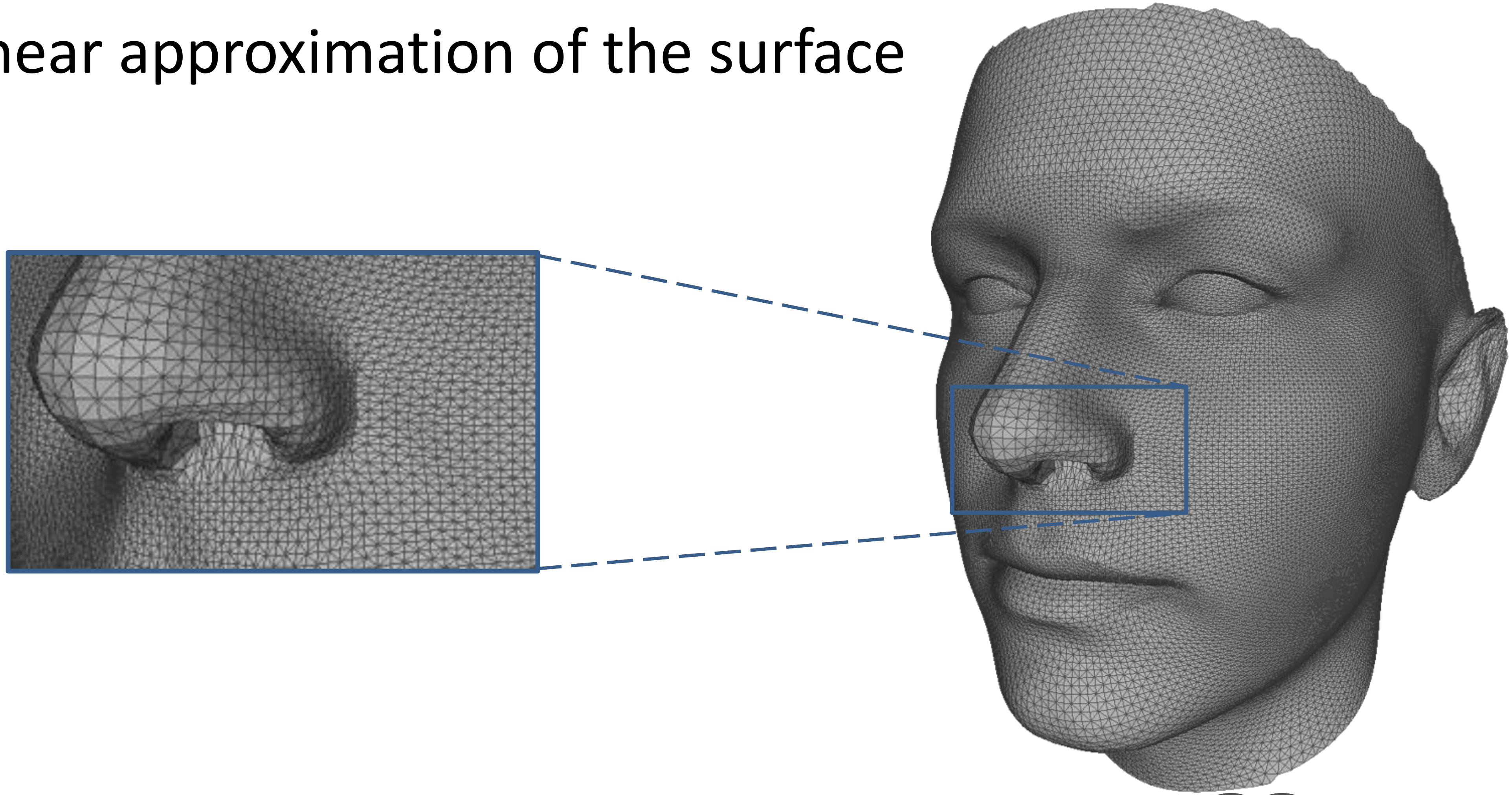






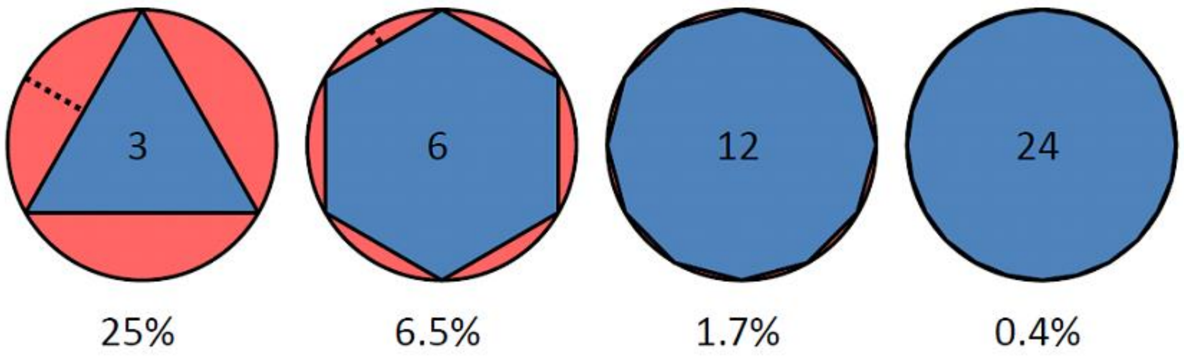
Polygonal Meshes

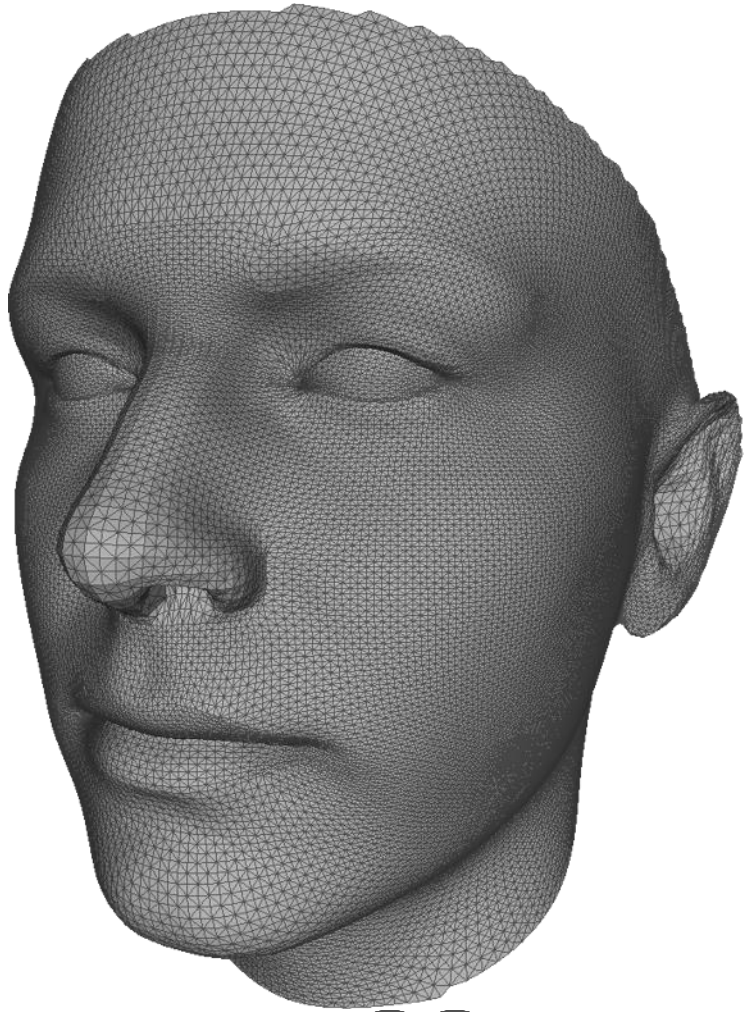
- Piecewise linear approximation of the surface



Polygonal Meshes

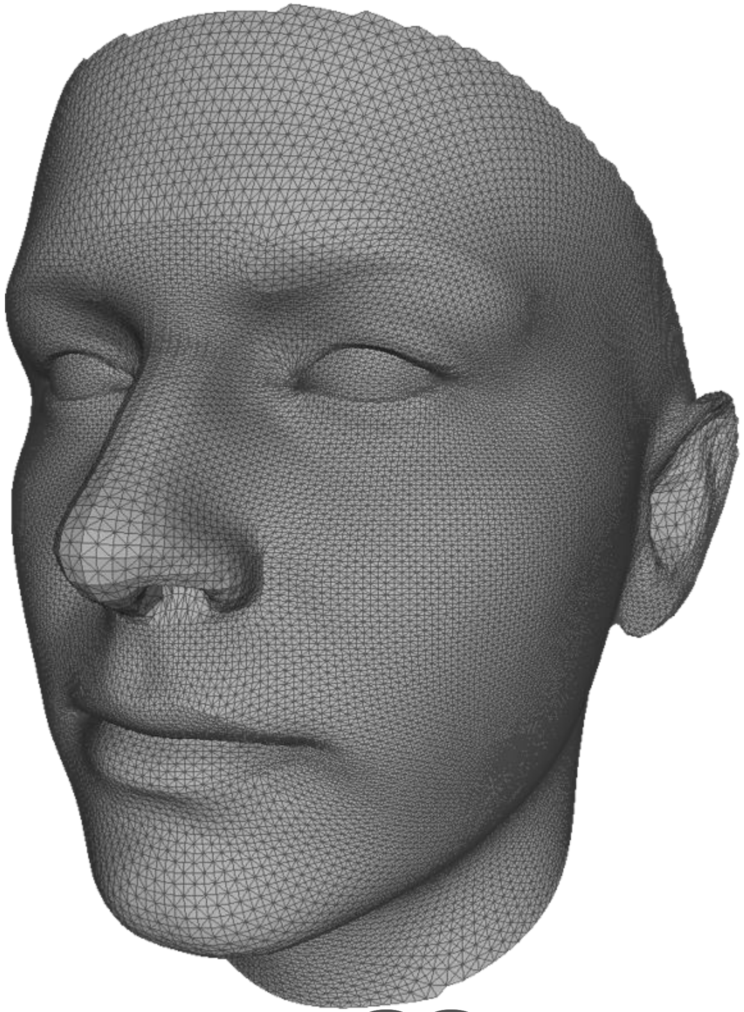
- Piecewise linear approximation of the surface
  - Error $O(h^2)$





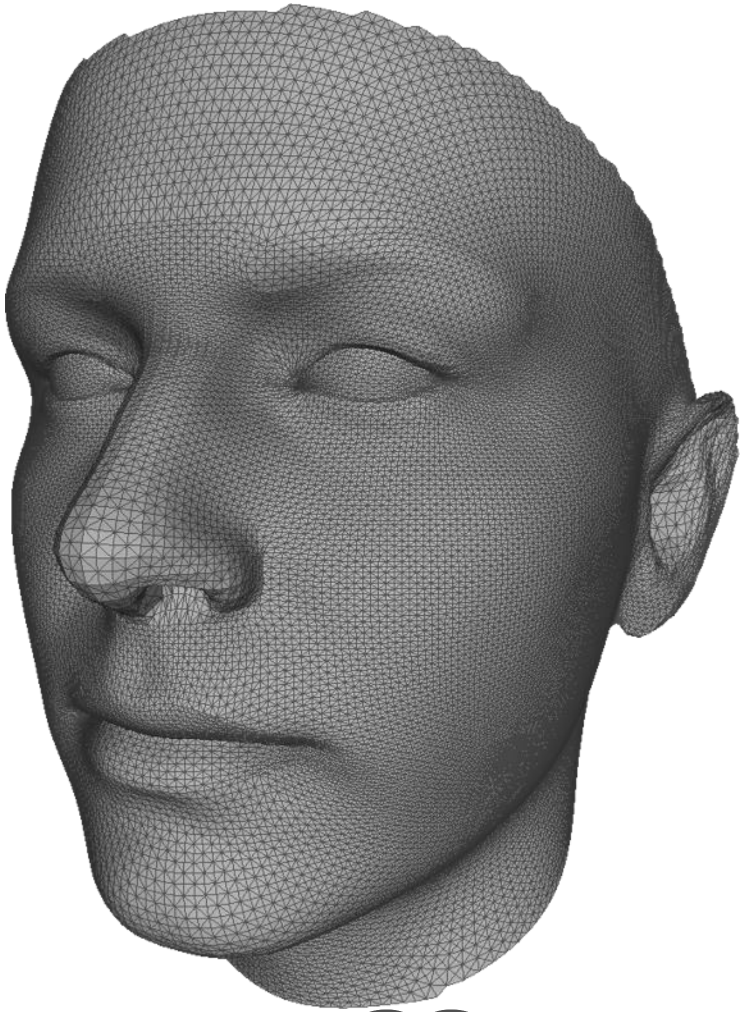
Polygonal Meshes

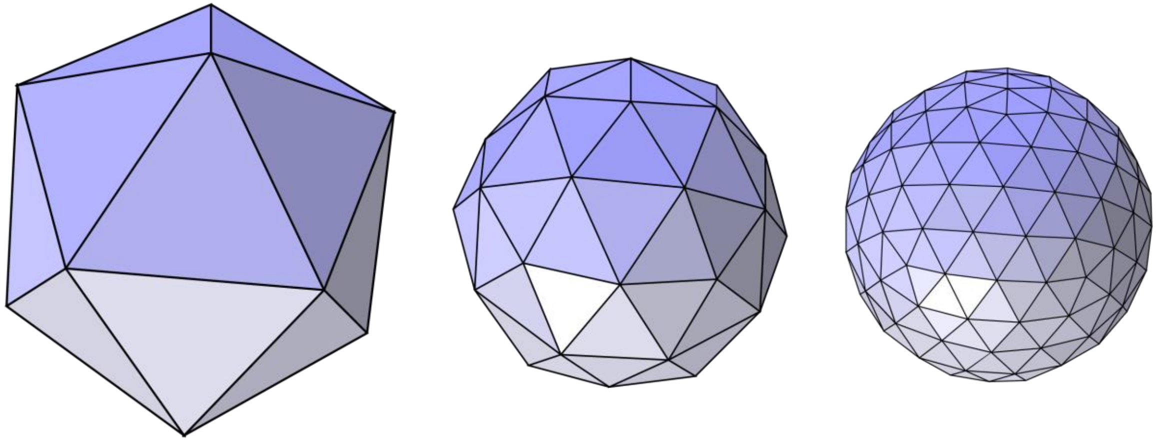
- Piecewise linear approximation of the surface
  - Error $O(h^2)$
  - Arbitrary topology



Polygonal Meshes

- Piecewise linear approximation of the surface
  - Error $O(h^2)$
  - Arbitrary topology
  - Adaptive refinement (subdivision)



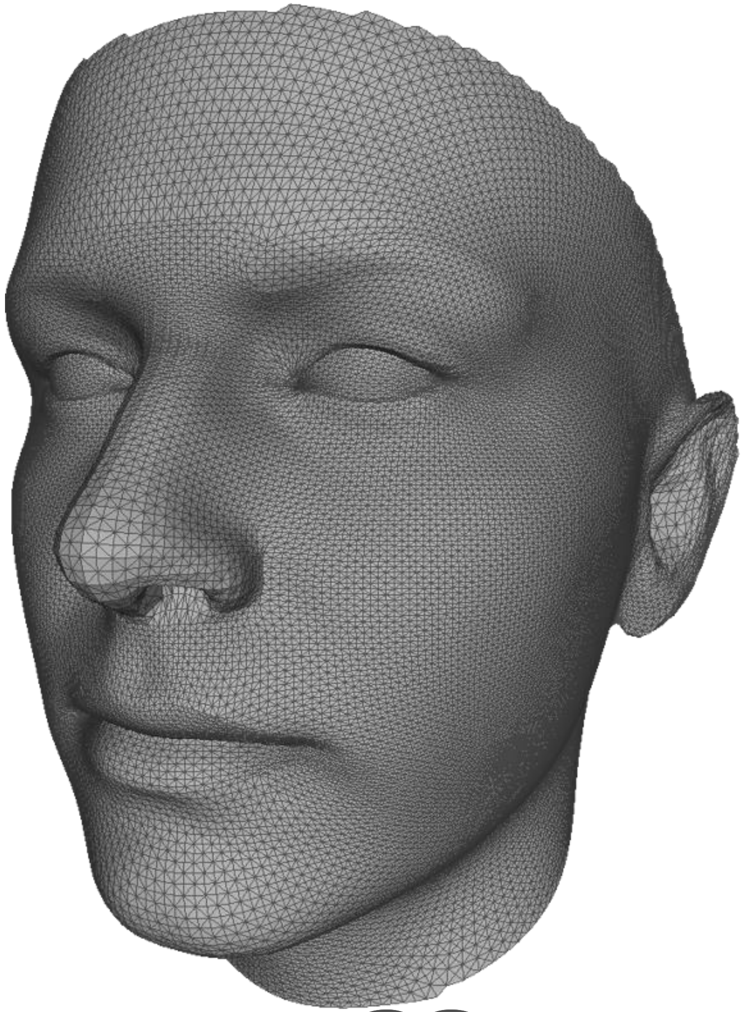


Loop Subdivision Surface

Polygonal Meshes

- Piecewise linear approximation of the surface
  - Error $O(h^2)$
  - Arbitrary topology
  - Adaptive refinement (subdivision)
  - Efficient rendering

Rasterization on modern GPUs is darn fast!
E.g., my old 1080 GTX rasters 2-3 mio polygons in 0.2-0.3ms (including textures, etc.)

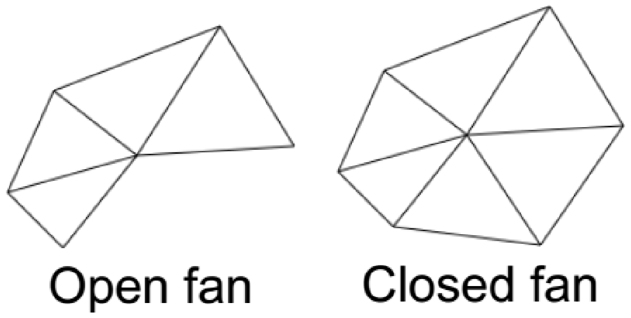


Polygonal Meshes

- A polygonal mesh is a collection of polygons satisfying certain restrictions

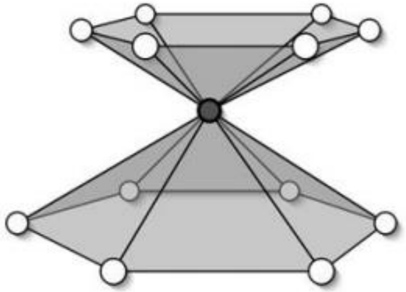
Polygonal Meshes

- A triangle mesh is called **manifold** if:
  - The intersection of two triangles is:
    - Empty
    - A common vertex
    - A common edge
  - Edges have
    - One adjacent triangle → border edge
    - Two adjacent triangles → inner edge
  - For a vertex the adjacent triangles
    - Build a single open fan → border vertex
    - Build a single closed fan → inner vertex

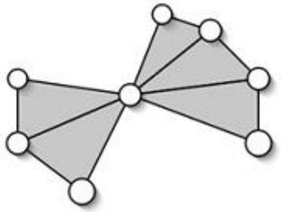


Polygonal Meshes

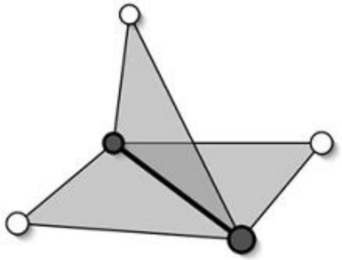
- Example of **non-manifold** meshes:



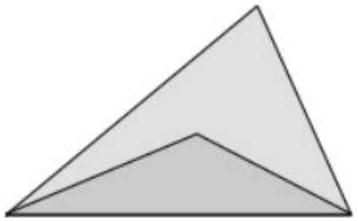
Two closed fans
at one vertex



Two open fans
at one vertex



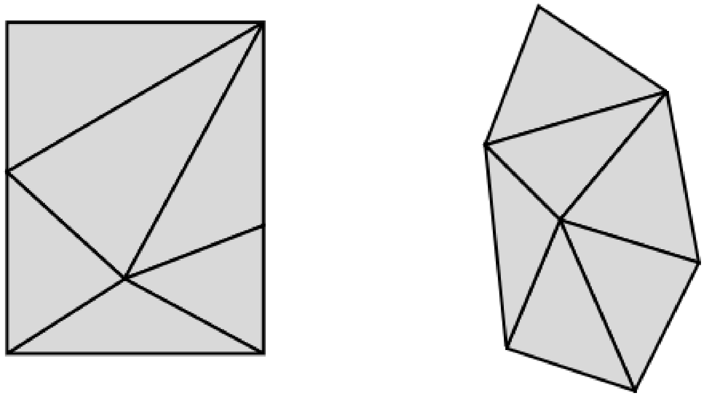
Three triangles
At one edge



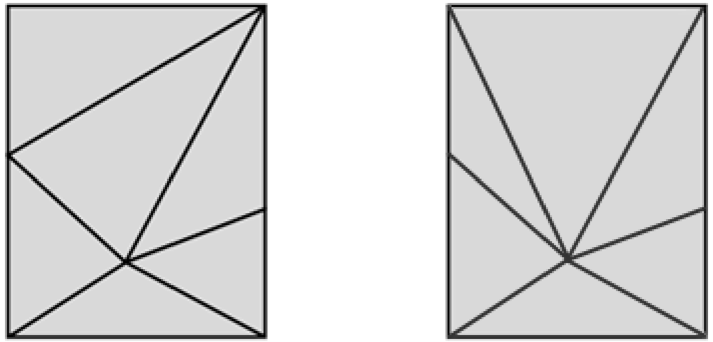
Intersection of triangles
Is an entire triangle

Polygonal Meshes

- Topology vs. Geometry
  - Topologically equivalent:
    - But different geometry



  - Geometrically equivalent:
    - But different topology

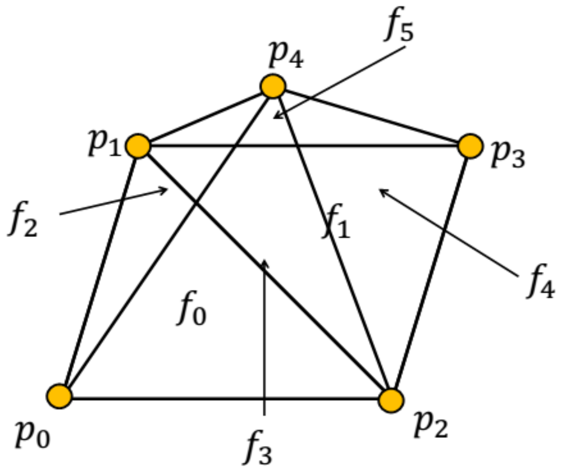


Polygonal Meshes

- Shared Vertex Data Structure
  - De facto standard for file formats (e.g., OFF, OBJ, STL, PLY, …)

vertex list
p_0: x_0, y_0, z_0;
p_1: x_1, y_1, z_1;
p_2: x_2, y_2, z_2;
p_3: x_3, y_3, z_3;
p_4: x_4, y_4, z_4;

face list
f_0: 0, 1, 2;
f_1: 2, 1, 3;
f_2: 1, 0, 4;
f_3: 4, 0, 2;
f_4: 4, 2, 3;
f_5: 4, 3, 1;



Polygonal Meshes

- Shared Vertex Data Structure
  - Cube.obj

```
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -0.999999
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000

f 2 3 4
f 8 7 6
f 5 6 2
f 6 7 3
f 3 7 8
f 1 4 8
f 1 2 4
f 5 8 6
f 1 5 2
f 2 6 3
f 4 3 8
f 5 1 8
```

Polygonal Meshes

- Shared Vertex Data Structure
  - Cube.ply

    ply also support supports binary encoding instead of ASCII

```
ply
format ascii 1.0
element vertex 8
property float x
property float y
property float z
element face 12
property list uchar uint vertex_indices
end_header
-1.000000 -1.000000 1.000000
-1.000000 1.000000 1.000000
-1.000000 1.000000 -1.000000
1.000000 1.000000 1.000000
1.000000 1.000000 -1.000000
1.000000 -1.000000 1.000000
1.000000 -1.000000 -1.000000
-1.000000 -1.000000 -1.000000
3 0 1 2
3 1 3 4
3 3 5 6
3 0 7 5
3 7 2 4
3 5 3 1
3 7 0 2
3 2 1 4
3 4 3 6
3 5 7 6
3 6 7 4
3 0 5 1
```

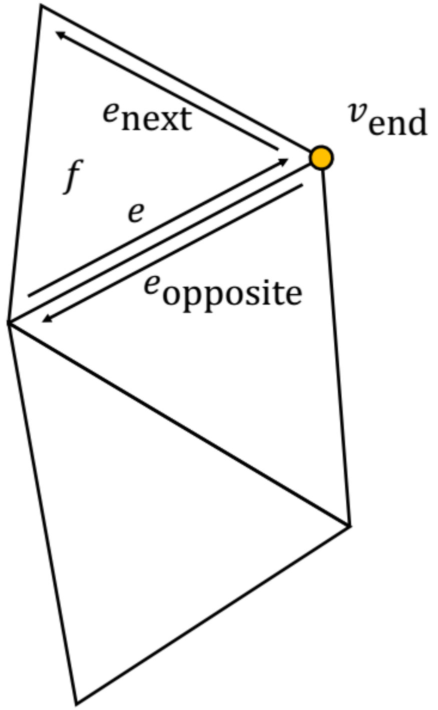
Polygonal Meshes

- Half-Edge Data Structure
  - Easy geometric queries → widely used in geometric computations
  - Only manifolds

Edge
- Edge ID e
- End vertex v_{end}
- Next edge e_{next}
- Opposite edge e_{opposite}
- Face f

Vertex
- Vertex ID
- Index of one incident edge

Face
- Face ID
- Index of one incident edge



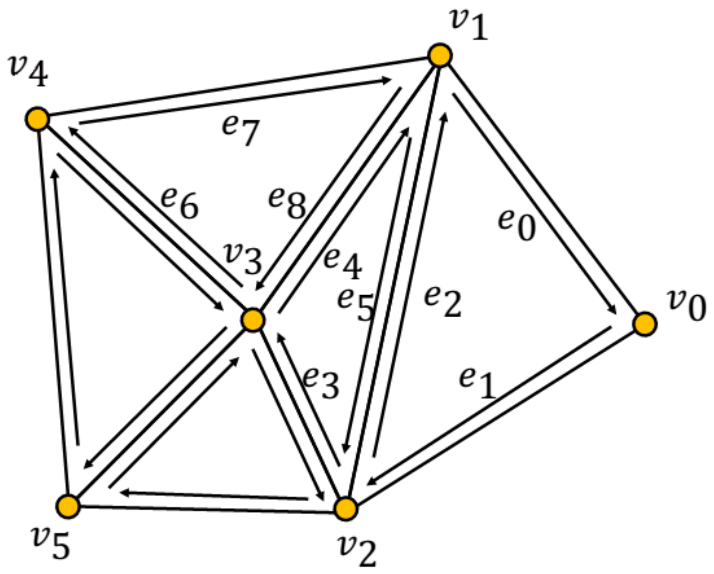
https://www.graphics.rwth-aachen.de/media/openmesh_static/Documentations/OpenMesh-6.1-Documentation/a00016.html

Polygonal Meshes

- Half-Edge Data Structure
  - Easy geometric queries → widely used in geometric computations
  - Only manifolds
  - Triangular Meshes → Directed Edge Data Structure

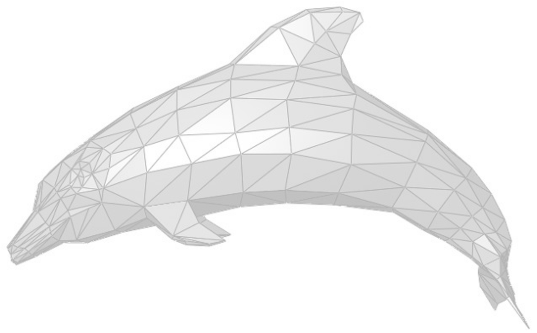
vertex list
p_0: x_0, y_0, z_0;
p_1: x_1, y_1, z_1;
p_2: x_2, y_2, z_2;
p_3: x_3, y_3, z_3;
p_4: x_4, y_4, z_4;
p_5: x_5, y_5, z_5;

edge list
$\left.\begin{array}{l} e_0\text{: 1, -1;} \\ e_1\text{: 0, -1;} \\ e_2\text{: 2, 5;} \end{array}\right\} t_0$
$\left.\begin{array}{l} e_3\text{: 2, X;} \\ e_4\text{: 3, 8;} \\ e_5\text{: 1, 2;} \end{array}\right\} t_1$
e_6: 3, X;
e_7: 4, -1;
…

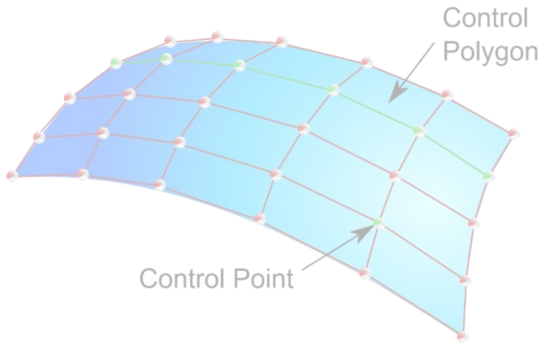


Overview

- Polygonal Meshes

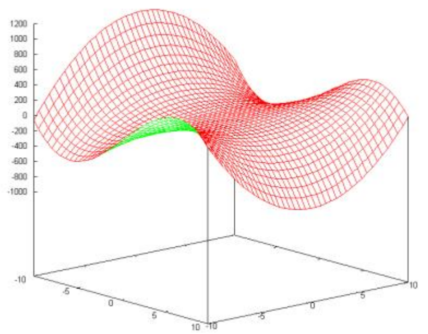


- Parametric Surfaces



- Implicit Surfaces

- Explicit Surfaces

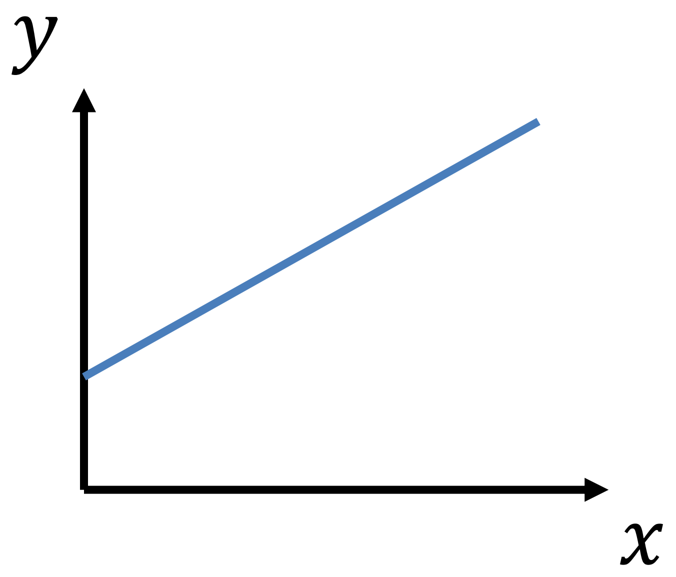


- Constructive Solid Geometry

Explicit Surfaces

- Explicit form:

$$f(x) = y = m \cdot x + c$$



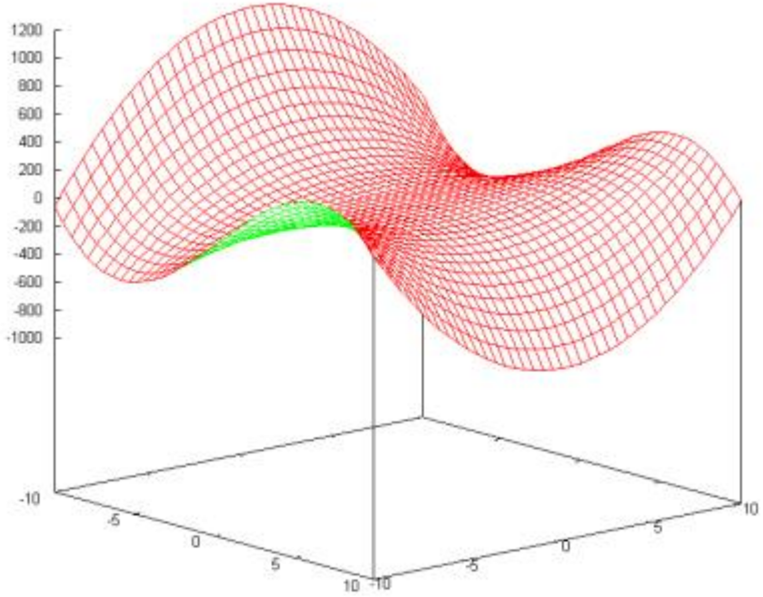
Explicit Surfaces

- Explicit form:

$$f(x, y)\colon\ \mathbb{R}^2 \to \mathbb{R}$$

- Surface is defined by:

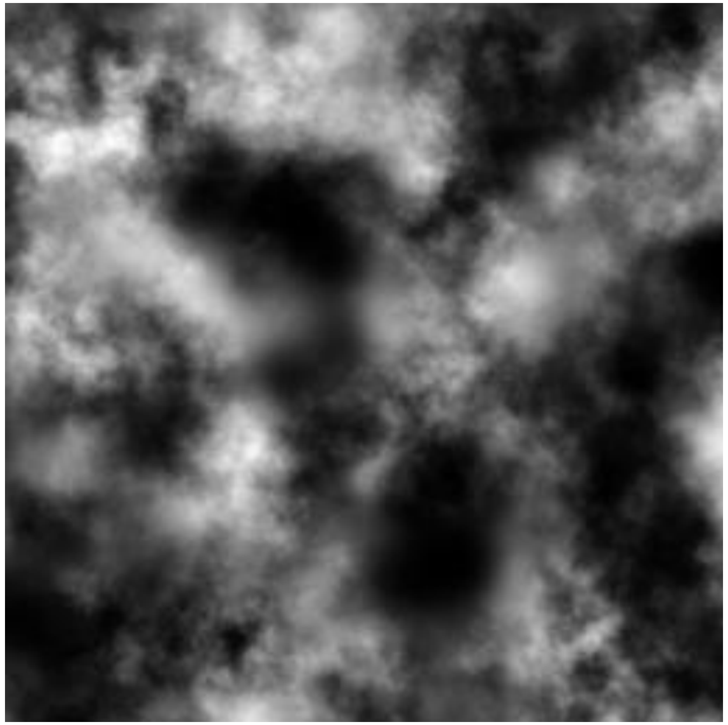
$$S(x, y) = (x, y, f(x, y))$$

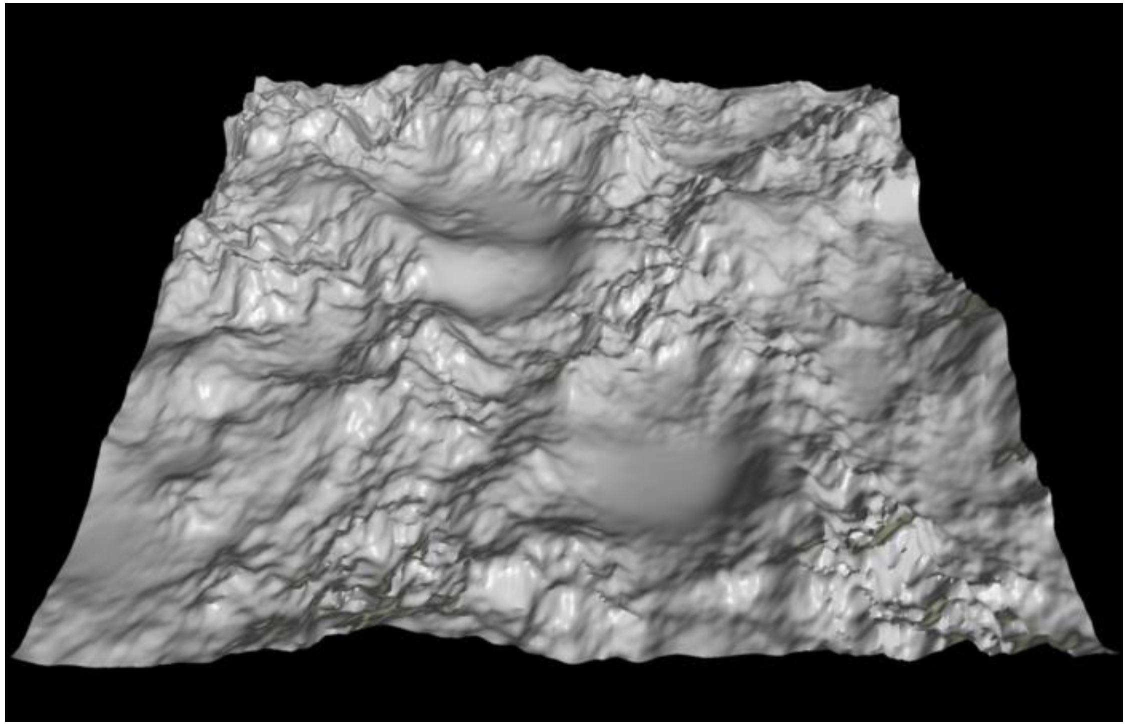


- Disadvantages
  - Restricted shapes, e.g., only one height value per (x,y) pair

Explicit Surfaces

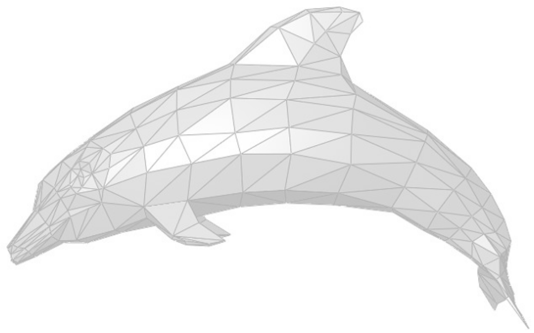
- E.g., height fields



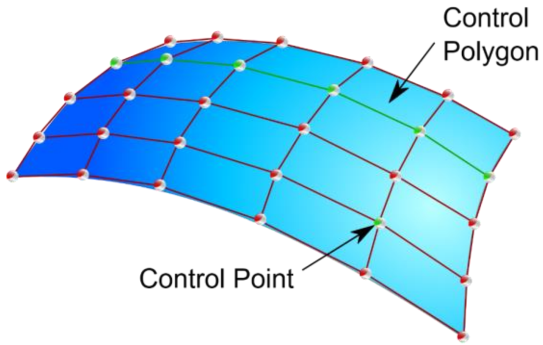


Overview

- Polygonal Meshes

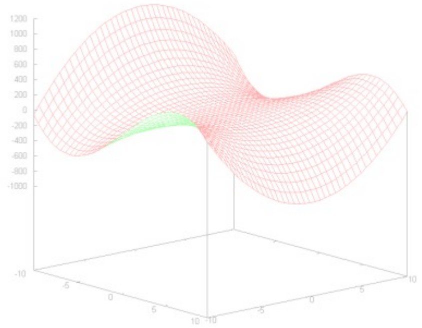


- Parametric Surfaces



- Implicit Surfaces

- Explicit Surfaces



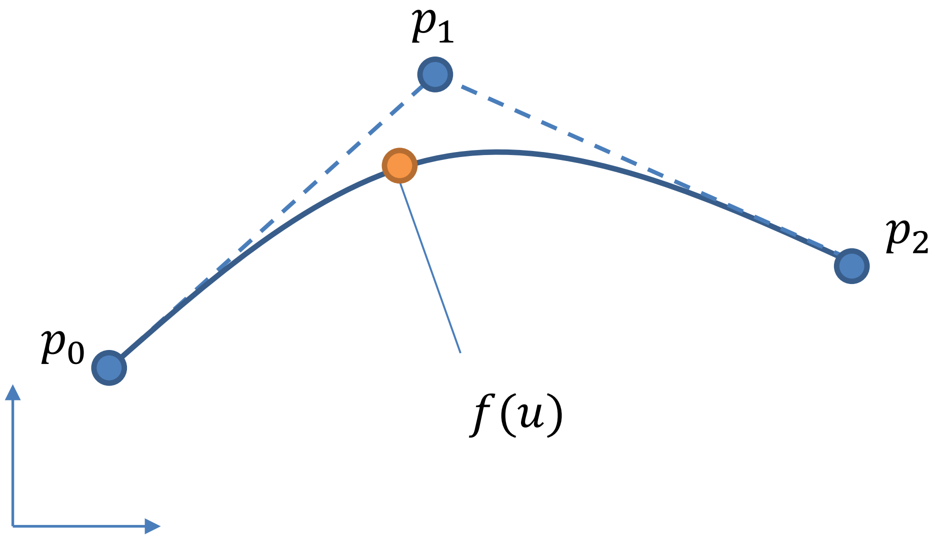
- Constructive Solid Geometry

Parametric Surfaces

– Used in design and construction → freeform surfaces

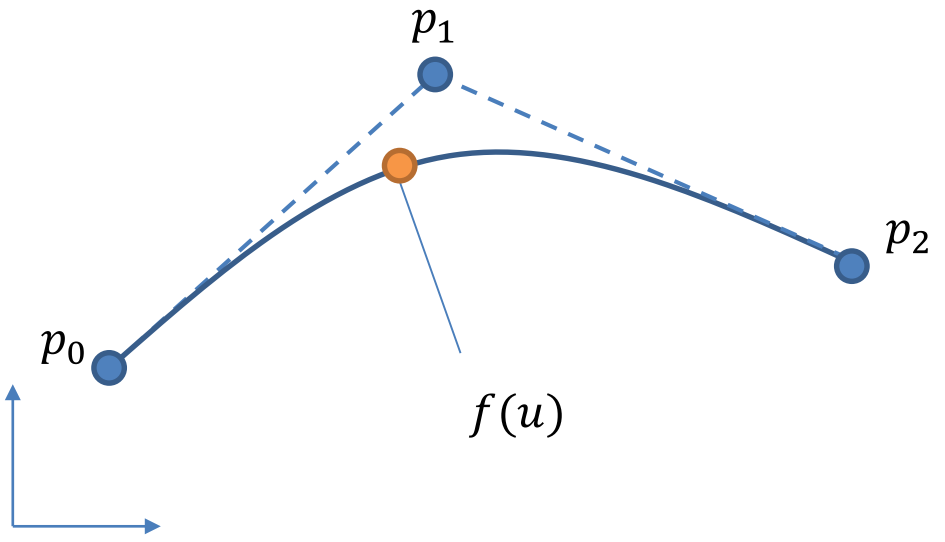
Parametric Surfaces

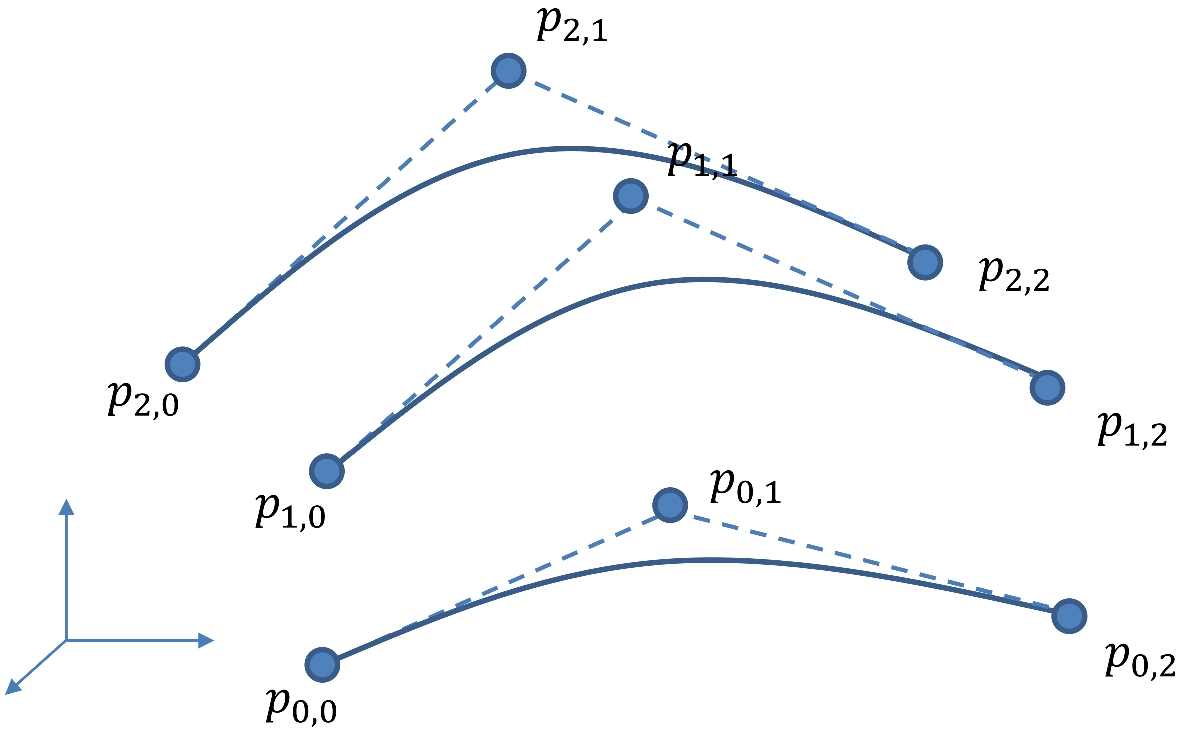
- Example: Bézier Curves & Tensor Product Surface



Parametric Surfaces

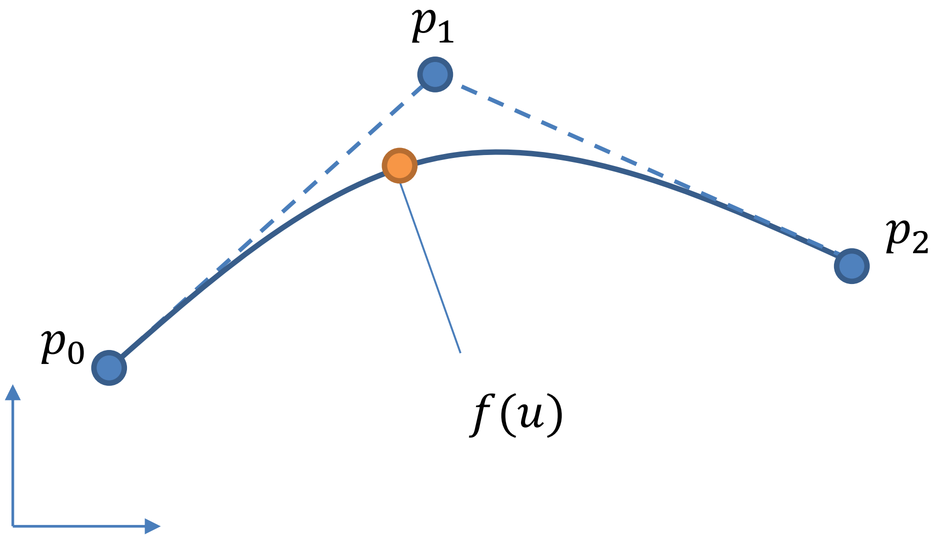
- Example: Bézier Curves & Tensor Product Surface

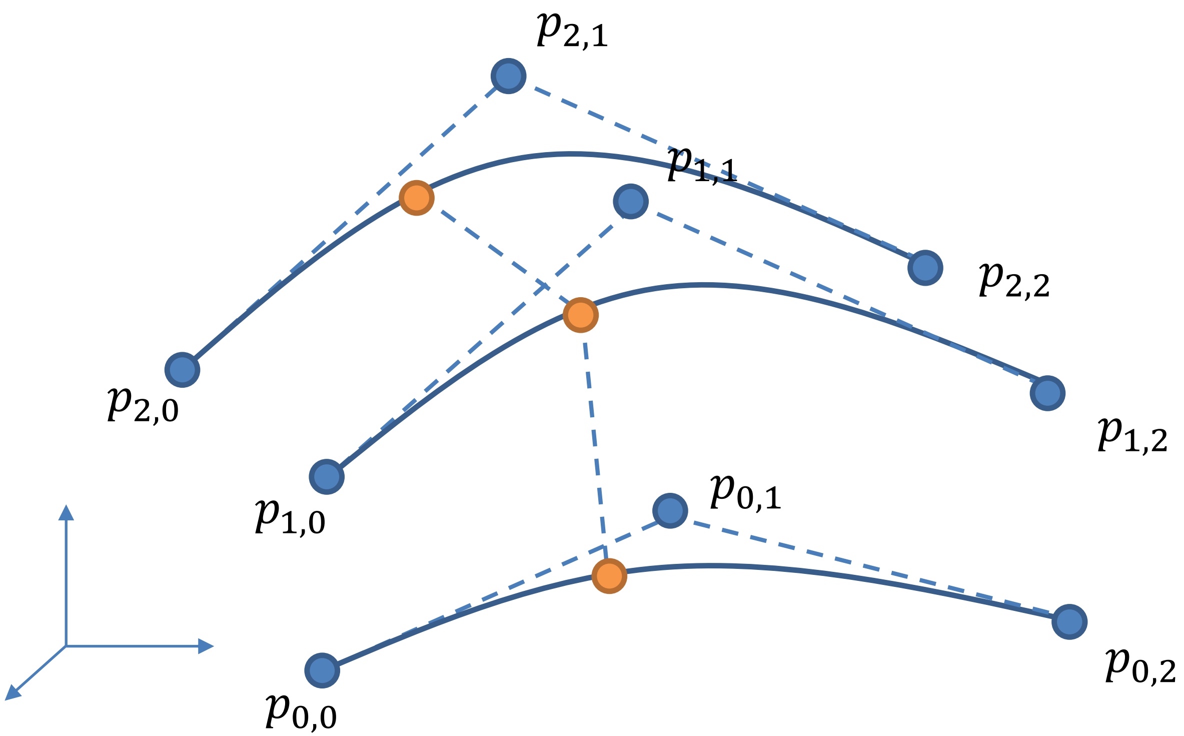




Parametric Surfaces

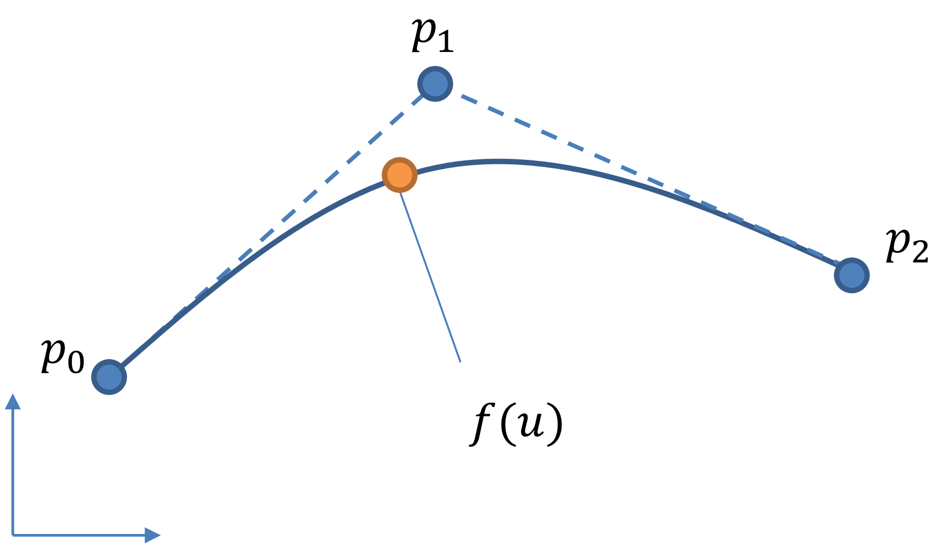
- Example: Bézier Curves & Tensor Product Surface

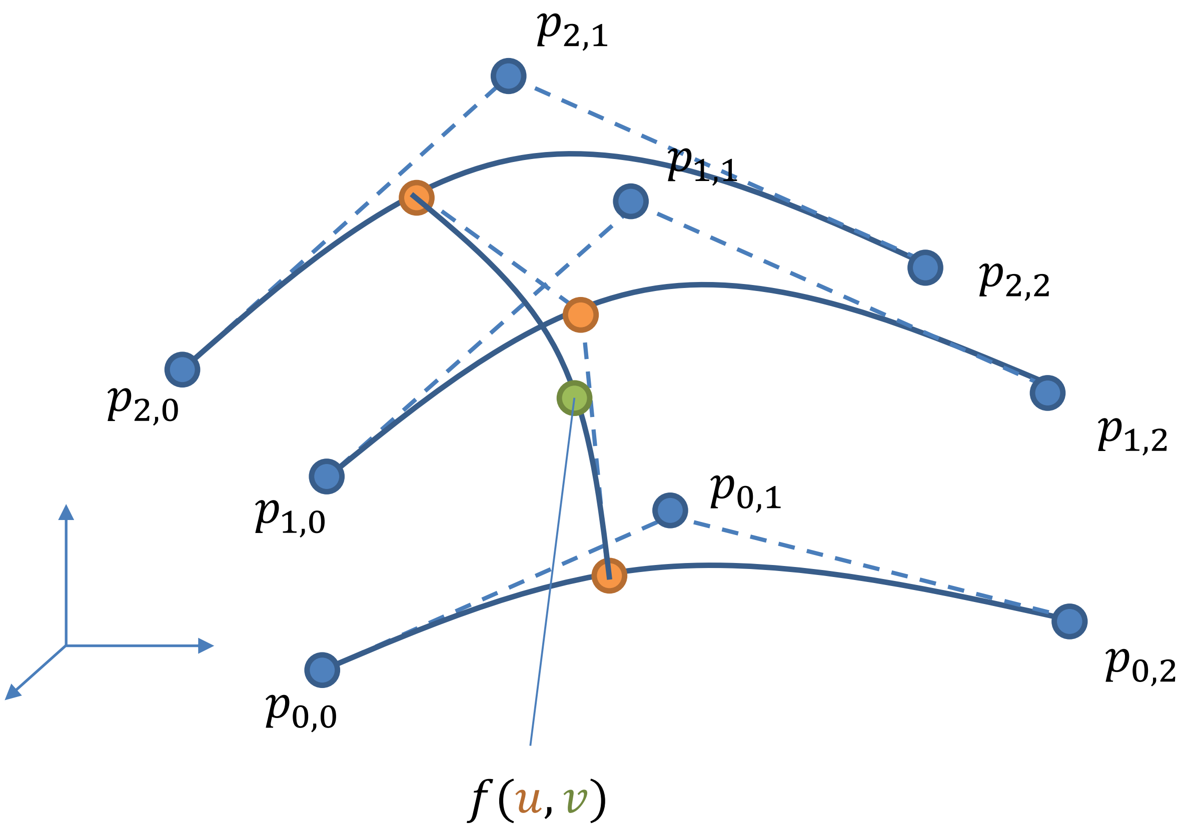




Parametric Surfaces

- Example: Bézier Curves & Tensor Product Surface

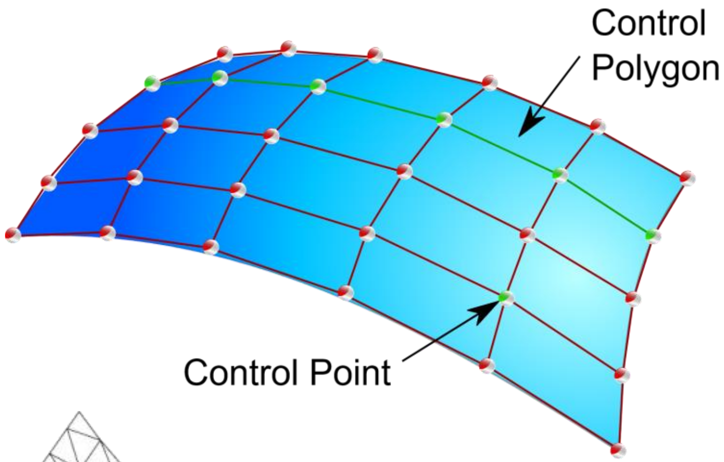




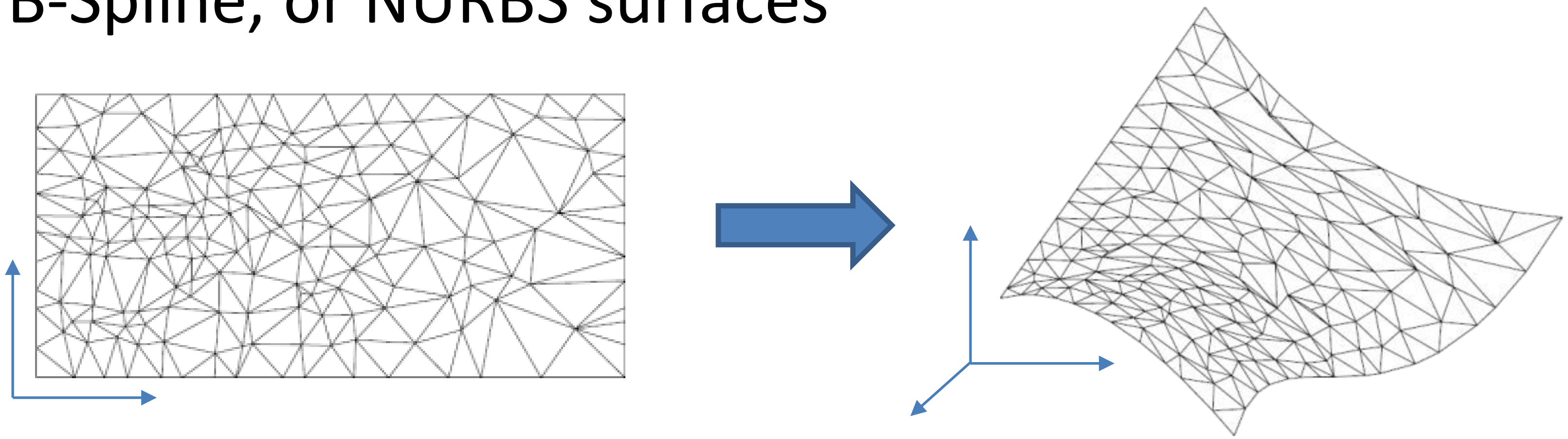
Parametric Surfaces

- Parametric form:

$$f(u, v)\colon \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



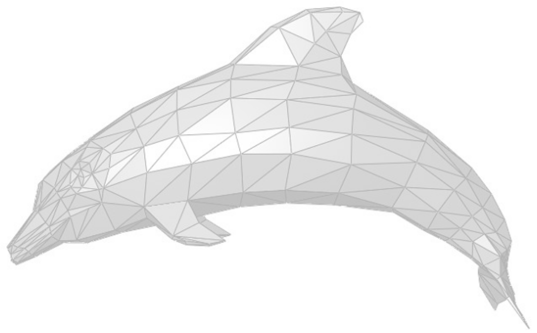
- Maps a bounded 2D domain to 3D
  – Bézier-, B-Spline, or NURBS surfaces



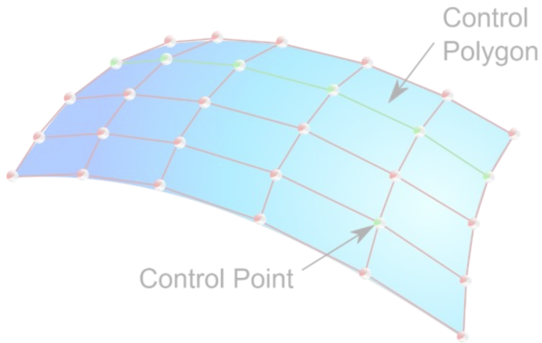
  – Used in design and construction → freeform surfaces

Overview

- Polygonal Meshes

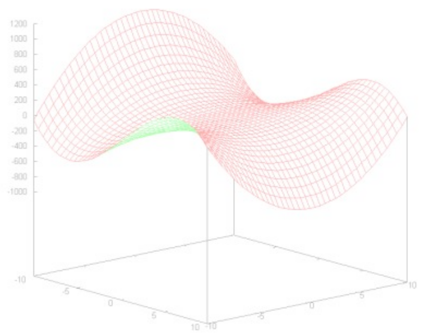


- Parametric Surfaces

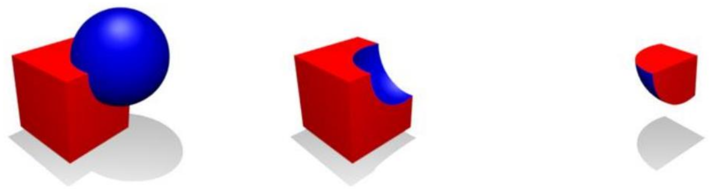


- Implicit Surfaces

- Explicit Surfaces

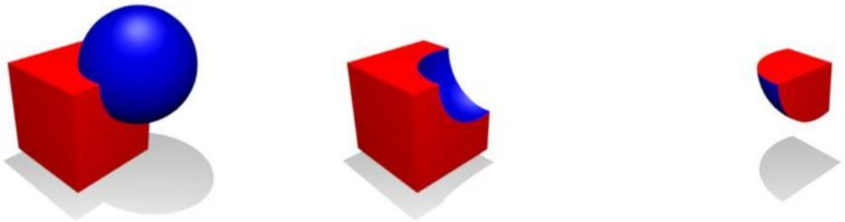


- Constructive Solid Geometry



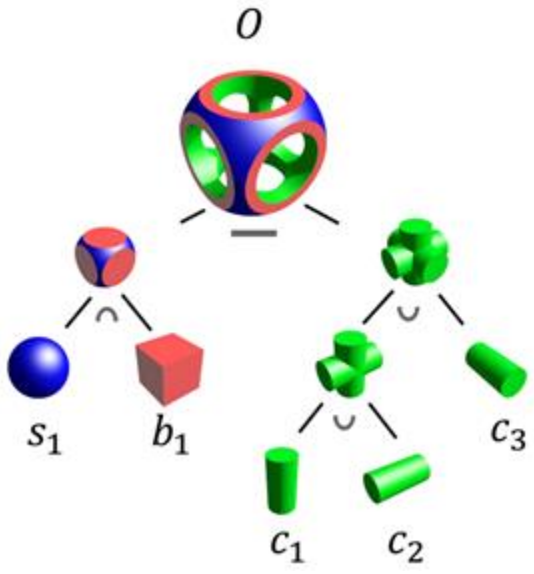
Constructive Solid Geometry

- Surface is defined as the boundary of a solid object that was created by Boolean operations on primitive solids



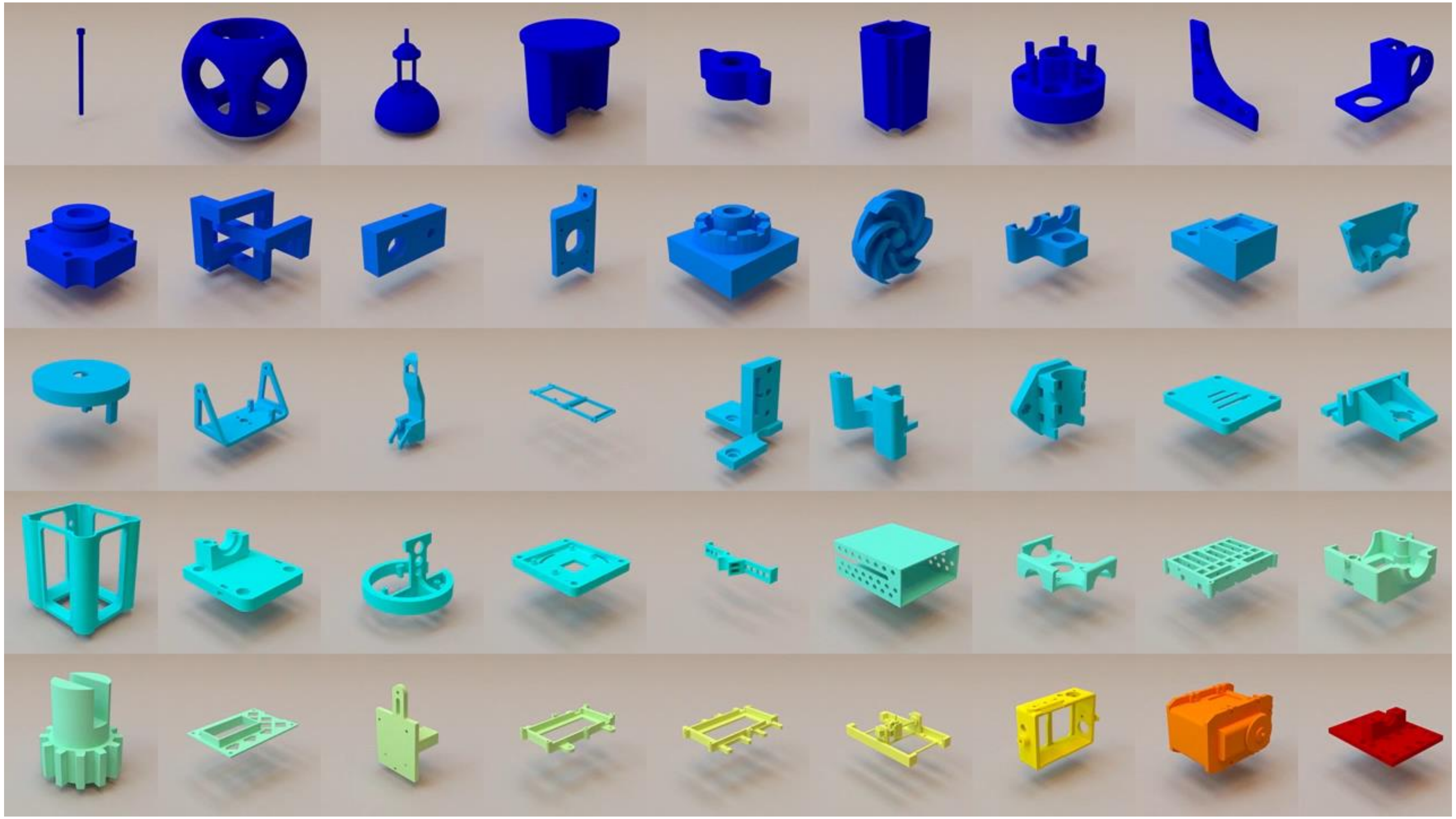
  – Example:

$$O = (s_1 \cap \mathrm{b}_1) - (\mathrm{c}_3 \cup (\mathrm{c}_1 \cup \mathrm{c}_2))$$



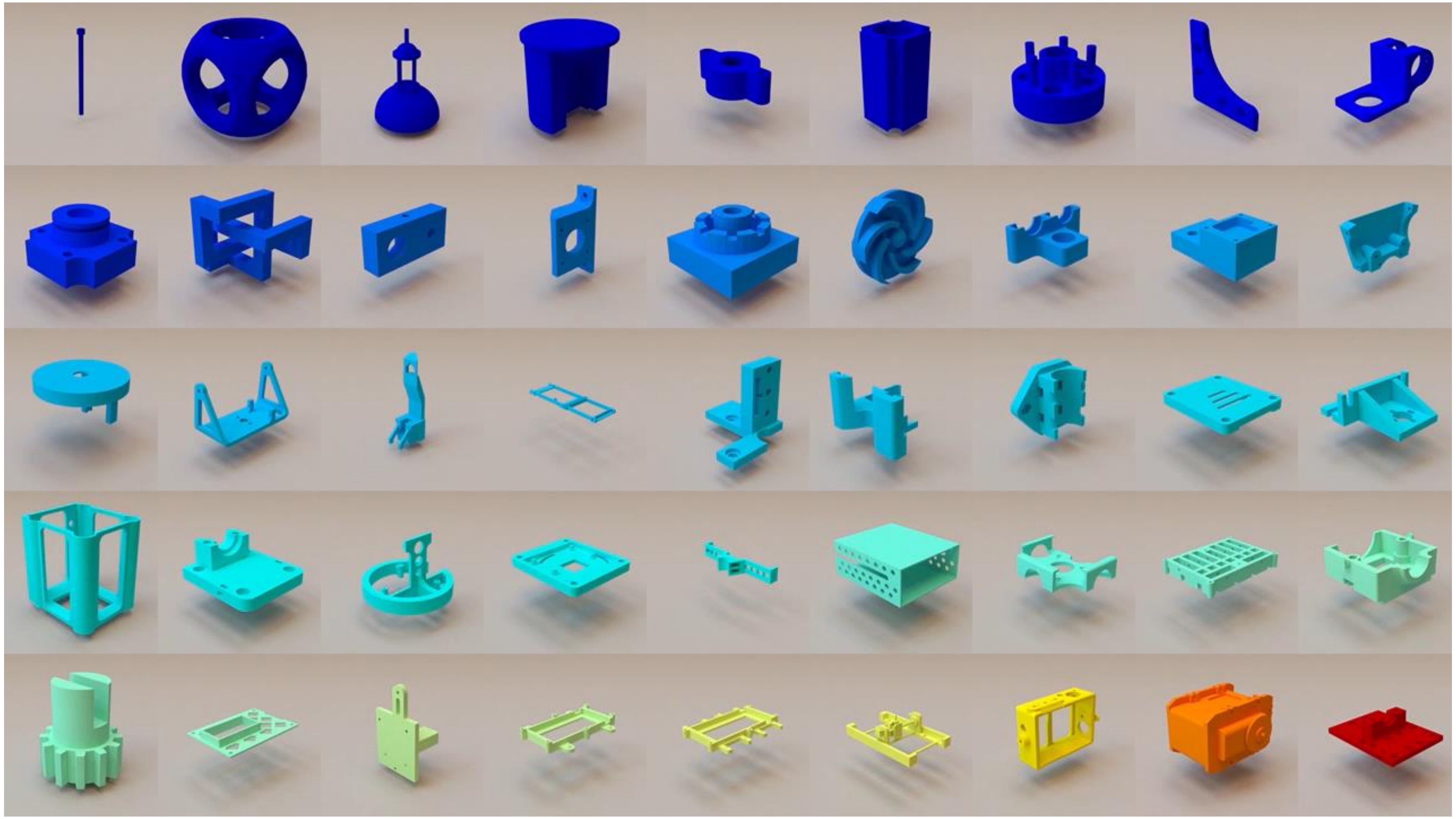
  – Used in design and construction
    - Hard to model “organic” shapes

Constructive Solid Geometry



[Du’18] T.Du et al., “InverseCSG: Automatic Conversion of 3D Models to CSG Trees“

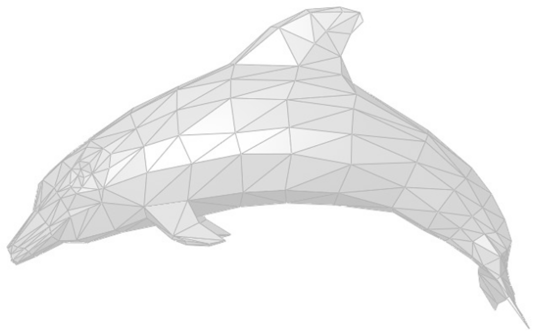
Constructive Solid Geometry



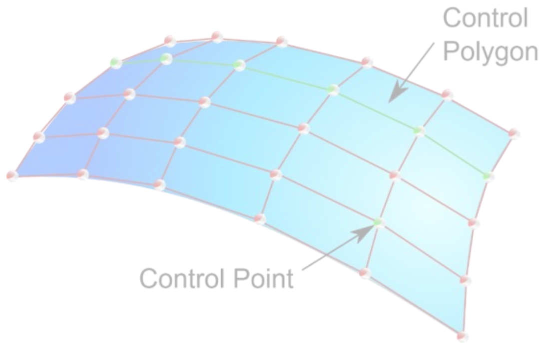
[Du'18] T.Du et al., "InverseCSG: Automatic Conversion of 3D Models to CSG Trees"

Overview

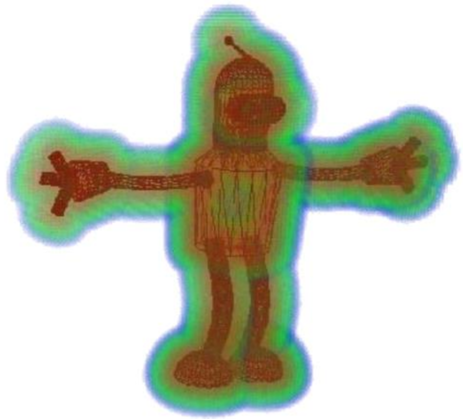
- Polygonal Meshes



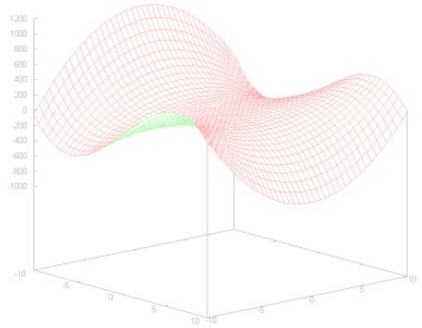
- Parametric Surfaces



- Implicit Surfaces



- Explicit Surfaces



- Constructive Solid Geometry

Implicit Surfaces

- Remember, mathematically:
  - Explicit function: $f(x) = y$
  - Implicit function: $x^2 + y^2 - 1 = 0$

Implicit Surfaces

- [Curless and Levoy], KinectFusion, VoxelHashing, …

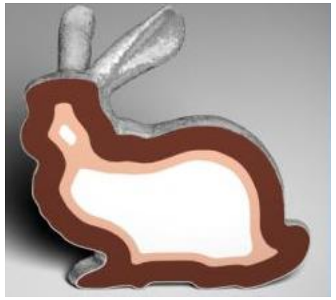
Implicit Surfaces

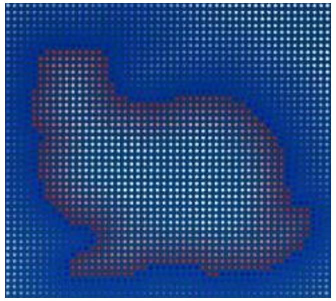
- Implicit form:

$$f(x, y, z)\colon \mathbb{R}^3 \to \mathbb{R}$$

- Surface is defined by the level set of the tri-variate scalar function

$$f(x, y, z) = c$$





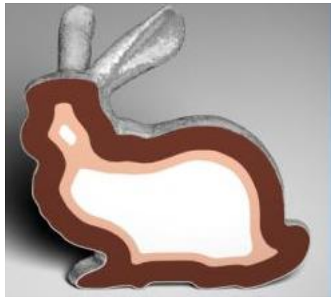
Implicit Surfaces

- Implicit form:

$$f(x, y, z): \mathbb{R}^3 \rightarrow \mathbb{R}$$

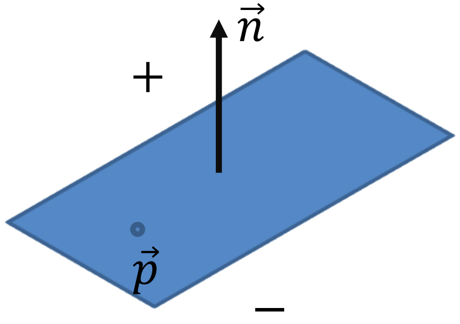
- Surface is defined by the level set of the tri-variate scalar function

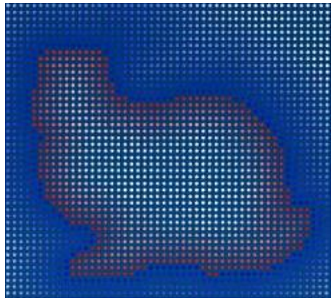
$$f(x, y, z) = c$$



- Example: Hesse normal form

$$f(x, y, z) = \left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \vec{p} \right) \cdot \vec{n} = 0$$





Implicit Surfaces

- Implicit form:

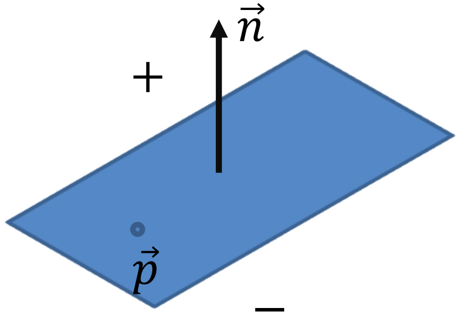
$$f(x, y, z): \mathbb{R}^3 \rightarrow \mathbb{R}$$

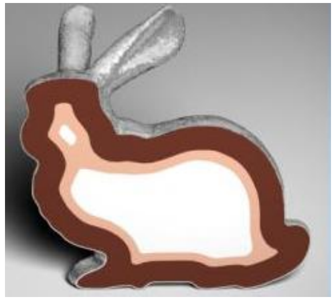
- Surface is defined by the level set of the tri-variate scalar function

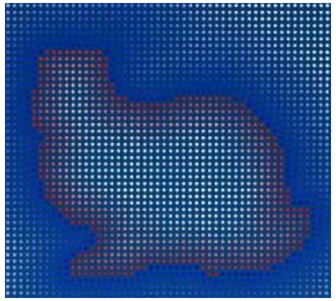
$$f(x, y, z) = c$$

- Example: Hesse normal form

$$f(x, y, z) = \left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \vec{p} \right) \cdot \vec{n} = 0$$



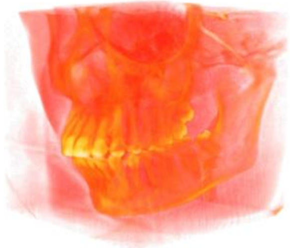


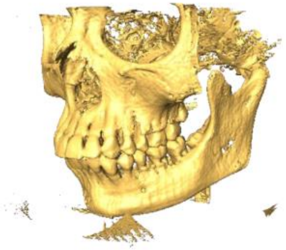


f is also called **Signed Distance Function** (SDF)

Implicit Surfaces

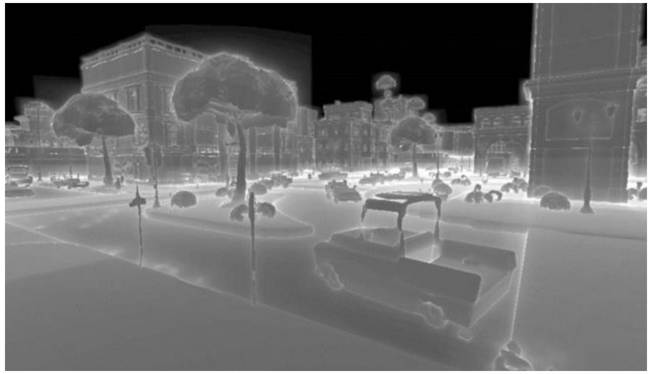
- Examples of such scalar functions $f(x, y, z)$
  - Density



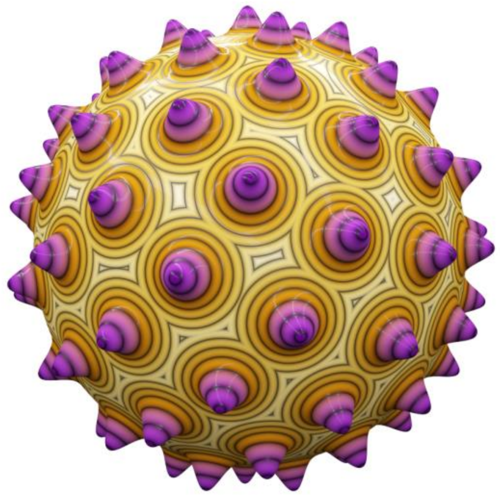


  - Heat

  - Distance

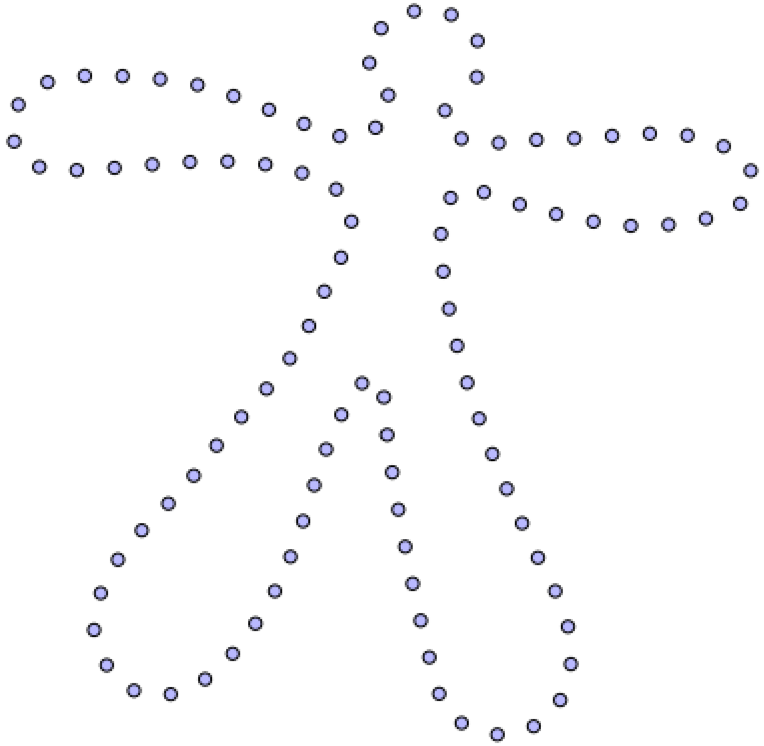


Used for soft shadows in game engines (e.g., Unreal Engine)



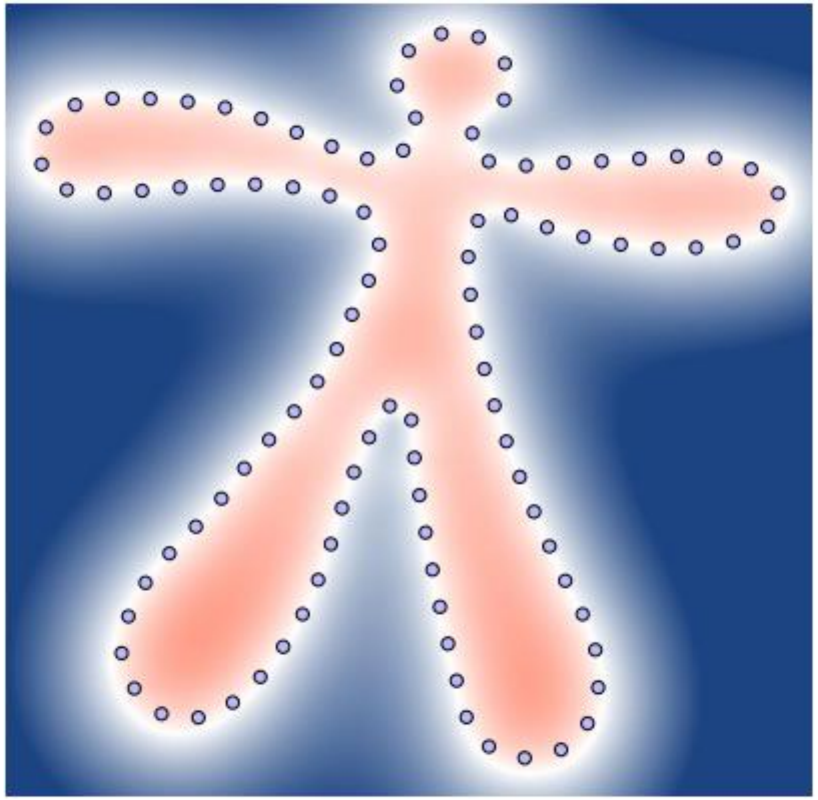
Implicit Surfaces

- How to find such a scalar function $f(x, y, z)$?
  - Given sample points



Implicit Surfaces

- How to find such a scalar function $f(x, y, z)$?
  - Given sample points
  - Find a scalar function that fulfils:
    - $f = 0$ at the sample points
    - $f < 0$ inside the object
    - $f > 0$ outside the object

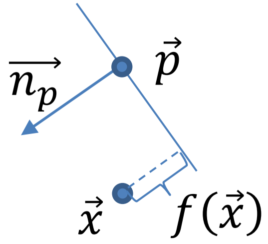


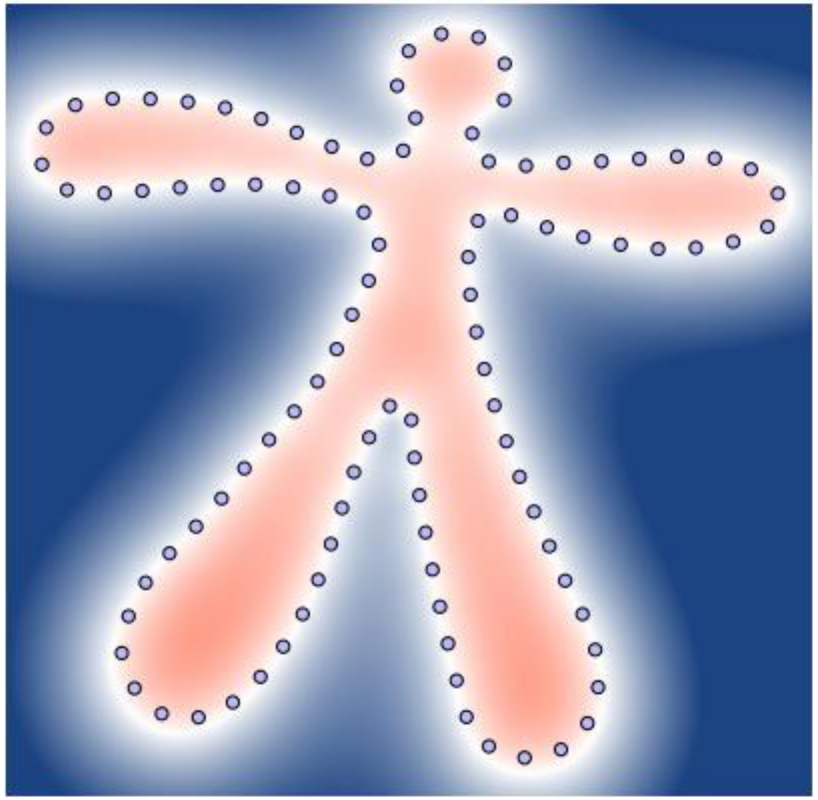
Implicit Surfaces

- How to find such a scalar function $f(x, y, z)$?
  - Hoppe'92:

$$f(\vec{x}) = (\vec{x} - \vec{p}) \cdot \overrightarrow{n_p}$$

$\vec{p}$ is the closest point to $\vec{x}$





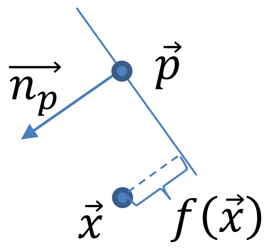
[Hoppe'92] H.Hoppe et al., "Surface reconstruction from unorganized points" (SIGGRAPH 92)

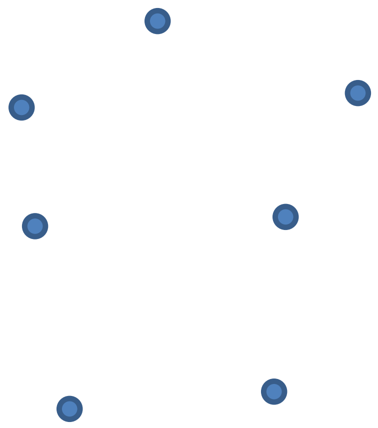
Implicit Surfaces

- How to find such a scalar function $f(x, y, z)$?
  - Hoppe'92:

$$f(\vec{x}) = (\vec{x} - \vec{p}) \cdot \overrightarrow{n_p}$$

$\vec{p}$ is the closest point to $\vec{x}$





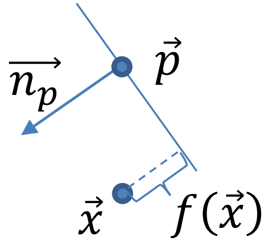
[Hoppe'92] H.Hoppe et al., "Surface reconstruction from unorganized points" (SIGGRAPH 92)

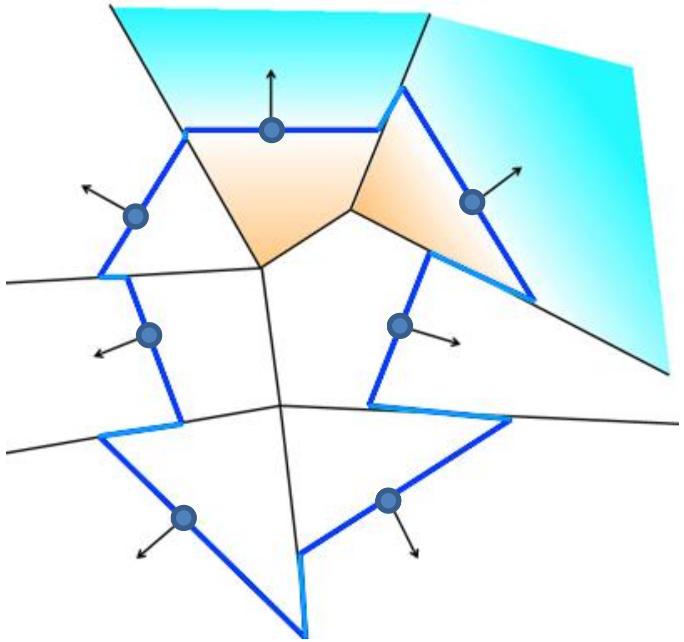
Implicit Surfaces

- How to find such a scalar function $f(x, y, z)$?
  - Hoppe'92:

$$f(\vec{x}) = (\vec{x} - \vec{p}) \cdot \overrightarrow{n_p}$$

$\vec{p}$ is the closest point to $\vec{x}$



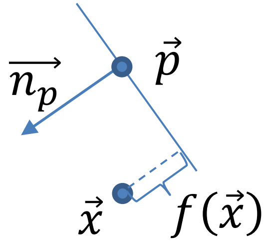


[Hoppe'92] H.Hoppe et al., "Surface reconstruction from unorganized points" (SIGGRAPH 92)

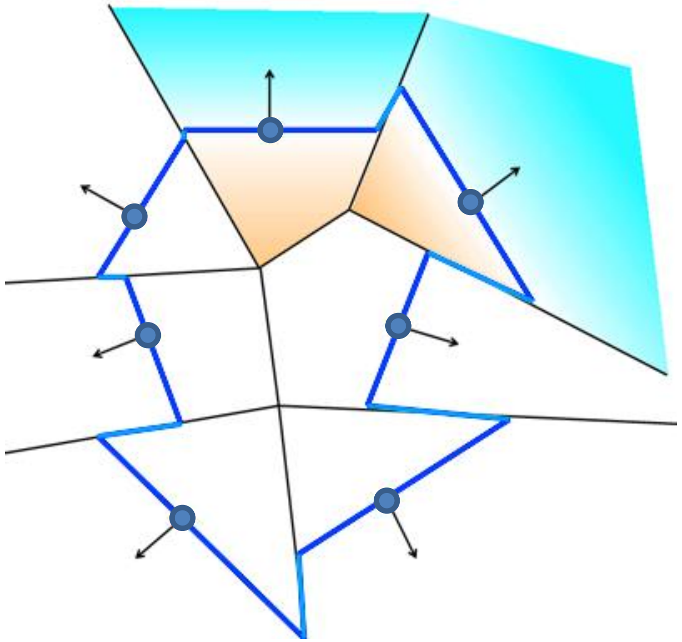
Implicit Surfaces

- How to find such a scalar function $f(x, y, z)$?
  - Hoppe'92:

$$f(\vec{x}) = (\vec{x} - \vec{p}) \cdot \overrightarrow{n_p}$$



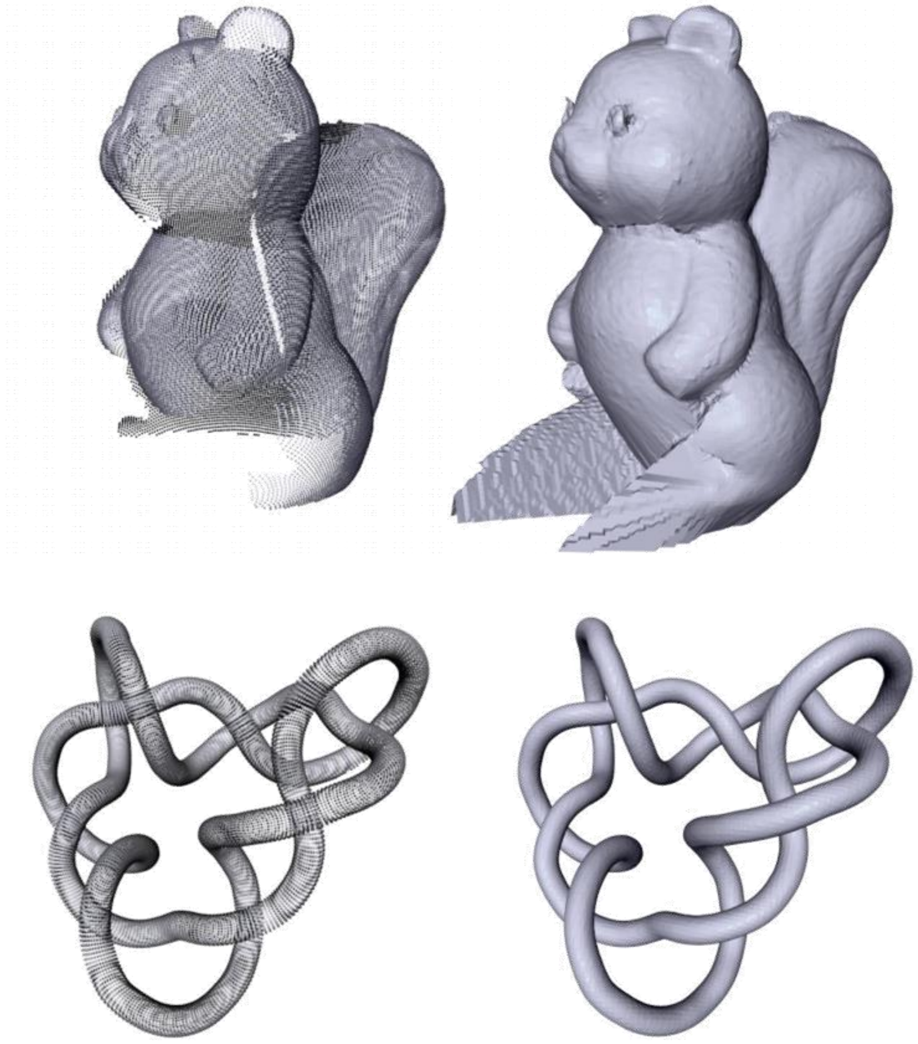
$\vec{p}$ is the closest point to $\vec{x}$



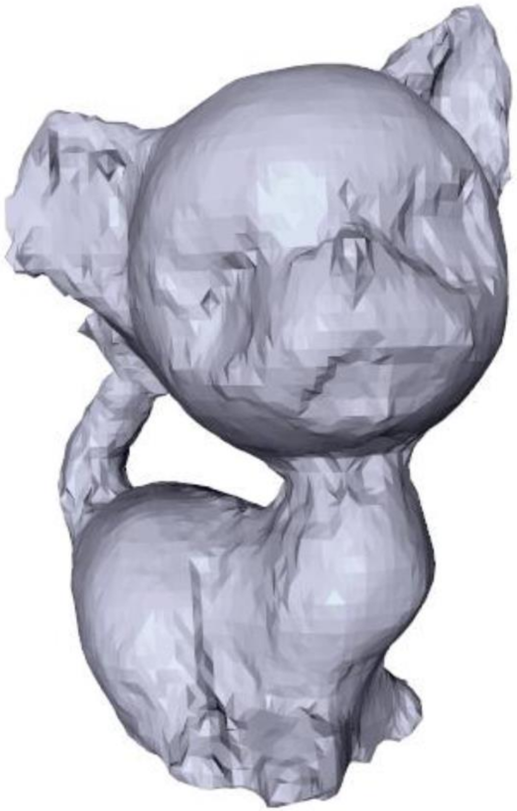
- Piecewise linear, defined on the Voronoi diagram of the input
- Discontinuous along Voronoi edges
- Dependent on the input density

[Hoppe'92] H.Hoppe et al., "Surface reconstruction from unorganized points" (SIGGRAPH 92)

Implicit Surfaces

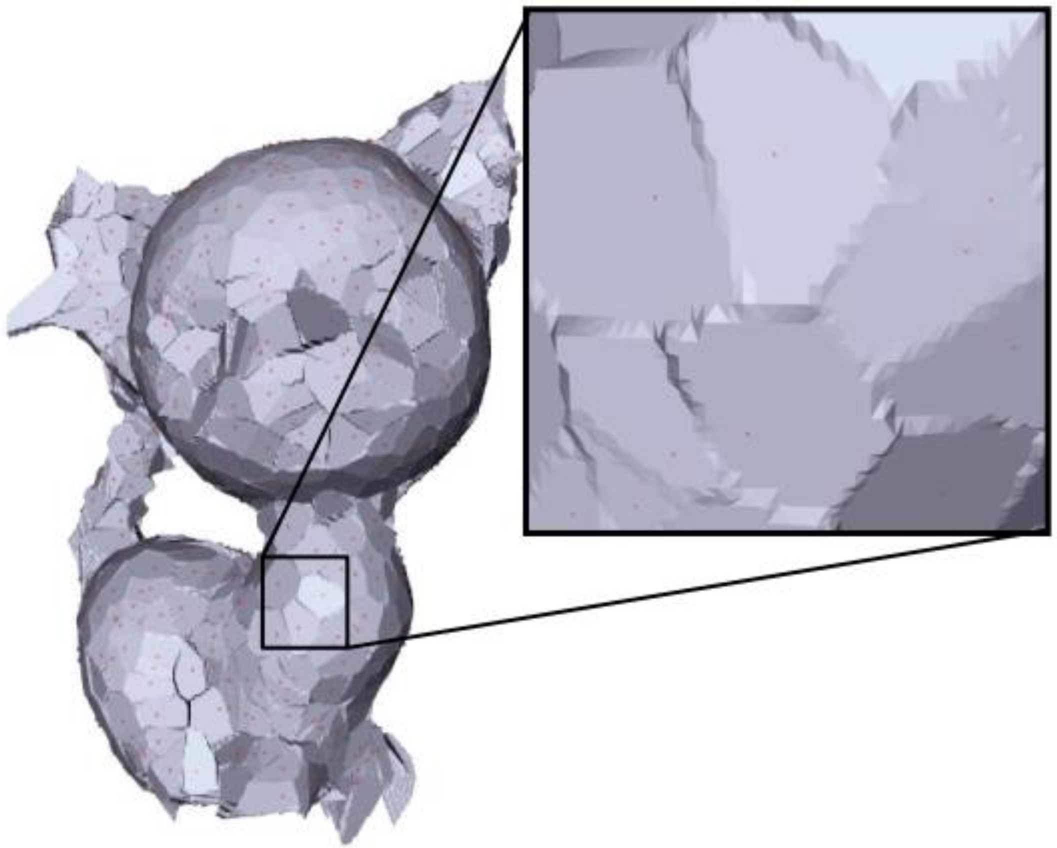


Implicit Surfaces



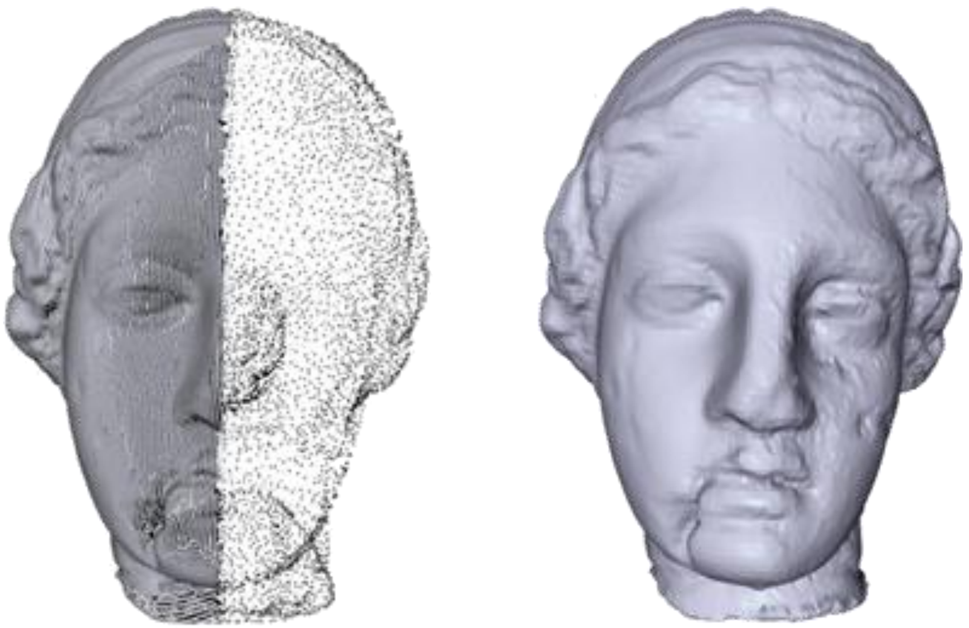
Piecewise linear surface approximation.

Implicit Surfaces



Piecewise linear surface approximation.

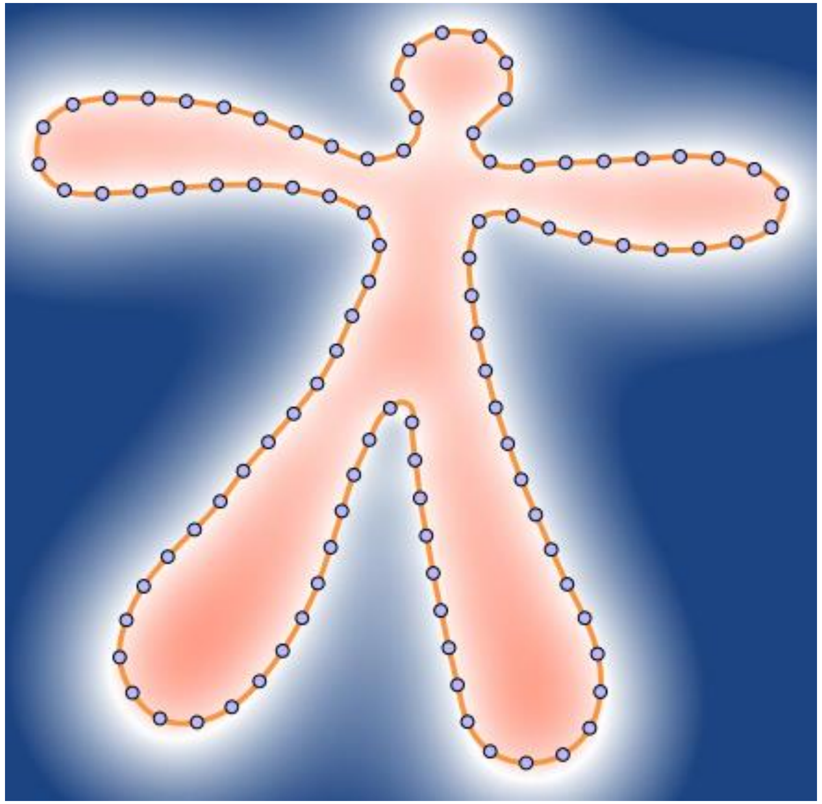
Implicit Surfaces



Hoppe'92 is dependent on the input density

Implicit Surfaces

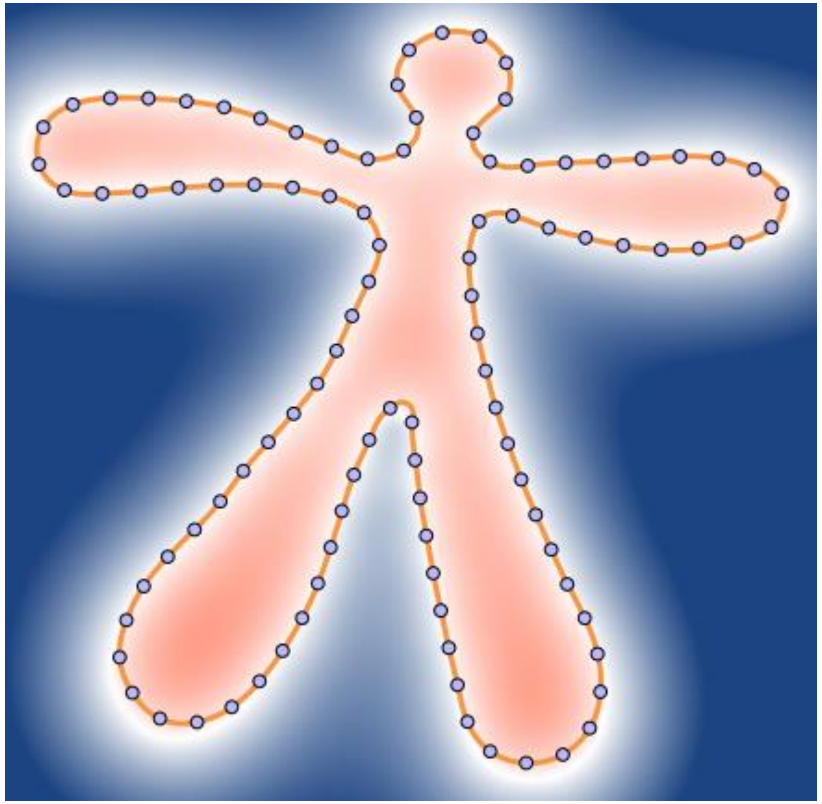
- How to find such a scalar function $f(x, y, z)$?
  - Given sample points
  - Find a scalar function that fulfils:
    - $f = 0$ at the sample points
    - $f < 0$ inside the object
    - $f > 0$ outside the object
    - $\boldsymbol{f}$ **should be smooth**



Implicit Surfaces

- How to find such a scalar function $f(x, y, z)$?
  - Given sample points
  - Find a scalar function that fulfils:
    - $f = 0$ at the sample points
    - $f < 0$ inside the object
    - $f > 0$ outside the object
    - $\boldsymbol{f}$ **should be smooth**

→ **Radial Basis Functions (RBF)**

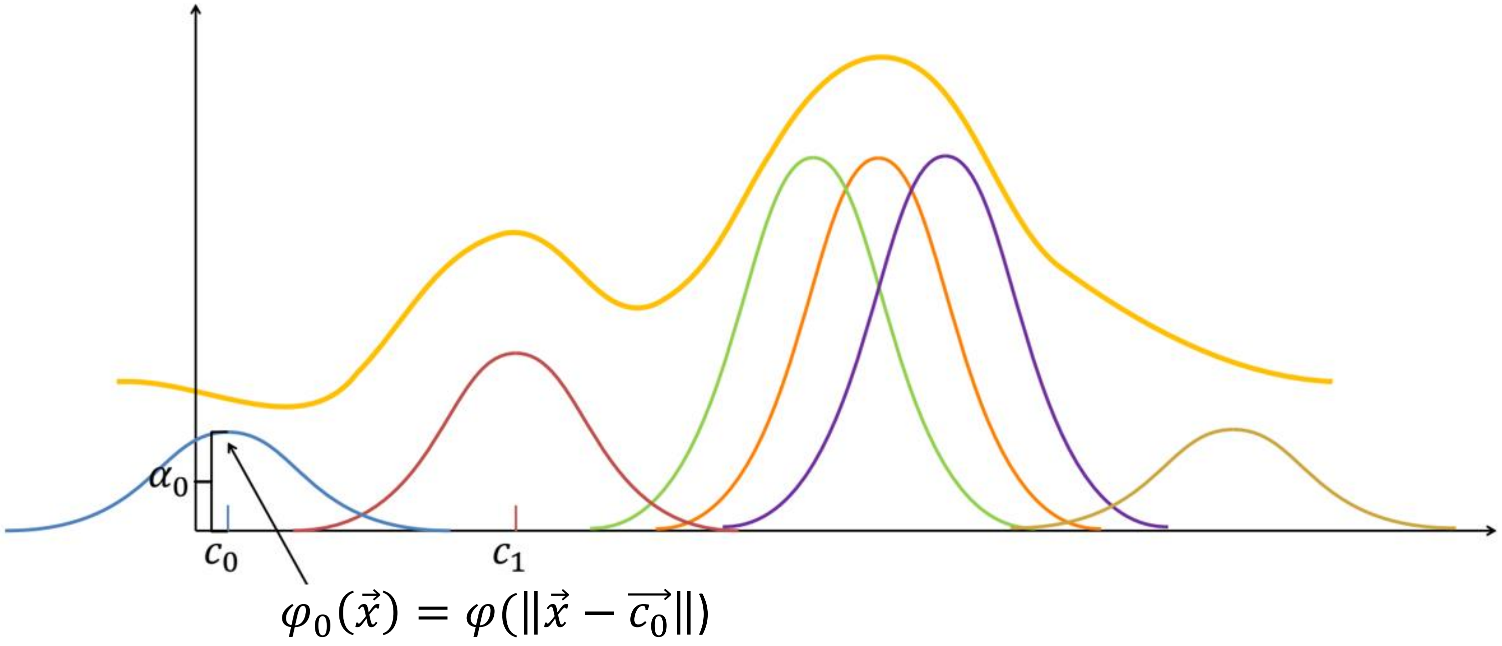


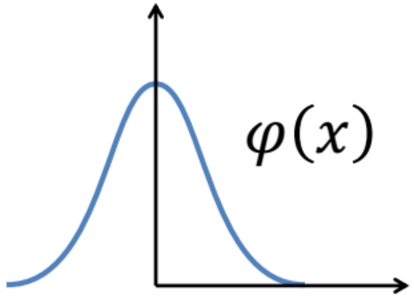
[Turk'99] G.Turk et al., "Variational Implicit Surfaces"

Implicit Surfaces – Radial Basis Functions

- Idea: each complex function can be approximated as the sum of simple scaled and translated kernel functions $\varphi(x)$:

$$g(\vec{x}) \approx f(\vec{x}) = \sum_i \alpha_i \cdot \varphi_i(\vec{x}) = \sum_i \alpha_i \cdot \varphi(\|\vec{x} - \vec{c_i}\|)$$





Implicit Surfaces – Radial Basis Functions

- A **Radial Basis Function** (RBF), also called a **Radial Basis Function Network** is defined as sum of translated and scaled kernels and a linear polynomial:

$$f(\vec{x}) = \sum_i \alpha_i \cdot \varphi_i(\vec{x}) + \underbrace{\vec{b} \cdot \vec{x} + d}_{\text{linear term}}$$

  - Allow for functions of arbitrary complexity!
    - complexity increases with the number of used kernel functions
  - If basis functions $\varphi_i(x)$ are smooth, $f(\vec{x})$ is smooth as well.

Implicit Surfaces – Radial Basis Functions

- Radial Basis Functions in 3D:

$$f(\vec{x}) = \sum_i \alpha_i \cdot \varphi_i(\vec{x}) + \vec{b} \cdot \vec{x} + d$$

  – Use the biharmonic radial basis function:

$$\varphi_i(\vec{x}) = \|\vec{p_i} - \vec{x}\|^3$$

  – Where $\vec{p_i}$ are the input sample points

$$f(\vec{x}) = \sum_i \alpha_i \cdot \|\vec{p_i} - \vec{x}\|^3 + \vec{b} \cdot \vec{x} + d$$

[Turk'99] G.Turk et al., "Variational Implicit Surfaces"

Implicit Surfaces – Radial Basis Functions

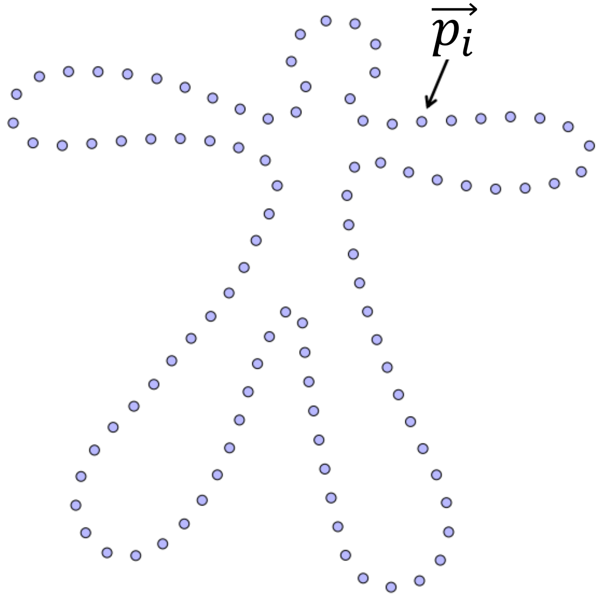
- Radial Basis Functions in 3D:

$$f(\vec{x}) = \sum_i \alpha_i \cdot \varphi_i(\vec{x}) + \vec{b} \cdot \vec{x} + d$$

  – Use the biharmonic radial basis function:

$$\varphi_i(\vec{x}) = \|\vec{p_i} - \vec{x}\|^3$$

  – Where $\vec{p_i}$ are the input sample points



$$f(\vec{x}) = \sum_i \boldsymbol{\alpha_i} \cdot \|\vec{p_i} - \vec{x}\|^3 + \vec{\mathbf{b}} \cdot \vec{x} + \boldsymbol{d}$$

unknowns

Implicit Surfaces – Radial Basis Functions

$$f(\vec{x}) = \sum_i \underbrace{\boldsymbol{\alpha_i}}_{} \cdot \|\vec{p_i} - \vec{x}\|^3 + \vec{\mathbf{b}} \cdot \vec{x} + \boldsymbol{d}$$

unknowns

- $f(\vec{x})$ should be zero at the sample points:
  - $f(\vec{p_i}) = 0 \ \rightarrow \ n$ equations   (n: number of sample points)

→Solve system of linear equations: $A\vec{x} = \vec{b}$

- Note: system is **underdetermined** since we have $n + 4$ unknowns
- Solving the system will result in the trivial result ($\vec{x} = \vec{0}$)

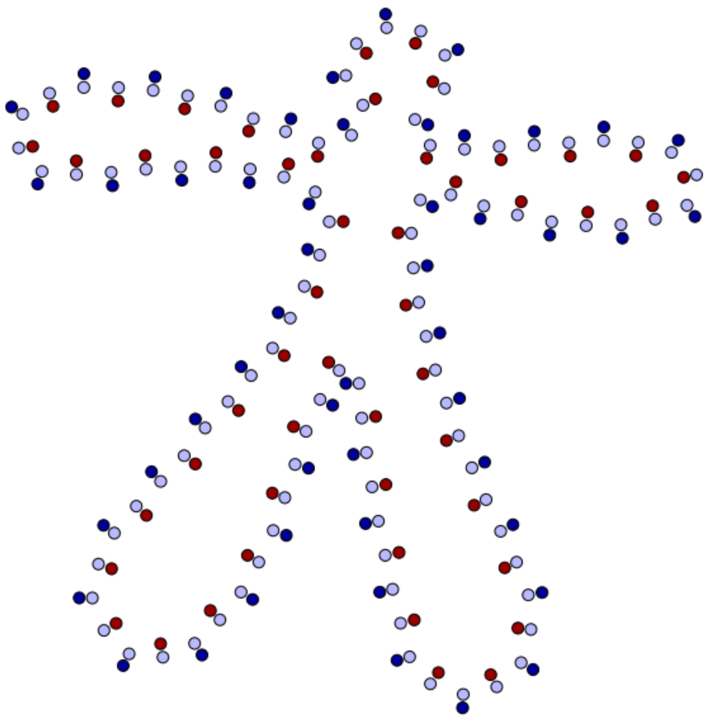
Implicit Surfaces – Radial Basis Functions

$$f(\vec{x}) = \sum_i \boldsymbol{\alpha_i} \cdot \|\vec{p_i} - \vec{x}\|^3 + \vec{\mathbf{b}} \cdot \vec{x} + \boldsymbol{d}$$

unknowns

- $f(\vec{x})$ should be zero at the sample points:
  - $f(\vec{p_i}) = 0 \;\rightarrow\; n$ equations
- Additional constraints where f is non-zero
  - Normal constraints:

    For each sample point add off-surface points by moving the points a little in $\pm$normal direction.
    Set the target distance value of these points to $\pm\epsilon$.



Implicit Surfaces – Radial Basis Functions

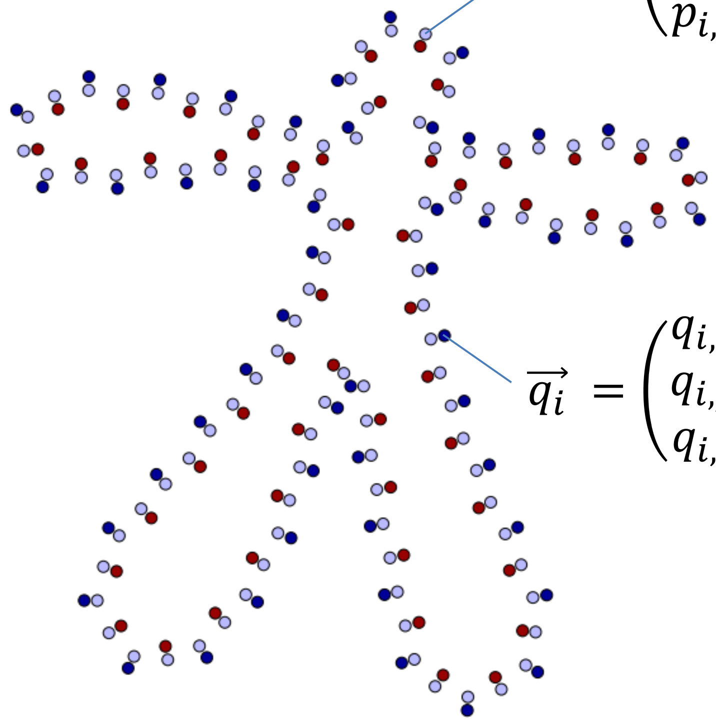
$$f(\vec{x}) = \sum_i \boldsymbol{\alpha_i} \cdot \|\vec{p_i} - \vec{x}\|^3 + \vec{\mathbf{b}} \cdot \vec{x} + \boldsymbol{d}$$

$$\begin{array}{c}
\text{on surface points} \\ \\ \\ \text{off surface points}
\end{array}
\underbrace{\begin{bmatrix}
\varphi_{1,1} & \cdots & \varphi_{1,\mathrm{n}} & p_{1,x} & p_{1,y} & p_{1,z} & 1 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\varphi_{\mathrm{n},1} & \cdots & \varphi_{\mathrm{n},\mathrm{n}} & p_{n,x} & p_{n,y} & p_{n,z} & 1 \\
\hat{\varphi}_{1,1} & \cdots & \hat{\varphi}_{1,\mathrm{n}} & q_{1,x} & q_{1,y} & q_{1,z} & 1 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\hat{\varphi}_{\mathrm{n},1} & \cdots & \hat{\varphi}_{\mathrm{n},\mathrm{n}} & q_{n,x} & q_{n,y} & q_{n,z} & 1
\end{bmatrix}}_{A}
\cdot
\underbrace{\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ b_1 \\ b_2 \\ b_3 \\ d \end{bmatrix}}_{\vec{x}}
=
\underbrace{\begin{bmatrix} 0 \\ \vdots \\ 0 \\ \pm\varepsilon \\ \vdots \\ \pm\varepsilon \end{bmatrix}}_{\vec{b}}$$

$$\varphi_{i,j} = \|\vec{p_i} - \vec{p_j}\|^3$$

$$\hat{\varphi}_{i,j} = \|\vec{q_i} - \vec{p_j}\|^3$$

$$\vec{p_i} = \begin{pmatrix} p_{i,x} \\ p_{i,y} \\ p_{i,z} \end{pmatrix}$$

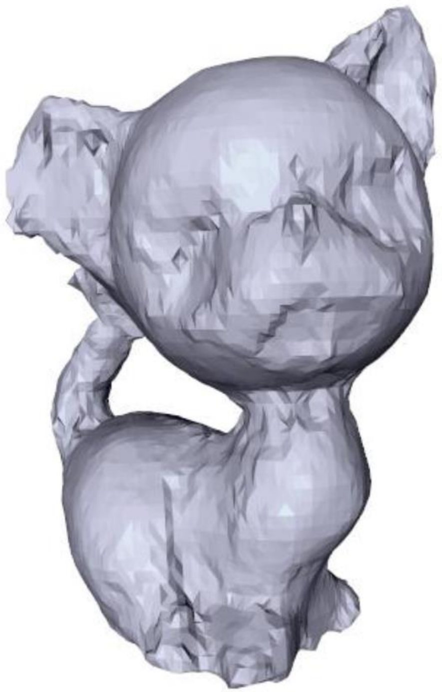


$$\vec{q_i} = \begin{pmatrix} q_{i,x} \\ q_{i,y} \\ q_{i,z} \end{pmatrix}$$

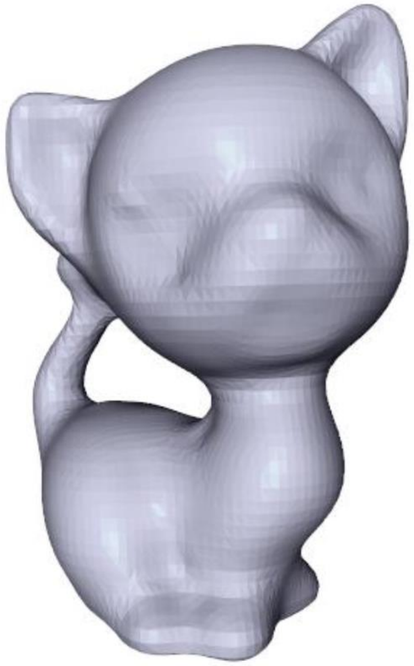
Note: System is **overdetermined**! → use least squares solution ($A^T A \cdot \vec{x} = A^T \vec{b}$)

Implicit Surfaces – Radial Basis Functions

Input Points

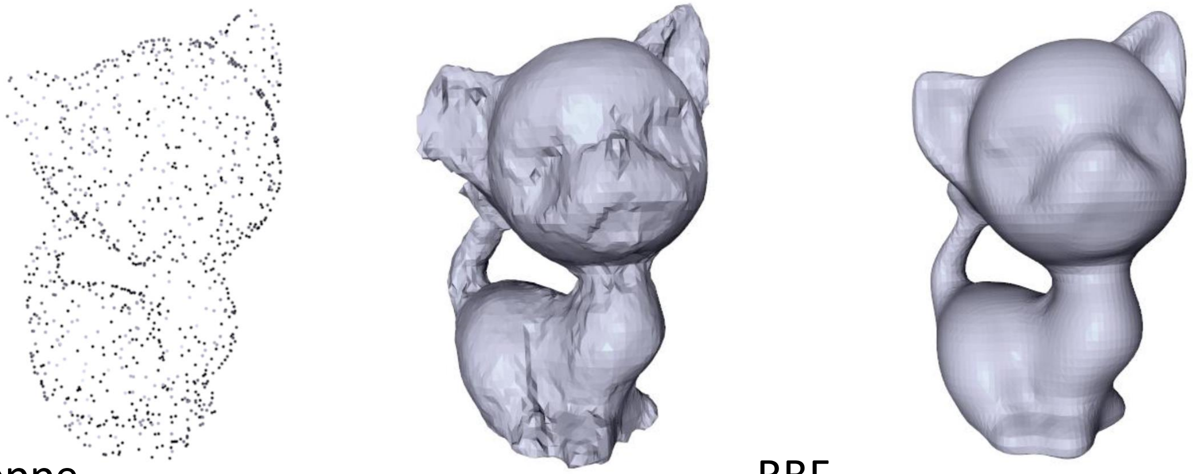


Hoppe



RBF

Implicit Surfaces – Radial Basis Functions



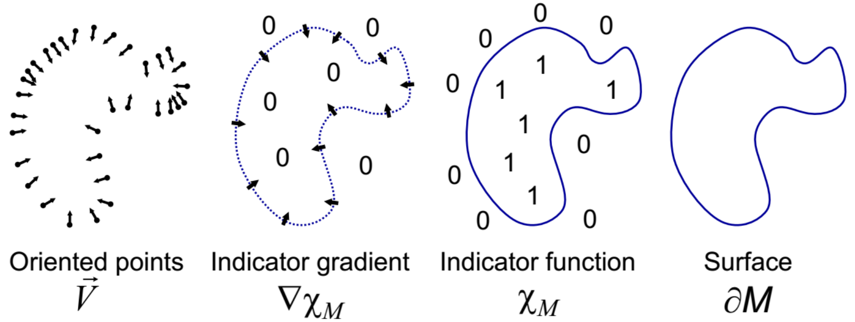
Hoppe
- Local method
- Fast and easy to implement
- Cannot handle noise, outliers, large holes

RBF
- Global method
- Requires solving a linear system (slow)
- Can only handle small point sets
- Can handle noise, outliers

➡ Find a compromise between local and global fitting
- Fit point cloud in a coarse-to-fine manner
- Segment point cloud into parts, fit individually ….

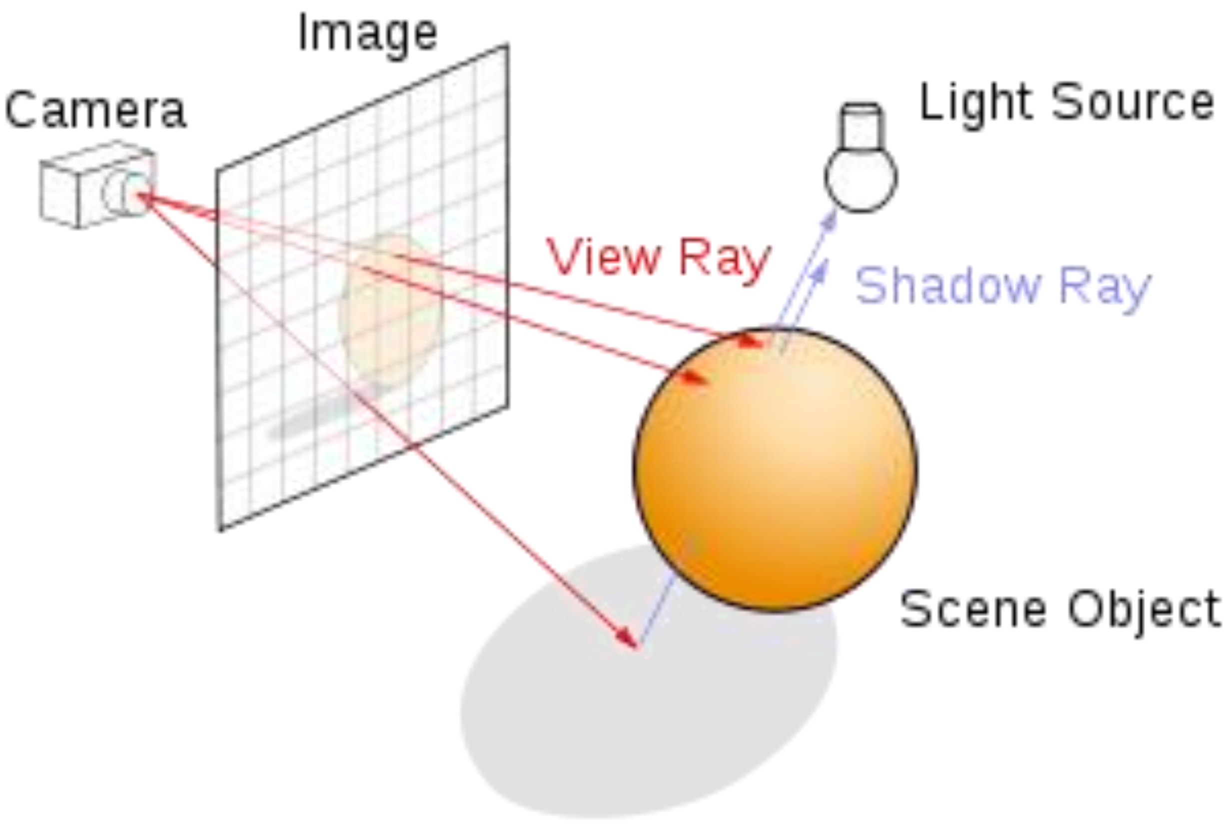
Implicit Surfaces

- Examples of further methods:
  - RBF with floating centers
    - Reduce number of RBF centers
    - Also optimize for the center positions → non-linear
    - [Süßmuth'10] J.Süßmuth "Surface Reconstruction based on Hierarchical Floating Radial Basis Functions"
  - Poisson Reconstruction
    - [Kazhdan'06] M.Kazhdan "Poisson Surface Reconstruction"



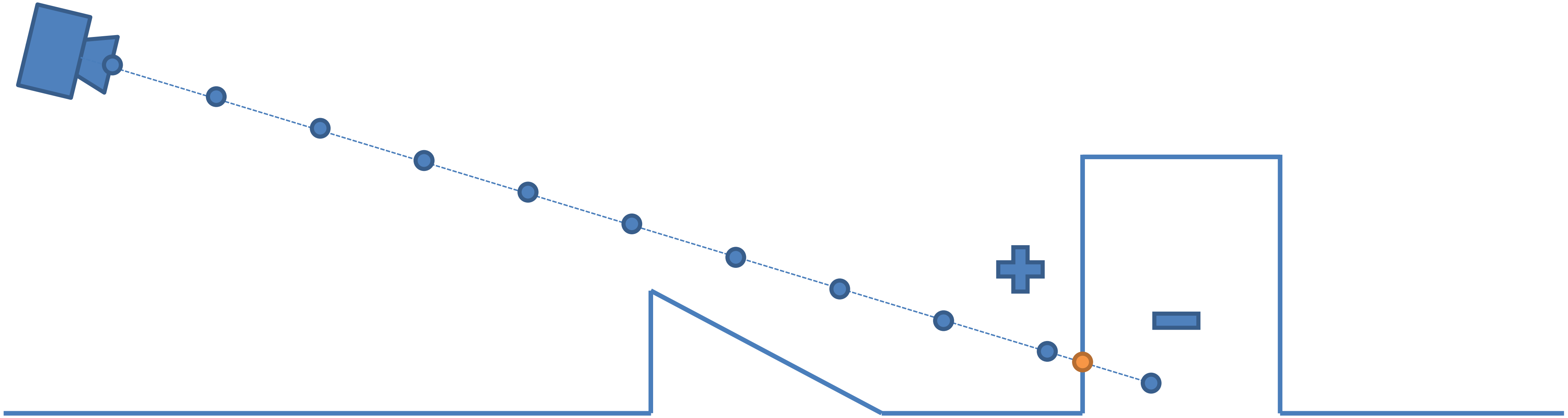
$$\Delta\chi \equiv \nabla \cdot \nabla\chi = \nabla \cdot \vec{V}$$

How to render Signed Distance Functions?



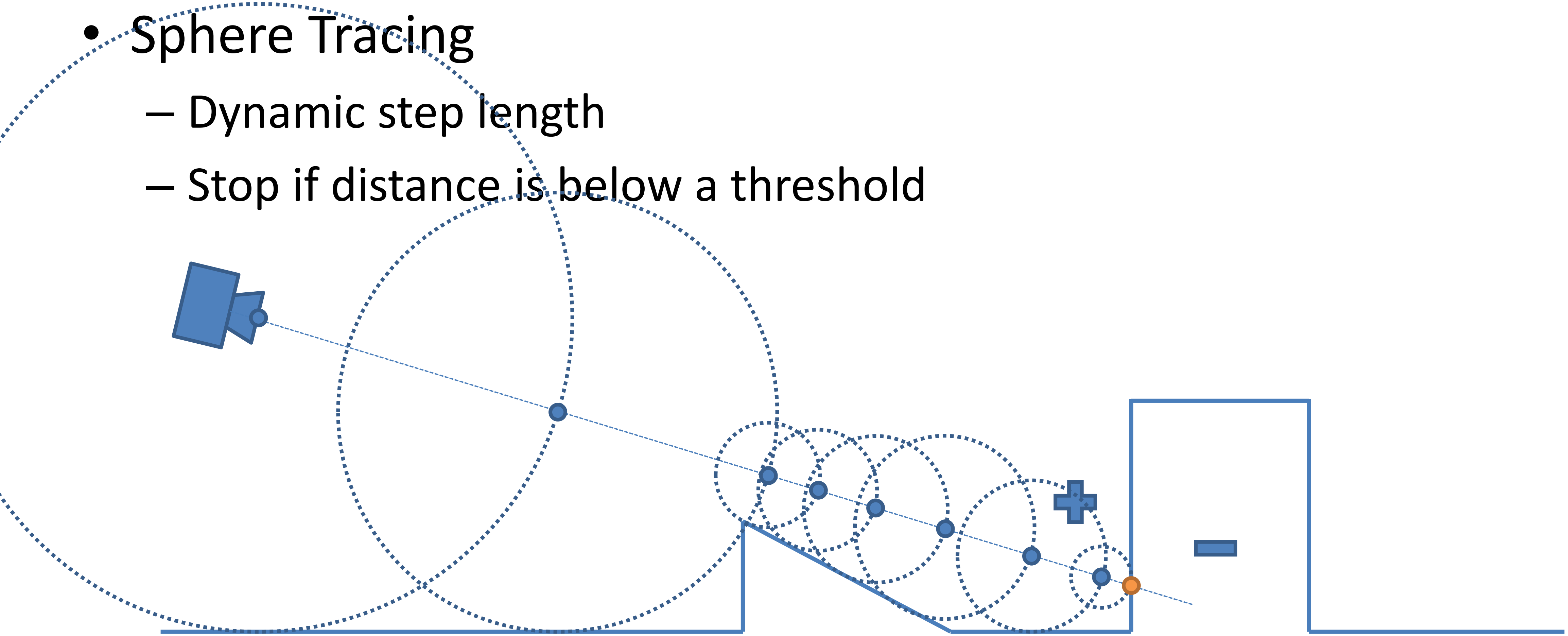
How to render Signed Distance Functions?

- Ray Marching
  - Fixed step length
  - Linear Interpolation, if zero crossing occurs



How to render Signed Distance Functions?

- Sphere Tracing
  - Dynamic step length
  - Stop if distance is below a threshold



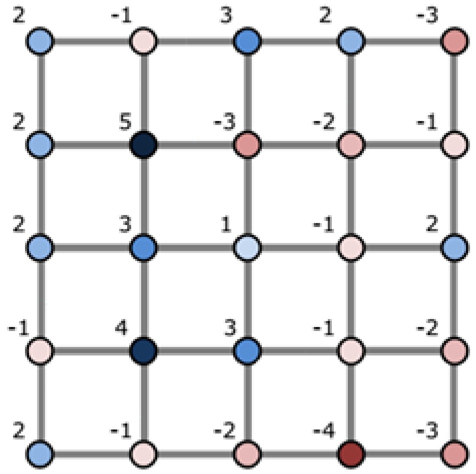
How to render Signed Distance Functions?

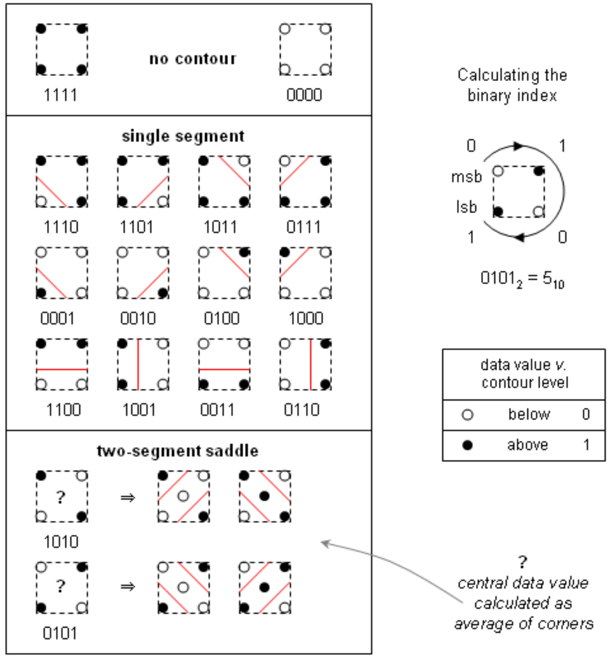
- Convert the iso-surface to polygonal mesh
  - Easy to render
  - Can be used by other post processing pipelines

Marching Cubes

Marching Squares (2D)

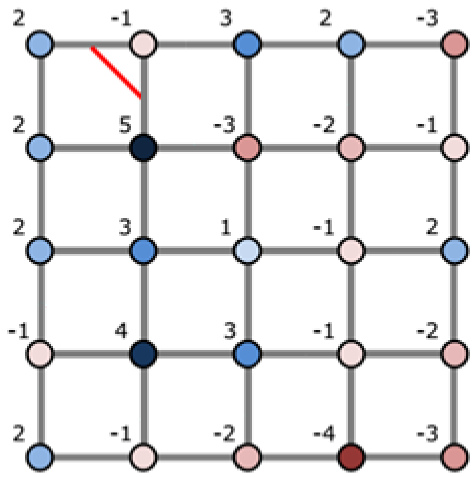
- Converts an iso-line of a bi-variat scalar function to a polygon
  - Given: A uniform sampling (on a 2D grid) of the implicit function
  - For every grid cell determine the zero crossings
    - There are $2^4 = 16$ possible combinations

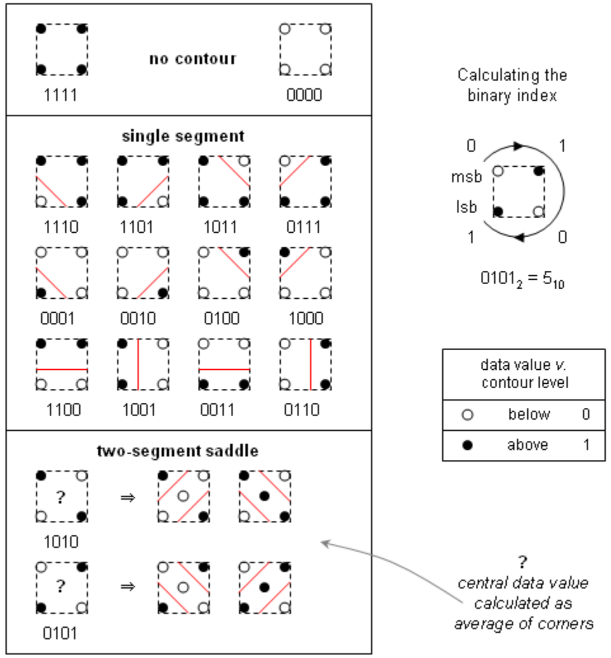




Marching Squares (2D)

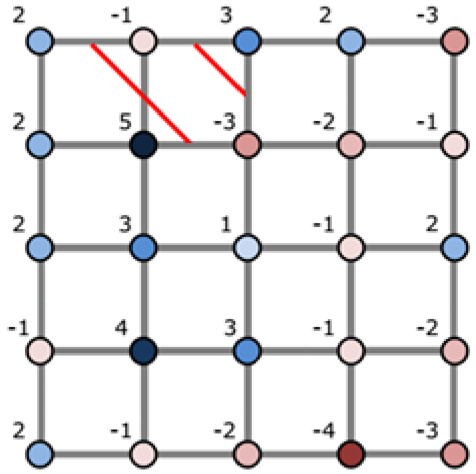
- Converts an iso-line of a bi-variat scalar function to a polygon
  - Given: A uniform sampling (on a 2D grid) of the implicit function
  - For every grid cell determine the zero crossings
    - There are $2^4 = 16$ possible combinations

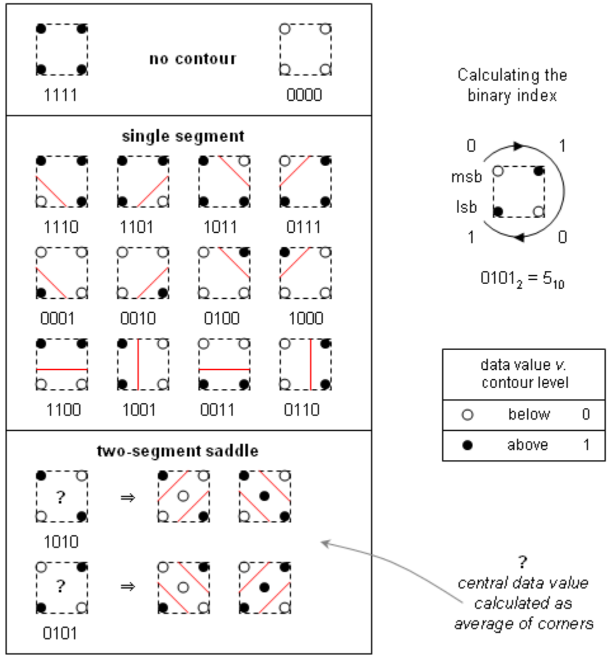




Marching Squares (2D)

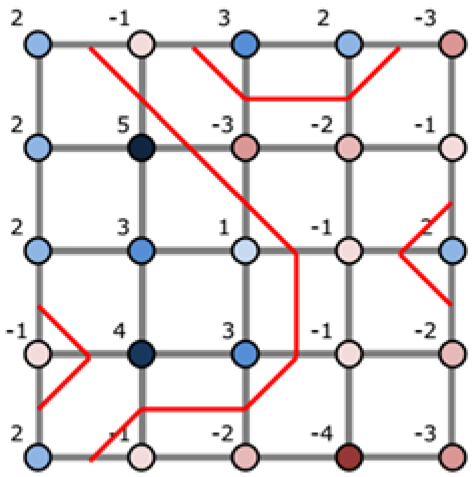
- Converts an iso-line of a bi-variat scalar function to a polygon
  - Given: A uniform sampling (on a 2D grid) of the implicit function
  - For every grid cell determine the zero crossings
    - There are $2^4 = 16$ possible combinations

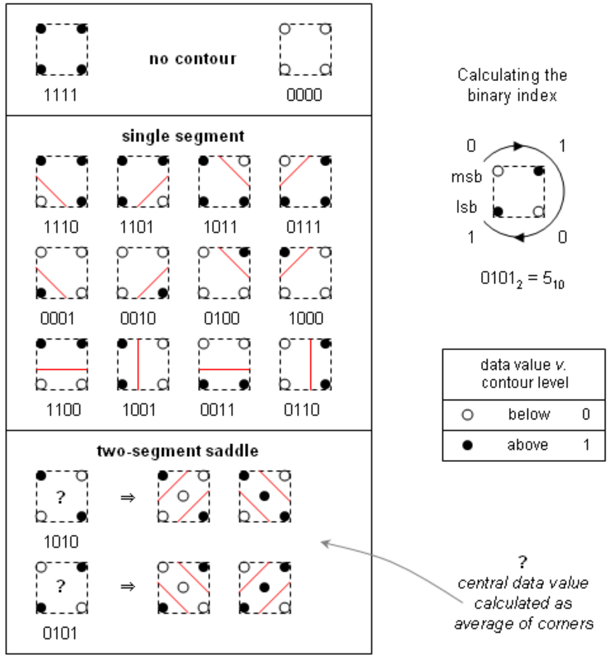




Marching Squares (2D)

- Converts an iso-line of a bi-variat scalar function to a polygon
  - Given: A uniform sampling (on a 2D grid) of the implicit function
  - For every grid cell determine the zero crossings
    - There are $2^4 = 16$ possible combinations





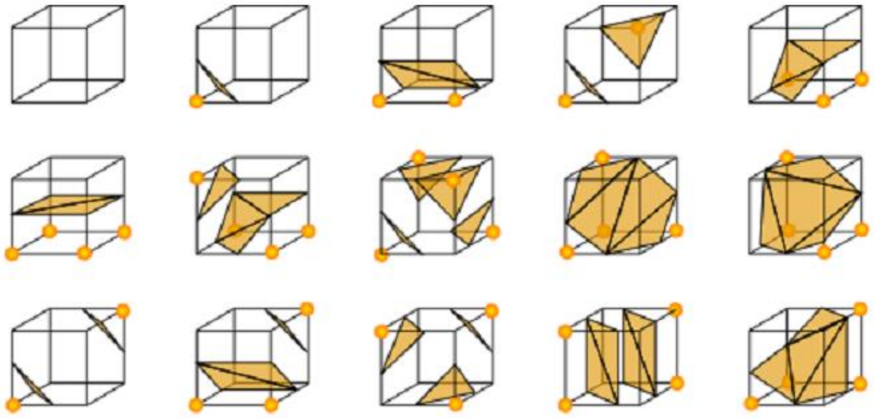
Marching Cubes (3D)

- Converts an iso-surface of a tri-variat scalar function to a polygonal mesh
  - Given: A uniform sampling (on a 3D grid) of the implicit function
  - For every grid cell determine the zero crossings
    - There are $2^8 = 256$ possible combinations
    - Use lookup table to find triangulation
    - Adjust vertex position according to linear interpolation

[Lorensen’87] W.Lorensen et al., “Marching cubes: A high resolution 3D surface construction algorithm“

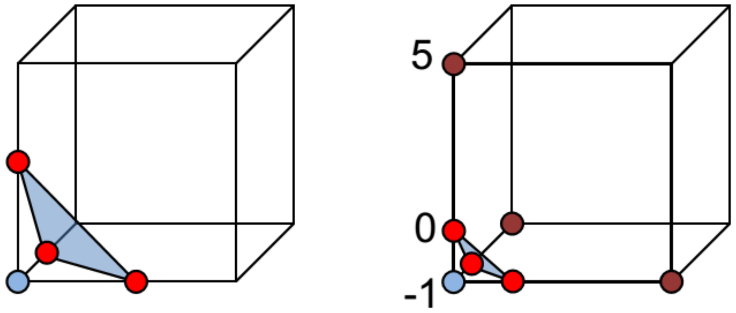
Marching Cubes (3D)

- Lookup table:



Marching Cubes table

- Linear interpolation:



From table Real intersections

Marching Cubes (3D)

```cpp


//////////////////////////////////////////////////////
// tables
//////////////////////////////////////////////////////

// Polygonising a scalar field
// Also known as: "3D Contouring", "Marching Cubes", "Surface Reconstruction"
// Written by Paul Bourke
// May 1994
// http://paulbourke.net/geometry/polygonise/


const static int edgeTable[256] = {
        0x0, 0x109, 0x203, 0x30a, 0x406, 0x50f, 0x605, 0x70c,
        0x80c, 0x905, 0xa0f, 0xb06, 0xc0a, 0xd03, 0xe09, 0xf00,
        0x190, 0x99, 0x393, 0x29a, 0x596, 0x49f, 0x795, 0x69c,
        0x99c, 0x895, 0xb9f, 0xa96, 0xd9a, 0xc93, 0xf99, 0xe90,
        0x230, 0x339, 0x33, 0x13a, 0x636, 0x73f, 0x435, 0x53c,
        0xa3c, 0xb35, 0x83f, 0x936, 0xe3a, 0xf33, 0xc39, 0xd30,
        0x3a0, 0x2a9, 0x1a3, 0xaa, 0x7a6, 0x6af, 0x5a5, 0x4ac,
        0xbac, 0xaa5, 0x9af, 0x8a6, 0xfaa, 0xea3, 0xda9, 0xca0,
        0x460, 0x569, 0x663, 0x76a, 0x66, 0x16f, 0x265, 0x36c,
        0xc6c, 0xd65, 0xe6f, 0xf66, 0x86a, 0x963, 0xa69, 0xb60,
        0x5f0, 0x4f9, 0x7f3, 0x6fa, 0x1f6, 0xff, 0x3f5, 0x2fc,
        0xdfc, 0xcf5, 0xfff, 0xef6, 0x9fa, 0x8f3, 0xbf9, 0xaf0,
        0x650, 0x759, 0x453, 0x55a, 0x256, 0x35f, 0x55, 0x15c,
        0xe5c, 0xf55, 0xc5f, 0xd56, 0xa5a, 0xb53, 0x859, 0x950,
        0x7c0, 0x6c9, 0x5c3, 0x4ca, 0x3c6, 0x2cf, 0x1c5, 0xcc,
        0xfcc, 0xec5, 0xdcf, 0xcc6, 0xbca, 0xac3, 0x9c9, 0x8c0,
        0x8c0, 0x9c9, 0xac3, 0xbca, 0xcc6, 0xdcf, 0xec5, 0xfcc,
        0xcc, 0x1c5, 0x2cf, 0x3c6, 0x4ca, 0x5c3, 0x6c9, 0x7c0,
        0x950, 0x859, 0xb53, 0xa5a, 0xd56, 0xc5f, 0xf55, 0xe5c,
```

```cpp
const static int triTable[256][16] =
{ { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 1, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 8, 3, 9, 8, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 3, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 2, 10, 0, 2, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 2, 8, 3, 2, 10, 8, 10, 9, 8, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 11, 2, 8, 11, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 9, 0, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 11, 2, 1, 9, 11, 9, 8, 11, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 10, 1, 11, 10, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 10, 1, 0, 8, 10, 8, 11, 10, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 9, 0, 3, 11, 9, 11, 10, 9, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 8, 10, 10, 8, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 7, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 3, 0, 7, 3, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 1, 9, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 1, 9, 4, 7, 1, 7, 3, 1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 2, 10, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 4, 7, 3, 0, 4, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 2, 10, 9, 0, 2, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1 },
{ 2, 10, 9, 2, 9, 7, 2, 7, 3, 7, 9, 4, -1, -1, -1, -1 },
{ 8, 4, 7, 3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 11, 4, 7, 11, 2, 4, 2, 0, 4, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 0, 1, 8, 4, 7, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 7, 11, 9, 4, 11, 9, 11, 2, 9, 2, 1, -1, -1, -1, -1 },
{ 3, 10, 1, 3, 11, 10, 7, 8, 4, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 11, 10, 1, 4, 11, 1, 0, 4, 7, 11, 4, -1, -1, -1, -1 },
{ 4, 7, 8, 9, 0, 11, 9, 11, 10, 11, 0, 3, -1, -1, -1, -1 },
{ 4, 7, 11, 4, 11, 9, 9, 11, 10, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 5, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
```

https://github.com/angeladai/VirtualScan/blob/master/src/Tables.h

Administrative

Reading Homework:

- Poisson Surface Reconstruction [Kazhdan et al.]
  https://hhoppe.com/poissonrecon.pdf
- Screened Poisson Surface Reconstruction [Kazhdan and Hoppe]
  https://www.cs.jhu.edu/~misha/MyPapers/ToG13.pdf

Next week:

- Overview of 3D reconstruction methods

Administrative

See you next week!