CS664 Computer Vision

7. Distance Transforms

Dan Huttenlocher

Cornell University
Faculty of Computing and Information Science

Comparing Binary Feature Maps

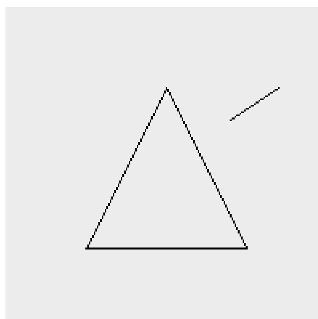
- Binary “image” specifying feature locations
  - In x,y or x,y,scale
- Even small variations will cause maps not to align precisely
- Distance transforms a natural way to “blur” feature locations geometrically
- Natural generalization also applies not just to binary data but to any cost or height map

Distance Transform

- Map of distances from any point to nearest point of some type
  - Distances to object boundaries in computer graphics, robotics and AI
  - Distances to image features in computer vision
- Generally used for data on grid
  - Pixels or voxels, 2D or 3D
  - Related to exact algorithms for Voronoi diagrams
- Efficient algorithms for computing
  - Linear in number of pixels, fast in practice

Uses of Distance Transforms

- Image matching and object recognition
  - Hausdorff and Chamfer matching
  - Skeletonization



- Path planning and navigation
  - High clearance paths

Uses of Distance Transforms

- Proximity-based matching
  - For each point of set A nearest point of set B
  - But not correspondence or one-to-one matching
  - Related to morphological dilation
    - Replace each point with disc

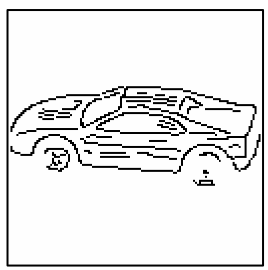
- Path planning and obstacle avoidance
  - Maximal clearance path
  - Re-compute if moving obstacles
    - But bound on how fast changes

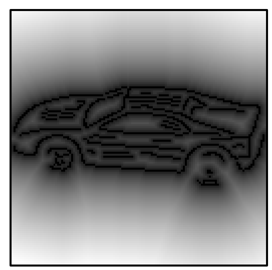
Distance Transform Formula

- Set of points, P, and measure of distance

$$DT(P)[x] = \min_{y \in P} \mathrm{dist}(x,y)$$

- For each location x distance to nearest point y in P
  - Can think of "cones" rooted at each $y \in P$
  - Min over all the cones (lower envelope)





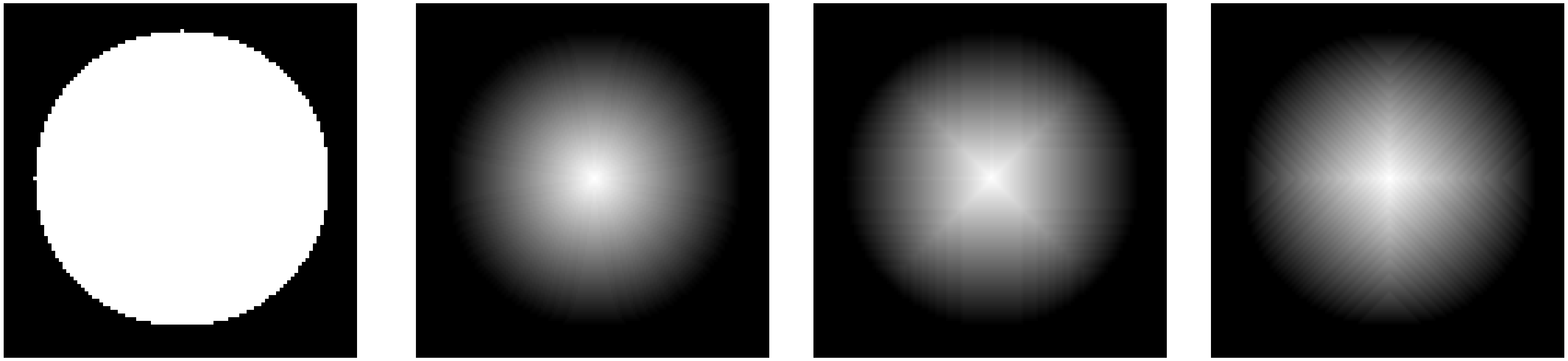
Different Distance Measures

- Euclidean distance (L_2 norm)

  $\mathrm{sqrt}((x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots)$
- City block distance (L_1 norm)

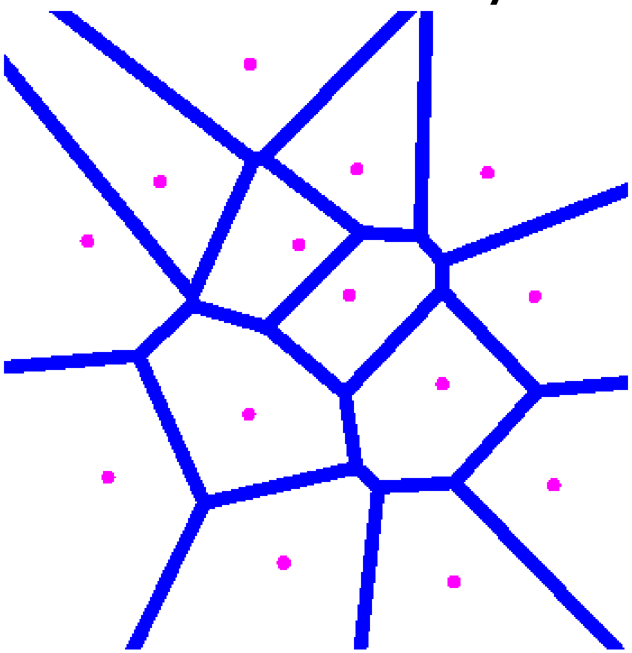
  $|x_1 - y_1| + |x_2 - y_2| + \ldots$
- Chessboard distance (L_∞ norm)

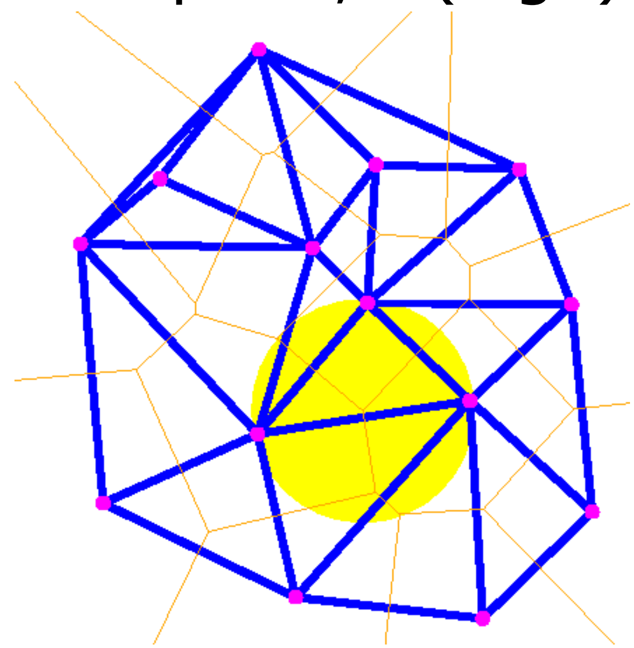
  $\max(|x_1 - y_1|, |x_2 - y_2|, \ldots)$



Relation to Voronoi Diagram

- Equidistant from two or more points
  - Dual of Delaunay triangulation
  - Compute in O(nlogn) time (Graham scan)
  - Use to efficiently find closest point, O(logn)



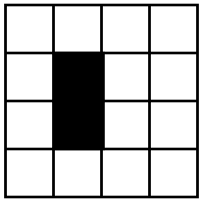


Grid Formulation of Distance Trans.

- Commonly computed on a grid Γ, for set of points $P \subseteq \Gamma$

$$DT(P)[x] = \min_{y \in \Gamma} (\mathrm{dist}(x,y) + 1_P(y))$$

- Where $1_P(y)$ indicator function for P
  - Value of 0 when $y \in P$, ∞ otherwise
  - Can think of cone rooted at each point of grid, rather than of P
  - Cones not at points of P are infinitely large so don't figure into minimum



2	1	2	3
1	0	1	2
1	0	1	2
2	1	2	3

Naïve Computation

- For each point on the grid, explicitly consider each point of P and minimize
  - For n grid points and m points in P take time O(mn)
  - Note that m is O(n), so $O(n^2)$ method
- Not very practical even for moderate size grids such as images
  - Even a low-resolution video frame has about 300K pixels
    - About 100 billion distance computations

Better Methods on Grid

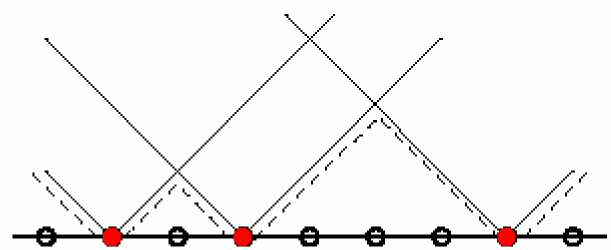
- 1D case, L_1 norm: $|x_1 - y_1| + |x_2 - y_2|$
  - Two passes:
    - Find closest point on left
    - Find closest on right if closer than one on left
  - Incremental:
    - Moving left-to-right, closest point on left either previous closest point or current point
    - Analogous for moving right-to-left
  - Can keep track of closest point as well as distance to it
    - Will illustrate distance only, less book-keeping

L_1 Distance Transform Algorithm

- Two pass O(n) algorithm for 1D L_1 norm (just distance and not source point)
  1. <u>Initialize</u>: For all j
     D[j] $\leftarrow$ 1_P[j]
  2. <u>Forward</u>: For j from 1 up to n-1
     D[j] $\leftarrow$ min(D[j],D[j-1]+1)
  3. <u>Backward</u>: For j from n-2 down to 0
     D[j] $\leftarrow$ min(D[j],D[j+1]+1)

1	0

0	1



∞	0	∞	0	∞	∞	∞	0	∞

∞	0	1	0	1	2	3	0	1

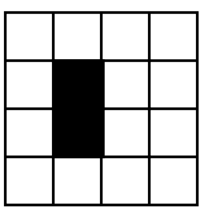
1	0	1	0	1	2	1	0	1

L_1 Distance Transform

- 2D case analogous to 1D
  - Initialization
  - Forward and backward pass
    - Forward pass adds one to closest above and to left, takes min with self
    - Backward pass analogous below and to right

0	1
1	s

s	1
1	0



∞	∞	∞	∞
∞	0	∞	∞
∞	0	∞	∞
∞	∞	∞	∞

∞	∞	∞	∞
∞	0	1	∞
∞	0	∞	∞
∞	∞	∞	∞

∞	∞	∞	∞
∞	0	1	2
∞	0	1	2
∞	1	2	3

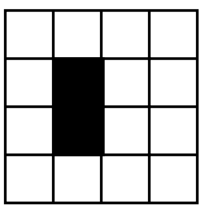
2	1	2	3
1	0	1	2
1	0	1	2
2	1	2	3

L_∞ Distance Transform

- What about Chessboard distance $\max(|x_1 - y_1|, |x_2 - y_2|)$?
- Same approach of initialization and two passes
  - Now also consider point one away on both axes

1	1
1	s

s	1
1	1



∞	∞	∞	∞
∞	0	∞	∞
∞	0	∞	∞
∞	∞	∞	∞

∞	∞	∞	∞
∞	0	1	∞
∞	0	∞	∞
∞	∞	∞	∞

∞	∞	∞	∞
∞	0	1	2
∞	0	1	2
∞	1	1	2

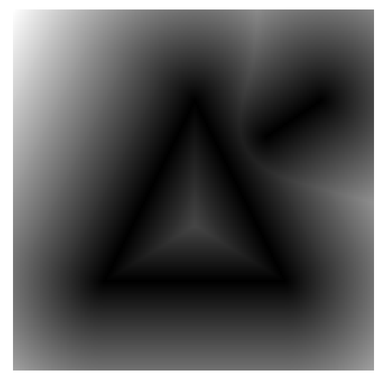
1	1	1	2
1	0	1	2
1	0	1	2
1	1	1	2

L_2 Distance Transform

- What about Euclidean distance $\mathrm{sqrt}((x_1 - y_1)^2 + (x_2 - y_2)^2)$?
- Not linear function of location on grid
  - Simple local propagation methods not correct

$\sqrt{2}$	1
1	s

- Local propagation just approximation
  - Introduces considerable error, particularly at larger distances
  - Bigger neighborhood can help but not fix



Exact L_2 Distance Transform

- 1D case doesn't seem helpful
  - Same as L_1
  - But just saw 2D case not same as L_1
- Several quite involved methods
  - Linear or O(nlogn) time, but at edge of practical
- Revisit 1D
  - Decompose 2D into two 1D transforms
  - Yield relatively simple method, though not local
  - Requires more advanced way of understanding running time – amortized analysis

Squared Distance on 2D Grid

- Consider $f(x,y)$ on grid
  - For instance, indicator function for membership in point set P, 0 or ∞
- Distance transform

$$D_f(x,y) = \min_{x',y'}((x-x')^2 + (y-y')^2 + f(x',y'))$$

- First term does not depend on y'

$$= \min_{x'}((x-x')^2 + \min_{y'}((y-y')^2 + f(x',y')))$$

- But then can view as 1D distance transform restricted to column indexed by x'

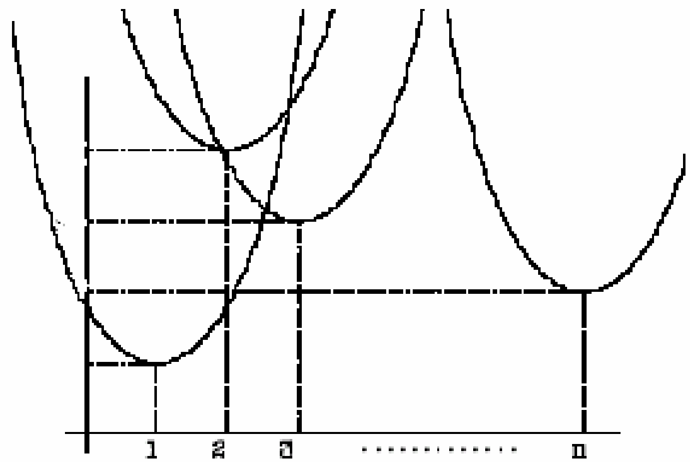
$$= \min_{x'}((x-x')^2 + D_{f|x'}(y))$$

Approach for L_2 Distance Transform

- Start with point set on grid
- Initialize to $0,\infty$ cost function
- Perform 1D transform on columns of cost function
- Perform 1D transform on rows of result
  - Cascade results in each dimension
- Compute square roots if actual distance needed
  - Note, as does not change minima, often more efficient to leave as squared distance

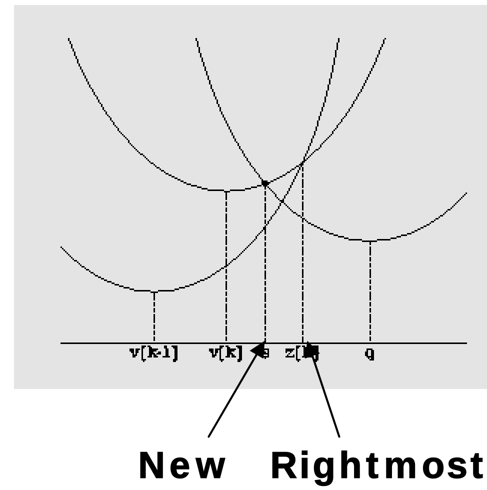
Computing 1D L_2^2 Transform Efficiently

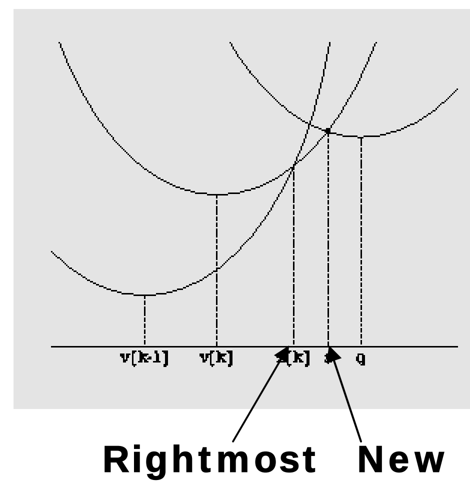
- Compute $h(x)=\min_{x'} ((x-x')^2+f(x'))$
- Intuition: each value defines a constraint
  - Geometric view: in one dimension, lower envelope of arrangement of n quadratics
    - Each rooted at (x,f(x))
      - Related to convex hull in computational geometry



Algorithm for 1D Lower Envelope

- Incrementally add quadratics
  - Keep only those “lower envelope”
    - Maintain ordered list of visible quadratics and the intersections of successive ones
- Consider in left-to-right order
  - Compare new intersection with rightmost quadratic to rightmost existing intersection
    - If to left, hides rightmost quadratic so remove and repeat





Running Time of LE Algorithm

- Consider adding each quadratic just once
  - Intersection and comparison constant time
  - Adding to lists constant time
  - Removing from lists constant time
    - But then need to try again
- Amortized analysis
  - Total number of removals O(n)
    - Each quadratic, once removed, never considered for removal again
- Thus overall running time O(n)

1D L_2^2 Distance Transform

```
static float *dt(float *f, int n) {
  float *d = new float[n], *z = new float[n];
  int *v = new int[n];
  int k = 0;
  v[0] = 0;
  z[0] = -INF;
  z[1] = +INF;
  for (int q = 1; q <= n-1; q++) {
    float s  = ((f[q]+square(q))-(f[v[k]]+square(v[k])))
                 /(2*q-2*v[k]);
    while (s <= z[k]) {
      k--;
      s  = ((f[q]+square(q))-(f[v[k]]+square(v[k])))
             /(2*q-2*v[k]);    }
    k++;
    v[k] = q;
    z[k] = s;
    z[k+1] = +INF; }
```

DT Values From Intersections

```
k = 0;
  for (int q = 0; q <= n-1; q++) {
    while (z[k+1] < q)
      k++;
    d[q] = square(q-v[k]) + f[v[k]];
  }
  return d;
}
```

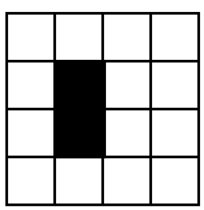
- 2D version easily runs at video rates
- No reason to approximate L_2 distance
  - Simple to implement as well as fast

Distance Transforms in Matching

- Chamfer measure – asymmetric
  - Sum of distance transform values
    - “Probe” DT at locations specified by model and sum resulting values
- Hausdorff distance (and generalizations)
  - Max-min distance which can be computed efficiently using distance transform
  - Generalization to quantile of distance transform values more useful in practice
    - Max sensitive to even single outlier

DT and Morphological Dilation

- Dilation operation replaces each point of P with some fixed point set Q
  - $P \oplus Q = \cup_{\mathbf{p}} \cup_{\mathbf{q}}\ p+q$
- Dilation by a "disc" C^d of radius d replaces each point with a disc
  - A point is in the dilation of P by $C^{\mathbf{d}}$ exactly when the distance transform value is no more than d (for appropriate disc and distance fcn.)
  - $x \in P \oplus C^{\mathbf{d}} \Leftrightarrow D_{\mathbf{P}}(x) \leq d$



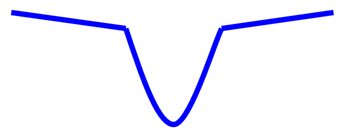
2	1	2	3
1	0	1	2
1	0	1	2
2	1	2	3

0	1	0	0
1	1	1	0
1	1	1	0
0	1	0	0

1	1	1	0
1	1	1	1
1	1	1	1
1	1	1	0

Generalizations of DT

- Combination distance functions
  - Robust "truncated quadratic" distance
    - Quadratic for small distances, linear for larger
    - Simply minimum of (weighted) quadratic and linear distance transforms



- DT of arbitrary functions: $\min_y \| x-y \| + f(y)$
  - Exact same algorithms apply
  - Combination of cost function f(y) at each location and distance function
    - Useful for certain energy minimization problems