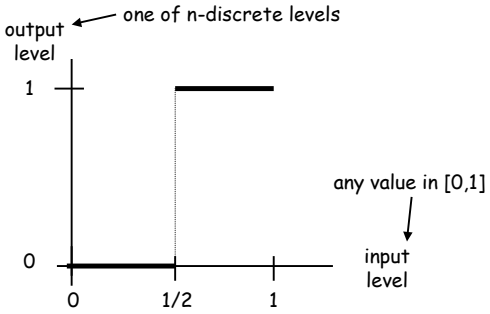
type of techniques

- simple pixel modification
- interpolation/extrapolation
- compositing
- convolution
- **dithering**
- warping
- morphing
- misc. effects

digital image

- sample at points on grid
- quantize color at sample points

n-level quantization (per channel)

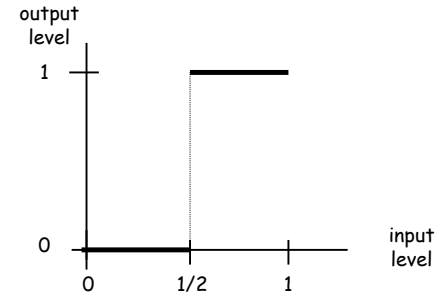


uniform quantization

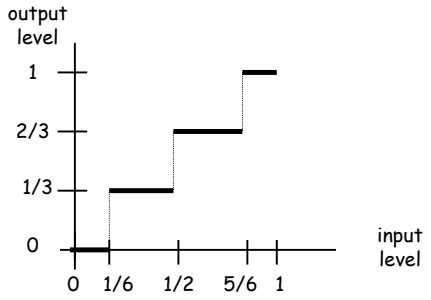
Guiding principles:

- 0 and 1 are valid output levels
- For any input level x, the output $y=Q_n(x)$ should be chosen so as to minimize $|y-x|$.

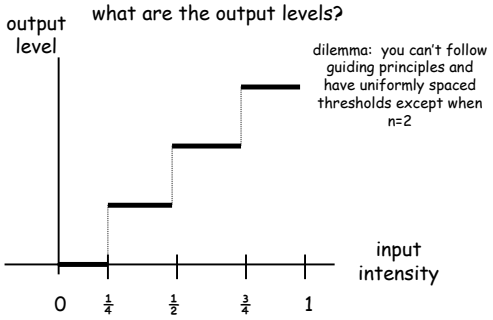
2-level uniform quantization



uniform quantization: 4 level



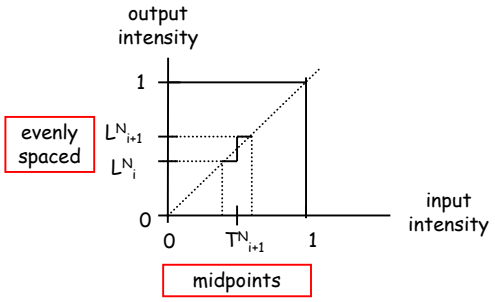
4-level uniformly spaced thresholds



n-level uniform quantization

- output levels:
  evenly spaced
- thresholds:
  midpoints

n- level uniform quantization



n-level uniform quantization

- output levels:
  $L^n_i = i/(n-1)$, $i = 0, \ldots, n-1$
- thresholds:
  $T^n_i = (L^n_{i-1} + L^n_i)/2 = (2i-1)/2(n-1)$, $i = 1, \ldots, n-1$
- quantization function:
  $Q_n:[0,1] \rightarrow \{0, 1/(n-1), 2/(n-1), \ldots, 1\}$
  $Q_n(v) = \lfloor v(n-1) + 0.5 \rfloor/(n-1)$

quantization error

8 bits per pixel per channel

1 bits per pixel per channel

quantization error

8 bits per pixel per channel

1 bit per pixel per channel

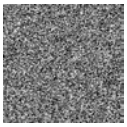
dithering

add "noise" then quantize

quantization error

8 bits per pixel per channel

1 bit per pixel per channel



1 bit per pixel per channel NOISIFIED

random dither

8 bits per pixel per channel

1 bits per pixel per channel

1 bits per pixel per channel noisy

random dither

for each pixel in the input image

- add random noise in $[-\varepsilon, \varepsilon]$ to pixel value
- clamp value to [0,1]
- uniformly quantize new value

dithering

- random
- ordered
- error diffusion

comparison

original 8 bits/pixel/ channel

random

ordered

error-diffusion

1 bit/pixel/channel

ordered dither

intuition

(We are going to assume we want to quantize to one bit per pixel per channel.)

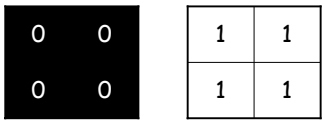
ordered dither intuition: 1 bit quantization

let's assume for the momment that 2x2 neighborhoods ARE uniform

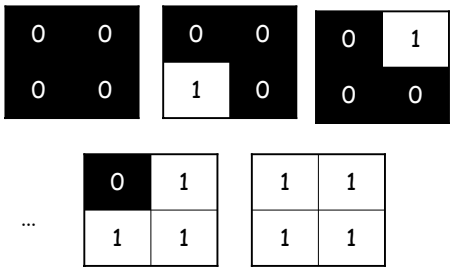
.2	.2	.6	.6	.3	.3
.2	.2	.6	.6	.3	.3
.4	.4	.9	.9	.2	.2
.4	.4	.9	.9	.2	.2

1 bit quantization

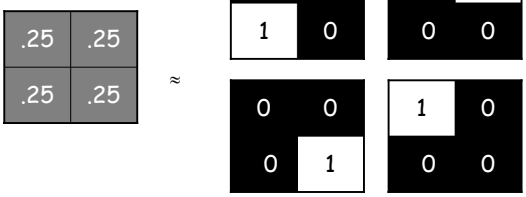
every neighborhood is quantized in one of two ways



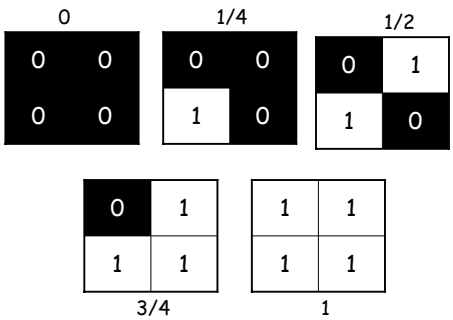
There are 16 possible ways to quantize a 2x2 block!



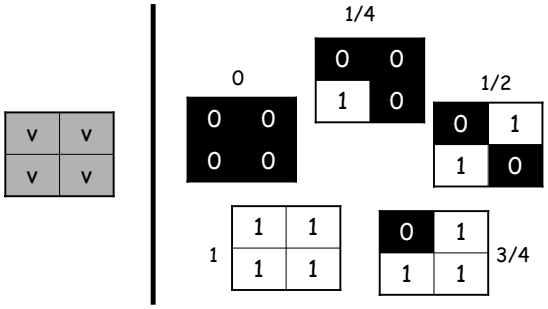
The ones we don't use could be useful!



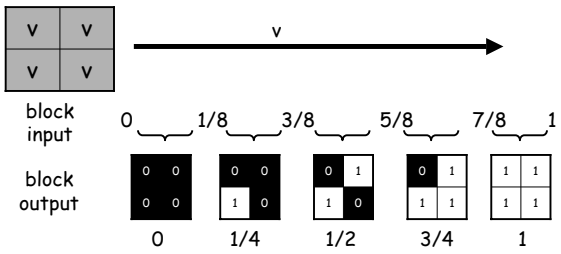
we can represent 5 different average intensities!



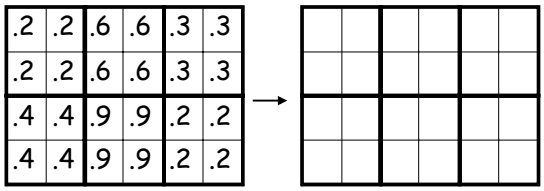
how should we quantize this 2x2 block?



quantization rule

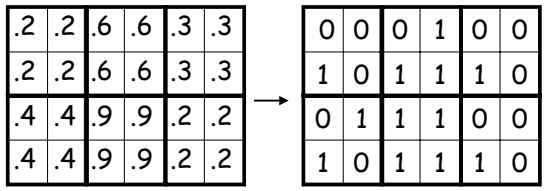


exercise



hint: 1/8=.125, 3/8=.375, 5/8=.625, 7/8=.875

quantization

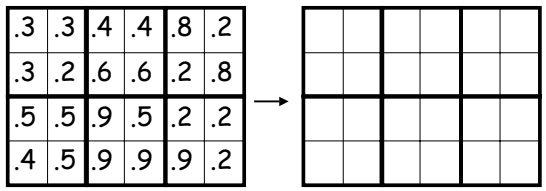


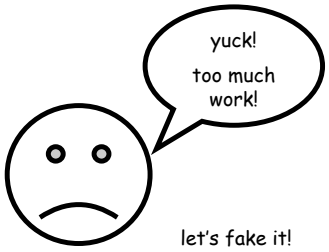
hint: 1/8=.125, 3/8=.375, 5/8=.625, 7/8=.875

non-uniform neighborhoods

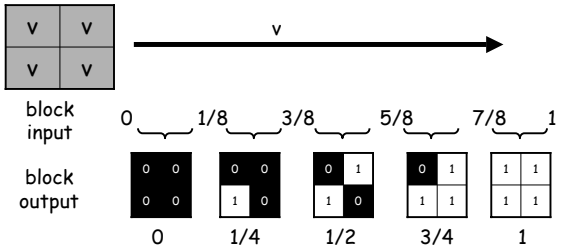
- average neighborhoods
- then use previous algorithm

exercise



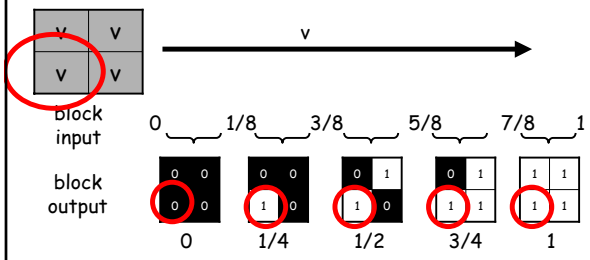


quantization rule revisited



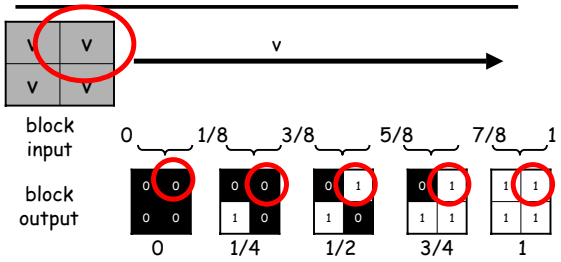
quantization rule

another way of looking at it

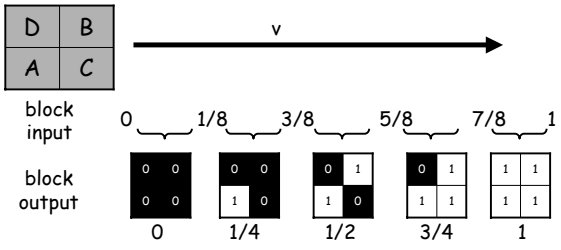


quantization rule

another way of looking at it

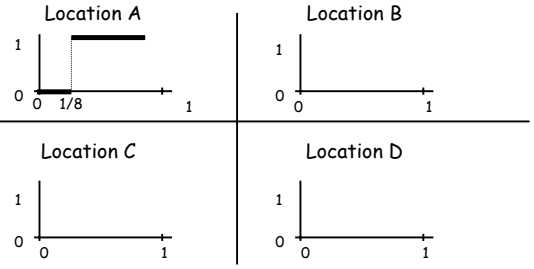


Exercise

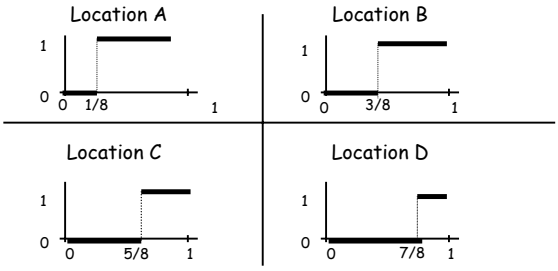


Exercise: Draw the quanitization functions for each of the pixel positions.

Exercise



quantization functions



we can use this algorithm even if we don't have uniform neighborhoods!

image coherence

adjacent pixels are usually similar

0.32	0.57
0.97	0.21

edge or noise

0.56	0.52
0.6	0.5

typical

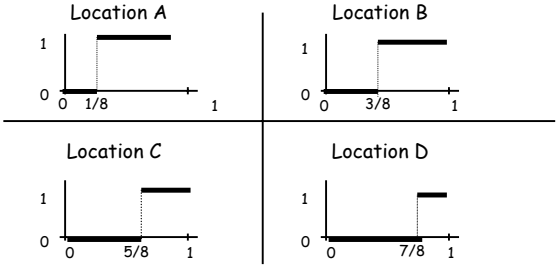
Ordered dither

1. find pixel location in block

D	B	D	B	D	B
A	C	A	C	A	C
D	B	D	B	D	B
A	C	A	C	A	C

Ordered dither

2. Quantize using threshold for position



exercise

.3	.3	.4	.4	.8	.2
.3	.2	.6	.6	.2	.8
.5	.5	.9	.5	.2	.2
.4	.5	.9	.9	.9	.2

→

exercise

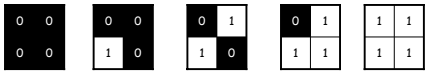
.3	.3	.4	.4	.8	.2
.3	.2	.6	.6	.2	.8
.5	.5	.9	.5	.2	.2
.1	.5	.9	.9	.9	.2

→

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

recap

Ordered dither simulates five output levels using only 2 levels.



Generalizations

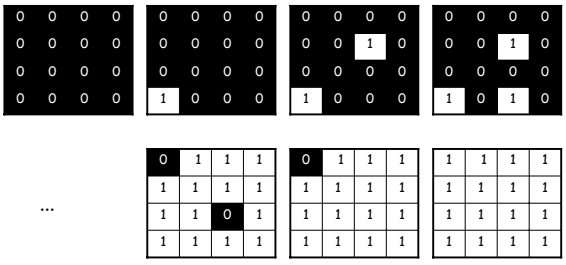
- Can we use larger blocks; e.g. 4x4?
- What if we have more than 1 bit; e.g. suppose we have 2 bits and four output levels?

Exercise

- How many levels can we simulate with using 4x4 blocks?
- What are the simulated levels?
- What are the corresponding thresholds?

kxk Ordered dither simulates k^2+1 levels

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



KxK ordered dither: simulated levels

Output levels: i/k^2 for i=0, 1, ..., k^2

Threshold between levels i/k^2 and $(i+1)/k^2$ is $(2i+1)/(2k^2)$.

Bayer's ordered 4x4: pixel locations

this can be derived recursively from 2x2

D	B
A	C

D3	B3	D1	B1
A3	C3	A1	C1
D0	B0	D2	B2
A0	C0	A2	C2

Bayer's ordered 4x4: pixel location

15	7	13	5
3	11	1	9
12	4	14	6
0	8	2	10

k^2+1 levels (where k is a power of 2)

- compute bayer kxk matrix
- compute thresholds for k^2+1 levels
- use threshold based on pixel location

For other values of k we need a good pattern. Bad patterns can create weird artifacts.

Alt view: ordered dither

2. Quantize using threshold for position

Location A | Location B

Location C | Location D

2. Add offset value based on position then quantize as usual.

A: 1/8-1/2	

$v+(1/8-1/2) < 1/2$ iff $v < 1/8$

Alt view: ordered dither

2. Quantize using threshold for position

Location A | Location B

Location C | Location D

2. Add offset value based on position then quantize as usual.

D: 7/8-1/2	B: 3/8- 1/2
A: 1/8-1/2	C: 5/8-1/2

A: $v < 1/8$ iff $v+(1/8-1/2) < 1/2$

Alternate view

KxK Algorithm: Add $[2j-k^2-1]/[2k^2]$ noise to pixel in location j then use uniform quantization rule.

Generalizations

- Can we use larger blocks; e.g. 4x4?
- What if we have more than 1 bit; e.g. suppose we have 2 bits and four output levels?

Ordered Dither: n levels, kxk neighborhood

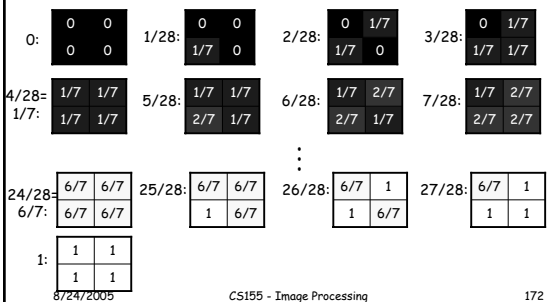
If we have n output levels and use kxk dithering neighborhoods we can simulate a total of $(n-1)*k^2+1$ output levels.

Example n=8, k=2: $(8-1)*2^2+1=29$ simulated levels

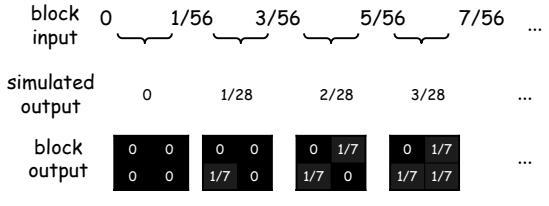
- 8 output levels: 0=0/7, 1/7, 2/7, … , 6/7, 7/7=1
- simulated levels: 0=0/28, 1/28, …, 27/28, 28/28=1

Example n=8, k=2: $(8-1)*2^2+1=29$ simulated levels

- We simulate the level by a corresponding 2x2 block:



Example n=8, k=2: Thresholds



For general n, k: Thresholds

The ith threshold ($i \in [1,n-1]$) for pixel location j ($j \in [1,k^2]$) is

$[2[j+k^2(i-1)]-1]/[2(n-1)k^2]$

dithering

- random
- ordered
- error diffusion

comparison

8 bits original | 1 bit ordered | 1 bit error-diffusion

error-diffusion dither intuition

quantize I_{00} using uniform quantization

I_{00}	I_{01}	I_{02}
I_{10}	I_{11}	I_{12}

Q_{00}		

error-diffusion dither intuition

this introduces some error ... suppose Q_{00} is too dark

I_{00}	I_{01}	I_{02}
I_{10}	I_{11}	I_{12}

Q_{00}		

error-diffusion dither intuition

we can compensate by brightening the neighbors of I_{00} before we quantize them.

I_{00}	I_{01}	I_{02}
I_{10}	I_{11}	I_{12}

Q_{00}		

error-diffusion dither intuition

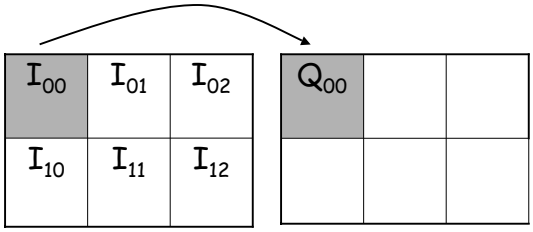
now continue quantization on modified image

I_{00}	I_{01}	I_{02}
I_{10}	I_{11}	I_{12}

Q_{00}		

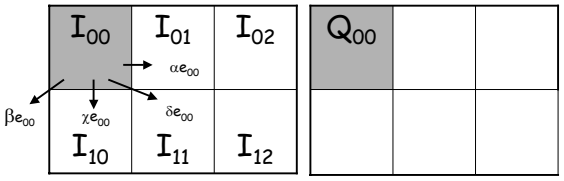
now for the details

quantize I_{00} using uniform quantization



error diffusion dither

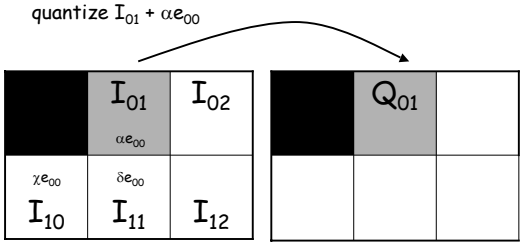
distribute error $e_{00}=I_{00} - Q_{00}$ to neighbors not yet quantized



$\alpha+\beta+\chi+\delta=1$

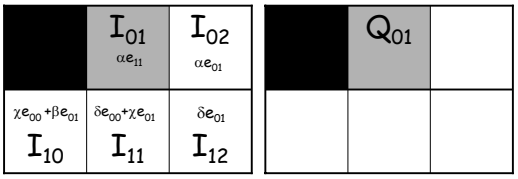
error diffusion dither

quantize $I_{01} + \alpha e_{00}$



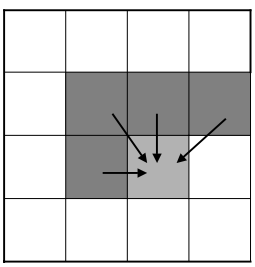
error diffusion dither

distribute error: $e_{01} = I_{01}+\alpha e_{00} - Q_{01}$



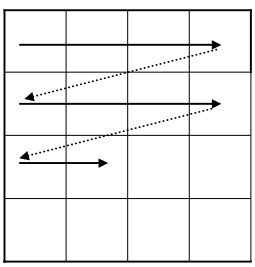
error diffusion dither

error contributions by upper & left neighbors



error diffusion dither

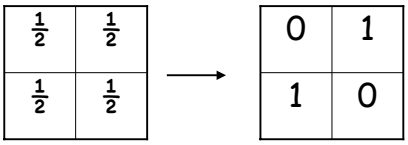
order of quantization is important



floyd-steinberg

$\alpha = 7/16$
$\beta = 3/16$
$\chi = 5/16$
$\delta = 1/16$

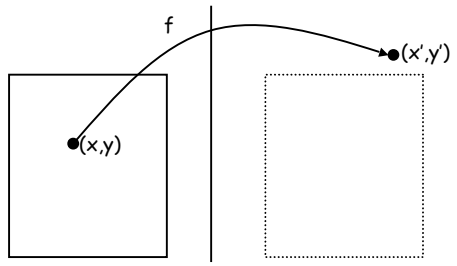
floyd-steinberg: example



types of techniques

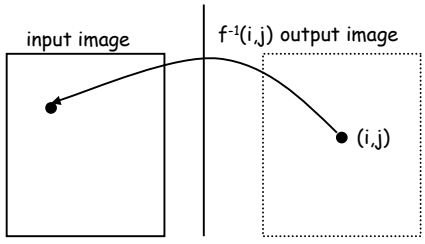
- simple pixel modification
- interpolation/extrapolation
- compositing
- convolution
- dithering
- **warping**
- morphing
- misc. effects

forward warp



f maps points in input image to the plane

forward warp



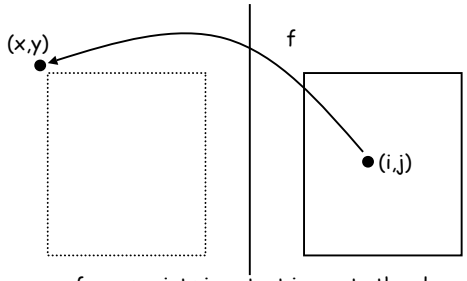
pixel at (i,j) in output image is assigned
the value at location $f^{-1}(i,j)$ in input image

forward warp: problems

if f is not bijective

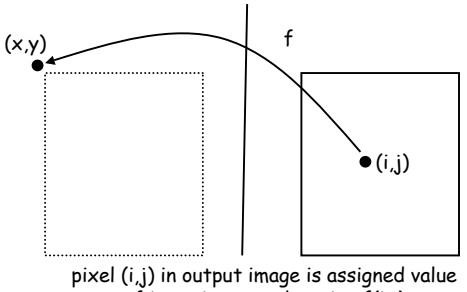
1. $f^{-1}(i,j)$ may not be defined
2. $f^{-1}(i,j)$ may not be unique

backward warp



f maps points in output image to the plane

backward warp



pixel (i,j) in output image is assigned value of input image at location f(i,j)

backward warp: problems

1. f(i,j) may lie outside the input image area
2. f(i,j) may not lie on a sample of the input image

backward warp: problems

1. f(i,j) may lie outside the input image area

   solution: give image an infinite, black (or other default) border
2. f(i,j) may not lie on a sample of the input image

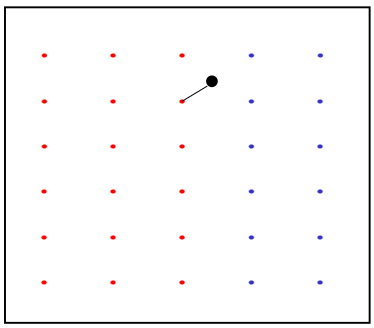
backward warp: problems

1. f(i,j) may lie outside the input image area

   solution: give image an infinite, black (or other default) border
2. f(i,j) may not lie on a sample of the input image

   solution: resample input

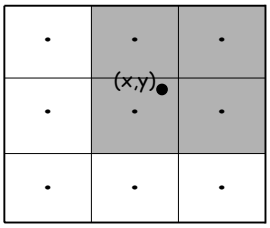
re-sample: estimate input image at arbitrary location



re-sample

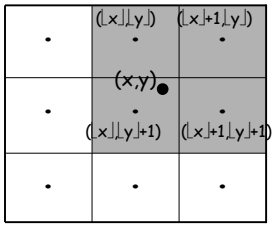
interpolate based on nearby samples
- nearest
- bilinear
- bicubic
- gaussian

which way is up?



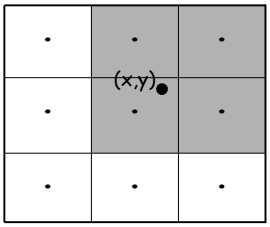
what are the coordinates of the pixels surrounding (x,y)?

which way is up?



what are the coordinates of the pixels surrounding (x,y)?

nearest

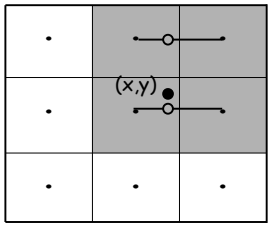


- compute distance between x,y and the locations of the neighboring samples
- set value at x,y to the value of the closest neighbor

re-sample

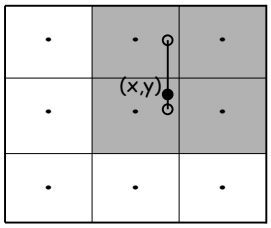
interpolate based on nearby samples
- nearest
- **bilinear**
- bicubic
- gaussian

bilinear interpolation



1. interpolate to find values at (x,⌊y⌋) and (x,⌊y⌋+1)

bilinear interpolation



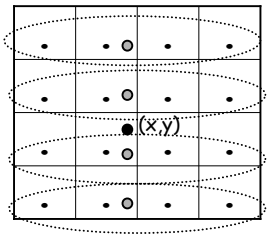
1. interpolate to find values at($x\lfloor y\rfloor$) and ($x\lfloor y\rfloor+1$)
2. interpolate to find value at x,y

re-sample

interpolate based on nearby samples

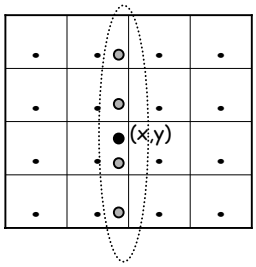
- nearest
- bilinear
- **bicubic**
- gaussian

bicubic



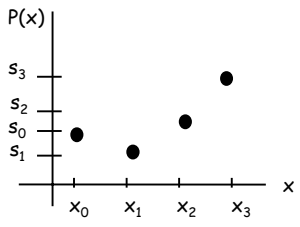
1. interpolate to find values at ($x\lfloor y\rfloor+i$)

bicubic



1. interpolate to find values at ($x\lfloor y\rfloor+i$)
2. interpolate to find value at (x,y)

bicubic: lagrangian



there is a unique cubic polynomial through any four distinct sample point

lagrange cubic polynomial

$$P(x) = \Sigma_{i=0,1,2,3}\, s_i \prod_{j=0,1,2,3,j\neq i} (x-x_j)/(x_i-x_j)$$

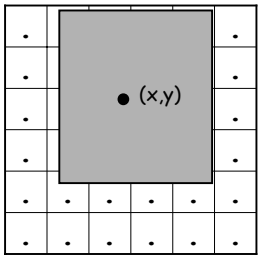
exercise: what is the value of $P(x_i)$ i=0,1,2,3

re-sample

interpolate based on nearby samples

- nearest
- bilinear
- bicubic
- **gaussian**

gaussian



interpolate nearby samples using normalized gaussian weights

unnormalized weight at (i,j) in window is

$\exp[-((x-i)^2+(y-j)^2)/\sigma^{2]}$

types of techniques

- simple pixel modification
- interpolation/extrapolation
- compositing
- convolution
- dithering
- warping
- **morphing**
- misc. effects