# CS559: Computer Graphics

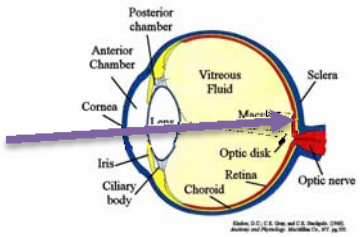
Lecture 6: Painterly Rendering and Edges

Li Zhang

Spring 2008

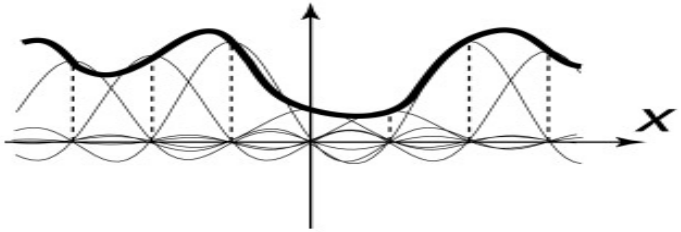
# So far

- Image formation: eyes and cameras

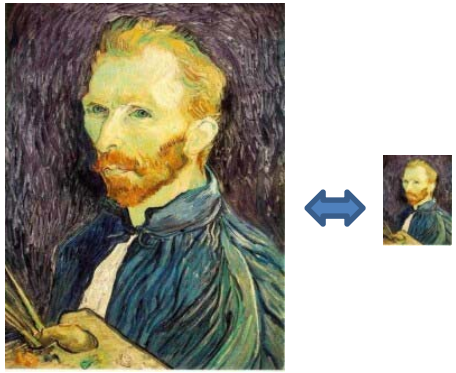


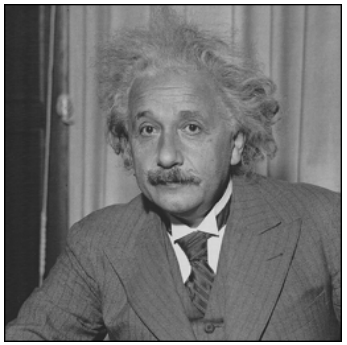


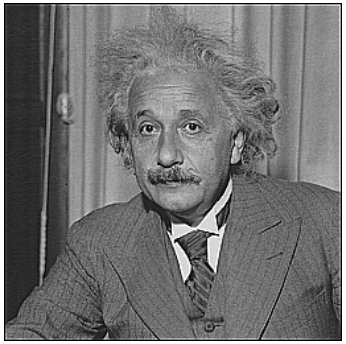
- Image sampling and reconstruction




- Image resampling and filtering







# Today

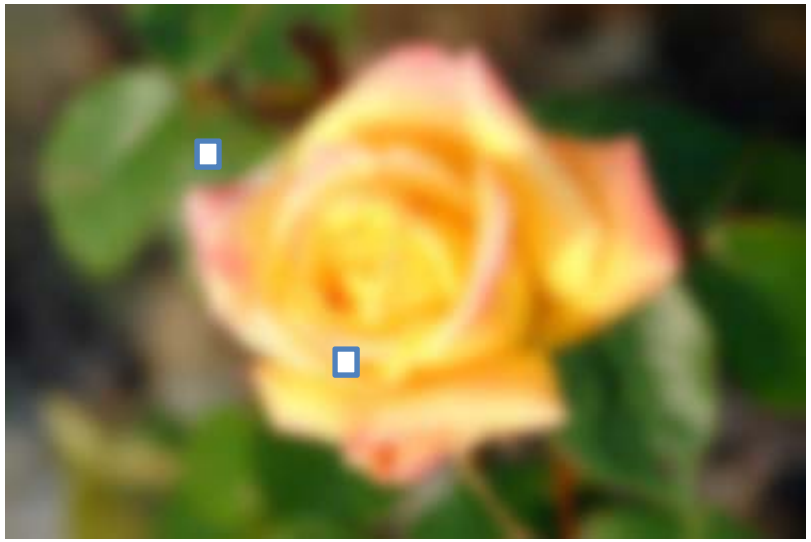
- Painterly rendering

- 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

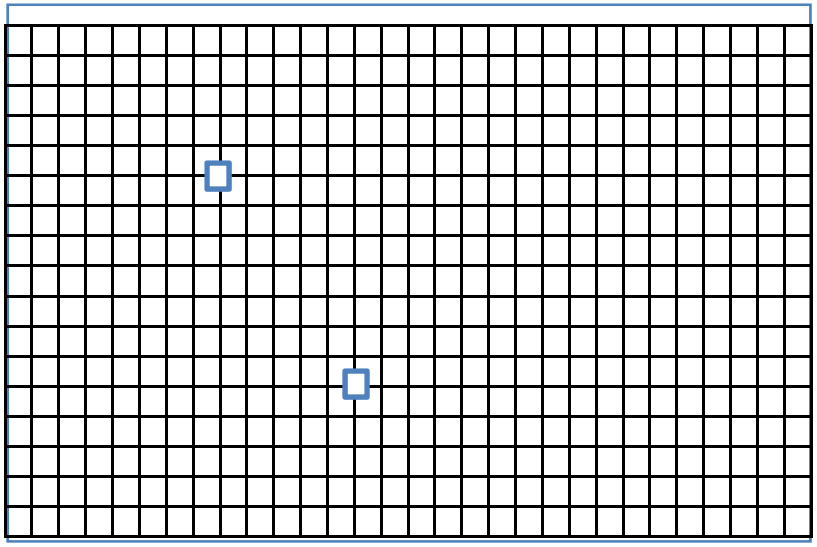
- Many methods have been proposed to make a photo look like a painting
  - A.k.a. Non-photorealistic Rendering
- Today we look at one: *Painterly-Rendering with Brushes of Multiple Sizes*
- Basic ideas:
  - Build painting one layer at a time, from biggest to smallest brushes
  - At each layer, add detail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

Input photo



Blurred input

Brush shape

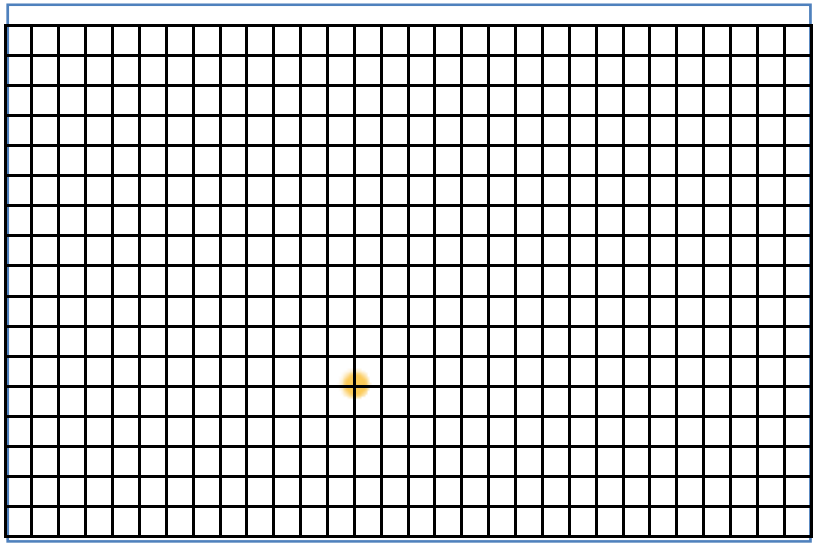


Canvas

Input photo

Blurred input

Brush shape

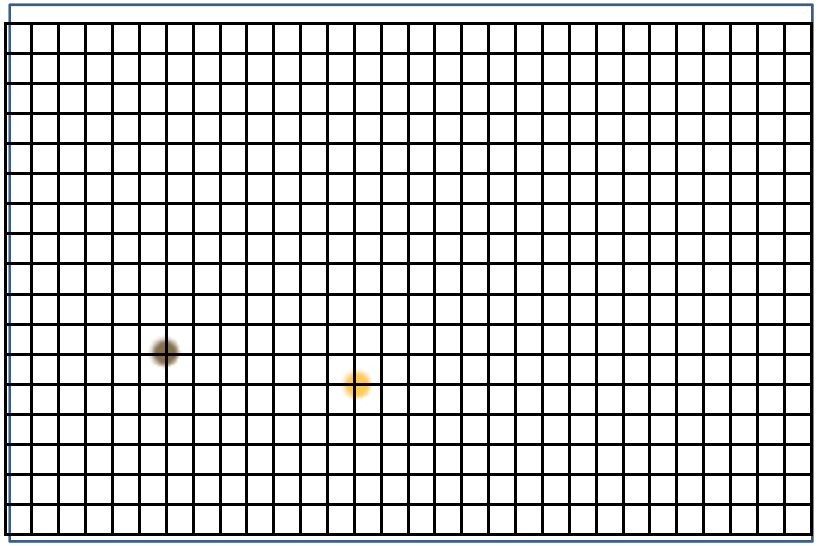


Canvas

Input photo

Blurred input

Brush shape



Canvas

Input photo

Blurred input

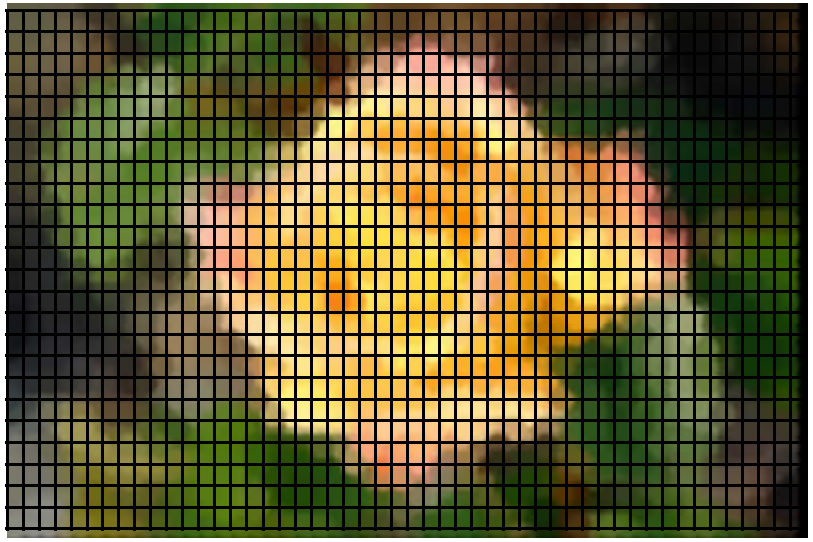
Brush shape

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

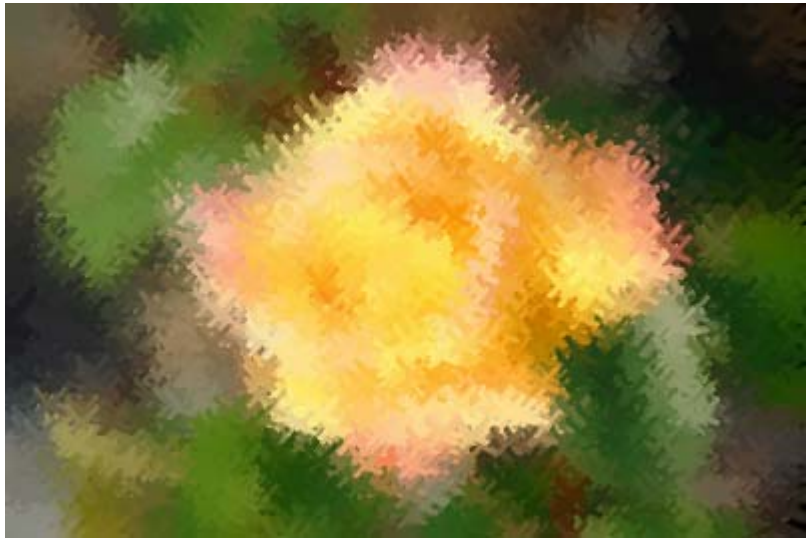
•

Brush shape

Canvas (3rd iteration)

Brush shape

Iteration 1



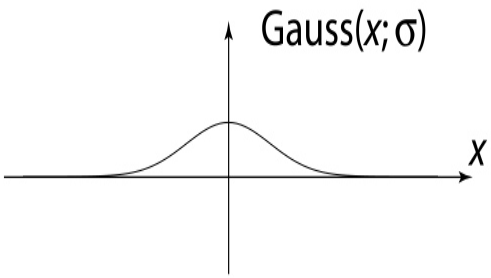
Iteration 2

Iteration 3

# How to blur an image?

- Continuous Gaussian Filter

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



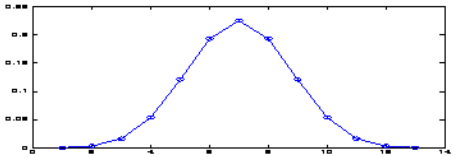

- Discrete Gaussian Filter

$$G_\sigma(i) = \frac{1}{Z} Gauss(i;\sigma),$$

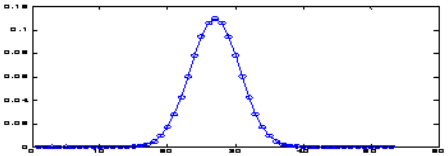
$$i \in [-N, N],$$

$$Z = \sum_{i=-N}^{N} Gauss(i;\sigma)$$

- 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 Filter Near Boundaries

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ? | 1 | 3 | 9 | 4 | 5 | 8 | 8 | 1 | 3 | 7 |

*

| | | |
|---|---|---|
| 0.25 | 0.5 | 0.25 |

=

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

- Zero padding
- Replication
- Reflection

# Image Filter Near Boundaries

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ? | 1 | 3 | 9 | 4 | 5 | 8 | 8 | 1 | 3 | 7 |

*

| | | |
|---|---|---|
| 0 | 0.5 | 0.25 |

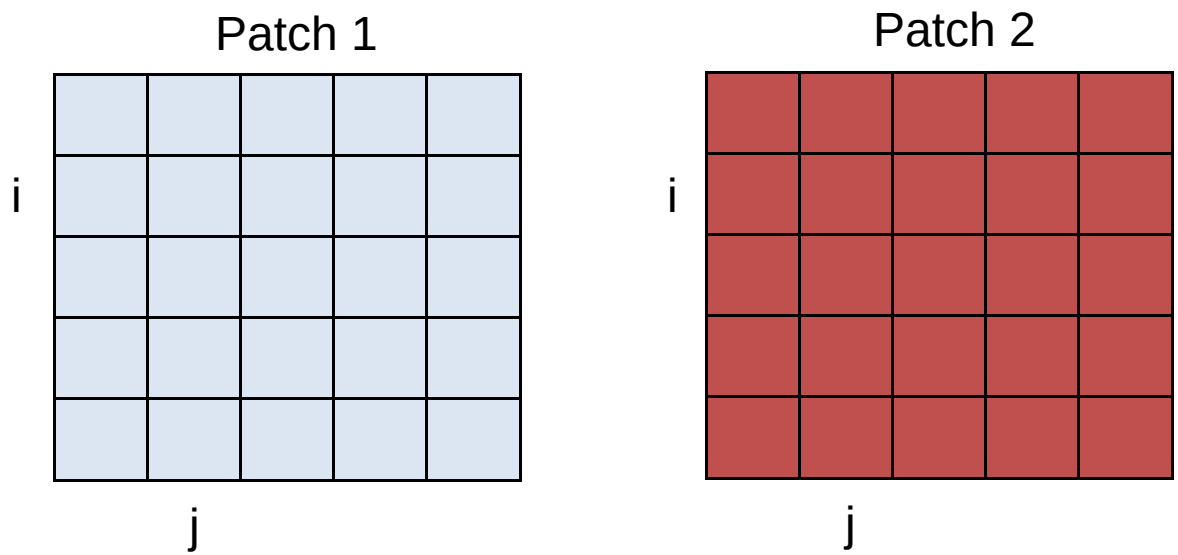
/ 0.75

=

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

- Zero padding
- Replication
- Reflection
- Kernel Renormalization

# 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, R_1 ... 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 R_i, from largest to smallest do
        {
                // Apply Gaussian smoothing with a filter of size f_σ R_i
                // Brush is intended to catch features at this scale
                referenceImage = sourceImage * G(f_σ R_i)
                // Paint a layer
                paintLayer(canvas, referenceImage, R_i)
        }
        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
}
```

# Results in the paper

Original

Biggest brush

Medium brush added

Finest brush added

# $f_\sigma$ and $f_g$

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

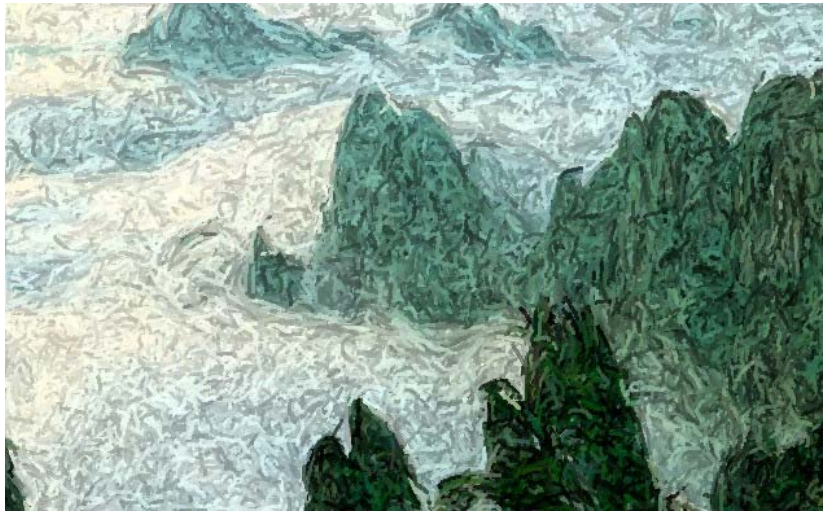
- Grid size = $f_g \cdot$ brush radius
  - Default $f_g = 1$

- Trying different parameters are optional

# 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

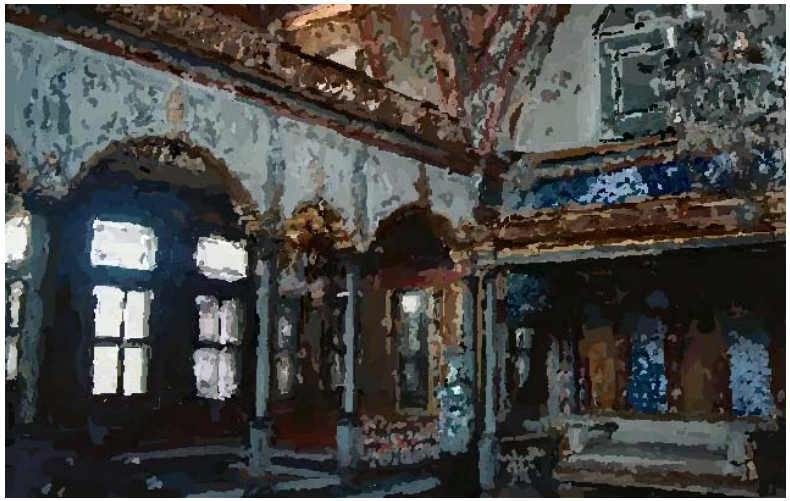
Expressionist, elongated stroke

Colorist wash, semitransparent stroke with color jitter

Densely-placed circles with random hue and saturation

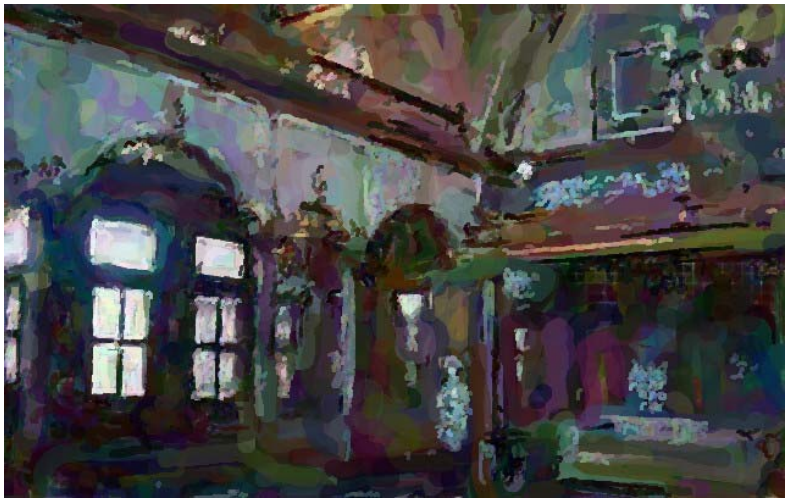
# Changing Parameters

# Changing Parameters



Impressionist, normal painting style

Expressionist, elongated stroke

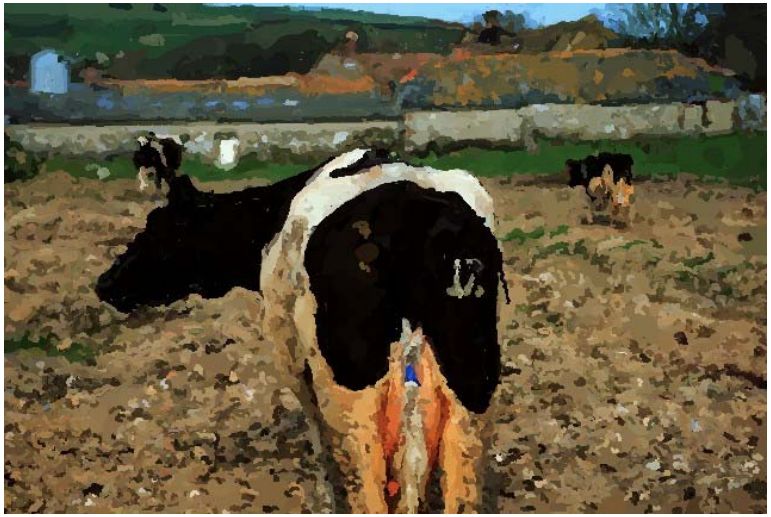


Colorist wash, semitransparent stroke with color jitter

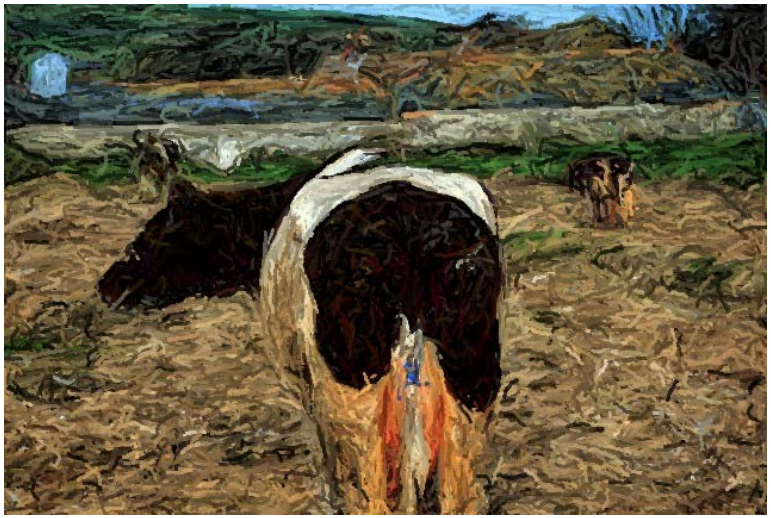
Densely-placed circles with random hue and saturation

# Changing Parameters

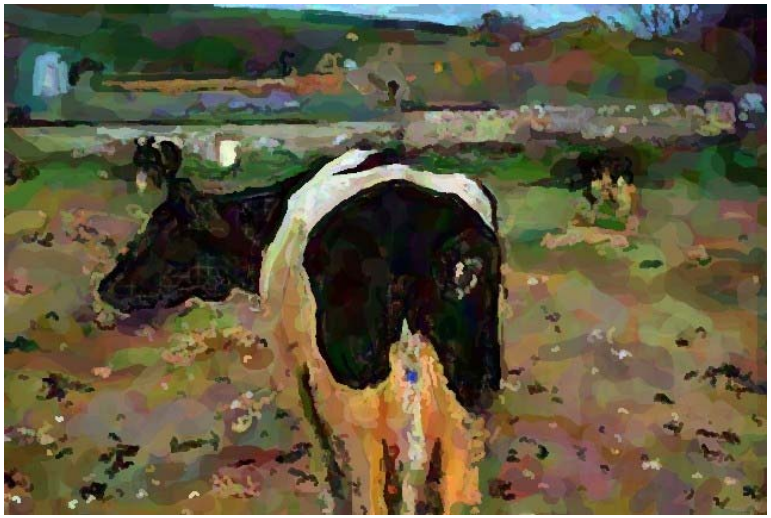
# Changing Parameters



Impressionist, normal painting style



Expressionist, elongated stroke



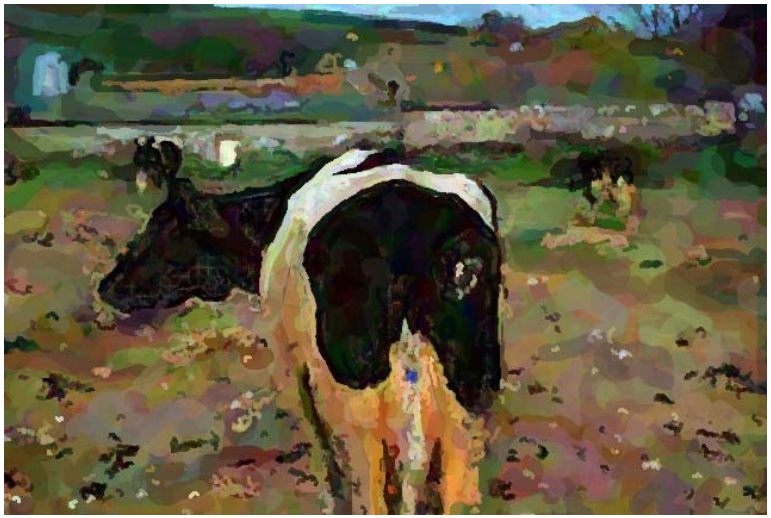
Colorist wash, semitransparent stroke with color jitter

Densely-placed circles with random hue and saturation

# Style Interpolation

Average style

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



Colorist wash, semitransparent stroke with color jitter

Densely-placed circles with random hue and saturation

# Another type of painterly rendering

- Line Drawing

http://www.cs.rutgers.edu/~decarlo/abstract.html

# Another type of painterly rendering

- Line Drawing

http://www.cs.rutgers.edu/~decarlo/abstract.html

# Another type of painterly rendering

- Line Drawing

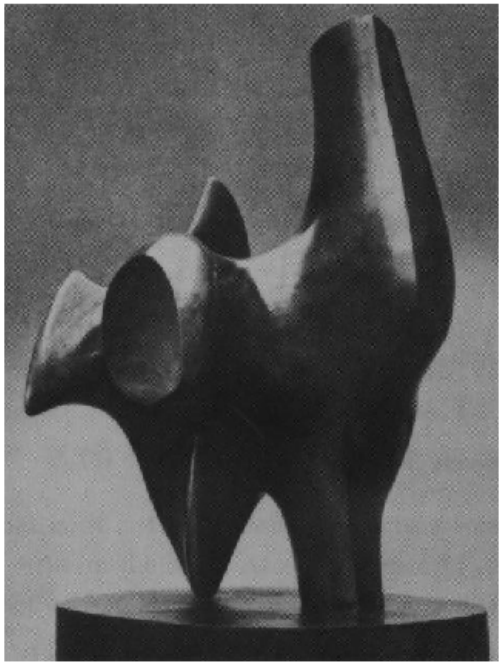

http://www.cs.rutgers.edu/~decarlo/abstract.html

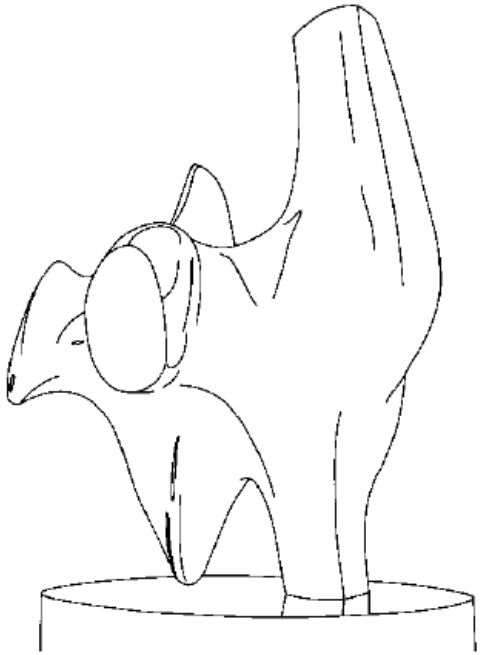
# Another type of painterly rendering

- Line Drawing

http://www.cs.rutgers.edu/~decarlo/abstract.html

# Edge Detection





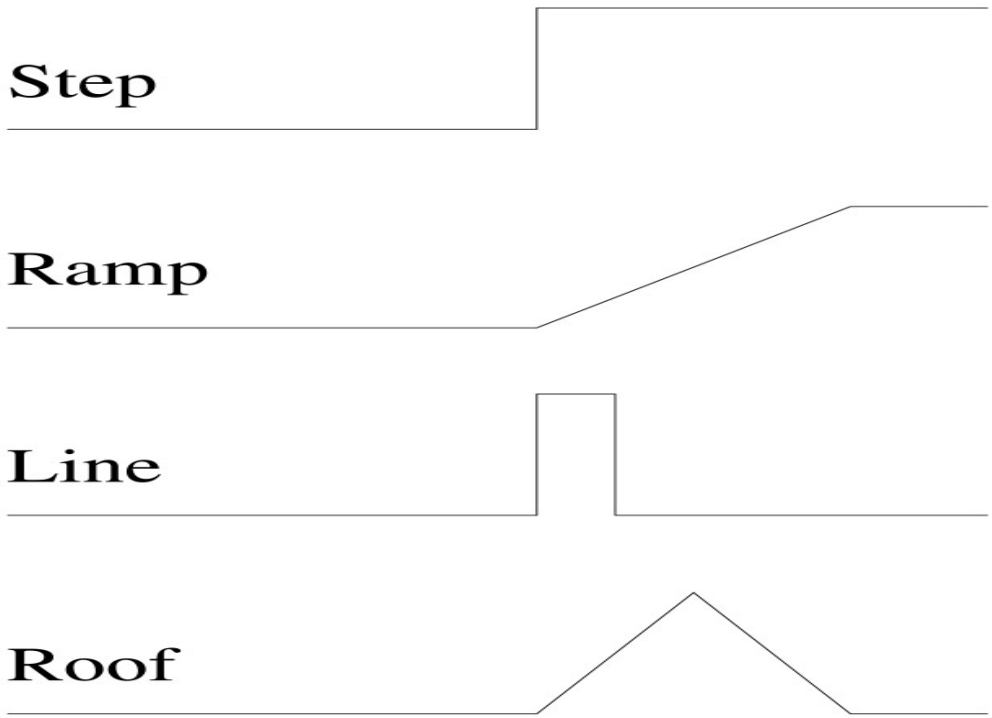
- Convert a 2D image into a set of curves
  - Extracts salient features of the scene

# Edge detection

- One of the most important uses of image processing is **edge detection:**
  - Really easy for humans
  - Really difficult for computers

  - Fundamental in computer vision
  - Important in many graphics applications

# What is an edge?




- **Q**: How might you detect an edge in 1D?

# Gradients

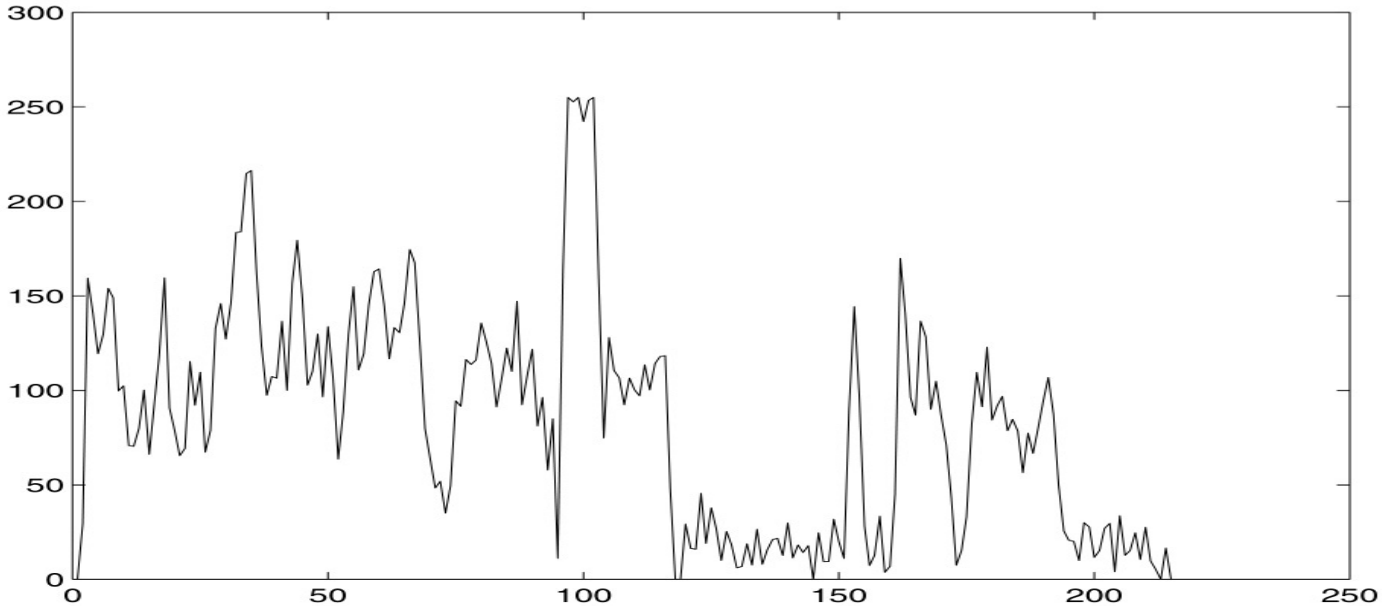
- The **gradient** is the 2D equivalent of the derivative: $\nabla f(x, y) = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$

  gx[i,j] = f[i+1,j] – f[i,j] and gy[i,j]=f[i,j+1]-f[i,j]
  Can write as mask [-1 1] and [1 –1]'

- Properties of the gradient
  - It's a vector
  - Points in the direction of maximum increase of $f$
  - Magnitude is rate of increase
- How can we approximate the gradient in a discrete image?

# Less than ideal edges

Pixels plotted



# Results of Sobel edge detection

Original

Smoothed

# Edge enhancement

- A popular gradient magnitude computation is the **Sobel operator**:

$$s_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$s_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

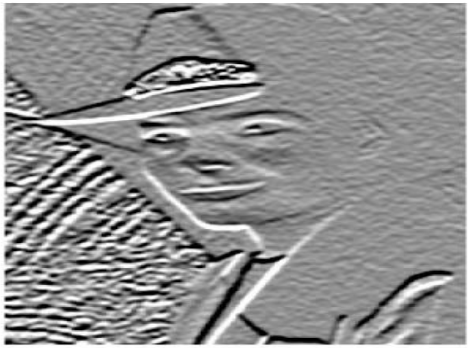
- We can then compute the magnitude of the vector $(s_x, s_y)$.

# Results of Sobel edge detection

Original

Smoothed

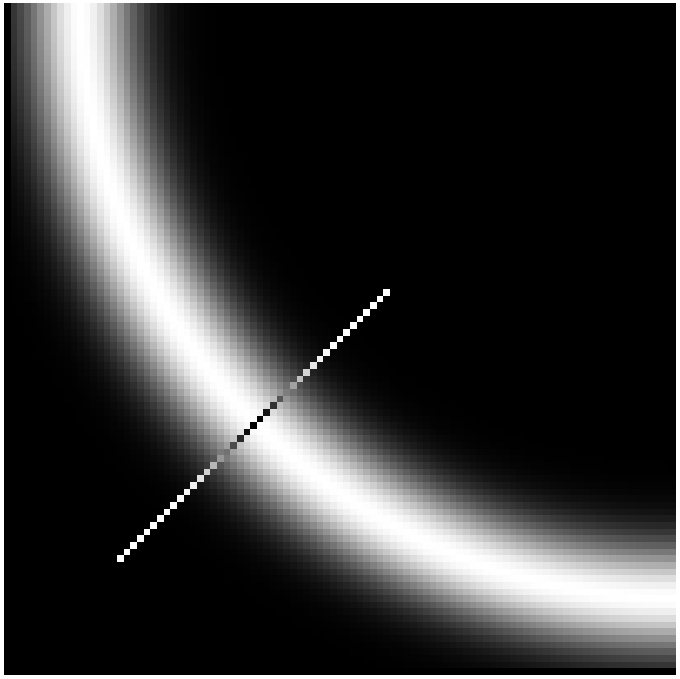
Sx + 128

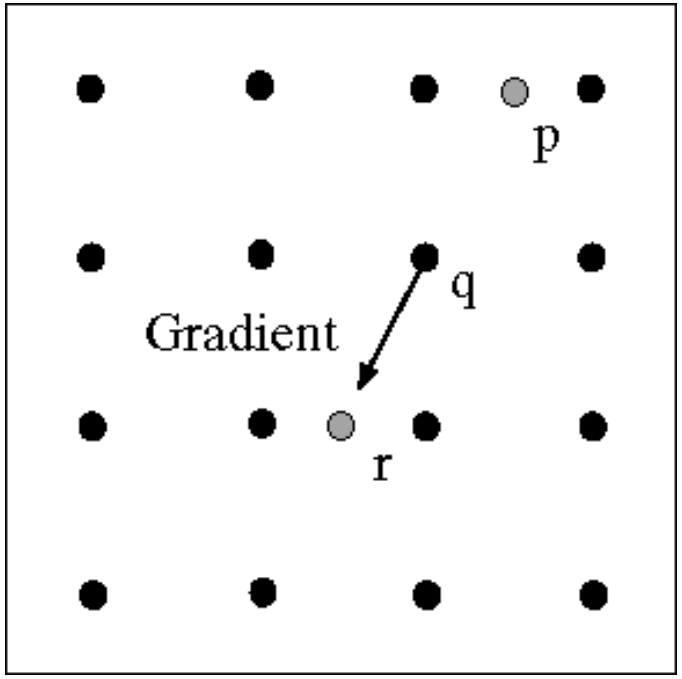


Sy + 128

Magnitude

# Non-maximum Suppression





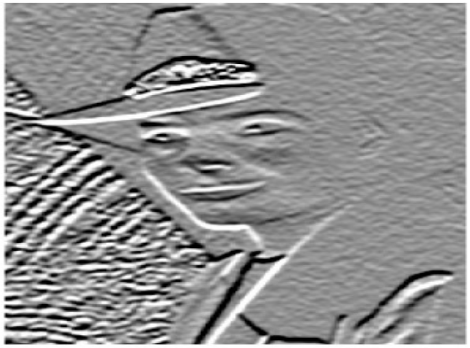

- Check if pixel is local maximum along gradient direction
  - requires checking interpolated pixels p and r

# Results of Sobel edge detection

Original

Smoothed

Sx + 128



Sy + 128

Magnitude

Threshold = 64

Threshold = 128

# Steps in edge detection

- Edge detection algorithms typically proceed in three or four steps:
  - **Filtering**: cut down on noise
  - **Enhancement**: amplify the difference between edges and non-edges
  - **Detection**: use a threshold operation
  - **Localization** (optional): estimate geometry of edges, which generally pass between pixels

# The Canny Edge Detector

original image (Lena)

# The Canny Edge Detector

magnitude of the gradient

# The Canny Edge Detector

After non-maximum suppression

# Canny Edge Detector

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

$\sigma$ : Gaussian filter parameter

- The choice of $\sigma$ depends on desired behavior
  - large $\sigma$ detects large scale edges
  - small $\sigma$ detects fine features