Recap

- What is an Image?
- What is a pixel?
- What is a Convolution?
- How do we store a continuous function in a computer?
  - Sampling
- What is aliasing?
  - Multiple continuous signals sharing the same sampling result
- What did Nyquist/Shannon do?
  - Theorem of sampling rate needed to represent a signal at a given frequency
- How does aliasing present itself in 2D graphics?

1

Review

- Properties of filters represented by convolution?
  - Linearly Separable
  - Shift Invariant
- Difference between moving average and convolution?
  - Convolution allows arbitrary normalization weights

2

Image Manipulation

Stephen J. Guy

Sep 18, 2014

Some Material Adopted from Peter Shirley and Steve Marshner & Greg Humphreys

Outline

- Warping
  - Scale, Rotate, Warps
- Sampling & Reconstruction
  - Resampling
  - Reconstruction Filters
- Image Manipulation
  - Filtering
  - Pixel Operation
- Quantization

4

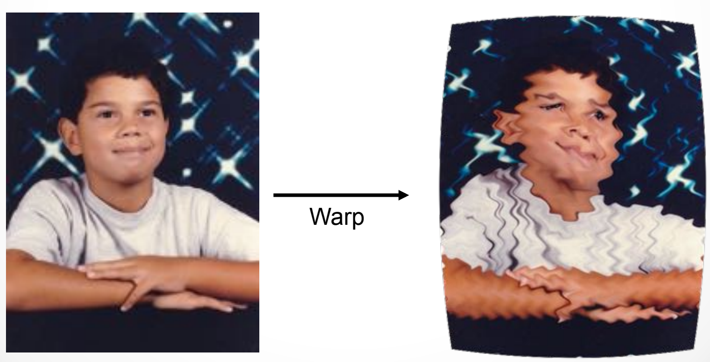
Outline

- **Warping**
  - **Scale, Rotate, Warps**
- Sampling & Reconstruction
  - Resampling
  - Reconstruction Filters
- Image Manipulation
  - Filtering
  - Pixel Operation
- Quantization

5

Warping

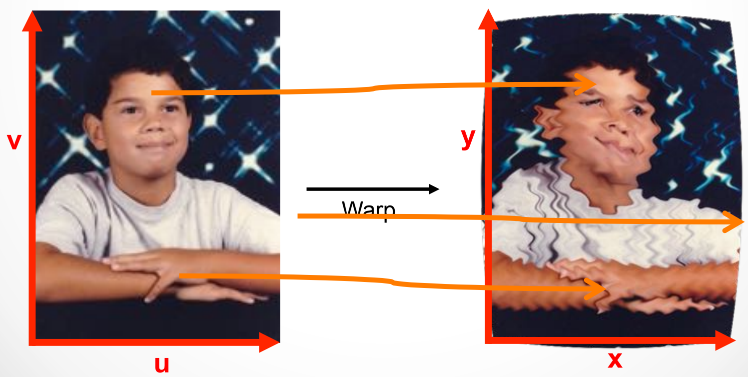
- Moving Pixels of Images
  - Mapping
  - Resampling



6

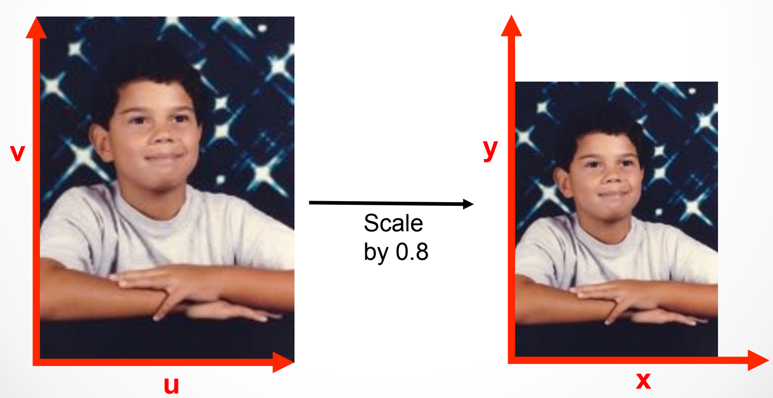
Mapping

- Defines transformation between two images
  - Describe a destination (x,y) for every location (u,v) in the source
  - Vice-versa if it's invertable



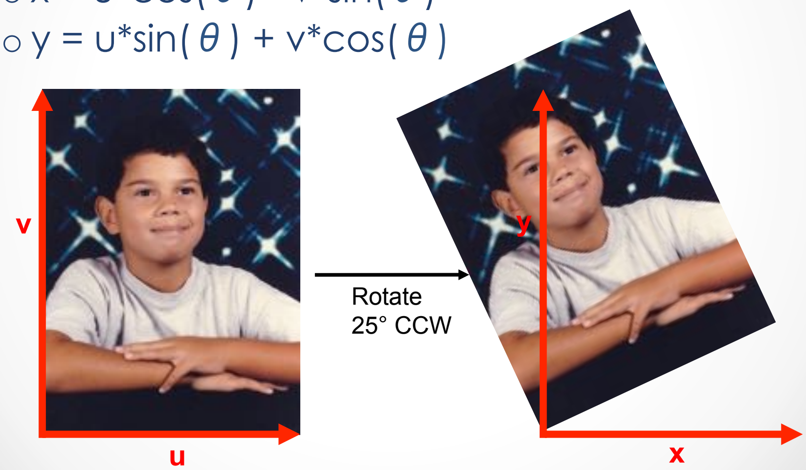
Example Mapping - Scale

- Scale by factor:
  - x = factor * u
  - y = factor * v



Example Mapping - Scale

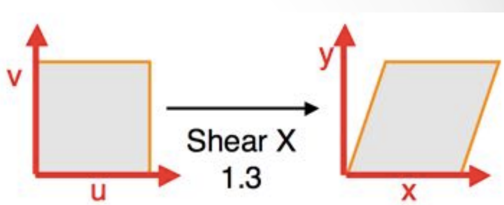
- Rotate by θ degrees
  - o $x = u*\cos(\theta) - v*\sin(\theta)$
  - o $y = u*\sin(\theta) + v*\cos(\theta)$



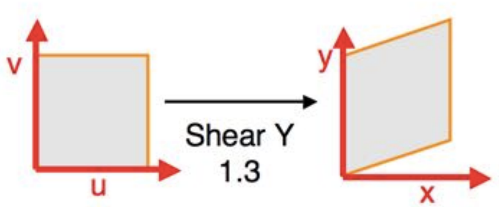
9

Example Mapping - Shear

- Shear in X by *factor*
  - o x = u + factor * v
  - o y = v



- Shear in Y by *factor*
  - o x = u
  - o y = v + factor*u



10

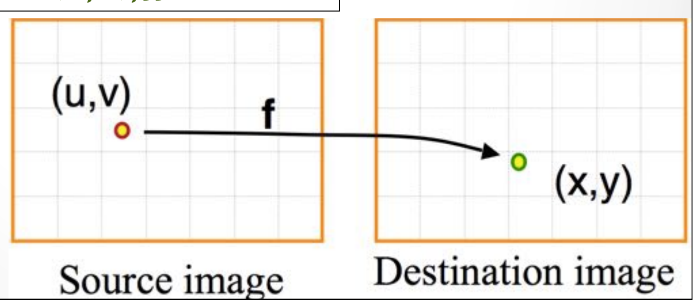
Other Mappings

- Any function of u and v:
  - $x = f_x(u,v)$
  - $y = f_y(u,v)$

Implementing Warping - 1

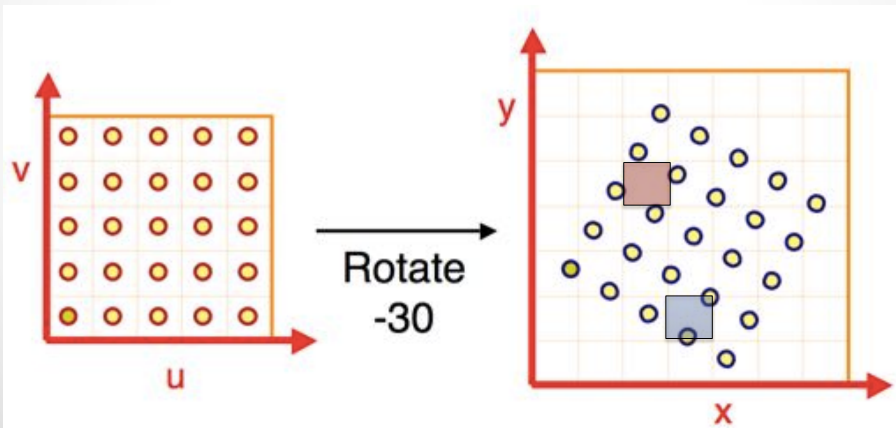
- Forward Mapping

```
for (int u = 0; u < umax; u++) {
 for (int v = 0; v < vmax; v++) {
  float x = fx(u,v);
  float y = fy(u,v);
  dest(x,y) = src(u,v);}}
```



Forward Mapping – Bad Idea!!

- Iterating over source image creates issues
  - Double coverage & Empty pixels

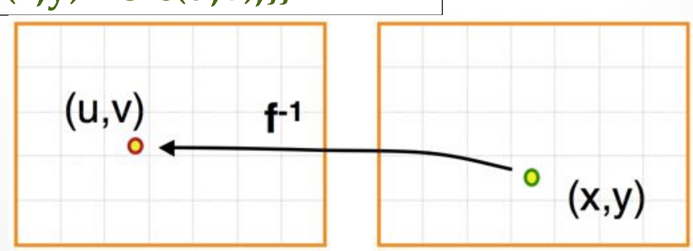


13

Better Warping

- Reverse Mapping

```
for (int x = 0; x < xmax; x++) {
  for (int y = 0; y < ymax; y++) {
    float u = f_x^-1(x,y);
    float v = f_y^-1(x,y);
    dest(x,y) = src(u,v);}}
```

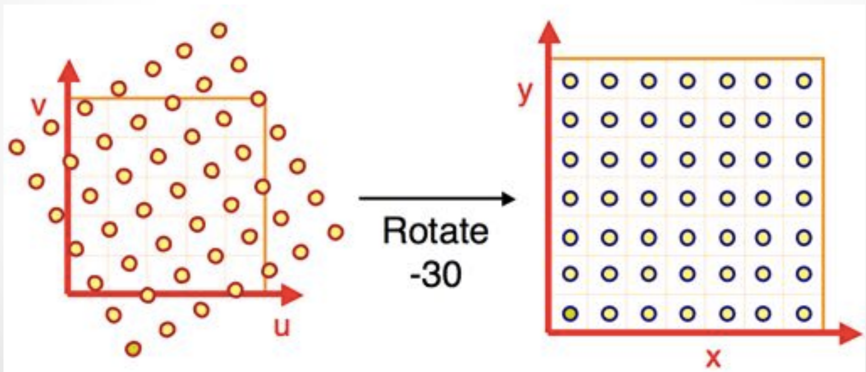


Source image   Destination image

14

Reverse Mapping

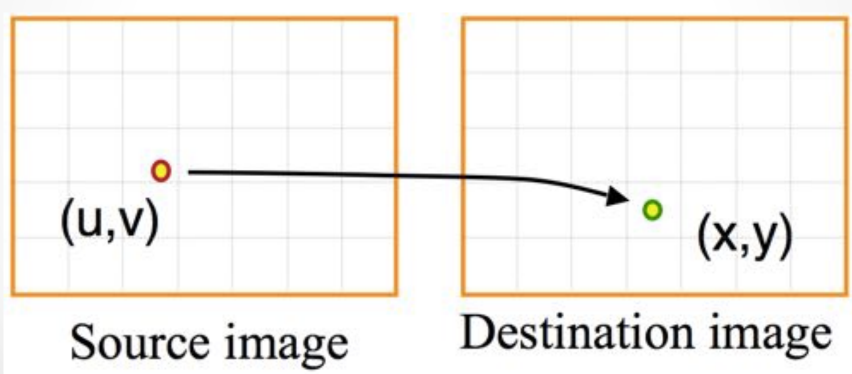
- Iterate over destination image
- May oversample, but much better!



15

Reverse Mapping (cont.)

- Evaluate arbitrary, non-integer (u,v) coords
- Resampling!!!



16

Outline

- Warping
  - Scale, Rotate, Warps
- **Sampling & Reconstruction**
  - **Resampling**
  - **Reconstruction Filters**
- Image Manipulation
  - Filtering
  - Pixel Operation
- Quantization

17

Signal Resampling

- Changing the sampling rate
  - Images: Enlarging or shrinking!
- Creating more samples:
  - Increasing sampling rate
  - “Upsampling”
  - “Enlarging”
- Ending up with fewer samples:
  - Decrease the sampling rate
  - “Downsamplign”
  - “Reducing”

18

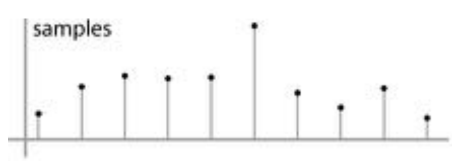
Reducing and Enlarging

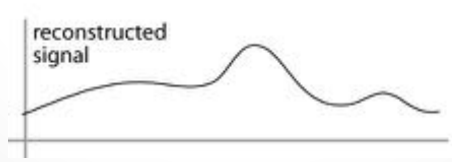
- Very common operation
  - Different device resolution (mobile vs HDTV)
  - Different memory availability
- Very commonly done poorly
- Simple approach: Drop/replicate pixels
- Correct approach: Use resampling
  - Step 1: Reconstruct image
  - Step 2: Sample reconstructed image

19

Signal Reconstruction

- Continuous-discrete convolution





20

Reconstruction Pseudocode

- Reconstruction is just continuous discrete filtering

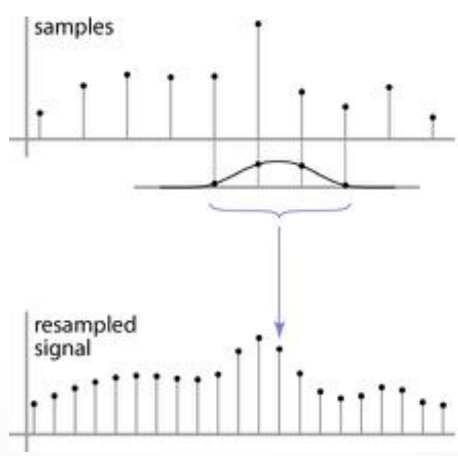
function reconstruct(sequence a, filter f, real x)
$s = 0$
$r = f$.radius
for $i = \lceil x - r \rceil$ to $\lfloor x + r \rfloor$ **do**
  $s = s + a[i]f(x - i)$
return s

- Resamples the signal at point x
- Different filters give different results

21

Resampling

- Reconstruction creates a continuous function
  - o Sample *this new* function



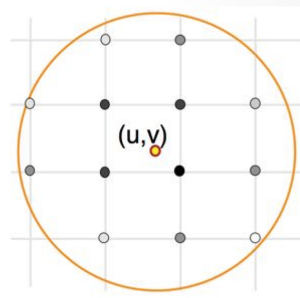
22

Cont.-Disc. Convolution in 2D

- Same convolution, but with two variables

$$(a \star f)(x, y) = \sum_{i,j} a[i, j] f(x - i, y - j)$$

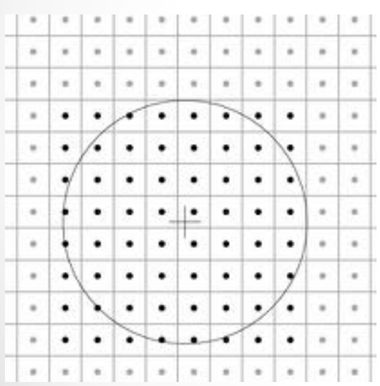
- Loop over nearby pixels
  - o Compute weights with f(x,y); Take weighted avg.
- Looks like discrete filter, but:
  - o Offsets are not integers
  - o Filter is continuous
- Remember: Filter will shift relative to the grid
  - o Can evaluate at any point

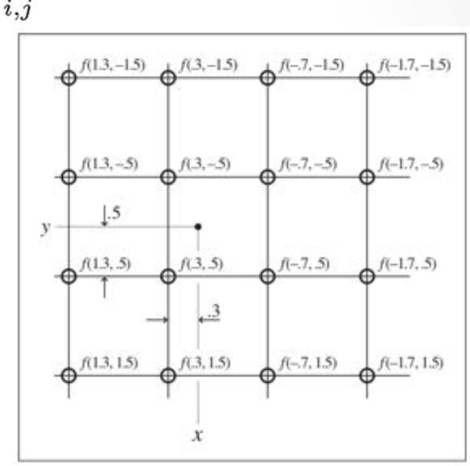


23

Cont.-Disc. Convolution in 2D

$$(a \star f)(x, y) = \sum_{i,j} a[i, j] f(x - i, y - j)$$





24

Reconstruction Filters

(A Gallery)

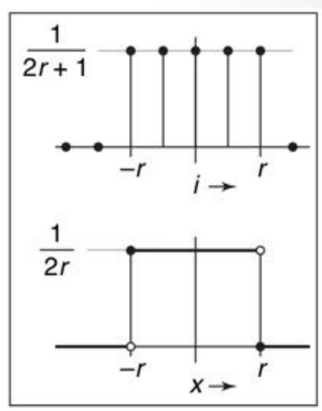
- Box filter
  - Simple & cheap
- Tent filter
  - Linear interpolation
- Gaussian filter
  - Very smooth antialiasing filter
- B-spline cubic
  - Very smooth
- Catmull-rom cubic
  - Interpolating
- Mitchell-Netravali cubic
  - Good for upsampling

Box Filter

- Simple average
  - Uniform weights

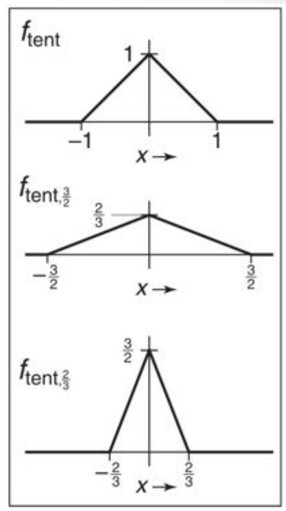
$$a_{\text{box},r}[i] = \begin{cases} 1/(2r+1) & |i| \le r, \\ 0 & \text{otherwise.} \end{cases}$$

$$f_{\text{box},r}(x) = \begin{cases} 1/(2r) & -r \le x < r, \\ 0 & \text{otherwise.} \end{cases}$$



Tent Filter

- Linear interpolation
  - Distance dependent weights



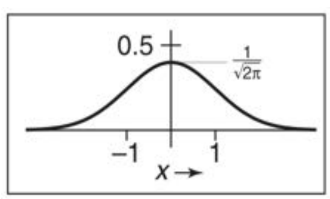
$$f_{\text{tent}}(x) = \begin{cases} 1 - |x| & |x| < 1, \\ 0 & \text{otherwise;} \end{cases}$$

$$f_{\text{tent},r}(x) = \frac{f_{\text{tent}}(x/r)}{r}.$$

27

Gaussian Filter

- Smoothed interpolation
  - Distance dependent weights

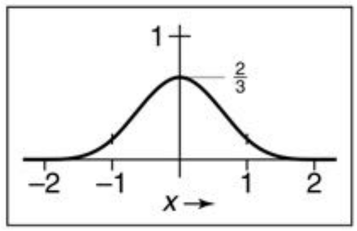


$$f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

28

B-Spline Cubic

- Smoothed interpolation
  - Distance dependent weights
  - Simpler to compute the Gaussian

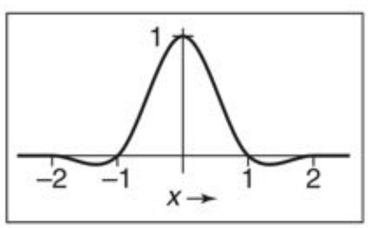


$$f_B(x) = \frac{1}{6}\begin{cases} -3(1-|x|)^3 + 3(1-|x|)^2 + 3(1-|x|) + 1 & -1 \le x \le 1, \\ (2-|x|)^3 & 1 \le |x| \le 2, \\ 0 & \text{otherwise.} \end{cases}$$

29

Catmull-Rom

- Has negative values!
  - Provides some sharpening to counteract blur

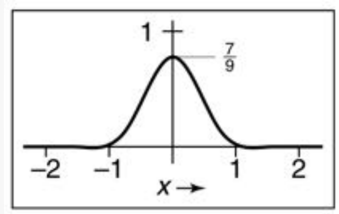


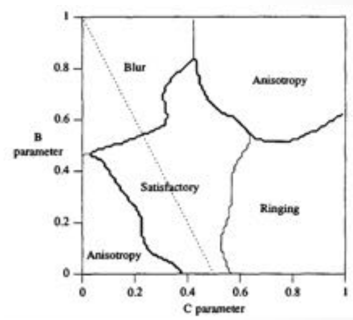
$$f_C(x) = \frac{1}{2}\begin{cases} -3(1-|x|)^3 + 4(1-|x|)^2 + (1-|x|) & -1 \le x \le 1, \\ (2-|x|)^3 - (2-|x|)^2 & 1 \le |x| \le 2, \\ 0 & \text{otherwise.} \end{cases}$$

30

Michell-Netravali Cubic

- Created a parameterized cubic curve
  - Held a user study to find best parameters





$$f_M(x) = \frac{1}{3} f_B(x) + \frac{2}{3} f_C(x)$$

$$= \frac{1}{18} \begin{cases} -21(1-|x|)^3 + 27(1-|x|)^2 + 9(1-|x|) + 1 & -1 \leq x \leq 1, \\ 7(2-|x|)^3 - 6(2-|x|)^2 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

Hacks as Filters

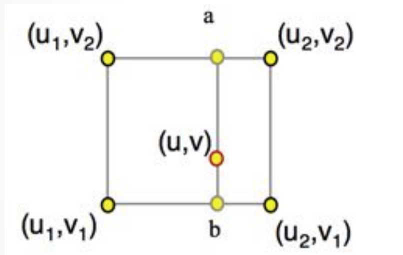
- Some simple approaches are same as filters

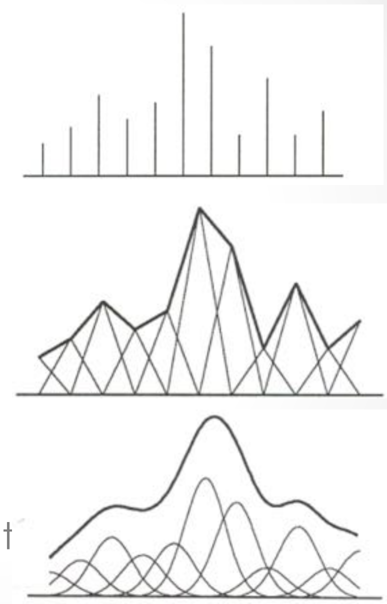
- Box filter (radius 0.5): Nearest Neighbors
  - Will only catch the one closest input
  - output[i, j] = input[round(x(i)), round(y(j))]

- Tent filter (radius 1): Linear Interpolation
  - Will catch exactly 2 points
  - Weights are a and (1-a)
  - Results in straight line interpolation of points

Common in Practice

- Box – Cause Jaggies
- Bilinear filtering (Tent)





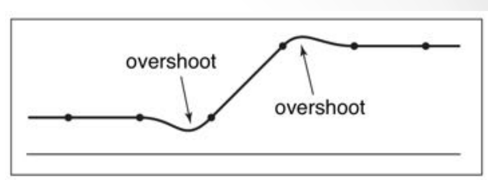
- Guassian filtering
  - o Width of Gaussian affects blurring amount

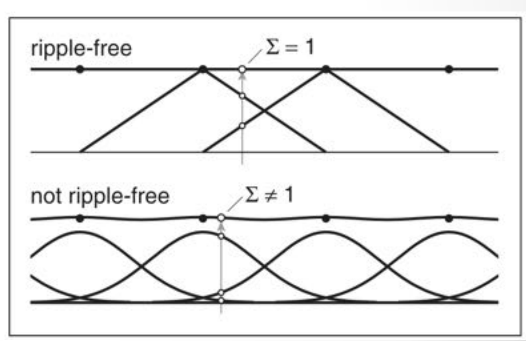
33

Ringing, Overshoot, & Ripples

- Overshoot
  - o Caused by negative filter values
- Ringing
  - o Reconstructing a straight line may not be constant

$$\sum_i f(x+i) = 1 \quad \text{for all } x.$$

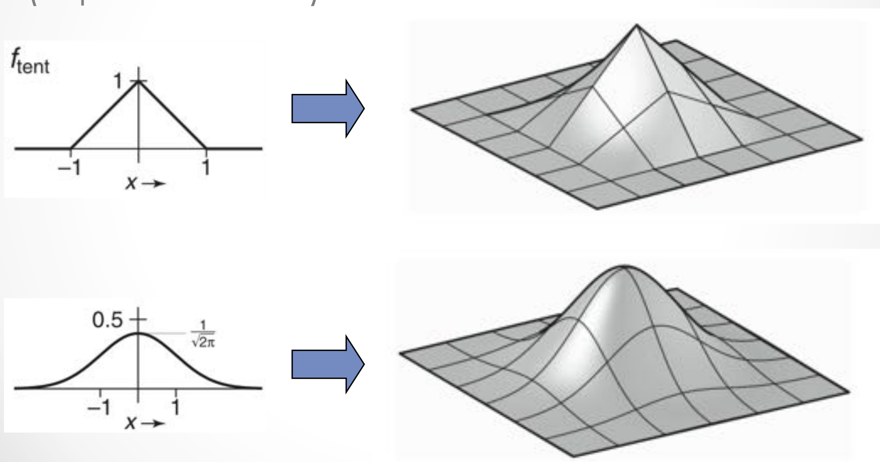




34

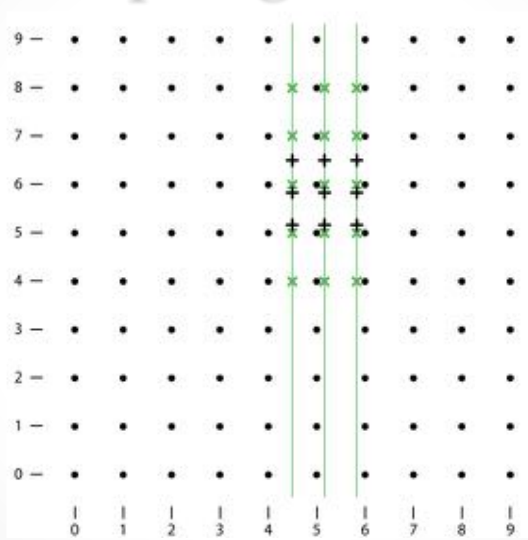
Making 2D Filters

- Compute dimensions independently (separable filter)



35

Upsampling Illustrated



36

Implementing Scale

- Scale (src, dst, sx, sy)

```
float r ≅ max(1.0/sx,1.0/sy);
for (int x = 0; x < xmax; x++) {
   for (int y = 0; y < ymax; y++) {
     float u = x / sx;
     float v = y / sy;
     dest(x,y) = resample_src(u,v,r);
   }
}
```

37

Implementing Rotate

- Rotate(src, dst, theta)

```
for (int x = 0; x < xmax; x++) {
   for (int y = 0; y < ymax; y++) {
     x = u*cos(-θ) - v*sin(-θ)
     y = u*sin(-θ) + v*cos(-θ)
     dest(x,y) = resample_src(u,v,r);
   }
}
```

38

Resampling Example

1000 pixel width

[Philip Greenspun]

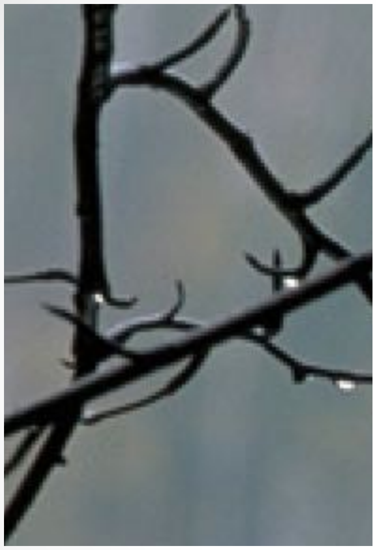
by dropping pixels

gaussian filter

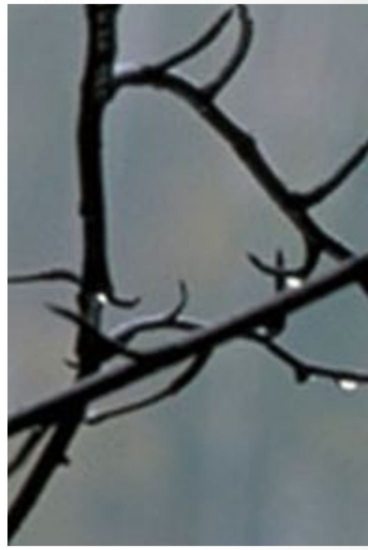
250 pixel width

[Philip Greenspun]

40



box reconstruction filter



bicubic reconstruction filter

[Philip Greenspun]

4000 pixel width

41

Separable Resampling (2 pass)

Must have intermediate storage
for 1st resampling pass

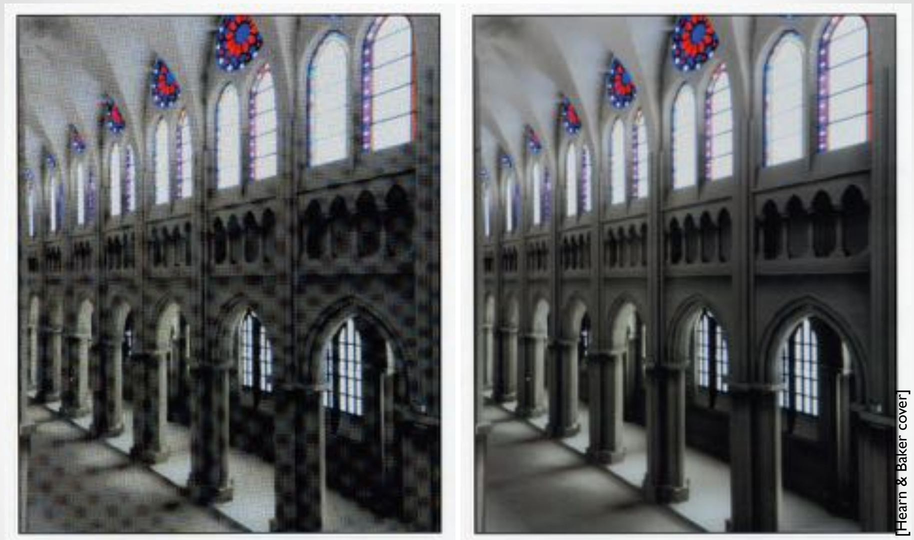
42

Artifacts

- Jaggies, Moiré patterns, etc.

[Hearn & Baker cover]

600ppi scan of a color halftone image



by dropping pixels

gaussian filter

downsampling a high resolution scan

45

Outline

- Warping
  - Scale, Rotate, Warps
- Sampling & Reconstruction
  - Resampling
  - Reconstruction Filters
- **Image Manipulation**
  - **Filtering**
  - Pixel Operation
- Quantization

46

Filter Examples

- Blur (and Sharpen)

Original

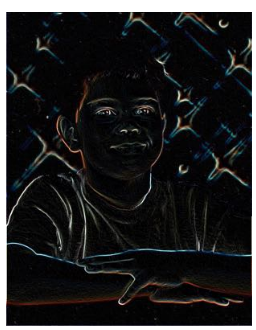
Blur

Sharpen

$$\text{Filter} = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Edge Detect

Original



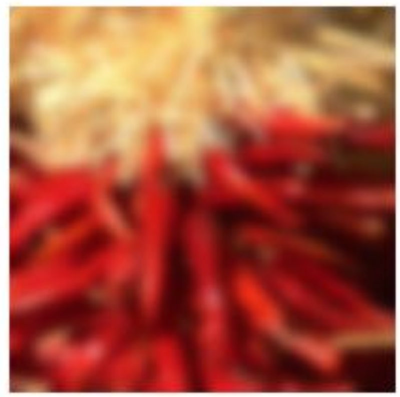
Detected Edges

$$\text{Filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

48

Ugly Details

- What about near the edge?
  - The filter window falls off the edge
  - Must extrapolate
  - Methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge



[Philip Greenspun]

49

Outline

- Warping
  - Scale, Rotate, Warps
- Sampling & Reconstruction
  - Resampling
  - Reconstruction Filters
- **Image Manipulation**
  - Filtering
  - **Pixel Operation**
- Quantization

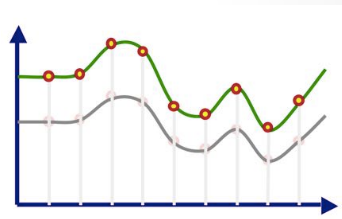
50

Brighten

- Simply scale pixel value
  - o Must clamp to range (e.g., 0 to 255)
- Trick: Interpolate/extrapolate form all black img.

Original

Brighten

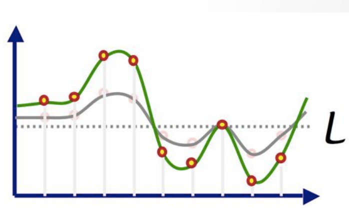


Contrast

- Compute mean luminance L over all pixels
  - o Luminance = 0.30*r + 0.59*g + 0.11*b
- Scale deviation from L for each pixel
- Trick: Interpolate/extrapolate from avg. grey img.

Original

More Contrast



Saturation

- Interpolate/extrapolate from grayscale version of image

Original | Less Saturated | More Saturated | Negatively Saturated

53

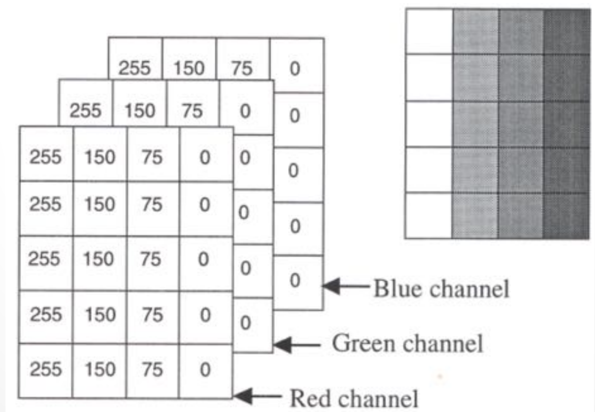
Outline

- Warping
  - Scale, Rotate, Warps
- Sampling & Reconstruction
  - Resampling
  - Reconstruction Filters
- Image Manipulation
  - Filtering
  - Pixel Operation
- **Quantization**

54

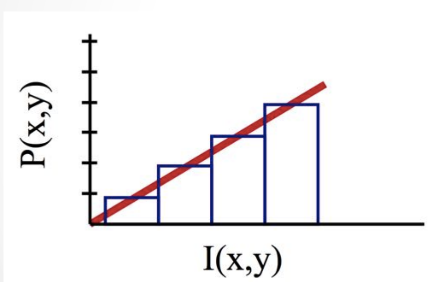
Quantization

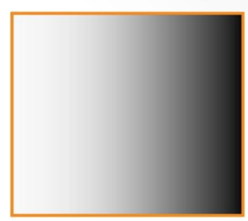
- Artifacts due to limited intensity resolution
  - Frame buffers have limited bits/pixel
  - Physical devices have limited dynamic range



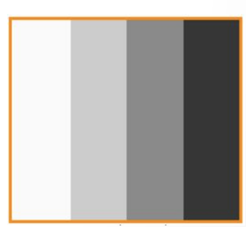
Uniform Quantization

- P(x,y) = trunc(I(x,y) + .5)





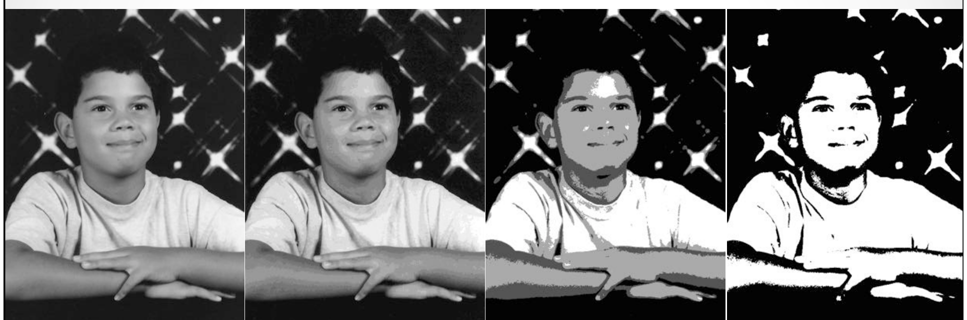
I(x,y)



P(x,y)
(2 bits per pixel)

Uniform Quantization

- Effect of decreasing bits/pixel



8 bits 4 bits 2 bits 1 bit

57

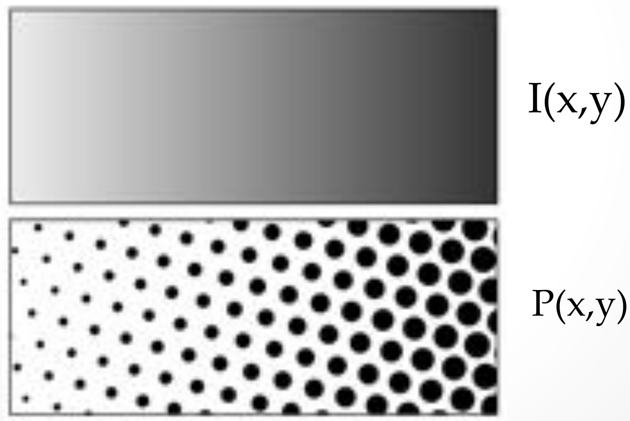
Reducing Quantization Effects

- Halftoning
    - o Classical Halftoning
- Dithering
    - o Random Dithering
    - o Ordered Dithering
    - o Error Diffusion Dithering

58

Classical Halftoning

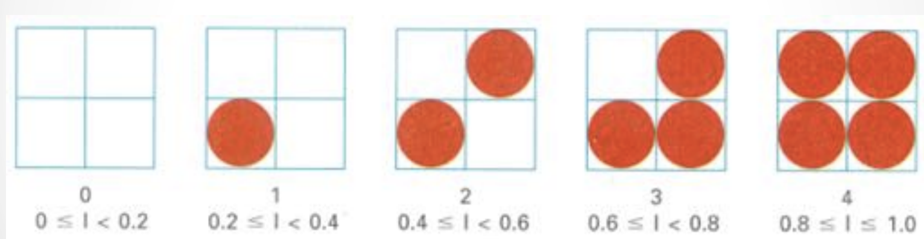
- Use dots of varying size to represent intensity
  - o Area of dot proportional to intensity



59

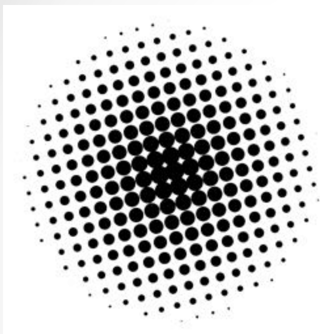
Halftone Patterns

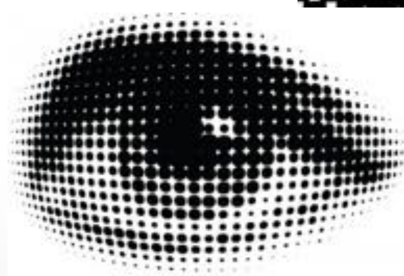
- Use clusters of pixels to represent intensity
  - o Trades spatial resolution for intensity resolution



60

Halftoning Examples





61

Dithering

- Distribute errors among pixels
  - Exploit eye's natural spatial integration
  - Display larger range of perceptible intensities

Original
(8 bits)

Uniform
Quantization
(1 bit)

Floyd-Steinberg
Dithering
(1 bit)

62

Random Dithering

- Randomize quantization errors
  - Errors appear as noise

P(x,y) = trunc(I(x,y) + noise(x,y) + .5)

Original
(8 bits)

Uniform Quantization
(1 bit)

Random Dither
(1 bit)

63

Ordered Dither

- Structured quantization error
  - Matrix store pattern of thresholds

```
i = x mod n
j = y mod n
e = I(x,y) - trunc(I(x,y))
if (e > D(i,j))
    P(x,y) = ceil(I(x, y))
else
    P(x,y) = floor(I(x,y))
```

$$D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$

64

Ordered Dither

- Bayer's ordered dither matrices

$$D_n = \begin{bmatrix} 4D_{n/2} + D_2(1,1)U_{n/2} & 4D_{n/2} + D_2(1,2)U_{n/2} \\ 4D_{n/2} + D_2(2,1)U_{n/2} & 4D_{n/2} + D_2(2,2)U_{n/2} \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \qquad D_4 = \begin{bmatrix} 15 & 7 & 13 & 5 \\ 3 & 11 & 1 & 9 \\ 12 & 4 & 14 & 6 \\ 0 & 8 & 2 & 10 \end{bmatrix}$$

65

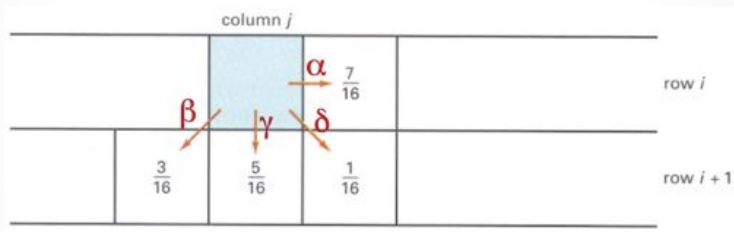
Ordered Dither - Compare

Original (8 bits) | Uniform Quantization (1 bit) | Ordered Dither (1 bit)

66

Error Diffusion Dither

- Spread quantization error over neighboring pixels
  - Error dispersed to pixels right and below



$$\alpha + \beta + \gamma + \delta = 1.0$$

67

Dither Comparison

Original (8 bits) | Random Dither (1 bit) | Ordered Dither (1 bit) | Floyd-Steinberg Dither (1 bit)

68

Image Operations

- Shrink/Enlarge
- Blur/Sharpen
- Edge detection
- Brighten/Darken
- Change Contrast
- Saturate/Desaturate
- Quantize
- Random Dither
- Ordered Dither
- Floyd-Stienberg

69

Assignment 1

- Shrink/Enlarge
- Blur/Sharpen
- Edge detection
- Brighten/Darken
- Change Contrast
- Saturate/Desaturate
- Quantize
- Random Dither
- Ordered Dither
- Floyd-Stienberg

70

Assign. 1 - Details

- Lots of code given already:
  - Read/Write a images (stb_image_write.h)
  - Command line processing
  - Brighten
  - Image & Pixel class
  - Bayer constants
  - Floyd-Stienberg constants
- Should compile very easily!
- Due two weeks from now.

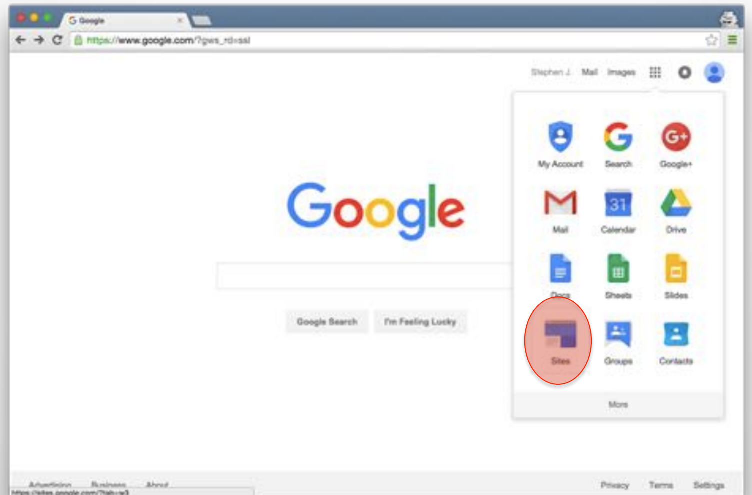
71

HW 0 Turn-in

- What to put on webpage:
  - Code & Any libraries needed to compile
  - Short write-up
  - <u>Pictures of working project!</u>
- Turn in webpage:
  - Through Moodle link
- Create Webpage
  - Easy: Just place html on a server
  - Very Easy: Google Sites through UMN

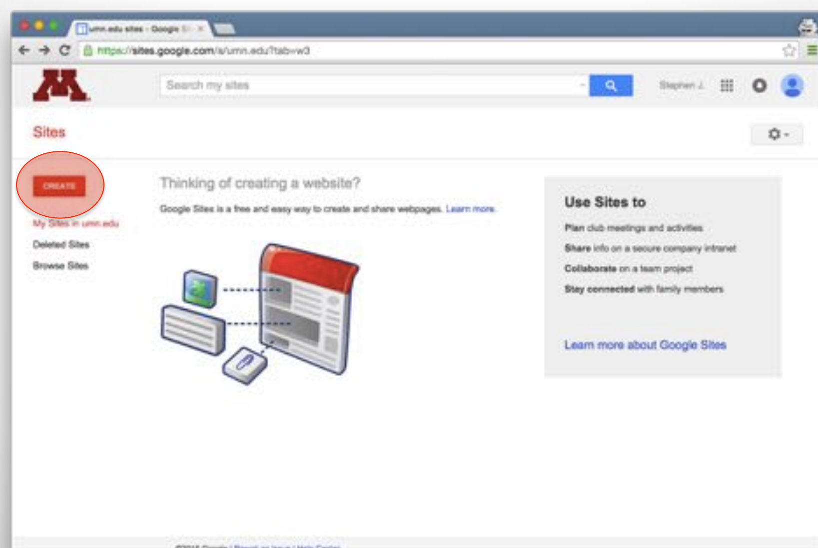
72

Creating a Google Site - 1



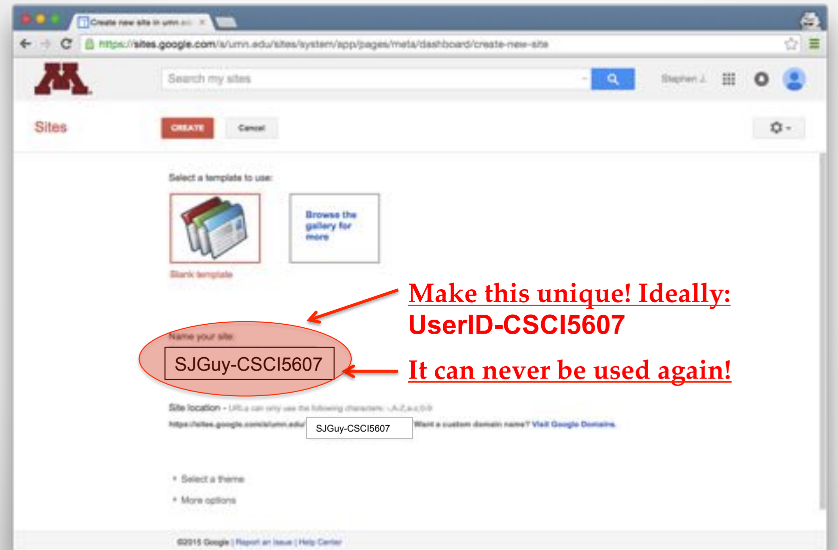
73

Creating a Google Site - 2



74

Creating a Google Site - 3



75

Permission

- If you use Google Sites (or a similar service):
  - Make sure you share it university-wide
- Ensures both me and the TA's can access the webpage

76