

Euclidean Distance Maps and Eikonal Equations

Yaakov Tsaig



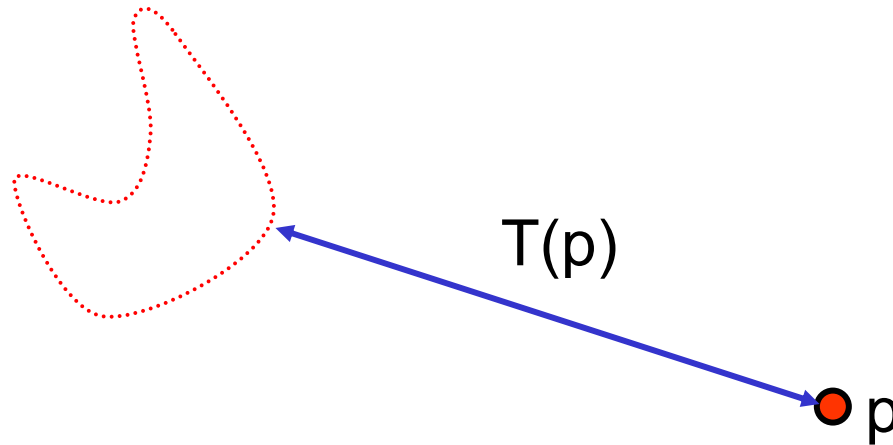
Papers

- H.K. Zhao, “*A fast sweeping method for eikonal equations*”.
- P.E. Danielsson, “*Euclidean distance mapping*”.
- H. Pottmann, S. Leopoldseder, H.K. Zhao, “*The d^2 -tree: A hierarchical representation of the squared distance function*”.
- **Book:** R. Kimmel, “*Numerical Geometry of Images*”, Springer-Verlag.

Distance Map

- Let S be a set of source points (representing a curve, surface, object), and D the domain of interest
- A distance map is a function $T : D \rightarrow R_+$, s.t.

$$T(p) = \inf_{q \in S} \|p - q\|_{L^2}$$





Computing Distance Maps

- **Q1:** So how do we compute distance maps?
- **A1:** For each point of interest in the domain D , scan all source points in S and find the closest one.
- **Drawback:** Will take forever...
- **Q2:** So how do we compute distance maps and get a result **in our lifetime**?
- **A2:** Sweeps with alternating directions.

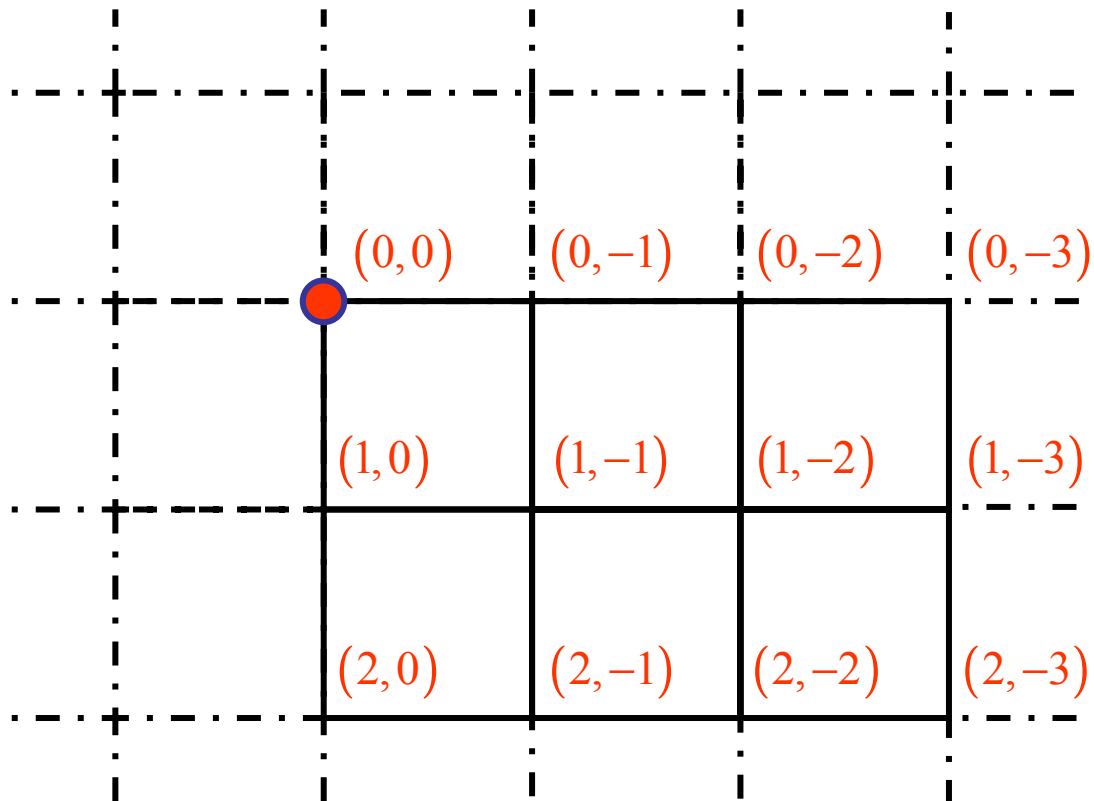


Danielsson's Algorithm

- 2-D case: For each point we store the (x,y) offset to the closest point.
- Initially, all offsets of points in S are $(0,0)$ and offsets of points not in S are (∞, ∞) .
- Scan the image 4 times in alternating directions (up/down, left/right).
- Each point checks the values of its four closest neighbors, and updates its own value accordingly.

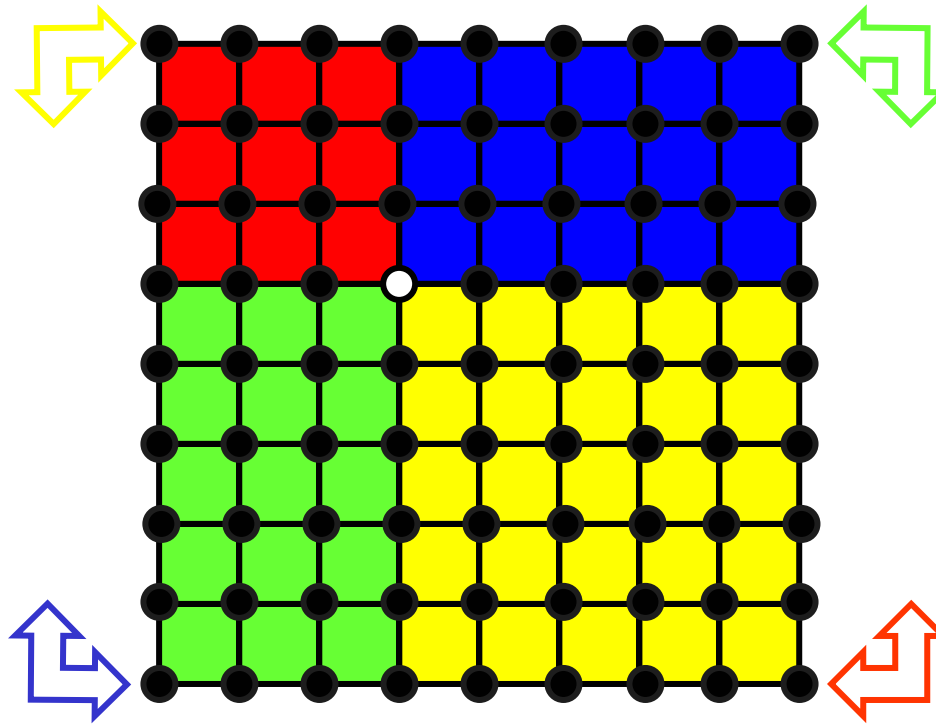
Danielsson's Algorithm

- Example: One source point



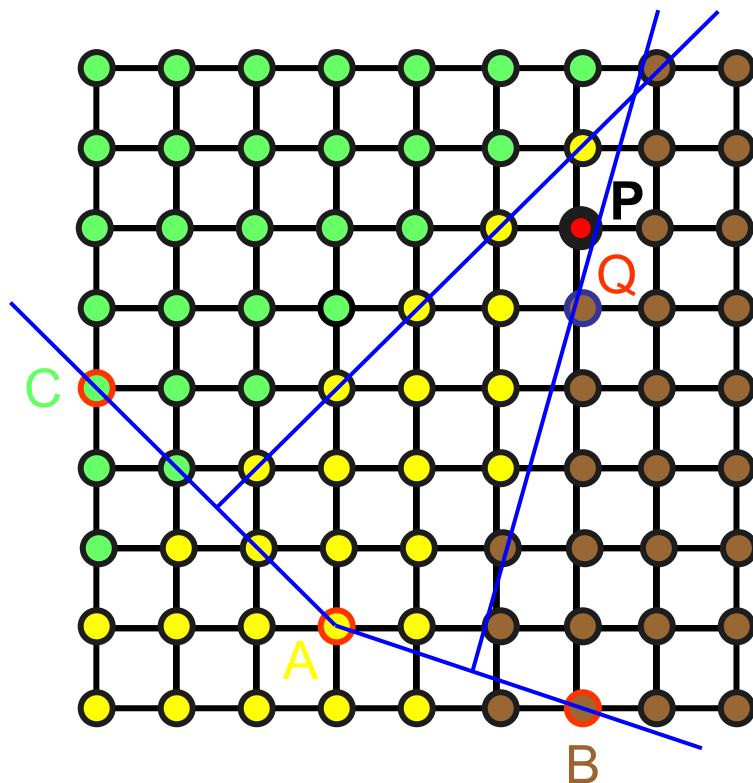
Danielsson's Algorithm

- How does it work?



Danielsson's Algorithm

- What can go awry?



$$d(Q,A) = 5$$

$$d(Q,B) = 5$$

Danielsson finds:

$$T(P) = 6$$

But actually:

$$T(P) = \sqrt{35} = 5.916$$



Danielsson's Algorithm

- This argument can be made more precise, to show that the error in the approximation is bounded by $0.29h$, where h is the mesh size.
- **Improvement:** use 8 connectivity instead of 4 connectivity.
- The error bound then becomes $0.076h$.
- However, it involves twice as much work.

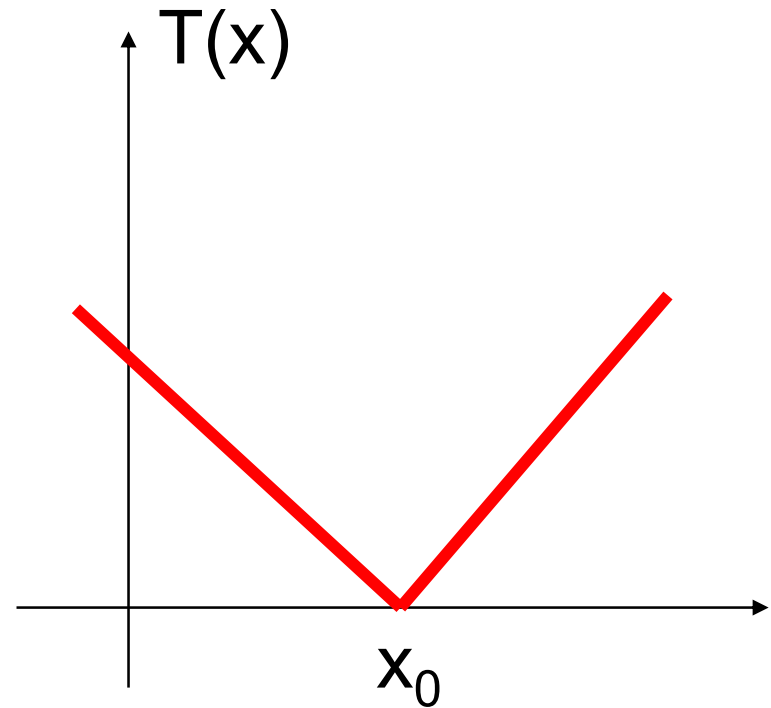


Extension to higher dimensions

- This method can be easily extended to other dimensions:
- **1-D:** 2 sweeps (left, right)
- **2-D:** 4 sweeps (left/right, up/down)
- **3-D:** 8 sweeps
- **n-D:** 2^n sweeps.

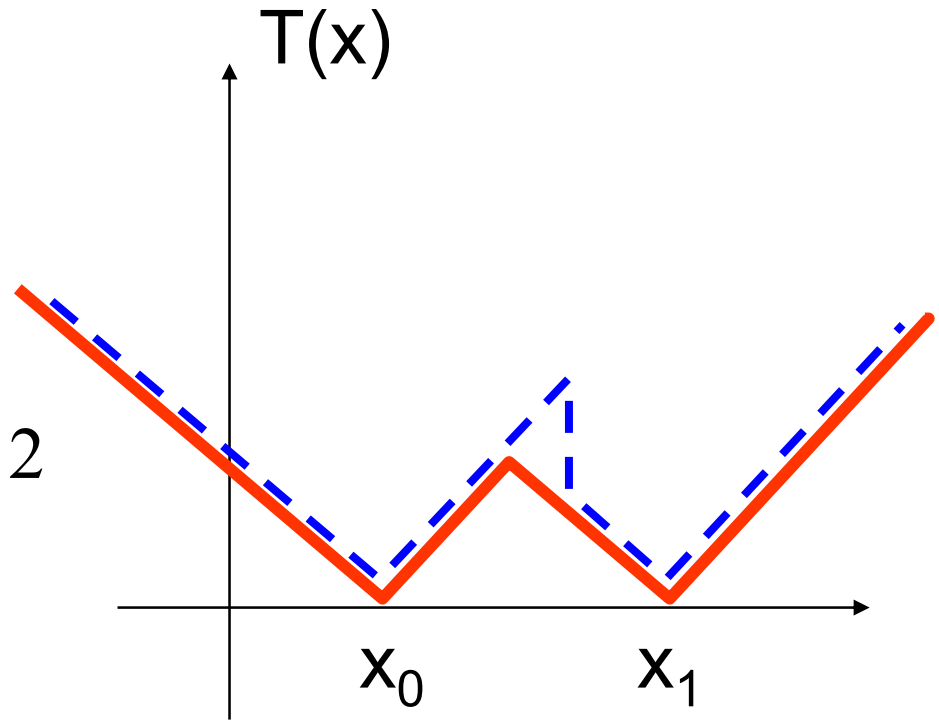
Distance Maps and Eikonal Equations

- 1 source point (x_0) .
- $T(x) = |x - x_0|$
- $|\partial_x T(x)| = 1$, except at x_0



Distance Maps and Eikonal Equations

- 2 source points (x_0, x_1) .
- $T(x) = \min\{|x-x_0|, |x-x_1|\}$
- $|\partial_x T(x)| = 1$,
except at $x_0, x_1, (x_0 + x_1)/2$
- The dashed line also satisfies $|\partial_x T(x)| = 1$,
in all but 3 points.





Distance Maps and Eikonal Equations

- So we have that for 1D distance maps:

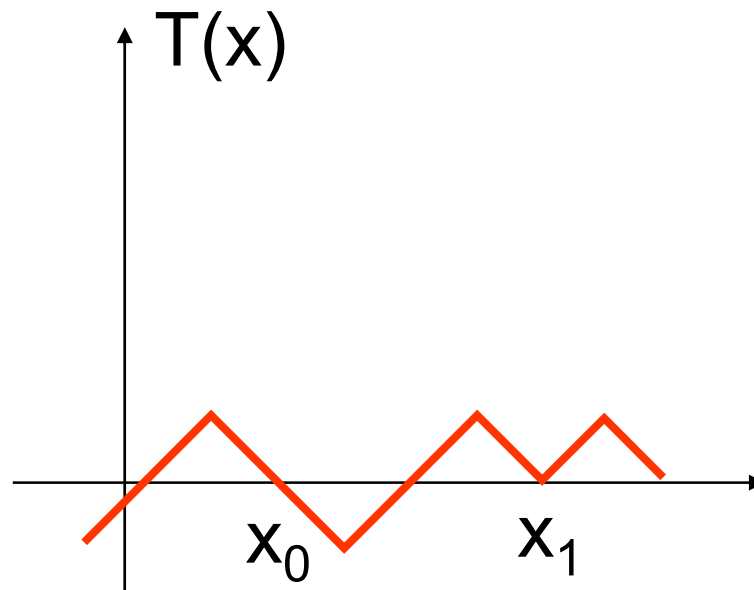
$$\begin{cases} |\partial_x T(x)| = 1, & x \in \Omega \\ T(x) = 0, & x \in \Gamma \end{cases}$$

- However, the converse is not necessarily true.
- Since the Eikonal equation does not uniquely specify a weak solution, we need to look for a specific solution – a **viscosity solution** or **entropy solution**

Discretizing the 1D differential operator

- Backward differencing:

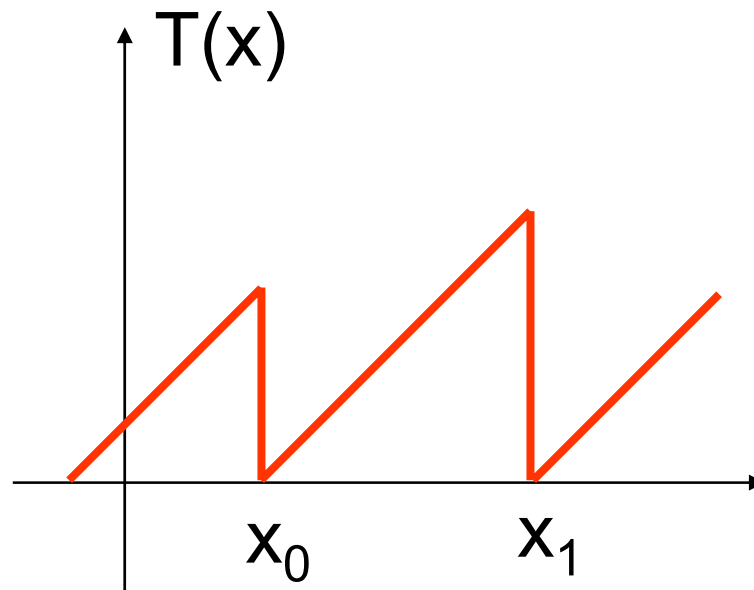
$$|\partial_x T(x)| = \left| \frac{T_i - T_{i-1}}{h} \right|$$



Discretizing the 1D differential operator

- Truncated Backward differencing:

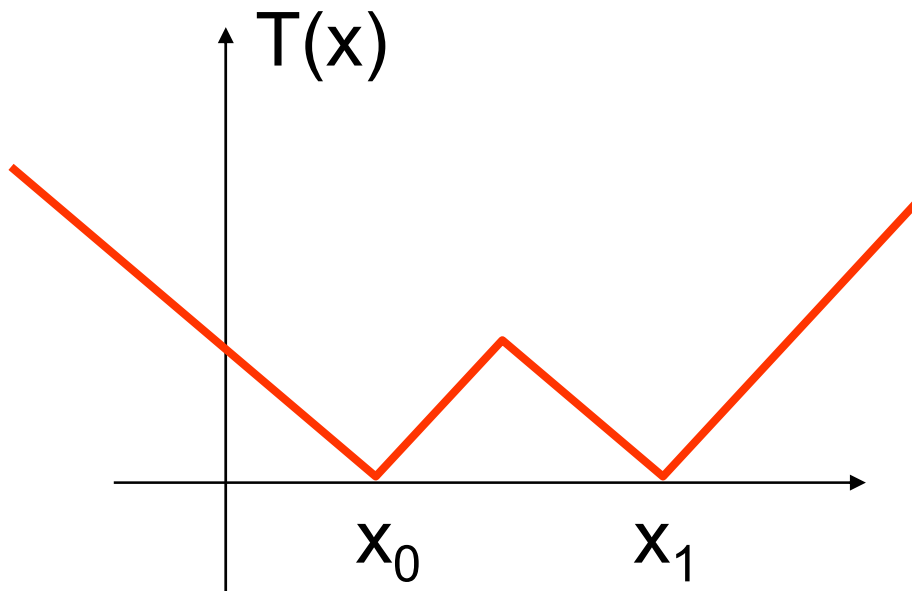
$$|\partial_x T(x)| = \left[\frac{T_i - T_{i-1}}{h} \right]^+$$



Discretizing the 1D differential operator

- Symmetrized differencing:

$$|\partial_x T(x)| = \left[\frac{T_i - T_{i-1}}{h}, \frac{T_i - T_{i+1}}{h} \right]^+$$





Discretizing the 1D differential operator

- We can rewrite this scheme as

$$|\partial_x T(x)| = \left[\frac{T_i - \min\{T_{i-1}, T_{i+1}\}}{h} \right]^+$$

- This numerical approximation is known as an **upwind** scheme, since it corresponds with the direction of information flow.
- Enforces **causality**.
- Retrieves the viscosity solution.

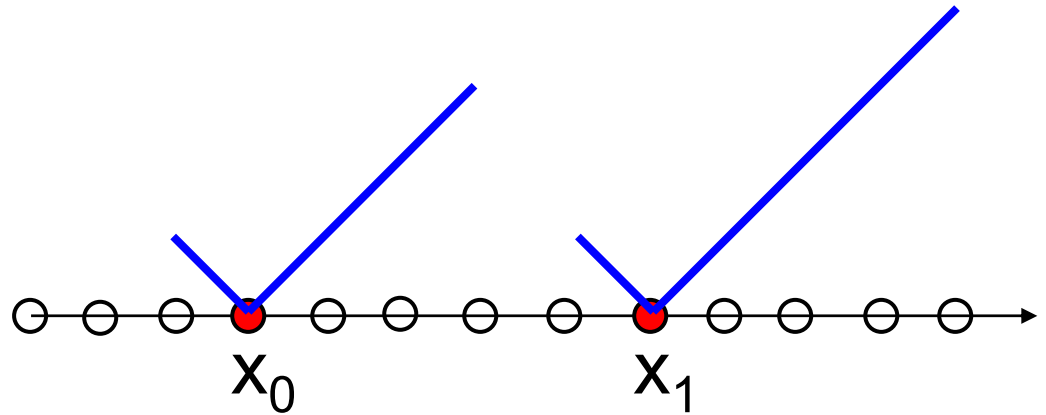


Updating order (1D case)

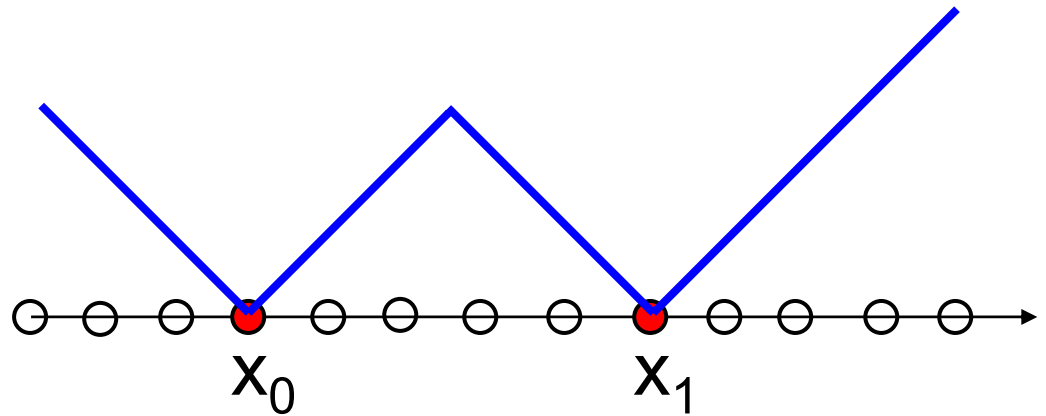
- **Q:** In which order should we scan the grid?
- **A1:** We can successively scan it from left to right. In the worst case scenario we will need N scans to converge.
- **A2:** Do a left-to-right sweep, followed by a right-to-left sweep. Convergence after 2 scans.
- **Why?** Because the distance value at any grid point can be computed exactly from its left or right neighbor.

Updating order (1D case)

Left-to-right sweep



Right-to-left sweep





n-D Eikonal Equations

- In a more general n-dimensional setting, the Eikonal equation becomes

$$\begin{cases} |\nabla T(\vec{x})| = F(\vec{x}), & \vec{x} \in R^n \\ T(\vec{x}) = 0, & \vec{x} \in \Gamma \subset R^n \end{cases}$$

- In 2D, the upwind difference scheme (a.k.a Godunov's scheme) has the form

$$\left(\left[T_{i,j} - \min \{ T_{i-1,j}, T_{i+1,j} \} \right]^+ \right)^2 + \left(\left[T_{i,j} - \min \{ T_{i,j-1}, T_{i,j+1} \} \right]^+ \right)^2 = h^2 F_{i,j}^2$$



Numerical Solution in 2D

- **Initialization:** $T(x) = 0$ for points in or near the source point set. Other points are assigned large positive values.
- **Updating:** Gauss-Seidel iterations.
- Apply Danielsson's algorithm to the Gauss-Seidel update scheme, i.e. use 4 sweeps with alternating directions (left/right, up/down).
- As we've seen before, for the case of 1 source point, 4 sweeps with alternating directions recover the exact distance function. In n-D, 2^n sweeps are required.
- When we have more than 1 source point, more than 2^n iterations may be needed for convergence.



A More General Analysis

- We consider the n-D Eikonal equation with $F = 1$, i.e. for recovering distance functions.

- Key Results:

- For a single source point x_θ , the numerical solution $T_h(x)$ converges in 2^n sweeps and satisfies

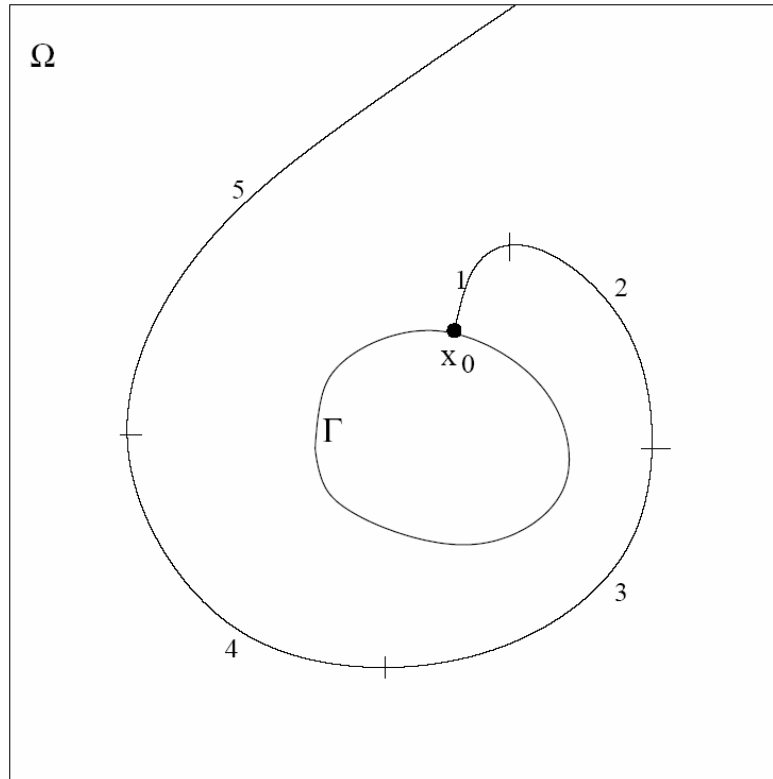
$$d(x) \leq T_h(x) \leq d(x) + O(h \log h)$$

- Let $S_h(x)$ denote the solution of the discrete Eikonal equation. For an arbitrary set of source points (not necessarily discrete), the numerical solution $T_h(x)$ after 2^n sweeps satisfies

$$S_h(x) \leq T_h(x) \leq d(x) + O(h \log h)$$

A More General Analysis

- Q: What happens when F is arbitrary?
- A: The number of iterations needed is no longer constant. It depends on the geometric structure of F .





Hierarchical Squared Distance Function

- We will now see how to use the fast sweeping algorithm in a hierarchical framework, to estimate the **squared distance function** of a surface.
- We assume we are given a triangulated surface M .
- The algorithm consists of the following 3 steps:
 1. Construct an octree encompassing the surface M .
 2. Use the fast sweeping algorithm to compute distances of corner points of cubes in the octree to the surface M .
 3. Generate a d^2 -tree, which is an octree representation of a piecewise quadratic approximation of the squared distance function of M .



Hierarchical Squared Distance Function

- We will now see how to use the fast sweeping algorithm in a hierarchical framework, to estimate the squared distance function of a surface.
- We assume we are given a triangulated surface M .
- The algorithm consists of the following 3 steps:
 - 1. Construct an octree encompassing the surface M .**

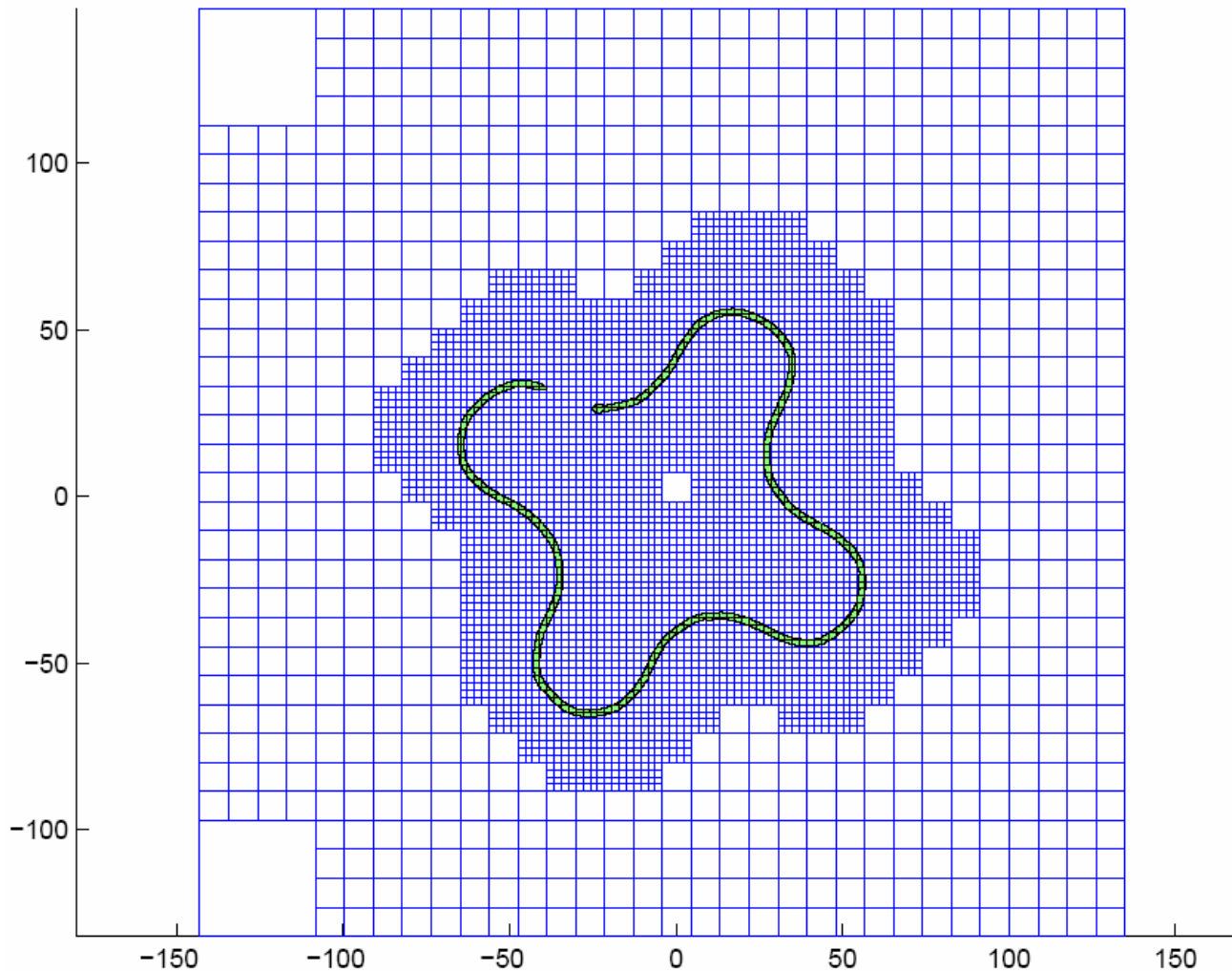


1. Constructing an Octree

- Start with a cube that encloses the object M .
- At level L (starting from $L=0$), subdivide twice, to get to level $L+2$.
- Continue in this fashion until we get to level L_{\max} , which is a precision parameter of the algorithm.
- Extending the subdivision: For each level L , certain cells C_j^L are already subdivided to level $L+2$. For each such C_j^L , subdivide all its “neighboring” cells to level $L+2$.

1. Constructing an Octree

- **Example:** A planar slice through the octree





Hierarchical Squared Distance Function

- We will now see how to use the fast sweeping algorithm in a hierarchical framework, to estimate the squared distance function of a surface.
- We assume we are given a triangulated surface M .
- The algorithm consists of the following 3 steps:
 - ✓ Construct an octree encompassing the surface M .
 - 2. Use the fast sweeping algorithm to compute distances of corner points of cubes in the octree to the surface M .**



2. Computing a distance function

- We apply the fast sweeping algorithm to compute distance values of corners of cubes, in a multilevel fashion, starting from the finest level and going up.
- **Initialization:** Run through all triangles of M that intersect a cube of level L_{\max} , and compute the distance to the corner points exactly.
- **At each level ($L_{\max}-2..0$):** Initialize with distances from finer level, and apply the sweeping algorithm on cubes of the current level.



2. Computing a distance function

- **Q:** How to do raster scans on a tree structure?
- **A:** Sort the cubes according to the order of the raster scan.
- Suppose all cells of level L are sorted in a list A_L . To sort level $L+2$, start with the list A_L , remove from it all cells that are not subdivided to level $L+2$, and sort the children of the remaining cells.



Hierarchical Squared Distance Function

- We will now see how to use the fast sweeping algorithm in a hierarchical framework, to estimate the squared distance function of a surface.
- We assume we are given a triangulated surface M .
- The algorithm consists of the following 3 steps:
 - ✓ Construct an octree encompassing the surface M .
 - ✓ Use the fast sweeping algorithm to compute distances of corner points of cubes in the octree to the surface M .
- 3. Generate a d^2 -tree, which is an octree representation of a piecewise quadratic approximation of the squared distance function of M .**



3. Computing the d^2 -tree

- We shall now construct a new octree, d^2 -tree, that will store for each cube a quadratic function of the form

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c$$

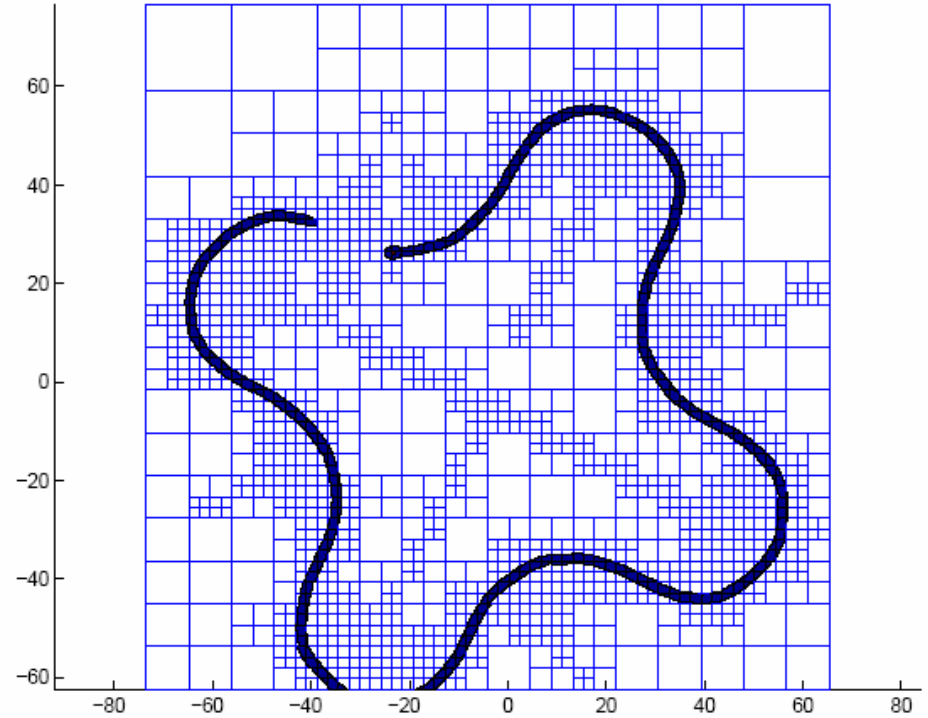
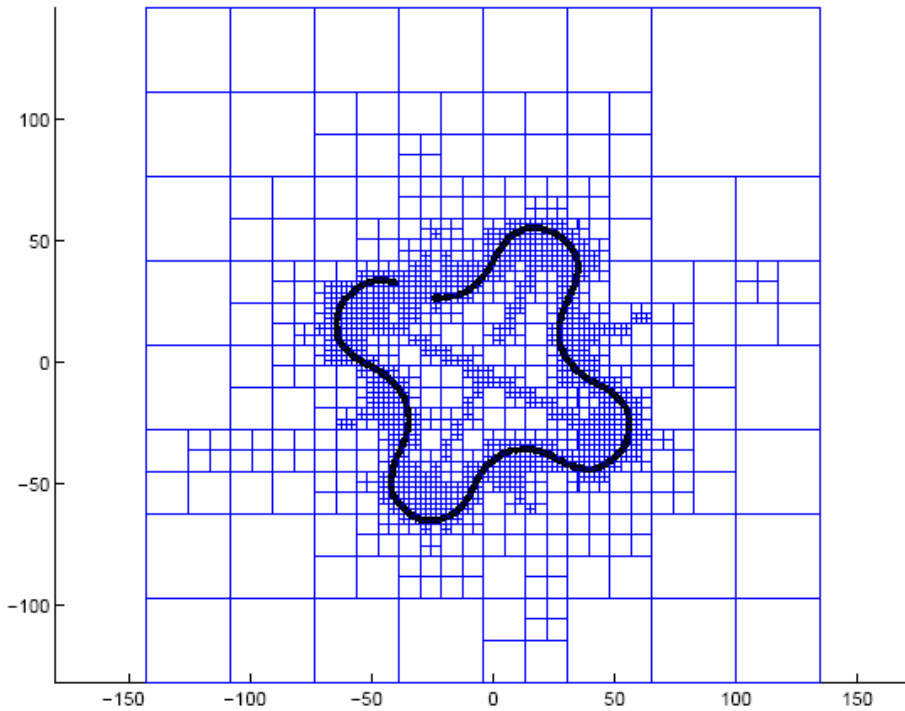
- Start off with the largest cell of the distance-octree, and compute a LS fit using all data points with known distance in the cell.
- If the residual is above a threshold, subdivide the cell, and fit a quadratic to each cell separately.
- Continue in this fashion until an adequate quadratic is obtained for each cell.



3. Computing the d^2 -tree

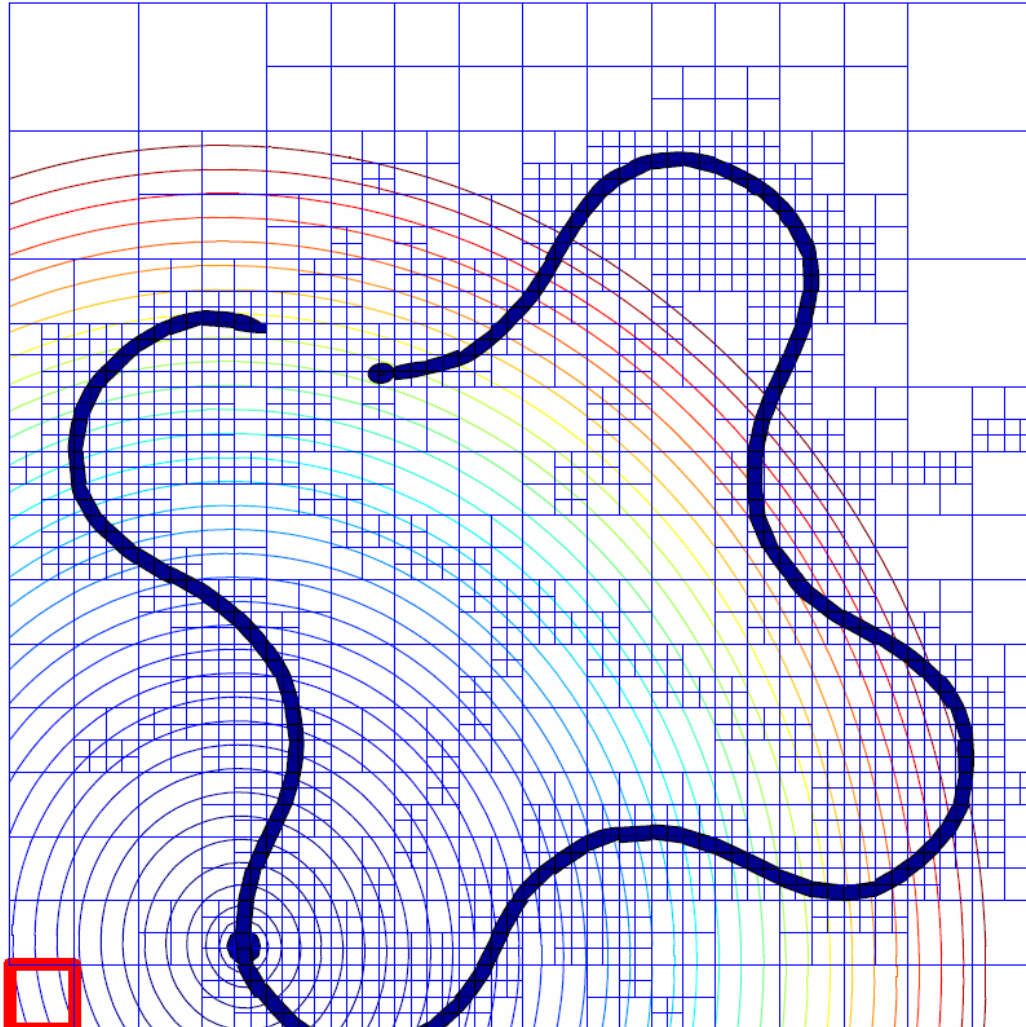
- To avoid excessive laboring in the coarser levels, rather than fitting all points at once, start by fitting only level 2 points, then, if necessary, add level 4 points, and so on.
- We end up with an d^2 -tree containing local quadratic approximations of the squared distance function.

d^2 -tree - Example



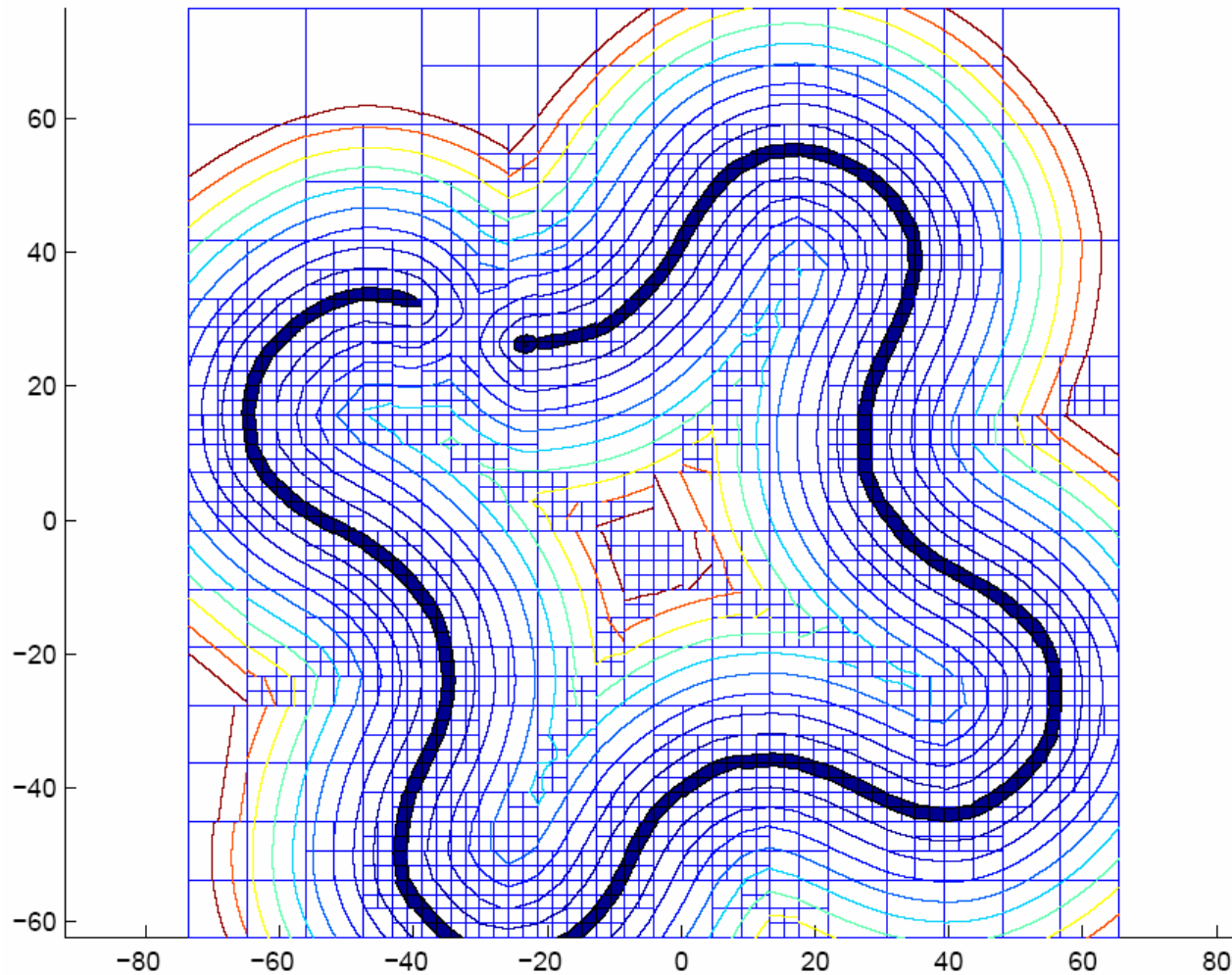
d²-tree - Example

- A local quadratic approximation with its level sets



d²-tree - Example

- Combined level sets of quadratic approximations



d^2 -tree - Example

- Piecewise-quadratic d^2 function

