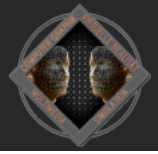
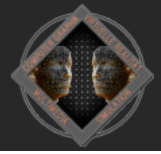
Advanced Topics in Computer Graphics II
Geometry Processing

Surface Representations II - Implicits, Moving Least Squares and Algebraic Point Set Surfaces

October 21, 2024

Implicit surface representations

Definition (Implicit surface)

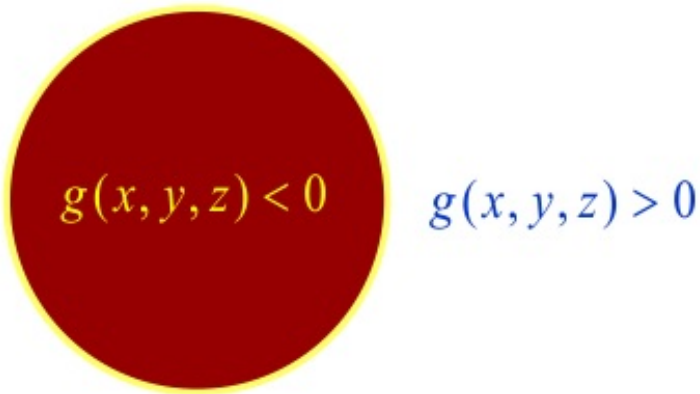
Let $g\colon \mathbb{R}^3 \to \mathbb{R}$ be a continuous function. Then the set

$$\mathcal{I} = \{g^{-1}(0)\} = \{\boldsymbol{p} \in \mathbb{R}^3 \mid g(\boldsymbol{p}) = 0\}$$

is called the *implicitly defined surface* or *isosurface*.

Implicit surfaces decompose the volume into three parts:

- $\{\boldsymbol{p} \in \mathbb{R}^3 \mid g(\boldsymbol{p}) < 0\}$: Interior
- $\{\boldsymbol{p} \in \mathbb{R}^3 \mid g(\boldsymbol{p}) = 0\}$: Surface
- $\{\boldsymbol{p} \in \mathbb{R}^3 \mid g(\boldsymbol{p}) > 0\}$: Exterior



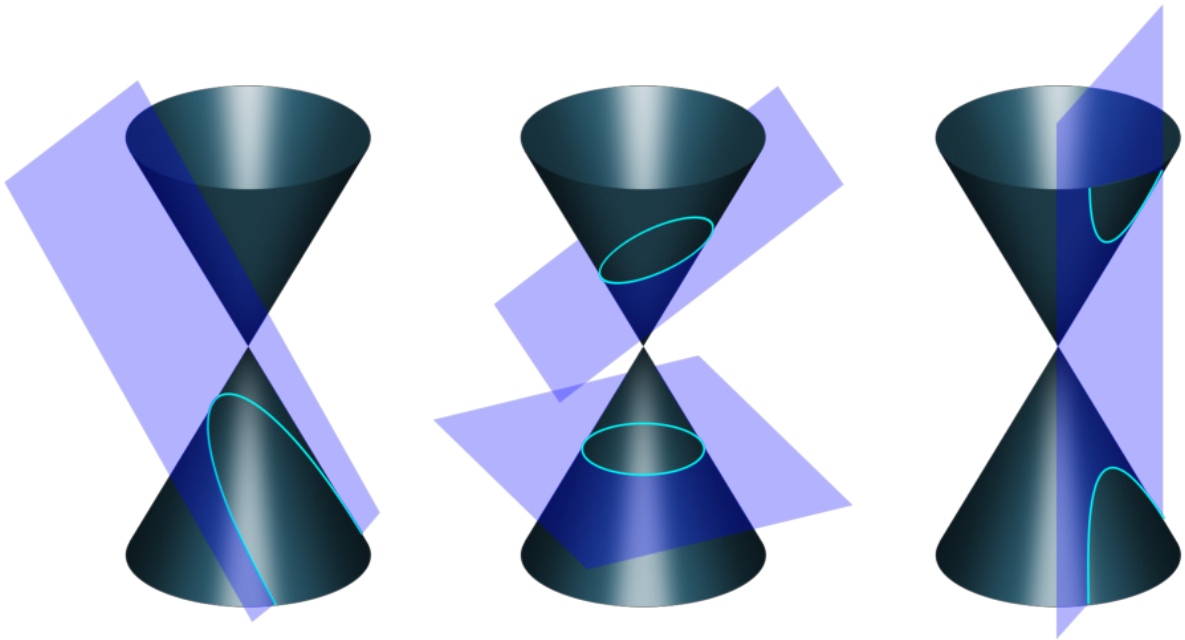
$$g(x,y,z) = x^2 + y^2 + z^2 - r^2$$

Definition (Quadrics)

Let $g\colon \mathbb{R}^2 \to \mathbb{R}$

$$g(x,y,z) = \sum_{i+j+k=2} a_{ijk} x^i y^i w^k = a_{200}c^2 + a_{100}xy + a_{101}xw + a_{020}y^2 + a_{011}yw + a_{002}w^2$$

The point set $Q := \{p \in \mathbb{R}^2 \colon g(p) = 0\}$ is called Quadric or Conic Section in $\mathbb{R}^2$



Algebraic Surfaces of degree n in $\mathbb{R}^3$ are implicitly defines by (**homogeneous**) polynomial functions.

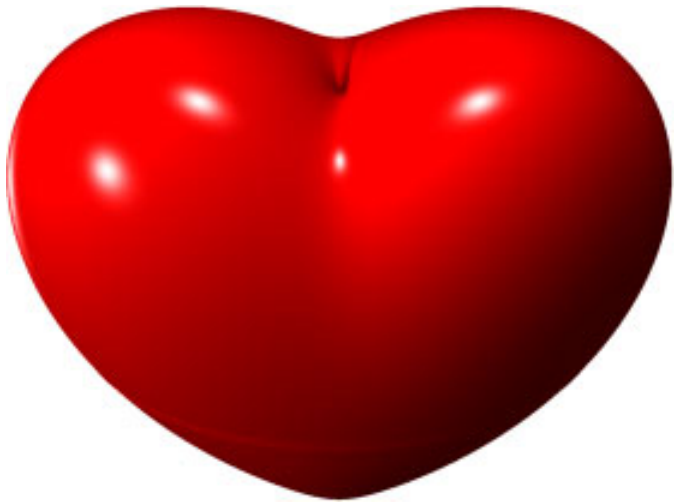
Definition (Algebraic surfaces of degree n)

Let $g\colon \mathbb{R}^3 \to \mathbb{R}$

$$g(x,y,z,w) = \sum_{i+j+k+l=n} a_{ijk} x^i y^i z^k w^l$$

Heart Surface[1]:

$$3\left(x^2 + \frac{9}{4}y^2 + z^2 - 1\right) - x^2 z^3 - \frac{9}{80} y^2 z^3 = 0$$



[1] G. Taubin. "Rasterizing algebraic curves and surfaces". In: *IEEE Computer graphics and applications* 14.2 (1994), pp. 14–23.

Definition (Gradient in a volume)

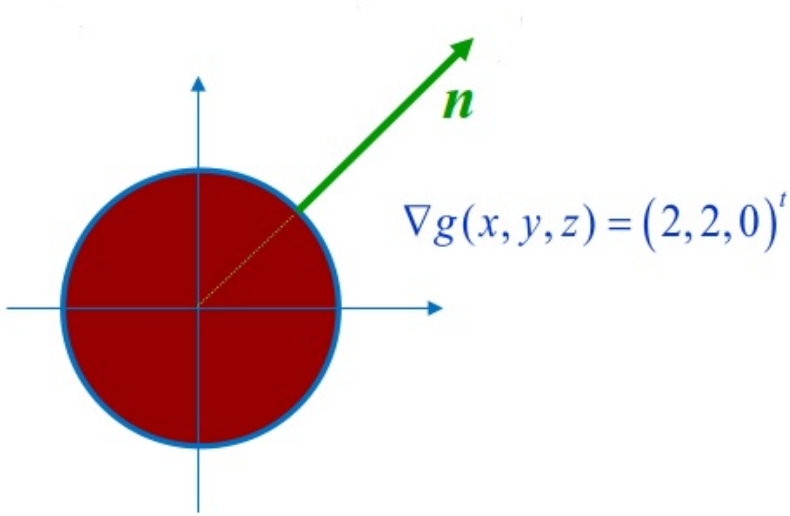
Let $g\colon \mathbb{R}^3 \to \mathbb{R}$ be a continuous function. The vector

$$\nabla g(x,y,z) = \left(\frac{\partial g(x,y,z)}{\partial x}, \frac{\partial g(x,y,z)}{\partial y}, \frac{\partial g(x,y,z)}{\partial z} \right)^T$$

is called the *gradient*. The normal of a point in the volume has the same direction as the gradient.

Example:

$$g(x,y,z) = x^2 + y^2 + z^2 - r^2$$
$$\nabla g(x,y,z) = (2x, 2y, 2z)^T$$



Theorem (Implicit Function Theorem)

Let $U \subset \mathbb{R}^2$ be open and $g\colon U \to \mathbb{R}$ have continuous partial derivatives $\partial_x g, \partial_y g$ on U. Assume that $(x_0, y_0) \in U$ is on the zero-level of g, i.e. $g(x_0, y_0) = 0$, and that $\partial_y g(x_0, y_0) \neq 0$.
Then there exist an $\epsilon > 0$ and a function $h\colon (x_0 - \epsilon, x_0 + \epsilon) \to \mathbb{R}$ such that the following hold

- $\forall x \in (x_0 - \epsilon, x_0 + \epsilon)\colon g(x, h(x)) = 0$
- *h is differentiable on its domain with*

$$h'(x) = \frac{-\partial_x g(x, h(x))}{\partial_y g(x, h(x))}$$

Note: The theorem can be generalized to $n \geq 2$.

This implies the following:

- The gradient on the surface is non-zero: $\nabla g \neq 0$
- There exists a well-defined tangent plane for all points on the surface

Definition (Union and Intersection of volumes)

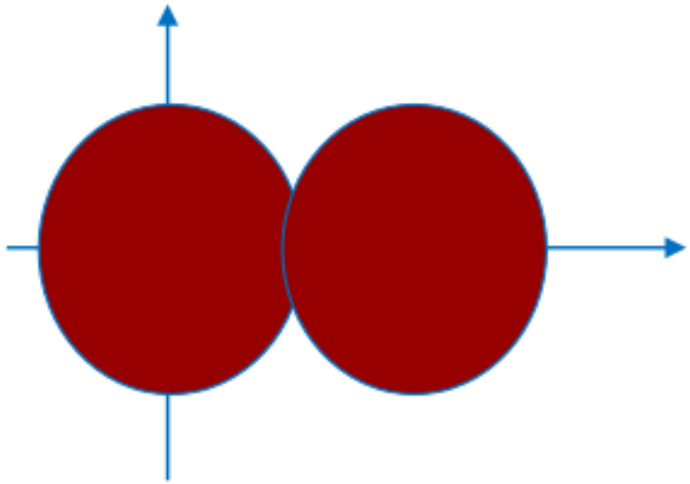
Let $g_1, \ldots, g_n : \mathbb{R}^3 \to \mathbb{R}$ be continuous functions. The union and intersection of them are defined as

$$\bigcup_{i=1}^{n} g_i(\boldsymbol{p}) := \min_{i=1,\ldots,n} g_i(\boldsymbol{p})$$

$$\bigcap_{i=1}^{n} g_i(\boldsymbol{p}) := \max_{i=1,\ldots,n} g_i(\boldsymbol{p})$$

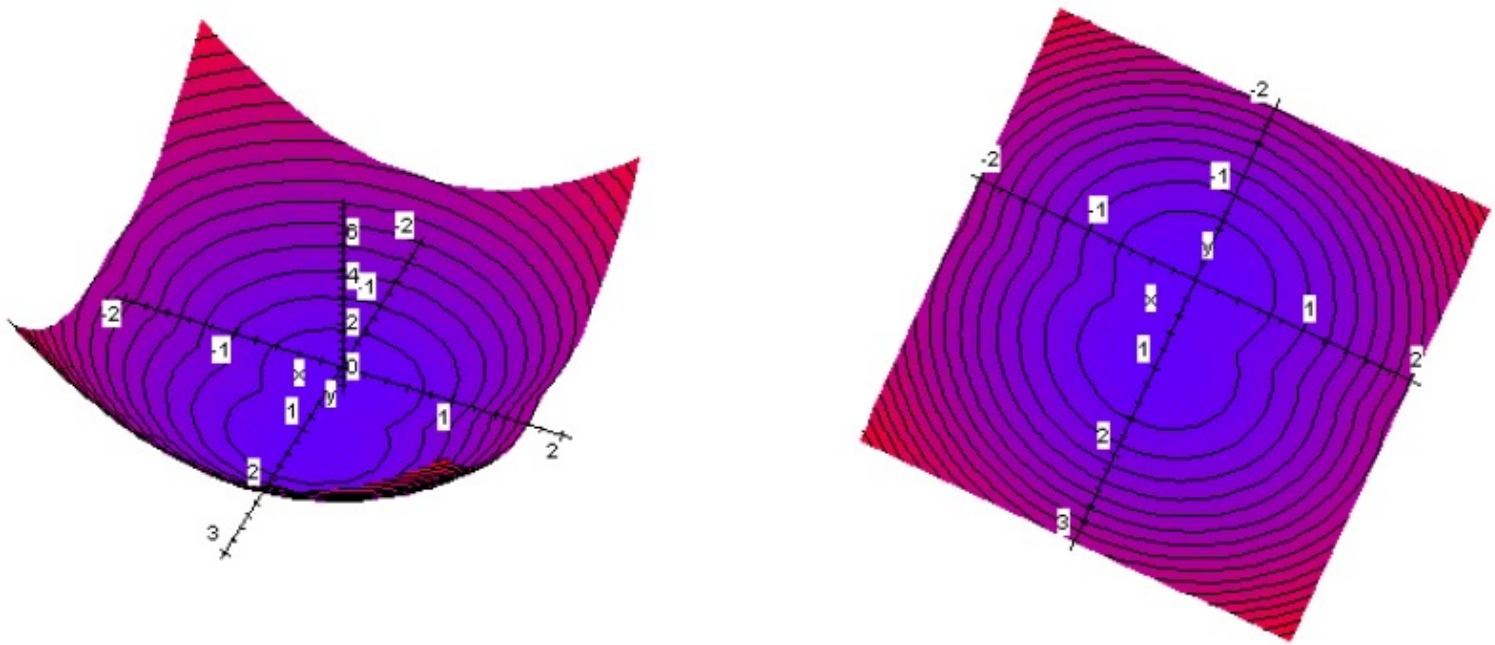
Example:

$$g_1(x, y, z) = (x - 0)^2 + (y - 0)^2 + (z - 0)^2 - 1$$
$$g_2(x, y, z) = (x - 0.9)^2 + (y - 0)^2 + (z - 0)^2 - 1$$



Example:

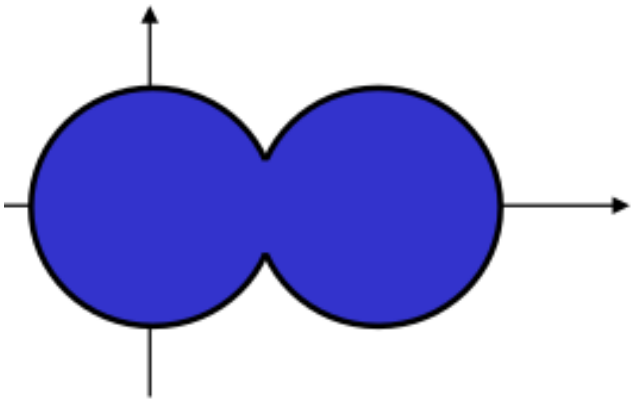
$$g_1(x, y, z) = (x - 0)^2 + (y - 0)^2 + (z - 0)^2 - 1$$
$$g_2(x, y, z) = (x - 0.9)^2 + (y - 0)^2 + (z - 0)^2 - 1$$

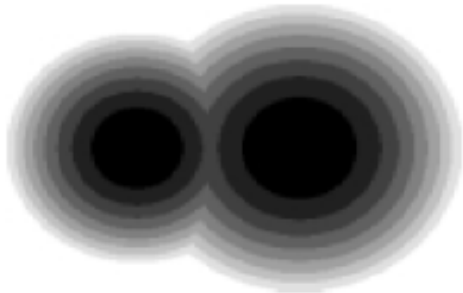


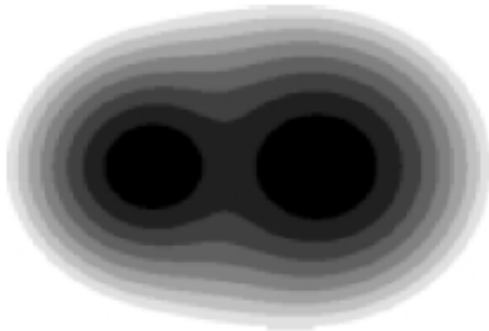
$$g(x, y, z) = \min \{g_1(x, y, z), g_2(x, y, z)\}$$

In many cases, smooth blending is desired[2]:

$$f \cup g = \frac{1}{1+\alpha}\left(f + g - \sqrt{f^2 + g^2 - 2\alpha f g}\right)$$
$$f \cap g = \frac{1}{1+\alpha}\left(f + g + \sqrt{f^2 + g^2 - 2\alpha f g}\right)$$

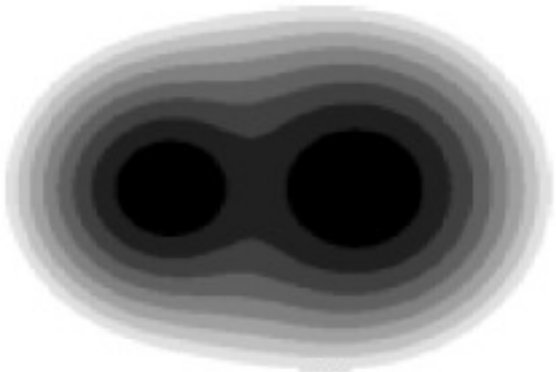




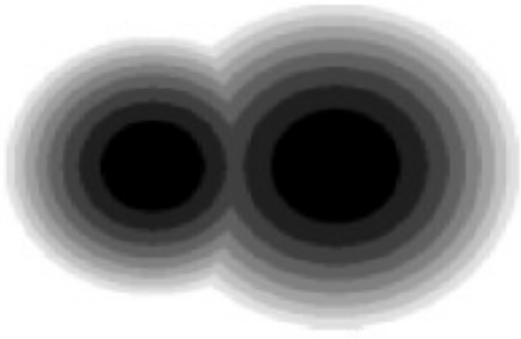


[2] A. Pasko et al. "Function representation in geometric modeling: concepts, implementation and applications". In: *The visual computer* 11.8 (1995), pp. 429–446.

Examples:



$\alpha = 0$



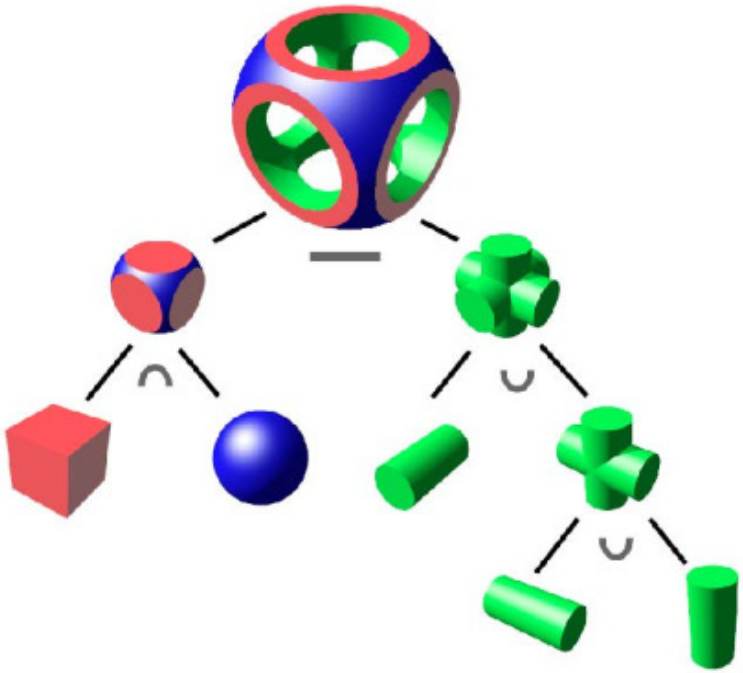
$\alpha = 1$

For $\alpha = 1$ this is equivalent to $\min$ and $\max$:

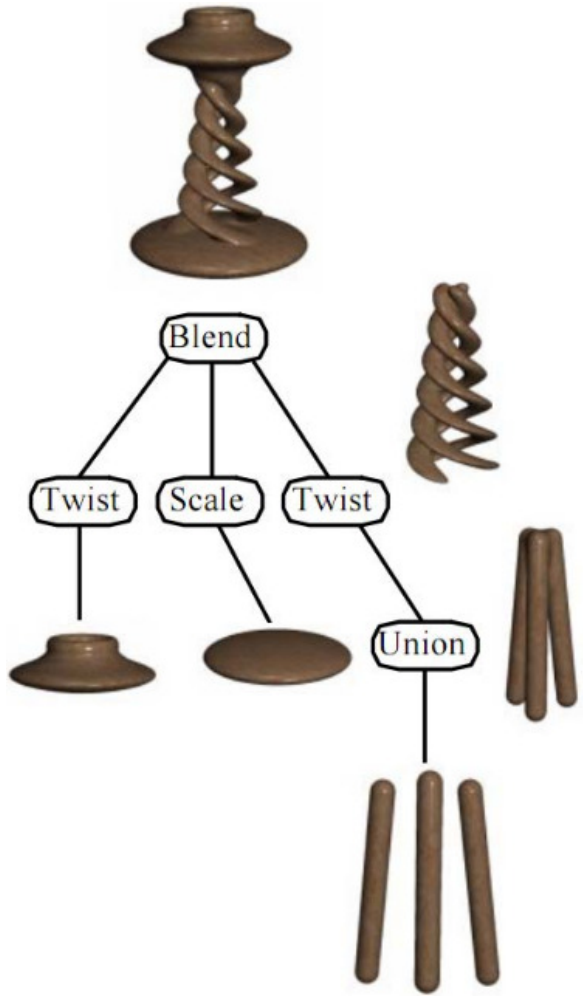
$$f \cup g = \frac{1}{2}\left(f + g - \sqrt{(f-g)^2}\right) = \frac{f+g}{2} - \frac{|f-g|}{2} = \min f, g$$

$$f \cap g = \frac{1}{2}\left(f + g + \sqrt{(f-g)^2}\right) = \frac{f+g}{2} + \frac{|f-g|}{2} = \max f, g$$

Combine multiple operations into a tree[3]:

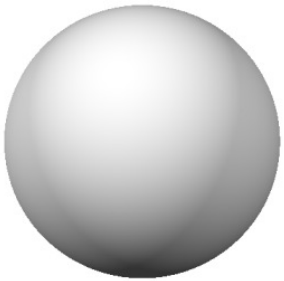


CSG tree



[3] B. Wyvill et al. "Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system". In: *Computer Graphics Forum*. Vol. 18. 2. Wiley Online Library. 1999, pp. 149–158.

Sphere as zero set of a function:



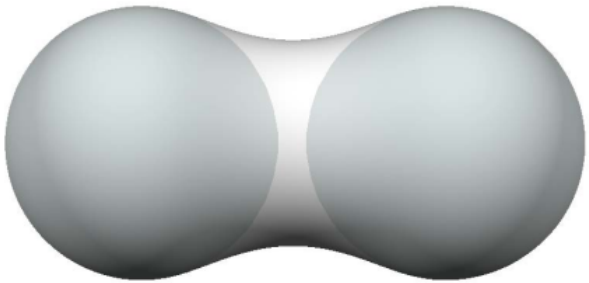
$$f(\boldsymbol{p}) = \|\boldsymbol{p}\|^2 - r^2$$

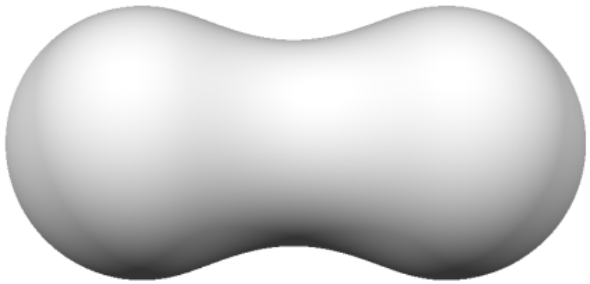
Same sphere can be defined as level set at e^{-1}:

$$f(\boldsymbol{p}) = e^{-\|\boldsymbol{p}\|^2/r^2} \quad \text{at } e^{-1}$$

- With smooth falloff functions, adding implicit functions generates a blend called **"Metaballs"** or **"Blobs"**:

$$f(\boldsymbol{p}) = e^{-\|\boldsymbol{p}-\boldsymbol{p}_1\|^2} + e^{-\|\boldsymbol{p}-\boldsymbol{p}_2\|^2}$$





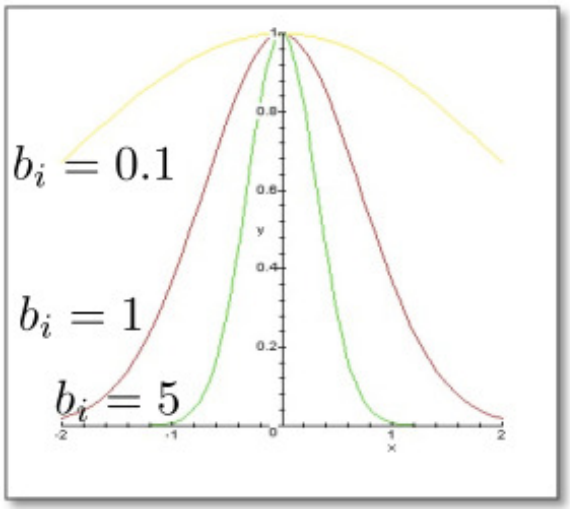
Blobs

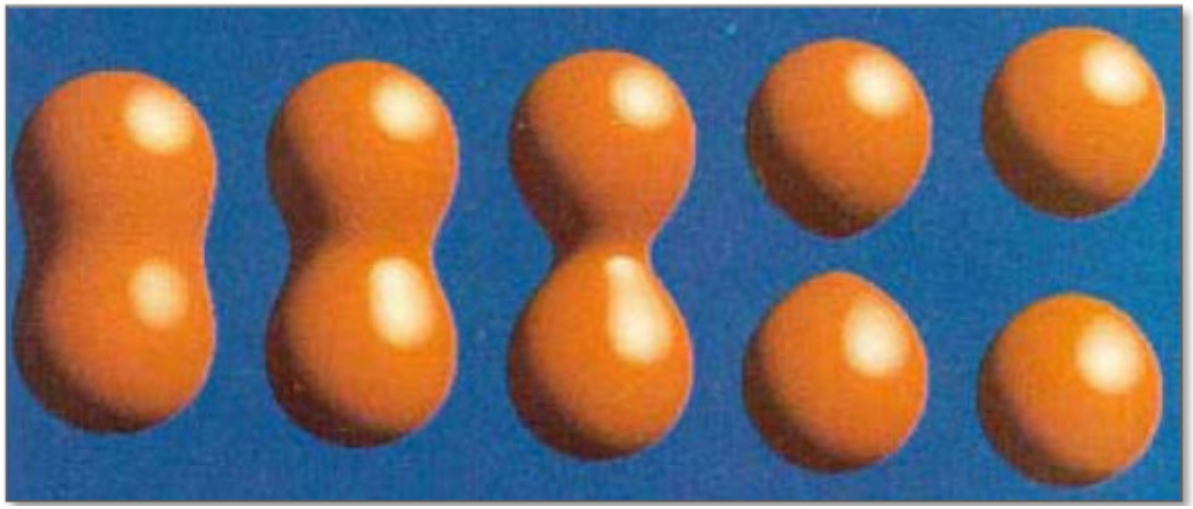
Suggested by Blinn, 1982

- Defined implicitly by a potential function around a point $\boldsymbol{p}_i$:

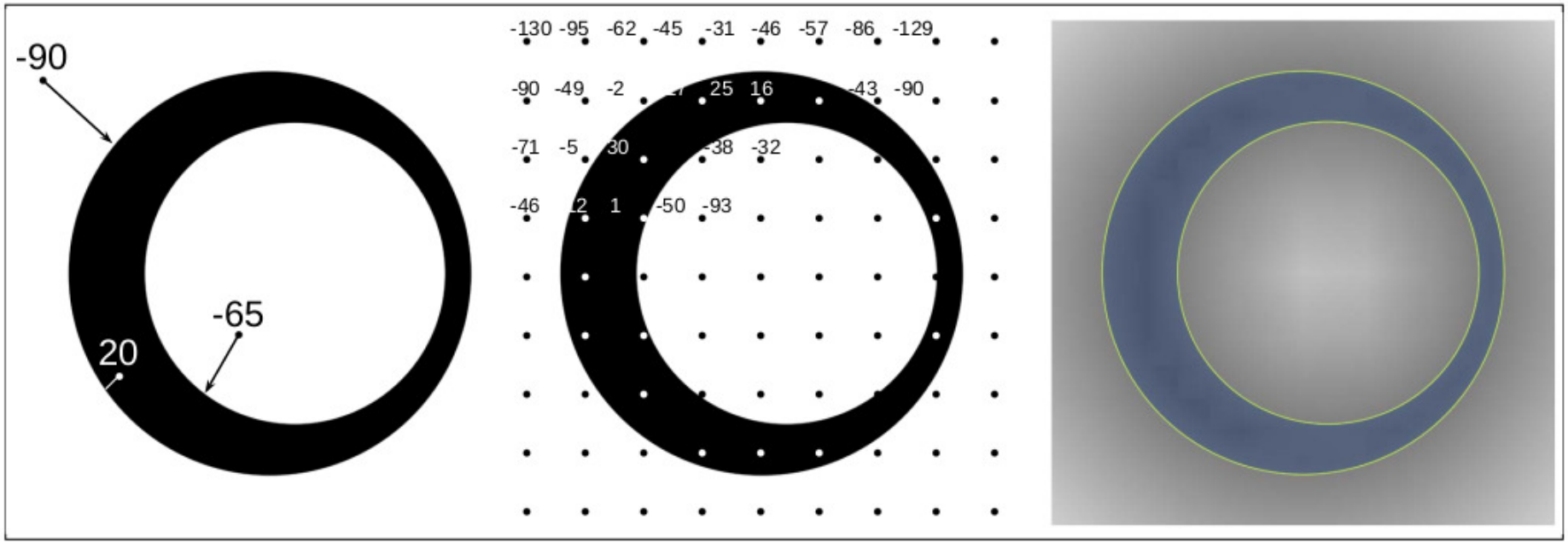
$$f(\boldsymbol{p}) = a_i e^{-b_i \|\boldsymbol{p}-\boldsymbol{p}_i\|^2}$$

- Set operations by simple addition/subtraction





Specify the (possibly) signed distance to a shape



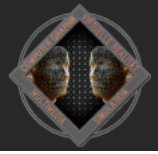
$2D$ shape with sampled distances to its edge

Regularly sampled distance values

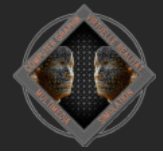
$2D$ distance field

Regularly Sampled Distance Fields

- Similar to regularly sampled images, insufficient sampling of distance fields results in aliasing
- Because dine detail requires dense sampling, excessive memory is required **with regularly** sampled distance fields when **any** fine detail is present



Adaptively Sampled Distance Fields (ADFs)[4]

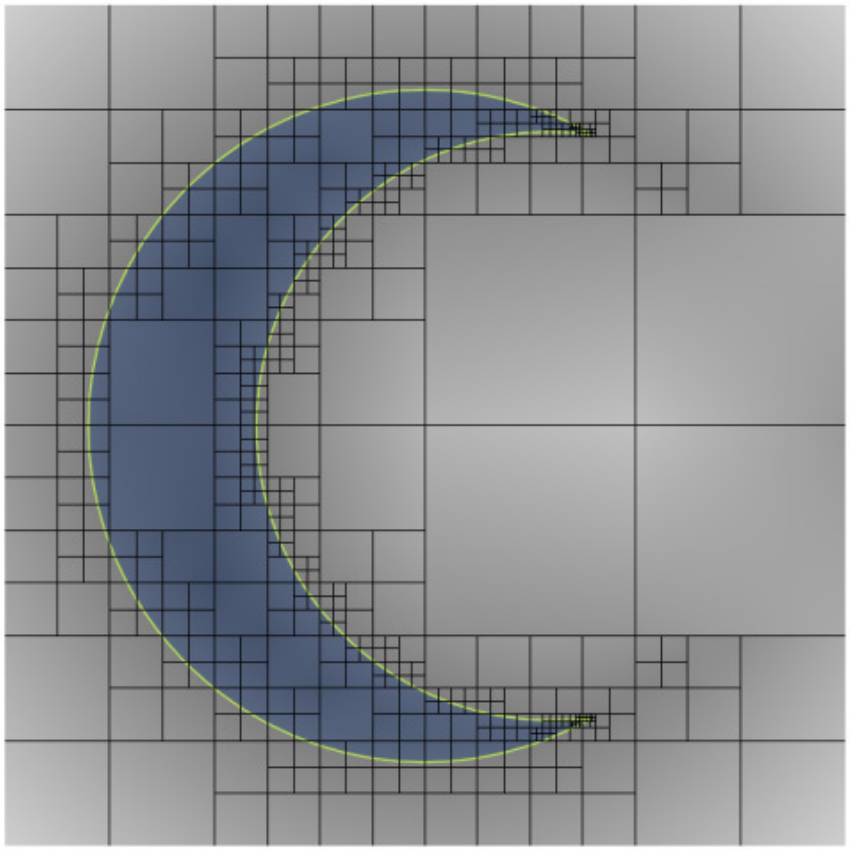


- Detail-directed sampling
  - High sampling rates only where needed
- Spatial data structure (e.g. an octree)
  - Fast localization for efficient processing
- Reconstruction method (e.g. trilinear interpolation)
  - For reconstructing the distance field and its gradient from the sampled distances values

S. F. Frisken et al. "Adaptively sampled distance fields: A general representation of shape for computer graphics". In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, pp. 249–254

[4] S. F. Frisken et al. "Adaptively sampled distance fields: A general representation of shape for computer graphics". In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, pp. 249–254.

Advantages of ADFs



ADFs provide:

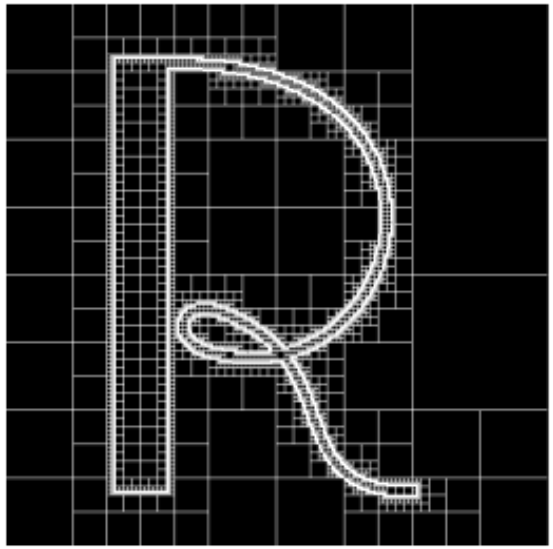
- Spatial hierarchy
- Distance field
- Object surface
- Object interior
- Object exterior
- Surface normal (gradient at surface)
- Direction to closest surface point (gradient off surface)

ADFs consolidate the data needed to represent complex objects

Comparison of 3-color Quadtrees and ADFs

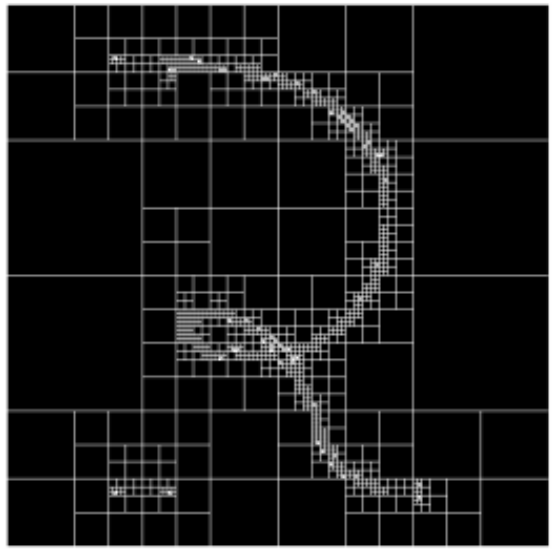
- 3-**color quadtree**: cells are subdivided to their maximum level if they contain the shape's boundary

  $23,573$ cells

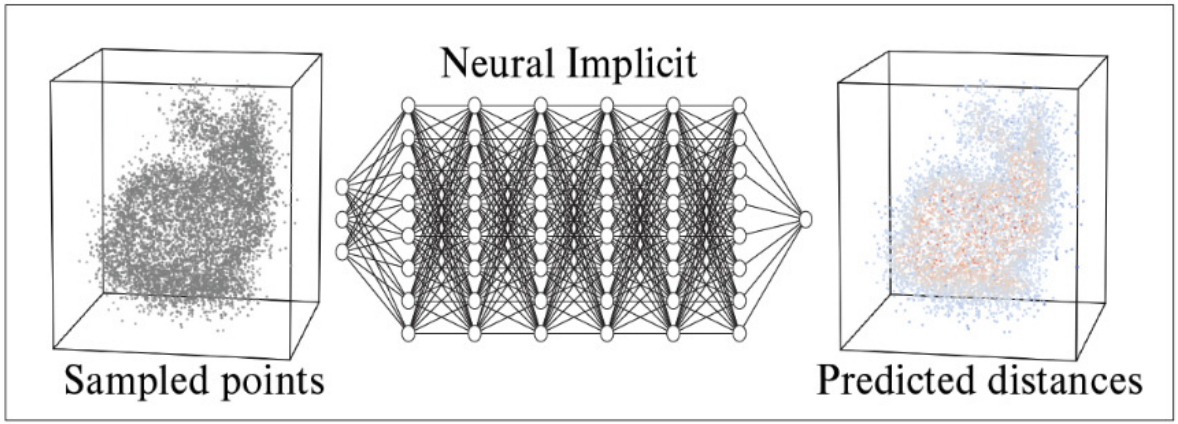


- **ADFs** with biquadratic reconstruction function in which cells are subdivided according to local detail in the distance field

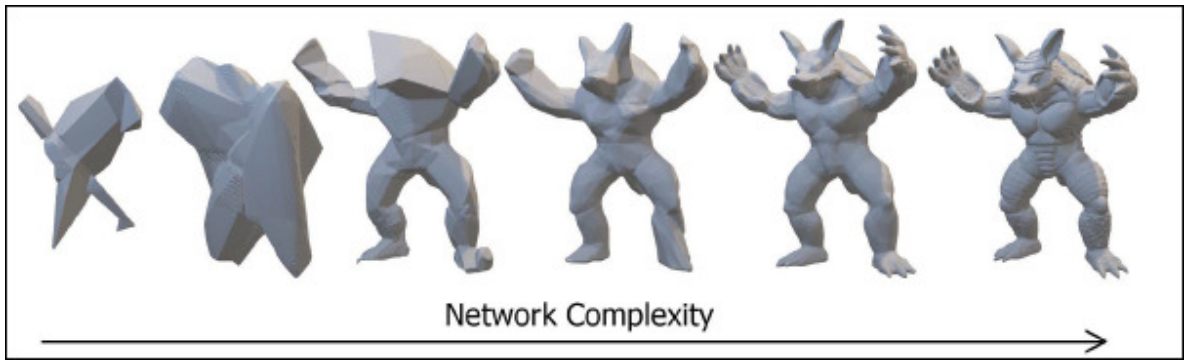
  1713 cells



Overfit Neural Networks



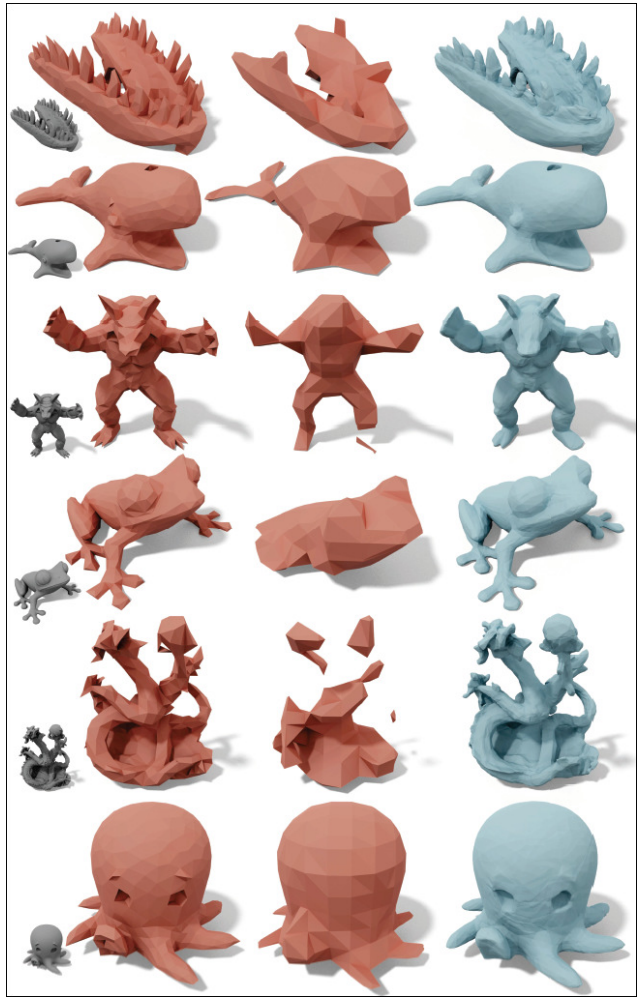
- Overfit SDF network architecture[5]: Given point samples of an object's SDF (left), we train a feed-forward neural network (middle) to predict the signed distances (right) of each input point



[5] T. Davies et al. "Overfit Neural Networks as a Compact Shape Representation". In: *arXiv preprint arXiv:2009.09808* (2020).

Overfit Neural Networks

- Neural implicit format (right) can be shown to better approximate the original surface (grey, inset) compared to adaptive decimation of the original triangle mesh using the Garland & Heckbert algorithm (left) and uniform signed distance grid (middle) with equal memory impact.



6

[6] M. Garland and P. S. Heckbert. "Surface simplification using quadric error metrics". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 209–216.

Let $D\colon \mathbb{R}^3 \to \mathbb{R}^3$ be a transformation or deformation and g a real valued function defining a volume. Then, the transformed volume is defined by a function $\tilde{g}$ with

$$\tilde{g}(D\boldsymbol{p}) = g(\boldsymbol{p}) \forall \boldsymbol{p} \in \mathbb{R}^3$$

and therefore we get

$$\tilde{g} = g \circ D^{-1}$$

Example: Translation of a circle around the origin by the vector (x_0, y_0):

$$\begin{aligned}\tilde{g}(x,y) &= g \circ D^{-1}(x,y) \\ &= g(x - x_0, y - y_0) \\ &= (x - x_0)^2 + (y - y_0)^2 - r^2\end{aligned}$$

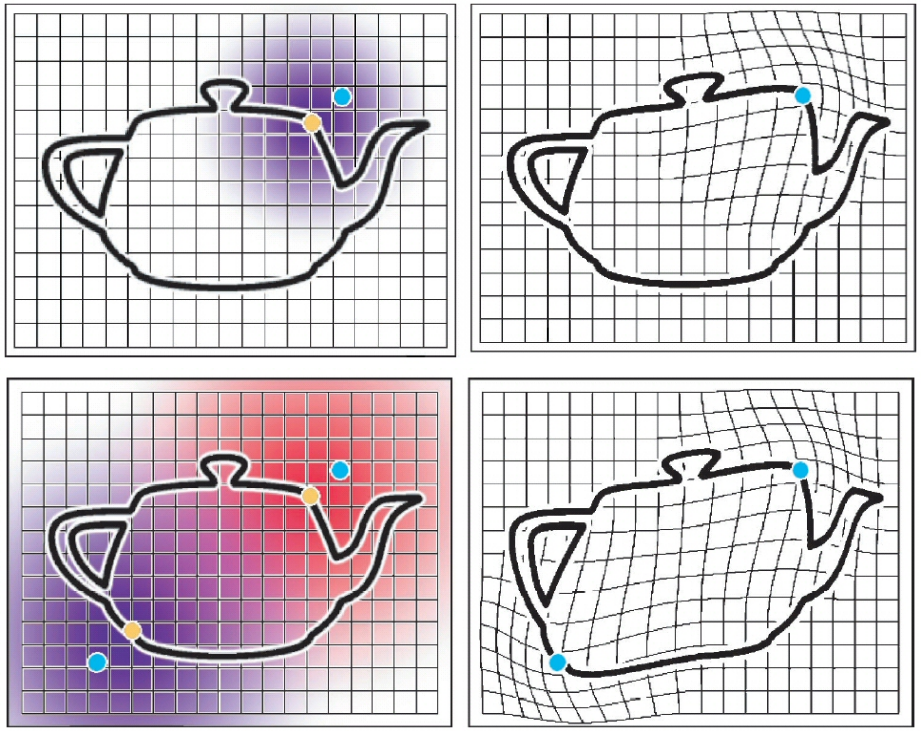
Deform the object's geometry

- control point / vertex manipulation
- Warp the space in which the object is embedded (**Spatial Deformation**)
- Main techniques:
  - Nonlinear Deformation (Barr)
  - Free Form Deformation (FFD)
  - Curve-based Deformation

Warps the space that the mesh is embedded in

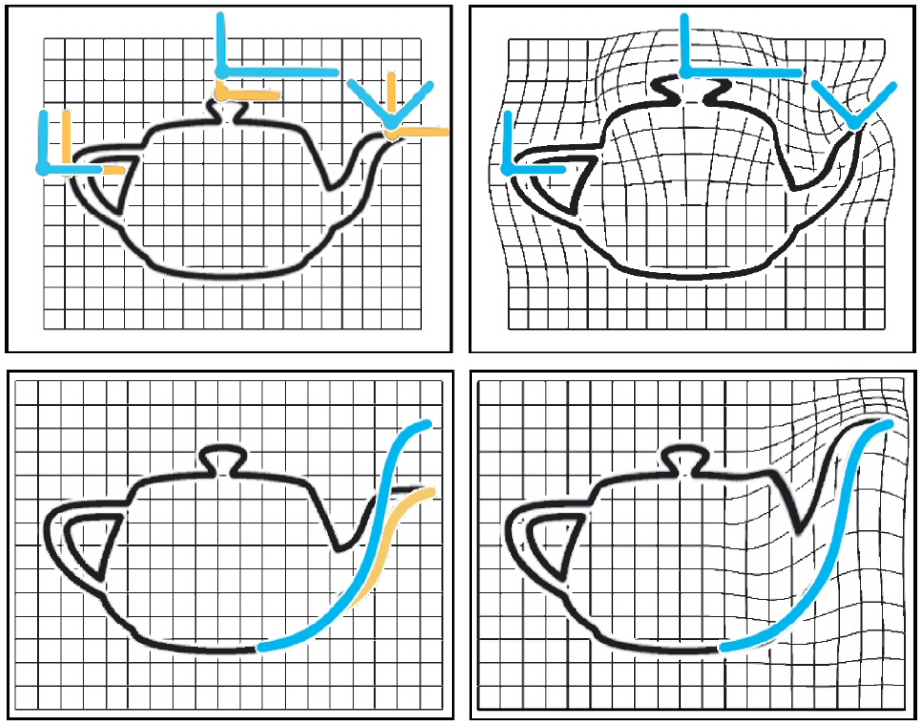
User inputs **pair of features** to guide the deformation

- Discrete point pairs
- Local coordinate frames
- Continuous curves



User inputs **pair of features** to guide the deformation

- Discrete point pairs
- Local coordinate frames
- Continuous curves

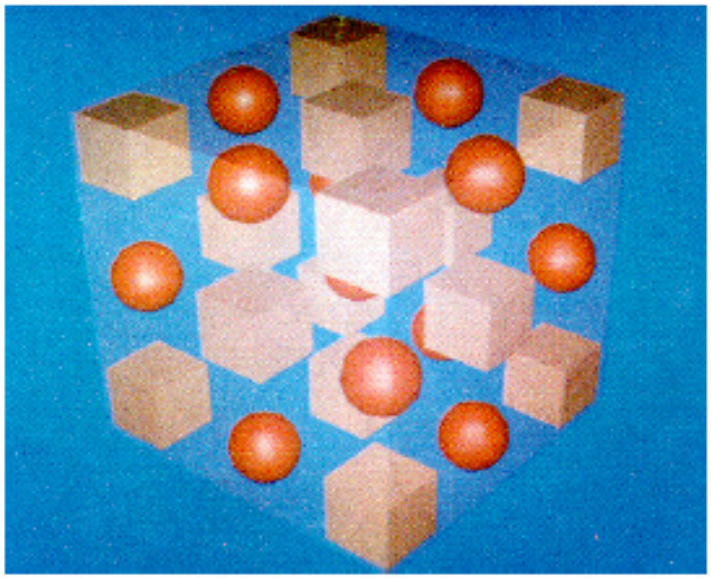


Spatial Deformation: General Framework

Spatial deformation defines a mapping on the space that surrounds the model

- It can be performed on all models that use point-based data
  - Pixels in images
  - Control points in spline surfaces
  - Vertices in a mesh
  - Point cloud data
- Deformation is propagated along the **space**
  - May not be what the user expects

Free Form Deformation (FFD)



T. W. Sederberg and S. R. Parry. “Free-form deformation of solid geometric models”. In: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 1986, pp. 151–160

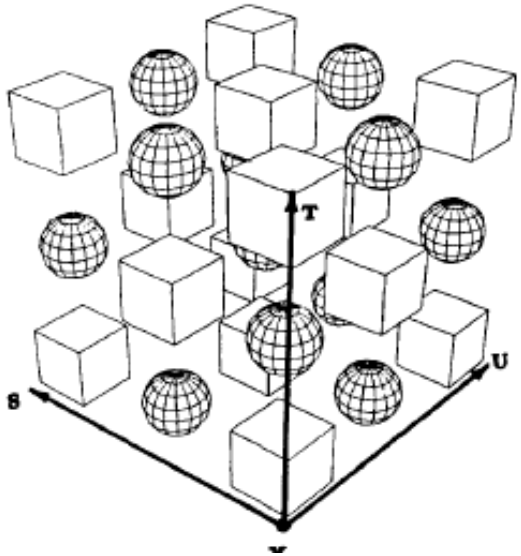
Free Form Deformation (FFD)

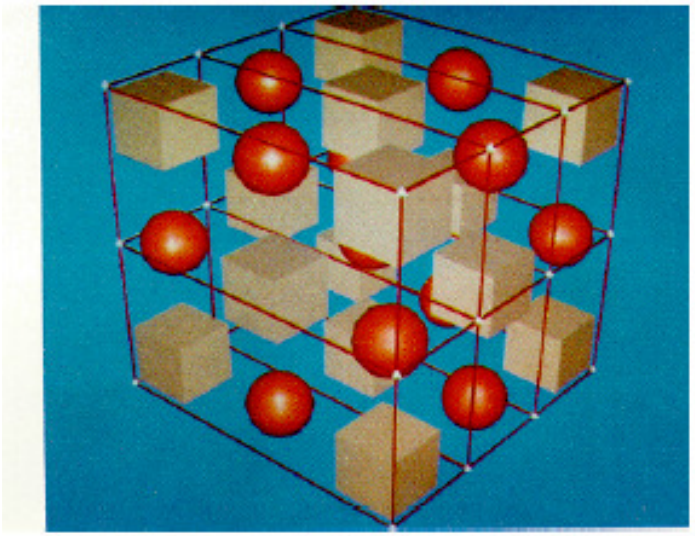
Define a volume using parallel piped **lattice**

Lattice defines a coordiante system (S, T, U)

Modify the lattice points

Deformation of a point in the space depends on its (S, T, U) coordinates





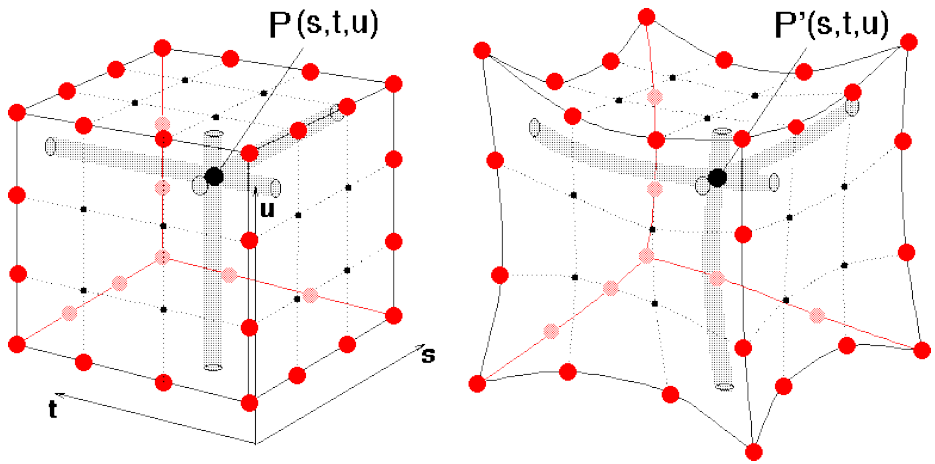
The lattice defines a Bezier volume:

$$\boldsymbol{P}(s,t,u) = \sum_{ijk} \boldsymbol{p}_{ijk} B_i(s) B_j(t) B_k(u)$$

For each object point v, determine its lattice coordinates (s,t,u)

Alter the lattice points $\boldsymbol{P}_{ijk}$

New position of v is evaluated as $\boldsymbol{P}(s,t,u)$



Free Form Deformation (FFD)

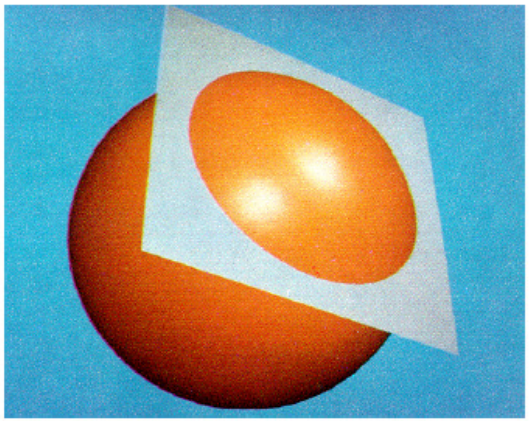


Fig. 8 Sphere and Plane

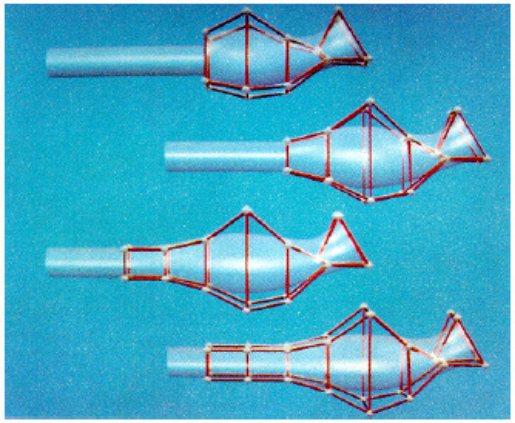


Fig. 11 Local C^1 Control Points

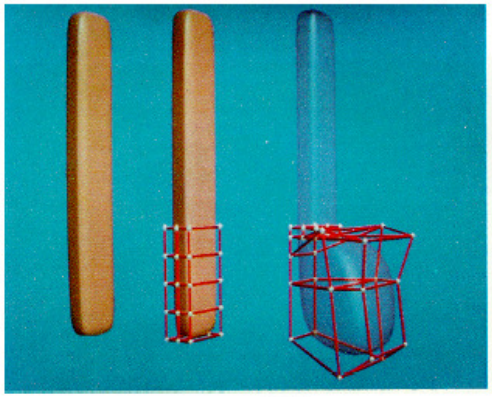
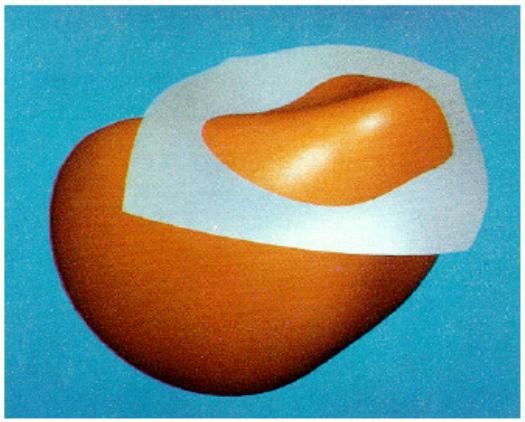
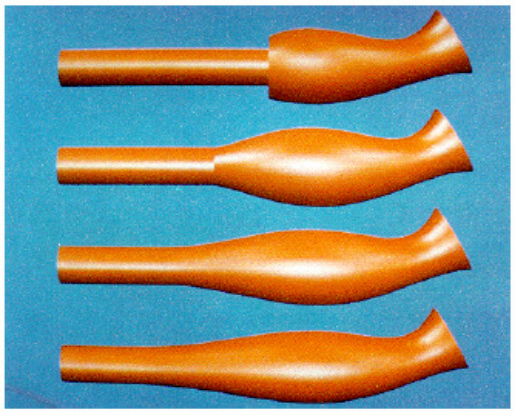
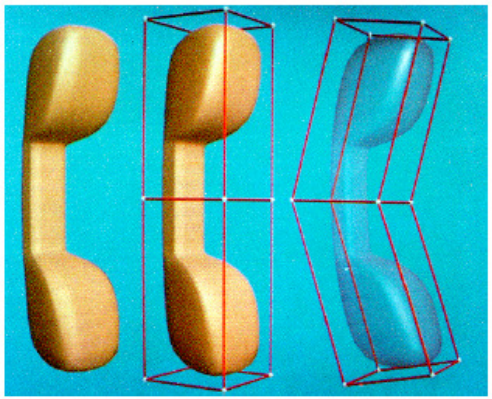


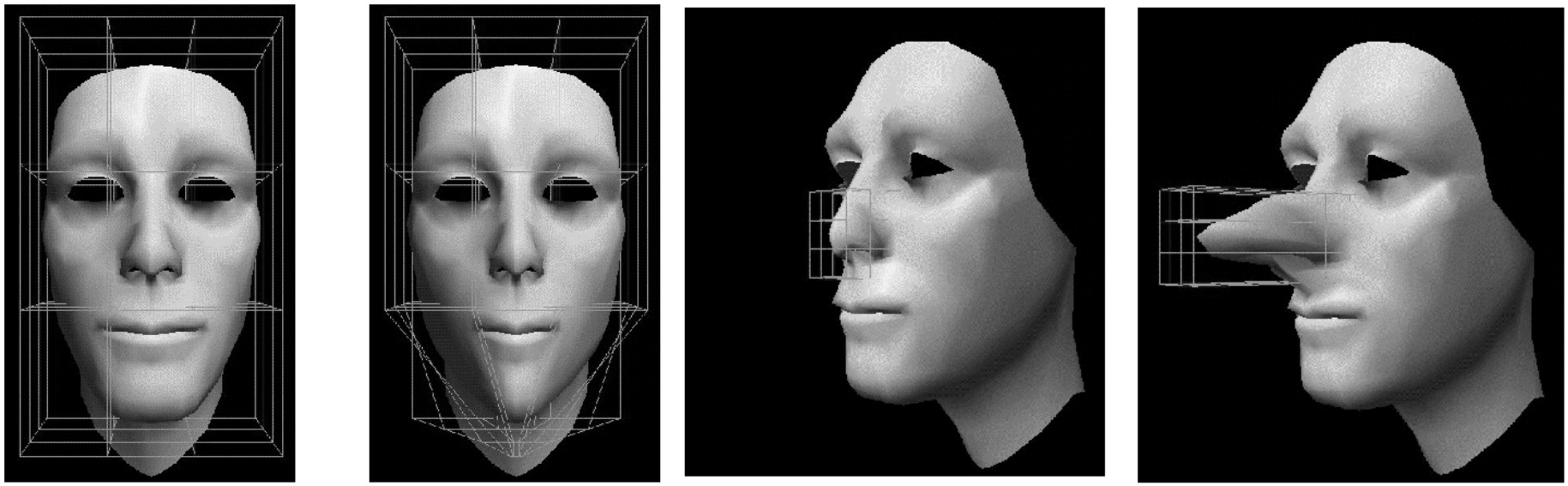
Fig. 14 Local FFD







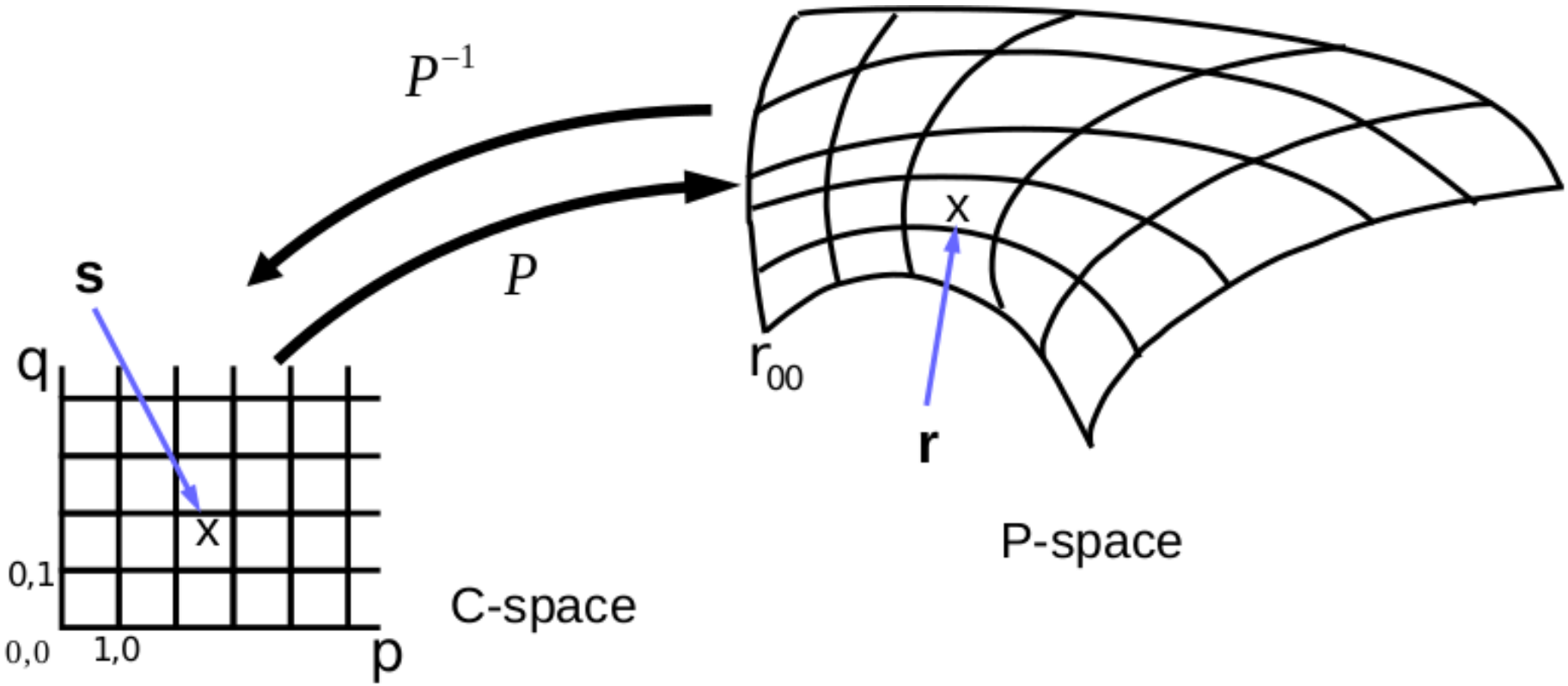
Free Form Deformation (FFD)



How to invert the Bezier Volume?

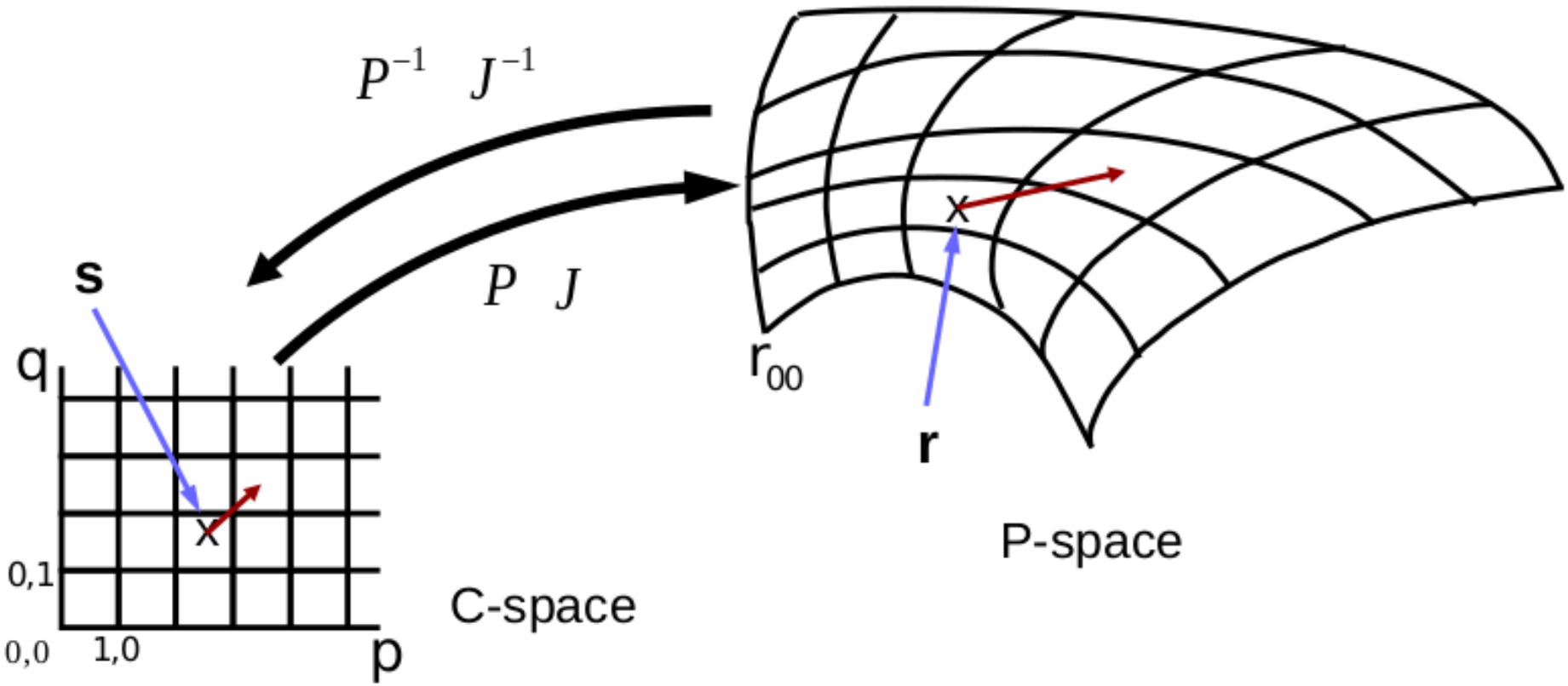
$$\tilde{g} = g \circ \boldsymbol{P}^{-1}$$

- C-space (computation space) vs. P-space (physical space)



Transformation of

- Points by $\boldsymbol{P}$
- Vectors by the Jacobian J



Transformation of points

- From C-space to P-space : $r = \boldsymbol{P}(s)$
- From P-space to C-space : $s = \boldsymbol{P}^{-1}(r)$

Transformation of vectors

- From C-space to P-space : $v = J \cdot u$
- From P-space to C-space : $u = J^{-1} \cdot v$
- J is the Jacobi matrix:

$$J = \begin{pmatrix} \frac{\partial \boldsymbol{P}_x}{\partial p} & \frac{\partial \boldsymbol{P}_x}{\partial q} \\ \frac{\partial \boldsymbol{P}_y}{\partial p} & \frac{\partial \boldsymbol{P}_y}{\partial q} \end{pmatrix}$$

($2D$ case)

$$J = \begin{pmatrix} \frac{\partial \boldsymbol{P}_x}{\partial p} & \frac{\partial \boldsymbol{P}_x}{\partial q} & \frac{\partial \boldsymbol{P}_x}{\partial r} \\ \frac{\partial \boldsymbol{P}_y}{\partial p} & \frac{\partial \boldsymbol{P}_y}{\partial q} & \frac{\partial \boldsymbol{P}_y}{\partial r} \\ \frac{\partial \boldsymbol{P}_z}{\partial p} & \frac{\partial \boldsymbol{P}_z}{\partial q} & \frac{\partial \boldsymbol{P}_z}{\partial r} \end{pmatrix}$$

($3D$ case)

Data: Given target position r_0 in P-space, required accuracy ε in C-space
Result: Guess of inverse point $s \approx \boldsymbol{P}^{-1}(r_0)$ in C-space
repeat
  $r = \boldsymbol{P}(s)$;
  $\Delta r = r_0 - r$;
  $\Delta s = J^{-1}(r)\Delta r$;
  if $|\Delta s| < \varepsilon$ **then**
    exit;
  end
  $s = s + \Delta s$;
  if s *outside C-space domain* **then**
    Guess new inverse point $s \approx \boldsymbol{P}^{-1}(r_0)$;
  end
until k *times*;

Algorithm 1: Newton Algorithm

Advantages:

- Simple definition of volumes (closed objects)
- Efficient collision detection
  - inside/outside test
  - distance to object
- Efficient ray-object intersections
  - important for Ray Tracing

Disadvantages:

- No direct parameterization/sampling of surface possible
- Memory overhead (whole volume instead of surface)
- Does not lend itself to (real-time) rendering

Problem: How to get an explicit representation of the isosurface?

Solution: Marching Cubes algorithm[7]

- Approximates the surface by a triangle mesh
- Surface is found by linear interpolation along cell edges
- Uses gradients as the normals of the isosurface
- Efficient computation by means of lookup tables

→ THE standard geometry-based isosurface extraction algorithm

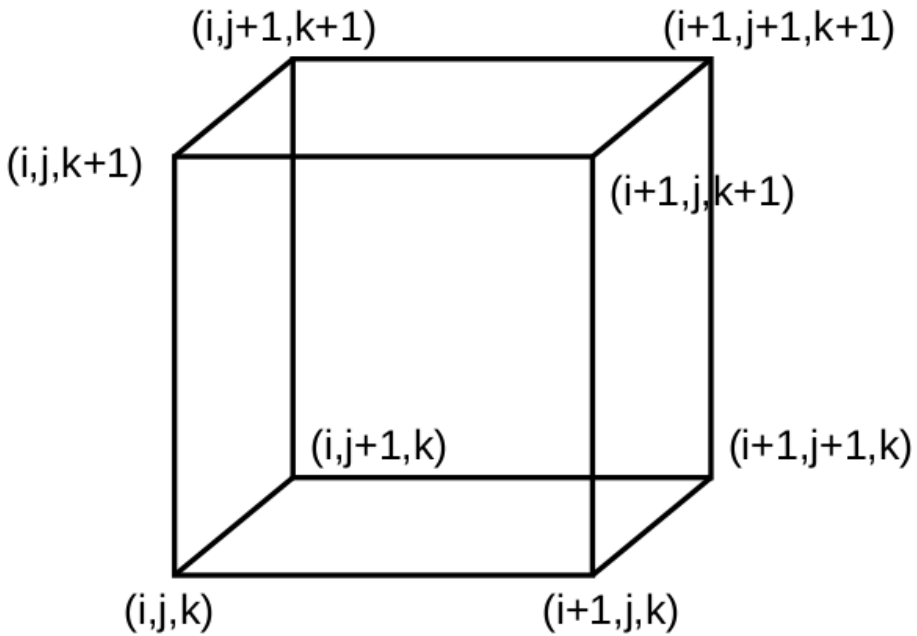
[7] W. E. Lorensen and H. E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *ACM siggraph computer graphics*. Vol. 21. 4. ACM. 1987, pp. 163–169.

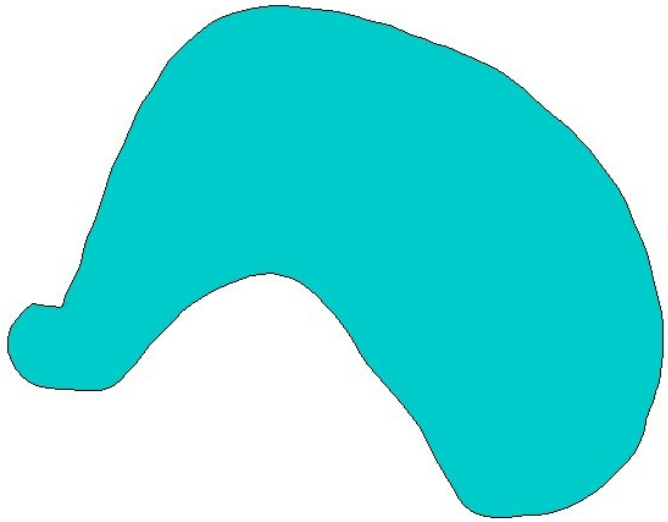
The core Marching Cubes algorithm:

- Cell consists of 4(8) pixel (voxel) values:

$$(i + [0, 1], j + [0, 1], k + [0, 1])$$

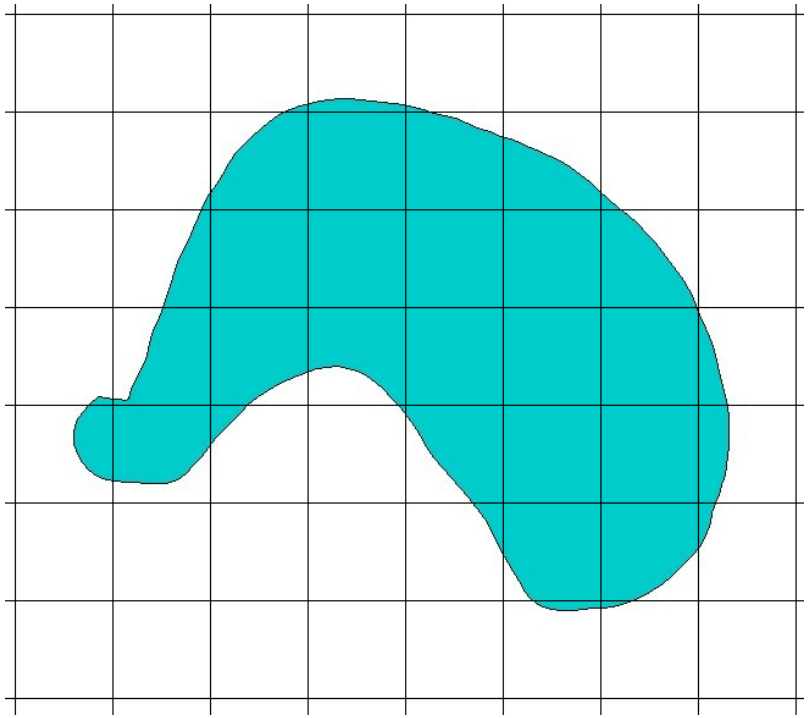
1. Consider a cell
2. Classify each vertex as inside or outside
3. Build an index
4. Get edge list from table[index]
5. Interpolate the edge location
6. Compute normals
7. Consider ambiguous cases
8. Go to next cell





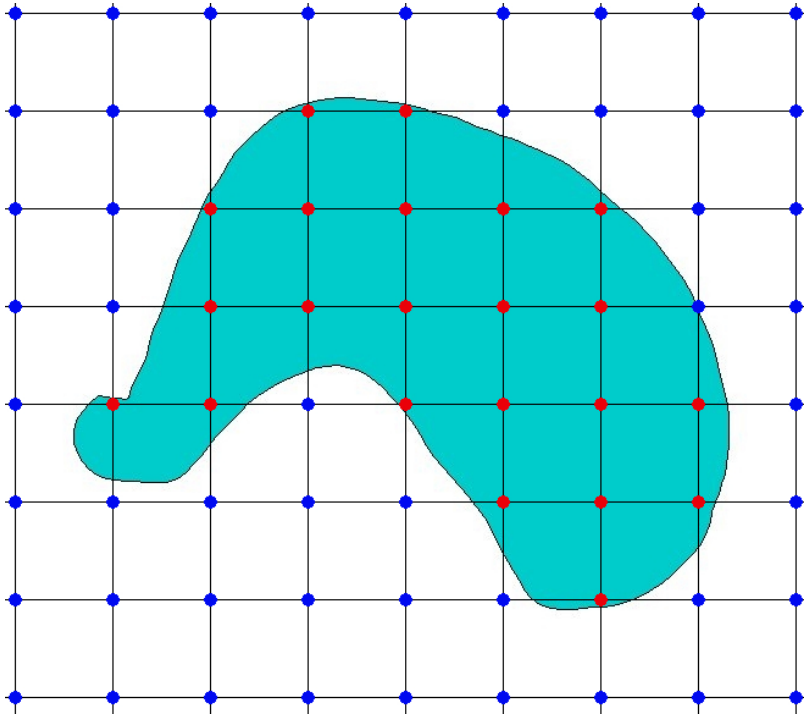
Example: In order to explain the Marching Cubes algorithm, let's assume we are given an object like this one

$\rightarrow$ For simplicity, the example is chosen in 2D since the generalization to 3D is straightforward



Step 0: Marching Cubes requires a signed distance field as an input

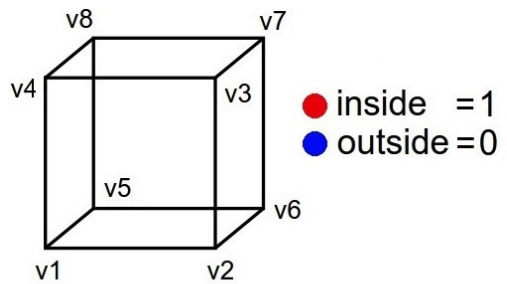
$\rightarrow$ Subdivide the space into squares (cubes) and compute signed distance values d for each grid point

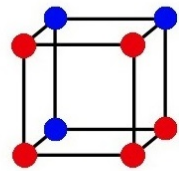


Step 1: For each grid point $(i, j, k) \in \mathbb{Z}^3$ decide whether it is inside or outside the object

$$d(i, j, k) \leq 0 \quad , \quad d(i, j, k) > 0$$

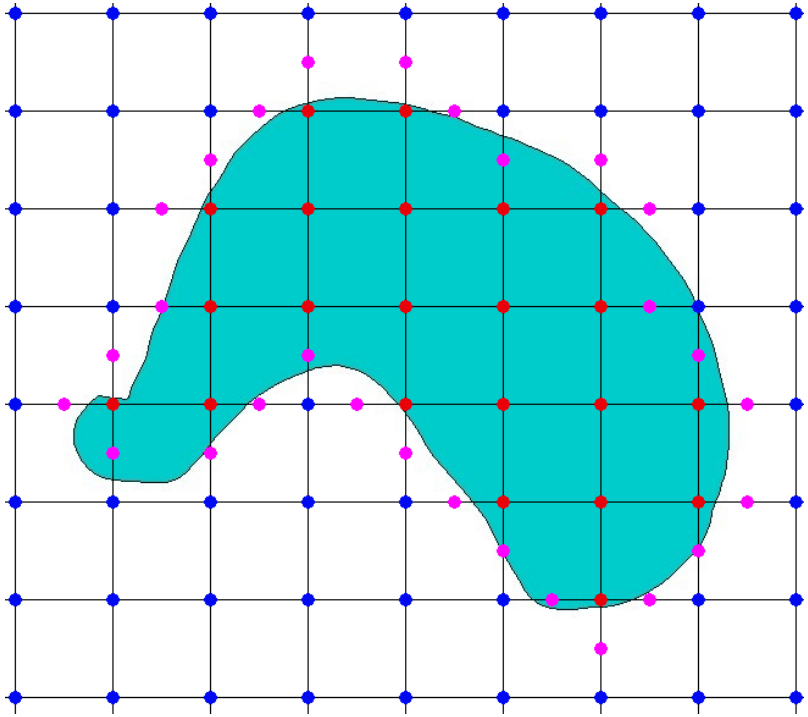
and calculate an index using its neighbors $i + [0, 1]$, $j + [0, 1]$, $k + [0, 1]$





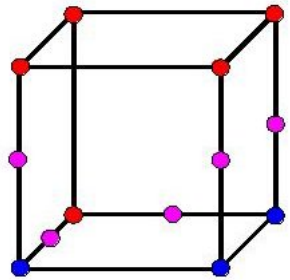
11110100

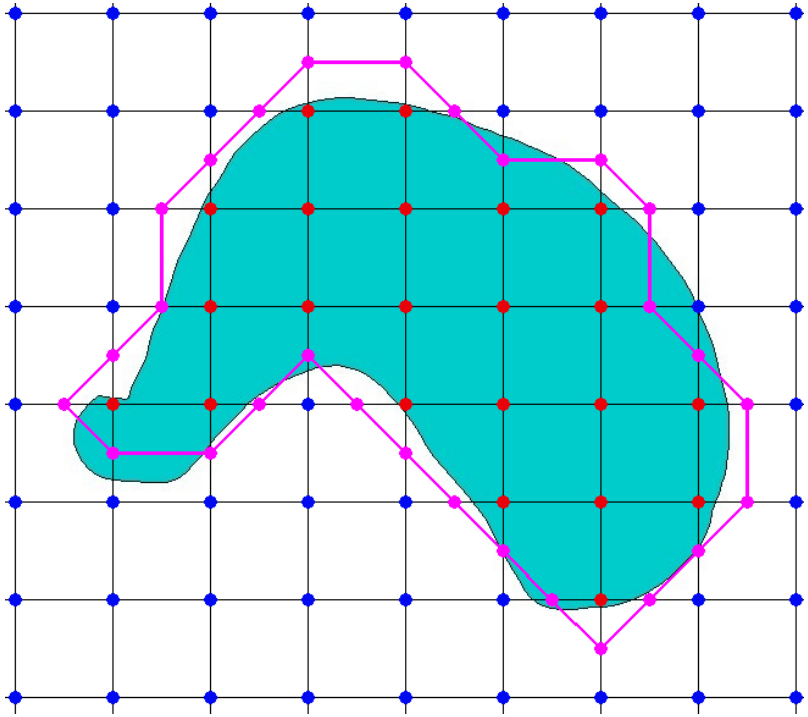
Index: | v1 | v2 | v3 | v4 | v5 | v6 | v7 | v8 |



Step 2: Compute the mesh vertices as the midpoints by using indices from a lookup table

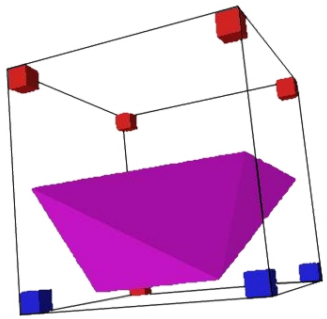
$\rightarrow$ Lookup table stores the locations of the corners having different labels based on the index





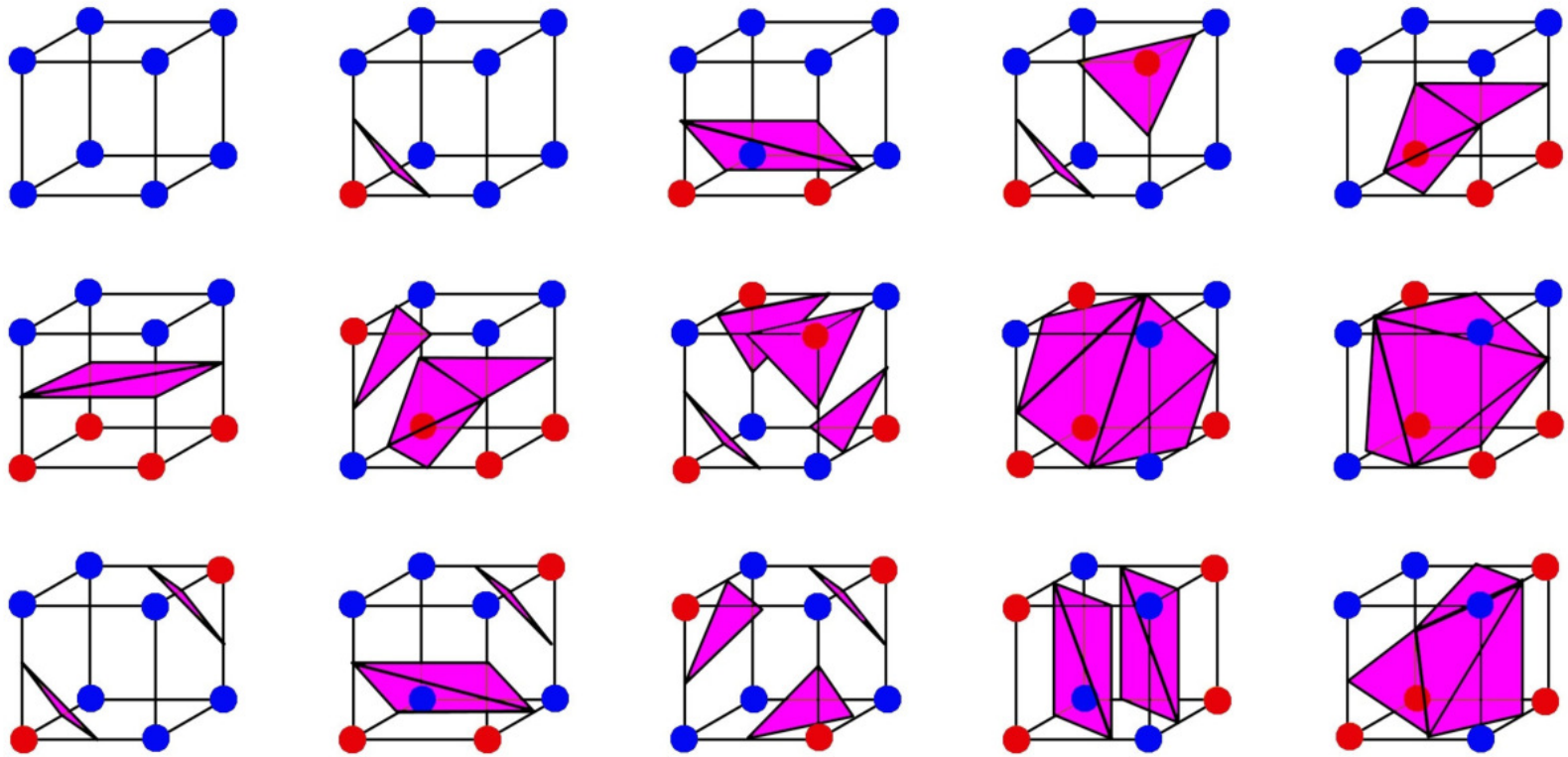
Step 3: Connect the mesh vertices through lines (faces) by using indices from a lookup table

$\rightarrow$ Lookup table stores the triangulation of the mesh vertices



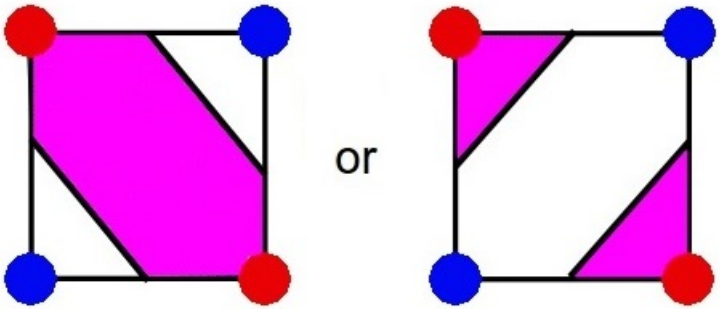
Step 3: Connect the mesh vertices through lines (faces) by using indices from a lookup table

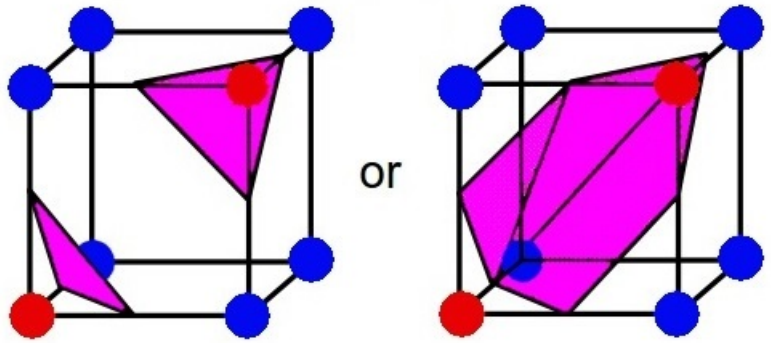
→ Lookup table stores the triangulation of the mesh vertices
→ All 256 cases can be derived from 15 base cases due to symmetries



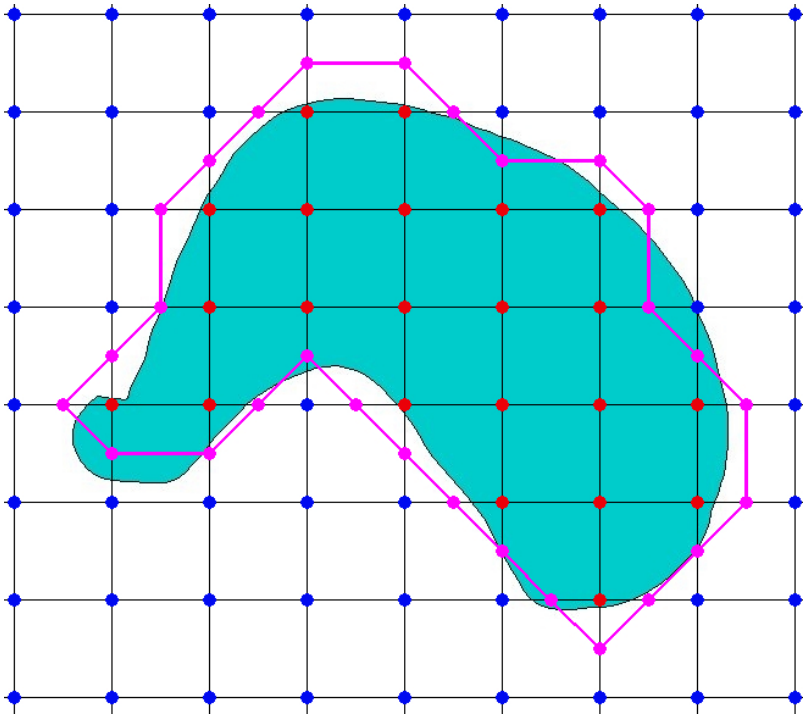
Problem: Ambiguous cases

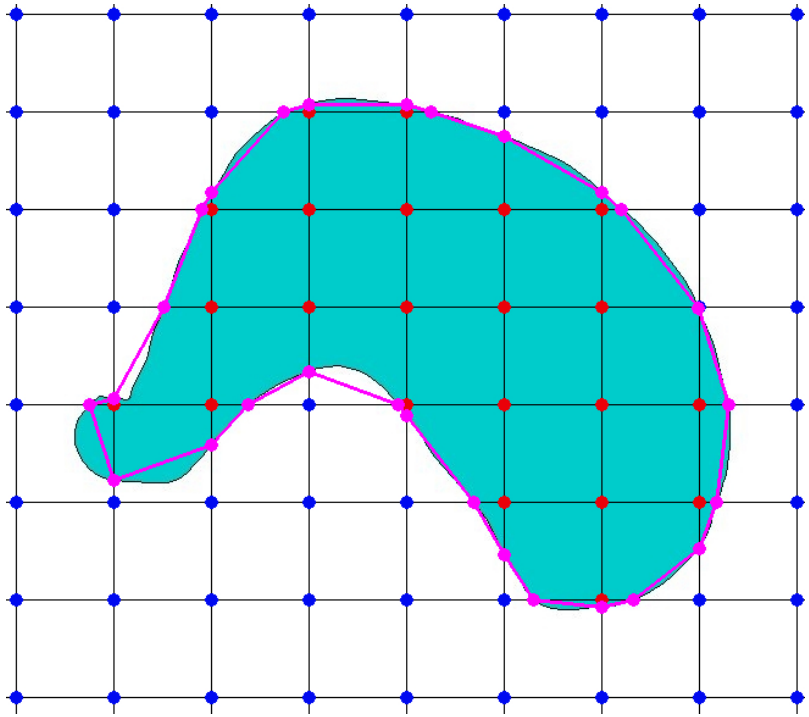
- Adjacent vertices: different states
- Diagonal vertices: same state





Solution: To avoid holes in the reconstruction, decide for one solution **consistently** for all possible cases!





Idea: In the simplest version of Marching Cubes, only midpoints are computed. However since the implicit function also indicates the signed distances to the surface, we can compute the mesh vertices more precisely by linear interpolation:

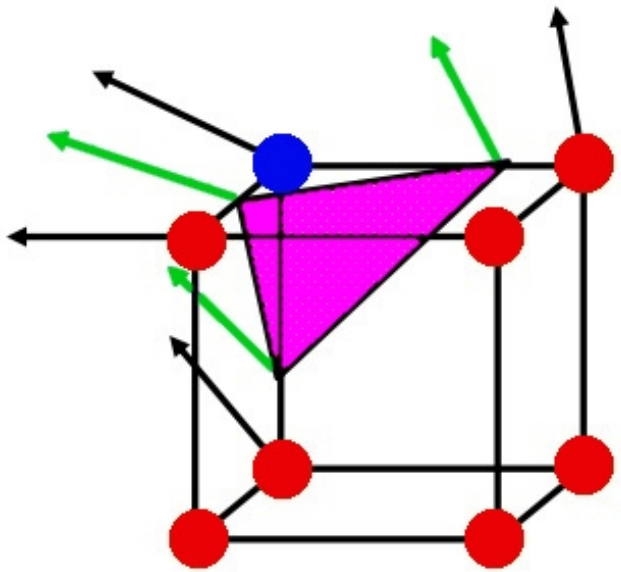
$$\boldsymbol{v}_{mesh} = \boldsymbol{v}_1 + \frac{0 - d_{\boldsymbol{v}_1}}{d_{\boldsymbol{v}_2} - d_{\boldsymbol{v}_1}} \cdot (\boldsymbol{v}_2 - \boldsymbol{v}_1)$$

Optional Step: Calculate the normal at each triangle vertex

- Calculate a normal at each cube vertex using the signed distance function $d(i, j, k)$

$$\nabla d(i,j,k) \approx \begin{pmatrix} d(i+1,j,k) - d(i-1,j,k) \\ d(i,j+1,k) - d(i,j-1,k) \\ d(i,j,k+1) - d(i,j,k-1) \end{pmatrix} \quad , \quad \boldsymbol{n} = \frac{\nabla d(i,j,k)}{\|\nabla d(i,j,k)\|}$$

- Use linear interpolation to compute the polygon vertex normal (of the isosurface)



Theorem (Marching Cubes)

If the ambiguous cases are handled consistently, then the Marching Cubes algorithm will output a triangle mesh without any holes that approximates the implicitly defined surface of the volume:

$$\mathcal{I} = \{g^{-1}(0)\} = \{\boldsymbol{p} \in \mathbb{R}^3 \mid g(\boldsymbol{p}) = 0\}$$

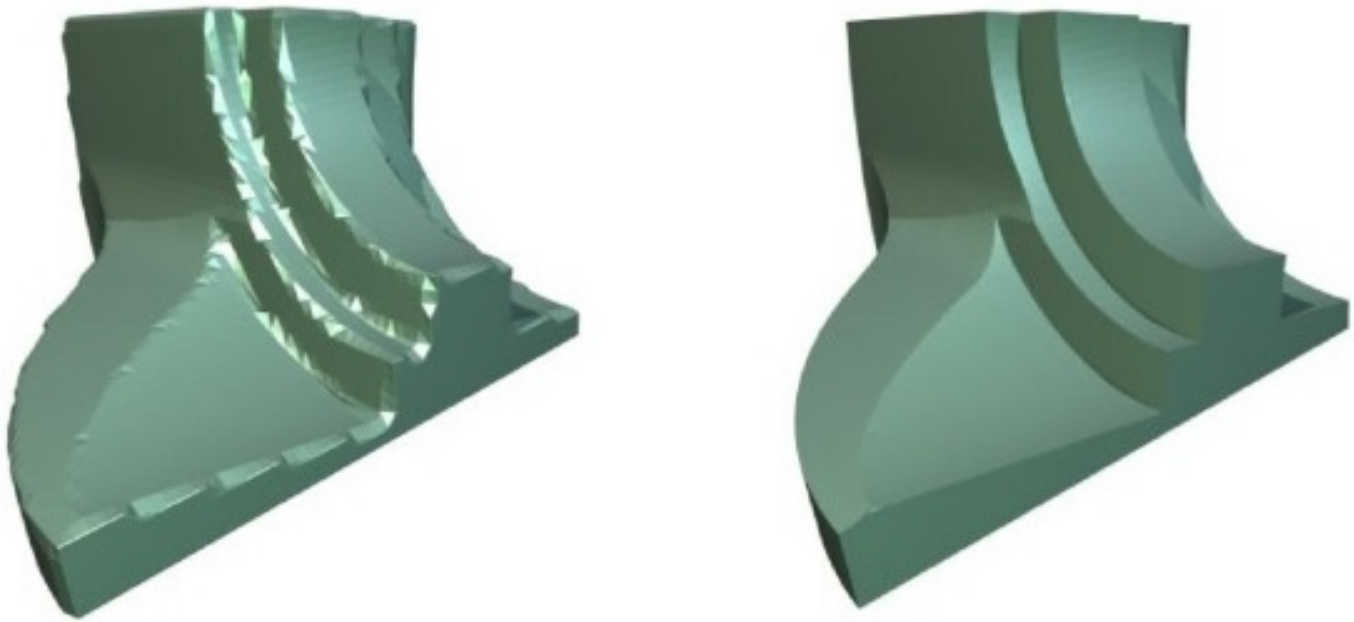
Proof.

Marching Cubes assumes a (sampled) signed distance field g. Since our surface is assumed to be smooth, this field is continuously differentiable.
According to the Implicit Function Theorem, g has non-zero gradients everywhere at the surface.
Therefore the extracted approximation by the Marching Cubes algorithm contains no holes. □

$\rightarrow$ Marching Cubes creates discretizations of 2-manifolds.

Limitations: Sharp features are smoothed out in the original Marching Cubes

Improvement: Model features in the volume[8]



[8] L. P. Kobbelt et al. "Feature sensitive surface extraction from volume data". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, pp. 57–66.

Given: Isosurface

$$\mathcal{S}(q) = \{\boldsymbol{p} \in \Omega \subset \mathbb{R}^3 \mid f(\boldsymbol{p}) = q\} \quad , \quad f\colon \mathbb{R}^3 \to \mathbb{R}$$

Properties:

- The defining function f is not unique
- One possible and very common choice is the signed distance function:

  $$f(\boldsymbol{p}) := \pm d(\boldsymbol{p}, \mathcal{S})$$

  with distance positive outside the volume and negative in the interior
- Efficient representation of the distance function f using a regular grid $\{\boldsymbol{G}(i,j,k)\}$ with step width h

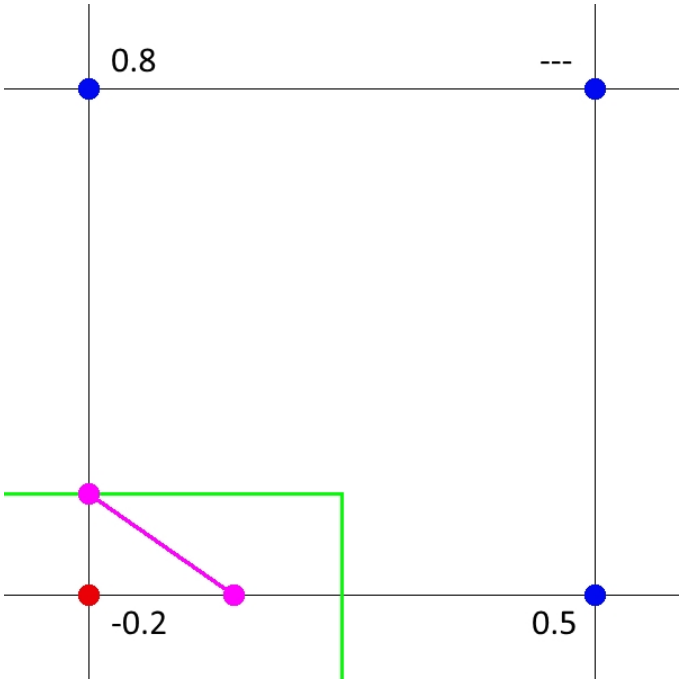
  $$\boldsymbol{G}(i,j,k) = (ih, jh, kh)^T$$

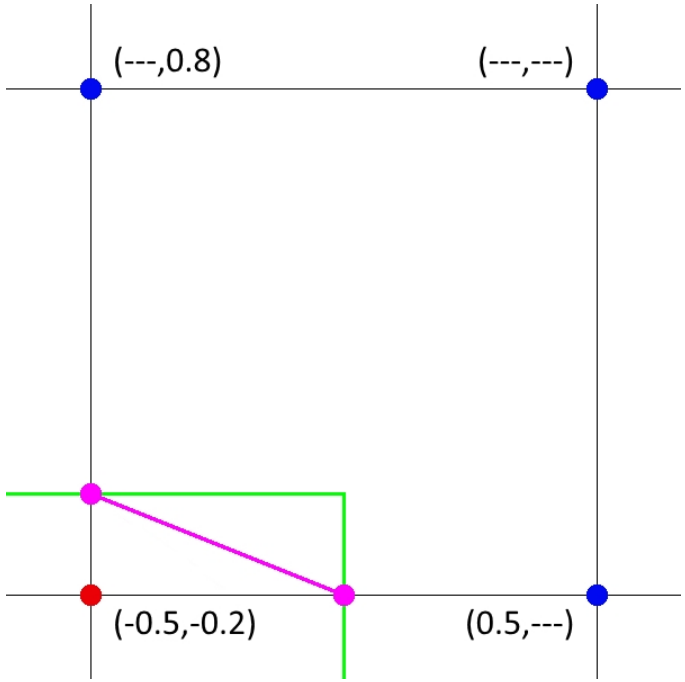
  where f is sampled according to the grid

  $$d(i,j,k,\mathcal{S}) = \pm f(\boldsymbol{G}(i,j,k)) = \pm f(ih, jh, kh)$$

  and tri-linear interpolated in-between

Problem: True distance function depends on the direction





Standard: Only smallest distance to surface stored

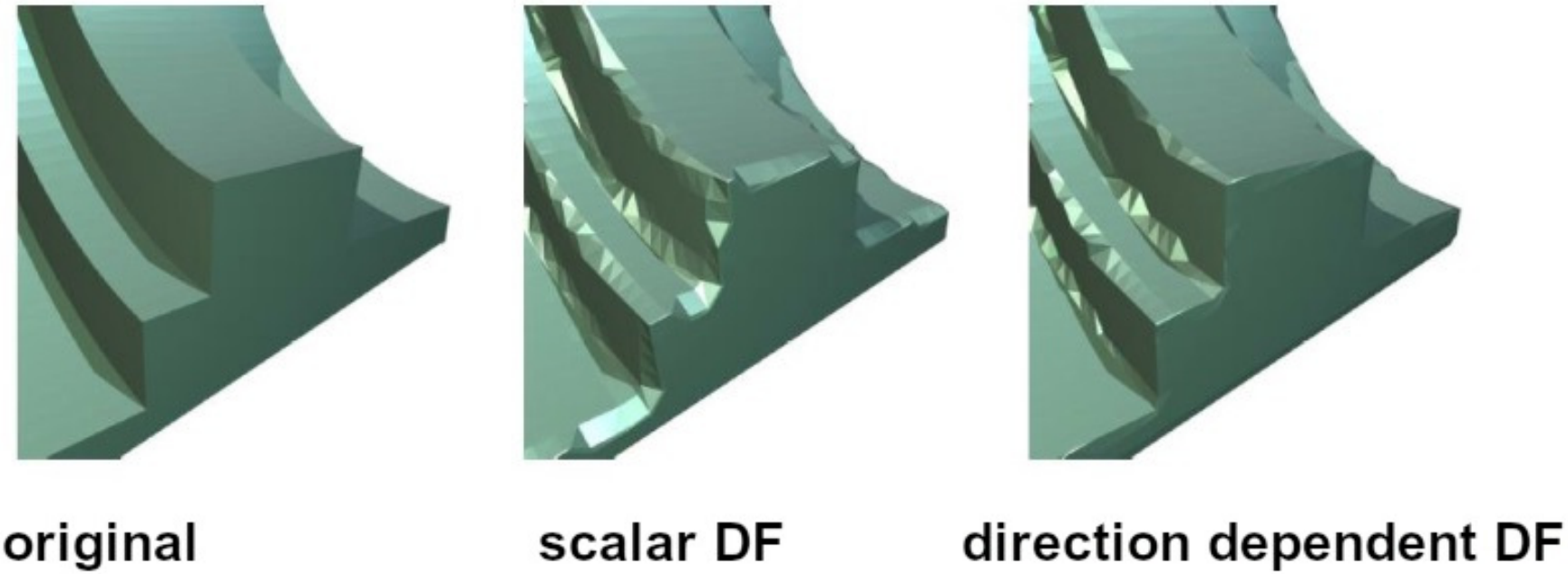
$$d(i,j,k,\mathcal{S}) = \pm f(ih,jh,kh)$$

$\rightarrow$ Close to a feature not a good choice

Improvement: Direction dependent distances stored

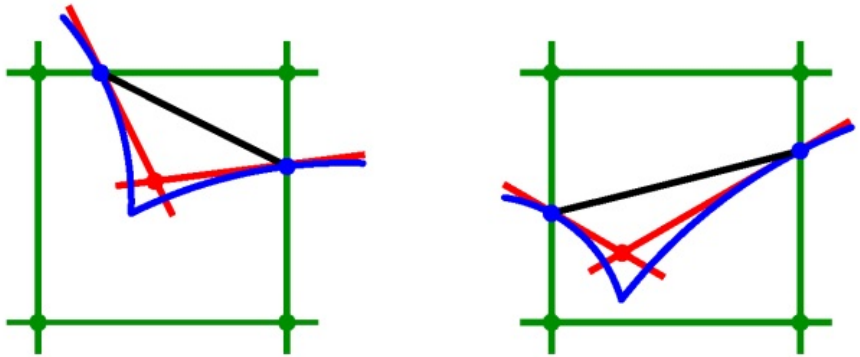
$$d(i,j,k,\mathcal{S}) = \begin{pmatrix} \pm f_x(ih,jh,kh) \\ \pm f_y(ih,jh,kh) \\ \pm f_z(ih,jh,kh) \end{pmatrix}$$

Results: Results with direction dependent distance field

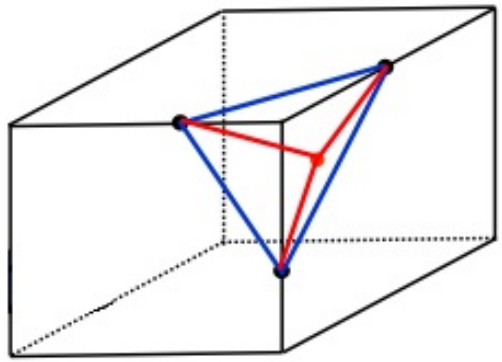


Although there is an improvement, features are still not reconstructed well

Idea: Using normal or tangent information in the intersection points on the grid edges allows the incorporation of additional feature points in the surface

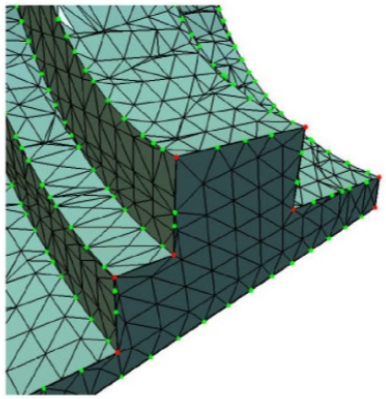


- Consider each cell independent of its neighbors. Perform the interpolation using direction dependent distance fields
- If there are no features within a cell, use standard MC triangulation
- If there is a feature point, its position is calculated and triangulated into the surface using a triangle fan

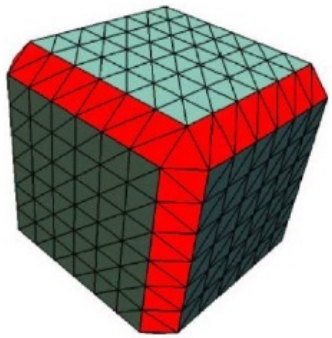


Distinguish between edge and vertex features

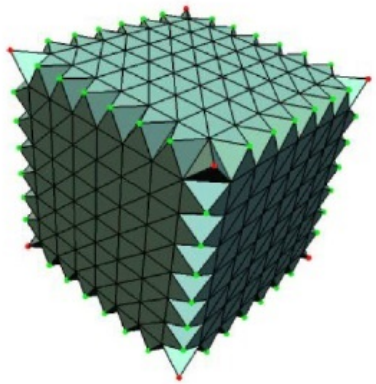
- Red: vertex feature
- Green: edge feature



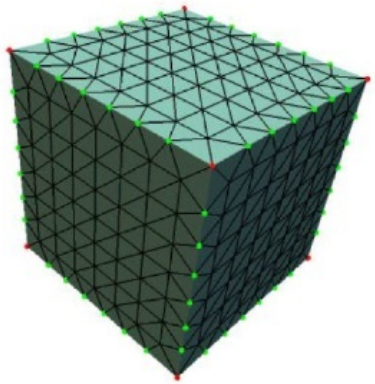
Flip edges between edge features appropriately:



Detection of the feature-cells

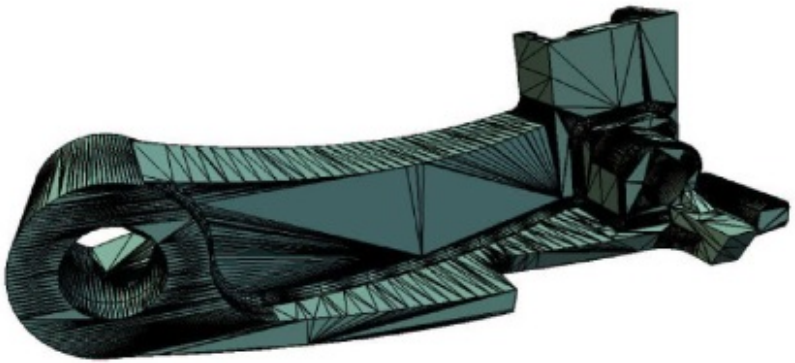


Insertion of one Feature point per cell

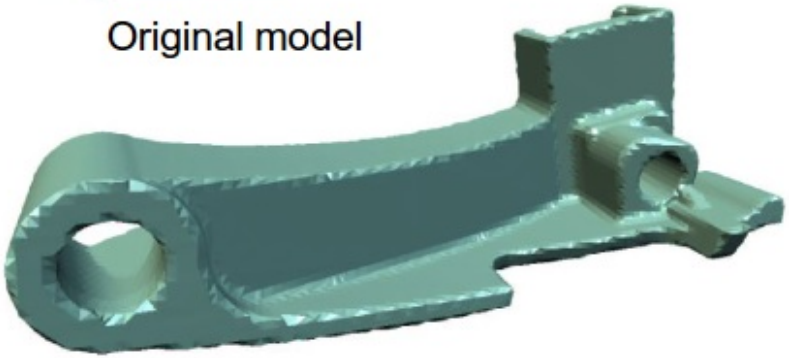


Flip of edges

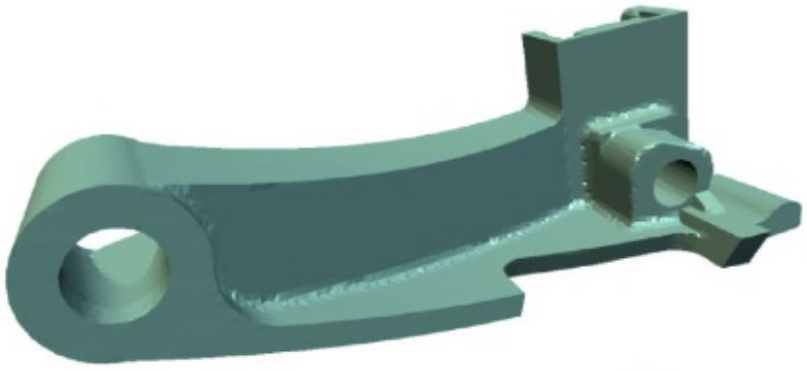
Results:



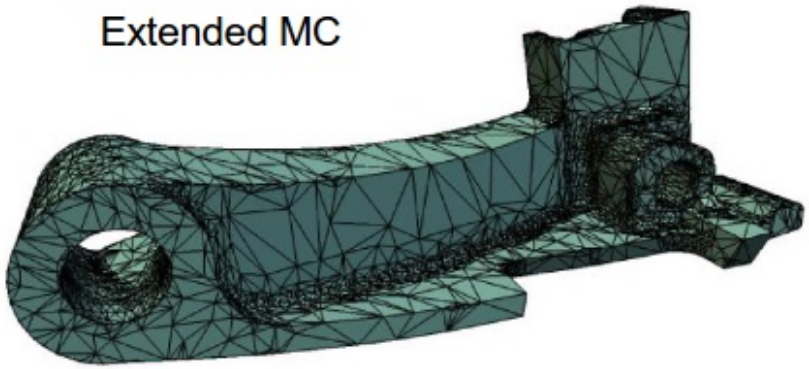
Original model



Standard MC



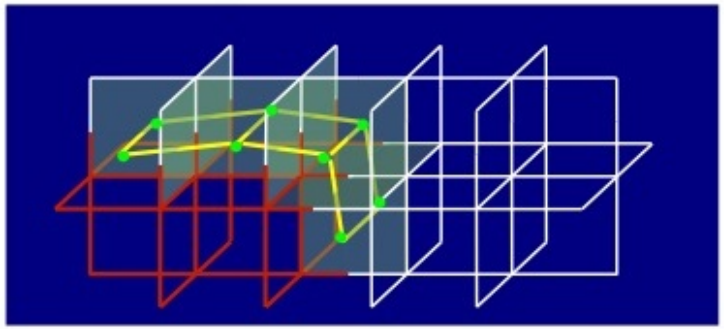
Extended MC

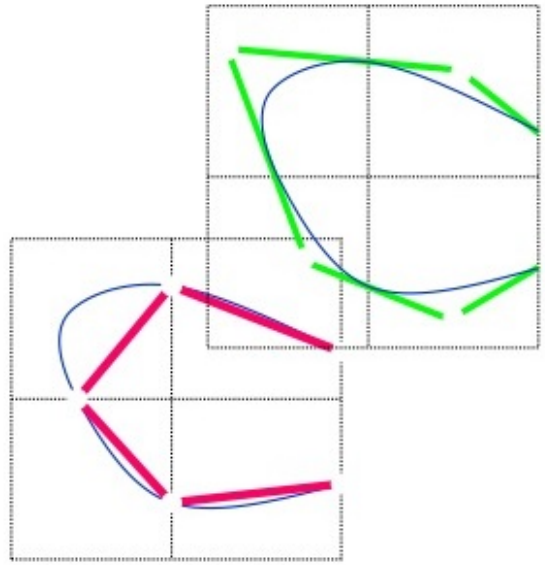


Remeshing with feature line preserving mesh decimation

Implicit to Explicit : Dual Contouring

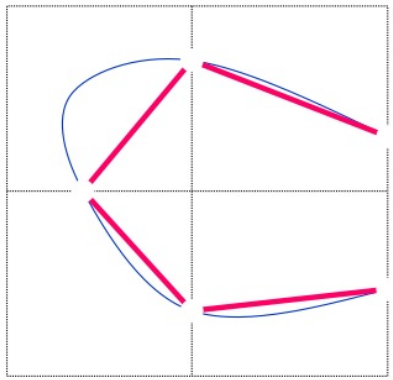
Another technique to overcome the limitations of MC is Dual Contouring[9].



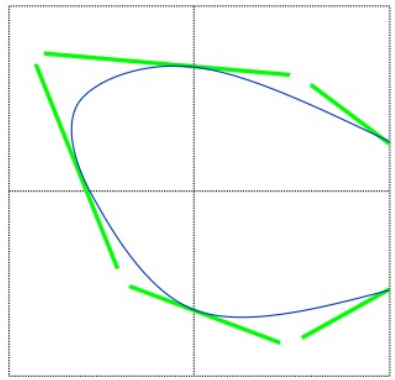


[9] A. Greß and R. Klein. "Efficient representation and extraction of 2-manifold isosurfaces using kd-trees". In: *Graphical Models* 66.6 (2004), pp. 370–397.

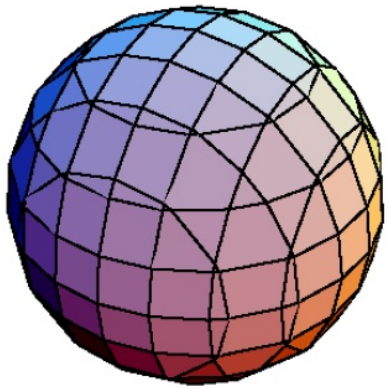
Examples:

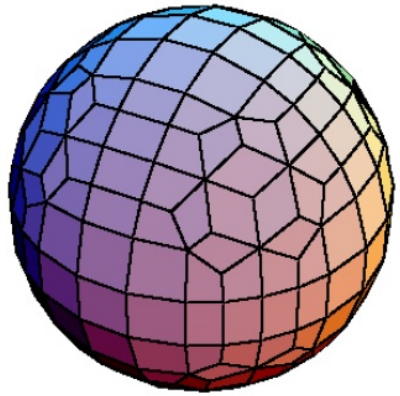


contour



dual contour





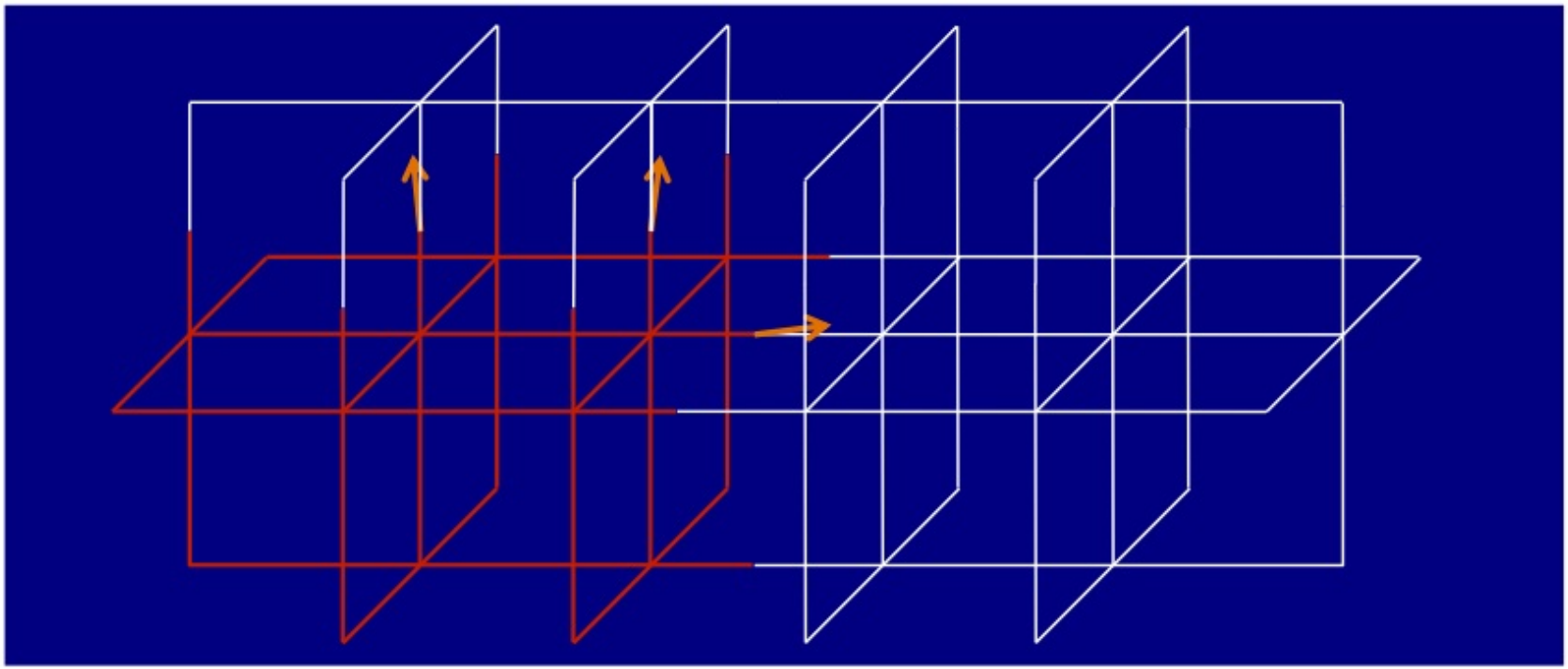
Notice the better aspect ratios of the polygons when using dual contouring.

Input: Uniform grid

- Inside/outside state for all grid points
- Exact intersection points and normals

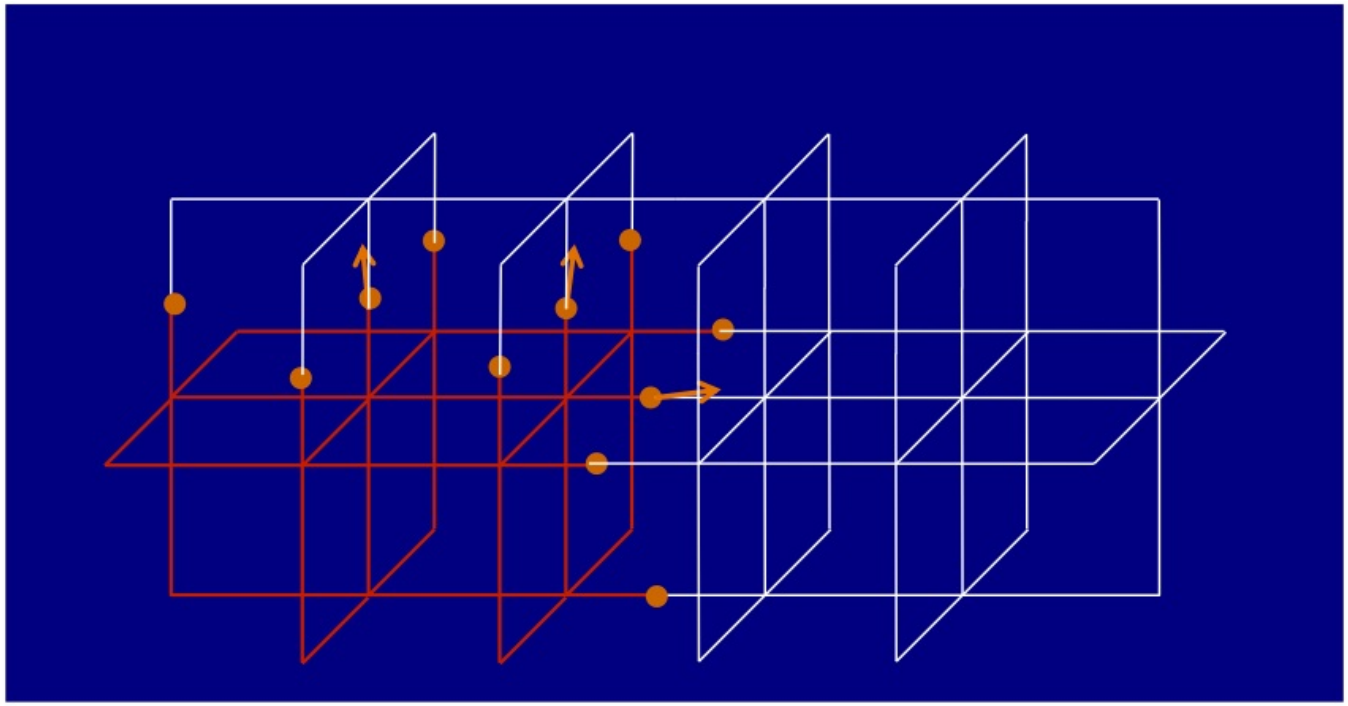
for all grid edges crossed by the isosurface (Hermite data)

Output: Quadrilateral mesh



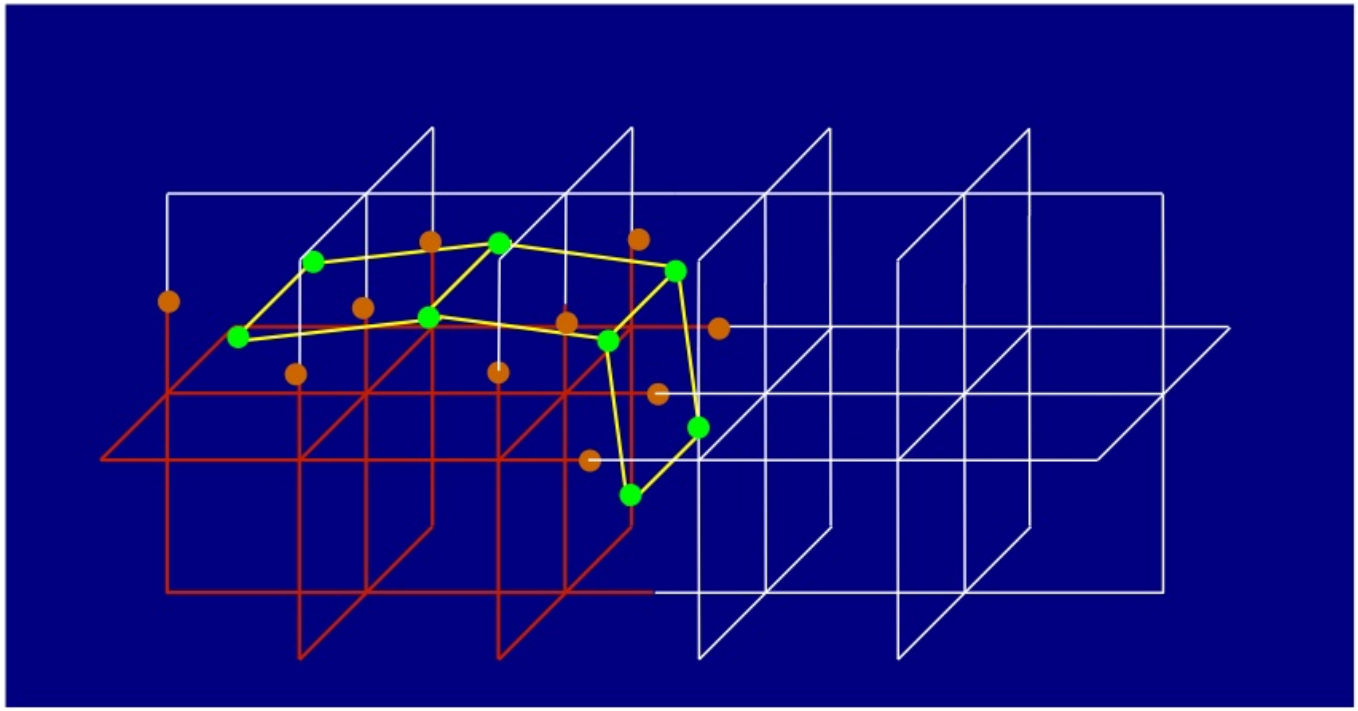
Review: SurfaceNets / Dual Contouring

1. Generate vertices



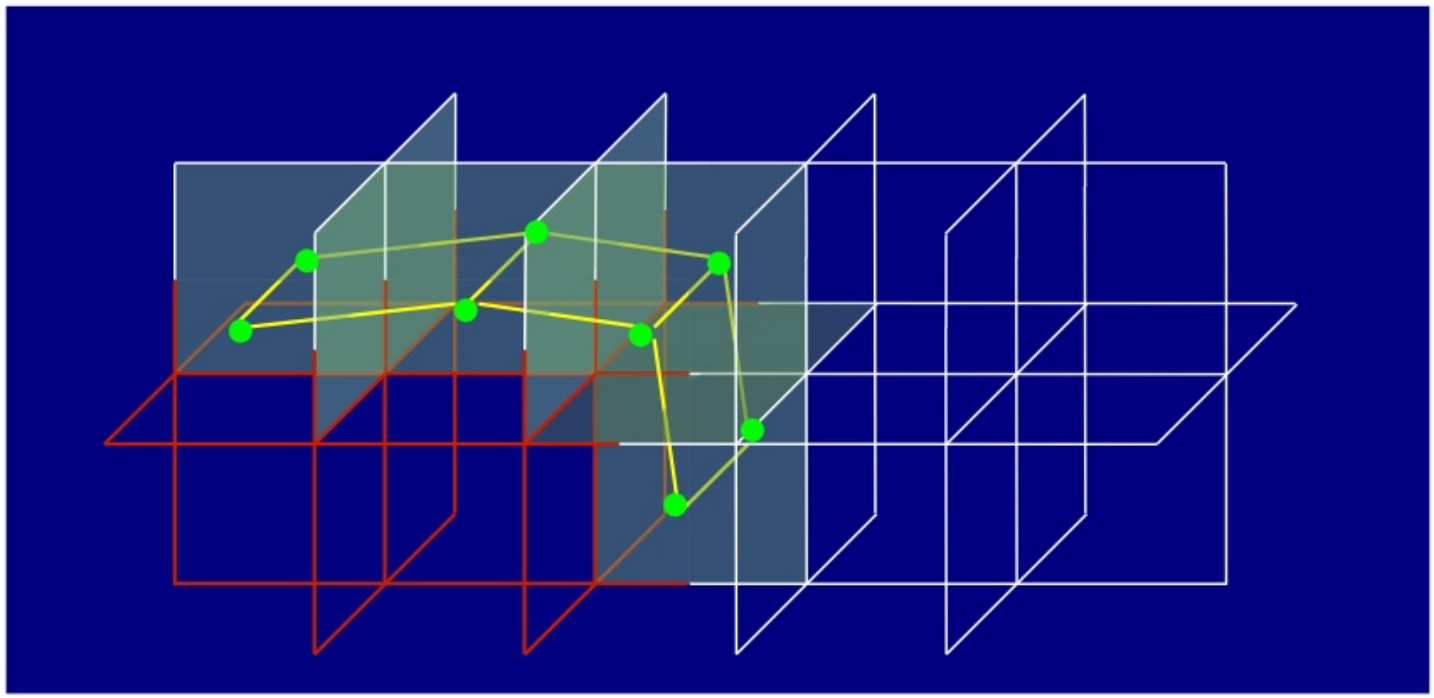
Review: SurfaceNets / Dual Contouring

2. Generate quadrilaterals



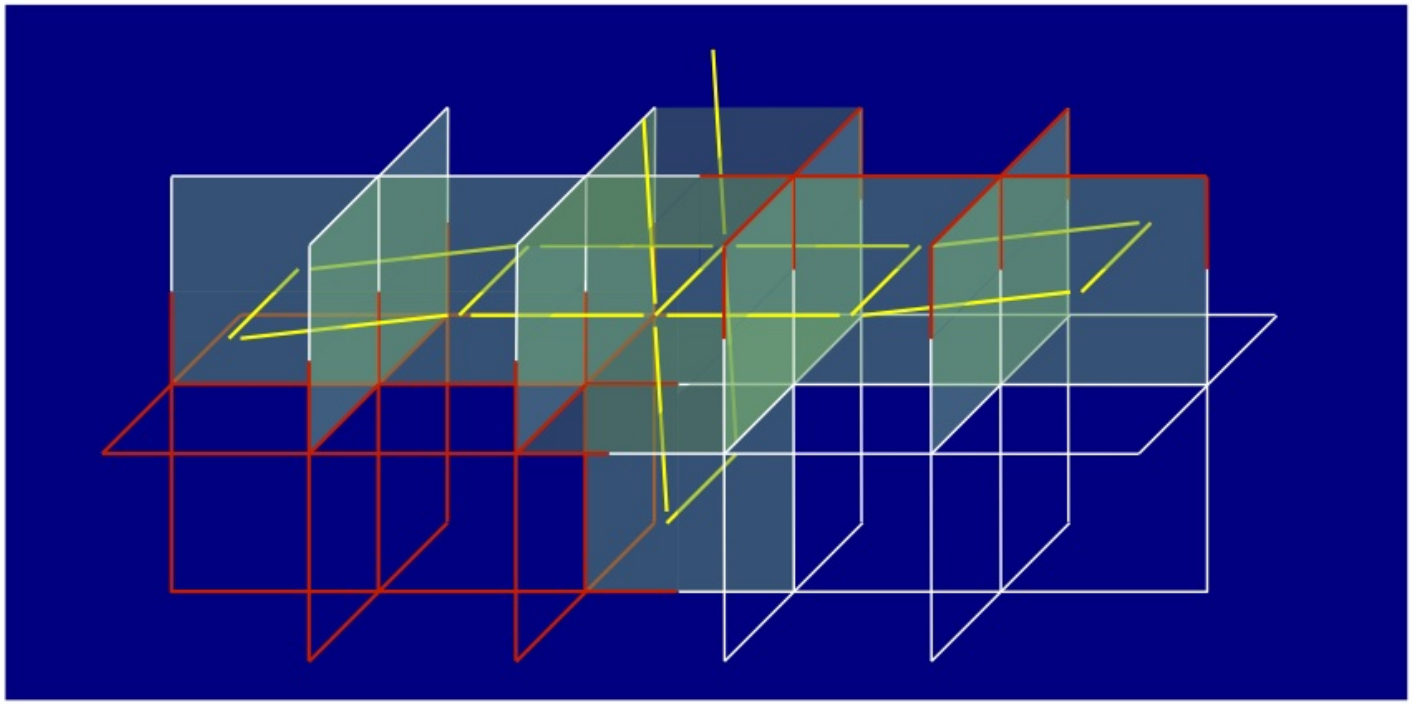
Review: SurfaceNets / Dual Contouring

$\rightarrow$ one edge generated for each grid face crossed by the isosurface



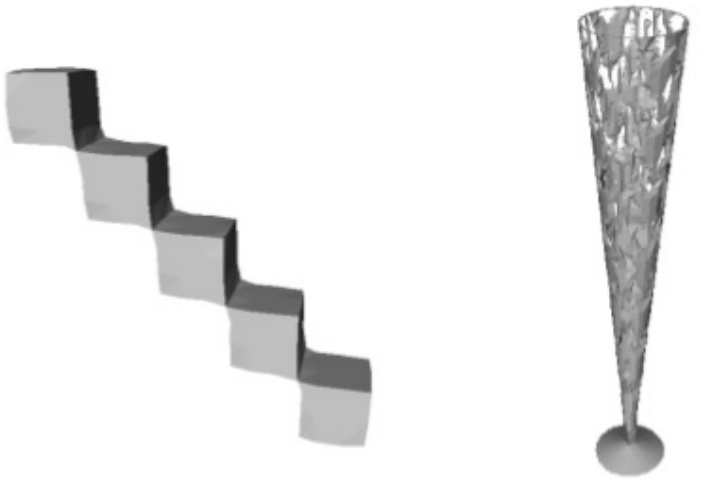
Review: SurfaceNets / Dual Contouring

→ **Problem**: Results in a complex edge if a grid face is crossed by multiple surface components



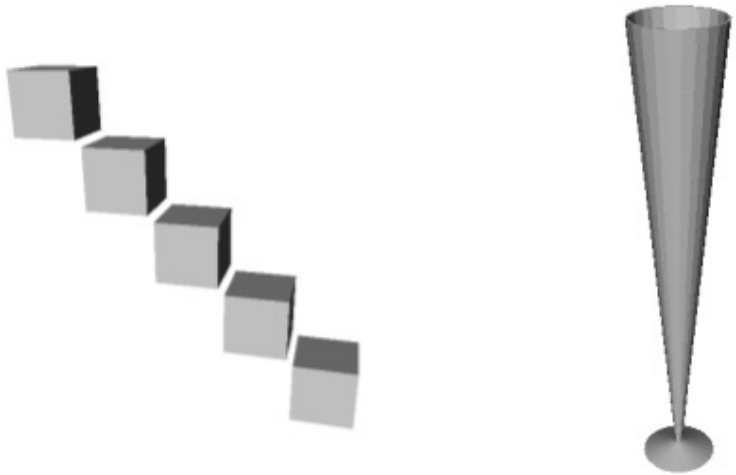
Difficulties inherent in SurfaceNets / Dual Contouring:

- Complex vertices and edges are generated for cells containing multiple surface components
- Prevent further simplification
- No topologically correct reconstruction if a cell edge is crossed by multiple surface components

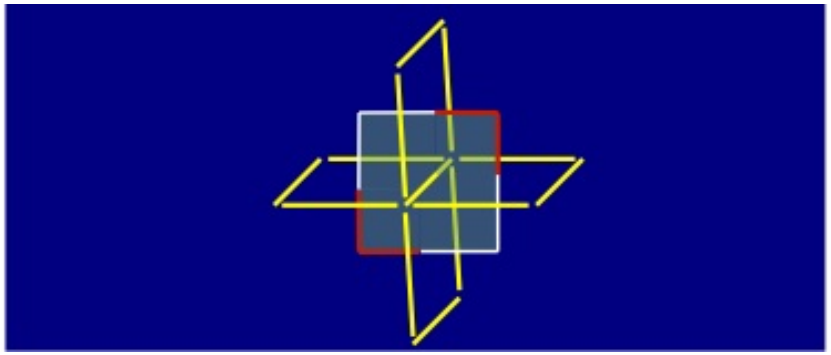


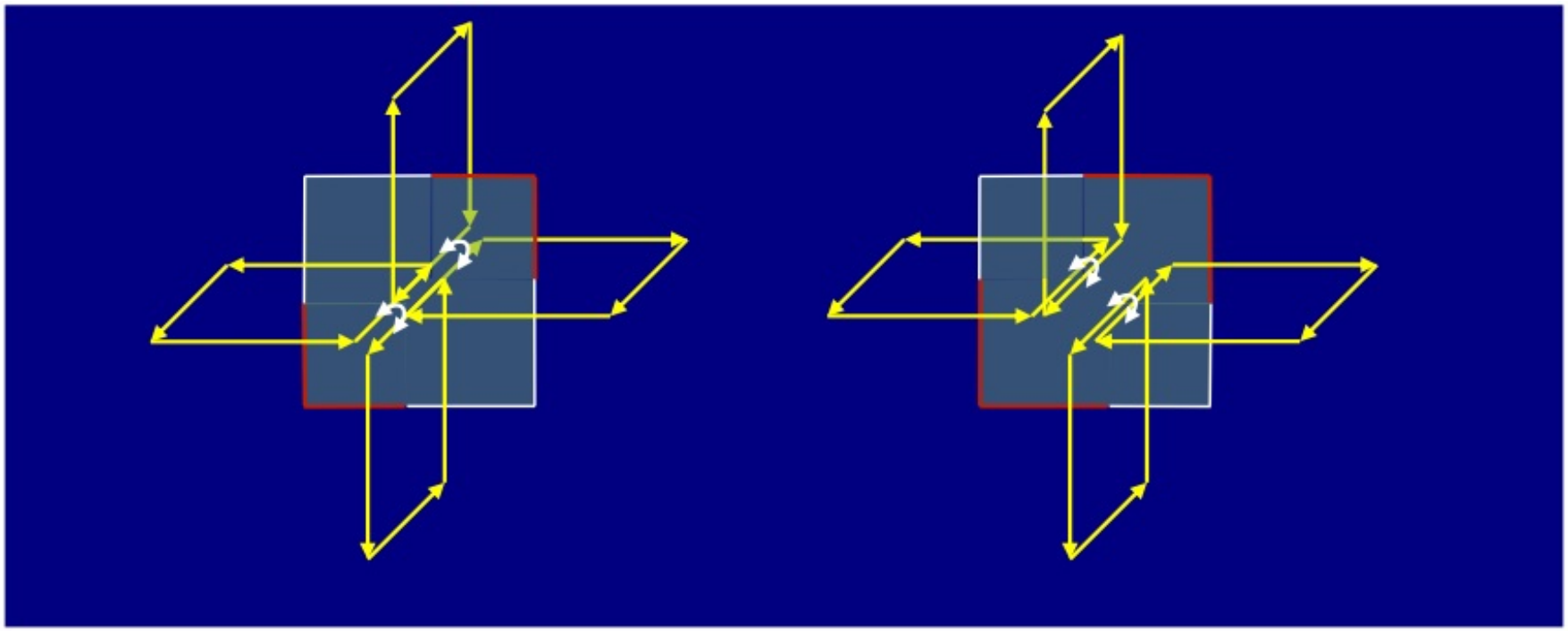
$\rightarrow$ **Isosurface extraction algorithm** (Greß and Klein (2004))

- Always generates 2-manifold meshes
- Improves the approximation of geometry for cells containing multiple surface components



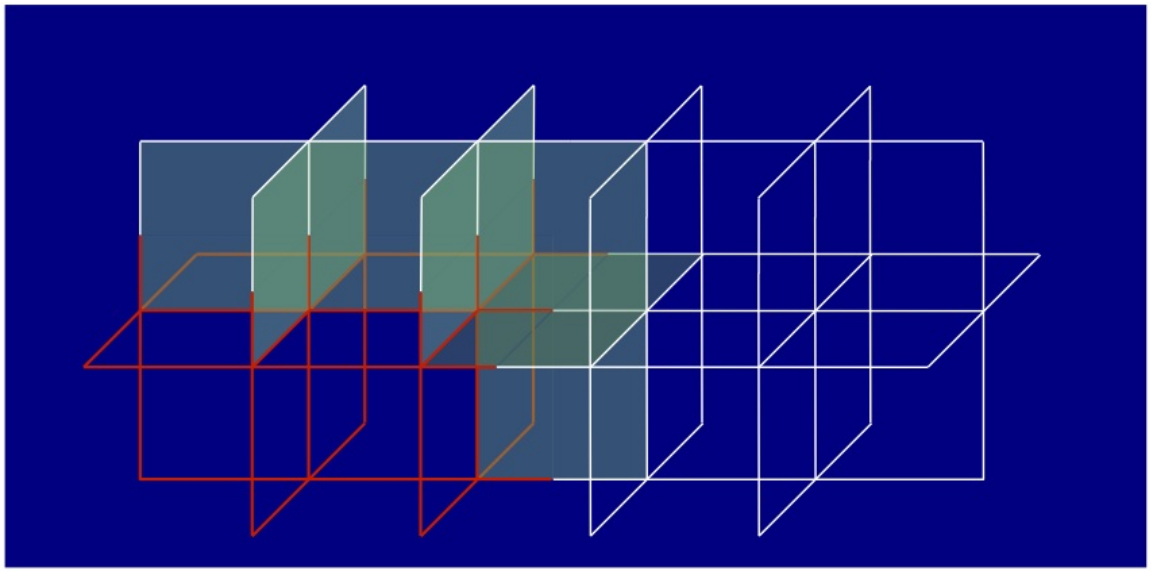
Solution: generate one half-edge per intersection point on the boundary of each grid face





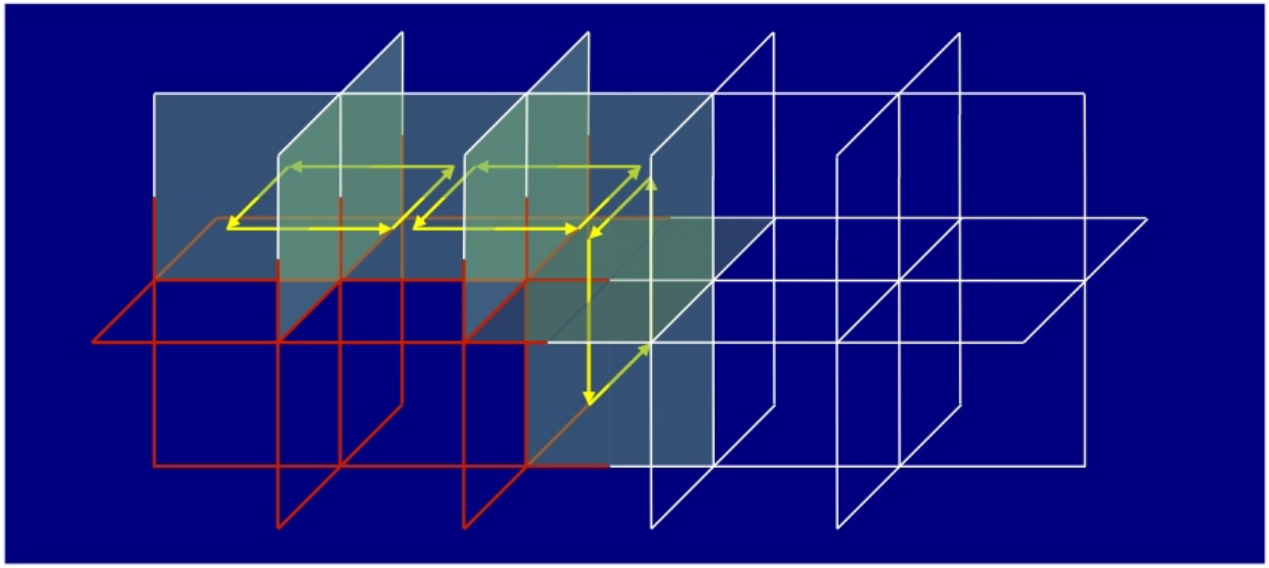
Generate the mesh connectivity:

- Iterate over all grid faces crossed by the isosurface



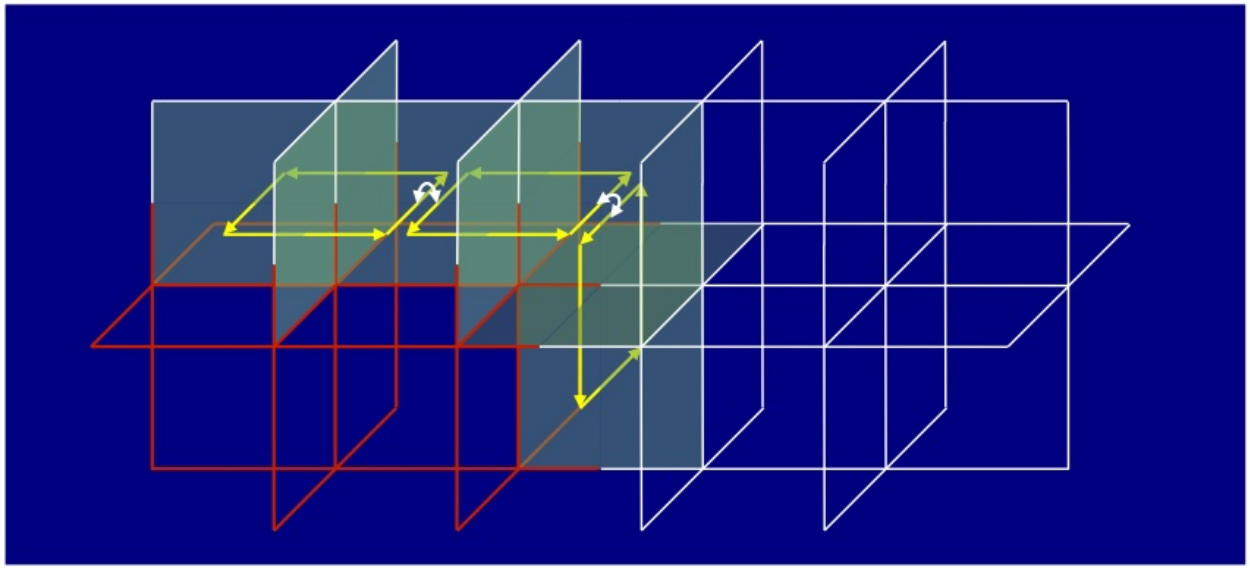
Generate the mesh connectivity:

- Iterate over all grid faces crossed by the isosurface
- Generate half-edges



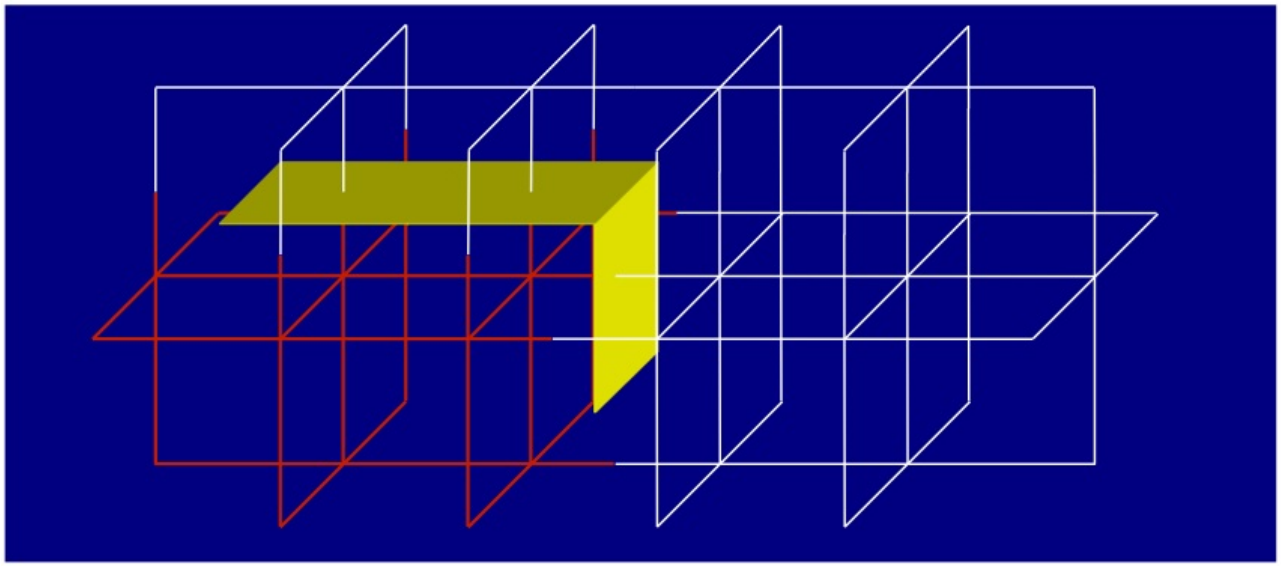
Generate the mesh connectivity:

- Iterate over all grid faces crossed by the isosurface
- Generate half-edges
- Determine pairs of half-edges



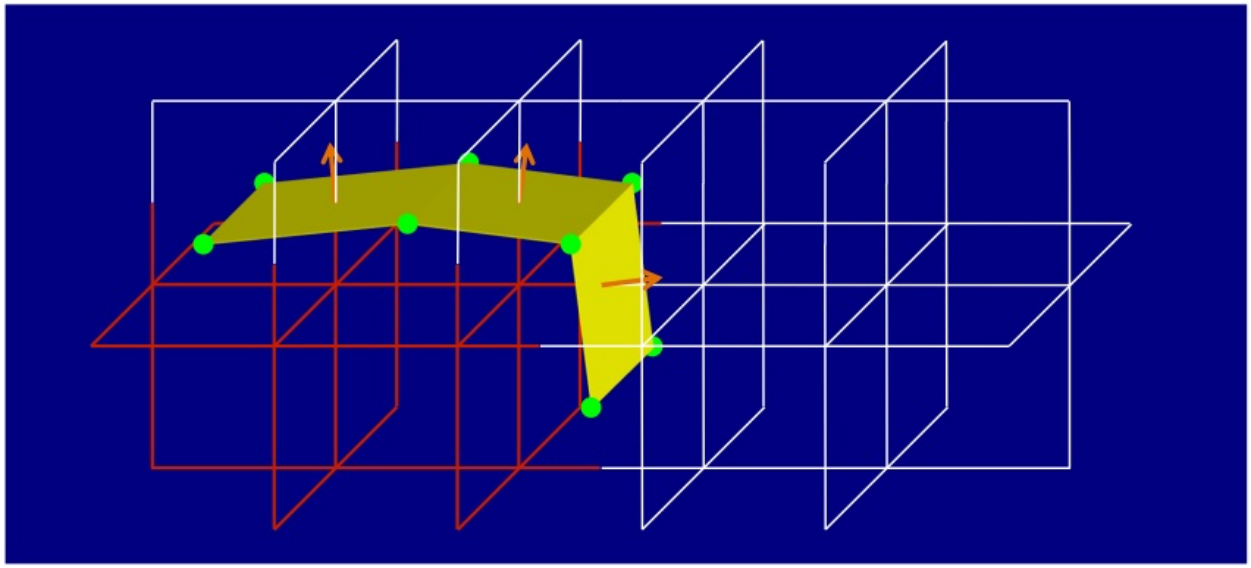
Generate the mesh connectivity:

- Iterate over all grid faces crossed by the isosurface
- Generate half-edges
- Determine pairs of half-edges
- Build a quadrilateral for each grid edge crossed by the isosurface by connecting the corresponding half-edges



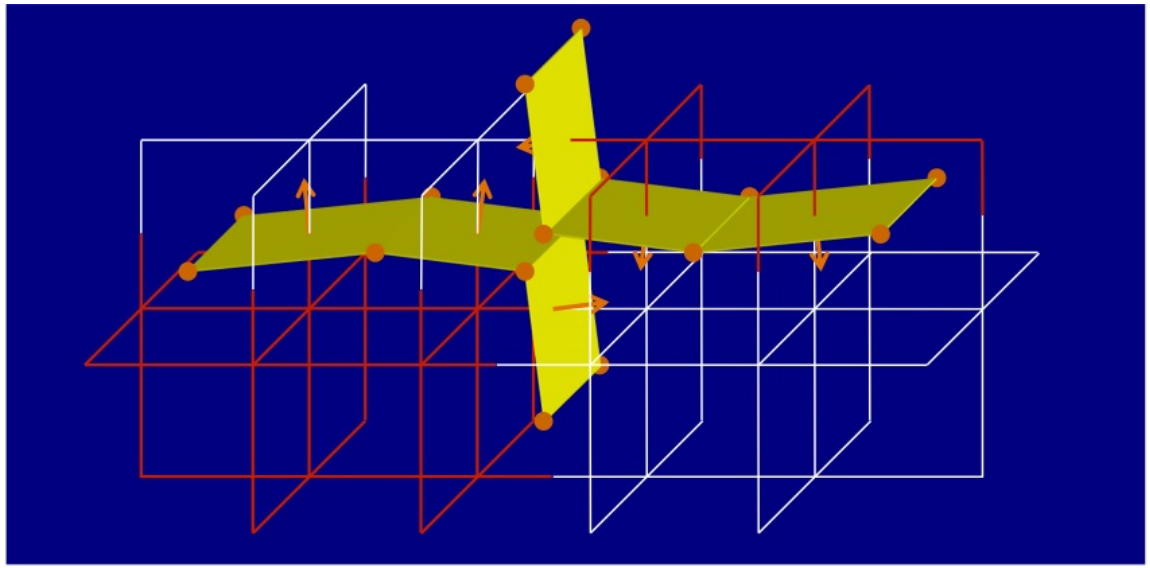
Generate the mesh connectivity:

- Iterate over all grid faces crossed by the isosurface
- Generate half-edges
- Determine pairs of half-edges
- Build a quadrilateral for each grid edge crossed by the isosurface by connecting the corresponding half-edges
- Determine vertex locations based on given intersection points and normals (using SVD, see Extended MC)

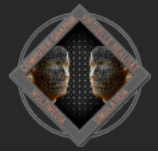
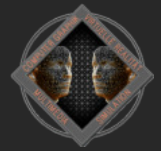


Generate the mesh connectivity:

- Iterate over all grid faces crossed by the isosurface
- Generate half-edges
- Determine pairs of half-edges
- Build a quadrilateral for each grid edge crossed by the isosurface by connecting the corresponding half-edges
- Determine vertex locations based on given intersection points and normals (using SVD, see Extended MC)



Results:

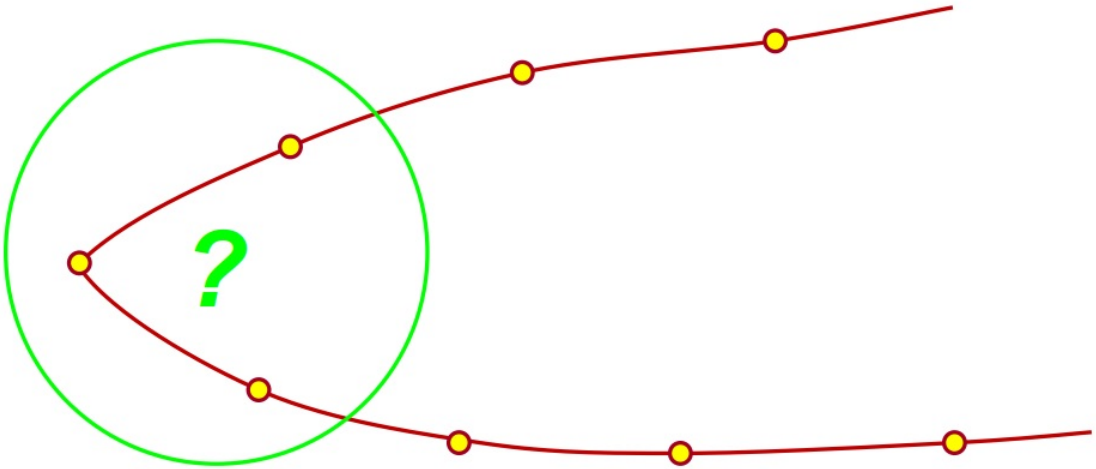
Explicit surface representations

Input: Sample of a surface F by an unordered set of points

$$\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\} \quad , \quad \boldsymbol{p}_i \in \mathbb{R}^3$$

Desired output: Reconstruct a triangle mesh out of this sample

Idea: Construct an implicit representation of the surface F by using $\mathcal{P}$. Then use Marching Cubes (or similar approaches) to reconstruct a mesh approximating F.



Problem: How to generate a signed distance function?

- Consistent inside/outside tests?
- Distance between points on the surface?

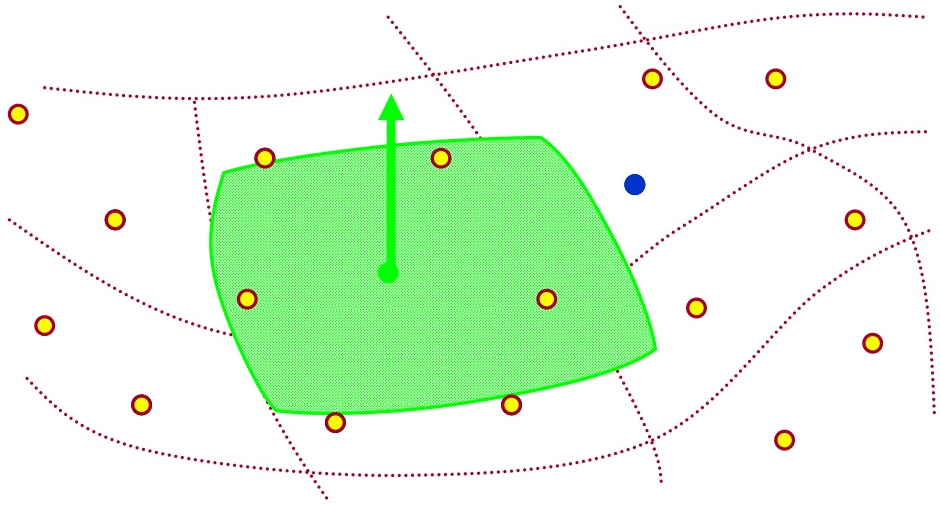
Algorithm: Fast Marching[10]

1. Compute signed distance of grid points $\boldsymbol{g}$ being in the immediate neighborhood of a sample point:

$$d(\boldsymbol{g}, \mathcal{P}) = (\boldsymbol{g} - \boldsymbol{c})^T \, \boldsymbol{n}(\boldsymbol{c})$$

   where $\boldsymbol{c}$ is the nearest neighbor of $\boldsymbol{g}$ in $\mathcal{P}$

2. Propagate distances to remaining grid points



Task: Estimate normals out of the set $\mathcal{P}$.

[10] J. A. Sethian. "A fast marching level set method for monotonically advancing fronts". In: *Proceedings of the National Academy of Sciences* 93.4 (1996), pp. 1591–1595.

Given: The local neighborhood of a point $\boldsymbol{p}$:

$$\mathcal{N}(\boldsymbol{p}) = \{\boldsymbol{p}_i \mid \boldsymbol{p}_i \text{ is close to } \boldsymbol{p}\}$$

Problem: Estimate a normal $\boldsymbol{n}$ from the objective function

$$\arg\min_{\boldsymbol{n},d} \frac{1}{n} \sum_{i=1}^{n} \left\| \boldsymbol{n}^T \boldsymbol{p}_i - d \right\|^2$$

with $\|\boldsymbol{n}\|^2 = 1$.

Theorem (Normal estimation)

The above minimization problem reduces to an eigenvalue problem

$$\boldsymbol{C}\,\boldsymbol{n} = \lambda\,\boldsymbol{n} \quad , \quad \|\boldsymbol{n}\|^2 = 1$$

with mean vector $\overline{\boldsymbol{p}}$ and covariance matrix $\boldsymbol{C}$:

$$\overline{\boldsymbol{p}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{p}_i \quad , \quad \boldsymbol{C} = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{p}_i - \overline{\boldsymbol{p}})(\boldsymbol{p}_i - \overline{\boldsymbol{p}})^T$$

Rearranging the error function leads to:

$$\arg\min_{\boldsymbol{n},d} \frac{1}{n}\sum_{i=1}^{n}\left\|\boldsymbol{n}^T\,\boldsymbol{p}_i - d\right\|^2$$

$$= \arg\min_{\boldsymbol{n},d} \frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{n}^T\,\boldsymbol{p}_i - d\right)^T\left(\boldsymbol{n}^T\,\boldsymbol{p}_i - d\right)$$

$$= \arg\min_{\boldsymbol{n},d} \frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{p}_i^T\,\boldsymbol{n}\,\boldsymbol{n}^T\,\boldsymbol{p}_i - 2\,d\,\boldsymbol{n}^T\,\boldsymbol{p}_i + d^2\right)$$

$$= \arg\min_{\boldsymbol{n},d} \frac{1}{n}\sum_{i=1}^{n}\left(\underbrace{\boldsymbol{n}^T\,\boldsymbol{p}_i}_{\text{scalar}}\,\underbrace{\boldsymbol{p}_i^T\,\boldsymbol{n}}_{\text{scalar}} - 2\,d\,\boldsymbol{n}^T\,\boldsymbol{p}_i + d^2\right)$$

$$= \arg\min_{\boldsymbol{n},d}\, \boldsymbol{n}^T\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right)\boldsymbol{n} - 2\,d\,\boldsymbol{n}^T\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\right) + d^2$$

$$= \arg\min_{\boldsymbol{n},d}\, \boldsymbol{n}^T\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right)\boldsymbol{n} - 2\,d\,\boldsymbol{n}^T\overline{\boldsymbol{p}} + d^2$$

The constraint can also be rewritten as

$$\|\boldsymbol{n}\|^2 = 1 \Leftrightarrow \boldsymbol{n}^T \boldsymbol{n} - 1 = 0$$

Now the constraint is a function of the parameters $\boldsymbol{n}$ and d.

Theorem (Multiplication rule of Lagrange)

Let $f : \mathbb{R}^n \to \mathbb{R}$ and $g_1, \ldots, g_m : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable functions. Then, any extremal point $\boldsymbol{x}_0$ fulfilling the constraints $g_1, \ldots, g_m$ (i.e. $g_1(\boldsymbol{x}_0) = \cdots = g_m(\boldsymbol{x}_0) = 0$) is a solution of the following equation system

$$\nabla f(\boldsymbol{x}) = \sum_{k=1}^{m} \lambda_k \cdot \nabla g_k(\boldsymbol{x})$$

$$g_1(\boldsymbol{x}) = 0 \ , \ \ldots \ , \ g_m(\boldsymbol{x}) = 0$$

Note: An alternative formulation is to look for an extremal point of the function

$$h(\boldsymbol{x}, \lambda_1, \ldots, \lambda_m) = f(\boldsymbol{x}) + \sum_{k=1}^{m} \lambda_k \, g_k(\boldsymbol{x})$$

Since we only have one constraint, we have to solve the equations:

$$\nabla f(\boldsymbol{n}, d) = \lambda \cdot \nabla g(\boldsymbol{n}, d) \tag{1}$$

$$\boldsymbol{n}^T \boldsymbol{n} - 1 = 0 \tag{2}$$

with

$$f(\boldsymbol{n}, d) = \boldsymbol{n}^T \left(\frac{1}{n} \sum_{i=1}^{n} \boldsymbol{p}_i \, \boldsymbol{p}_i^T \right) \boldsymbol{n} - 2\, d\, \boldsymbol{n}^T \, \overline{\boldsymbol{p}} + d^2$$

$$g(\boldsymbol{n}, d) = \boldsymbol{n}^T \, \boldsymbol{n} - 1$$

Now compute the partial derivatives:

$$\frac{\partial f}{\partial \boldsymbol{n}} = 2 \left(\frac{1}{n} \sum_{i=1}^{n} \boldsymbol{p}_i \, \boldsymbol{p}_i^T \right) \boldsymbol{n} - 2\, d\, \overline{\boldsymbol{p}} \qquad \frac{\partial g}{\partial \boldsymbol{n}} = 2\, \boldsymbol{n}$$

$$\frac{\partial f}{\partial d} = -2\, \boldsymbol{n}^T \, \overline{\boldsymbol{p}} + 2\, d \qquad \frac{\partial g}{\partial d} = 0$$

This leads to the following equation system:

$$2\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right)\boldsymbol{n}-2\,d\,\overline{\boldsymbol{p}}=\lambda\cdot 2\,\boldsymbol{n} \tag{1}$$

$$-2\,\boldsymbol{n}^T\,\overline{\boldsymbol{p}}+2\,d=\lambda\cdot 0 \tag{2}$$

$$\boldsymbol{n}^T\,\boldsymbol{n}-1=0 \tag{3}$$

We eliminate the constant 2 from the first equation and plug in $d=\boldsymbol{n}^T\overline{\boldsymbol{p}}$ which we get from the second one:

$$\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right)\boldsymbol{n}-\boldsymbol{n}^T\,\overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}=\lambda\,\boldsymbol{n}$$

$$\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right)\boldsymbol{n}-\overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T\boldsymbol{n}=\lambda\,\boldsymbol{n}$$

$$\left[\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right)-\overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T\right]\boldsymbol{n}=\lambda\,\boldsymbol{n}$$

Finally, we see that the matrix in our equation is a covariance matrix.

$$\left[\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right)-\overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T\right]\boldsymbol{n}=\lambda\,\boldsymbol{n}$$

Lemma (Covariance matrix)

Let $\{\boldsymbol{p}_1,\ldots,\boldsymbol{p}_n\}$ be a set of points. Let

$$\overline{\boldsymbol{p}}=\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i \quad , \quad \boldsymbol{C}=\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{p}_i-\overline{\boldsymbol{p}})(\boldsymbol{p}_i-\overline{\boldsymbol{p}})^T$$

be the mean vector and the covariance matrix of the point set. Then the following identity holds:

$$\boldsymbol{C}=\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right)-\overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T$$

Proof.

After rearranging the terms and using the linearity of the operators, we get:

$$
\begin{aligned}
\boldsymbol{C} &= \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{p}_i - \overline{\boldsymbol{p}})(\boldsymbol{p}_i - \overline{\boldsymbol{p}})^T \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{p}_i\,\boldsymbol{p}_i^T - \boldsymbol{p}_i\,\overline{\boldsymbol{p}}^T - \overline{\boldsymbol{p}}\,\boldsymbol{p}_i^T + \overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T\right) \\
&= \left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right) - \left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\right)\overline{\boldsymbol{p}}^T - \overline{\boldsymbol{p}}\left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\right)^T + \overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T \\
&= \left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right) - \overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T - \overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T + \overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T \\
&= \left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i\,\boldsymbol{p}_i^T\right) - \overline{\boldsymbol{p}}\,\overline{\boldsymbol{p}}^T
\end{aligned}
$$

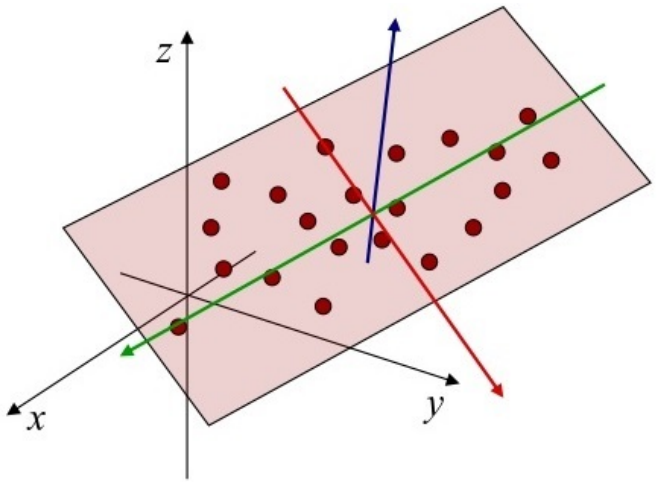
□

So we proved that the normal estimation problem is an eigenvalue problem:

$$\boldsymbol{C}\,\boldsymbol{n} = \lambda\,\boldsymbol{n} \quad , \quad \|\boldsymbol{n}\|^2 = 1$$

$\rightarrow$ Principal Component Analysis (PCA):

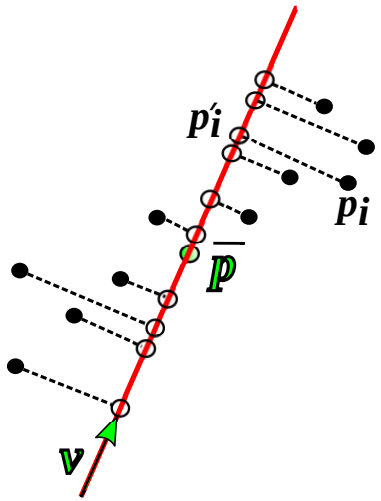
Problem: Which eigenvector corresponds to the normal?



Lemma (Variance of projected points to a line)

Let L be the line through $\overline{\boldsymbol{p}}$ in direction $\boldsymbol{v}$ with $\|\boldsymbol{v}\| = 1$. Then the variance of the projections $\boldsymbol{p}_i'$ onto L is given by

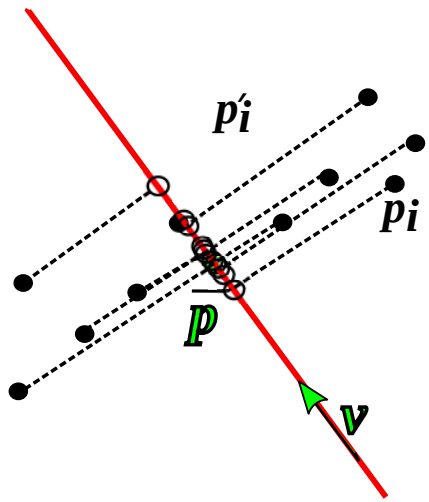
$$\sigma^2(L) := \frac{1}{n}\sum_{i=1}^{n}\left\|\boldsymbol{p}_i' - \overline{\boldsymbol{p}}\right\|_2^2 = \boldsymbol{v}^T\,\boldsymbol{C}\,\boldsymbol{v}$$



Proof.

$$
\begin{aligned}
\sigma^2(L) &= \frac{1}{n}\sum_{i=1}^{n}\left\|\boldsymbol{p}_i^{'} - \overline{\boldsymbol{p}}\right\|_2^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\langle\, \boldsymbol{v} \,|\, \boldsymbol{p_i} - \overline{\boldsymbol{p}} \,\rangle^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\langle\, \boldsymbol{v} \,|\, \boldsymbol{p_i} - \overline{\boldsymbol{p}} \,\rangle \cdot \langle\, \boldsymbol{p_i} - \overline{\boldsymbol{p}} \,|\, \boldsymbol{v} \,\rangle \\
&= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{v}^T\left(\boldsymbol{p_i} - \overline{\boldsymbol{p}}\right)\left(\boldsymbol{p_i} - \overline{\boldsymbol{p}}\right)^T\boldsymbol{v} \\
&= \boldsymbol{v}^T\left(\frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{p_i} - \overline{\boldsymbol{p}}\right)\left(\boldsymbol{p_i} - \overline{\boldsymbol{p}}\right)^T\right)\boldsymbol{v} \\
&= \boldsymbol{v}^T\,\boldsymbol{C}\,\boldsymbol{v}
\end{aligned}
$$

□



So intuitively, our normal is given as the eigenvector to the smallest eigenvalue of C since we would expect minimal variance in this direction.

Problem: But what is the correct sign of the normal for a consistent orientation?

Solution: Formulate it as a graph optimization problem[11]

- Consider two plane centers $\overline{\boldsymbol{p}}_i$ and $\overline{\boldsymbol{p}}_j$ that are close to each other
- Normals to sufficiently close surface patches of a smooth surface should point in similar directions:

$$\boldsymbol{n}_i^T \, \boldsymbol{n}_j \approx 1$$

- If not: Switch sign of either $\boldsymbol{n}_i$ or $\boldsymbol{n}_j$

$\rightarrow$ Search for normal orientation that minimizes the cost function

Complexity: Reduction to MAX-CUT $\Rightarrow$ NP-hard

[11] H. Hoppe et al. "Surface Reconstruction from Unorganized Points". In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '92. ACM, 1992, pp. 71–78.

Solution: Formulate it as a graph optimization problem

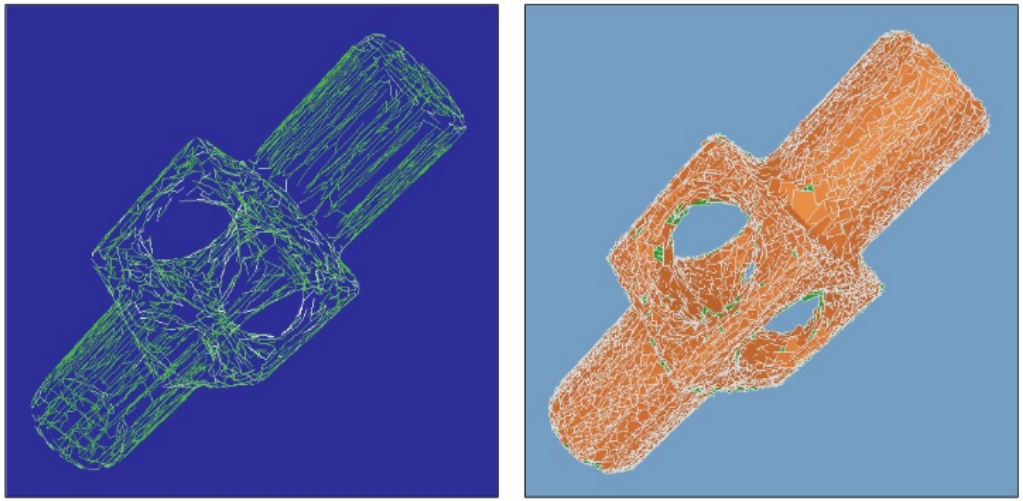
Complexity: Reduction to MAX-CUT $\Rightarrow$ NP-hard

Approximate solution:

- As mentioned before, create graph of the k closest centers (Riemannian Graph)
- Weight for each edge with $c(i, j)$
- Compute Minimum Spanning Tree (MST)
- Search center with the largest z-coordinate. Set sign of normal towards z-direction
- Propagate normal orientations along the MST

Complexity: $\mathcal{O}(n \log n)$

Results:

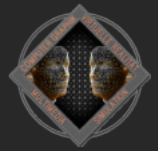
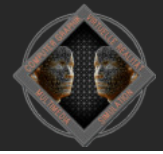


(a) Traversal order of orientation propagation

(b) Oriented tangent planes ($Tp(\mathbf{x}_i)$)

(c) Cubes visited during contouring

(d) Estimated signed distance (shown as $\mathbf{p} - \mathbf{z}$)

Least Squares Methods

Given: Given a set of n points located at positions $\boldsymbol{x}_i \in \mathbb{R}^d$ with scalar values f_i (describing e.g. some surface properties).

Problem: Find a globally defined function $f(\boldsymbol{x})$ that approximates the given scalar values at the given points in the least squares sense

$$\min_{f \in \prod_m^d} E_{LS} = \min_{f \in \prod_m^d} \sum_{i=1}^{n} \| f(\boldsymbol{x}_i) - f_i \|_2^2$$

where f is taken from the space of polynomials $\prod_m^d$ of total degree m in d spatial dimensions. This function can be written as

$$f(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x})^T \boldsymbol{c} = \langle \boldsymbol{b}(\boldsymbol{x}) \,|\, \boldsymbol{c} \rangle$$

where $\boldsymbol{b}(\boldsymbol{x}) = (b_1 \boldsymbol{x}, \ldots, b_k \boldsymbol{x})^T$ is the polynomial basis vector and $\boldsymbol{c} = (c_1, \ldots, c_k)^T$ is the vector of unknown coefficients.

Example: For $m = 2$ and $d = 2$, a polynomial basis is given by:

$$\boldsymbol{b}(\boldsymbol{x}) = (1, x_1, x_2, x_1^2, x_1\, x_2, x_2^2)^T$$

Solution: We can minimize the error function by setting the partial derivatives to zero:

$$0 = \frac{\partial E_{WLS}}{\partial c_k} = \frac{\partial}{\partial c_k} \sum_{i=1}^{n} \left\| \boldsymbol{b}(\boldsymbol{x}_i)^T \boldsymbol{c} - f_i \right\|_2^2 = \sum_{i=1}^{n} 2 \left(\boldsymbol{b}(\boldsymbol{x}_i)^T \boldsymbol{c} - f_i \right) b_k(\boldsymbol{x}_i)$$

In matrix-vector notation, this can be written as

$$\mathbf{0} = 2 \sum_{i=1}^{n} \boldsymbol{b}(\boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)^T \boldsymbol{c} - \boldsymbol{b}(\boldsymbol{x}_i)\, f_i$$

Rearranging this equation yields the least-squares equation

$$\underbrace{\sum_{i=1}^{n} \boldsymbol{b}(\boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)^T}_{\boldsymbol{A}_{LS}} \boldsymbol{c} = \underbrace{\sum_{i=1}^{n} \boldsymbol{b}(\boldsymbol{x}_i)\, f_i}_{\boldsymbol{b}_{LS}}$$

which can be solved by

$$\boldsymbol{c} = \left(\sum_{i=1}^{n} \boldsymbol{b}(\boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)^T \right)^{-1} \sum_{i=1}^{n} \boldsymbol{b}(\boldsymbol{x}_i)\, f_i$$

Problem: Find a globally defined function $f(\boldsymbol{x})$ that approximates the given scalar values at the given points in the least squares sense

$$\min_{f \in \prod_m^d} E_{LS} = \min_{f \in \prod_m^d} \sum_{i=1}^{n} \theta(\overline{\boldsymbol{x}} - \boldsymbol{x}_i) \left\| f_{\overline{\boldsymbol{x}}}(\boldsymbol{x}_i) - f_i \right\|_2^2$$

where $\overline{\boldsymbol{x}} \in \mathbb{R}^d$ is a fixed reference point used for weighting the individual error terms. The function f can now be written as

$$f_{\overline{\boldsymbol{x}}}(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x})^T \, \boldsymbol{c}(\overline{\boldsymbol{x}}) = \langle \, \boldsymbol{b}(\boldsymbol{x}) \, | \, \boldsymbol{c}(\overline{\boldsymbol{x}}) \, \rangle$$

and is only defined locally within a radius r around $\overline{\boldsymbol{x}}$, i.e. $\|\boldsymbol{x} - \overline{\boldsymbol{x}}\| < r$.

Weight Function Choices:

- Gaussian: $\theta(d) = e^{-\frac{d^2}{h^2}}$
- Wendland function: $\theta(d) = \left(1 - \frac{d}{h}\right)^4 \left(\frac{4\,d}{h} + 1\right)$
- Other: $\theta(d) = \frac{1}{d^2 + \epsilon^2}$

Solution: Similar to the unweighted version of the least-squares approximation problem, we can minimize the error function by setting the partial derivatives to zero:

$$0 = \frac{\partial E_{WLS}}{\partial c_k} = \sum_{i=1}^{n} \theta(\overline{\boldsymbol{x}} - \boldsymbol{x}_i)\, 2 \left(\boldsymbol{b}(\boldsymbol{x}_i)^T \boldsymbol{c}(\overline{\boldsymbol{x}}) - f_i \right) b_k(\boldsymbol{x}_i)$$

Rearranging this equation in matrix-vector notation yields the weighted least-squares equation

$$\underbrace{\sum_{i=1}^{n} \theta(\overline{\boldsymbol{x}} - \boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)^T}_{\boldsymbol{A}_{WLS}} \boldsymbol{c}(\overline{\boldsymbol{x}}) = \underbrace{\sum_{i=1}^{n} \theta(\overline{\boldsymbol{x}} - \boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)\, f_i}_{\boldsymbol{b}_{WLS}}$$

which can be solved by

$$\boldsymbol{c}(\overline{\boldsymbol{x}}) = \left(\sum_{i=1}^{n} \theta(\overline{\boldsymbol{x}} - \boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)^T \right)^{-1} \sum_{i=1}^{n} \theta(\overline{\boldsymbol{x}} - \boldsymbol{x}_i)\, \boldsymbol{b}(\boldsymbol{x}_i)\, f_i$$

Here, the major difference is that the coefficients $\boldsymbol{c}$ now depend on the local reference point $\overline{\boldsymbol{x}}$ and need to be recompted for every new reference point.

Idea: Use the WLS approximation method and move the reference point $\overline{x}$ over the entire parameter domain:

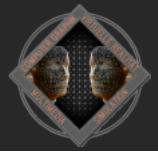
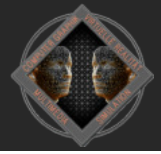
$$f(\boldsymbol{x}) = f_{\boldsymbol{x}}(\boldsymbol{x})$$

This leads to the following MLS error function

$$\min_{f \in \prod_m^d} E_{MLS}$$

$$= \min_{f \in \prod_m^d} \sum_{i=1}^{n} \theta(\boldsymbol{x} - \boldsymbol{x}_i) \, \| f_{\boldsymbol{x}}(\boldsymbol{x}_i) - f_i \|_2^2$$

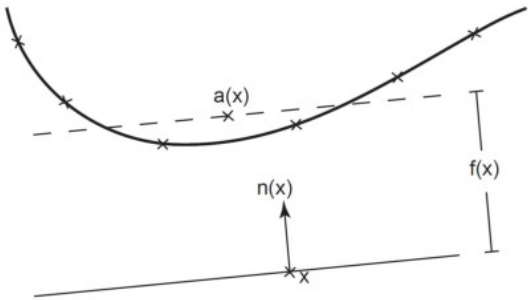
which can be solved like the WLS problem.

Levin's Projection Procedure

Let $f : \mathbb{R}^3 \mapsto \mathbb{R}$ be defined as

$$f(\boldsymbol{x}) = \boldsymbol{n}(\boldsymbol{x})^T \left(\boldsymbol{x} - \boldsymbol{a}(\boldsymbol{x})\right)$$

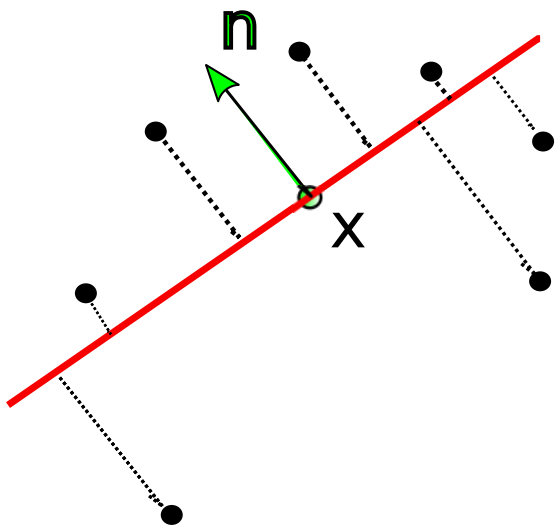


where

$$\boldsymbol{a}(\boldsymbol{x}) = \frac{\sum_{i=1}^{n} \theta(\|\boldsymbol{x} - \boldsymbol{p}_i\|)\, \boldsymbol{p}_i}{\sum_{i=1}^{n} \theta(\|\boldsymbol{x} - \boldsymbol{p}_i\|)}$$

and $\boldsymbol{n}$ is derived from a least squares fit of a plane with unit normal $\boldsymbol{n}$ through $\boldsymbol{x}$, i.e. is the minimizer of

$$\min_{\|\boldsymbol{n}\|=1} \frac{\sum_{i=1}^{n} \theta(\|\boldsymbol{x} - \boldsymbol{p}_i\|) \left\|\boldsymbol{n}^T \left(\boldsymbol{x} - \boldsymbol{p}_i\right)\right\|}{\sum_{i=1}^{n} \theta(\|\boldsymbol{x} - \boldsymbol{p}_i\|)}$$



Then the approximating surface is defined as

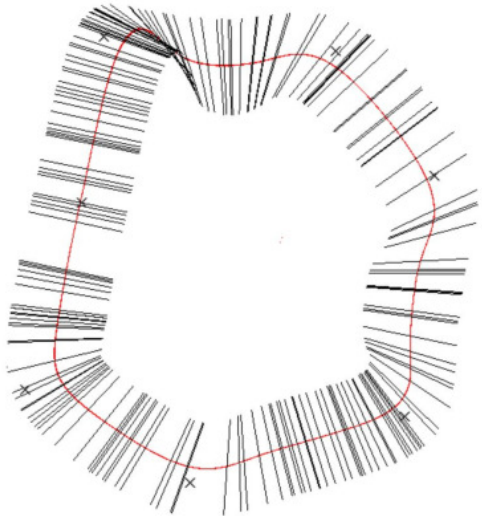
$$\hat{\mathcal{S}} = \{\boldsymbol{x} \in \mathbf{R} \mid f(\boldsymbol{x}) = 0\}$$

12

[12] M. Alexa and A. Adamson. "On Normals and Projection Operators for Surfaces Defined by Point Sets.". In: *SPBG* 4 (2004), pp. 149–155

The constraint minimizing problem

$$\min_{\|\boldsymbol{n}\|=1} \frac{\sum_{i=1}^{n} \theta(\|\boldsymbol{x}-\boldsymbol{p}_i\|) \left\|\boldsymbol{n}^T(\boldsymbol{x}-\boldsymbol{p}_i)\right\|}{\sum_{i=1}^{n} \theta(\|\boldsymbol{x}-\boldsymbol{p}_i\|)}$$



is solved by the smallest eigenvectors of the covariance matrix $\boldsymbol{W}(\boldsymbol{x}) = \{w_{jk}\}$ where

$$w_{jk} = \sum_i \left(\boldsymbol{e}_j^T(\boldsymbol{x}-\boldsymbol{p}_i)\right)\left(\boldsymbol{e}_k^T(\boldsymbol{x}-\boldsymbol{p}_i)\right)\theta(\|\boldsymbol{x}-\boldsymbol{p}_i\|)$$

If each point $\boldsymbol{p}_i$ carries a normal $\boldsymbol{n}_i$, we can define the normal $\boldsymbol{n}(\boldsymbol{x})$ using the weighted averages as for the points:

$$\boldsymbol{n}(\boldsymbol{x}) = \frac{\sum_{i=1}^{n} \theta(\|\boldsymbol{x}-\boldsymbol{p}_i\|)\,\boldsymbol{n}_i}{\left\|\sum_{i=1}^{n} \theta(\|\boldsymbol{x}-\boldsymbol{p}_i\|)\,\boldsymbol{n}_i\right\|}$$

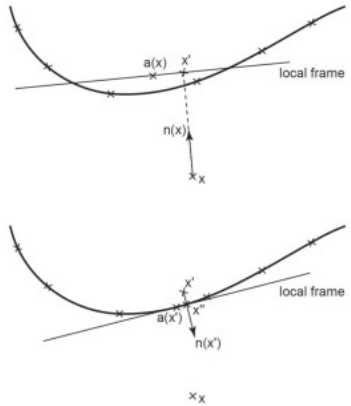
Let $\boldsymbol{n}(\boldsymbol{x})$ and $\boldsymbol{a}(\boldsymbol{x})$ define a tangent frame with origin in $\boldsymbol{a}(\boldsymbol{x})$ and let the projection of $\boldsymbol{x}$ onto the tangent be

$$\boldsymbol{p}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{n}(\boldsymbol{x})^T \left(\boldsymbol{a}(\boldsymbol{x}) - \boldsymbol{x}\right) \boldsymbol{n}(\boldsymbol{x})$$

Observation: Levin, 2004

$$\boldsymbol{p}(\boldsymbol{x}) = \boldsymbol{x} \Leftrightarrow \boldsymbol{x} \in \mathcal{S}$$

Idea: Given a current point $\boldsymbol{x}$, we iteratively define a series of points $\boldsymbol{q}_i$, such that $\boldsymbol{q}_{i+1}$ is the orthogonal projection of $\boldsymbol{x}$ onto the tangent plane defined by $\boldsymbol{a}(\boldsymbol{q}_i)$ and $\boldsymbol{n}(\boldsymbol{q}_i)$.

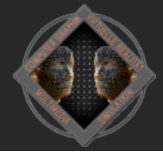


Starting with $\boldsymbol{q}_0 = \boldsymbol{x}$, perform the following steps:

1. Compute $\boldsymbol{a}(\boldsymbol{q}_i)$ and set

$$\boldsymbol{q}_{i+1} = \boldsymbol{a}(\boldsymbol{q}_i)$$

2. Compute $\boldsymbol{a}(\boldsymbol{q}_i)$,$\boldsymbol{n}(\boldsymbol{q}_{i+1})$ and set $\boldsymbol{q}_{i+1} = \boldsymbol{p}(\boldsymbol{q}_{i+1})$
3. If $\left\| \boldsymbol{n}(\boldsymbol{q}_{i+1})^T \left(\boldsymbol{a}(\boldsymbol{q}_{i+1}) - \boldsymbol{q}_{i+1}\right) \right\| > \epsilon$ go back to 2

Algebraic Point Set Surfaces

An algebraic sphere[13] in d dimensions with center $\boldsymbol{c}$ and radius r is the zero-set of

$$s_{\boldsymbol{c},r}(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{c}\|_2^2 - r^2$$

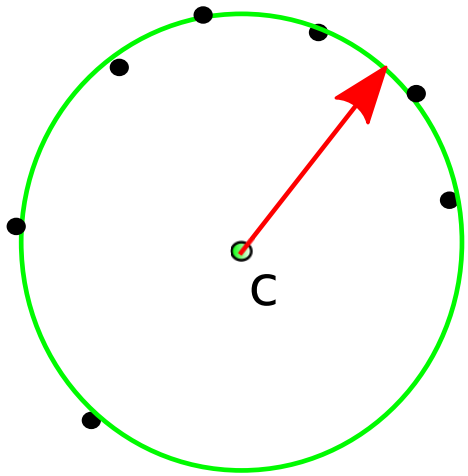
This can be rewritten as

$$s_{\boldsymbol{u}}(\boldsymbol{x}) = \begin{pmatrix} 1 & \boldsymbol{x} & \boldsymbol{x}^T\,\boldsymbol{x} \end{pmatrix} \cdot \boldsymbol{u}$$

where $\boldsymbol{u} = (u_0, \ldots, u_{d+1})^T \in \mathbb{R}^{d+2}$ determines the sphere.
For $u_{d+1} \neq 0$, we get:

$$\boldsymbol{c} = -\frac{1}{2\,u_{d+1}} (u_1, \ldots, u_d)^T$$

$$r = \sqrt{\boldsymbol{c}^T\,\boldsymbol{c} - \frac{u_0}{u_{d+1}}}$$



In degenerate cases, $\boldsymbol{u}$ corresponds to the coefficients of a plane equation with u_0 representing the plane's distance from the origin and $\boldsymbol{u}_{1d} = (u_1, \ldots, u_d)^T$ being its normal.

[13] G. Guennebaud and M. Gross. "Algebraic point set surfaces". In: *ACM Transactions on Graphics (TOG)*. vol. 26. 3. ACM. 2007, p. 23.

Let n be the number of points and let $\boldsymbol{W}(\boldsymbol{x})$ and $\boldsymbol{D}$ respectively the $n \times n$ and the $n \times (d+2)$ design matrix defined as:

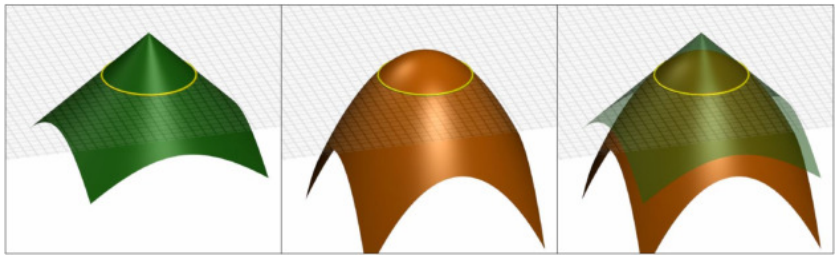
$$\boldsymbol{W}(\boldsymbol{x}) = \begin{pmatrix} w_0(\boldsymbol{x}) & & \\ & \ddots & \\ & & w_{n-1}(\boldsymbol{x}) \end{pmatrix} \quad , \quad \boldsymbol{D} = \begin{pmatrix} 1 & \boldsymbol{p}_0^T & \boldsymbol{p}_0^T\,\boldsymbol{p}_0 \\ \vdots & \vdots & \vdots \\ 1 & \boldsymbol{p}_{n-1}^T & \boldsymbol{p}_{n-1}^T\,\boldsymbol{p}_{n-1} \end{pmatrix}$$

Then, the solution $\boldsymbol{u}(\boldsymbol{x})$ of our algebraic sphere fit at a given point $\boldsymbol{x} \in \mathbb{R}^d$ can be expressed as:

$$\boldsymbol{u}(\boldsymbol{x}) = \underset{\boldsymbol{u}, \boldsymbol{u} \neq 0}{\arg\min} \left\| \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{x})\, \boldsymbol{D}\, \boldsymbol{u} \right\|_2^2$$

To avoid the trivial solution $\boldsymbol{u} = 0$, the norm of the gradient at the surface is fixed to unit length (Pratt's constraint):

$$\left\| (u_1, \ldots, u_d)^T \right\|_2^2 - 4\, u_0\, u_{d+1} = 1$$



This constraint can be written in the form

$$\boldsymbol{u}^T \boldsymbol{C}\, \boldsymbol{u} = 1 \quad , \quad \boldsymbol{C} = \begin{pmatrix} 0 & 0 & \ldots & 0 & -2 \\ 0 & 1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & & 1 & 0 \\ -2 & 0 & \ldots & 0 & 0 \end{pmatrix}$$

Using Lagrangian multipliers to solve the constraint minimization problem, the solution $\boldsymbol{u}(\boldsymbol{x})$ leads to the following generalized Eigen problem:

$$\boldsymbol{D}^T \boldsymbol{W}\, \boldsymbol{D}\, \boldsymbol{u}(\boldsymbol{x}) = \lambda\, \boldsymbol{C}\, \boldsymbol{u}(\boldsymbol{x})$$

where $\lambda \in \mathbb{R}$ is a Lagrangian Multiplier.

Point normals can be used as additional constraints to the optimization problem

$$\boldsymbol{u}(\boldsymbol{x}) = \underset{\boldsymbol{u},\boldsymbol{u}\neq 0}{\arg\min} \left\| \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{x})\, \boldsymbol{D}\, \boldsymbol{u} \right\|_2^2$$

by adding derivative constraints

$$\nabla s_{\boldsymbol{u}}(\boldsymbol{p}_i) = \boldsymbol{n}_i$$

with $\|\boldsymbol{n}_i\| = 1$. This leads to the following system of equations:

$$\boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{x})\, \boldsymbol{D}\, \boldsymbol{u} = \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{x})\, \boldsymbol{b}$$

where

$$\boldsymbol{W}(\boldsymbol{x}) = \begin{pmatrix} \ddots & & & & \\ & w_i(\boldsymbol{x}) & & & \\ & & \beta\, w_i(\boldsymbol{x}) & & \\ & & & \ddots & \\ & & & & \beta\, w_i(\boldsymbol{x}) \\ & & & & & \ddots \end{pmatrix}, \quad \boldsymbol{D} = \begin{pmatrix} \vdots & \vdots & \vdots \\ 1 & \boldsymbol{p}_i^T & \boldsymbol{p}_i \\ 0 & \boldsymbol{e}_0^T & 2\,\boldsymbol{e}_0^T\,\boldsymbol{p}_i \\ \vdots & \vdots & \vdots \\ 0 & \boldsymbol{e}_{d-1}^T & 2\,\boldsymbol{e}_{d-1}^T\,\boldsymbol{p}_i \\ \vdots & \vdots & \vdots \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} \vdots \\ 0 \\ \boldsymbol{e}_0^T\,\boldsymbol{n}_i \\ \vdots \\ \boldsymbol{e}_{d-1}^T\,\boldsymbol{n}_i \\ \vdots \end{pmatrix},$$

$\boldsymbol{e}_k$ denotes the k-th unit basis vector and $\beta \in \mathbb{R}$ weights the normal constraints.

This leads to the following system of equations:

$$\boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{x})\,\boldsymbol{D}\,\boldsymbol{u} = \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{x})\,\boldsymbol{b}$$

The equation is solved by using the the pseudo-inverse method:

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{A}^{-1}(\boldsymbol{x})\,\hat{\boldsymbol{b}}(\boldsymbol{x})$$

where

$$\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{D}^T\,\boldsymbol{W}(\boldsymbol{x})\,\boldsymbol{D} \quad , \quad \hat{\boldsymbol{b}}(\boldsymbol{x}) = \boldsymbol{D}^T\,\boldsymbol{W}(\boldsymbol{x})\,\boldsymbol{b}$$

can be directly and efficiently computed by weighted sums of the point coefficients.

Idea: Define a Moving Least Squared (MLS) surface based on sphere fits.

The implicit scalar filed $f(\boldsymbol{x})$ represents the algebraic distance between the evaluation point $\boldsymbol{x}$ and the fitted sphere $\boldsymbol{u}(\boldsymbol{x})$:

$$f(\boldsymbol{x}) = s_{\boldsymbol{u}(\boldsymbol{x})}(\boldsymbol{x}) = \begin{pmatrix} 1 & \boldsymbol{x} & \boldsymbol{x}^T\,\boldsymbol{x} \end{pmatrix} \cdot \boldsymbol{u}(\boldsymbol{x}) = 0$$

The gradients of the scalar field can be used to estimate MLS surface normals or perform an orthogonal projection onto the MLS surface:

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} 1 & \boldsymbol{x} & \boldsymbol{x}^T\,\boldsymbol{x} \end{pmatrix} \cdot \nabla \boldsymbol{u}(\boldsymbol{x}) + \begin{pmatrix} 0 & \boldsymbol{e}_0^T & 2\,\boldsymbol{e}_0^T\,\boldsymbol{x} \\ \vdots & \vdots & \vdots \\ 0 & \boldsymbol{e}_{d-1}^T & 2\,\boldsymbol{e}_{d-1}^T\,\boldsymbol{x} \end{pmatrix} \cdot \boldsymbol{u}(\boldsymbol{x})$$

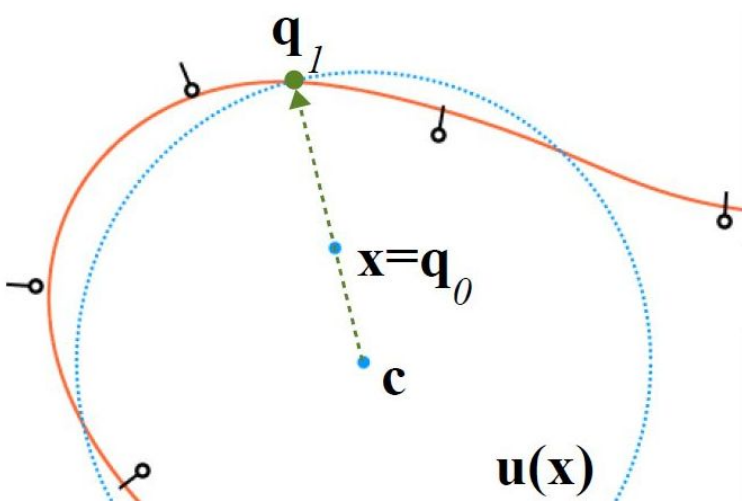
The values of the gradient $\nabla \boldsymbol{u}(\boldsymbol{x})$ can be computed using the solution of the pseudo-inverse method:

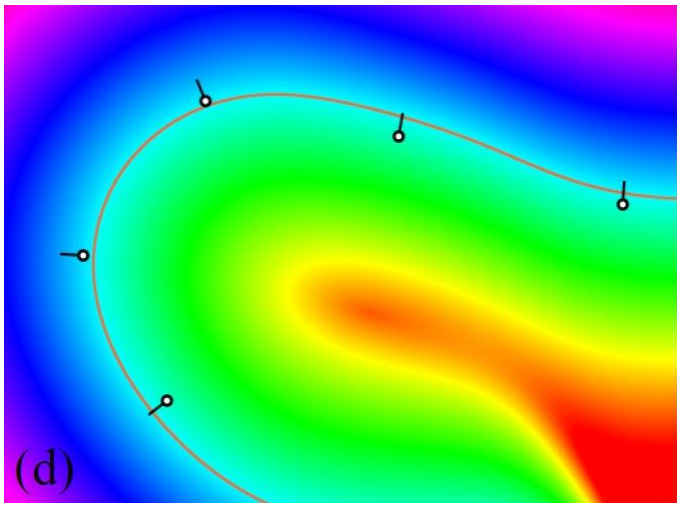
$$\frac{\partial \boldsymbol{u}(\boldsymbol{x})}{\partial x_k} = \boldsymbol{A}^{-1}(\boldsymbol{x}) \left(-\frac{\partial \boldsymbol{A}^{-1}(\boldsymbol{x})}{\partial x_k}\,\boldsymbol{u}(\boldsymbol{x}) + \frac{\partial \hat{\boldsymbol{b}}(\boldsymbol{x})}{\partial x_k} \right)$$

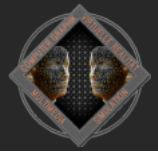
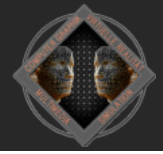
Projection procedure: Projection operators for the plane case can easily be adapted to our setting by simply replacing the planar projection by a spherical projection.

Given a current point $\boldsymbol{x}$, we iteratively define a series of points $\boldsymbol{q}_i$, such that $\boldsymbol{q}_{i+1}$ is the orthogonal projection of $\boldsymbol{x}$ onto the algebraic sphere defined by $\boldsymbol{u}(\boldsymbol{q}_i)$.

Starting with $\boldsymbol{q}_0 = \boldsymbol{x}$, the projection of $\boldsymbol{x}$ onto the surface is asymptotically $\boldsymbol{q}_\infty$.





References I

[1] M. Alexa and A. Adamson. "On Normals and Projection Operators for Surfaces Defined by Point Sets.". In: *SPBG* 4 (2004), pp. 149–155.

[2] J. F. Blinn. "A generalization of algebraic surface drawing". In: *ACM transactions on graphics (TOG)* 1.3 (1982), pp. 235–256.

[3] T. Davies, D. Nowrouzezahrai, and A. Jacobson. "Overfit Neural Networks as a Compact Shape Representation". In: *arXiv preprint arXiv:2009.09808* (2020).

[4] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. "Adaptively sampled distance fields: A general representation of shape for computer graphics". In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, pp. 249–254.

[5] M. Garland and P. S. Heckbert. "Surface simplification using quadric error metrics". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 209–216.

[6] A. Greß and R. Klein. "Efficient representation and extraction of 2-manifold isosurfaces using kd-trees". In: *Graphical Models* 66.6 (2004), pp. 370–397.

[7] G. Guennebaud and M. Gross. "Algebraic point set surfaces". In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 23.

[8] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. "Surface Reconstruction from Unorganized Points". In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '92. ACM, 1992, pp. 71–78.

[9] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. "Feature sensitive surface extraction from volume data". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, pp. 57–66.

[10] D. Levin. "Mesh-independent surface interpolation". In: *Geometric modeling for scientific visualization*. Springer, 2004, pp. 37–49.

[11] W. E. Lorensen and H. E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *ACM siggraph computer graphics*. Vol. 21. 4. ACM. 1987, pp. 163–169.

[12] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. "Function representation in geometric modeling: concepts, implementation and applications". In: *The visual computer* 11.8 (1995), pp. 429–446.

[13] T. W. Sederberg and S. R. Parry. “Free-form deformation of solid geometric models”. In: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 1986, pp. 151–160.

[14] J. A. Sethian. “A fast marching level set method for monotonically advancing fronts”. In: *Proceedings of the National Academy of Sciences* 93.4 (1996), pp. 1591–1595.

[15] G. Taubin. “Rasterizing algebraic curves and surfaces”. In: *IEEE Computer graphics and applications* 14.2 (1994), pp. 14–23.

[16] B. Wyvill, A. Guy, and E. Galin. “Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system”. In: *Computer Graphics Forum*. Vol. 18. 2. Wiley Online Library. 1999, pp. 149–158.