Distance Transform

Outline

• What is distance transform?
  • Distance metrics
• Different approaches for distance transform:
  • Naïve approach
  • Morphology-based approach
  • Dynamic programming approach

Applications of Distance Transform

• Path planning and obstacle avoidance:
  • Find maximal clearance paths in robotics & games
• Skeletonization:
  • Compute the skeleton of a given shape
• Image matching and object recognition
  • Compute Hausdorff distance between two shapes

\[ H(A, B) = \max_{a \in A} \min_{b \in B} D(a, b) \]

Distance Metrics

• Define the distance between pixels (p,q) & (s,t)
• Euclidean distance:
  \[ D_{\text{Eucl}} = (|p-s|^2 + |q-t|^2)^{1/2} \]
  \( L_2 \) norm
• City block (a.k.a. taxicab) distance:
  \[ D_{\text{City}} = |p-s| + |q-t| \]
  \( L_1 \) norm
• Chessboard distance:
  \[ D_{\text{Chess}} = \max(|p-s|, |q-t|) \]
  \( L_{\infty} \) norm

Effects of Different Metrics

<table>
<thead>
<tr>
<th>City block</th>
<th>Chessboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 2 2 2 1</td>
<td>1 1 2 1 1</td>
</tr>
<tr>
<td>1 2 3 3 2 1</td>
<td>1 1 2 2 1 1</td>
</tr>
<tr>
<td>1 2 3 3 2 1</td>
<td>1 2 3 3 2 1</td>
</tr>
<tr>
<td>1 2 2 2 2 1</td>
<td>1 2 2 2 2 1</td>
</tr>
<tr>
<td>1 1 1 1 1 1</td>
<td>1 1 1 1 1 1</td>
</tr>
</tbody>
</table>
Naïve Approach

- Perform transform according to the definition:
  - For each pixel in the foreground:
    - Calculate its distance to all features
    - Keep the minimum of the distance
    - Can output both the minimum distance and the closest feature
  - Works for all distance metrics
- Complexity:
  - $O(M \times N \times K)$
  - $K$ is the number of features

Implementation for Naïve Approach

```java
for (int q=0; q<height; q++)
  for (int p=0; p<width; p++) {
    if (image.getRGB(p,q) == Color.black.getRGB()) {
      D[q][p] = 0; continue;
    }
    D[q][p] = radius;
    for (int t=0; t<height; t++)
      for (int s=0; s<width; s++)
        if (image.getRGB(s,t) == Color.black.getRGB()) {
          if (method.equals("Euclidean")
            dist = (int)Math.sqrt((s-p)*(s-p)+(t-q)*(t-q));
          else if (method.equals("Chessboard")
            dist = Math.max(Math.abs(s-p),Math.abs(t-q));
          else
            dist = Math.abs(s-p) + Math.abs(t-q);
          D[q][p] = Math.min(D[q][p], dist);
        }
    max = Math.max(max, D[q][p]);
  }
```

Morphology-based Approach

- Use erosion operator:
  - Perform multiple successive erosions with a suitable structuring element until all foreground regions of the image have been eroded away
  - Label each pixel with the number of erosions that had to be performed before it disappeared

Morphology-based Approach (Cont’d)

- Which structuring element should be used depends on the distance metrics
  - City block distance:
    - 4 neighbors -> 3x3 cross
  - Chessboard distance:
    - 8 neighbors -> 3x3 square
- Complexity:
  - $O(M \times N \times K)$
  - $K$ is the number of erosion passes needed
    - Equals to the maximum distance in the result transform

Dynamic Programming Approach

- The basic idea in 1D case
  - Use two passes:
    - Forward pass calculates the distance to the closest feature on the left
    - Backward pass calculates the distance to the closest feature on the right
  - Incremental approach:
    - Calculate from left to right in the forward pass
    - Calculate from right to left in the backward pass
  - Can output both the minimum distance and the closest feature

An Example in 1D Case

### Forward pass
```
∞ 0 0 0 0 0 0 ∞
```

### Backward pass
```
1 1 1 1 1 1 1
```

Black indicates features or background pixels
2D Cases (City Block Distance)

Check top & left in the forward pass

Check bottom & right in the backward pass

Implementation for Dynamic Programming

Examples I

Examples II