

CENG 789 – Digital Geometry Processing

06- Surface Reconstruction,

Scientific Visualization and Information Visualization

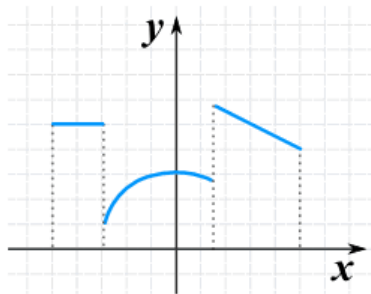
Prof. Dr. Yusuf Sahillioğlu

Computer Eng. Dept,  MIDDLE EAST TECHNICAL UNIVERSITY, Turkey

Explicit Representation

2 / 98

- ✓ RECALL: Polygon meshes are piecewise linear surface representations.
- ✓ Analogous to piecewise functions:



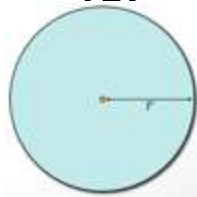
$$f(x) = \begin{cases} 6 & \text{if } x < -2 \\ x^2 & \text{if } x \geq -2 \text{ and } x \leq 2 \\ 10 - x & \text{if } x > 2 \end{cases}$$

- ✓ Think surface as (the range of) a “shape” function.

$$\mathbf{f} : [0, 2\pi] \rightarrow \mathbb{R}^2$$

vs.

$$\mathbf{f}(t) = \begin{pmatrix} r \cos(t) \\ r \sin(t) \end{pmatrix}$$



$$\mathbf{f} : [0, 2\pi] \rightarrow \mathbb{R}^2$$

$$\mathbf{f}(t) = \begin{pmatrix} ? \\ ? \end{pmatrix}$$



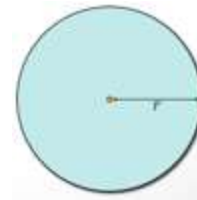
Explicit Surface Representation

3 / 98

- ✓ For varying t , we get 2D shape (circle) points: t in $[0, 2\pi]$.
- ✓ Parameterization \mathbf{f} maps a 1D parameter domain, e.g., $[0, 2\pi]$, to the curve (surface) embedded in \mathbb{R}^2 (\mathbb{R}^3).

$$\mathbf{f} : [0, 2\pi] \rightarrow \mathbb{R}^2$$

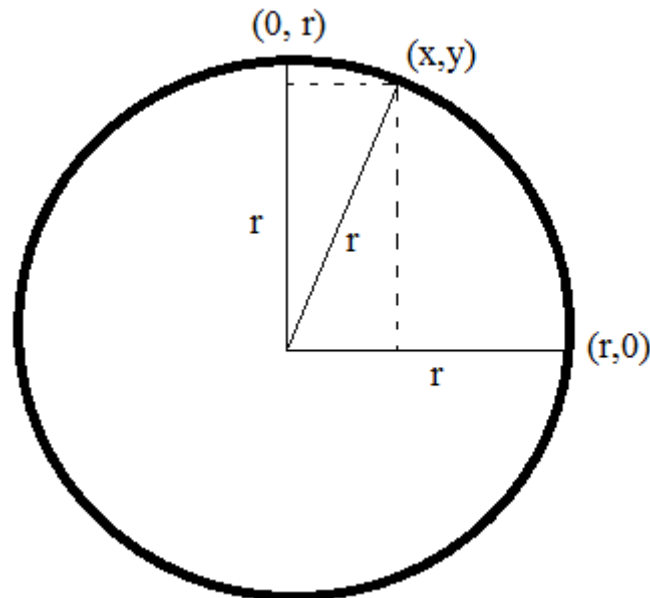
$$\mathbf{f}(t) = \begin{pmatrix} r \cos(t) \\ r \sin(t) \end{pmatrix}$$



Implicit Surface Representation

4 / 98

- ✓ Implicit or indirect surface representation.
- ✓ Surface is defined to be the zero-set of a scalar-valued function F .
 - ✓ That is, we have $F: \mathbb{R}^2 \rightarrow \mathbb{R}$, and then surface $\mathcal{S} = \{x \in \mathbb{R}^2 \mid F(x) = 0\}$.
 - ✓ Circle of radius r : $(x, y) \mapsto \sqrt{x^2 + y^2} - r$
is defined by (x, y) pairs
that make the right-hand-side 0

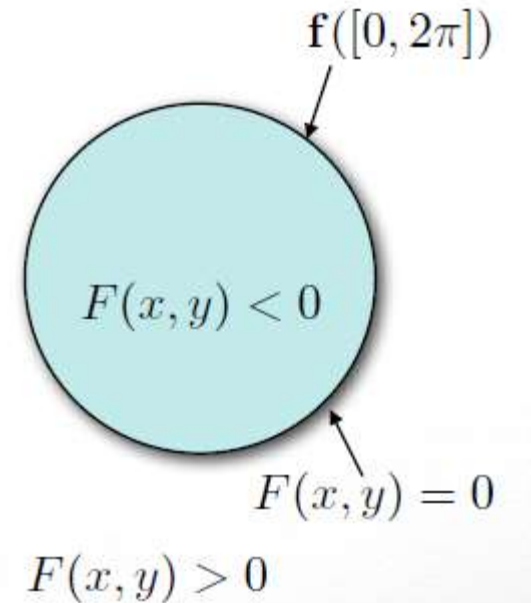


Explicit vs. Implicit

5 / 98

- ✓ Explicit: $\mathbf{f}(t) = \begin{pmatrix} r \cos(t) \\ r \sin(t) \end{pmatrix}$
 - ✓ Range of a parameterization function.

- ✓ Implicit: $F(x, y) = \sqrt{x^2 + y^2} - r$
 - ✓ Kernel of an implicit function.



Explicit vs. Implicit

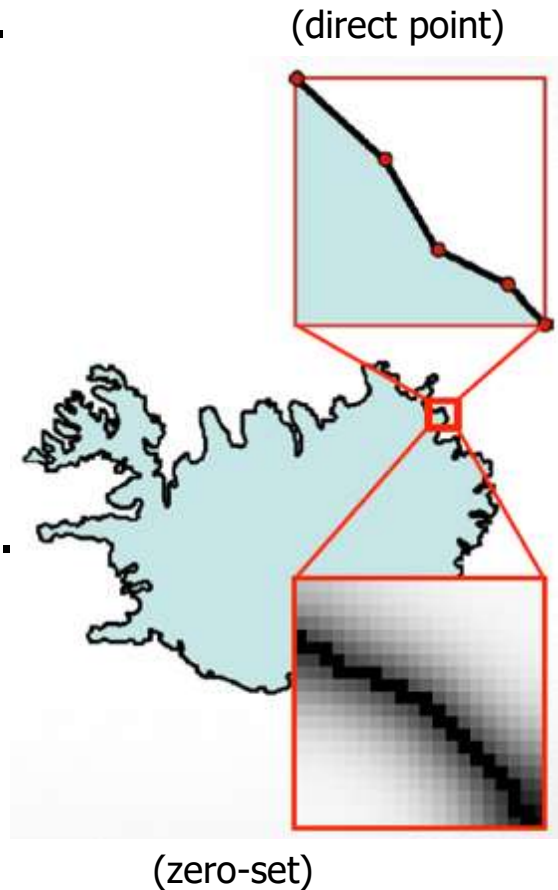
6 / 98

✓ Explicit: for non-mathematical/arbitrary shapes.

- ✓ Range of a parameterization function.
- ✓ Piecewise approximation.

✓ Implicit: for non-mathematical/arbitrary shapes.

- ✓ Kernel of an implicit function.
- ✓ Piecewise approximation.



Explicit vs. Implicit

7 / 98

✓ Explicit:

- ✓ Range of a parameterization function.
- ✓ Piecewise approximation.
- ✓ Splines, triangle meshes, points.
- ✓ Easy rendering.
- ✓ Easy geom. modif. (deformation or new t).

✓ Implicit:

- ✓ Kernel of an implicit function.
- ✓ Piecewise approximation.
- ✓ Scalar-valued 3D grid (mesh lies here).
- ✓ Easy in/out test (just evaluate F).
- ✓ Easy topology modification.

(a) The line in \mathbb{R}^2 through the points $(1, 2)$ and $(2, 3)$.

explicit: $(1, 2), (2, 3)$

implicit: $x - y = -1$

(b) The point $(0, 1, 2)$ in \mathbb{R}^3 .

explicit: $(0, 1, 2)$

implicit: $x = 0 \wedge y = 1 \wedge z = 2$

(c) The plane in \mathbb{R}^3 through the point $(0, 1, 1)$ and perpendicular to the vector $(2, 1, 0)$.

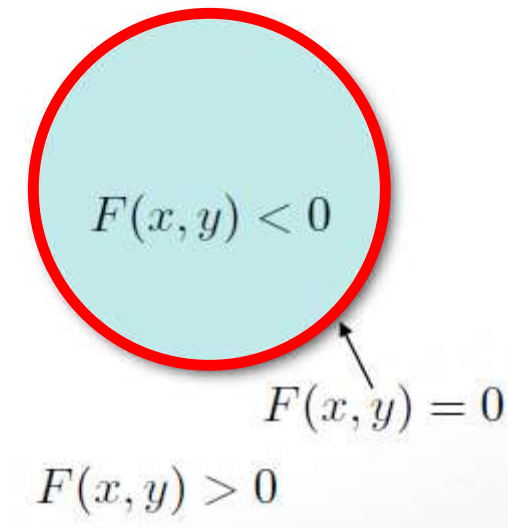
explicit: $(0, 1, 0), (0, 1, 1), (-1, 3, 0)$

implicit: $2x + y = 1$

Implicit/Indirect Surfaces

8 / 98

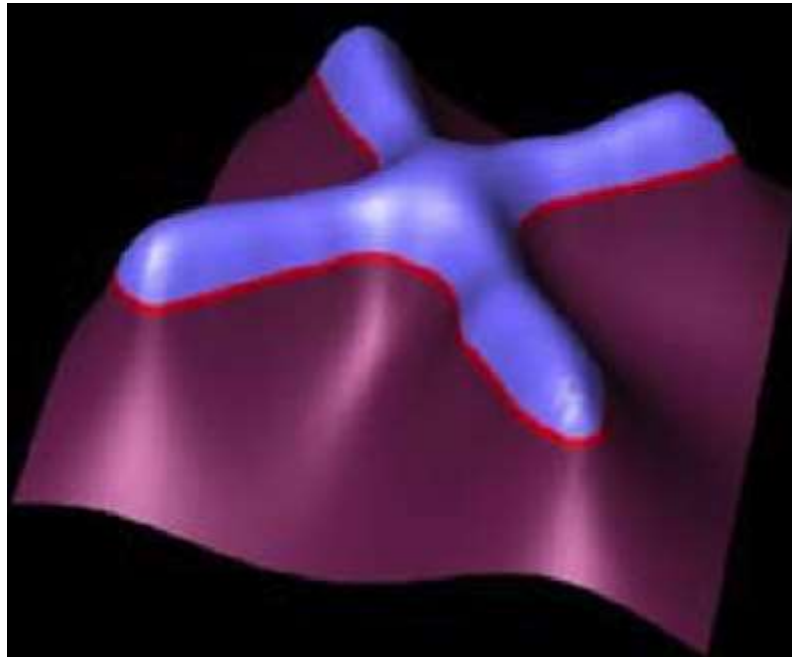
- ✓ Level-set of a function defines the shape.
- ✓ Level means function values are the same = at the same level. Traditionally, level is 0, hence 0-set of the function F is sought.



Implicit Surfaces

9 / 98

- ✓ Level-set of 2D function $F(x, y)$ defines 1D curve (embedded in 2D).



Implicit Surfaces

10 / 98

- ✓ Level-set of 3D function $F(x, y, z)$ defines 2D surface (embedded in 3D).



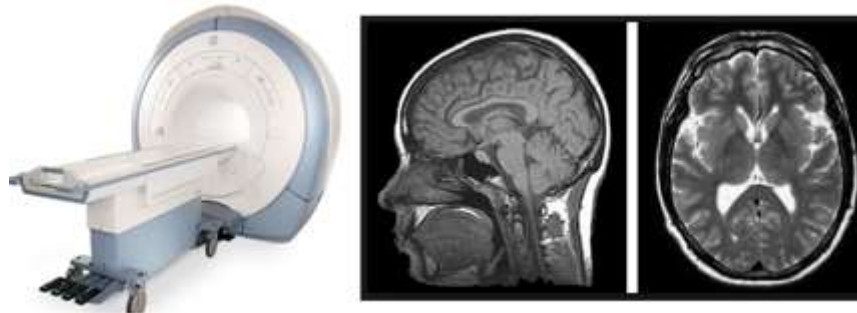
Implicit Function F

11 / 98

- ✓ Most common and natural representation is Signed Distance Function.
- ✓ Maps each 3D point x to its signed distance $F(x)$ from the surface.

$$F: \mathbb{R}^3 \rightarrow \mathbb{R}.$$

- ✓ Sign indicates inside/outside. $F(x)=0$ means x is on the surface.
- ✓ Other than Signed Distance Function (SDF), continuous intensity function from CT/MRI scans are also common. Here, we will focus on the computation of SDF as it is a geometric problem. CT/MRI devices provide intensity info with their underlying radiologic technology.



Implicit Function F

12 / 98

- ✓ Most common and natural representation is Signed Distance Function.
- ✓ Maps each 3D point \mathbf{x} to its signed distance $F(\mathbf{x})$ from the surface.

$$F: \mathbb{R}^3 \rightarrow \mathbb{R}.$$

- ✓ Sign indicates inside/outside. $F(\mathbf{x})=0$ means \mathbf{x} is on the surface.
- ✓ Other than Signed Distance Function (SDF), continuous intensity function from CT/MRI scans are also common. Here, we will focus on the computation of SDF as it is a geometric problem. CT/MRI devices provide intensity info with their underlying radiologic technology.
 - ✓ Intensity: amount of vibration caused by magnets in MRI (no radiation).
 - ✓ Intensity: x-rays (x radiation) are beamed & received along the tunnel in CT.

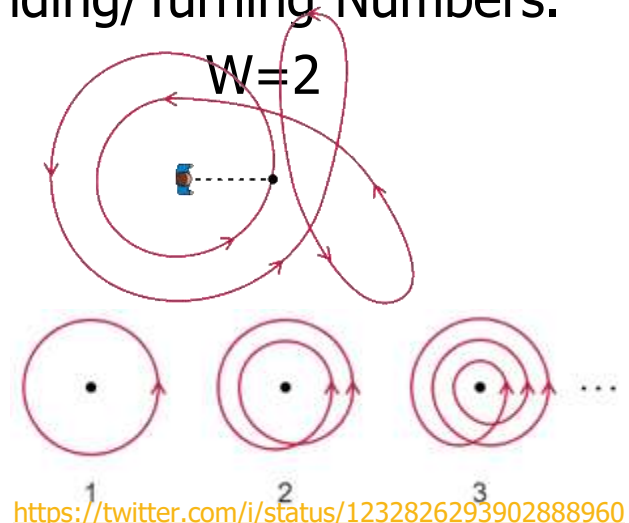
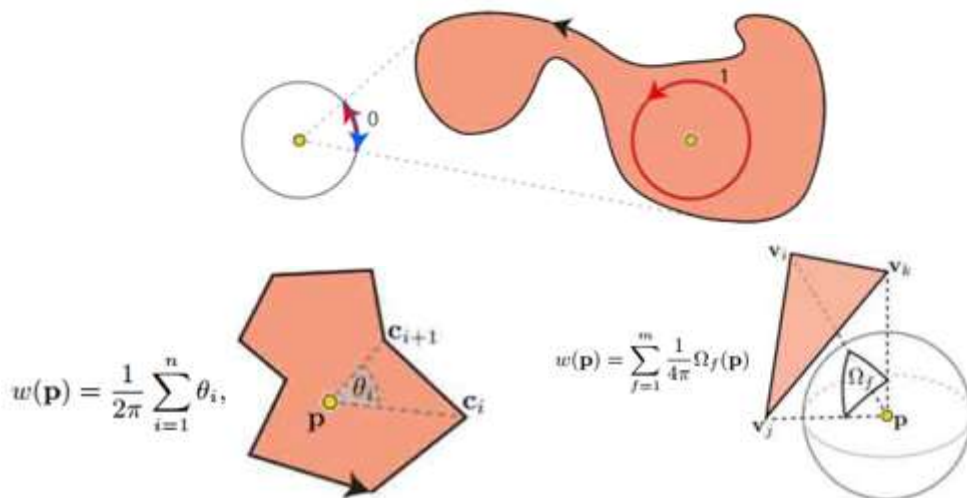
Implicit Function F

13 / 98

- ✓ Most common and natural representation is Signed Distance Function.
- ✓ Maps each 3D point x to its signed distance $F(x)$ from the surface.

$$F: \mathbb{R}^3 \rightarrow \mathbb{R}.$$

- ✓ Sign indicates inside/outside. $F(x)=0$ means x is on the surface.
- ✓ Another function for inside/outside test: Winding/Turning Numbers.



<https://twitter.com/i/status/1232826293902888960>

Implicit Function F

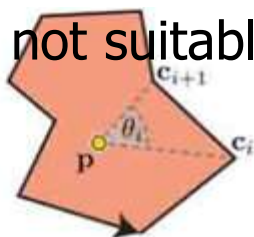
14 / 98

- ✓ Most common and natural representation is Signed Distance Function.
- ✓ Maps each 3D point x to its signed distance $F(x)$ from the surface.

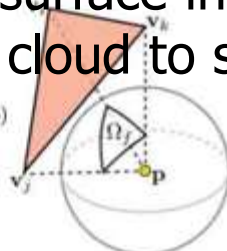
$$F: \mathbb{R}^3 \rightarrow \mathbb{R}.$$

- ✓ Sign indicates inside/outside. $F(x)=0$ means x is on the surface.
- ✓ Another function for inside/outside test: Winding/Turning Numbers.
- ✓ Originally defined for closed curves (2D) and surfaces (3D).
- ✓ Made robust to self-intersections, open boundaries, and non-manifold pieces by Jacobson et al., 2013.
- ✓ Still needs some sort of surface input (tests point in/out of a **surface**), so not suitable for point cloud to surface conversion.

$$w(\mathbf{p}) = \frac{1}{2\pi} \sum_{i=1}^n \theta_i,$$



$$w(\mathbf{p}) = \sum_{j=1}^m \frac{1}{4\pi} \Omega_j(\mathbf{p})$$



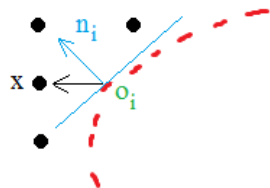
Implicit Function F

15 / 98

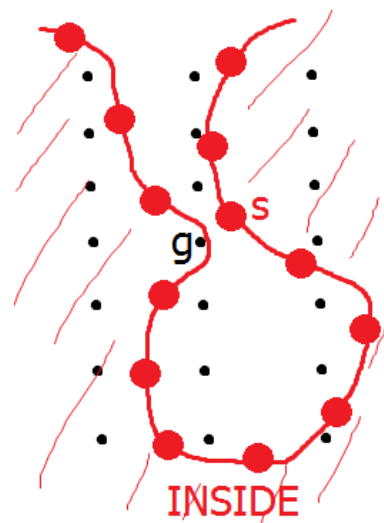
- ✓ Most common and natural representation is Signed Distance Function.
- ✓ Maps each 3D point x to its signed distance $F(x)$ from the surface.

$$F: \mathbb{R}^3 \rightarrow \mathbb{R}.$$

- ✓ Sign indicates inside/outside. $F(x)=0$ means x is on the surface.
- ✓ Another function for inside/outside test: Winding/Turning Numbers.
 - ✓ More robust than a signed distance function.
 - ✓ Although inside, grid point g will be wrongly treated as outside as it is coupled w/ the closest sample, which yields a positive signed distance (see slide 39).
 - ✓ No such problem w/ Winding Number test.



$$F(x) = (x - o_i) \cdot n_i$$



Implicit Function F

16 / 98

- ✓ Most common and natural representation is Signed Distance Function.
- ✓ Maps each 3D point x to its signed distance $F(x)$ from the surface.

$$F: \mathbb{R}^3 \rightarrow \mathbb{R}.$$

- ✓ Sign indicates inside/outside. $F(x)=0$ means x is on the surface.
- ✓ Another function for inside/outside test of a **closed surface**: ray count.
- ✓ Shooting a ray from the point and count # of times it intersects the surface: odd number means the point is inside, even means outside.



Implicit Function F

17 / 98

- ✓ Most common and natural representation is Signed Distance Function.
- ✓ Maps each 3D point x to its signed distance $F(x)$ from the surface.

$$F: \mathbb{R}^3 \rightarrow \mathbb{R}.$$

- ✓ Sign indicates inside/outside. $F(x)=0$ means x is on the surface.
- ✓ A function for unsigned distance to a point cloud: UDF vs. SDF.
- ✓ Average the sum of squared distances to K -nearest neighbors.

$$d_U(x) = \sqrt{\frac{1}{K} \sum_{p \in N_K(x)} \|x - p\|^2}$$

- ✓ Doesn't need a surface input and also pretty robust 😊.
- ✓ Can you sign this up to make it useful for Marching Cubes?
 - ✓ First do UDF (robust) and then sign it using the signs (not values) in SDF?

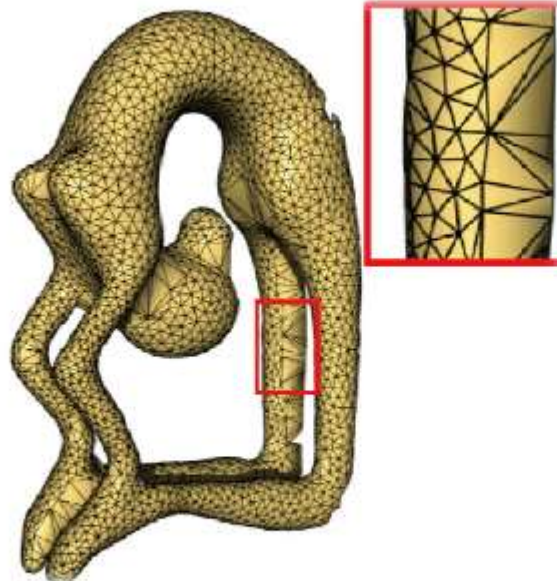
Surface Reconstruction

18 / 98

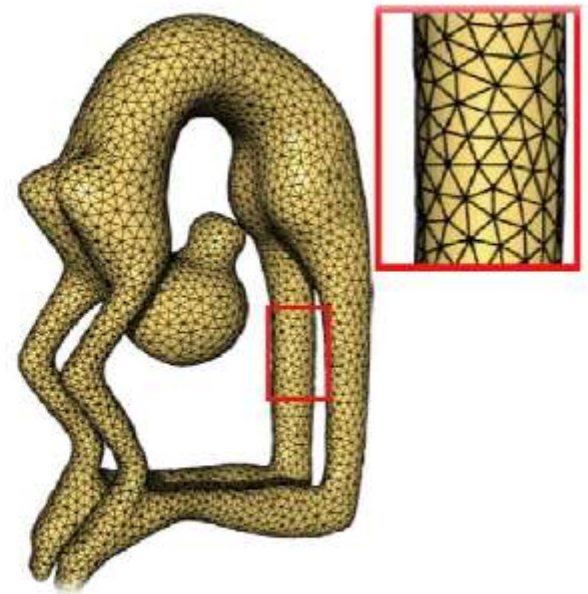
- ✓ Given a set of 3D points, compare Explicit vs. Implicit reconstruction.



Input



Explicit



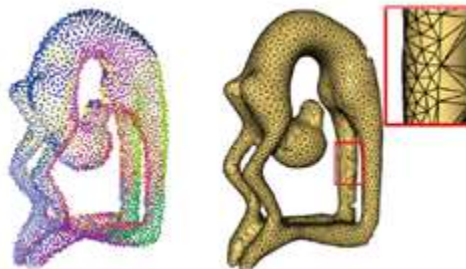
Implicit

- ✓ Implicit algorithms *approximate*, explicit ones *interpolate* input data.

Surface Reconstruction

19 / 98

- ✓ Given a set of 3D points, compare Explicit vs. Implicit reconstruction.
- ✓ Explicit reconstruction method:
 - ✓ Connect sample points by triangles (connect the dots).
 - ✓ Exact interpolation of sample points.
 - ✓ Bad for noisy or misaligned data (common in scans).
 - ✓ May lead to holes or non-manifoldness.



Surface Reconstruction

20 / 98

- ✓ Given a set of 3D points, compare Explicit vs. Implicit reconstruction.
- ✓ Implicit reconstruction method:
 - ✓ Estimate signed distance function (SDF) for each grid point (scalar-valued grid).
 - ✓ Extract level zero iso-surface from this grid (Marching Cubes).
 - ✓ Approximation of input points (better in noisy situations).
 - ✓ Manifoldness by construction.



Surface Reconstruction

21 / 98

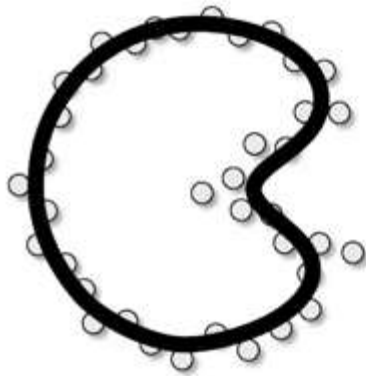
- ✓ Given a set of 3D points, compare Explicit vs. Implicit reconstruction.
- ✓ Implicit reconstruction method:
 - ✓ The distance function is interpolated and polygonalized by the marching squares/cubes algorithm.
 - ✓ The distance function is, however, based on an approximation of the input surface (via tangent planes), hence the whole process is an approximation.



Surface Reconstruction

22 / 98

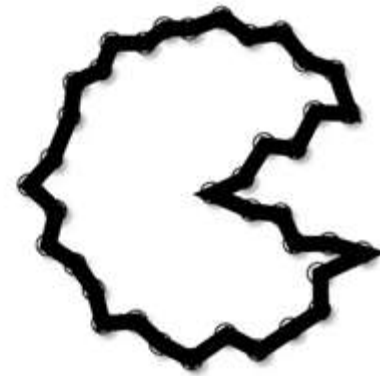
- ✓ Given a set of 3D points, compare Explicit vs. Implicit reconstruction.
- ✓ **Implicit** vs. **Explicit**:



Smooth



Piecewise Smooth



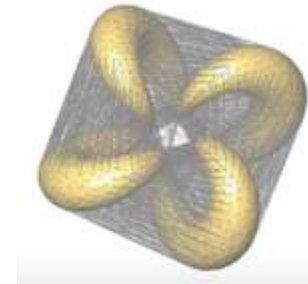
Interpolant
"Connect the Dots"

- ✓ Surface reconstruction is an ill-posed problem: want sharp corner?
- ✓ Solution: regularize via priors, e.g., smooth, piecewise smooth, simple.

Explicit Reconstruction

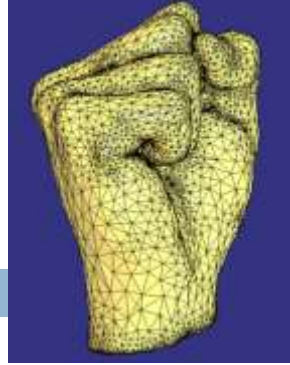


- ✓ Crust Algorithm: Since Delaunay triangulation encodes the proximity b/w the points, if points are dense enough, we should have the desired triangles within the Delaunay triangulation. Tri \rightarrow Tetrahedron in 3D.



- ✓ Compute Delaunay triangulation of the input point set.
- ✓ So we have the edges we need, but also the extras.

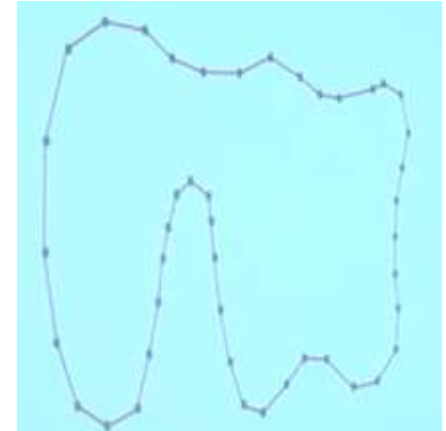




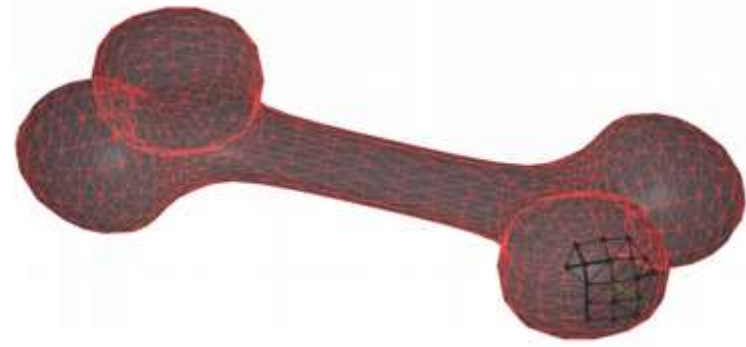
Explicit Reconstruction

24 / 98

- ✓ Crust Algorithm: Since Delaunay triangulation encodes the proximity b/w the points, if points are dense enough, we should have the desired triangles within the Delaunay triangulation. Tri \rightarrow Tetrahedron in 3D.
- ✓ Get rid of extras: Compute Voronoi diagram. Insert Voronoi vertices (pink) to the Delaunay triangulation. Keep edges b/w black dots only.



Explicit Reconstruction



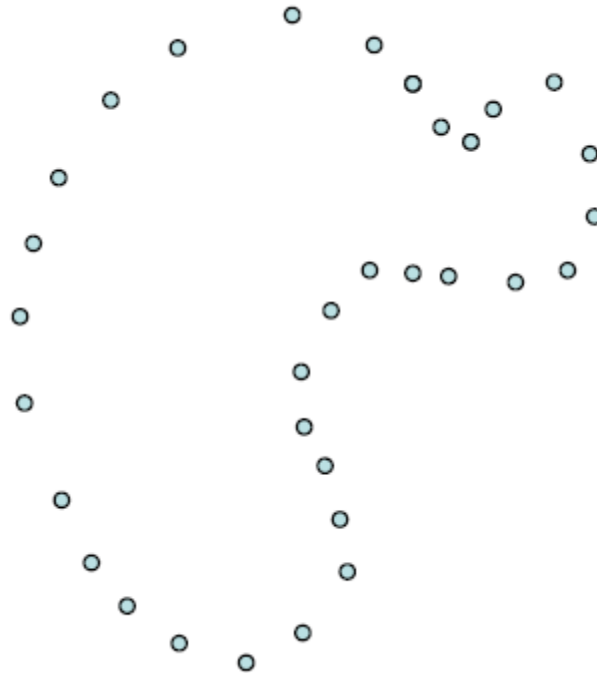
25 / 98

- ✓ Find local neighborhood L_i of each point p_i in the 3D point cloud input.
 - ✓ Closest k points (using a k-d tree).
- ✓ For each L_i compute tangent plane using PCA.
 - ✓ If the input is oriented, just take the average of all normals for the plane normal. Use the mean pnt as the plane pnt (for both oriented/unoriented).
- ✓ Project all points in L_i to the tangent plane and compute their 2D Delaunay triangulation D_i .
- ✓ Final triangulation is the composition of all these local triangulations.
- ✓ Yet another explicit algorithm is Poisson Reconstruction.

Implicit Reconstruction

26 / 98

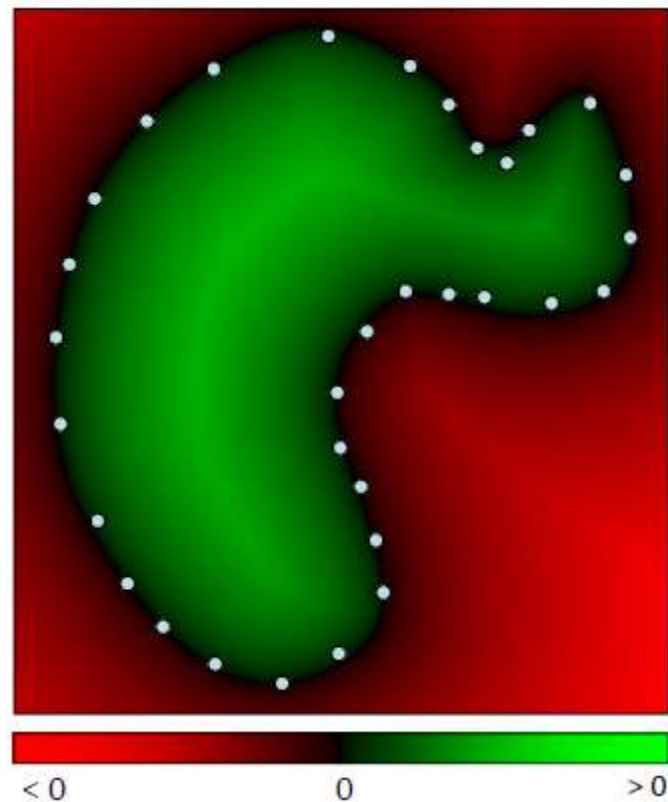
- ✓ Implicit function approach.



Implicit Reconstruction

27 / 98

- ✓ Implicit function approach.

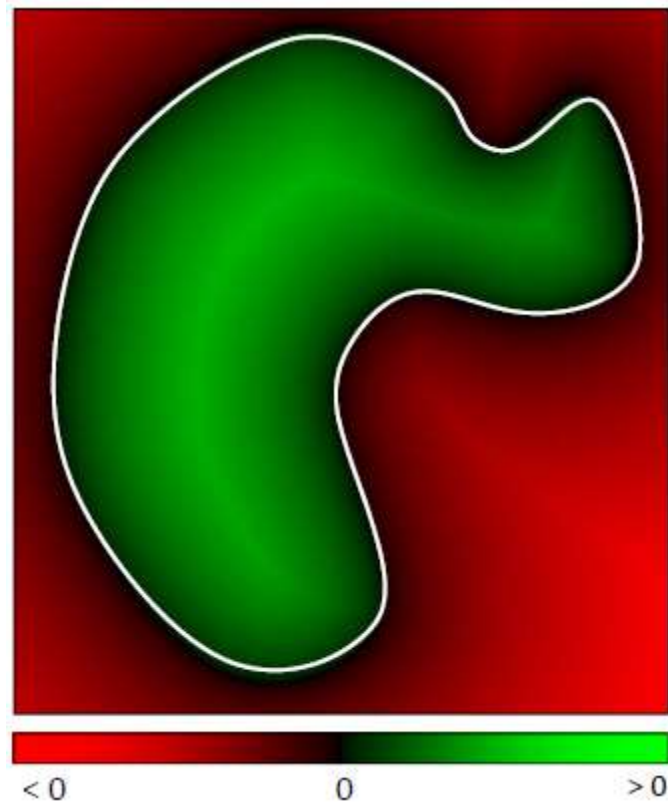


- ✓ Define a function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$ with value < 0 outside, > 0 inside, $= 0$ on.

Implicit Reconstruction

28 / 98

- ✓ Implicit function approach.



- ✓ Extract the zero-set isosurface, i.e., $S = \{x \text{ in } \mathbb{R}^3 / F(x) = 0\}$.

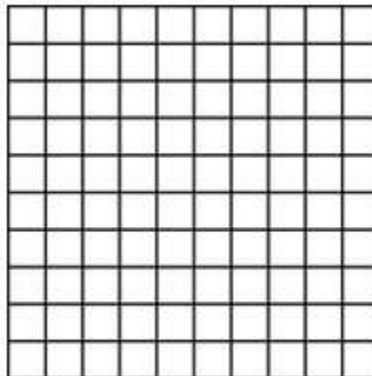
Implicit Reconstruction

29 / 98

- ✓ Implicit function: Signed distance function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$.
- ✓ Construction algorithm.
 - ✓ Input: point samples representing the object to be reconstructed.



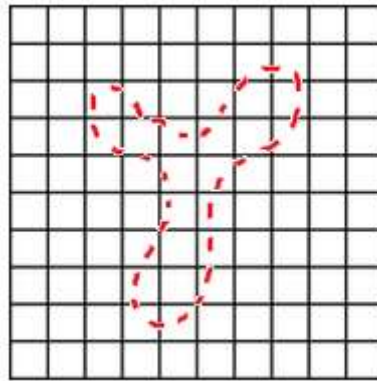
- ✓ Output: F value at each grid point, i.e., signed distance of that grid pnt to the sampled surface.



Implicit Reconstruction

30 / 98

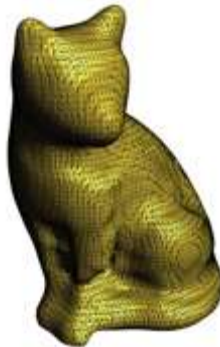
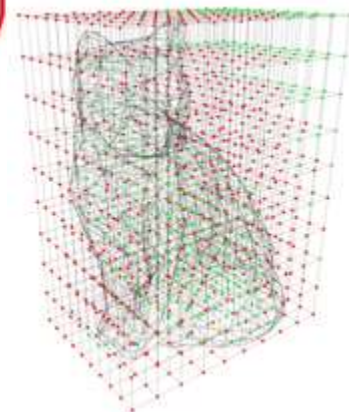
- ✓ So we need a grid that encapsulates the sampled surface.



- ✓ Uniform 2D grid.
 - ✓ Mesh to be extracted: 1D (closed) curve →



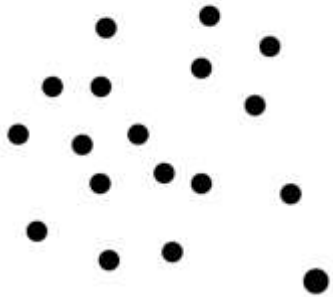
- ✓ Uniform 3D grid.
 - ✓ Mesh to be extracted: 2D (closed) surface →→→→



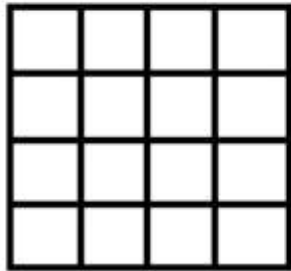
Implicit Reconstruction

31 / 98

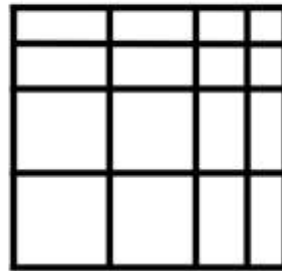
✓ Other grid types.



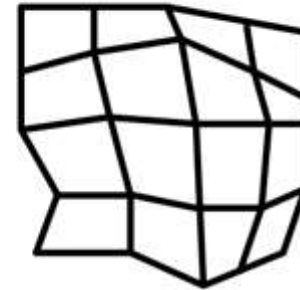
scattered



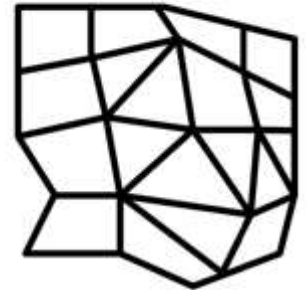
uniform



rectilinear



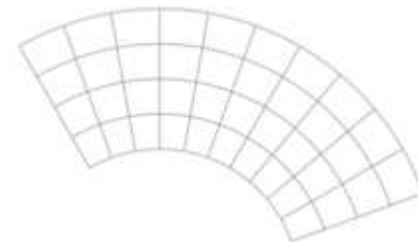
structured



unstructured

Not a grid at all: no connectivity.

Uniform aka regular grid.

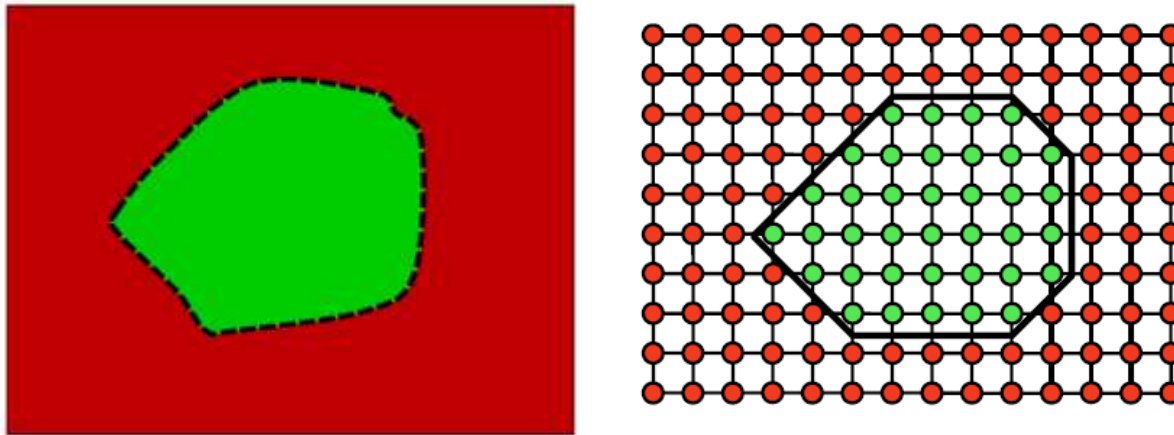


curvilinear

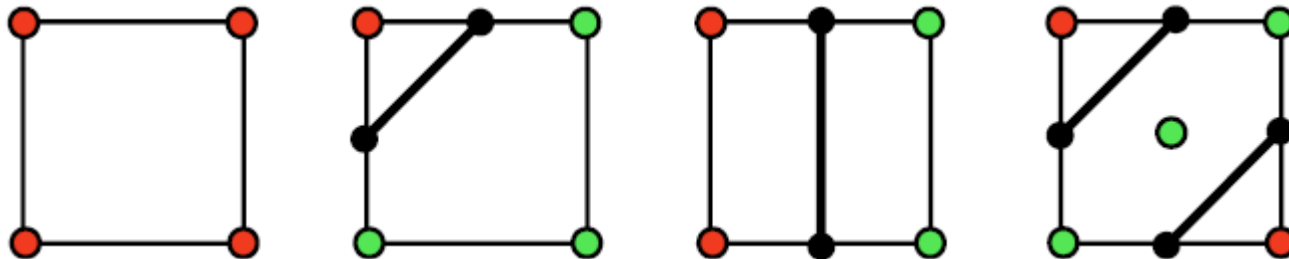
Marching X

32 / 98

- ✓ Mesh extraction idea.
- ✓ **Marching Squares (2D)**, Marching Cubes (3D).



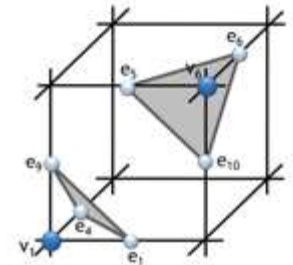
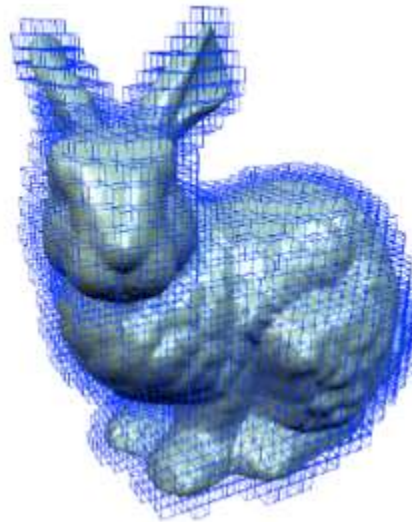
- ✓ 2^4 cases. May be reduced by symmetry, e.g., all red and all gre. equal.



Marching X

33 / 98

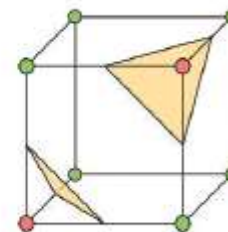
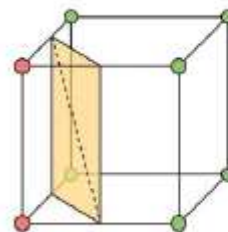
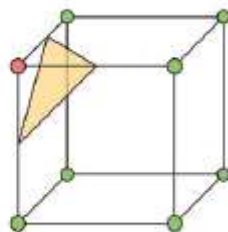
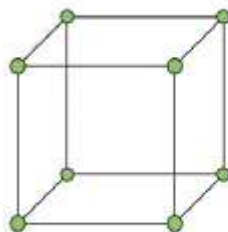
- ✓ Mesh extraction idea.
- ✓ Marching Squares (2D), [Marching Cubes \(3D\)](#).



- ✓ 2^8 cases. Sym. applies. Fast look up: use 8-bit case index: index =

0	0	1	0	0	0	0	1
---	---	---	---	---	---	---	---

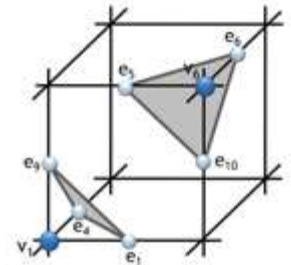
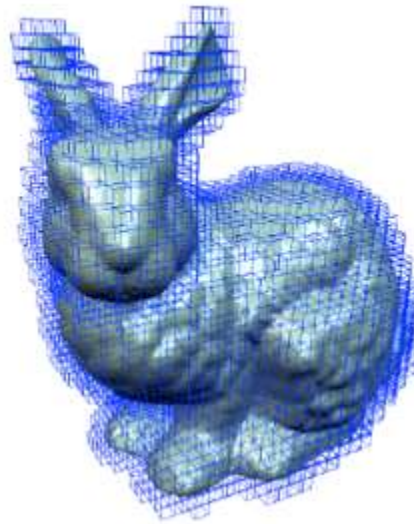
 = 33



Marching X

34 / 98

- ✓ Mesh extraction idea.
- ✓ Marching Squares (2D), [Marching Cubes \(3D\)](#).



- ✓ 2^8 cases. Sym. applies. Fast look-up: use 8-bit case index: index =

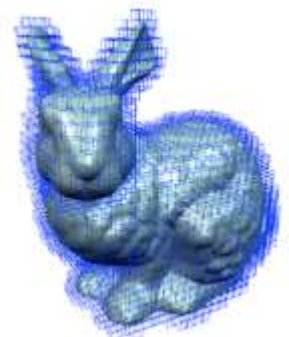
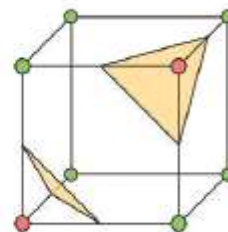
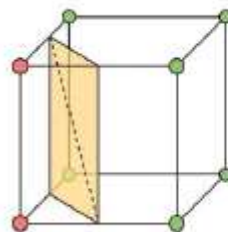
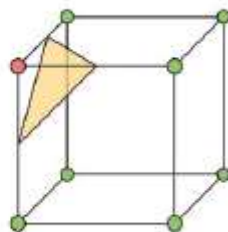
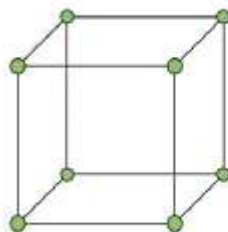
0	0	1	0	0	0	0	1
---	---	---	---	---	---	---	---

 = 33.
- ✓ Case index brings connectivity from the look-up table.
 - ✓ Cut edges are e1, e4, e5, e6, e9, e10.
 - ✓ Output triangles are (e1, e9, e4), (e5, e10, e6).

Marching X

35 / 98

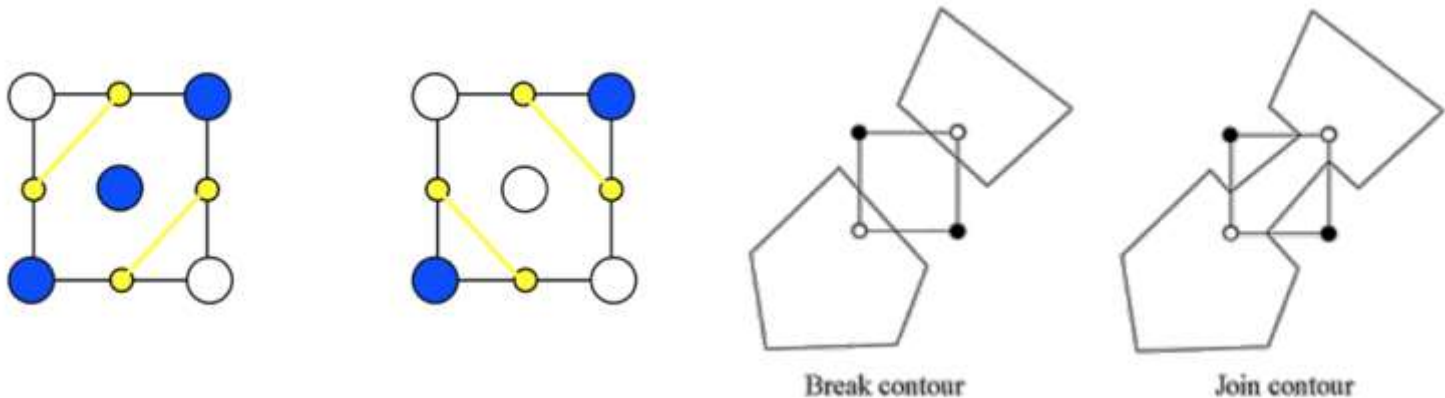
- ✓ Steps for Marching Squares/Cubes
 - ✓ Discretize space (use a regular grid enclosing the surface).
 - ✓ Evaluate signed distance function on grid.
 - ✓ Classify grid points (inside/outside w.r.t. surface).
 - ✓ Classify grid edges (one endpoint inside, one outside).
 - ✓ Compute intersections.
 - ✓ Connect intersections.



Marching X

36 / 98

- ✓ Steps for Marching Squares/Cubes
 - ✓ Discretize space (use a regular grid enclosing the surface).
 - ✓ Evaluate signed distance function on grid.
 - ✓ Classify grid points (inside/outside w.r.t. surface).
 - ✓ Classify grid edges (one endpoint inside, one outside).
 - ✓ Compute intersections.
 - ✓ Connect intersections (ambiguities may arise).

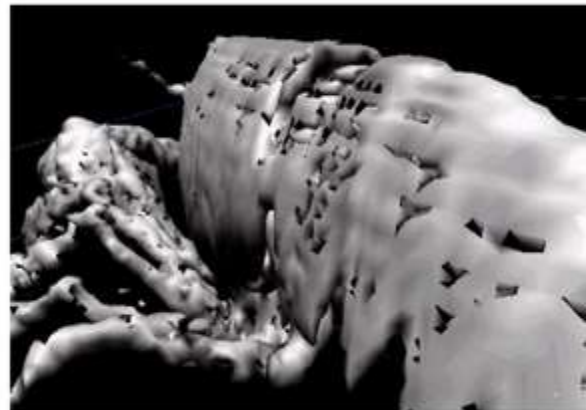
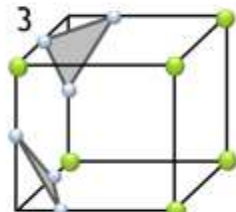
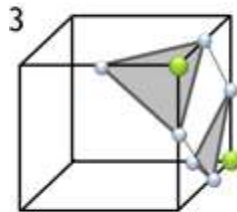


- ✓ Handle by subsampling inside a cell, or randomly picking 1 of the 2 possibilities.

Marching X

37 / 98

- ✓ Steps for Marching Squares/Cubes
 - ✓ Discretize space (use a regular grid enclosing the surface).
 - ✓ Evaluate signed distance function on grid.
 - ✓ Classify grid points (inside/outside w.r.t. surface).
 - ✓ Classify grid edges (one endpoint inside, one outside).
 - ✓ Compute intersections.
 - ✓ Connect intersections (ambiguities may arise).

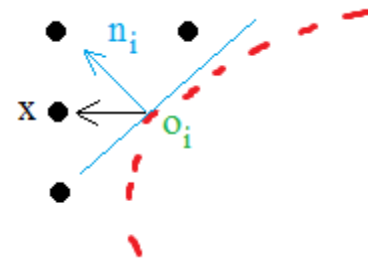
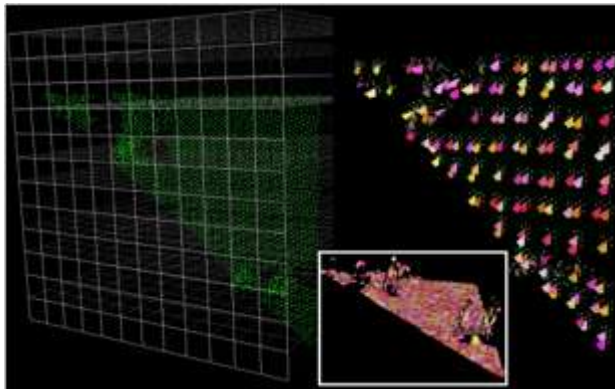


- ✓ If random pick, then make choices consistently or this happens. Fixed:

Signed Distance Function (SDF)

38 / 98

- ✓ Back to the construction of the Signed Distance Function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$.
- ✓ Input: sample points.
- ✓ Output: F value at each grid point.
- ✓ Algo:
 - ✓ Associate a tangent plane with each sample point.
 - ✓ Why? Tangent planes are local linear approximations to the surface.



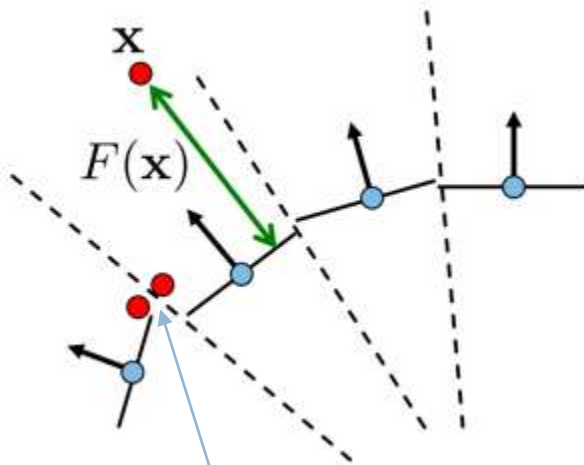
- ✓ Compute $F(x) = (x - o_i) \cdot n_i$

SDF

39 / 98

- ✓ Back to the construction of the Signed Distance Function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$.
- ✓ Input: sample points.
- ✓ Output: F value at each grid point.

✓

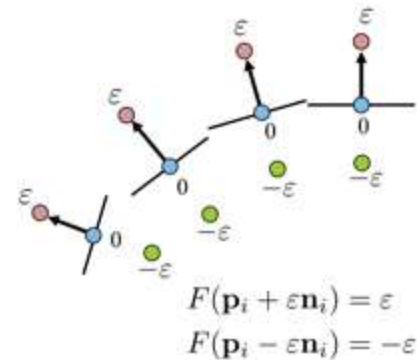


✓ $F(x) = (x - o_i) \cdot n_i$

- ✓ $F(x)$ is discontinuous. How to fix? Hint: consider off-set constraints.

- ✓ RBF, MLS, DeepSDF, <your idea here>.

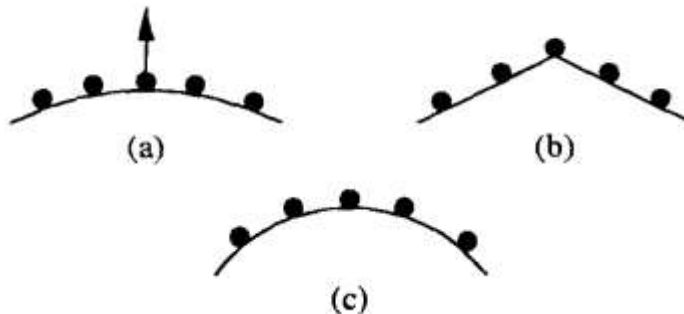
- ✓ Reconstruction and representation of 3D objects with radial basis functions, 2001.
- ✓ Interpolating and Approximating Implicit Surfaces from Polygon Soup, 2004.
- ✓ DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, 2019.



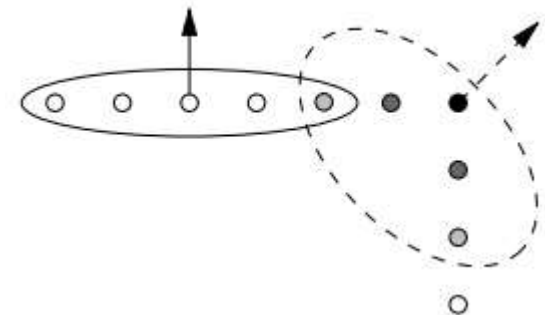
SDF

40 / 98

- ✓ Tangent plane for sample point s is found as follows:
 - ✓ Find k -nearest neighbors of s .
 - ✓ Apply principle component analysis on this set of close points.
 - ✓ Find the covariance matrix of the set.
 - ✓ Take the eigenvectors of this matrix as the principal axes.
 - ✓ Use the third best axis (w/ smallest eigenvalue) as the normal of the tangent plane, i.e. n_i .
 - ✓ Use the mean of this point set as the centering point of the plane, i.e., o_i .
 - ✓ The smaller the 3rd eigenvalue, the more confident the plane is, a is confident, b, c are not.
($\lambda_3/\lambda_1 < \Gamma$)



Darker = more ambiguous



SDF

41 / 98

- ✓ Tangent planes have arbitrary normal signs; need a normal-orienter for smooth/consistent normals across the point set.
- ✓ MST-based: Surface Reconstruction from Unorganized Points, 1992.
- ✓ Ray cast-based: A Simple Method for Correcting Facet Orientations, '15.
- ✓ BFS-based: breadth-first search to enforce a consistent facet orientation.
 - ✓ Last two are designed for surface/mesh inputs, thus not directly applicable.



k-d Tree for Nearest Neighbors (NN)

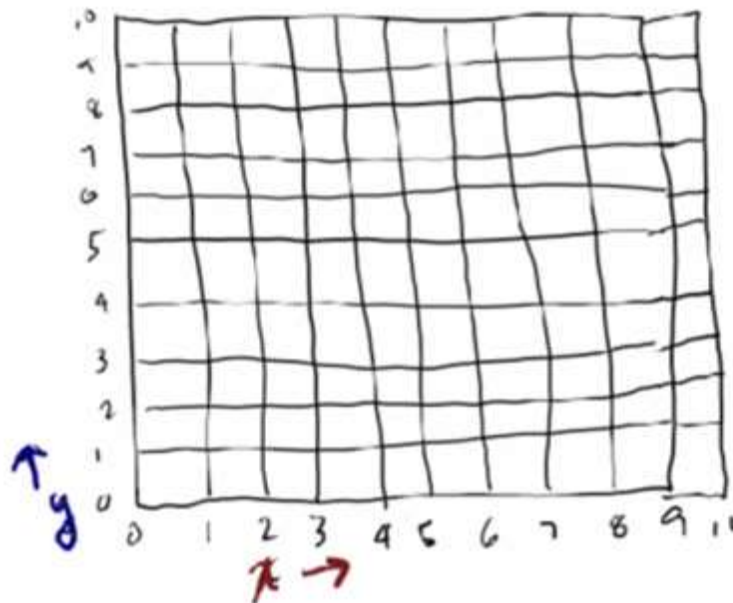
42 / 98

- ✓ *k*-nearest points.
- ✓ Brute force: $O(k * N)$ when we have N samples. // 10^3 steps
- ✓ Can be done in $O(k * \lg N)$ using *k*-d trees. // 10 steps.
- ✓ *k*-d tree: cells are axis-aligned bounding boxes.
 - ✓ *k*-d dimensional binary search tree.
 - ✓ Not to be confused w/ the *k* in *k*-nearest neighbors.

k-d Tree for NN

43 / 98

- ✓ k-d tree insertion. k = # of dimensions = 2 in example below.
- ✓ Idea: every time you go down in tree, you toggle decision-making dimension.



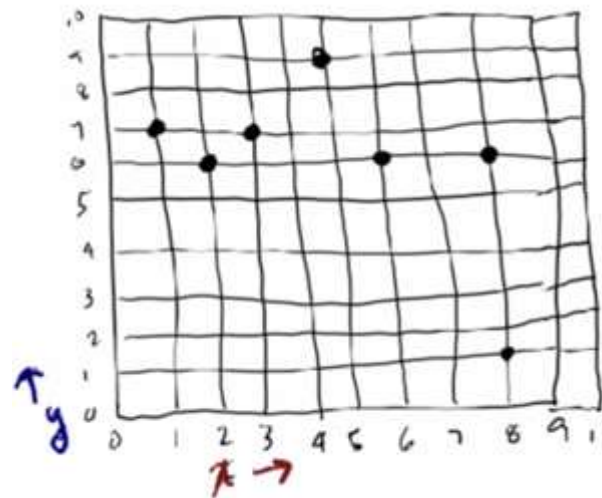
Data
3,7
8,1
6,6
2,6
1,7
8,6
5,9

k-d Tree for NN

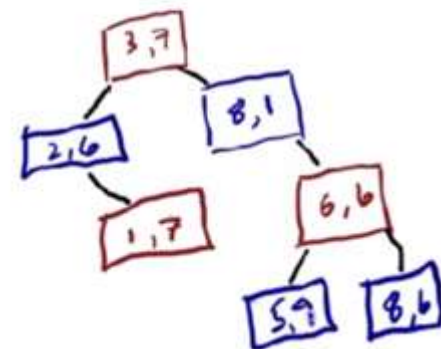
44 / 98

- ✓ k-d tree insertion. $k = \#$ of dimensions = 2 in example below.
- ✓ Idea: every time you go down in tree, you toggle decision-making dimension.

- 1) 3,7 is inserted
- 2) $8 > 3$ (1st dimension) of 3,7, so put 8,1 to right; colored in blue 'cos if may have to make a decision at this point, i'll be deciding at the 2nd dim (red means 1st dim)
- 3) $6 > 3$ go right, $6 > 1$ go right
- 4) $2 < 3$ go left
- 5) $1 < 3$ go left, $7 > 6$ go right
- 6) $8 > 3$ go right, $6 > 1$ go right, $8 > 6$ go right
- 7) $5 > 3$ go right, $9 > 1$ go right, $5 < 6$ go left



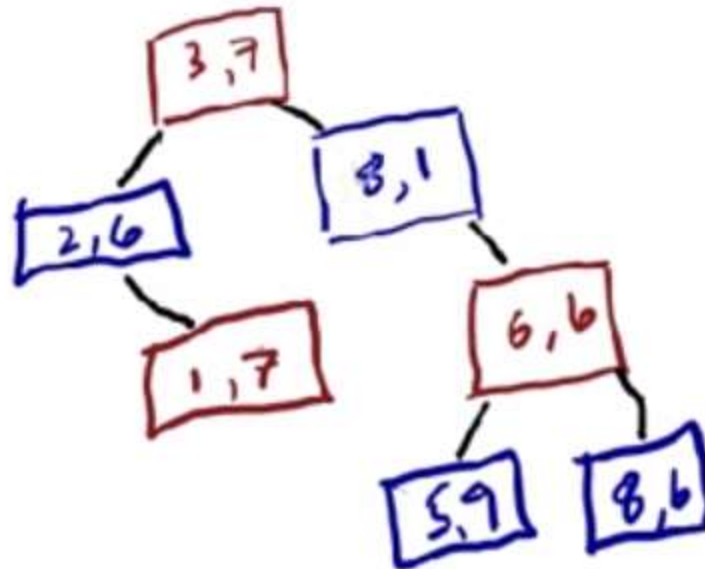
Data
3,7 ✓
8,1 ✓
6,6 ✓
2,6 ✓
1,7 ✓
8,6 ✓
5,9 ✓



k-d Tree for NN

45 / 98

- ✓ k-d tree search operation. Search (5, 2) below.
- ✓ $5 > 3$ go right, $2 > 1$ go right, $5 < 6$ go left, $2 < 9$ look left, not found.

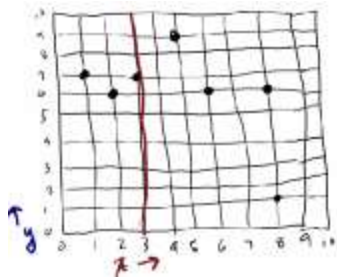


k-d Tree for NN

46 / 98

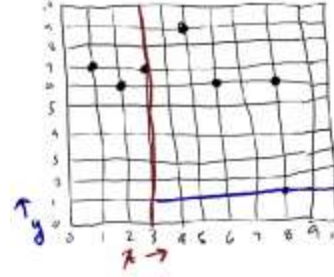
- ✓ k-d tree partitioning.
- ✓ Going left or right means partitioning the space.

(3,7) inserted



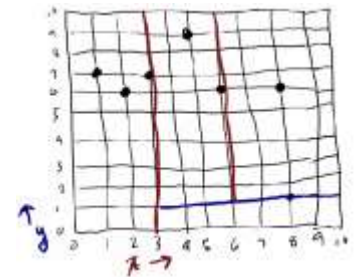
partition the space
into 2 axis-aligned
bounding boxes, i.e.,
everything ≤ 3 to left
everything > 3 to right.

(8,1) inserted



everything ≤ 1
everything > 1
(w.r.t. y dimension)

(6,6) right of the
3-line, top of the
1-line:

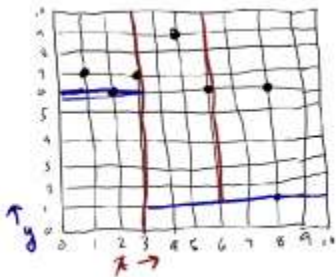


k-d Tree for NN

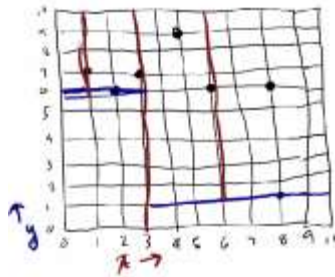
47 / 98

- ✓ k-d tree partitioning.
- ✓ Going left or right means partitioning the space.

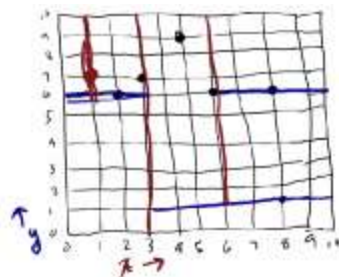
(2,6) inserted



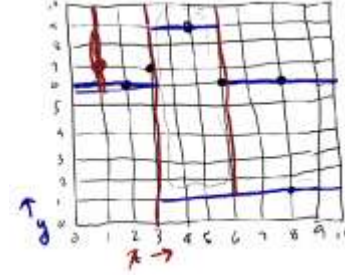
(1,7) inserted



(8,6) inserted



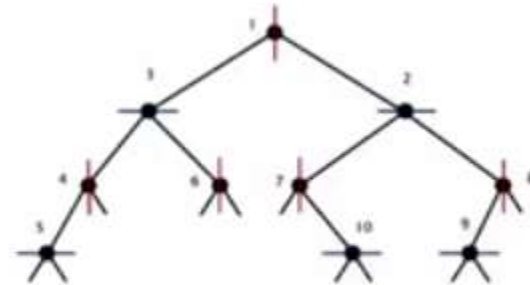
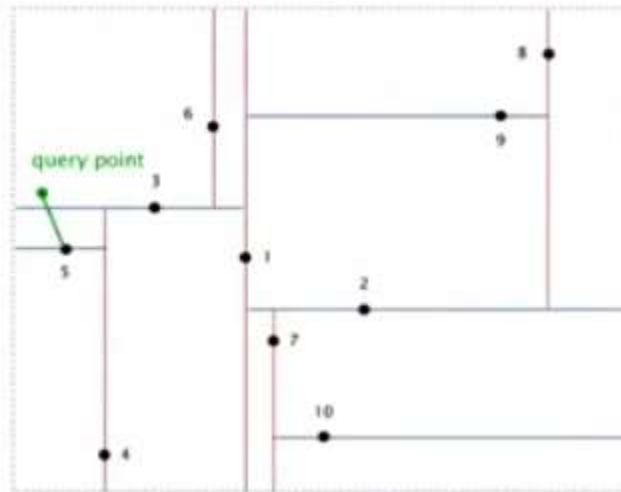
(5,9) inserted



k-d Tree for NN

48 / 98

- ✓ k-d tree nearest-neighbor to a query point (see Note below).
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.

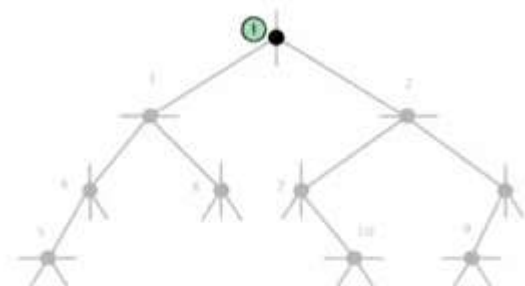
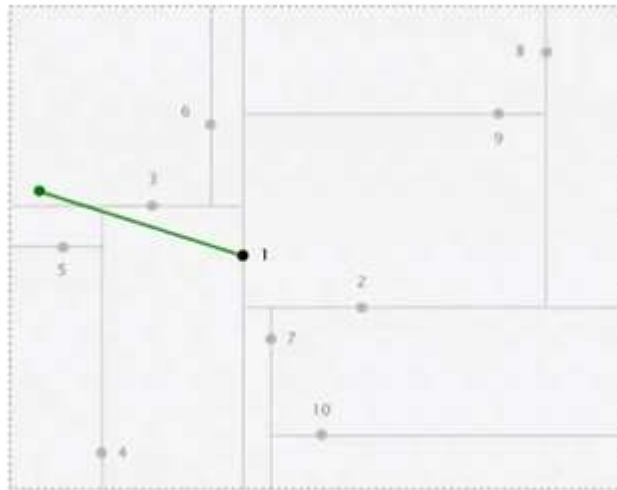


- ✓ Our algo should return point 5 (as the nearest one to the query).

k-d Tree for NN

49 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



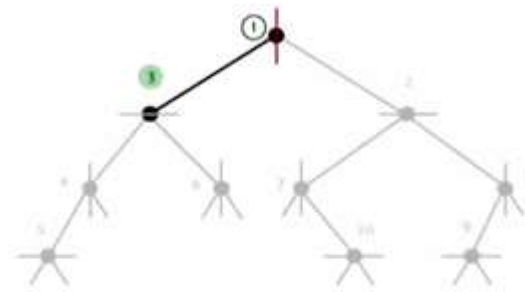
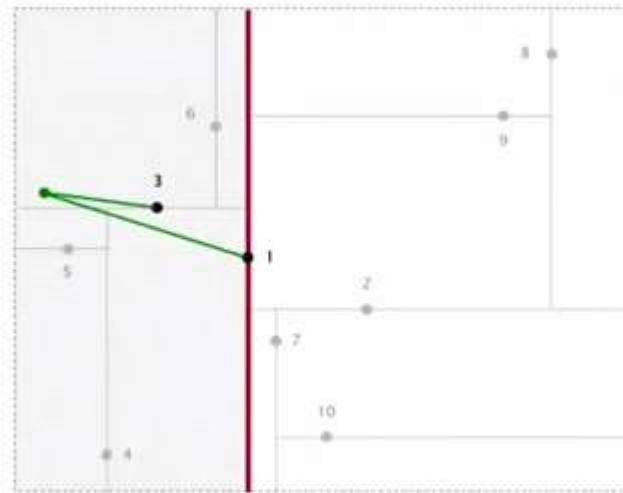
search root node
compute distance from query point to 1
(update champion nearest neighbor)

- ✓ Go towards query point, i.e., left. (there might be a closer pnt than 1 in the right; at least we think so now).

k-d Tree for NN

50 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



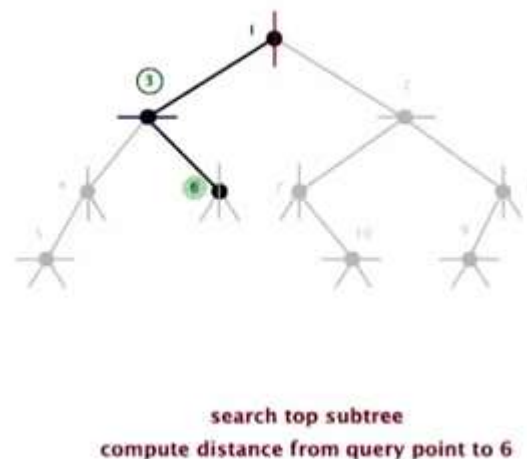
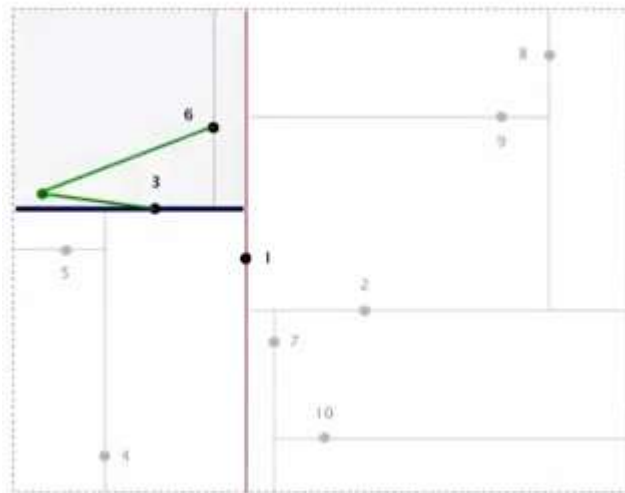
search left subtree
compute distance from query point to 3
(update champion)

- ✓ When recursion gets back to pnt 1, we won't search the right subtree 'cos there could be no point closer to query than 3. //cutting out 😊

k-d Tree for NN

51 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.

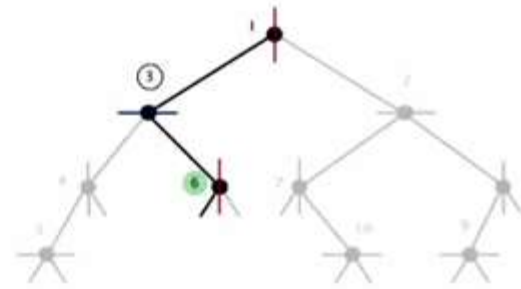
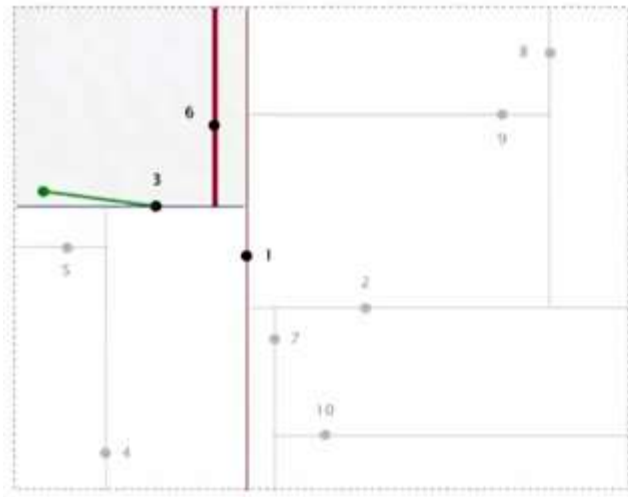


- ✓ Going towards the query point, so checking the top of 3. Pnt 6 is not better so do not update the champion.

k-d Tree for NN

52 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



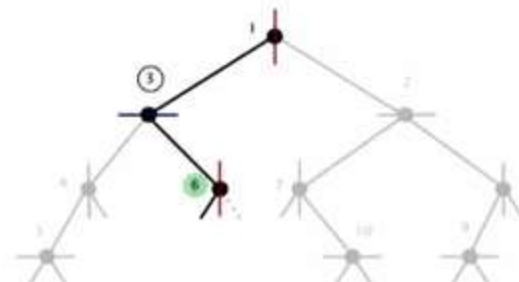
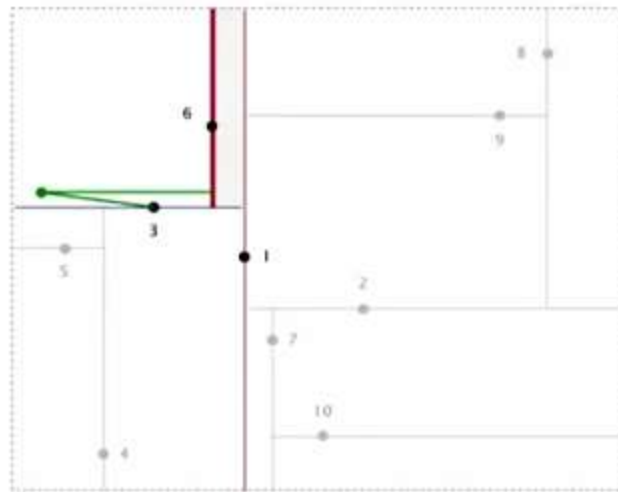
search left subtree
return since empty

- ✓ Check left subtree of pnt 6 (going towards query), empty, so do nothing.

k-d Tree for NN

53 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



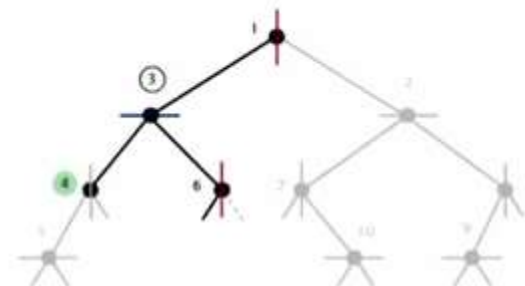
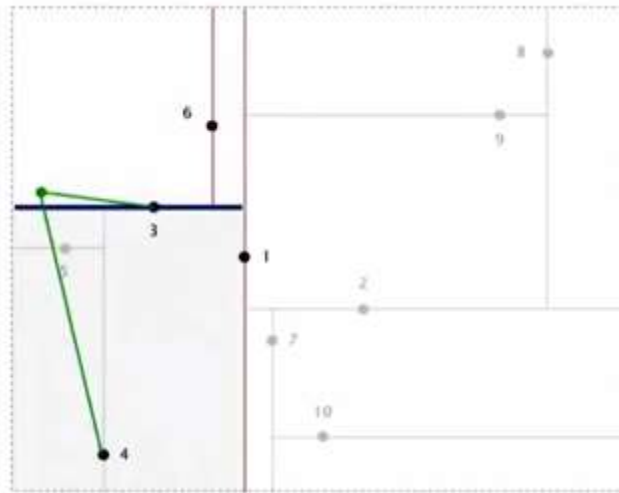
search right subtree
prune since nearest neighbor
can't be in

- ✓ Do not check the right subtree of pnt 6 'cos they can't be closer than 3.

k-d Tree for NN

54 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.

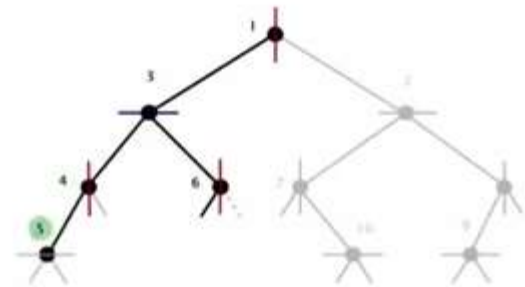
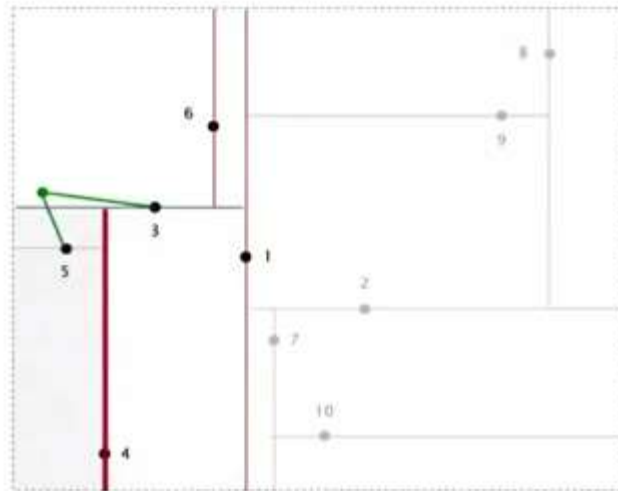


- ✓ Recursion checks the bottom subtree of pnt 3, which takes us to pnt 4, which is not closer; so 3 is still the champion.

k-d Tree for NN

55 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



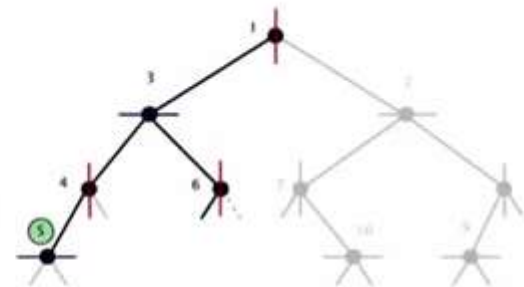
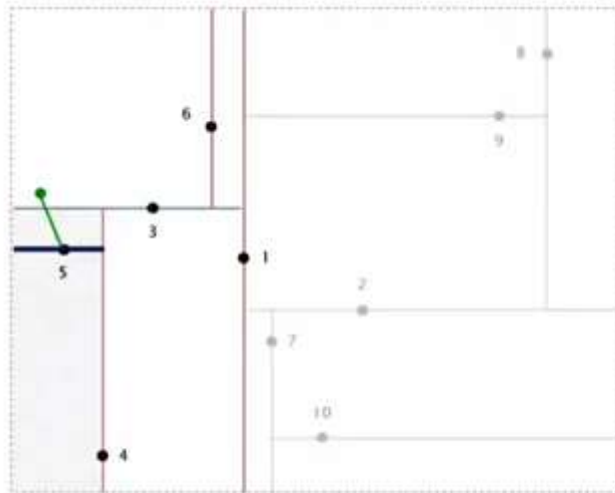
search left subtree
compute distance from query point to 5
(update champion)

- ✓ Check left subtree of pnt 4 (towards query), where we have pnt 5, the new champion.

k-d Tree for NN

56 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



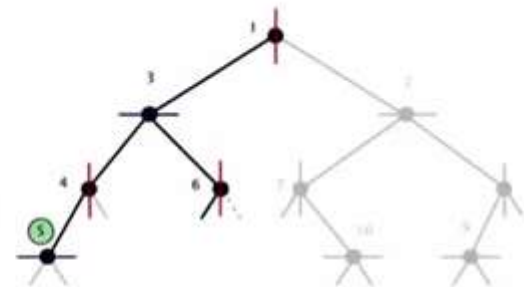
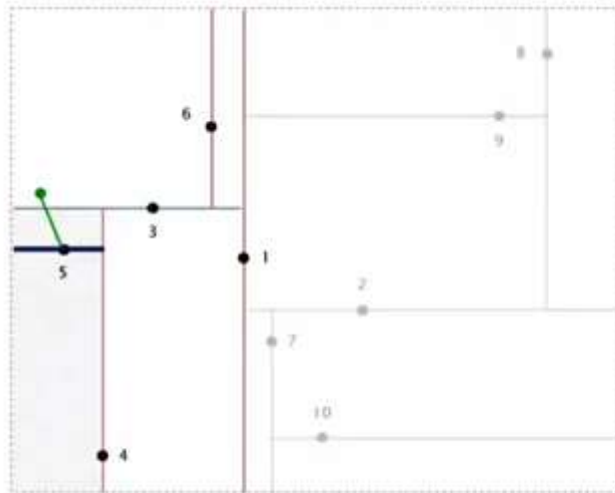
query point is above splitting line
search top subtree first

- ✓ Check top of pnt 5 (no pnts there), do nothing.

k-d Tree for NN

57 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



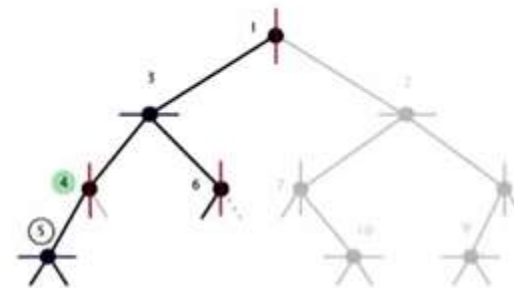
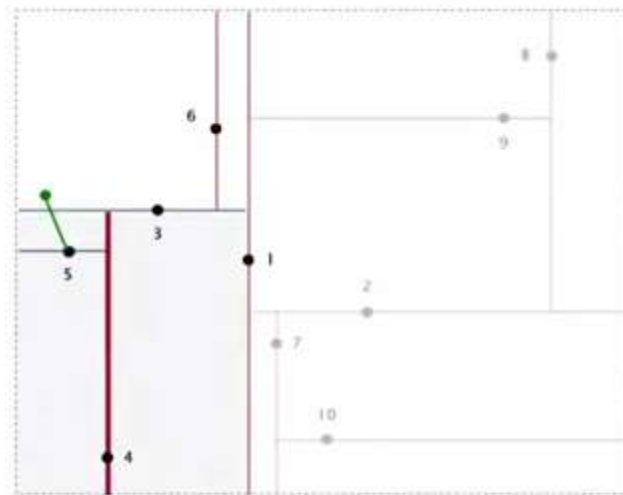
query point is above splitting line
search top subtree first

- ✓ Check bottom of pnt 5 (no pnts there), do nothing.

k-d Tree for NN

58 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



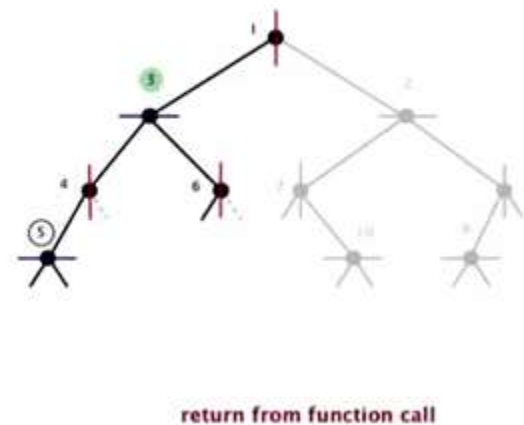
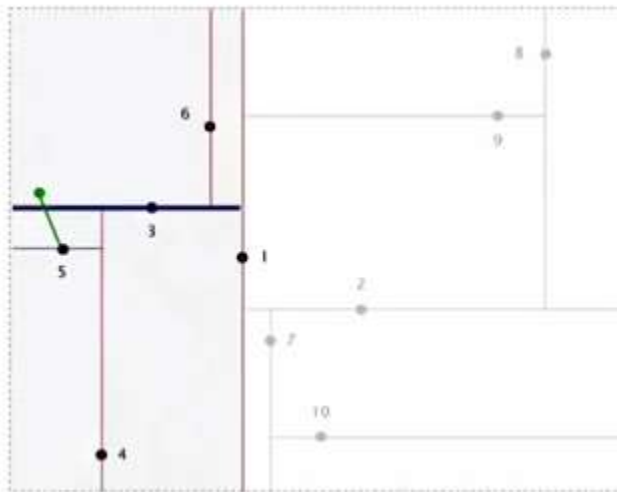
return from function call
search right subtree next

- ✓ Recursion comes back to the right subtree of pnt 4, but cut it out (no search) 'cos there can't be a closer pnt there than pnt 5.

k-d Tree for NN

59 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.

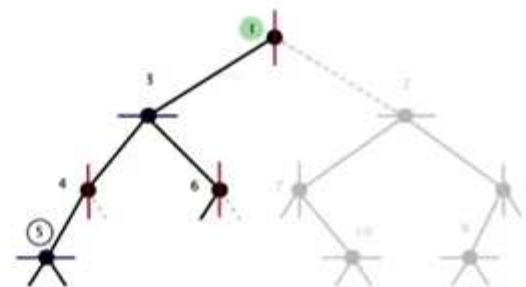
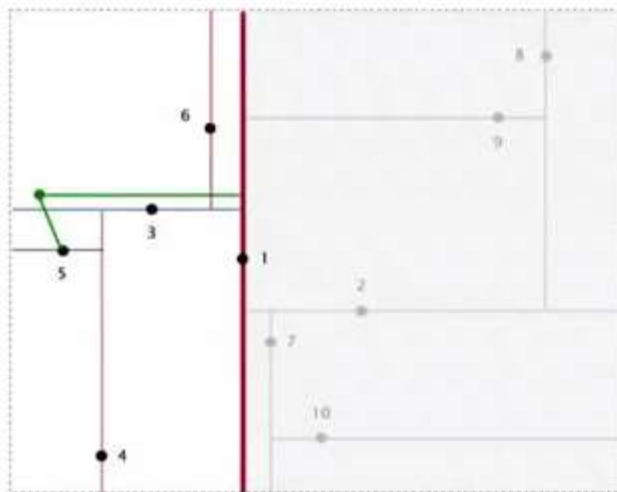


- ✓ Recursion comes back to pnt 3, whose both subtrees are handled.

k-d Tree for NN

60 / 98

- ✓ k-d tree nearest-neighbor to a query point.
- ✓ Idea: As we get closer to the query point, we are cutting out all the subtrees that are away.



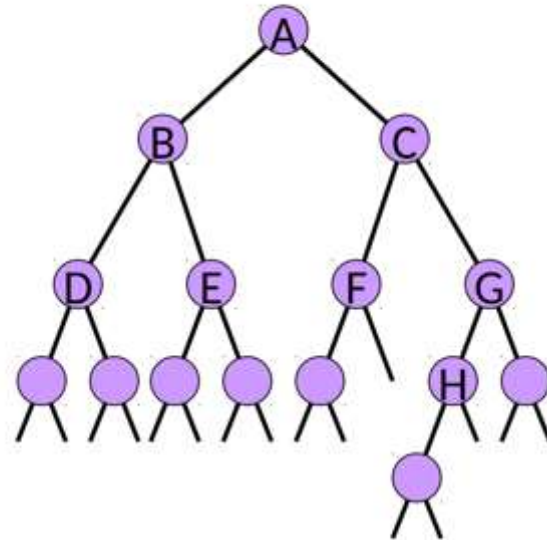
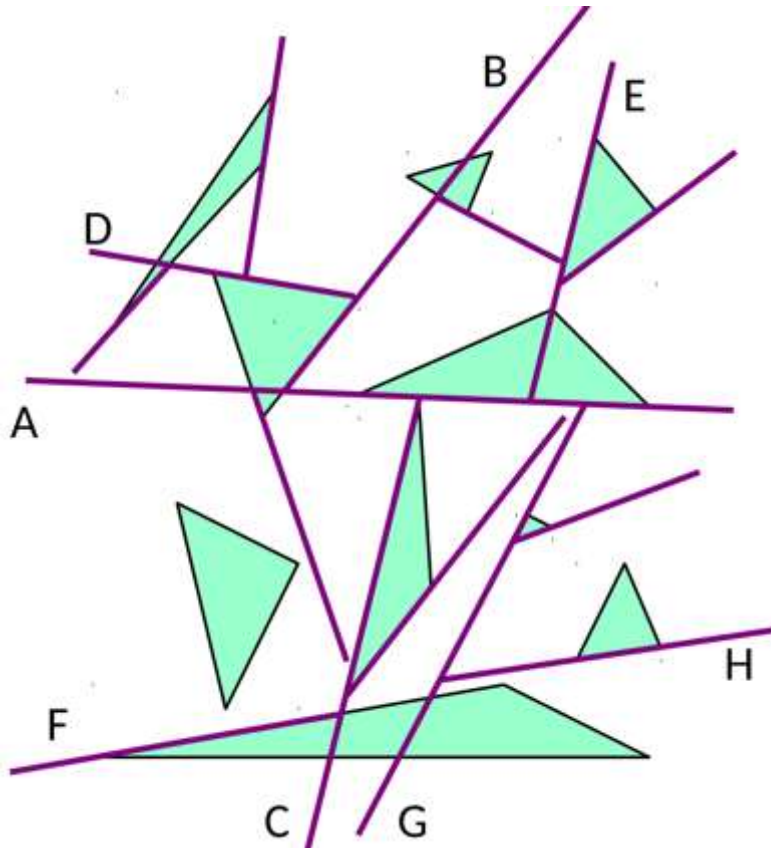
search right subtree
prune since nearest neighbor
can't be in

- ✓ Recursion comes back to pnt 1, right subtree is next, but cut it out (no search) 'cos there can't be a closer pnt there than pnt 5.

k-d Tree Alternative: BSP Tree

61 / 98

- ✓ k-d tree: cells are axis-aligned bounding boxes.
- ✓ BSP-tree: cells are arbitrarily shaped; splitting lines/planes through edge/triangles of the set to be stored. BSP: Binary Space Partitioning.

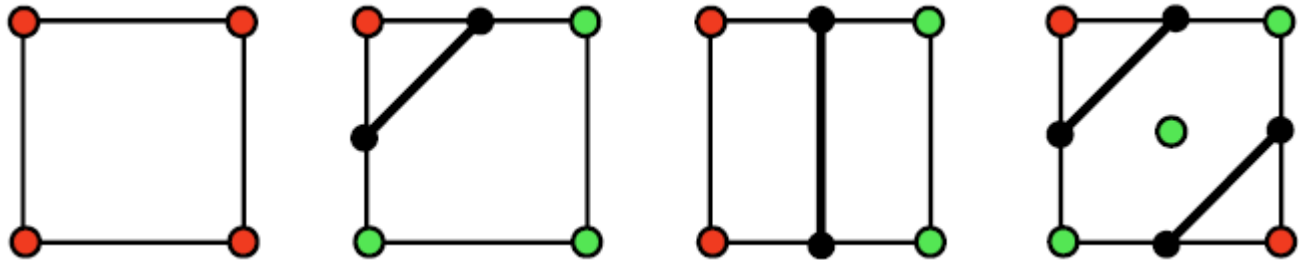


Marching X

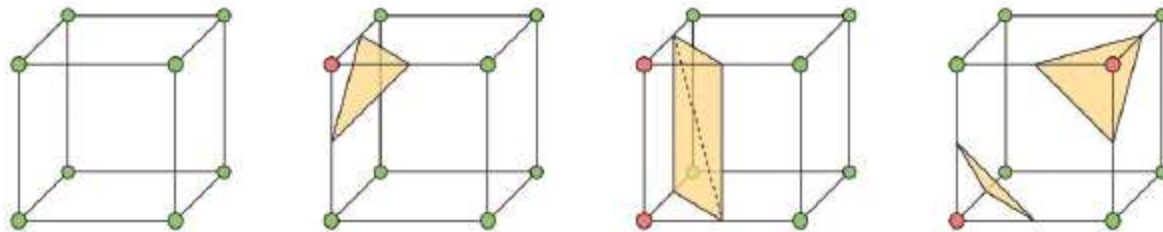
62 / 98

- ✓ We efficiently (k-d tree) computed signed distance function F .
- ✓ The only thing left is to find the intersection point on the square/cube edge:

✓ Square:



✓ Cube:

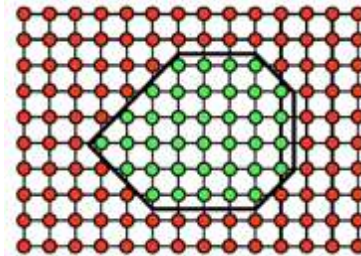


Marching X

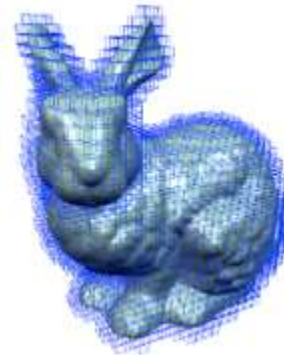
63 / 98

- ✓ We efficiently (k-d tree) computed signed distance function F .
- ✓ The only thing left is to find the intersection point on the square/cube edge:

- ✓ Square:



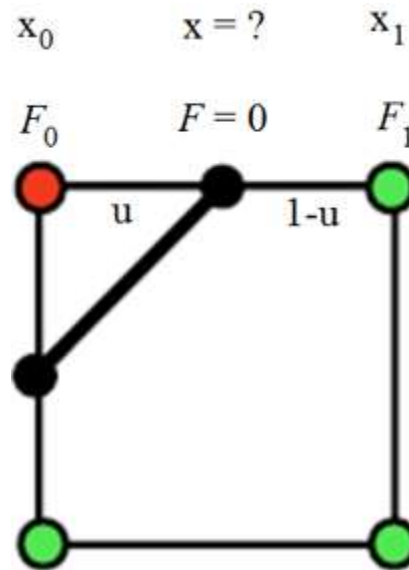
- ✓ Cube:



Marching X

64 / 98

- ✓ We efficiently (k-d tree) computed signed distance function F .
- ✓ The only thing left is to find the intersection point x on the square/cube edge:

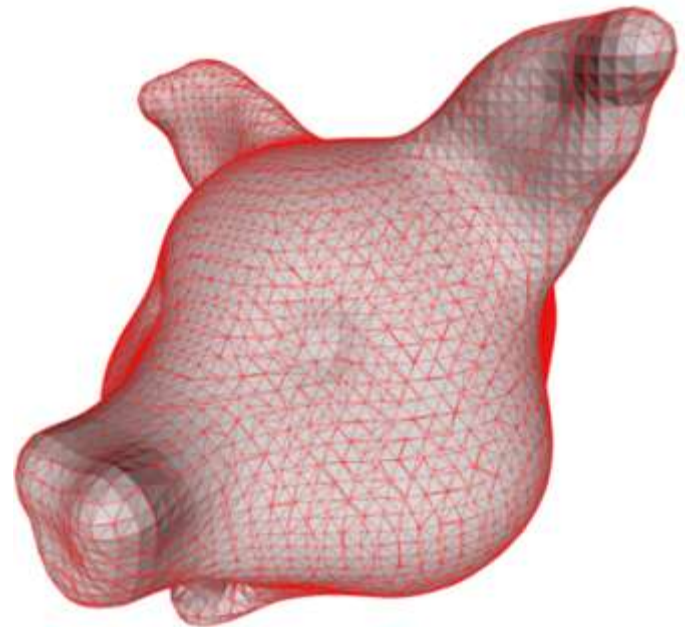
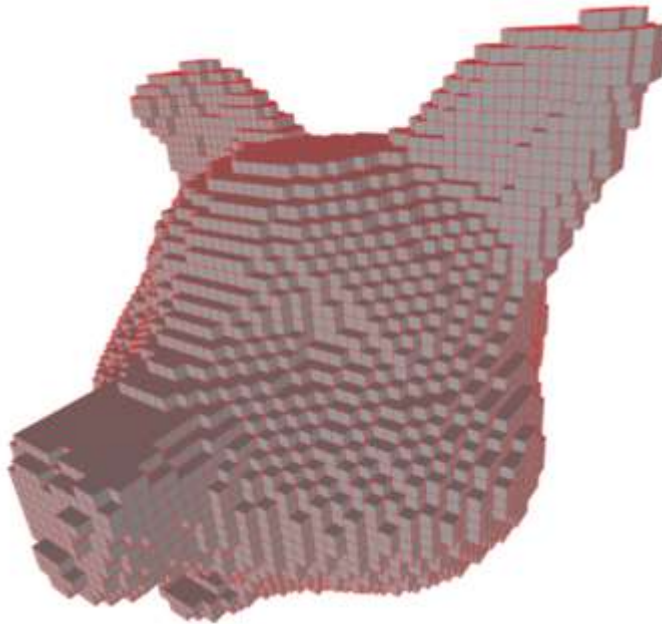


- ✓ $F = F_0 * (1-u) + F_1 * u \rightarrow$ for $F=0$, $u = F_0 / (F_0 - F_1)$ // F is linear on edges.
- ✓ $x = x_0 + u * (x_1 - x_0) \rightarrow$ use u computed above and locate x 😊.
- ✓ Interpolation schemes discussed in Deformation lecture slides 26-31.

Marching X vs. Cuberille

65 / 98

- ✓ We efficiently (k-d tree) computed signed distance function F .
- ✓ Once we have this function, one can use the Cuberille method (1979) for a cruder approximation than Marching Cubes (1987).

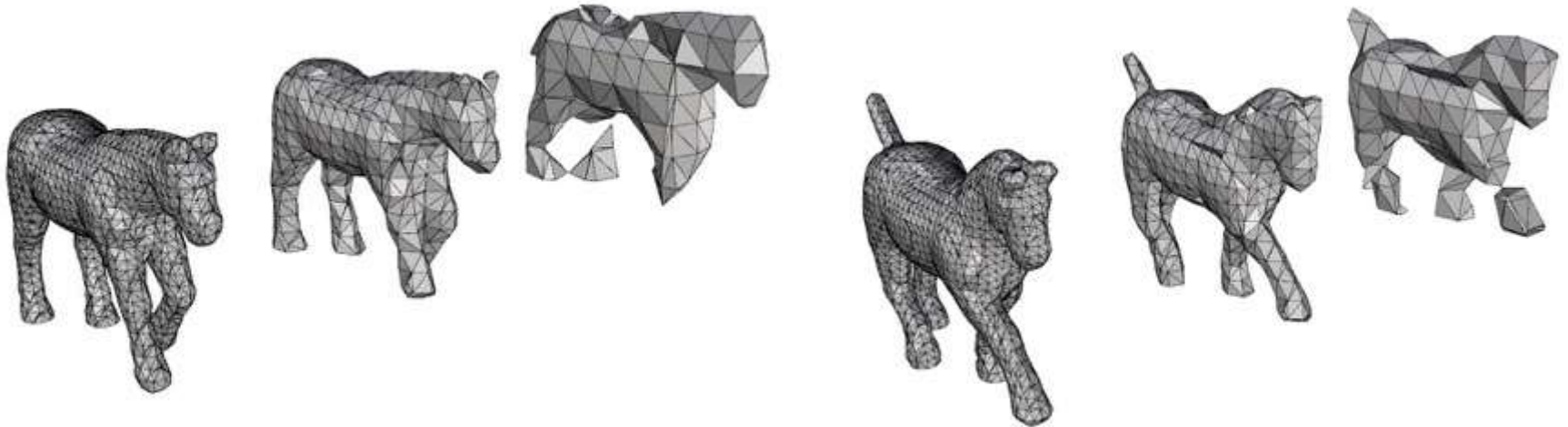


Check function at each cube center.

Marching Cubes Results

66 / 98

- ✓ Size of each voxel in the uniform grid determines level-of-detail.



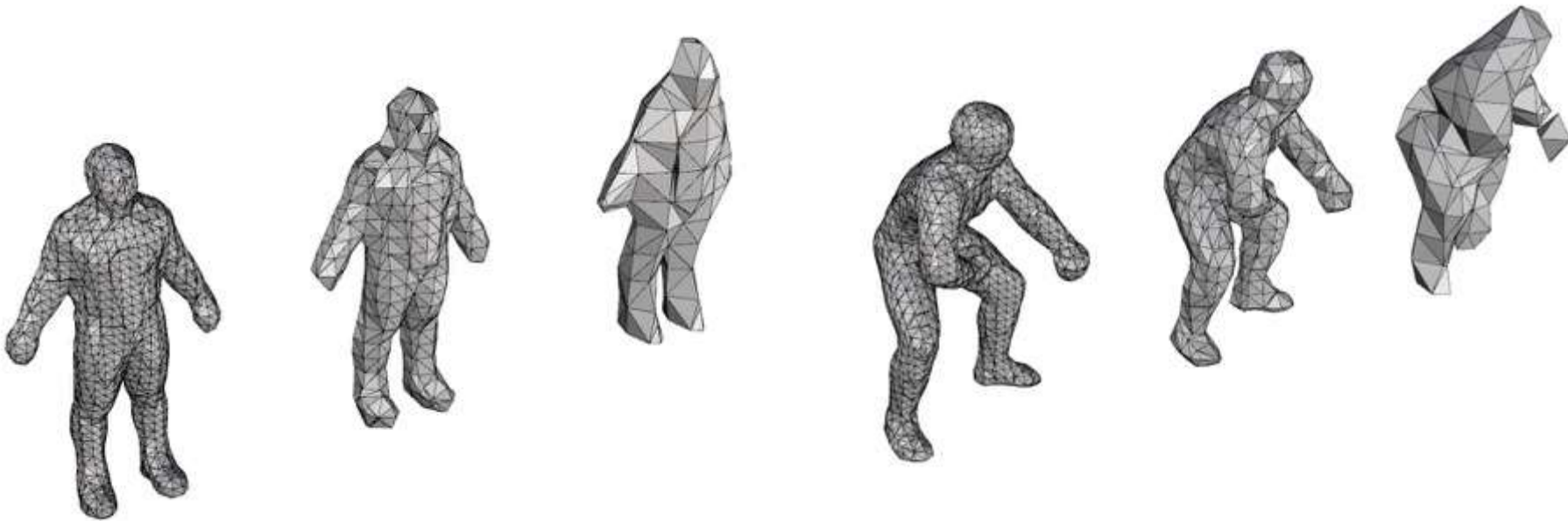
- ✓ Grid not adaptive ☹️.



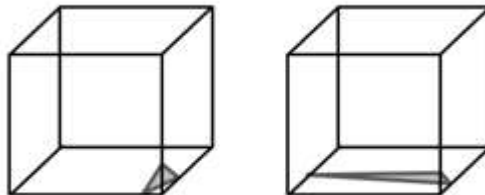
Marching Cubes Results

67 / 98

- ✓ Size of each voxel in the uniform grid determines level-of-detail.



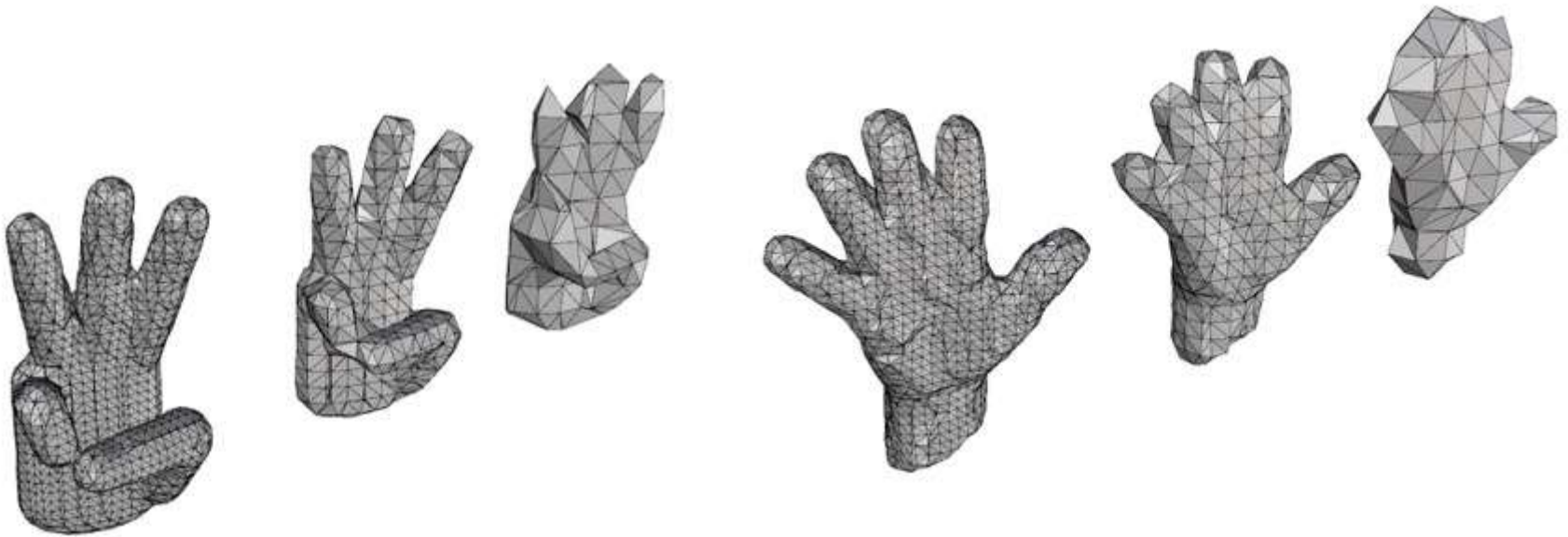
- ✓ Grid sampling can cause short triang edges, rendering tri uninformative.



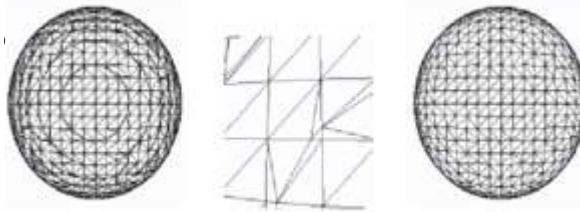
Marching Cubes Results

68 / 98

- ✓ Size of each voxel in the uniform grid determines level-of-detail.



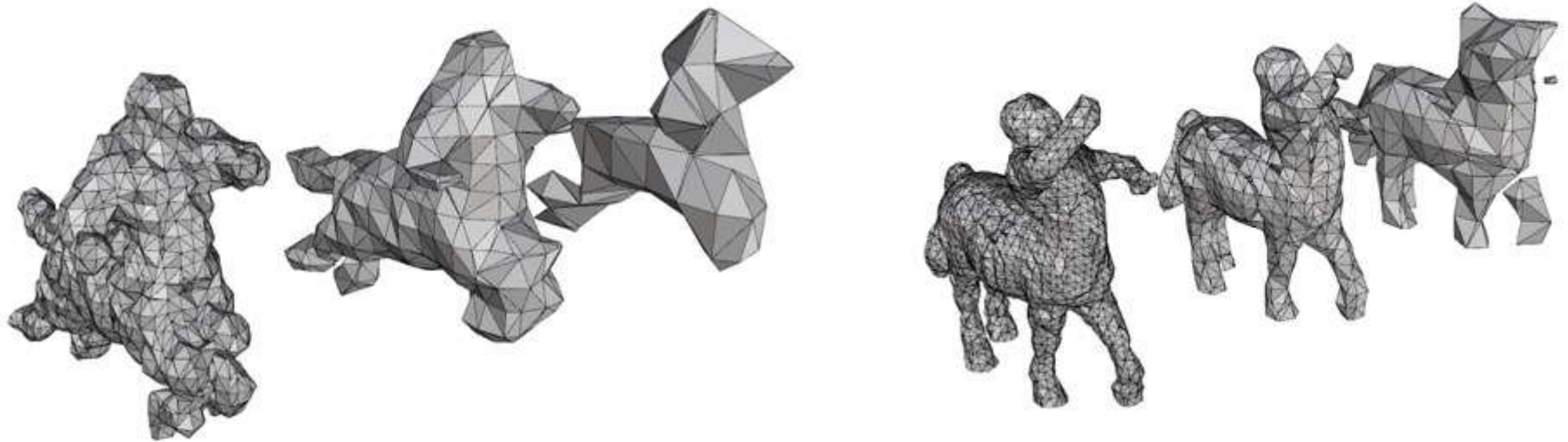
- ✓ Solution: if such an edge arises, snap the edge vertex to cube corner. If more than 1 vertex of



Marching Cubes Results

69 / 98

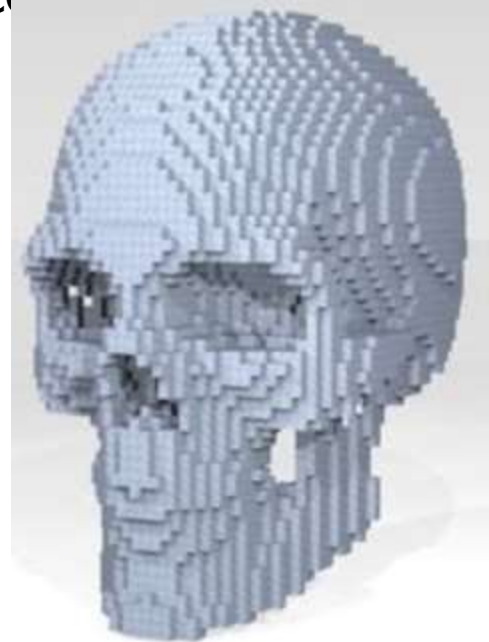
- ✓ Size of each voxel in the uniform grid determines level-of-detail.



Marching Cubes Case Study: CT Scan View

70 / 98

- ✓ In medical imaging, uniform grids are built based on CT/MRI scans.
- ✓ SDF is replaced by intensity values captured by radiology technology.
- ✓ CT scans generally come in DICOM format. Here is one way to convert them to NIfTI (nii) format, which can be parsed easily.
 - ✓ Use dcm2nii.exe software: <https://people.cas.sc.edu/rorden/mricron/dcm2nii.html>
 - ✓ E:\Postdoc\skull\mricron>dcm2nii.exe ..\CTsample\DICOM
- ✓ Unzip output above somescan.nii.gz
- ✓ Learn target intensity using a software like ITK-SNAP, say it is e.
 - ✓ E.g., move cursor around skull and read intensities to decide on e.
- ✓ Marching cubes will extract the level-set corresponding to value e.



Marching Cubes Case Study: CT Scan View

71 / 98

- ✓ Read header and data.
 - ✓ #include "nifti1.h"

```
nifti1.h      x
/** \file nifti1.h
    \brief Official definition of the nifti1 header.  Written by Bob Cox, SSCC, NIMH.

    HISTORY:

        29 Nov 2007 [rickr]
            - added DT_RGBA32 and NIFTI_TYPE_RGBA32
            - added NIFTI_INTENT codes:
              TIME_SERIES, NODE_INDEX, RGB_VECTOR, RGBA_VECTOR, SHAPE
*/

#ifndef _NIFTI_HEADER_
#define _NIFTI_HEADER_

/*****
** This file defines the "NIFTI-1" header format.
** It is derived from 2 meetings at the NIH (31 Mar 2003 and
** 02 Sep 2003) of the Data Format Working Group (DFWG),
** chartered by the NIFTI (Neuroimaging Informatics Technology
** Initiative) at the National Institutes of Health (NIH).
**
** Neither the National Institutes of Health (NIH), the DFWG,
** nor any of the members or employees of these institutions
** imply any warranty of usefulness of this material for any
** purpose, and do not assume any liability for damages,
** incidental or otherwise, caused by any use of this document.
** If these conditions are not acceptable, do not use this!
**
** Author:  Robert W Cox (NIMH, Bethesda)
** Advisors: John Ashburner (FIL, London),
**           Stephen Smith (FMRIB, Oxford),
**           Mark Jenkinson (FMRIB, Oxford)
*****/
```

Marching Cubes

72 / 98

- ✓ Read header and data.
 - ✓ #include "nifti1.h"

```
void VoxelSet::loadNii(char* niiFile)
{
    //binary nii format reader adapted from http://sourceforge.net/projects/niftilib/

    cout << "Voxel grid initializing (to " << niiFile << ")..\n";

    nifti_1_header hdr;
    FILE *fp;
    int ret,i;
    double totalIntensity;
    MY_DATATYPE *data = NULL;

    /***** open and read header */
    fp = fopen(niiFile,"rb"); //seeing this took my 2 hours :(
    if (fp == NULL) {
        fprintf(stderr, "\nError opening header file %s\n",niiFile);
        exit(1);
    }
    ret = fread(&hdr, MIN_HEADER_SIZE, 1, fp);
    if (ret != 1) {
        fprintf(stderr, "\nError reading header file %s\n",niiFile);
        exit(1);
    }

    fclose(fp); //close here 'cos i'll reopen it below with a jump to the start of the image data

    /***** print a little header information */
    fprintf(stderr, "\nnii file header information:");
    fprintf(stderr, "\nXYZ dimensions: %d %d %d %d",hdr.dim[1],hdr.dim[2],hdr.dim[3],hdr.dim[4]);
    fprintf(stderr, "\nDatatype code and bits/pixel: %d %d",hdr.datatype,hdr.bitpix);
    fprintf(stderr, "\nScaling slope and intercept: %.6f %.6f",hdr.scl_slope,hdr.scl_inter);
    fprintf(stderr, "\nByte offset to data in datafile: %ld", (long)(hdr.vox_offset));
    fprintf(stderr, "\nVoxel spacing: %f, %f, %f\n", hdr.pixdim[1], hdr.pixdim[2], hdr.pixdim[3]);

    /***** open the datafile, jump to data offset */
    fp = fopen(niiFile,"rb");
    if (fp == NULL) {
        fprintf(stderr, "\nError opening data file %s\n",niiFile);
        exit(1);
    }

    ret = fseek(fp, (long)(hdr.vox_offset), SEEK_SET); //jump
    if (ret != 0) {
        fprintf(stderr, "\nError doing fseek() to %ld in data file %s\n",
            (long)(hdr.vox_offset), niiFile);
        exit(1);
    }

    /***** allocate buffer and read first 3D volume from data file */
    data = (MY_DATATYPE *) malloc(sizeof(MY_DATATYPE) * hdr.dim[1]*hdr.dim[2]*hdr.dim[3]);
    if (data == NULL) {
        fprintf(stderr, "\nError allocating data buffer for %s\n",niiFile);
        exit(1);
    }
    fread(data, sizeof(MY_DATATYPE), hdr.dim[1]*hdr.dim[2]*hdr.dim[3], fp);
    fclose(fp);
}
```


Marching Cubes Case Study: CT Scan View

73 / 98

✓ Create voxels useful to Marching Cubes,
e.g., the ones that contain skull intensities.

```
//learn width, height, depth of each voxel and then initialize each one (new Voxel)
width = hdr.pixdim[1];
height = hdr.pixdim[2];
depth = hdr.pixdim[3]; //of each cell to be added to voxels[] below
int toRight = 0, toUp = 0, toFront = 0;
float x, y, z;
//all voxels that includes bones, brains, eyes, .. everything; just need skull bones here
for (i = 0; i < hdr.dim[1]*hdr.dim[2]*hdr.dim[3]; i++)
{
    //id, intensity, and coord of this current voxel
    x = toRight * hdr.pixdim[1]; //voxel spacing, space b/w x coords, i.e. pixdim[1]
    toRight++;
    y = toUp * hdr.pixdim[2]; //voxel spacing, space b/w y coords, i.e. pixdim[2]
    z = toFront * hdr.pixdim[3]; //voxel spacing, space b/w z coords, i.e. pixdim[3]
    if ( isSkullVoxel(data[i], x, y, z) ) //returns true if (intensity > 200 && intensity < 1000) where intensity is data[i]
    {
        if (toRight % 5 == 0 && toUp % 5 == 0 && toFront % 5 == 0) //extra thresholding to downsample the bone voxels by picking
                                                                    //every fifth in x-direction and in y-direction and in z-direction
        {
            float* coords = new float[3];
            coords[0] = x;
            coords[1] = y;
            coords[2] = z;
            voxels.push_back( new Voxel(i, data[i], coords) ); //coords is the center of this voxel
        }
    }

    //direction/pointer adjustments
    if (toRight >= hdr.dim[1])
    {
        toRight = 0; //reset to the left end
        toUp++; //1 level up
        if (toUp >= hdr.dim[2])
        {
            toFront++; //1 slice towards me, that is towards front
            toUp = 0; //start from the bottom level
        }
    }
} //end of i

//you may then create coarse voxels by merging every non-overlapping 3x3x3 box (of 27 voxels)
//in data[] to 1 big coarseVoxel if 1+ constituent element is a segmented skull/bone voxel
//..
```

Scientific Visualization

74 / 98

- ✓ Scalar field: a scalar value is associated with each point in n -d space.
 - ✓ Can be obtained by associating SDF or intensity values to grid points.
- ✓ Vector field: an n -d vector is associated with each point in n -d space.
 - ✓ Can be obtained by simulations.
- ✓ Scalar fields can be visualized via
 - ✓ indirect (volume) rendering: isosurface/mesh extraction and rendering (corresponding to a certain scalar value – Marching Cubes in 3D).
 - ✓ *We handled this already.*
 - ✓ direct (volume) rendering – ray casting.
 - ✓ *Upcoming.*
- ✓ Vector fields can be visualized via
 - ✓ vector arrows located at each point.
 - ✓ *Upcoming.*

Direct Scalar Field Visualization

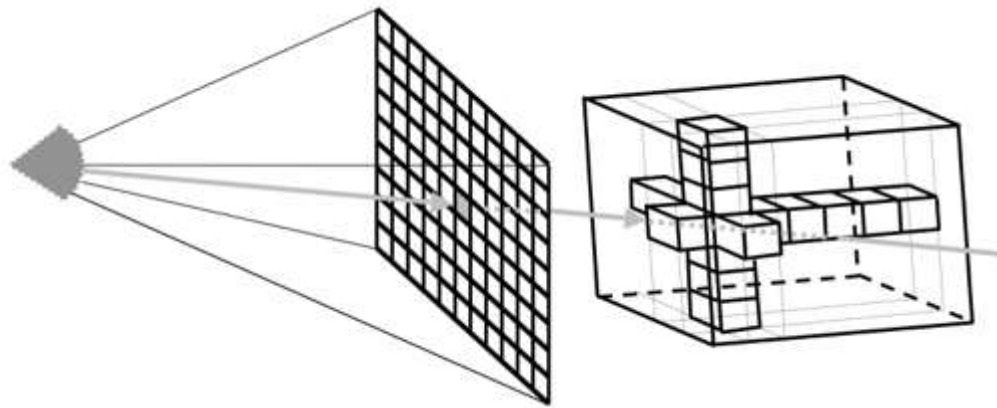
75 / 98

- ✓ So far we've dealt with visualizing a scalar field indirectly, via isosurface extraction and rendering.
- ✓ Let's now do it directly via ray casting.
- ✓ Aka direct volume rendering (DVR) if the field is defined in 3D.

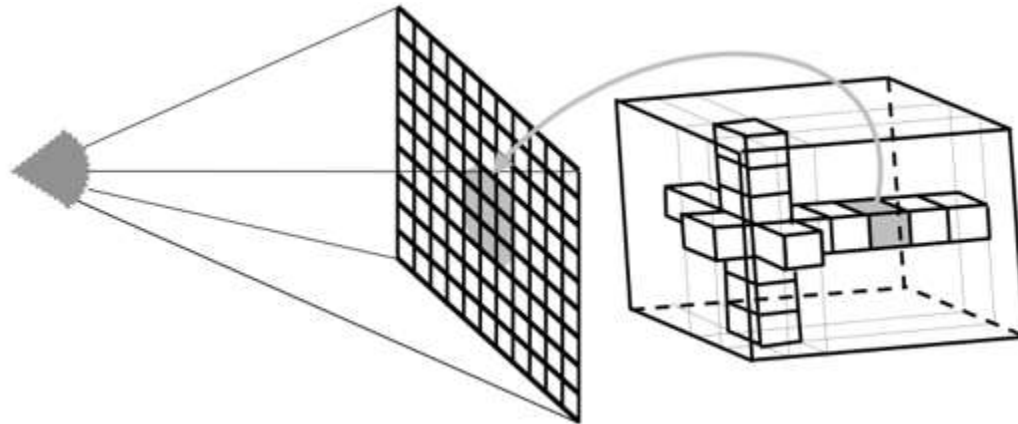
Direct Scalar Field Visualization

76 / 98

- ✓ Ray casting is a backward method where we send rays from the eye.



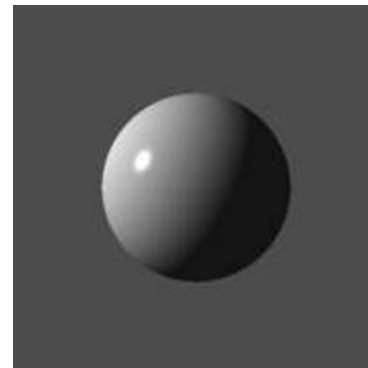
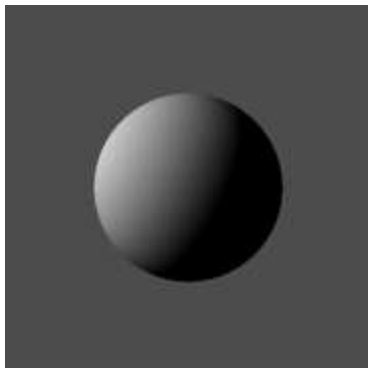
- ✓ Opposite of the forward method where we use projections to the eye.



Direct Scalar Field Visualization

77 / 98

- ✓ Ray casting is similar to ray tracing in surface-based computer graphics.
 - ✓ Ray tracing deals with primary rays (first rays cast into the scene from the camera/eye) and secondary rays (recursive rays to generate shadows, reflections, refractions, etc.)
 - ✓ 1979, Turner Whitted.
 - ✓ Ray casting deals only with primary rays.
 - ✓ 1968, Arthur Appel.
- ✓ Check out CENG477 Computer Graphics course for details.
 - ✓ <http://user.ceng.metu.edu.tr/~ys/ceng477-gfx/>
 - ✓ Look at Ray Tracing parts 1 and 2, up to the Recursive Ray Tracing section.

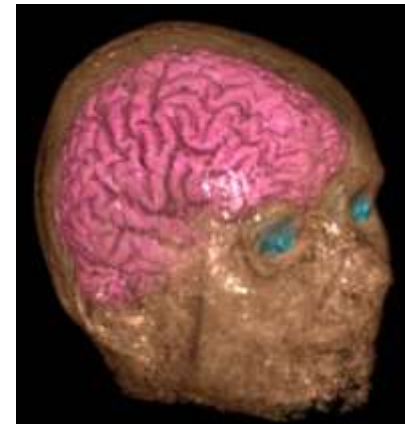
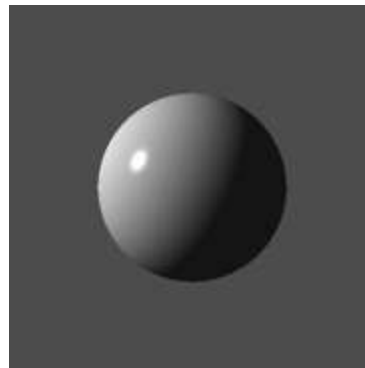
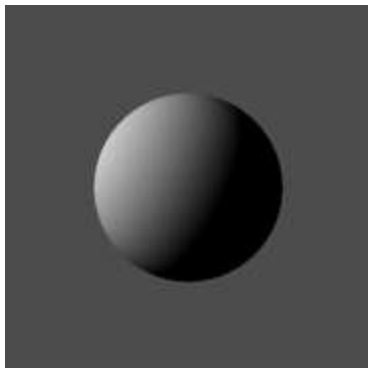


Additional color
due to lighting.

Direct Scalar Field Visualization

78 / 98

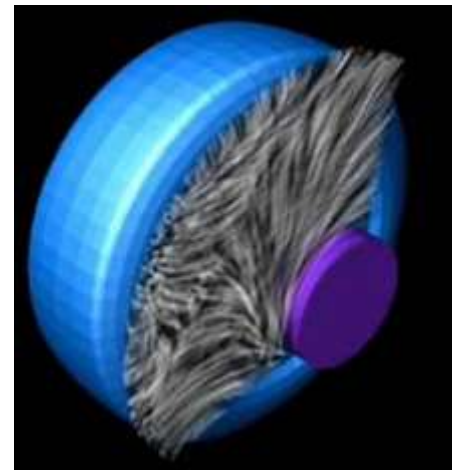
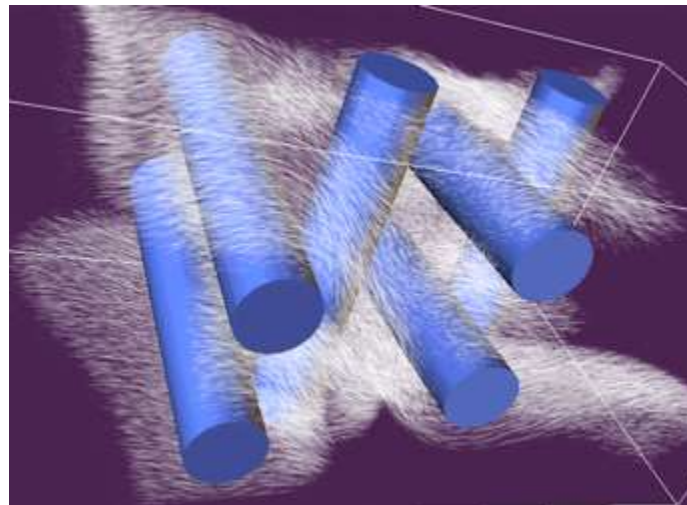
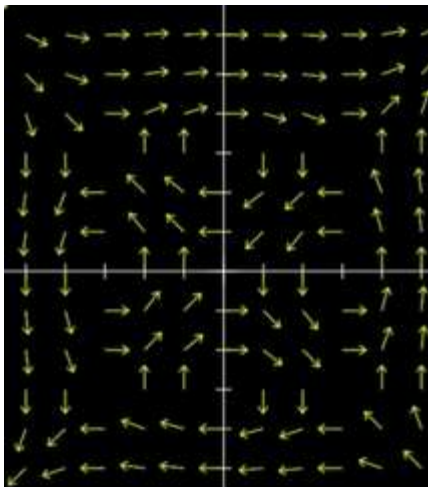
- ✓ How to calculate ray-surface intersection since we've no surfaces here.
- ✓ Step through the volume.
- ✓ A ray is cast into the volume, sampling the volume at certain intervals.
- ✓ Sampling intervals are usually equidistant, but don't have to be (e.g. importance sampling).
- ✓ At each sampling location, a sample is interpolated / reconstructed from the voxel grid.
- ✓ Opacity and color in each cell is predefined according to classification.



Vector Field Visualization

79 / 98

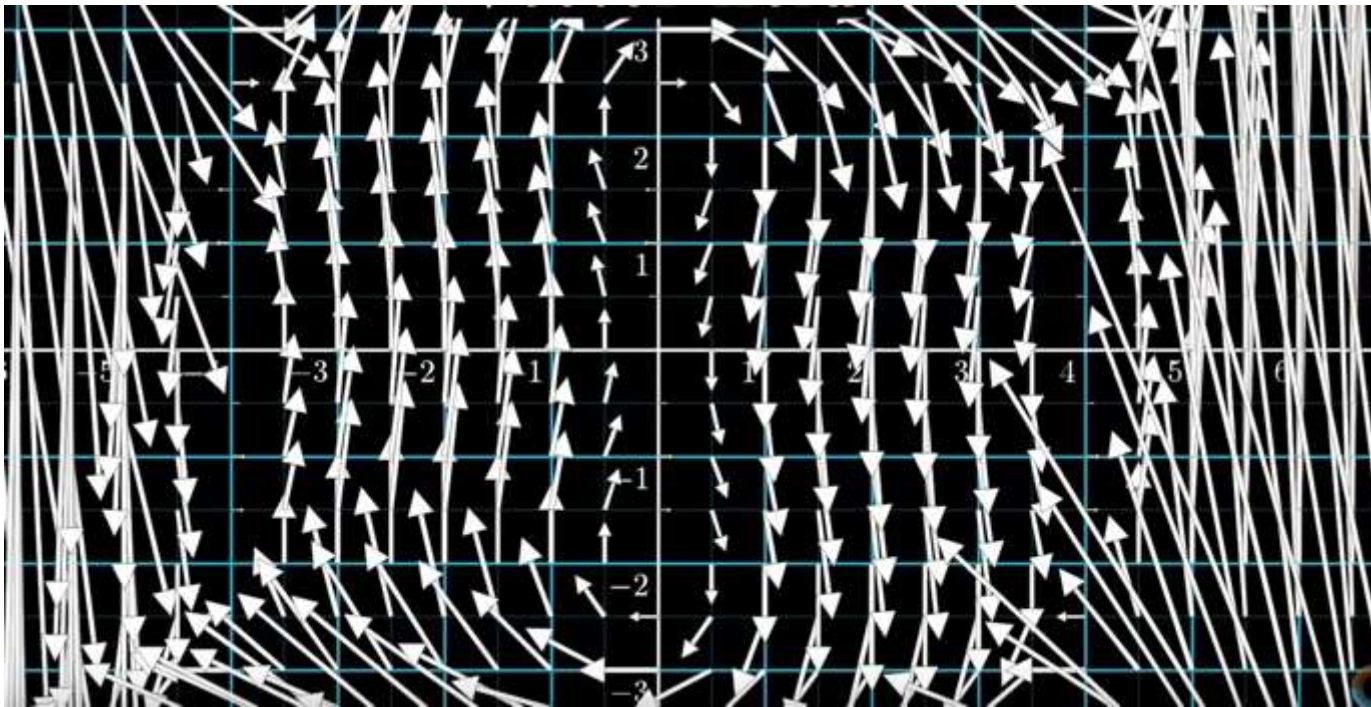
- ✓ So far we've dealt with visualizing a scalar field, i.e., a grid with function values at each grid point, for a given scalar value.
 - ✓ Arises frequently in medical imaging, 3D point cloud to surface conversions.
- ✓ Let's now extend this to vector field visualization, where we have an n-dimensional space with n-dimensional vectors attached at each point.
 - ✓ Arises frequently in fluid simulation, air flow simulation, or in general visualizing functions that have the same # of dimensions in their input as in their output.
 - ✓ $f(x, y) = [x^2 + 5y \quad xy]^T$, $f(x, y, z) = [4x \quad x^3(y^2 - 5) \quad 33]^T$, and so on.



Vector Field Visualization

80 / 98

- ✓ If all vectors are drawn proportional to their magnitudes, the longer ones mess up the screen.

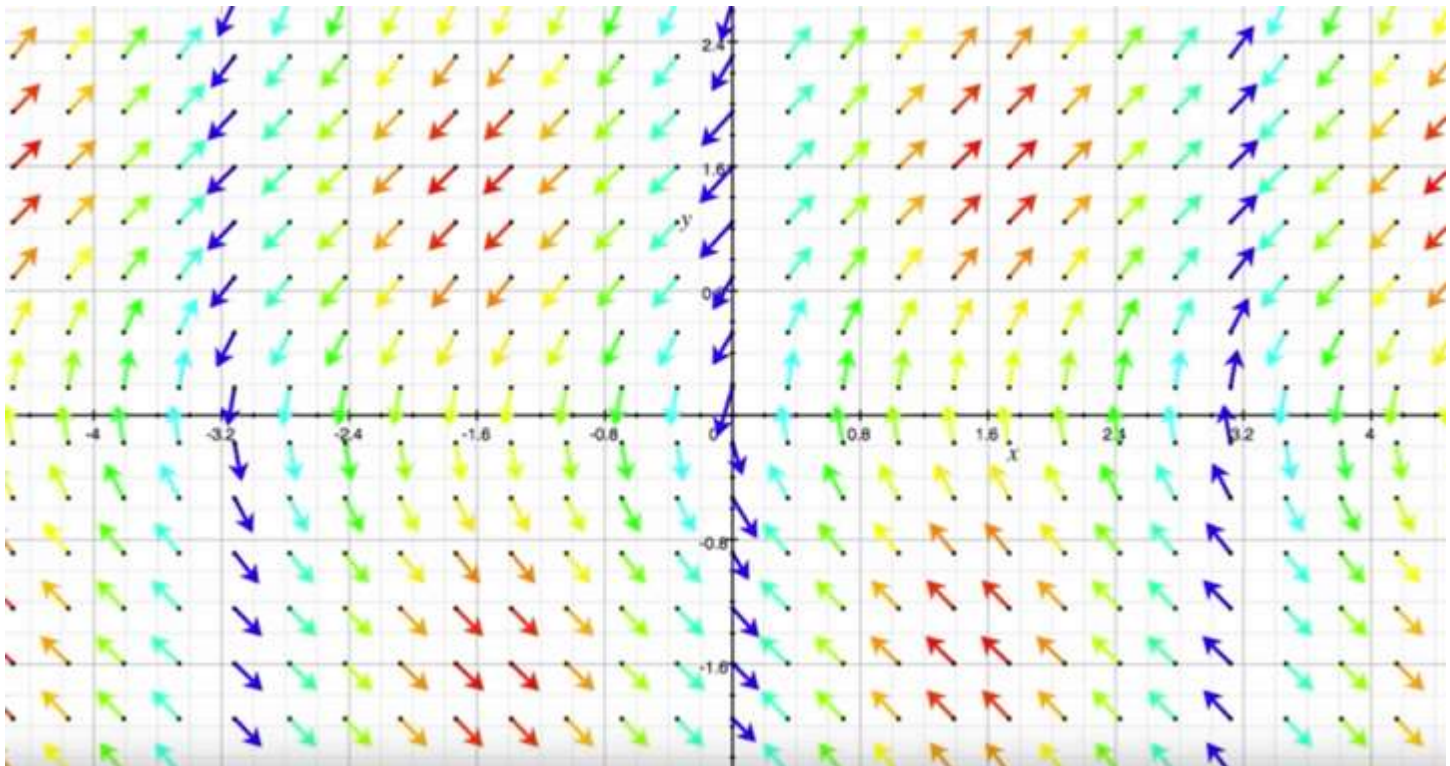


- ✓ This one and the subsequent visuals're collected from the fluid flow youtube series of Khan Academy and 3Blue1Brown.

Vector Field Visualization

81 / 98

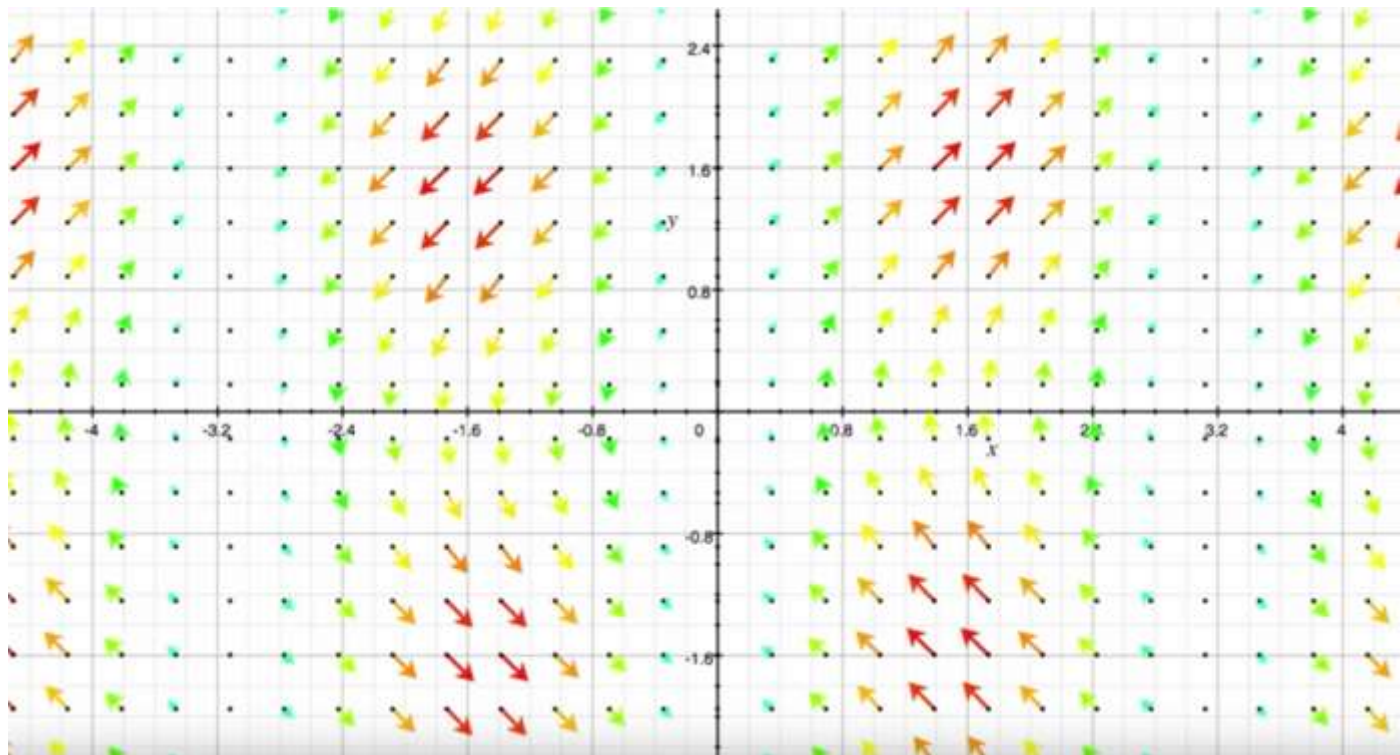
- ✓ All vectors are made unit (length 1) for visual clarity.
- ✓ We can still infer their magnitudes via color coding: red long blue short.



Vector Field Visualization

82 / 98

- ✓ All vectors are made unit (length 1) for visual clarity.
- ✓ Combined w/ proper scaling where max length is 1 (prevent mess).

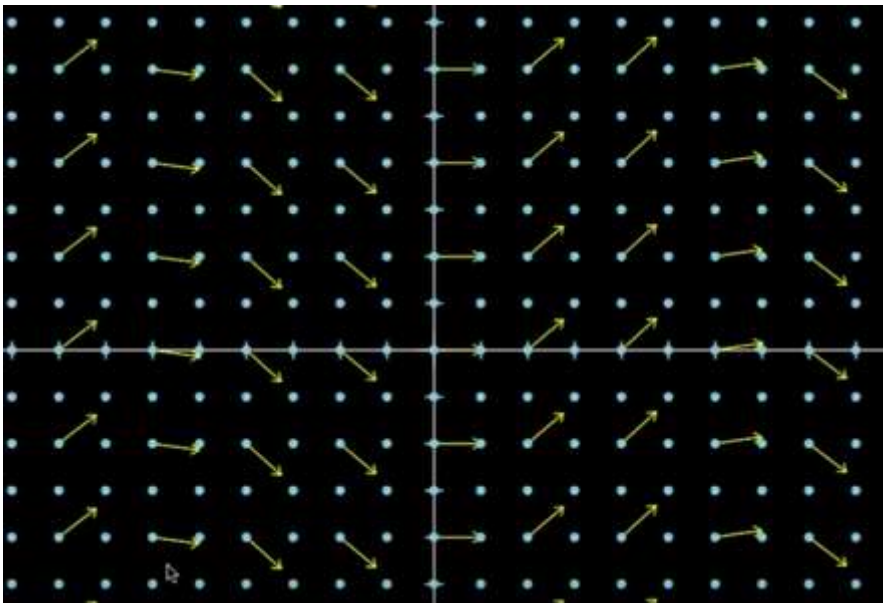


Vector Field Visualization

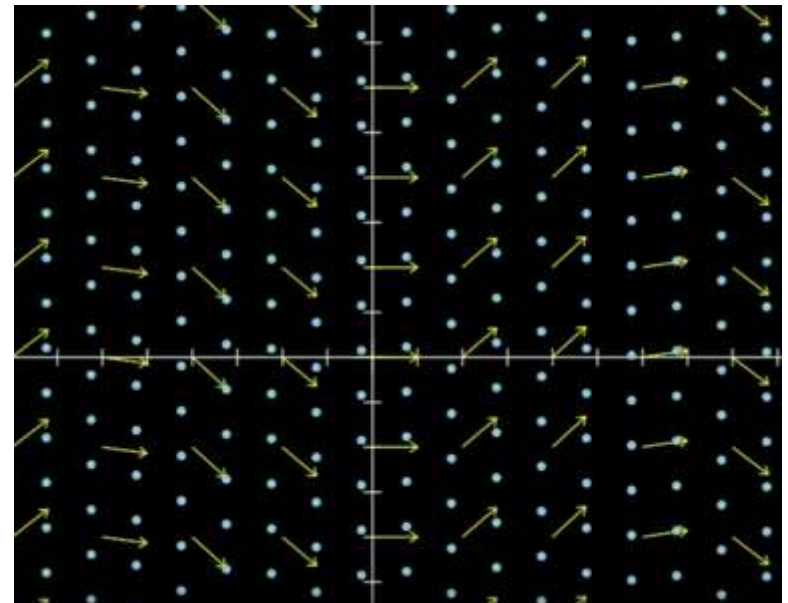
83 / 98

- ✓ Case study: fluid flow.
- ✓ Droplets of water are moving according to the vectors visualized.

initial state



$t + 10$

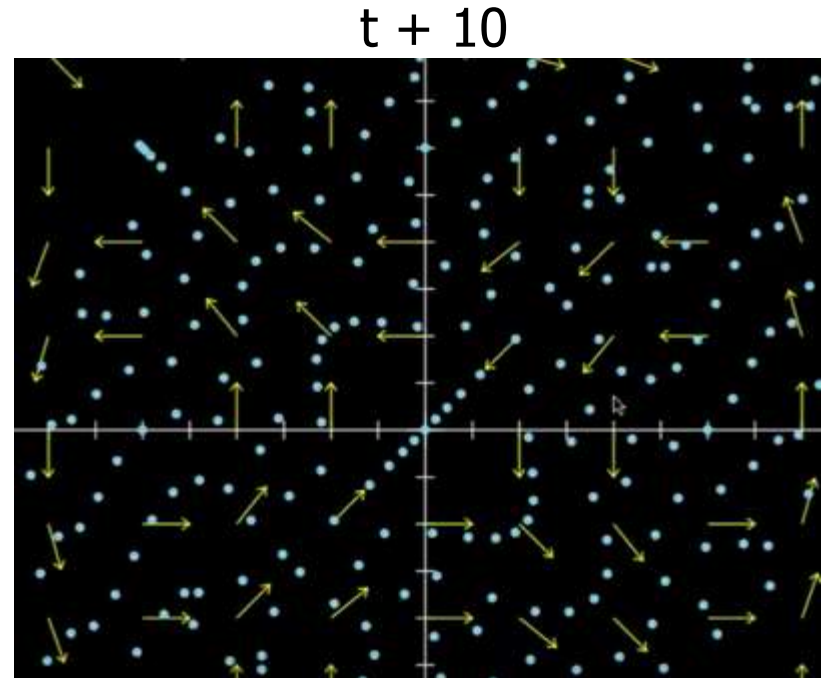
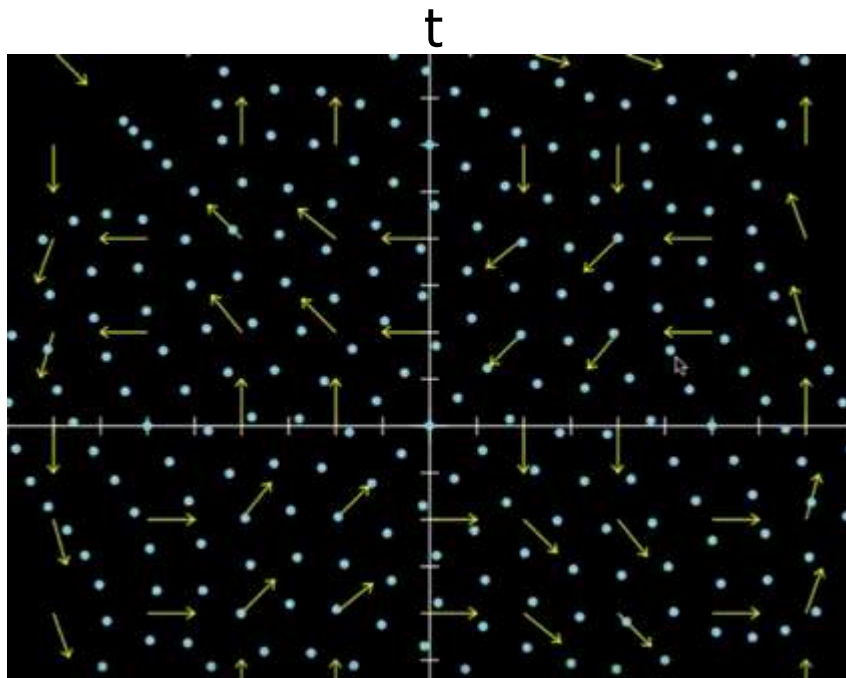


- ✓ Vectors are representing the velocities of particles of fluids here.
 - ✓ Alternatives: force of gravity, magnetic field, etc.

Vector Field Visualization

84 / 98

- ✓ Case study: fluid flow.
- ✓ Droplets of water are moving according to the vectors visualized.



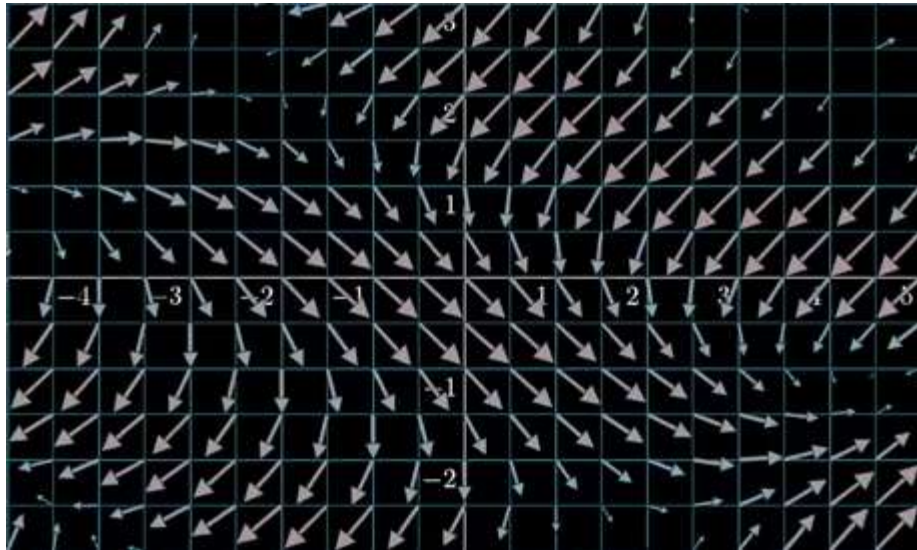
- ✓ This is a static vector field depicting a steady state system.

Vector Field Visualization

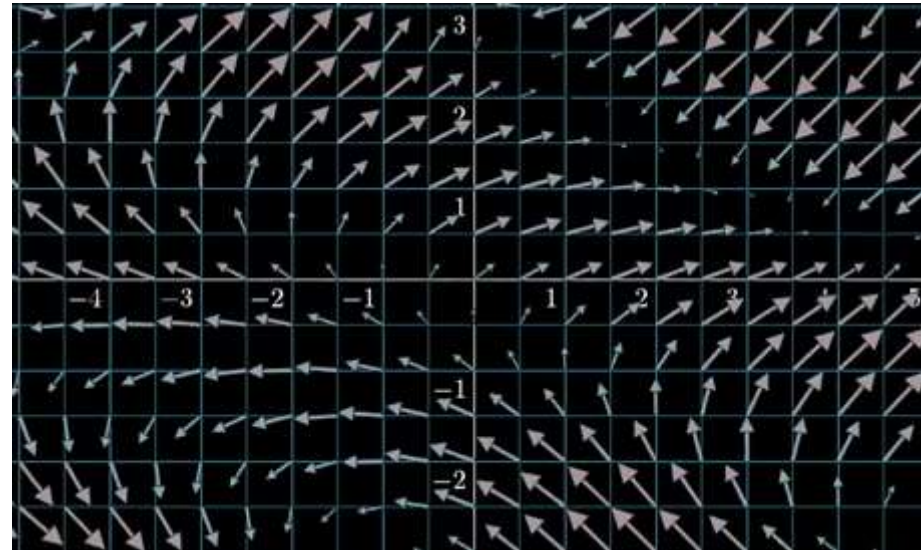
85 / 98

- ✓ Case study: fluid flow.
- ✓ Vector fields may change over time, e.g., in real-world fluid flow, the velocities of particles change in response to the surrounding context.

t



$t + 10$

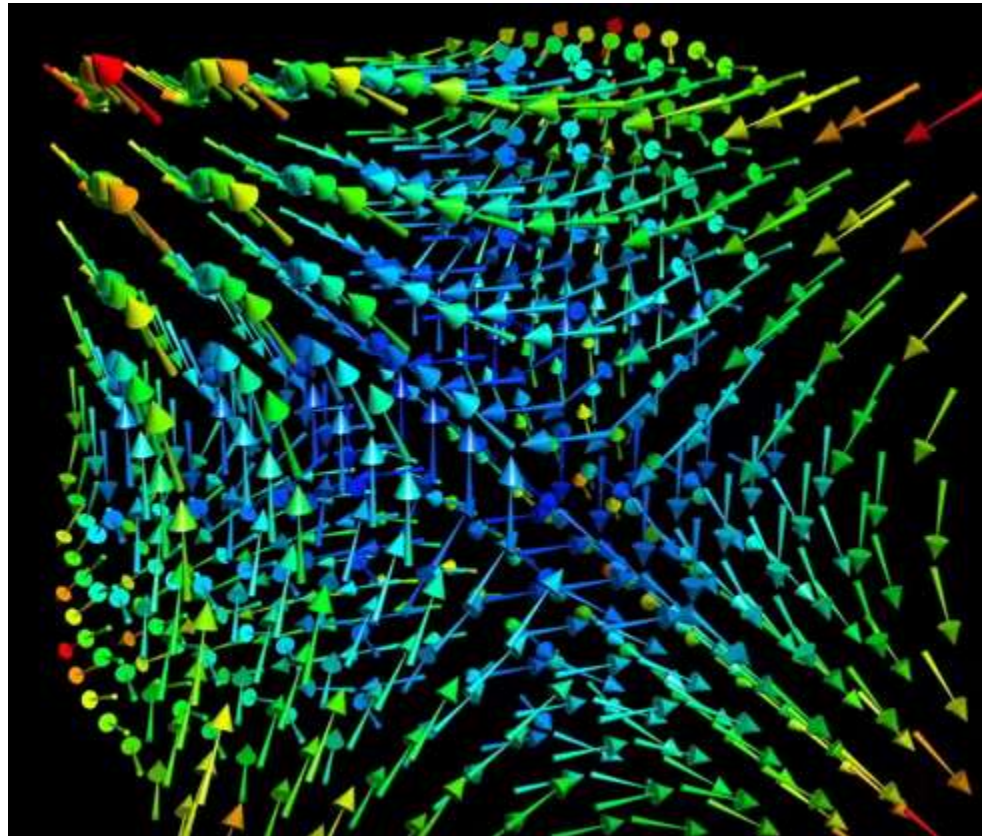


- ✓ This is a dynamic vector field depicting a more realistic fluid flow.
 - ✓ Wind is not constant; electric field not constant (charged particles move).

Vector Field Visualization

86 / 98

- ✓ n-d spaces are equipped with n-d vectors.
- ✓ So far we studied n=2. Here n=3:

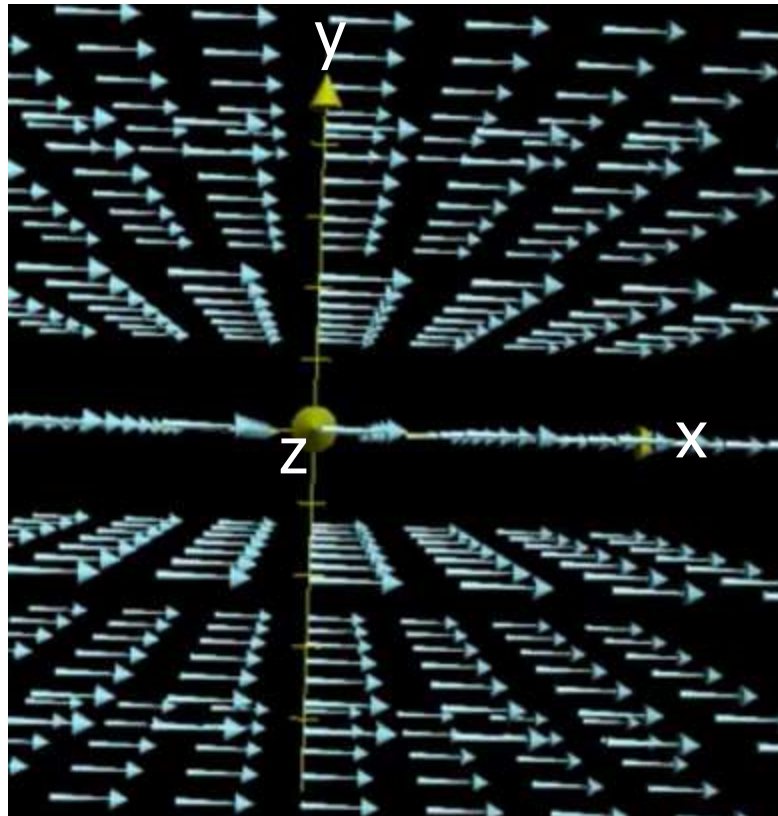


- ✓ Vectors at each point are defined as $f(x, y, z) = [yz \quad xz \quad xy]^T$.

Vector Field Visualization

87 / 98

- ✓ n-d spaces are equipped with n-d vectors.
- ✓ So far we studied n=2. Here n=3:

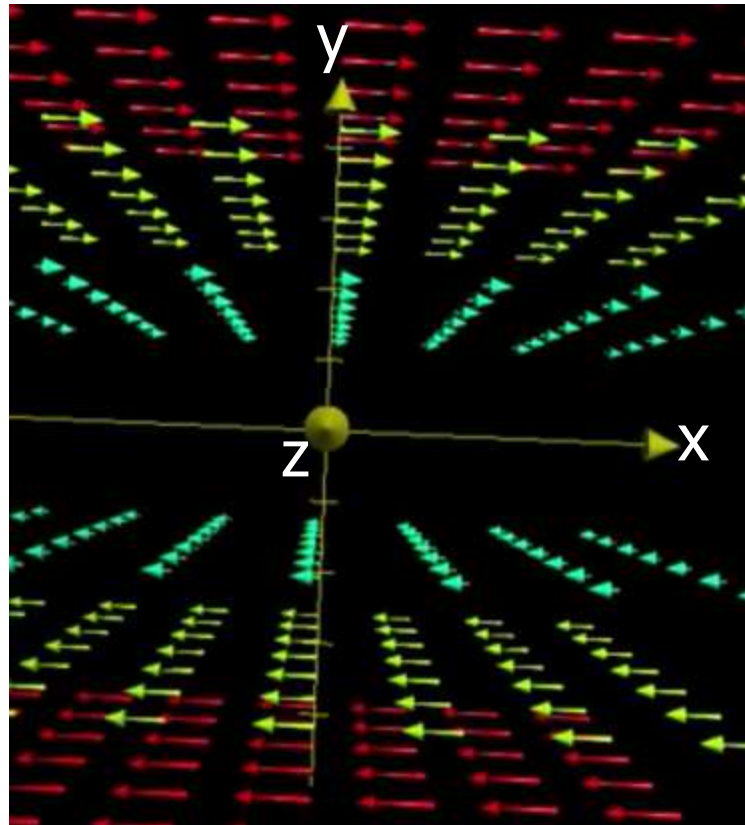


- ✓ Vectors at each point are defined as $f(x, y, z) = [1 \ 0 \ 0]^T$.

Vector Field Visualization

88 / 98

- ✓ n-d spaces are equipped with n-d vectors.
- ✓ So far we studied $n=2$. Here $n=3$:

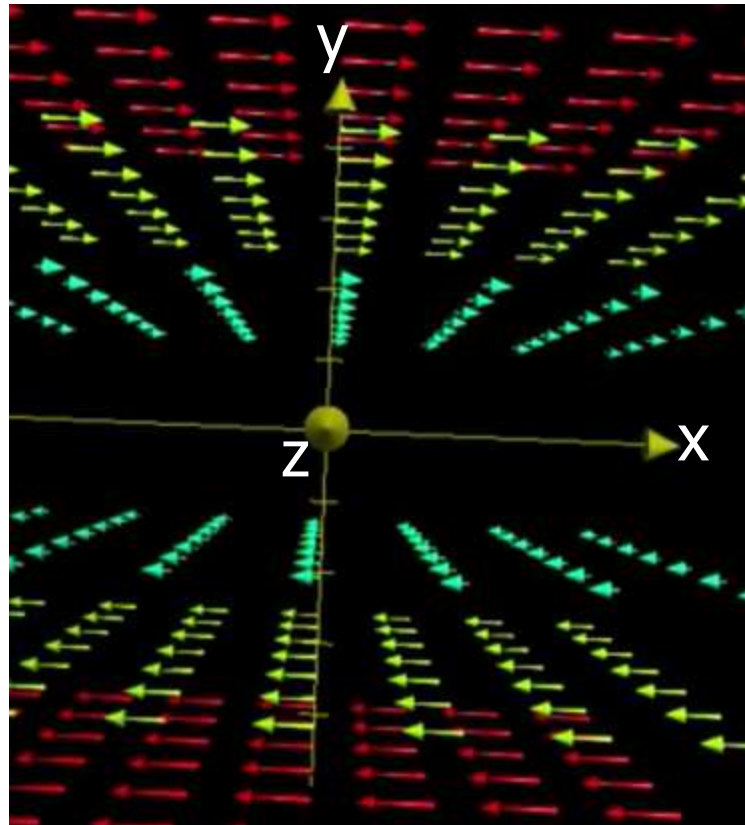


- ✓ Vectors at each point are defined as $f(x, y, z) = ?$. Color \sim magnitude.

Vector Field Visualization

89 / 98

- ✓ n-d spaces are equipped with n-d vectors.
- ✓ So far we studied n=2. Here n=3:

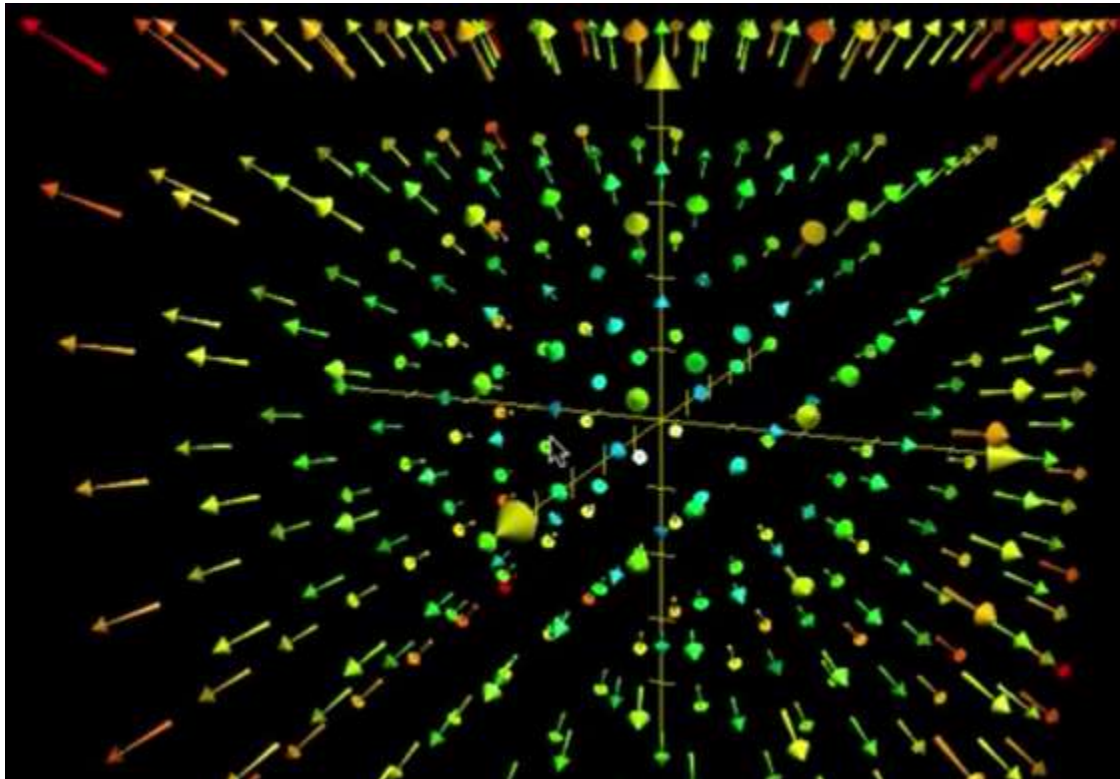


- ✓ Vectors at each point are defined as $f(x, y, z) = [y \ 0 \ 0]^T$.

Vector Field Visualization

90 / 98

- ✓ n-d spaces are equipped with n-d vectors.
- ✓ So far we studied n=2. Here n=3:

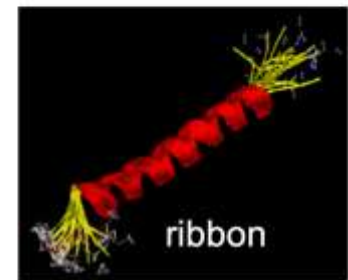
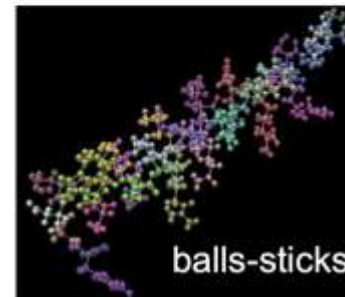
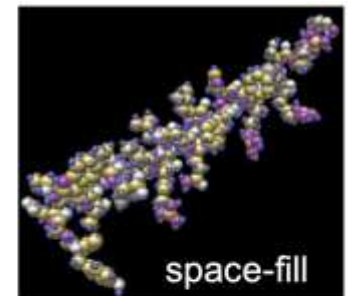
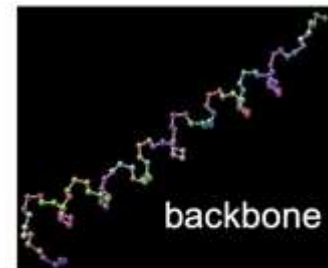
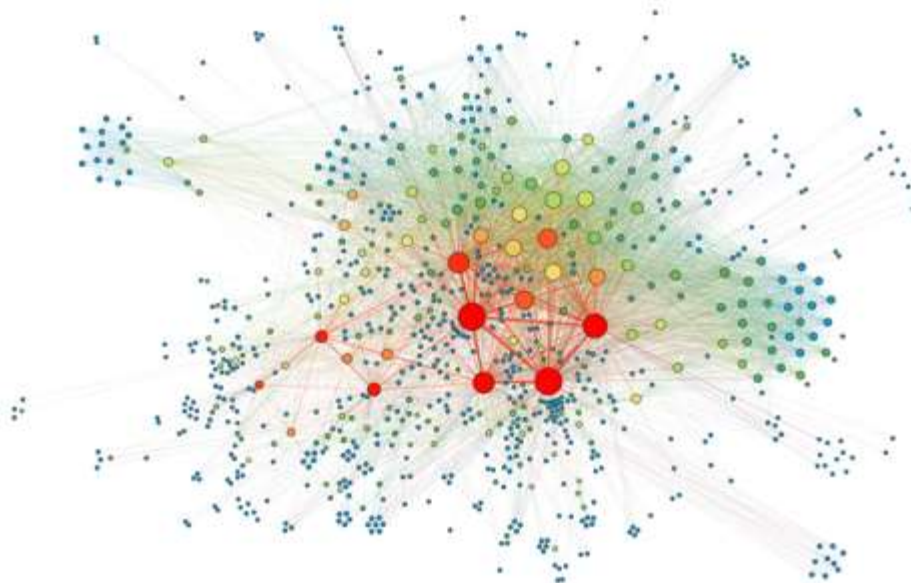


- ✓ Vectors at each point are defined as $f(x, y, z) = [x \ y \ z]^T$.

Information Visualization

91 / 98

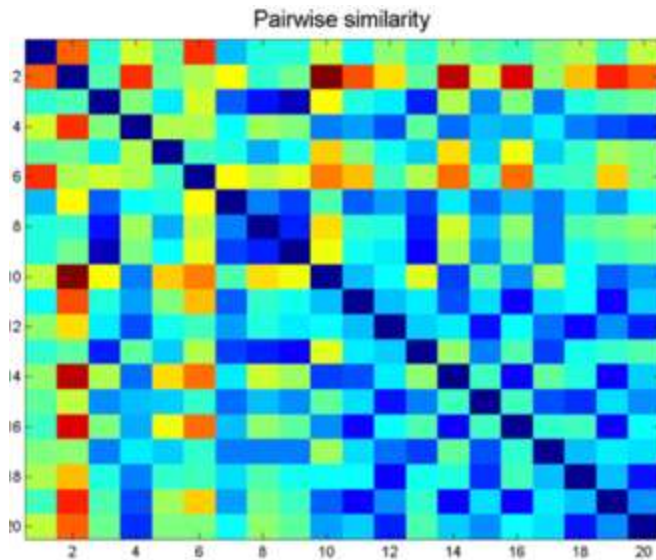
- ✓ Study of visual representations of abstract data to reinforce human cognition.
- ✓ Differs from Scientific Visualization: it is infovis (information visualization) when the spatial representation is chosen, and it is scivis (scientific visualization) when the spatial representation is given.
- ✓ Some examples include social network analysis, protein visualization.



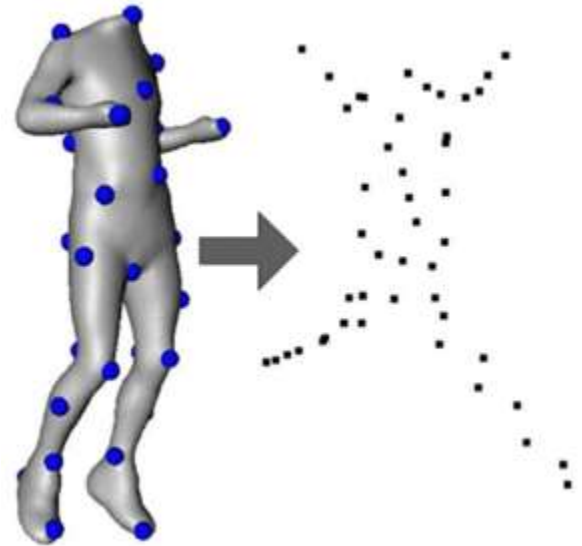
Information Visualization - MDS

92 / 98

- ✓ Multidimension Scaling (MDS) is a powerful tool to visualize pairwise dissimilarity data.
- ✓ Translates “info about the pairwise 'distances' among a set of n objects” into a configuration of n points mapped into the Euclidean space.
 - ✓ A form of non-linear dimensionality reduction: see more in Shape Gen. slides.



Geodesic distances b/w sample pairs.



MDS mapping of samples.

Information Visualization - MDS

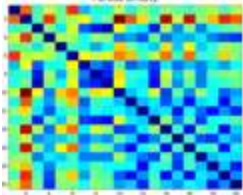
93 / 98

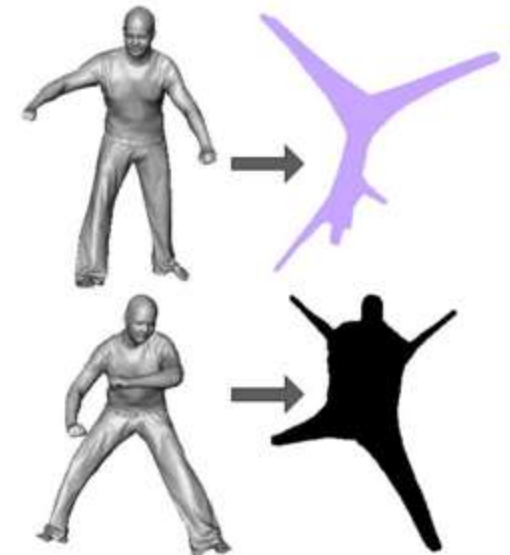
- ✓ Given a distance matrix with the distances $\delta_{i,j}$ between each pair of objects, and a chosen number of dimensions k , an MDS algorithm places each object into k -dimensional Euclidean space such that the between-object distances $\|x_i - x_j\|$ are preserved as well as possible. For $k=1, 2$, and 3 , the resulting points can be visualized on a scatter plot.

$$\min_{x_1, \dots, x_I} \sum_{i < j} (\|x_i - x_j\| - \delta_{i,j})^2$$

Information Visualization - MDS

94 / 98

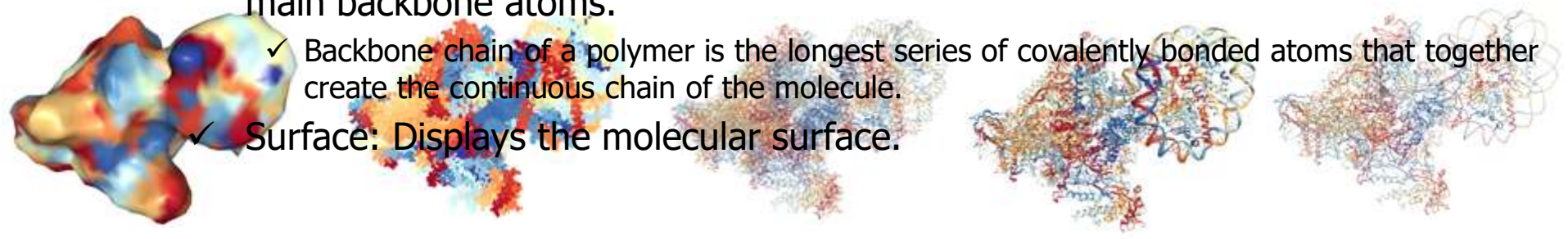
- ✓ Main MDS techniques are as follows.
- ✓ Least-squares MDS: minimizes the energy:
$$\min_{x_1, \dots, x_I} \sum_{i < j} (\|x_i - x_j\| - \delta_{i,j})^2$$
- ✓ Classical MDS: uses k leading eigenvectors of the associated affinity matrix () to obtain the k-d configuration.
- ✓ Landmark MDS: embeds a small number of landmark points via least-squares or classical method, and then computes the embedding coordinates of the remaining data points based on their distances from the landmark points.



Information Visualization – Protein Vis.

95 / 98

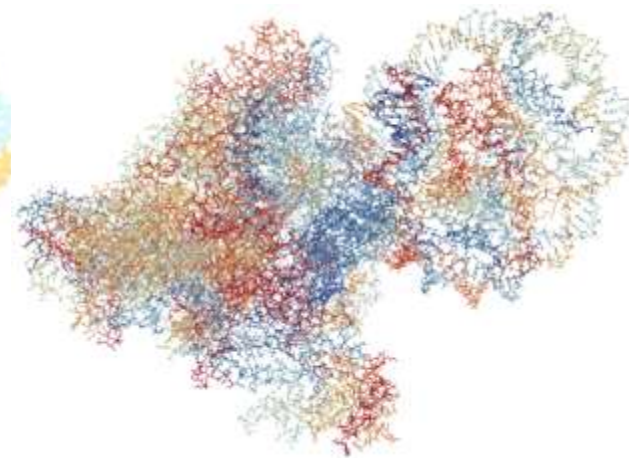
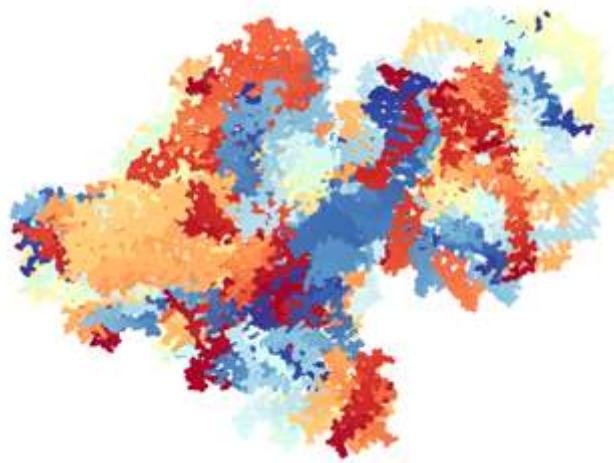
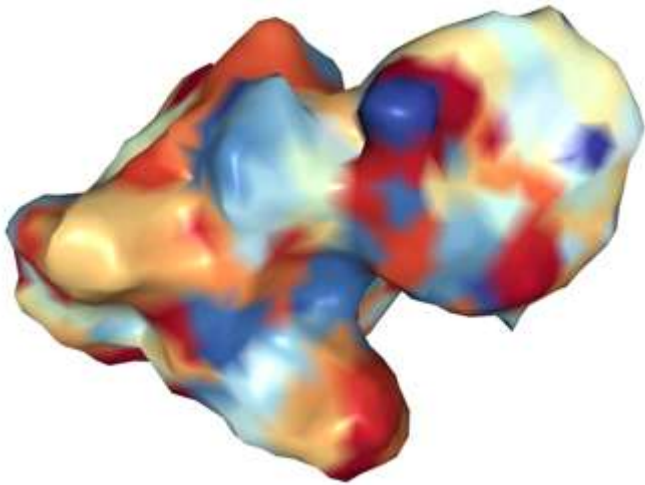
- ✓ Some popular visualization modes:
 - ✓ Ball+stick: Atoms are displayed as spheres (balls) and bonds as cylinders (sticks). Aspect ratio defines how much bigger sph radius compared to cyl radius.
 - ✓ Licorice: Ball+stick variant where balls and sticks have the same radius.
 - ✓ Line: Bonds are displayed by a flat, unshaded line.
 - ✓ Point: Atoms are displayed by textured points.
 - ✓ Spacefill: Atoms are displayed as a set of space-filling spheres.
 - ✓ Trace: A flat, unshaded line is displayed along the main backbone trace.
 - ✓ Ribbon: A thin ribbon is displayed along the main backbone trace.
 - ✓ Cartoon: A smooth trace connects successive residues of unbroken chains by their main backbone atoms.
 - ✓ Backbone: Cylinders connect successive residues of unbroken chains by their main backbone atoms.
 - ✓ Surface: Displays the molecular surface.



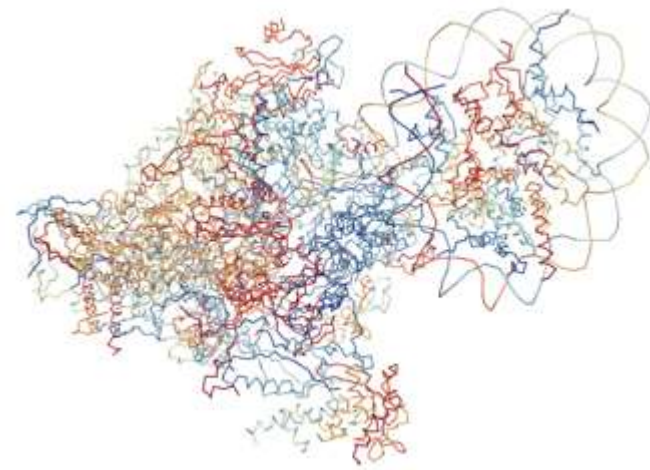
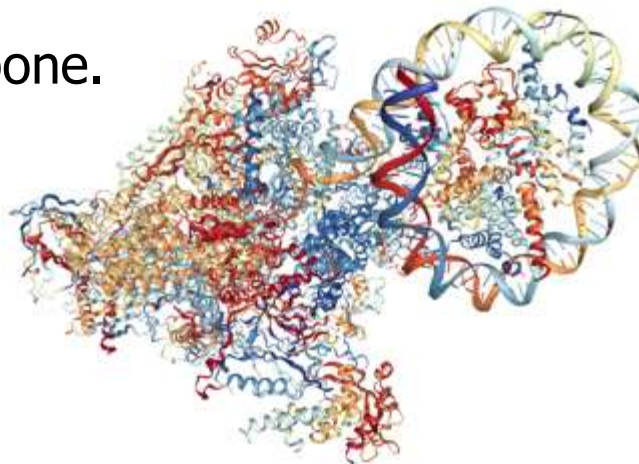
Information Visualization – Protein Vis.

96 / 98

- ✓ Other modes from left to right: surface, line, licorice ..



- ✓ .. cartoon, backbone.



Information Visualization – Protein Vis.

97 / 98

- ✓ <https://www.rcsb.org/> provides state-of-the-art visualizations of protein structures in different intuitive ways, e.g., <http://www.rcsb.org/3d-view/2BHM>

The screenshot shows the RCSB PDB website's 3D View interface for the protein structure 2BHM. The main title is "2BHM" with the subtitle "Crystal structure of VirB8 from Brucella suis". A note instructs users to use their mouse to drag, rotate, and zoom in and out of the structure. The central 3D view displays the protein structure in a spacefill mode, with different subunits colored in blue, red, orange, and yellow. To the right, there is a control panel with tabs for "Structure View", "Electron Density Maps", and "Ligand View". Under "Structure View", there are dropdown menus for "Assembly" (Bioassembly 1), "Model" (Model 1), "Symmetry" (None), "Style" (Spacefill), "Color" (Rainbow), "Ligand" (Ball & Stick), and "Quality" (Automatic). There are also checkboxes for "Water" (unchecked), "Hydrogens" (checked), "Ions" (checked), and "Clashes" (unchecked). At the bottom, there are buttons for "Spin", "Center", "Fullscreen", "Screenshot", "Perspective Camera", "White background", and a "Focus" slider.

Spacefill mode.

Potential Project Topics

98 / 98

- ✓ Implement Marching Cubes algorithm for implicit surface reconstruction.
 - ✓ Address the related research questions posed in Slide 17* and Slide 39**.
 - * Rather than contouring the surface with 0 isovalue, take the isovalue as the median signed distance value over the input points, suggested by the paper *Signing the Unsigned: Robust Surface Reconstruction from Raw Pointsets*.
 - ** Fixing the discontinuity issue in the original SDF.
- ✓ Implement Crust algorithm for explicit surface reconstruction.
- ✓ Implement MDS by casting it as a mass-spring problem.