

# CS 664 Lecture 2

## Distance Transforms



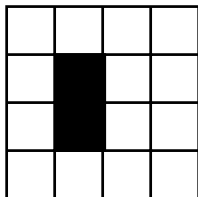
**Prof. Dan Huttenlocher**  
**Fall 2003**

# Distance Transforms

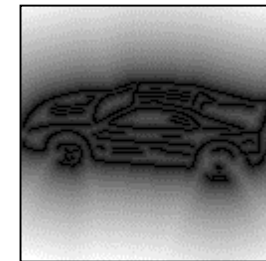
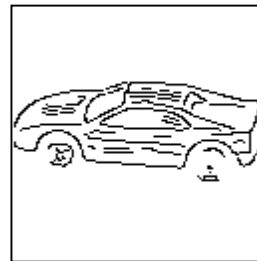
- Map of distance to nearest features
  - Computed from map of feature locations
    - E.g., edge detector output
- Powerful and widely applicable
  - Can think of as “smoothing in feature space”
  - Related to morphological dilation operation
  - Often preferable to explicitly searching for correspondences of features
- Efficiently computable using DP
  - Time linear in number of pixels, fast in practice

# Distance Transform Definition

- Set of points,  $P$ , some distance  $\|\bullet\|$   
$$D_P(x) = \min_{y \in P} \|x - y\|$$
  - For each location  $x$  distance to nearest  $y$  in  $P$
  - Think of as cones rooted at each point of  $P$
- Commonly computed on a grid  $\Gamma$  using  
$$D_P(x) = \min_{y \in \Gamma} (\|x - y\| + 1_P(y))$$
  - Where  $1_P(y) = 0$  when  $y \in P$ ,  $\infty$  otherwise



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



# DP for $L_1$ Distance Transform

- 1D case
  - 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 moving right-to-left for closest point on right
  - Can keep track of closest point as well as distance to it
    - Will illustrate distance; point follows easily

# L<sub>1</sub>Distance Transform Algorithm

- Two pass O(n) algorithm for 1D L<sub>1</sub> norm (for simplicity just distance)

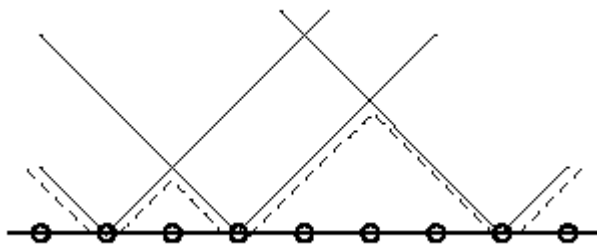
1. Initialize: For all j  
 $D[j] \leftarrow 1_p[j]$

2. Forward: For j from 1 up to n-1  
 $D[j] \leftarrow \min(D[j], D[j-1]+1)$

1 0

3. Backward: For j from n-2 down to 0  
 $D[j] \leftarrow \min(D[j], D[j+1]+1)$

0 1



$\infty$	0	$\infty$	0	$\infty$	$\infty$	$\infty$	0	$\infty$
----------	---	----------	---	----------	----------	----------	---	----------

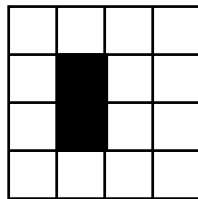
$\infty$	0	1	0	1	2	3	0	1
----------	---	---	---	---	---	---	---	---

1	0	1	0	1	2	1	0	1
---	---	---	---	---	---	---	---	---

# $L_1$ Distance Transform

- 2D case analogous to 1D
  - Initialization
  - Forward and backward pass
    - Fwd pass finds closest above and to left
    - Bwd pass finds closest below and to right
- Note nothing depends on  $0, \infty$  form of initialization
  - Can “distance transform” arbitrary array

-	1
1	0
0	1
1	-



$\infty$	$\infty$	$\infty$	$\infty$
$\infty$	0	$\infty$	$\infty$
$\infty$	0	$\infty$	$\infty$
$\infty$	$\infty$	$\infty$	$\infty$

$\infty$	$\infty$	$\infty$	$\infty$
$\infty$	0	1	$\infty$
$\infty$	0	$\infty$	$\infty$
$\infty$	$\infty$	$\infty$	$\infty$

$\infty$	$\infty$	$\infty$	$\infty$
$\infty$	0	1	2
$\infty$	0	1	2
$\infty$	1	2	3

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

# $L_2$ Distance Transform

- Approximations using fixed size masks
  - Analogous to 1D case
  - Simple to understand but not best methods
- Exact linear time method for  $L_2^2$ 
  - Can compute sqrt (but usually not needed)
  - Fast in practice, easy to implement
  - Harder to understand than  $L_1$  algorithm
  - Uses important general algorithmic technique of amortized analysis
- 1D case – lower envelope of quadratics

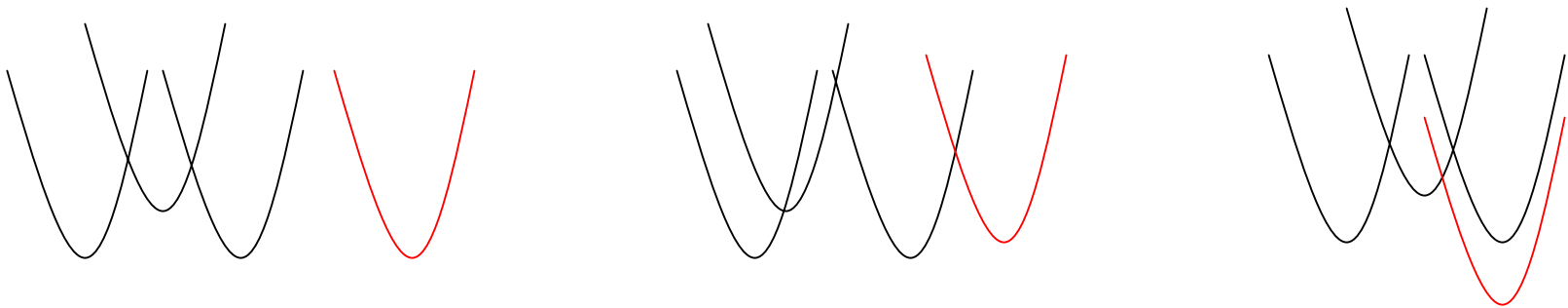
# 1D $L_2^2$ Distance Transform

- Single left-to-right pass
  - Adding  $k$ -th quadratic to lower envelope (LE) of first  $k-1$  quadratics
  - Quadratics differ only in location of their base
- Concerned about intersection of  $k$ -th quadratic and LE of first  $k-1$ 
  - Consider only rightmost quadratic visible in LE
  - Keep track of locations of bases of *visible quadratics* (VQ), ordered left-to-right
  - Keep track of *visible intersections* of adjacent quadratics (VI), ordered left-to-right



# Adding k-th Quadratic to LE

- Case 1: intersection of k and rightmost VQ (RVQ) outside range, k not visible on LE
- Case 2: intersection of k and RVQ to right of rightmost VI (RVI), k added to right
- Case 3: intersection of k and RVQ to left of RVI, k covers at least RVQ, remove RVQ and try adding again



# Running Time of 1D Algorithm

- Traditional analysis would consider time for each case, multiplied by  $n$  iterations
  - Cases 1 and 2  $O(1)$ , but case 3 ??
- Amortized analysis: charge work done by algorithm to “events” that can be bounded
  - Three event types
    - $K$ -th quadratic initially excluded
    - $K$ -th quadratic added
    - $K$ -th quadratic removed
  - Each event happens at most once per quadratic (note once removed, never again)
  - Algorithm does constant work per event

# 2D Algorithm

- Horizontal pass of 1D algorithm
  - Computes minimum  $x^2$  distance
- Vertical pass of 1D algorithm on result of horizontal pass
  - Computes minimum  $x^2+y^2$  distance
  - Note algorithm applies to any input (quadratics can be at any location)
- Actual code straightforward and fast
  - Each pass maintains arrays of indexes of visible parabolas and the intersections
  - Fills in distance values at each pixel after determining which parabolas visible

# Horizontal Pass of 2D $L_2^2$ DT

```
for (y = 0; y < height; y++) {
  k = 0; /* Number of boundaries between parabolas */
  z[0] = 0; /* Indexes of locations of boundaries */
  z[1] = width; /* No current boundaries (first at end of array) */
  v[0] = 0; /* Indexes of locations of visible parabola bases */
  for (x = 1; x < width; x++) {
    do {
      /* intersection of this parabola with rightmost visible parabola */
      s = ((imRef(im, x, y) + x*x) - (imRef(im, v[k], y) + v[k]*v[k])) /
          (2 * (x - v[k]));
      sp = ceil(s);
      /* case one: intersection off end, this parabola not visible */
      if (sp >= width)
        break;
      /* case two: intersection is rightmost, add it to end*/
      if (sp > z[k]) {
        z[k+1] = sp; z[k+2] = width; v[k+1] = x; k++;
        break; }
      /* case three: intersection is not rightmost, hides rightmost
         parabola and perhaps others, remove rightmost and try again */
      if (k == 0) {
        v[0] = x; break;
      } else {
        z[k] = width; k--; }
    } while (1);
  }
}
```

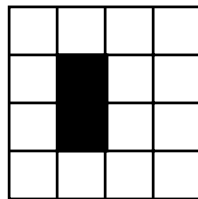
# DT Values From Intersections

```
/* get value of input image at each parabola base */
for (x = 0; x <= k; x++) {
    vref[x] = imRef(im, v[x], y);
}
k = 0;
/* iterate over pixels, calculating value for closest parabola */
for (x = 0; x < width; x++) {
    if (x == z[k+1])
        k++;
    imRef(im, x, y) = vref[k] + (v[k]-x)*(v[k]-x);
}
```

- No reason to approximate  $L_2$  distance!
- Code available at [www.cs.cornell.edu/~dph/matchalgs/](http://www.cs.cornell.edu/~dph/matchalgs/)

# DT and Morphological Dilation

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

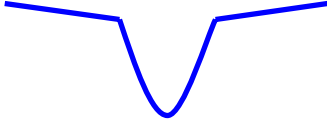


2	1	2	3
1	0	1	2
1	0	1	2
2	1	2	3

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

# 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
- Iterated closest point (ICP) like methods
  - Fitzgibbons