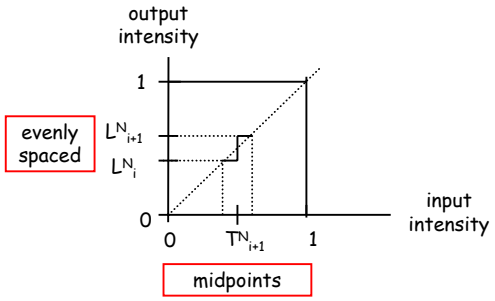
uniform quantization: N levels

- output levels:
  evenly spaced
- thresholds:
  midpoints

N level uniform quantization



uniform quantization: N level

- output levels:
  $L^N_i = i/(N-1)$, i = 0, ,..., N-1
- thresholds:
  $T^N_i = (L^N_{i-1} + L^N_i)/2 = (2i-1)/2(N-1)$, i = 1, ..., N-1

quantization error

8 bits per channel per pixel

1 bits per channel per pixel

dithering

- technique for camouflaging error
- algorithms
  - random
  - ordered
  - error diffusion

comparison

original 8 bits/pixel/ channel

random

ordered

error-diffusion

1 bit/pixel/channel

random dither

add noise to camouflage quantization artifacts

8 bits/pixel/channel | 1 bits/pixel/channel | 1 bits/pixel/channel dithered

random dither

for each pixel in the input image
- add random noise to pixel value
- uniformly quantize new value

random dither

add noise to camouflage quantization artifacts

8 bits/pixel/channel | 1 bits/pixel/channel | 1 bits/pixel/channel dithered

ordered dither

add pseudo-random noise to camouflage quantization artifacts

8 bits/pixel/channel | 1 bits/pixel/channel | 1 bits/pixel/channel dithered

ordered dither intuition: 2 level output

suppose all 2x2 neighborhoods were uniform

.2	.2	.6	.6	.3	.3
.2	.2	.6	.6	.3	.3
.4	.4	.9	.9	.2	.2
.4	.4	.9	.9	.2	.2

2 level output: quantized blocks

every neighborhood is quantized in one of two ways

0	0
0	0

1	1
1	1

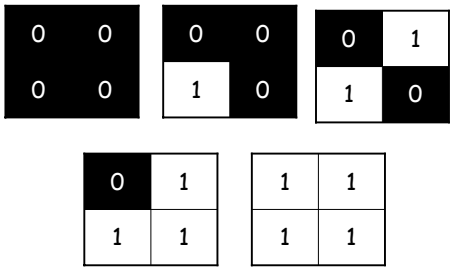
ordered dither intuition: 2 level output

suppose all 2x2 neighborhoods were uniform

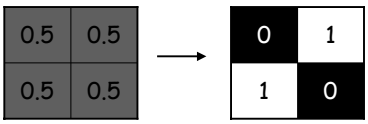
we could quantize neighborhoods together

.2	.2	.6	.6	.3	.3
.2	.2	.6	.6	.3	.3
.4	.4	.9	.9	.2	.2
.4	.4	.9	.9	.2	.2

neighborhood blocks

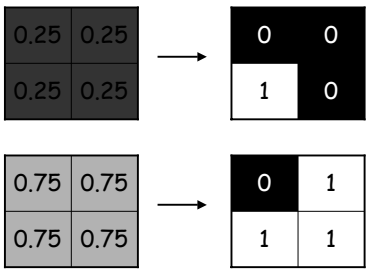


intuition

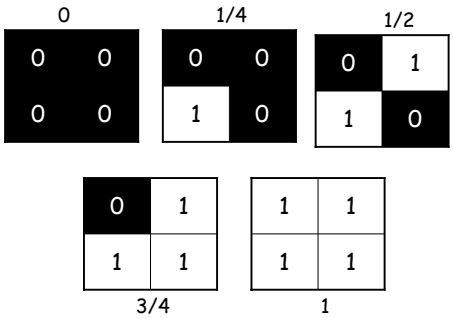


try to preserve average intensity over neighborhoods

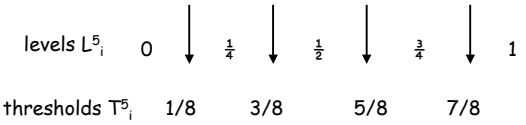
more intuition



average intensity simulates 5 output levels



midpoint thresholds for 5 output levels



quantization

.2	.2	.6	.6	.3	.3
.2	.2	.6	.6	.3	.3
.4	.4	.9	.9	.2	.2
.4	.4	.9	.9	.2	.2

→

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

but we don't have uniform neighborhoods...

0.32	0.57
0.97	0.21

→ ?

or do we?

image coherence

ordered dither intuition

D	B
A	C

choose thresholds for each neighborhood position that preserve, as well as possible, average intensity over uniform neighborhoods

ordered dither: step 1

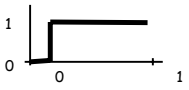
determine pixel location in neighborhood

4	**2**	4	1	4	2	4
1	**3**	1	3	1	3	1
4	2	4	2	4	2	4
1	3	1	3	1	3	1
4	2	4	2	4	2	4

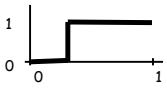
ordered dither: step 2

quantize based on threshold for pixel location

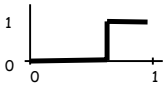
Location 1



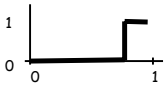
Location 2:



Location 3



Location 4



5 simulated output levels

0

0	0
0	0

1/4

0	0
1	0

1/2

0	1
1	0

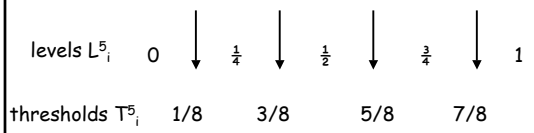
0	1
1	1

3/4

1	1
1	1

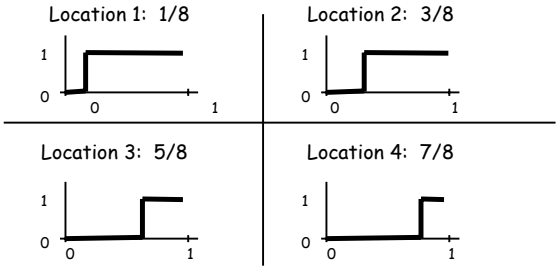
1

midpoint thresholds for 5 output levels



thresholds

threshold for location i: $T^5_i = (2i-1)/8$, i = 1, …, 4



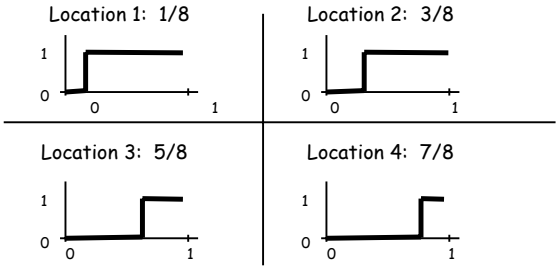
ordered dither recap: step 1

determine pixel location in neighborhood

4	**2**	4	2	4	2	4
1	**3**	1	3	1	3	1
4	2	4	2	4	2	4
1	3	1	3	1	3	1
4	2	4	2	4	2	4

ordered dither recap: step 2

quantize based on threshold T^5_i for pixel location i



ordered dither

- **another view**
  - easier to implement in ip framework
- **generalizations**
  - simulate more output levels
  - use more bits/pixel/channel

ordered dither

1. determine pixel location in neighborhood

4	**2**	4
1	**3**	1
4	2	4

2. quantize using threshold T^5_i for pixel in location i

2. add $\frac{1}{2} - T^5_i$ "noise" to pixel value in location i then use uniform 2-level quantization

step 2: comparison

Approach 1:

quantize based on threshold T^5_i for pixel location i

Approach 2:

a) add $\frac{1}{2} - T^5_i$ to pixel value for location i

b) quantize using $\frac{1}{2}$ threshold

the two approaches yield identical results for any pixel value v and location i since

$v < T^5_i$ iff $v + \frac{1}{2} - T^5_i < \frac{1}{2}$

step 2: comparison

this may be easier to implement in project 1 since the image class will perform step (b) automatically

Approach 2:

a) add $\frac{1}{2} - T^5_i$ to pixel value for location i

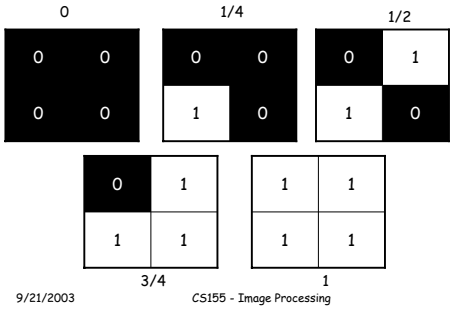
b) quantize using $\frac{1}{2}$ threshold

ordered dither

- **another view**
  - easier to implement in ip framework
- **generalizations**
  - simulate more output levels
  - use more bits/pixel/channel

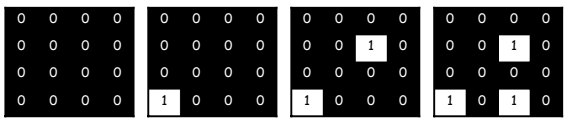
with window size kxk we can simulate k^2+1 levels

k=2, $k^2+1=5$



with window size kxk we can simulate k^2+1 levels

k=4, $k^2+1=17$



0 1 2 3

etc.

Bayer's ordered 4x4: pixel locations

this can be derived recursively from 2x2

16	8	14	6
4	12	2	10
13	5	15	7
1	9	3	11

Bayer's ordered 4x4: pixel location

16	8	14	6
4	12	2	10
13	5	15	7
1	9	3	11

Bayer's ordered 2x2: pixel locations

4	2
1	3

Bayer's ordered 4x4: pixel location

16	8	14	6
4	12	2	10
13	5	15	7
1	9	3	11

etc.

other sizes

- k=2,4, ..., 2^i use Bayer's
- for others make up your own but be careful of creating artifacts

ordered dither

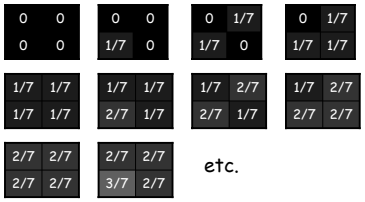
- **another view**
  - easier to implement in ip framework
- **generalizations**
  - simulate more output levels
  - use more bits/pixel/channel

Ordered Dither: n bits, kxk neighborhood

- we can use any one of $N=2^n$ output levels for each pixel
- we can simulate a total of $(N-1)*k^2+1$ output levels

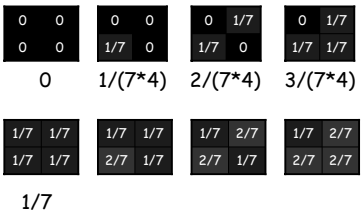
Example n=3, k=2

- 2^n=8 output levels:
  0=0/7, 1/7, 2/7, … , 6/7, 7/7=1
- simulated levels

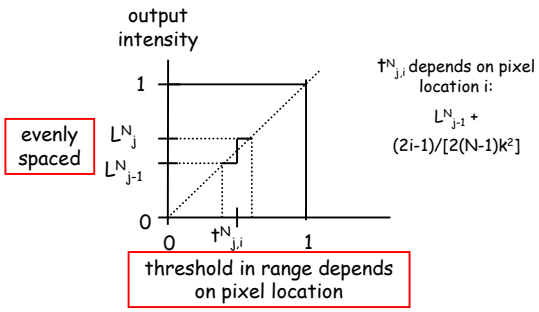


Example n=3, k=2

simulated levels: n^2-1=7, k^2=4



Ordered Dither: n bits, N=2^n output levels



Alternate View Ordered Dither: n bits N=2^n levels