CS559: Computer Graphics

Lecture 5: Image Resampling and Painterly Rendering

Li Zhang

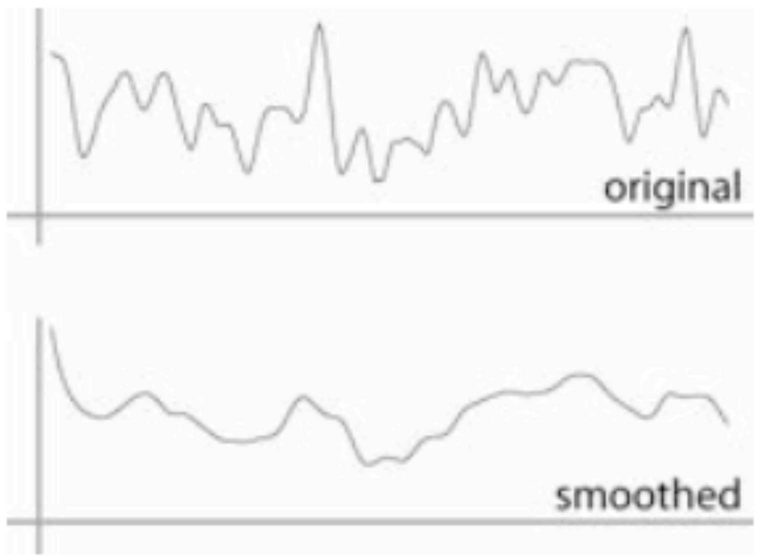
Spring 2010

Announcement

- In-class midterm is re-scheduled on March 22 (Monday)

Last time: Image Convolution and Reconstruction

- Can apply sliding-window average to a continuous function just as well
  - output is continuous
  - integration replaces summation



Continuous convolution

- Sliding average expressed mathematically:

$$g_{\text{smooth}}(x) = \frac{1}{2r} \int_{x-r}^{x+r} g(t)dt$$

note difference in normalization (only for box)

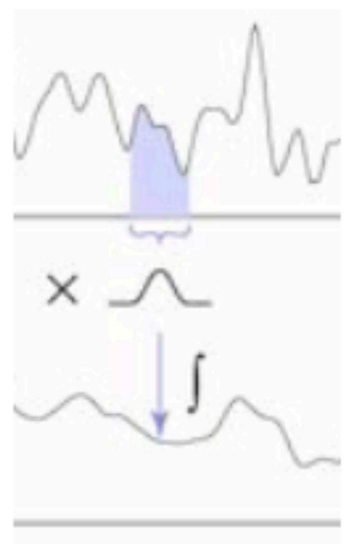
- Convolution just adds weights

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$

weighting is now by a function

weighted integral is like weighted average

again bounds are set by support of *f(x)*



One more convolution

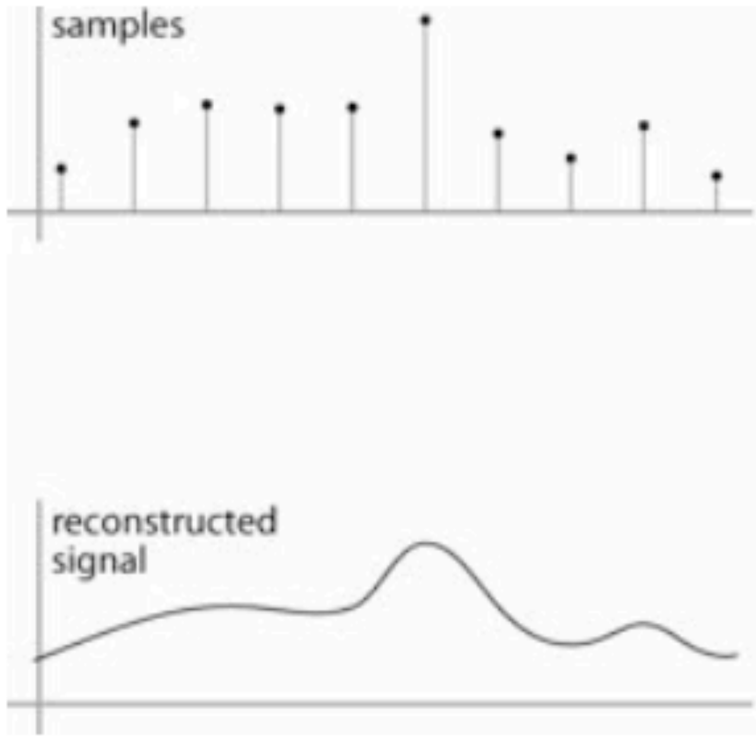
- Continuous–discrete convolution

$$(a \star f)(x) = \sum_i a[i] f(x - i)$$

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

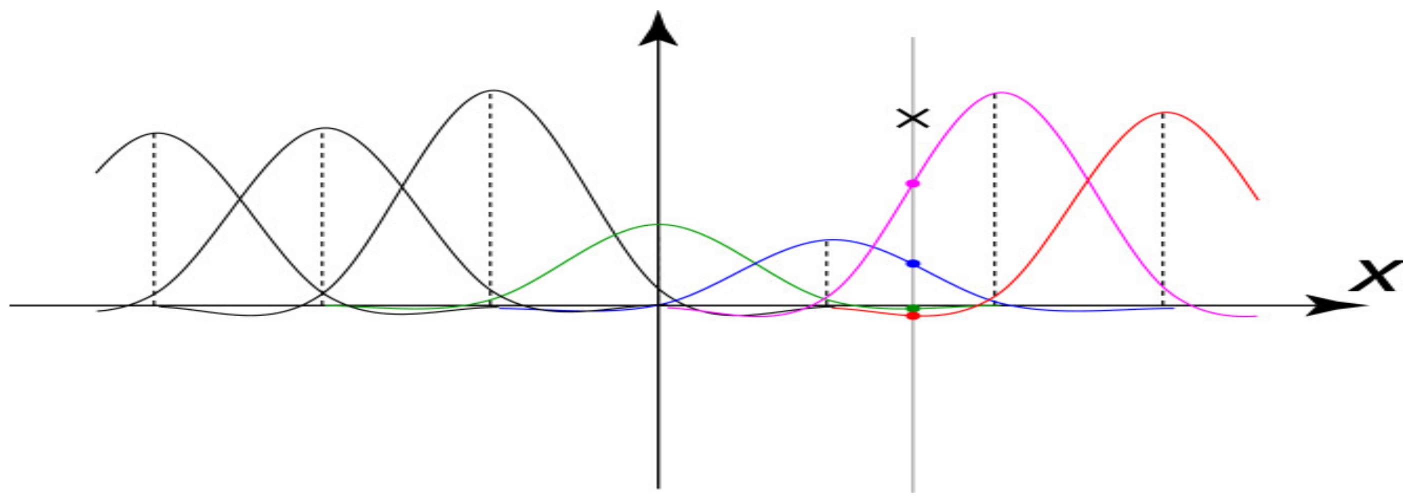
used for reconstruction and resampling

Continuous-discrete convolution



1. putting the flipped reconstruction filter at the desired location
2. evaluating at the original sample positions
3. taking products with the sample values themselves
4. summing it up

Another view on continuous-discrete convolution

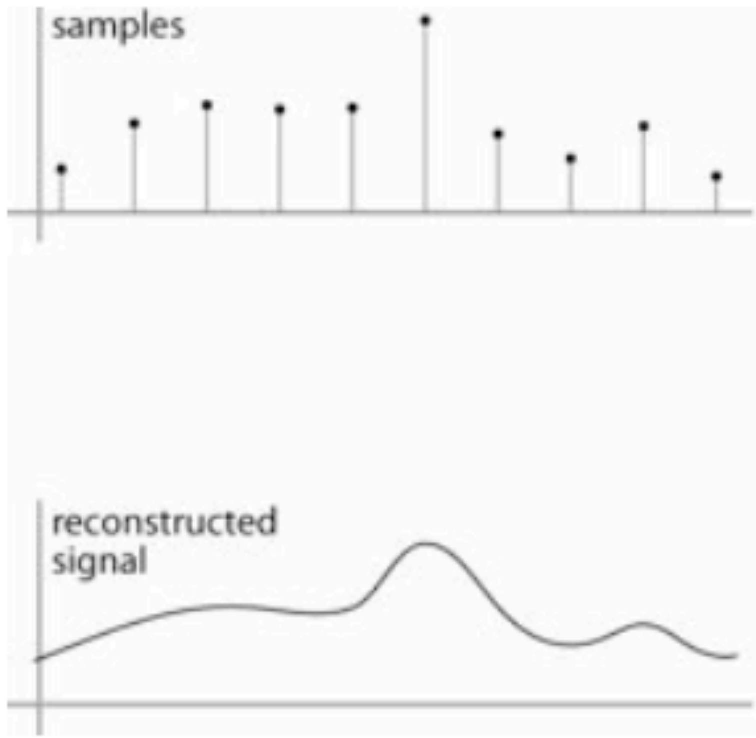


Reconstruction (discrete-continuous convolution) as a sum of shifted copies of the filter

Same view also holds for discrete convolution

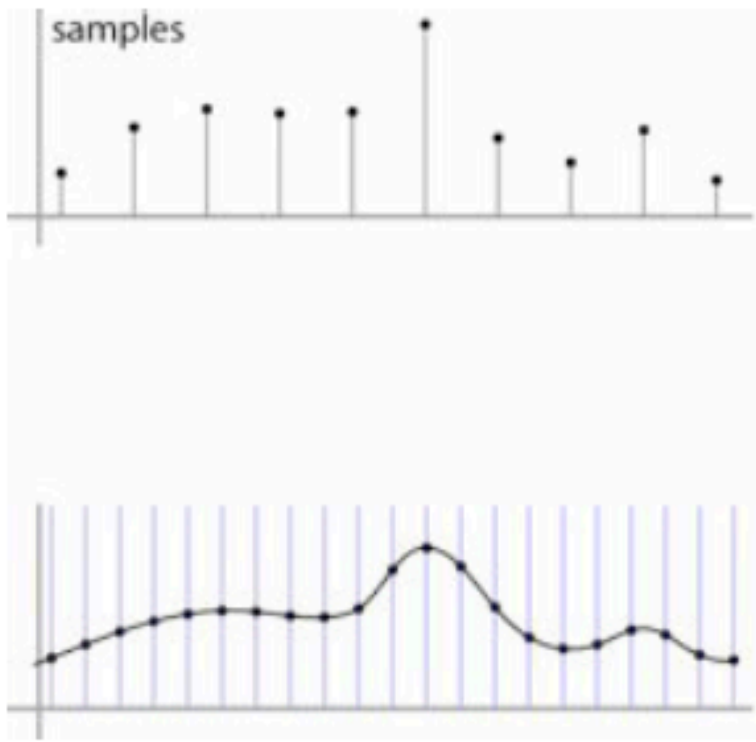
Resampling

- Reconstruction creates a continuous function
  - forget its origins, go ahead and sample it



Resampling

- Reconstruction creates a continuous function
  - forget its origins, go ahead and sample it



Cont.–disc. convolution in 2D

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

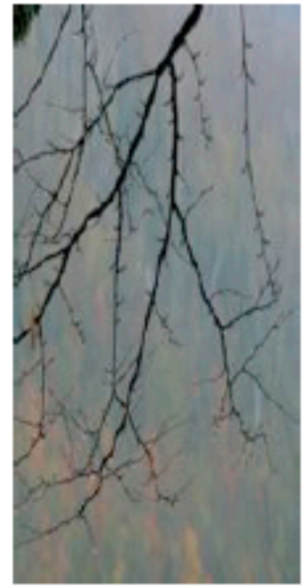
An Example:

Separable filters for resampling

- just as in filtering, separable filters are useful
  - separability in this context is a statement about a continuous filter, rather than a discrete one:

$$f_2(x, y) = f_1(x)f_1(y)$$

[Philip Greenspun]



two-stage resampling using a separable filter

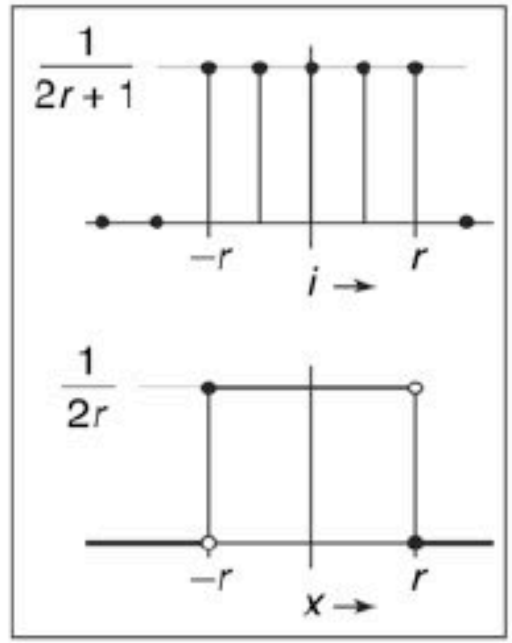
A gallery of filters

- Box filter
  - Simple and 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 image upsampling

Box filter

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

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



Discontinuous Reconstruction

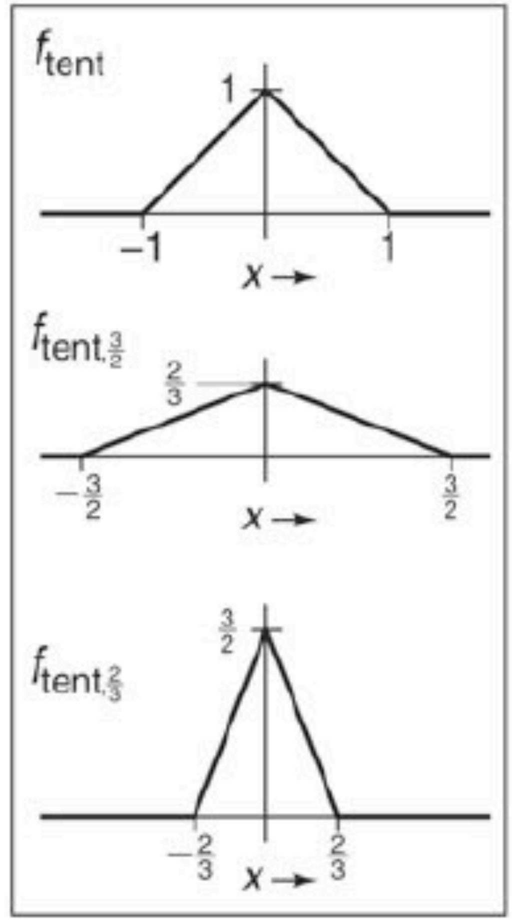
Today’s topics

- Finish Resampling
- Painterly Rendering
- Edges

Tent filter

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

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



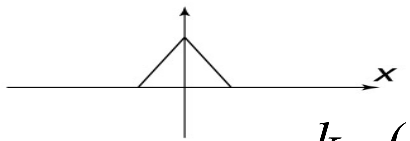
How to use tent filter

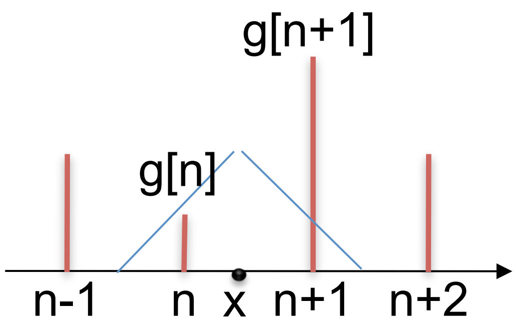
- Method 1

- Method 2

Reconstruction using 1D tent filter

1D example:





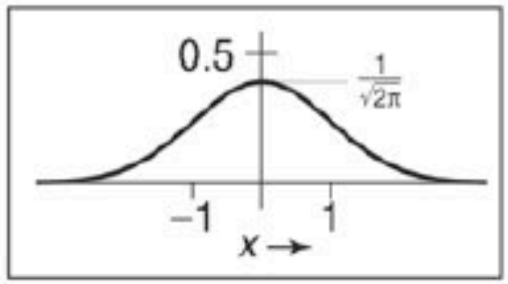
$$k_{1D}(x) = \begin{cases} 1 - |x| & |x| < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta x = x - n$$

$$f(x) = g[n] \cdot (1 - \Delta x) + g[n+1] \cdot \Delta x$$

Tent filter reconstruction: Zero-order continuity
Use only one multiplication?

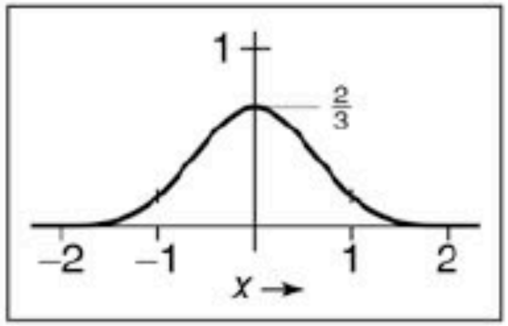
Gaussian filter



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

Infinitely smooth, negligible beyond [-3,3]

B-Spline cubic



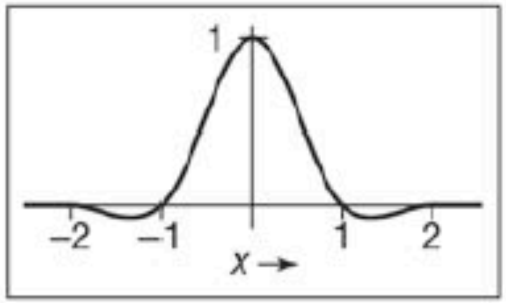
$$f_B(x) = \frac{1}{6}\begin{cases} -3(1-|x|)^3 + 3(1-|x|)^2 + 3(1-|x|) + 1 & -1 \le x \le 1, \\ (2-|x|)^3 & 1 \le |x| \le 2, \\ 0 & \text{otherwise.} \end{cases}$$

C2 Smoothness

Can be obtained by convolving a box filter four times

What's the problem to use it as a reconstruction filter?

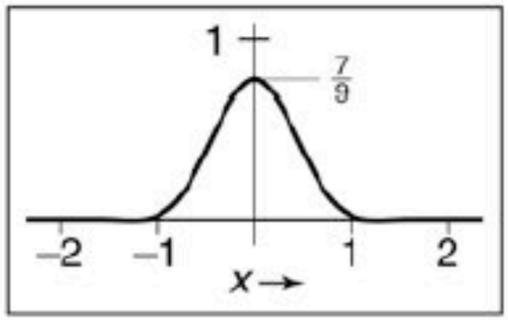
Catmull-Rom cubic



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

C1 Smoothness
It interpolates samples: “connecting the dots”

Michell-Netravali cubic



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

All-around best choice [Mitchell & Netravali 1988]

Effects of reconstruction filters

- For some filters, the reconstruction process winds up implementing a simple algorithm
- Box filter (radius 0.5): nearest neighbor sampling
  - box always catches exactly one input point
  - it is the input point nearest the output point
  - so output[i, j] = input[round(x(i)), round(y(j))]
    - x(i) computes the position of the output coordinate i on the input grid

Effects of reconstruction filters

- For some filters, the reconstruction process winds up implementing a simple algorithm
- Box filter (radius 0.5): nearest neighbor sampling

  box always catches exactly one input point

  it is the input point nearest the output point

  so output[i, j] = input[round(x(i)), round(y(j))]
    x(i) computes the position of the output coordinate i on the input grid
- Tent filter (radius 1): linear interpolation

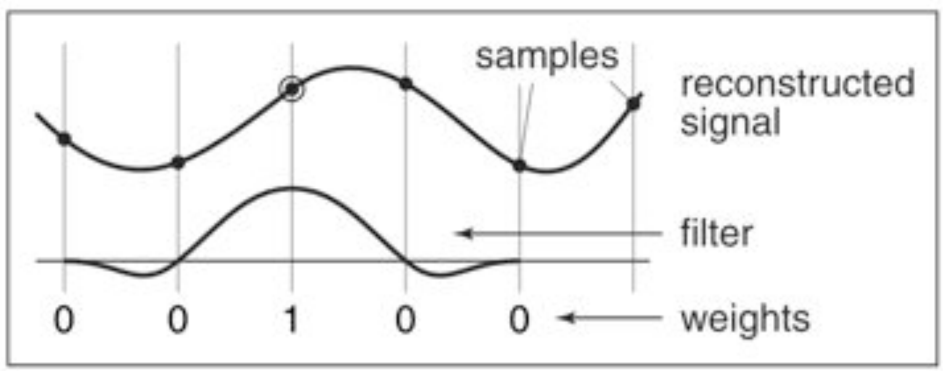
  tent catches exactly 2 input points

  weights are a and $(1 - a)$

  result is straight-line interpolation from one point to the next

Properties of Kernels

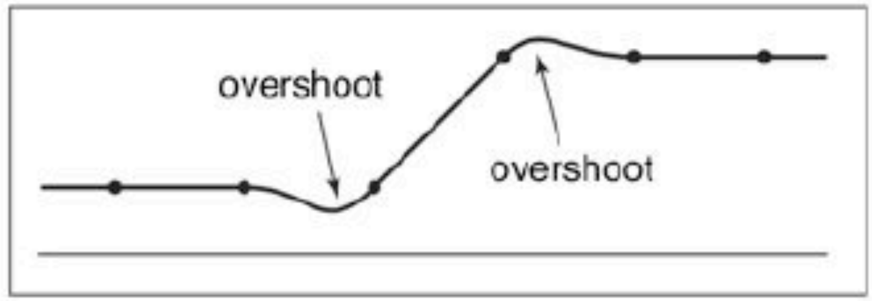
- Filter, Impulse Response, or kernel function, same concept but different names
- Degrees of continuity
- Interpolating or no
- Ringing or overshooting



Interpolating filter for reconstruction

Ringing, overshoot, ripples

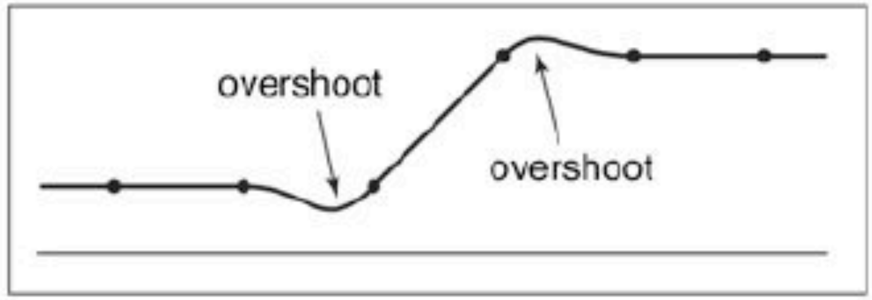
- Overshoot
  - caused by
    - negative filter values

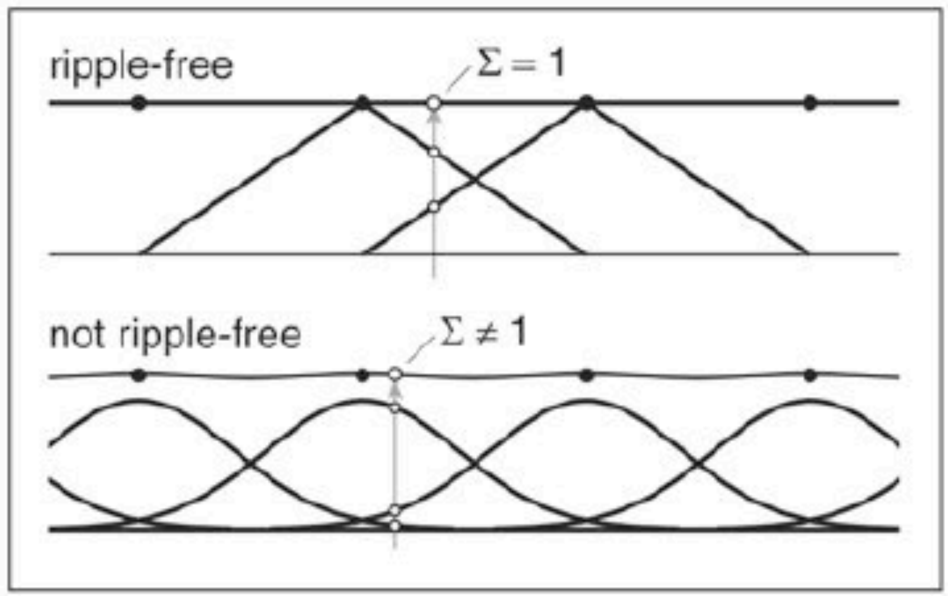


Ringing, overshoot, ripples

- Overshoot
  - caused by
    - negative filter values
- Ripples
  - constant in,
    - non-const. out
  - ripple free when:

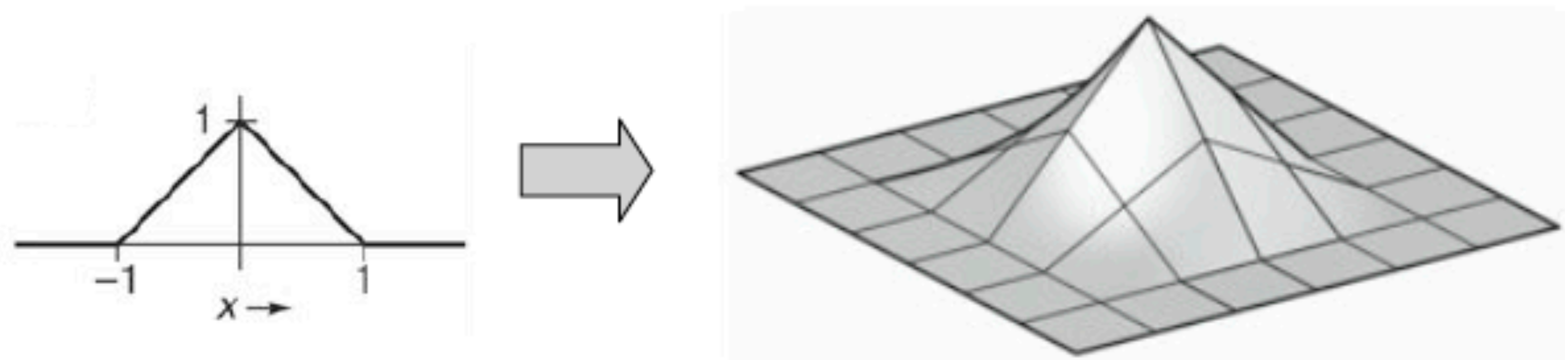
$$\sum_i f(x+i) = 1 \quad \text{for all } x.$$

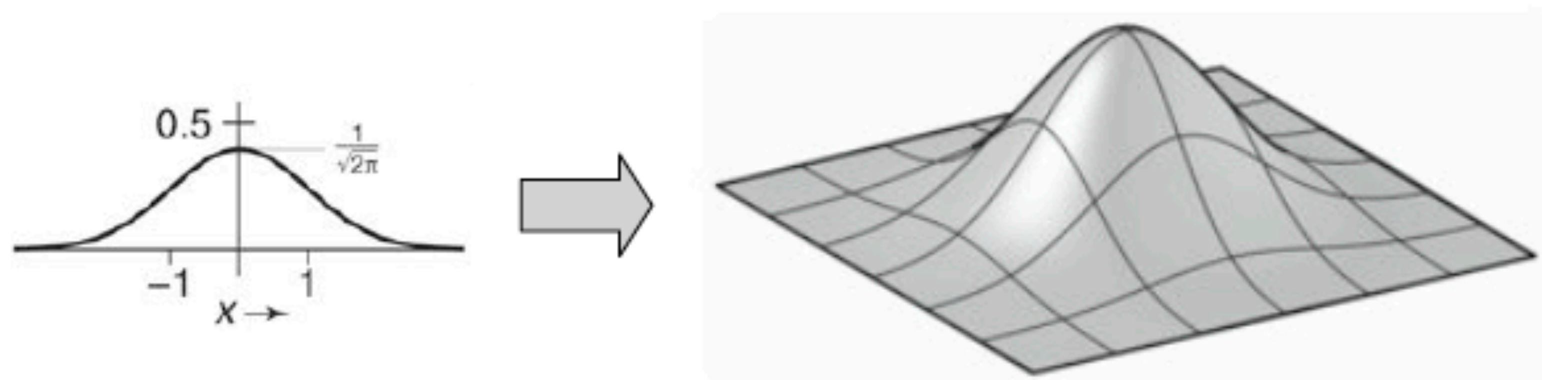




Constructing 2D filters

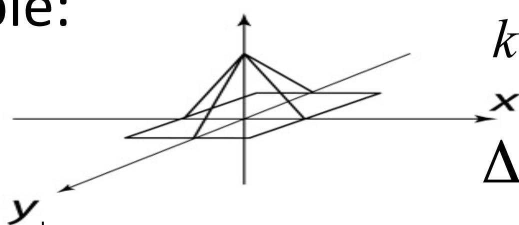
- Separable filters (most common approach)

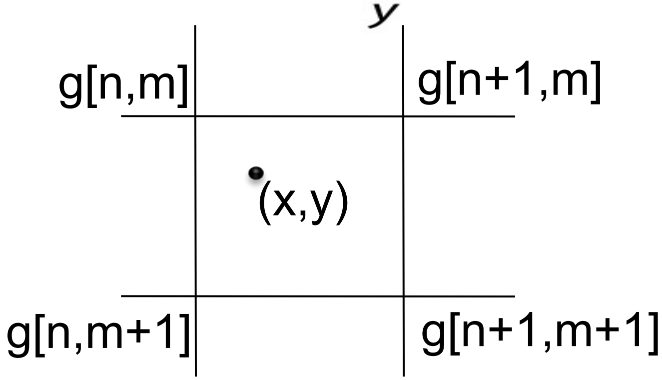




Reconstruction filter Examples in 2D

2D example:





$$k_{2D}(x,y) = \begin{cases} (1-|x|)(1-|y|) & |x|<1, |y|<1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta x = x - n, \Delta y = y - m$$

$$\begin{aligned} f(x,y) = & \; g[n,m]\cdot(1-\Delta x)\cdot(1-\Delta y) \\ & + g[n+1,m]\cdot \Delta x \cdot (1-\Delta y) \\ & + g[n,m+1]\cdot(1-\Delta x)\cdot \Delta y \\ & + g[n+1,m+1]\cdot \Delta x \cdot \Delta y \end{aligned}$$

How to simplify the calculation?

Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:

[Philip Greenspun]

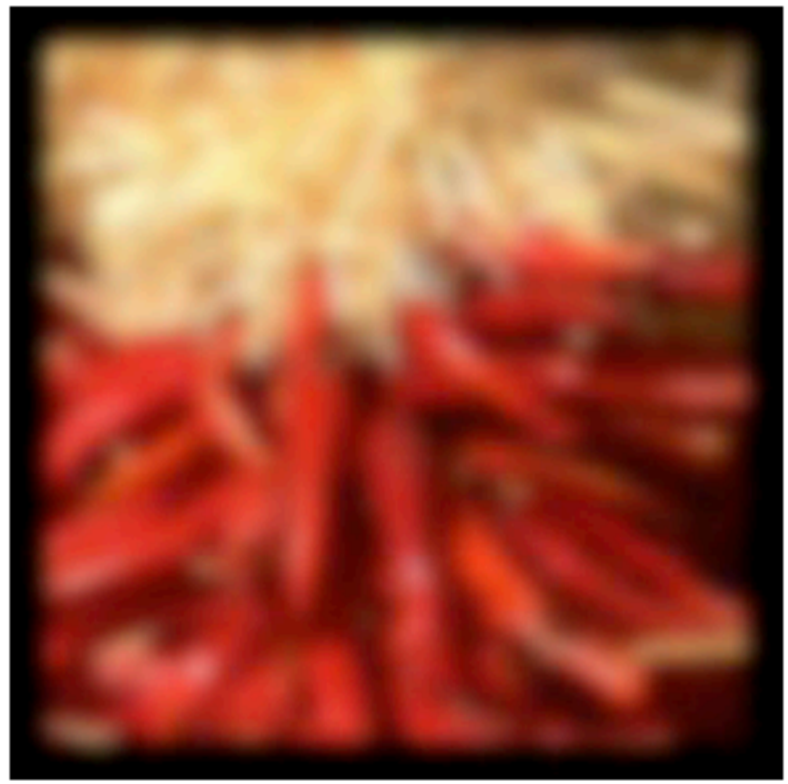
Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)

[Philip Greenspun]

Yucky details

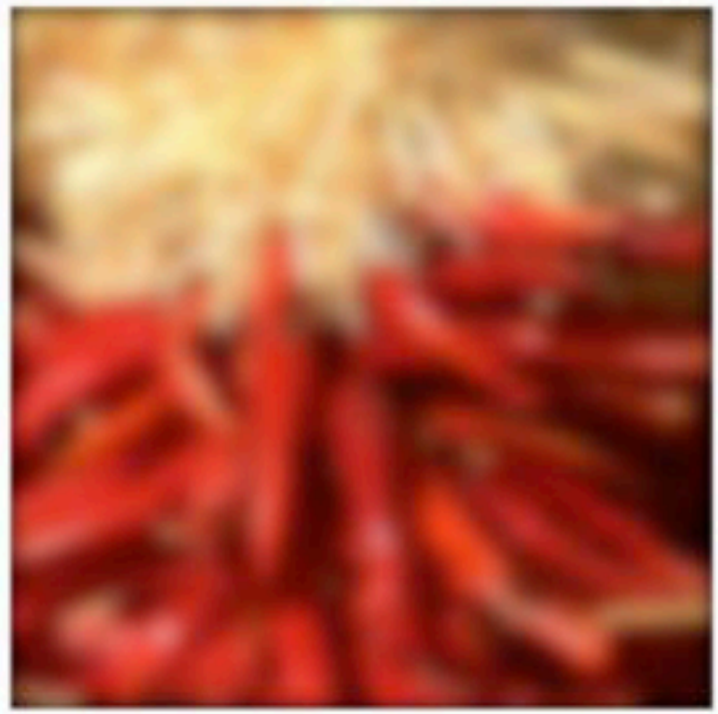
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)



[Philip Greenspun]

Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)



[Philip Greenspun]

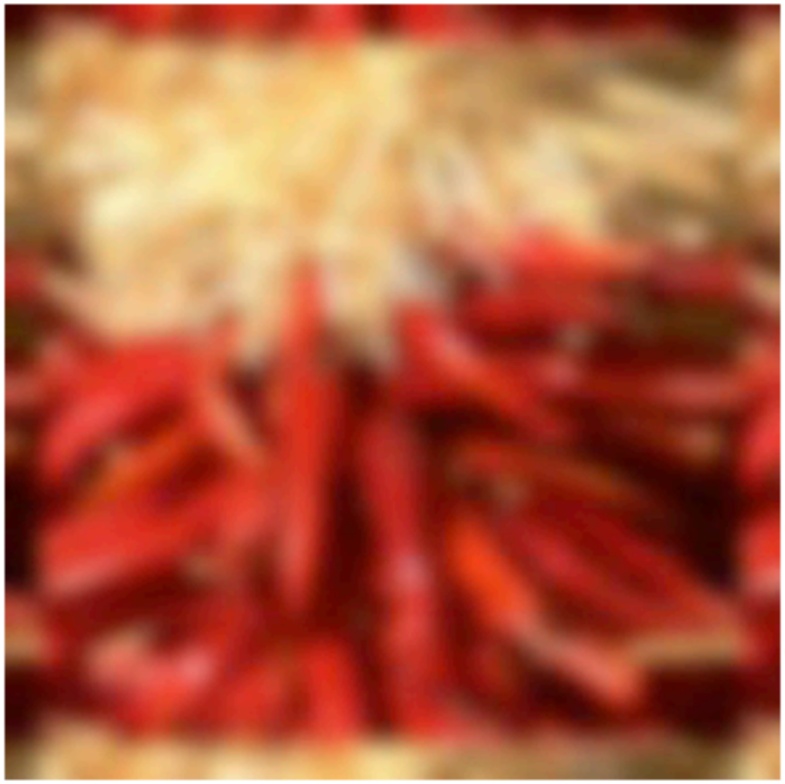
Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around

[Philip Greenspun]

Yucky details

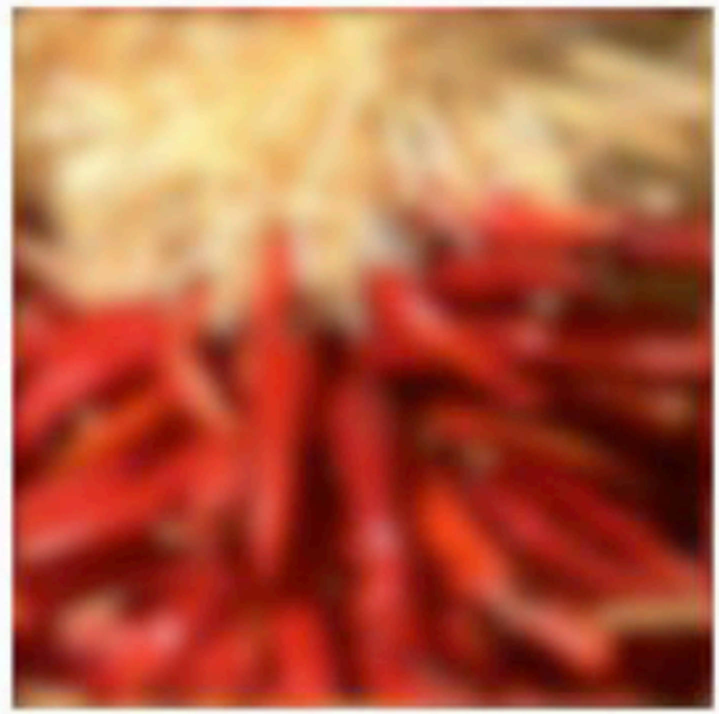
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around



[Philip Greenspun]

Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around



[Philip Greenspun]

Yucky details

- What about near the edge?

  the filter window falls off the edge of the image

  need to extrapolate

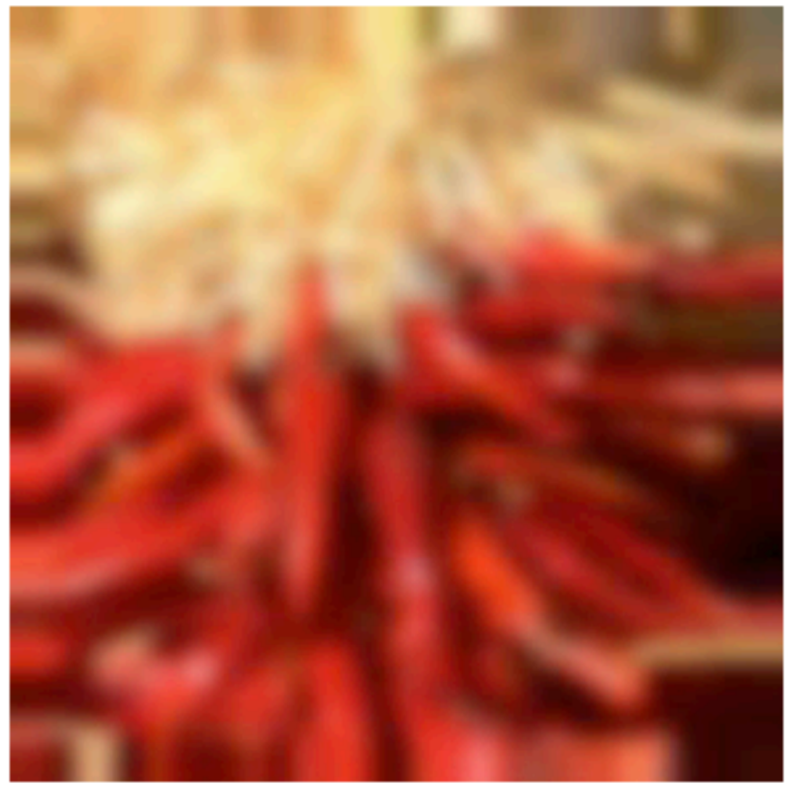
  methods:

  - clip filter (black)
  - wrap around
  - copy edge

[Philip Greenspun]

Yucky details

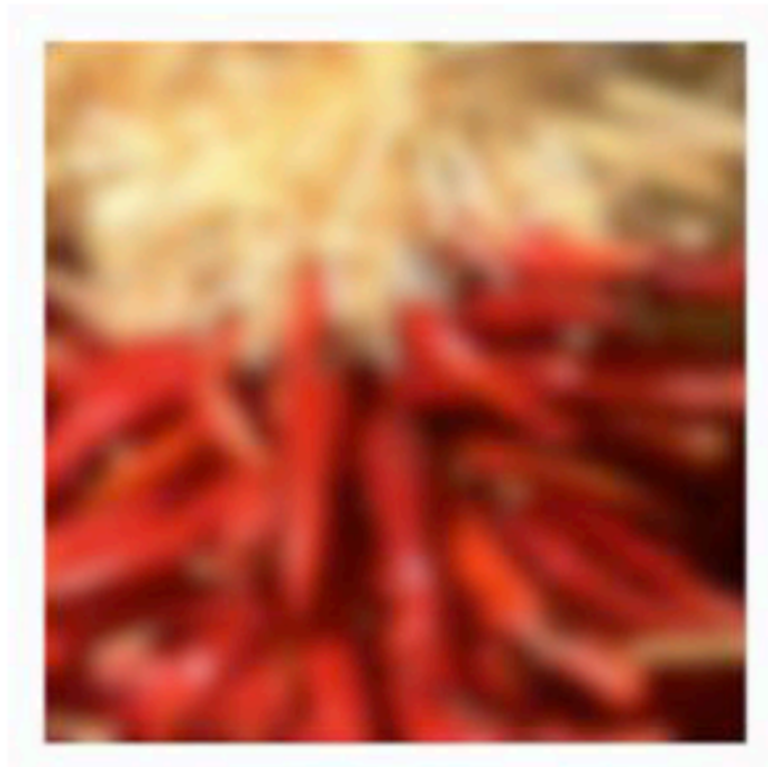
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge



[Philip Greenspun]

Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge



[Philip Greenspun]

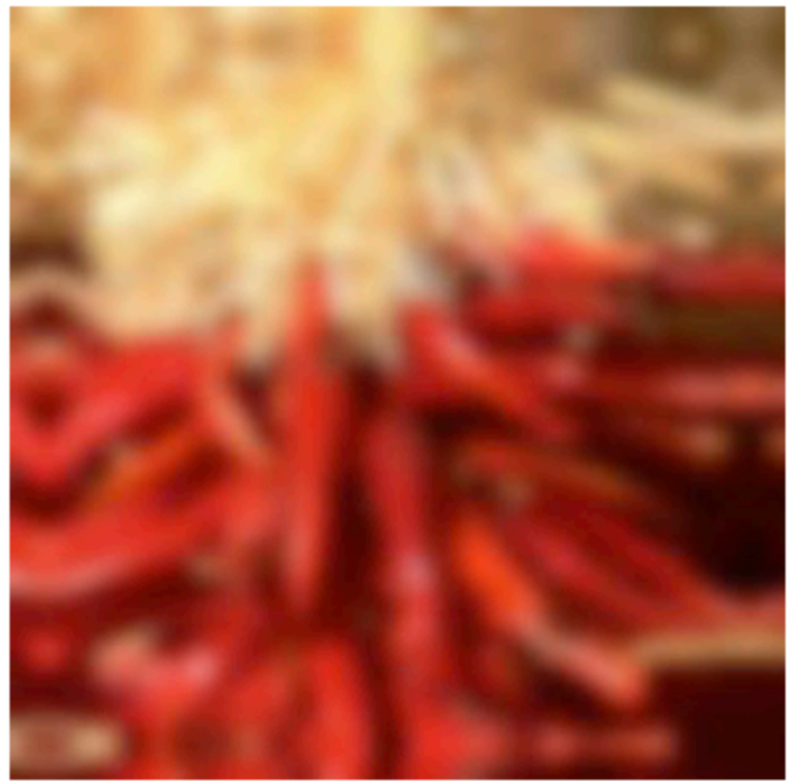
Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge

[Philip Greenspun]

Yucky details

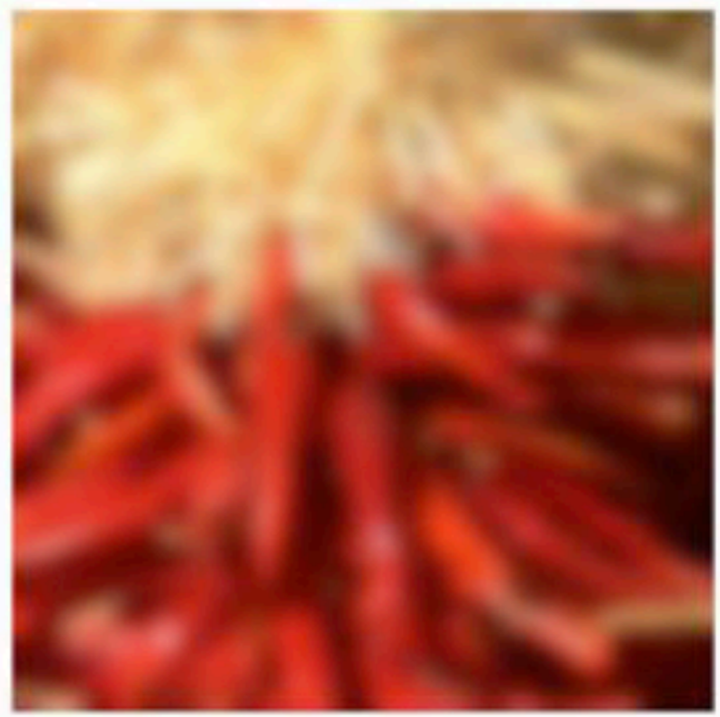
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge



[Philip Greenspun]

Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge



[Philip Greenspun]

Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge

Image Filter Near Boundaries

?	1	3	9	4	5	8	8	1	3	7

*

0.25	0.5	0.25

=

- Kernel Renormalization

Image Filter Near Boundaries

?	1	3	9	4	5	8	8	1	3	7

*

0	0.5	0.25

/ 0.75

=

- Kernel Renormalization

Reducing and enlarging

- Very common operation
  - devices have differing resolutions
  - applications have different memory/quality tradeoffs
- Also very commonly done poorly
- Simple approach: drop/replicate pixels
- Correct approach: use resampling

Practical upsampling

- This can also be viewed as:
  1. putting the reconstruction filter at the desired location
  2. evaluating at the original sample positions
  3. taking products with the sample values themselves
  4. summing it up

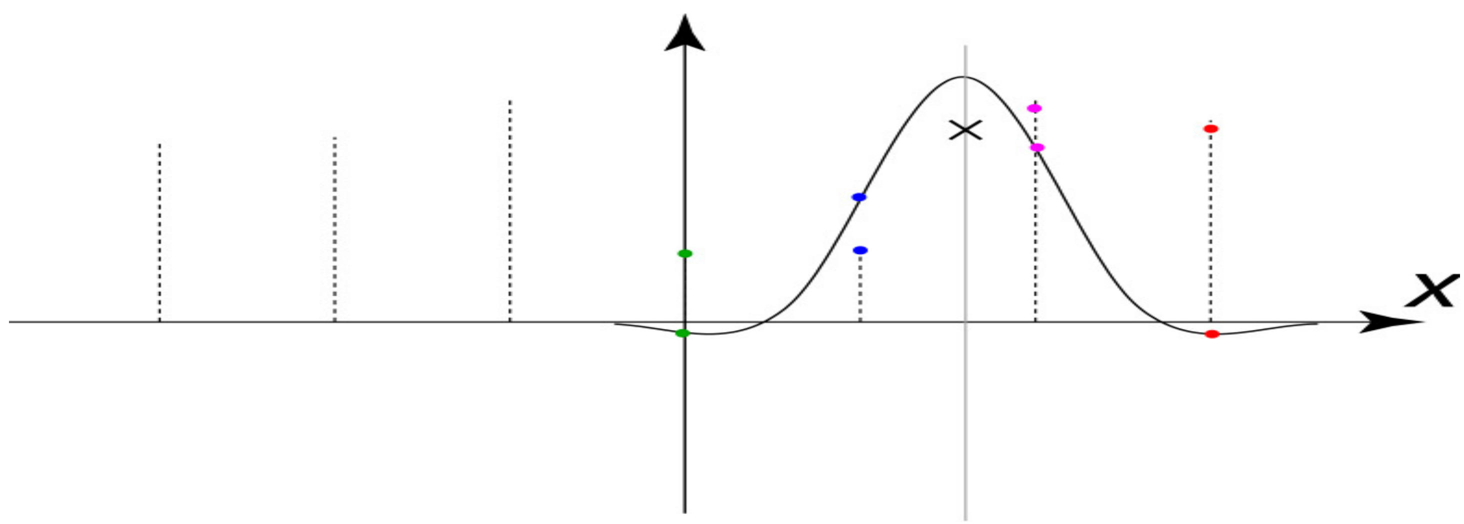
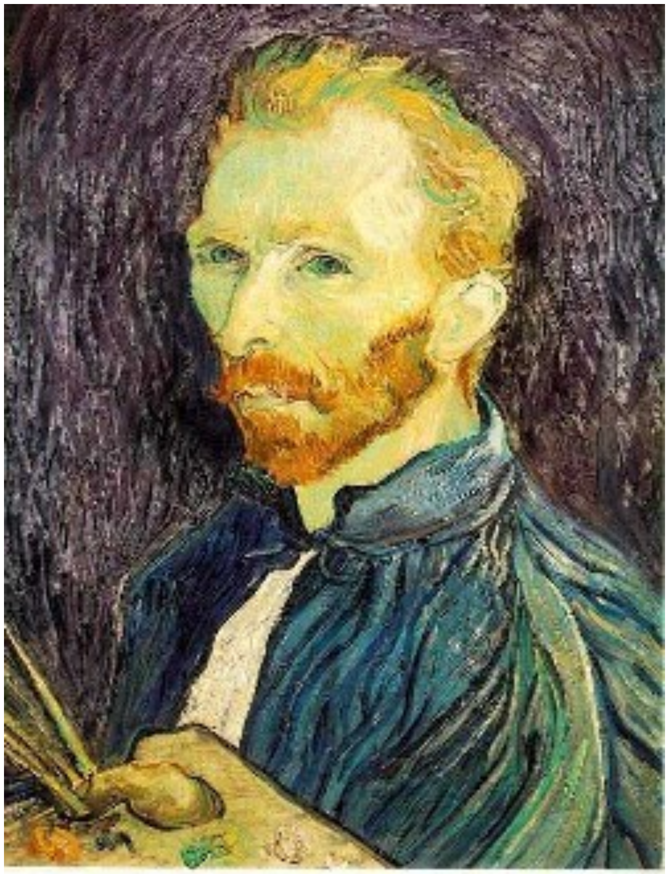


Image Downsampling

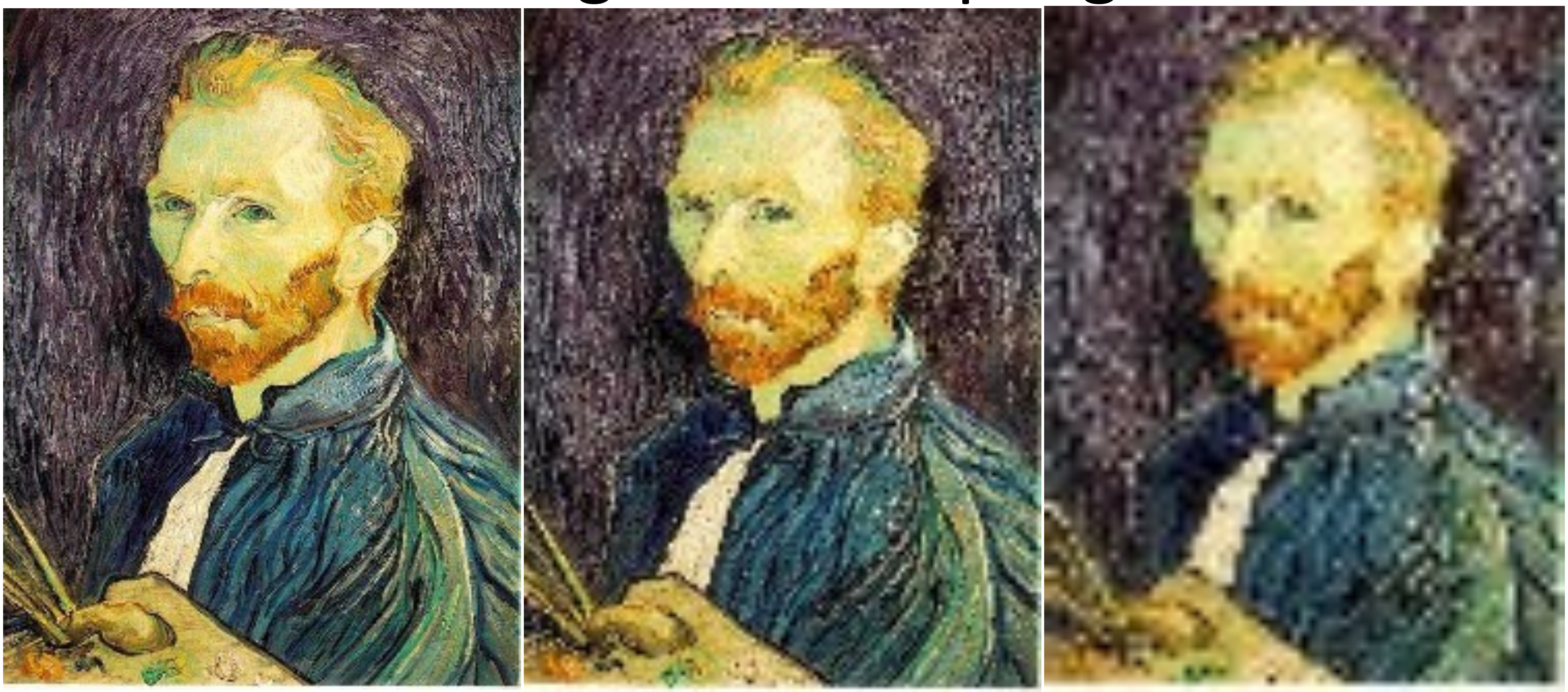


Throw away every other row and
column to create a 1/2 size image
- called *image sub-sampling*

1/4

1/8

Image sub-sampling

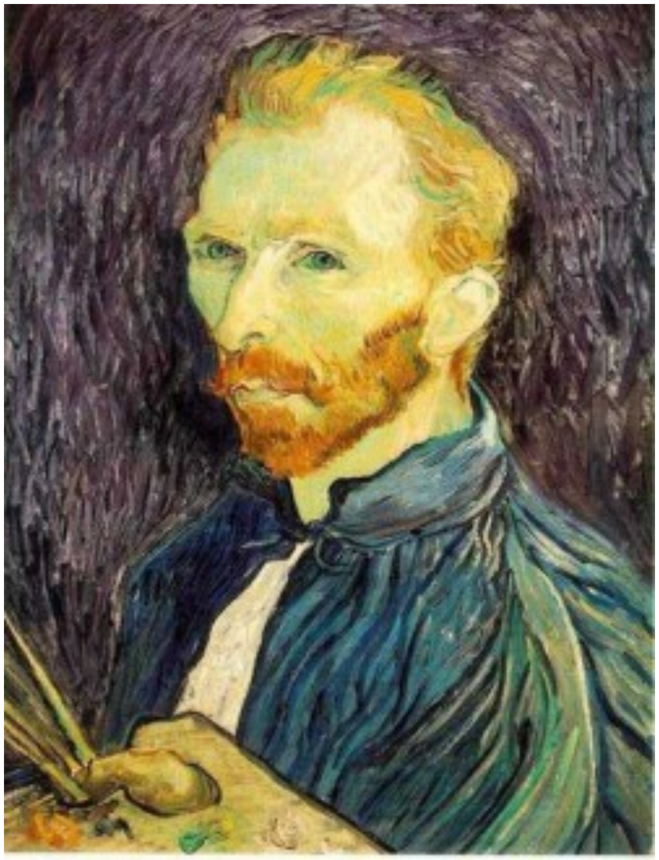


1/2 | 1/4 (2x zoom) | 1/8 (4x zoom)

Why does this look so crufty?

Minimum Sampling requirement is not satisfied – resulting in **Aliasing effect**

Subsampling with Gaussian pre-filtering



Gaussian 1/2

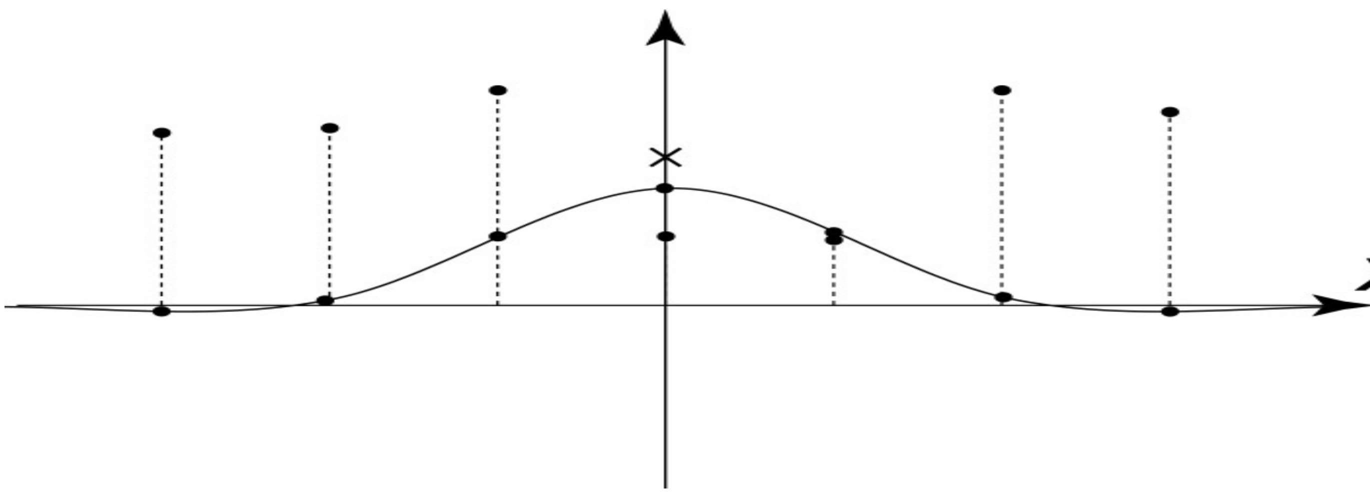
G 1/4

G 1/8

- Solution: filter the image, *then* subsample

Practical downsampling

- Downsampling is similar, but filter has larger support and smaller amplitude.
- Operationally:
    1. Choose reconstruction filter in downsampled space.
    2. Compute the downsampling rate, *d*, ratio of new sampling rate to old sampling rate
    3. Stretch the filter by 1/*d* and scale it down by *d*
    4. Follow upsampling procedure (previous slides) to compute new values (need normalization)



Filter Choice: speed vs quality

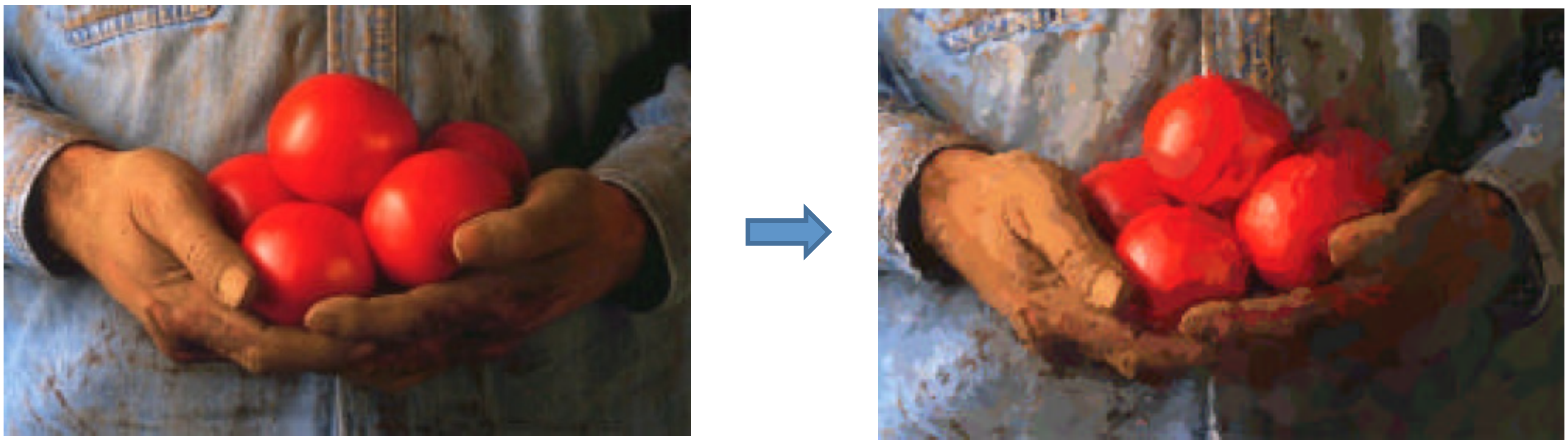
Box filter: very fast

Tent filter: moderate quality

Cubic filter: excellent quality, for example Mitchell filter.

Today

- Painterly rendering



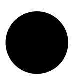
- Reading
  - Hertzmann, *Painterly Rendering with Curved Brush Strokes of Multiple Sizes,* SIGGRAPH 1998, section 2.1 (required), others (optional)
  - Doug DeCarlo, Anthony Santella. Stylization and Abstraction of Photographs In SIGGRAPH 2002, pp. 769-776. (optional)
  - Edge Detection Tutorial (recommended but optional)

Painterly Filters

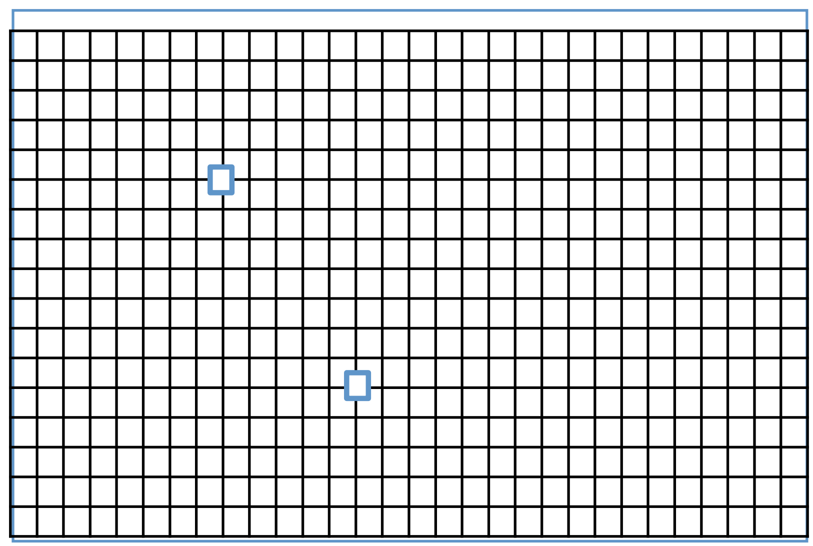
- Many methods have been proposed to make a photo look like a painting
  - A.k.a. Non-photorealistic Rendering
- Today we look at one: *Painterly-Rendering with Brushes of Multiple Sizes*
- Basic ideas:
  - Build painting one layer at a time, from biggest to smallest brushes
  - At each layer, add detail missing from previous layer

Input photo

Blurred input



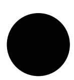
Brush shape



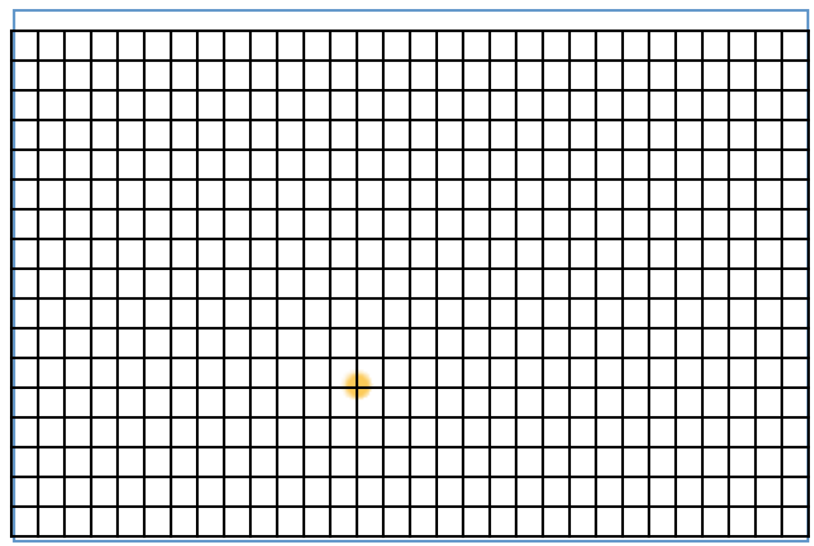
Canvas

Input photo

Blurred input



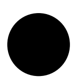
Brush shape



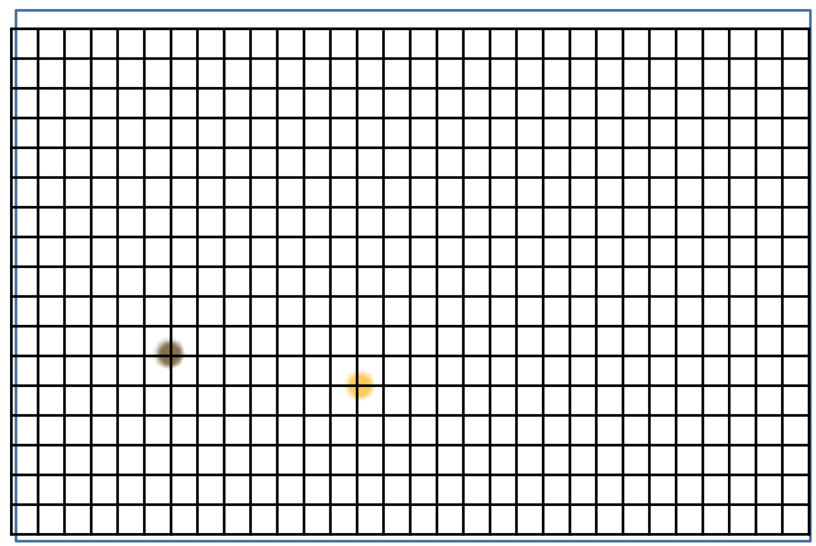
Canvas

Input photo

Blurred input



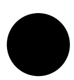
Brush shape



Canvas

Input photo

Blurred input



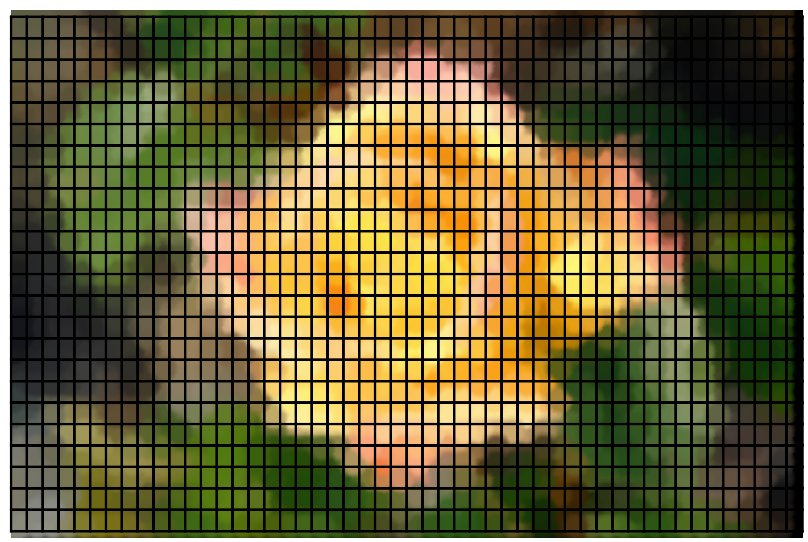
Brush shape

Canvas

Input photo

Blurred input

Brush shape



Canvas

Input photo

Blurred input

Canvas (1st iteration)

Canvas (2nd iteration)

Input photo

Blurred input

Brush shape

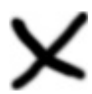
Canvas (2nd iteration)

Input photo

Blurred input

Brush shape

Canvas (3rd iteration)



Brush shape

Iteration 1

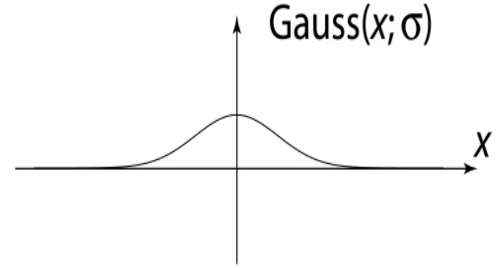
Iteration 2

Iteration 3

How to blur an image?

- Continuous Gaussian Filter

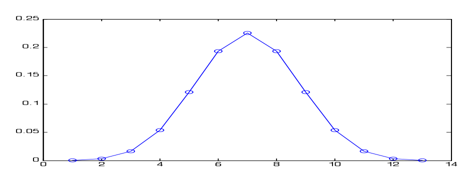
$$Gauss(x;\sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$$



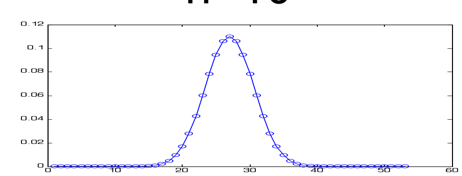
- Discrete Gaussian Filter

$$G_\sigma(i) = \frac{1}{Z} Gauss(i;\sigma),$$
$$i \in [-N, N],$$
$$Z = \sum_{i=-N}^{N} Gauss(i;\sigma)$$

- Binomial Filter
  - $B_1 = [1, 1]/2$
  - $B_2 = B_1 * B_1 = [1,2,1]/4$
  - $B_3 = B_2 * B_1 = [1,3,3,1]/8$
  - $B_4 = B_3 * B_1 = [1,4,6,4,1]/16$
  - …
  - $B_n = B_{n-1} * B_1$

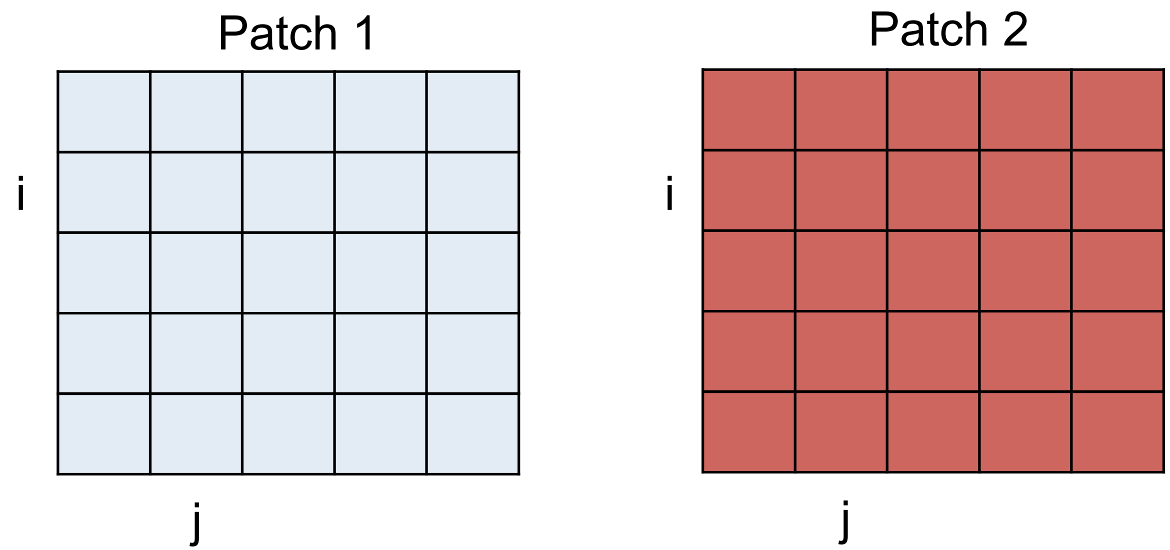


n=10



n=50

Image Patch Difference



$$D_{i,j} = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

Algorithm (outer loop)

function paint(sourceImage,$\boldsymbol{R_1}$ ... $\boldsymbol{R_n}$) *// take source and several brush sizes*

```
{
    canvas := a new constant color image
    // paint the canvas with decreasing sized brushes
    for each brush radius Ri, from largest to smallest do
    {
        // Apply Gaussian smoothing with a filter of size fσRi
        // Brush is intended to catch features at this scale
        referenceImage = sourceImage * G(fσRi)
        // Paint a layer
        paintLayer(canvas, referenceImage, Ri)
    }
    return canvas
}
```

Algorithm (inner loop)

```
procedure paintLayer(canvas,referenceImage, R) // Add a layer of strokes
{
        S := a new set of strokes, initially empty
        D := difference(canvas,referenceImage) // euclidean distance at every pixel
        for x=0 to imageWidth stepsize grid do // step in size f_g R that depends on brush radius
                for y=0 to imageHeight stepsize grid do {
                         // sum the error near (x,y)
                        M := the region (x-grid/2..x+grid/2, y-grid/2..y+grid/2)
                        areaError := sum(D_i,j for i,j in M) / grid^2
                        if (areaError > T) then {
                                // find the largest error point
                                (x1,y1) := max D_i,j in M
                                s :=makeStroke(R,x1,y1,referenceImage)
                                add s to S
                        }
                }
        paint all strokes in S on the canvas, in random order
}
```

f_σ and f_g

- Gauss sigma = $f_\sigma \cdot$ brush radius
  - Or use binomial filter of length 2 · brush radius + 1

- Grid size = $f_g \cdot$ brush radius
  - Default $f_g = 1$

- Trying different parameters are optional

Results in the paper

Original

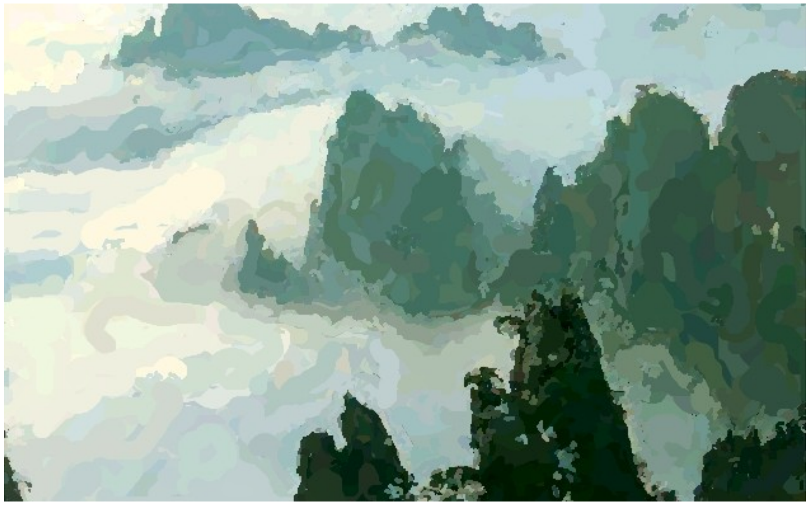
Biggest brush

Medium brush added

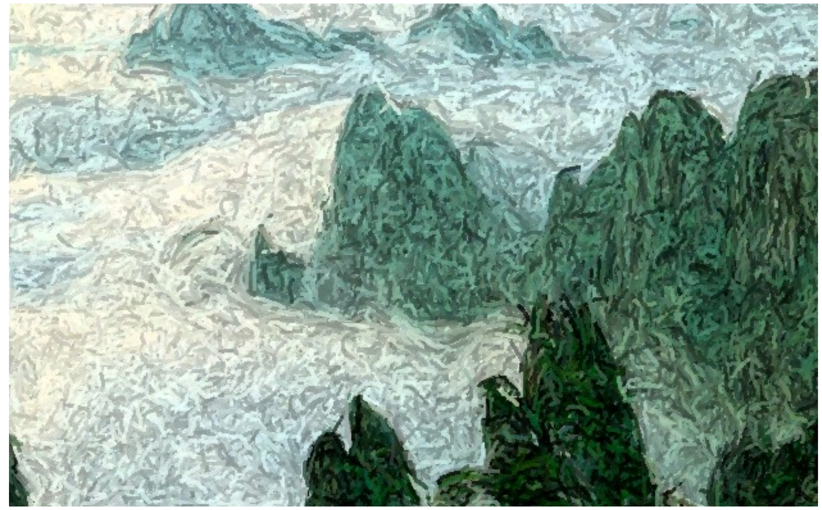
Finest brush added

Changing Parameters

Changing Parameters

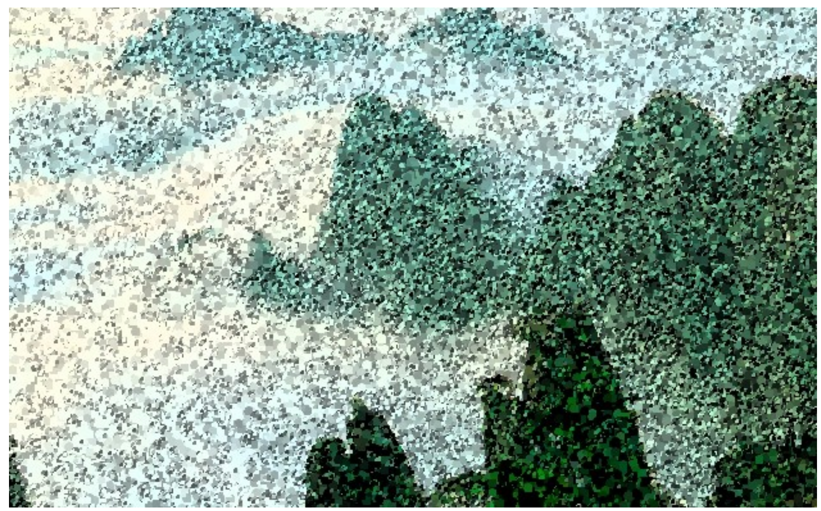


Impressionist, normal painting style



Expressionist, elongated stroke

Colorist wash, semitransparent stroke with color jitter



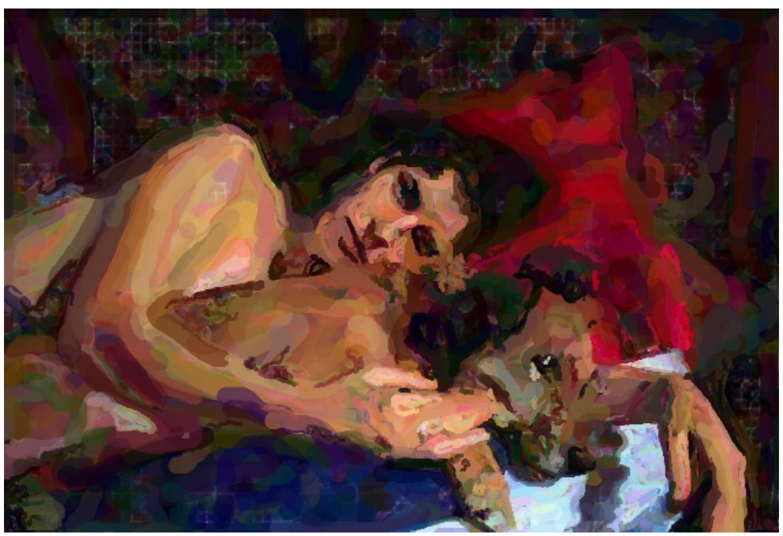
Densely-placed circles with random hue and saturation

Changing Parameters

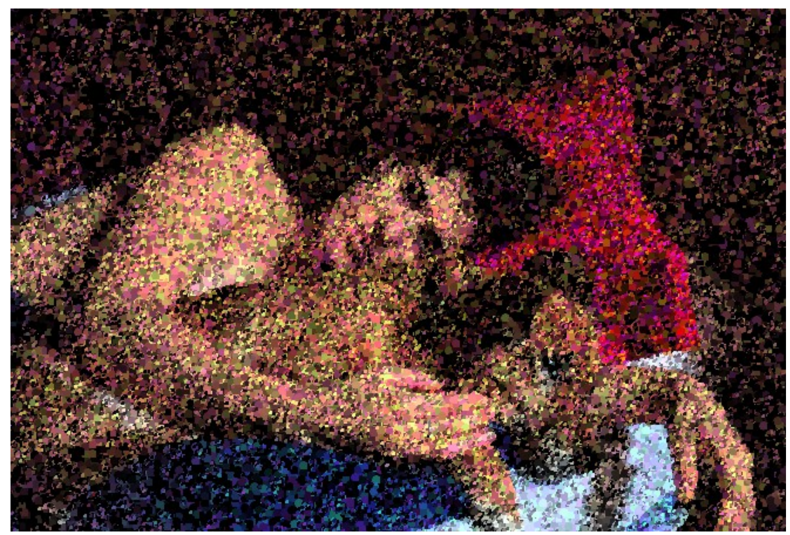
Changing Parameters

Impressionist, normal painting style

Expressionist, elongated stroke



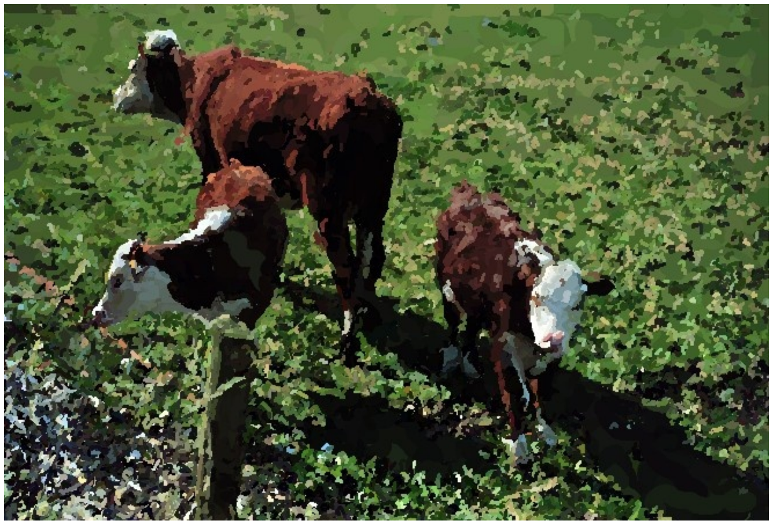
Colorist wash, semitransparent stroke with color jitter



Densely-placed circles with random hue and saturation

Changing Parameters

Changing Parameters



Impressionist, normal painting style

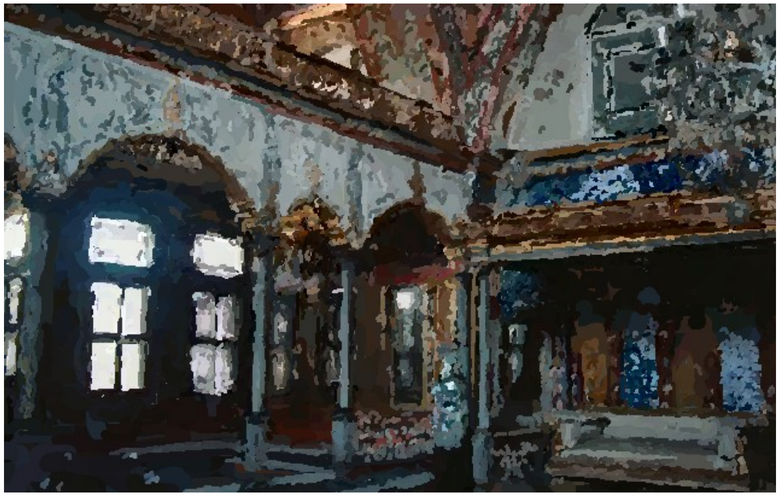
Expressionist, elongated stroke

Colorist wash, semitransparent stroke with color jitter

Densely-placed circles with random hue and saturation

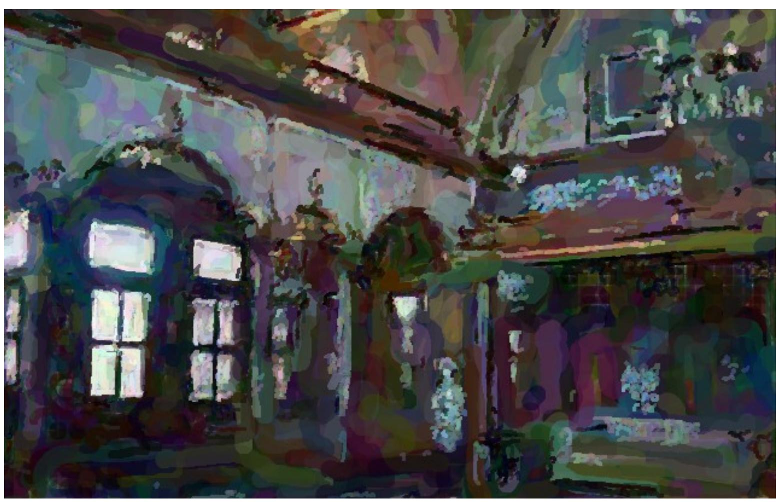
Changing Parameters

Changing Parameters



Impressionist, normal painting style

Expressionist, elongated stroke

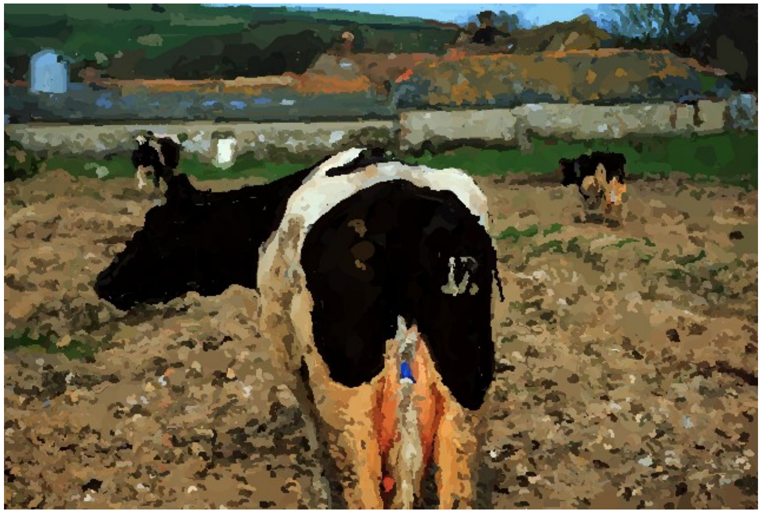


Colorist wash, semitransparent stroke with color jitter

Densely-placed circles with random hue and saturation

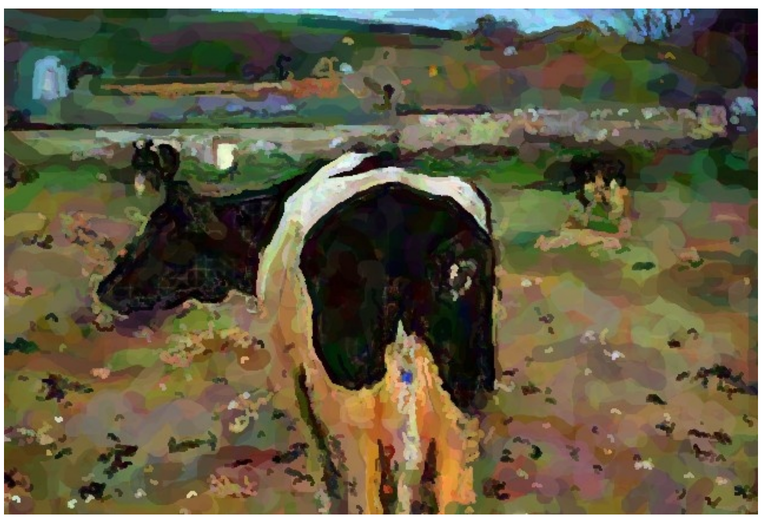
Changing Parameters

Changing Parameters

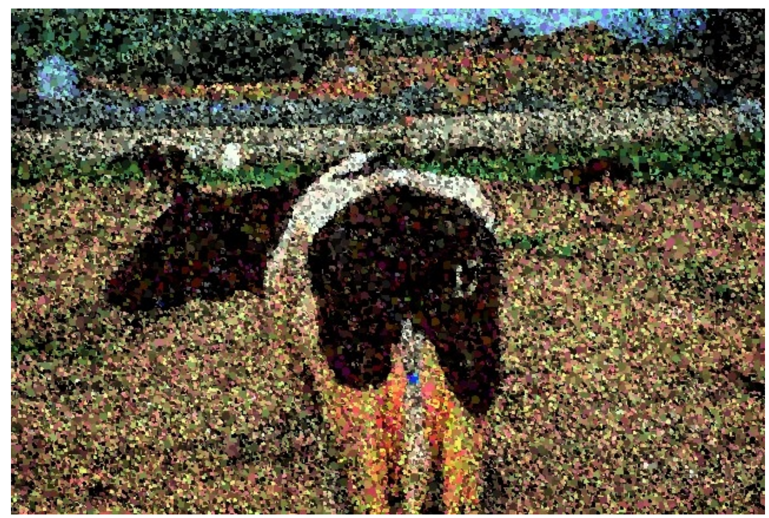


Impressionist, normal painting style

Expressionist, elongated stroke

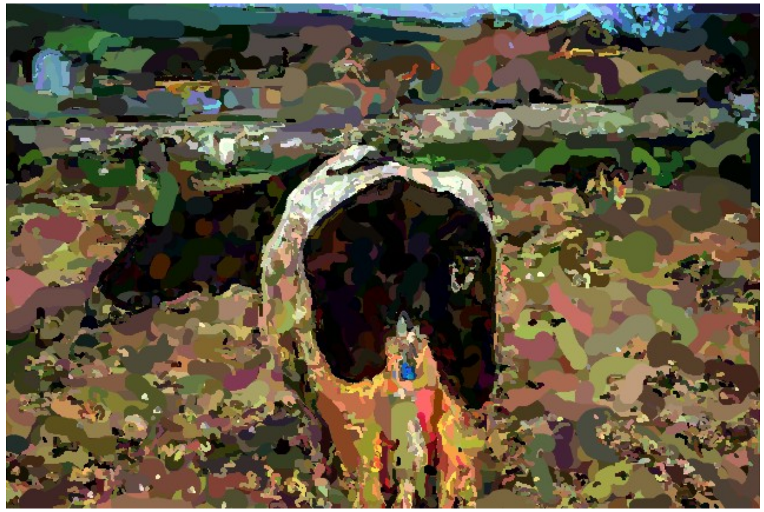


Colorist wash, semitransparent stroke with color jitter



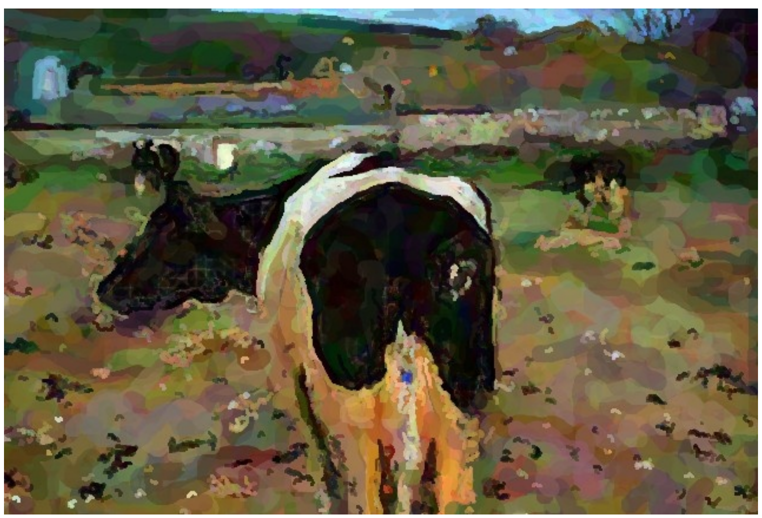
Densely-placed circles with random hue and saturation

Style Interpolation

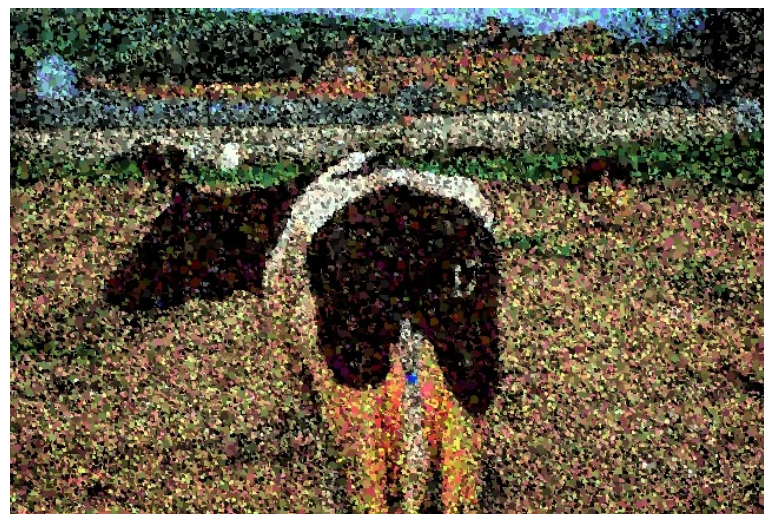


http://mrl.nyu.edu/projects/npr/painterly/

Average style



Colorist wash, semitransparent stroke with color jitter



Densely-placed circles with random hue and saturation