Scribe: Junhwan Kim[1] Lecturer: Ramin Zabih

1 Image restoration

Regularization is one method to solve the image restoration problem, i.e. the attempt to recover the uncontaminated image from the image that resulted from the image formation process. For example, assume we point the camera at a patch of (textureless) wall that is evenly lit from a distance (see Figure (1)).

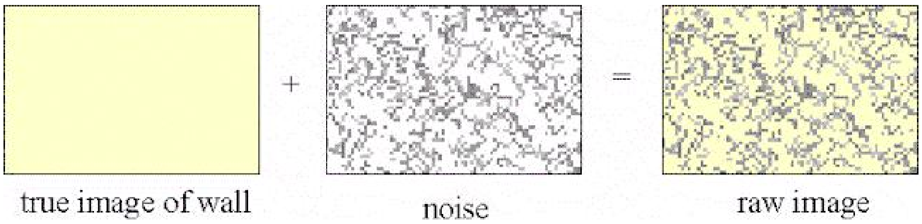

Figure 1: Image contamination (from http://www.cs.cornell.edu/CS664/regularization.htm)

Clearly, we cannot reconstruct the truth from the raw data without making further assumptions about the true image and the noise. Image restoration is thus an ill-posed problem.

Regularization problem can be stated as follows: *Given an input, compute "true" intensities* We need to make assumptions about what a true solution looks like, otherwise, actual observed measurement might be the true solution. For instance, we can assume that true answer is piecewise constant. This is an example of spatial coherence, i.e., pixels are *mostly like* their neighbors.

2 Local Filtering

Local smoothing operations, one of the classic computer vision algorithms, work by picking a local window, assume that it has diameter $d = 2r + 1(odd)$, and compute for each pixel a function of the nearby pixels. In 1D, for input array $in(x)$, we have $out(x) = mean(in([x - r : x + r]))$. Similarly in 2D, for input array $in(x, y)$, we have $out(x, y) = mean2(in([x - r : x + r], [y - r : y + r]))$.

2.1 Median vs Average

Instead of average, we might take median. This helps with ramp problem but not cornering or thin object problems. On the other hand, it is less stable in continuous regions. Also it has efficiency problems, i.e., we cannot do such clever dynamic programming trick as described below.

[1]This subscription is based on Amy Gale's subscription

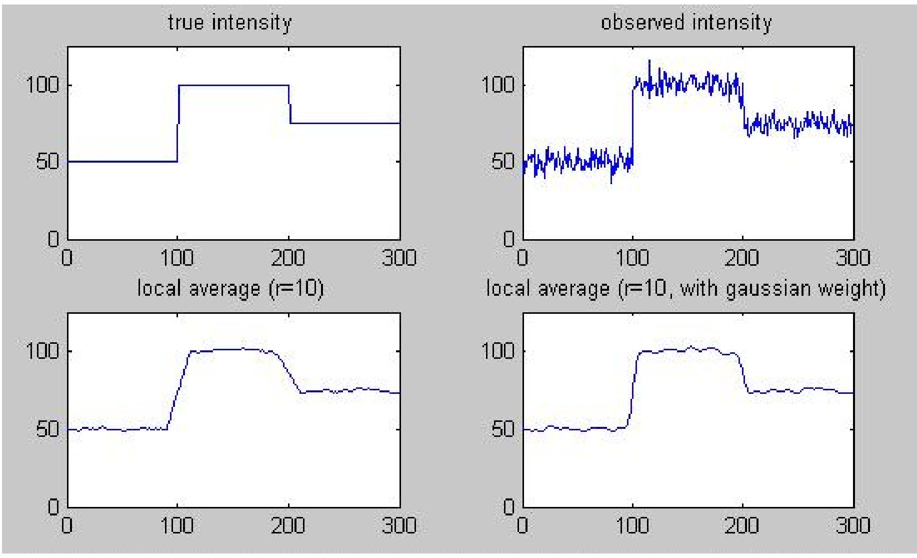

Figure 2: Local smoothing in 1D

2.2 Dynamic programming for averaging

In 1D, if we compute average in the obvious way, the cost is $O(nd)$, where n is the number of pixels and d is size of the window. We can cut down to $O(n)$ using dynamic programming, i.e. $out(x) = out(x-1) - in(x-r-1)/d + in(x+r)/d$. In 2D, we need to store the column sums (First programming assignment).

At the boundaries, we can either (1) append dummy pixels , (2) pretend it wraps on a torus, or (3) do something else reasonable.

2.3 Disadvantage of averaging

Far from discontinuities, local averaging gives a reasonable answer. On the other hand, near discontinuities, we get 'ramps' (See Fig. (2)). In 2D, a thin object will be spread out. If an object has a corner, it will be dimmed (corner becomes rounded)(3).

2.4 Discontinuity and value

Imagine we had a discontinuity oracle. Then we could do a much better job, i.e., take the average of pixel with neighbors on same side of any discontinuity. In that case, we can compute the value easily. But also vice versa. A value oracle allows much easier computation of discontinuities.

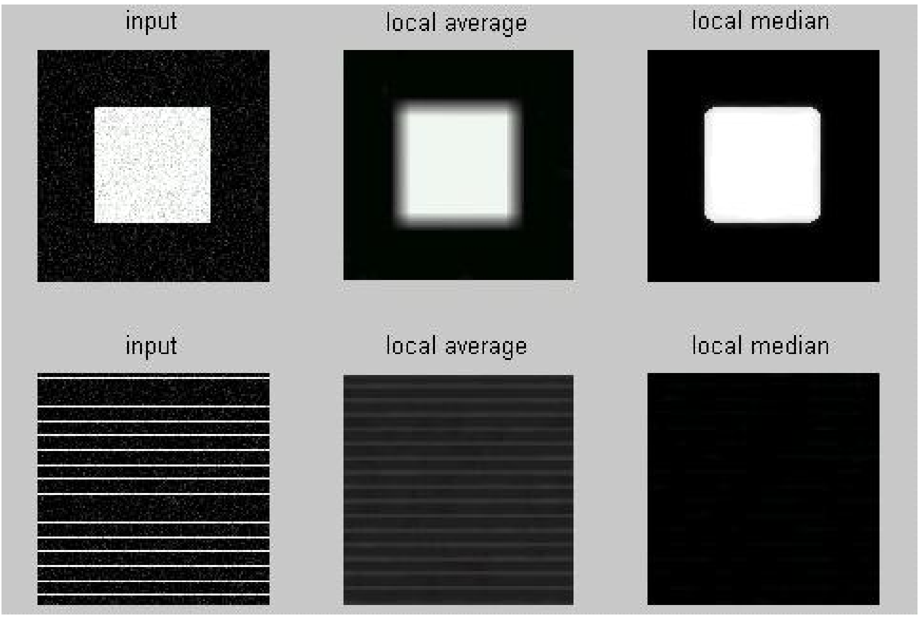

Figure 3: Local smoothing in 2D

2.5 Weighted average

Averaging of local pixels assumes that all pixels are equally important. Problem with this is that it ignores the fact that close neighbors tend to be relevant. Worse yet, in 2D, since we have number of neighbors square of distance, more neighbors further away than close ones. The larger the window, the wider the ramp, the less idea we have about the exact location of the discontinuity.

One obvious solution is to reduce the weight of the further-away pixels. For example, consider a 3 element local average. More generally, $out(x) = a * in(x-1) + b * in(x) + c * in(x+1)$ for $[a, b, c] = [1/3, 1/3, 1/3]$. In case of 5 element local average, the stencil is $[1/5, 1/5, 1/5, 1/5, 1/5]$. Instead, we can use a weighting that decreases as we get further away from the center, e.g. $[0.1, 0.2, 0.4, 0.2, 0.1]$. This has the effect of narrowing the ramp.

Let stencil S_r for window size $2r + 1$, with assumption of symmetry:

$$out(x) = \sum_{j=-r}^{r} S_r(j) \cdot in(x - j) \tag{1}$$

We call Eq.(1) *convolution*, or generalization of local averaging.

3 Convolution in Vision

In vision, convolution is used for (1) smoothing, (2) edge detection, (3) texture analysis, (4) motion measurement. We often write convolution as $out(x) = in(x) \otimes S_r(x)$.

3.1 Properties of convolution

Convolution has the following properties:

- Commutative: $A \otimes B = B \otimes A$. In the context of smoothing, it implies smoothing stencil with image = smoothing image with stencil.
- Associative: $(A \otimes B) \otimes C = A \otimes (B \otimes C)$

Linear operation B on an image takes in an image and gives back another image such that for any images I, I', and scalars α, β, we have

$$(B \circ (\alpha I + \beta I'))(x) = \alpha(B \circ I)(x) + \beta(B \circ I')(x)$$

For a linear operation, input and output are bounded. Average and weighted operations are linear operations on an image, whereas median isn't. Problems that arises from linear operations include difficulties at discontinuities.

Any local operations tend to be shift invariant, i.e., the exact position of pixel doesn't matter very much.

If not shift invariant, we are caring about exact pixel locations (most of sensible operations are shift invariant). JPEG compression is not shift invariant. We can observe the artifacts as "buzzing" in compression video. Formally,

$$SHIFT_\Delta I(x) = I(x - \Delta)$$

Linear operation B is shift invariant iff

$$SHIFT_\Delta (B \circ I)(x) = (B \circ (SHIFT_\Delta (I)))(x)$$

3.2 Equivalency of LSI and convolution

- Fact: Convolution is an LSI (linear, shift invariant) operation.
- Fact: Any LSI operation is convolution.
- Proof: In the next lecture