Universität Konstanz

Illustrative Computer Graphics

Week 5

Color I

Lecturers:
Oliver Deussen
KC Kwan

Homework Done?

- Sketch 5

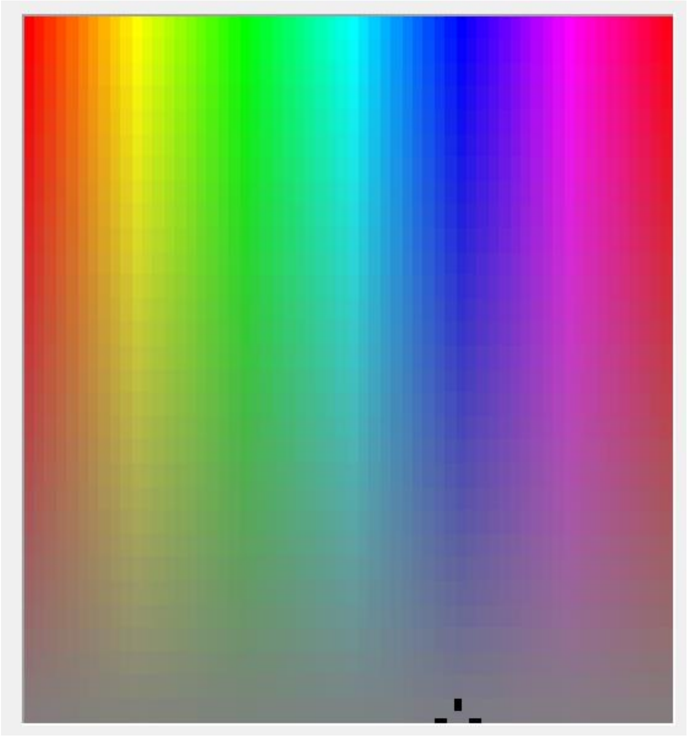
Color!

Chapter 5.1 – Color Quantization

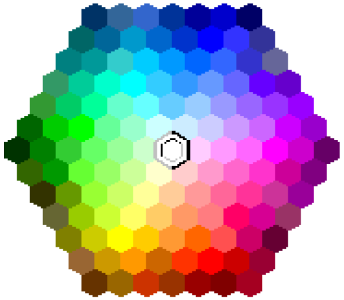
slides in part adapted from Stefan Schlechtweg

Color Quantization

- Represent images with less number of colors



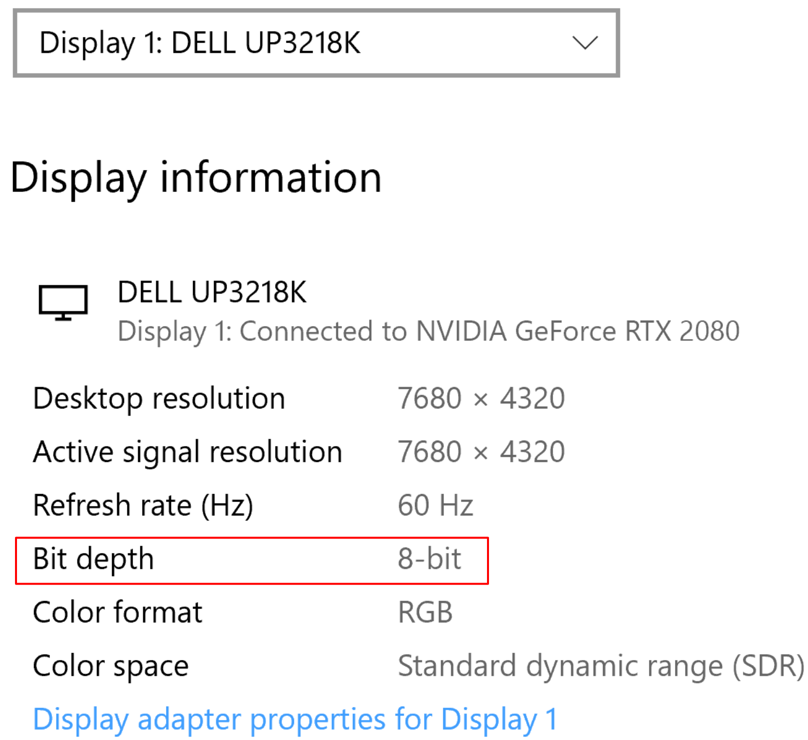
16,777,216 Colors



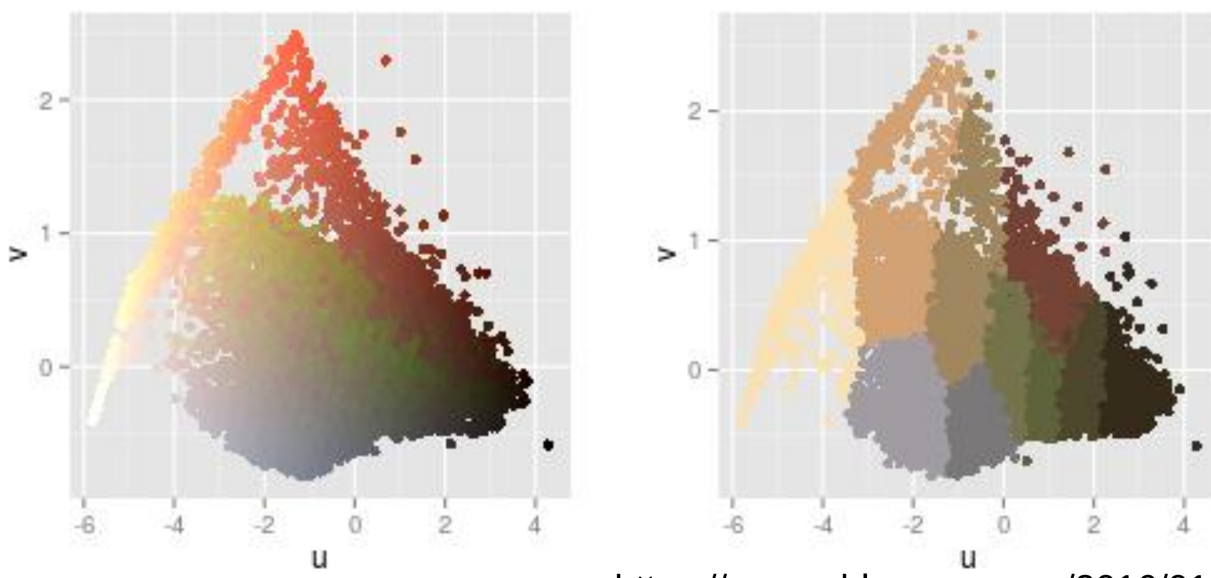
127 Colors

Why?

- Compress the color image
- Some displays represent less color
  - 6-bit or 8-bit or 10-bit
- We can have 32-bit per channel images!



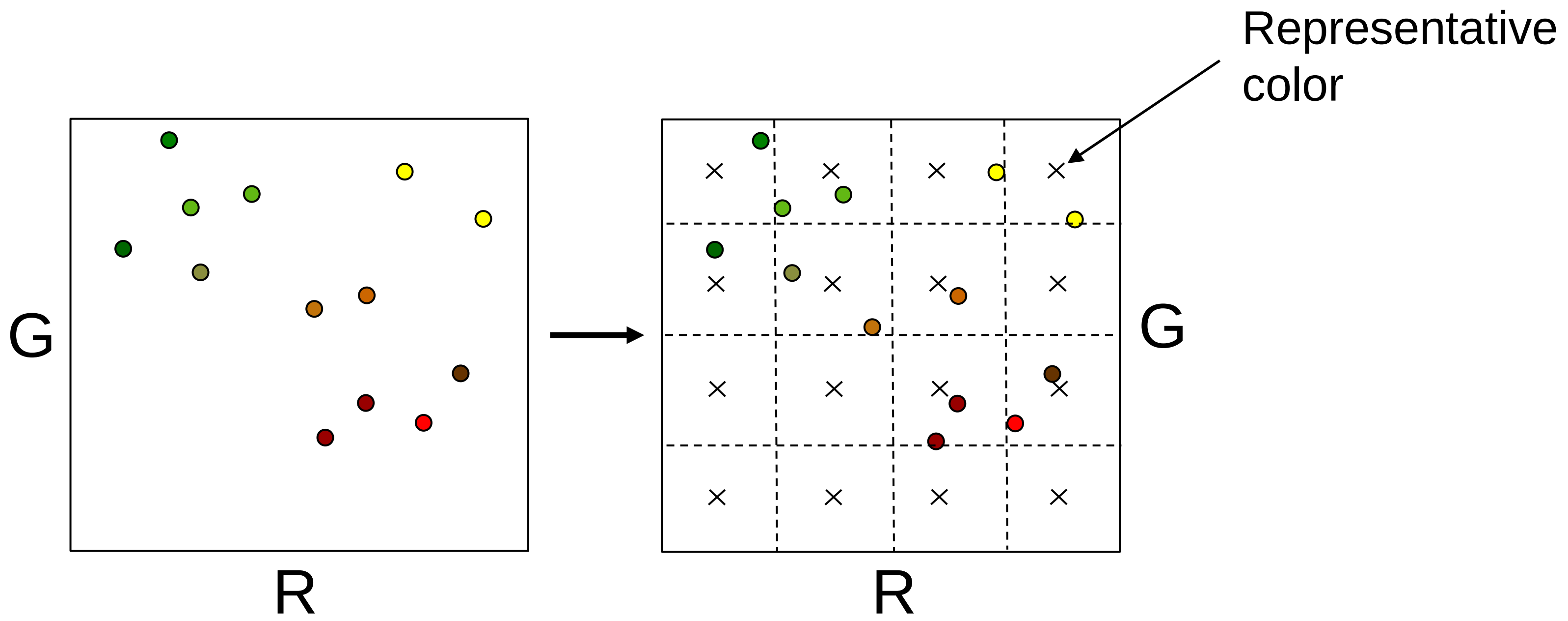
Color Quantization



https://www.r-bloggers.com/2016/01/color-quantization-in-r/

Uniform Quantization

- Break the color space into uniform cells
- Map each color to the center in its cell



Color space (2D)

Uniform Quantization

- Equivalent to dividing each color by some number and taking the integer part

8-bit	8-bit	8-bit
3-bit	3-bit	2-bit

R	G	B
256	256	256

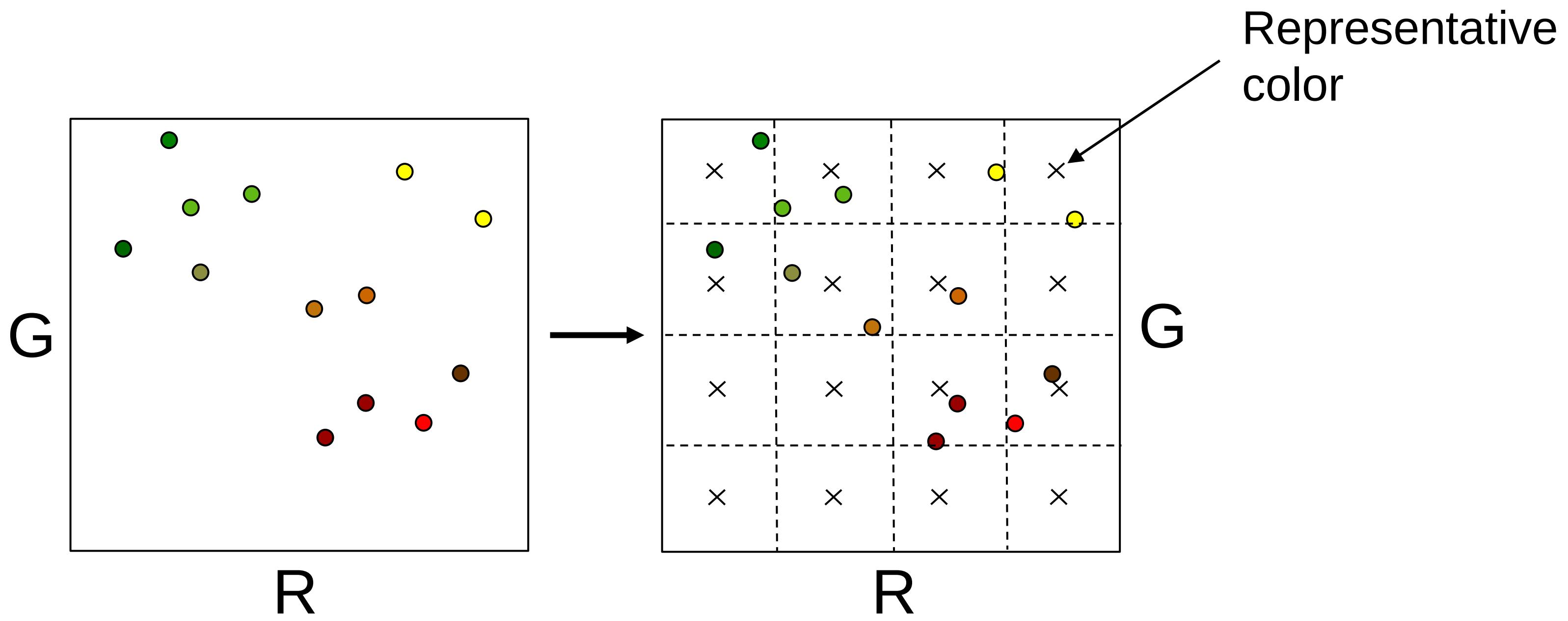
R’	G’	B’
8	8	4

$$R', G', B' = \text{floor}(R\,\frac{8}{256}),\ \text{floor}(\mathrm{G}\,\frac{8}{256}),\ \text{floor}(\mathrm{B}\,\frac{4}{256})$$

RGB(1,1,1) != R’G’B’(1,1,1)

Uniform Quantization

- Problem:
  - Fails to capture the distribution of colors
  - Empty cells are wasted



Color space (2D)

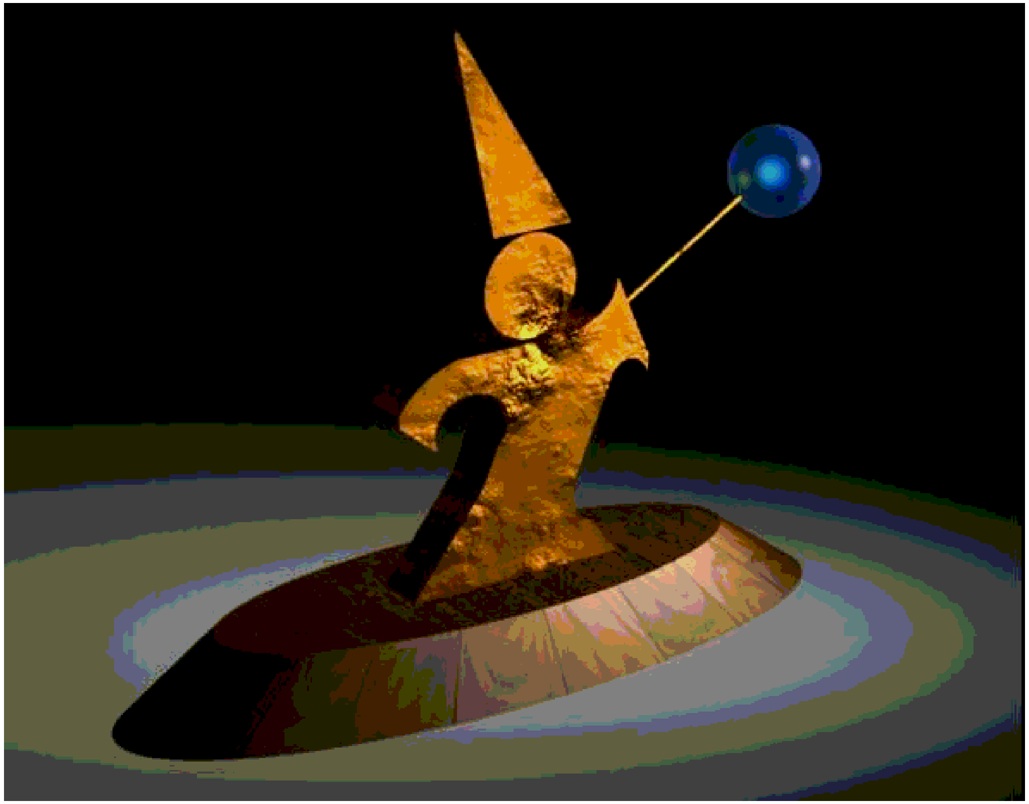
Example (24 bit color)

Uniform Quantization

8-bit color (3-3-2)

- Poor gradients
- Colors are wrong

Improve use information from the image?



Color Quantization

Sub problems:

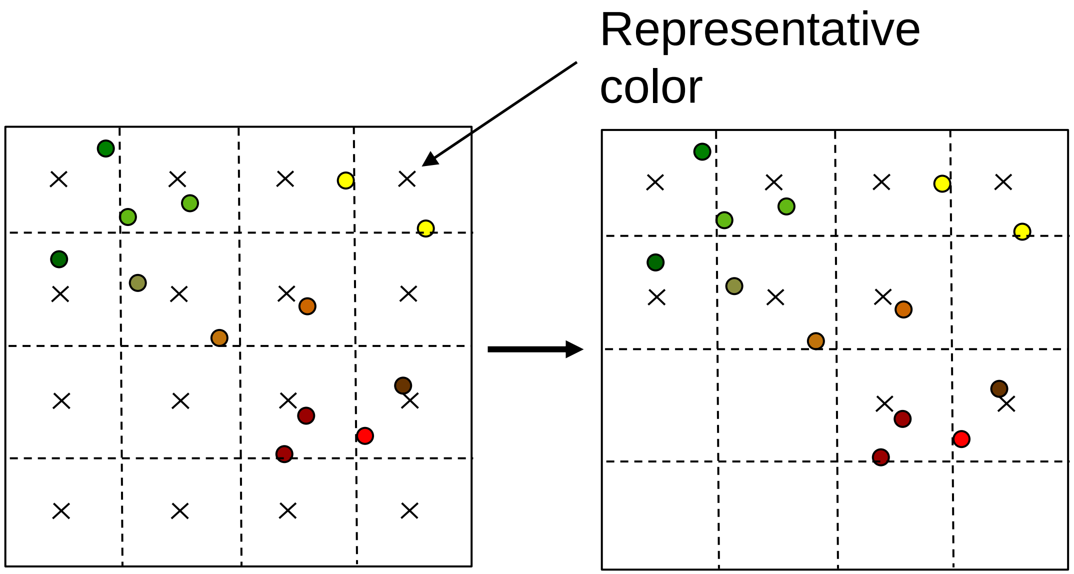
- Which colors to use?
- How to map these colors?

Chapter 5.2 – Colors Selection

slides in part adapted from Stefan Schlechtweg

Empty Cells

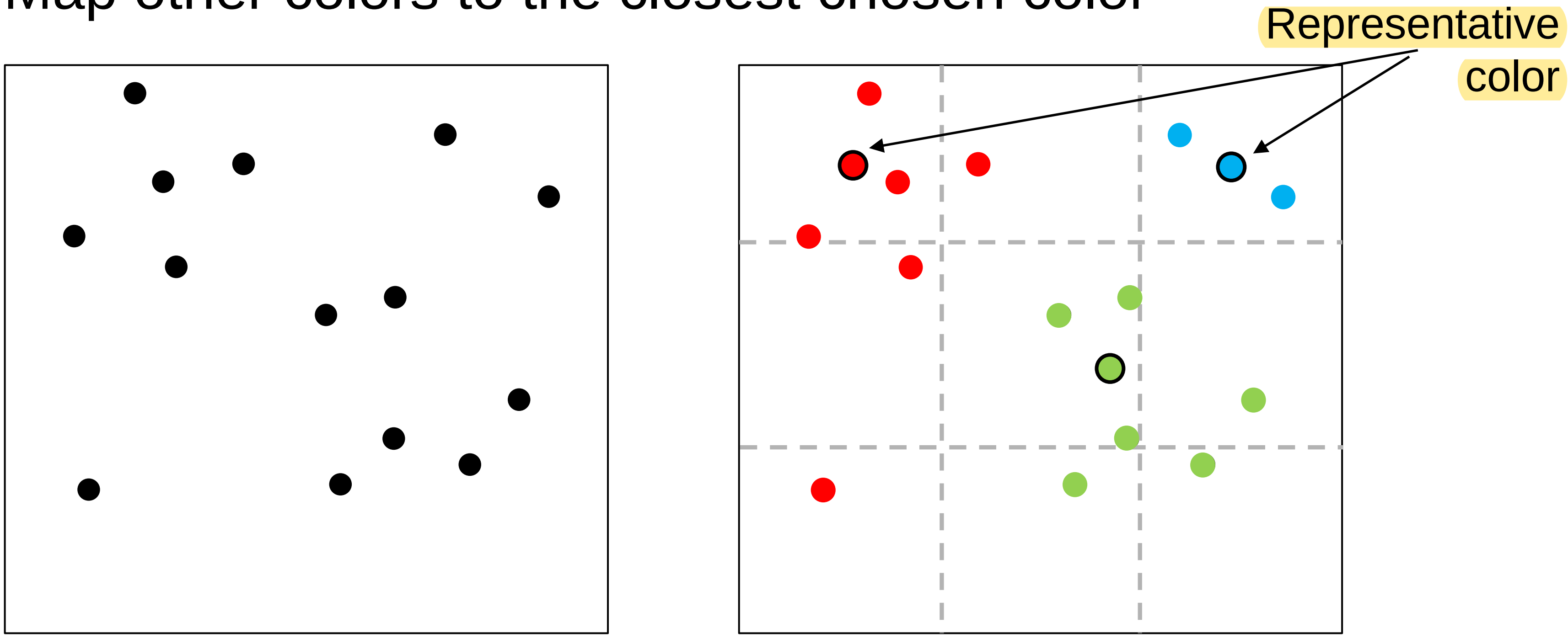
Can we only use some representative colors?



Color space (2D)

Populosity (Popularity) Algorithm

- Color histogram: count the number of sample in each cell
- Choose the *n* most commonly occurring cells
- Use the average of colors in each selected cell
- Map other colors to the closest chosen color

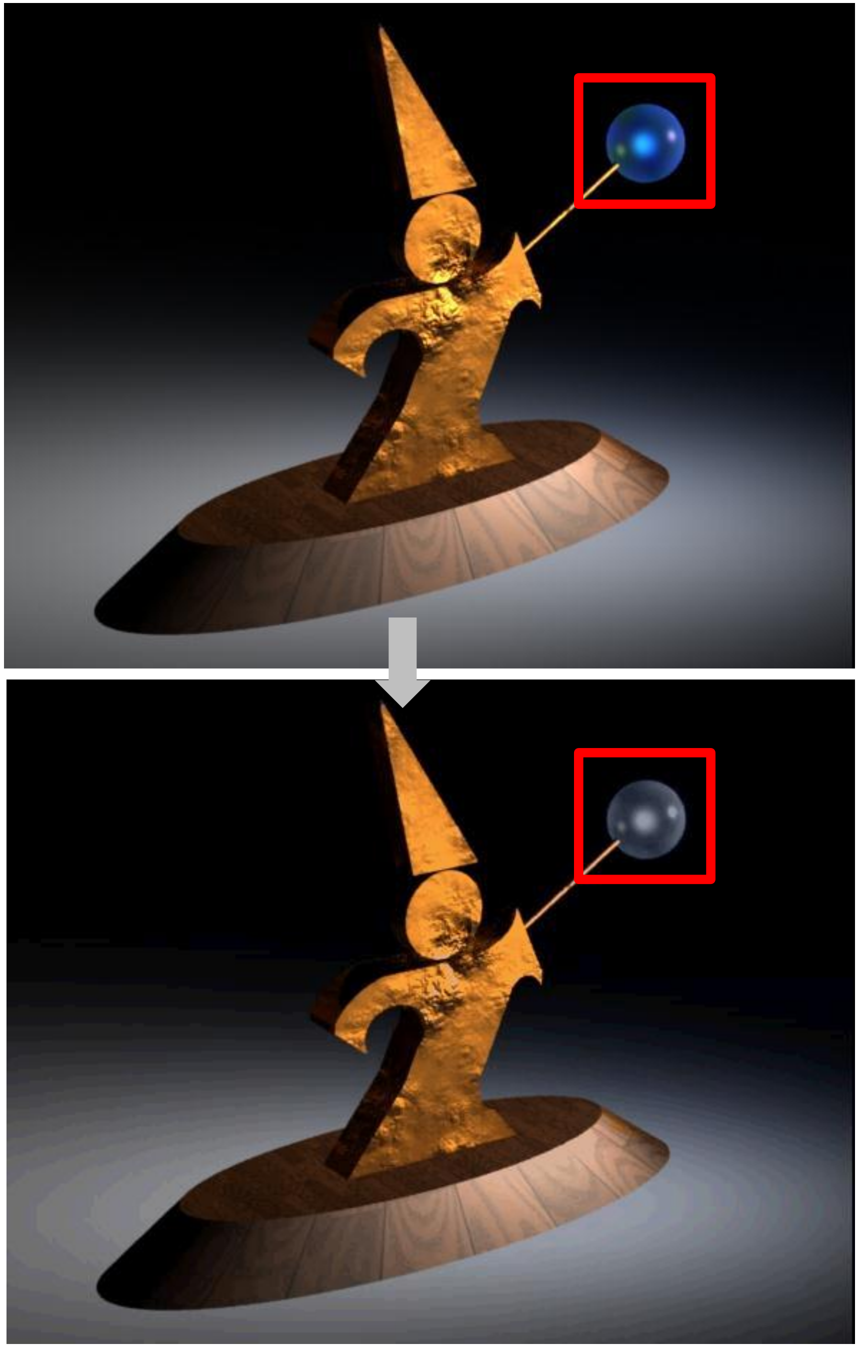


Populosity (Popularity) Algorithm

- Color histogram: count the number of sample in each cell
- Choose the *n* most commonly occurring cells
- Use the average of colors in each selected cell
- Map other colors to the closest chosen color

Populosity Algorithm

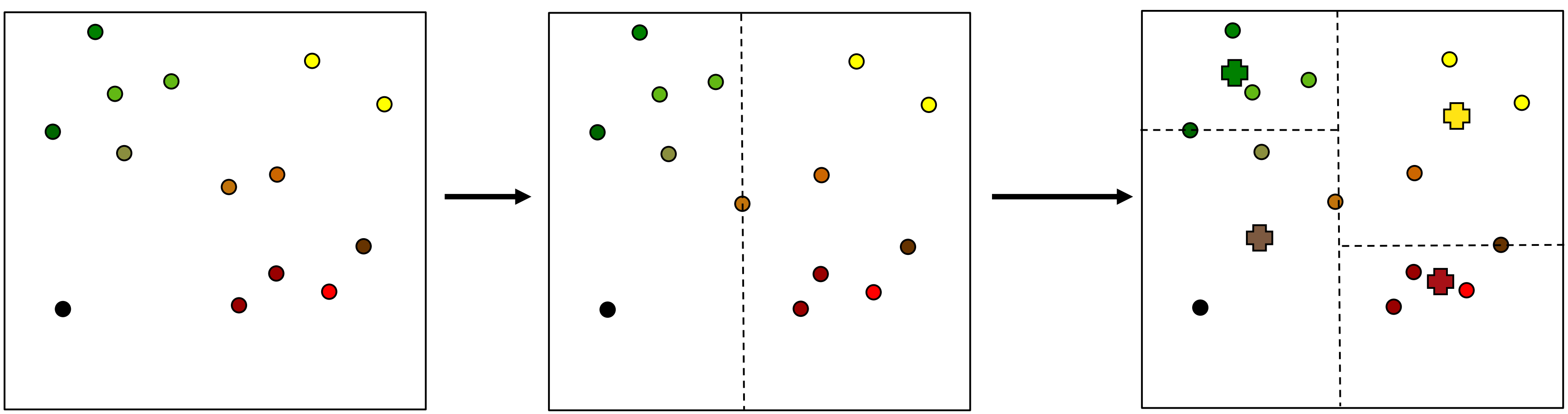
- 24-bit to 8-bit
- Blue is not popular
  - Missing in image
- Populosity ignores rare but important colors!




A. Hofstetter
photography

Median Cut

- Cut the colors sample into two half's
  - From the longest dimensions
  - Cut at median
- Create representative color by averaging



Samples in 2D color space

Recursively Median cut

Cut and create representative color

Median Cut

- Cut the colors sample into two halves
  - From the longest dimensions
  - Each side has equal / similar number of colors
- Create representative color by averaging

- Similar algorithm to build kD-tree
  - A common spatial data structure
  - For fast neighborhood search
  - Useful in many other areas of CG

Median Cut

- We have blue now!

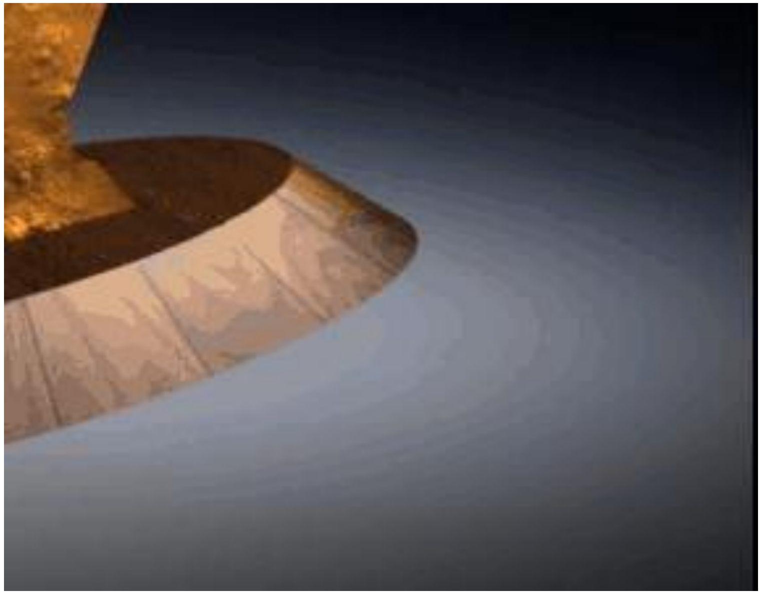
Optimization Algorithms

- Find a set of colors with lowest quantization error
  - Many way to do so

- Not very popular
  - Expensive
  - Unless the number of colors is small

Perceptual Problems

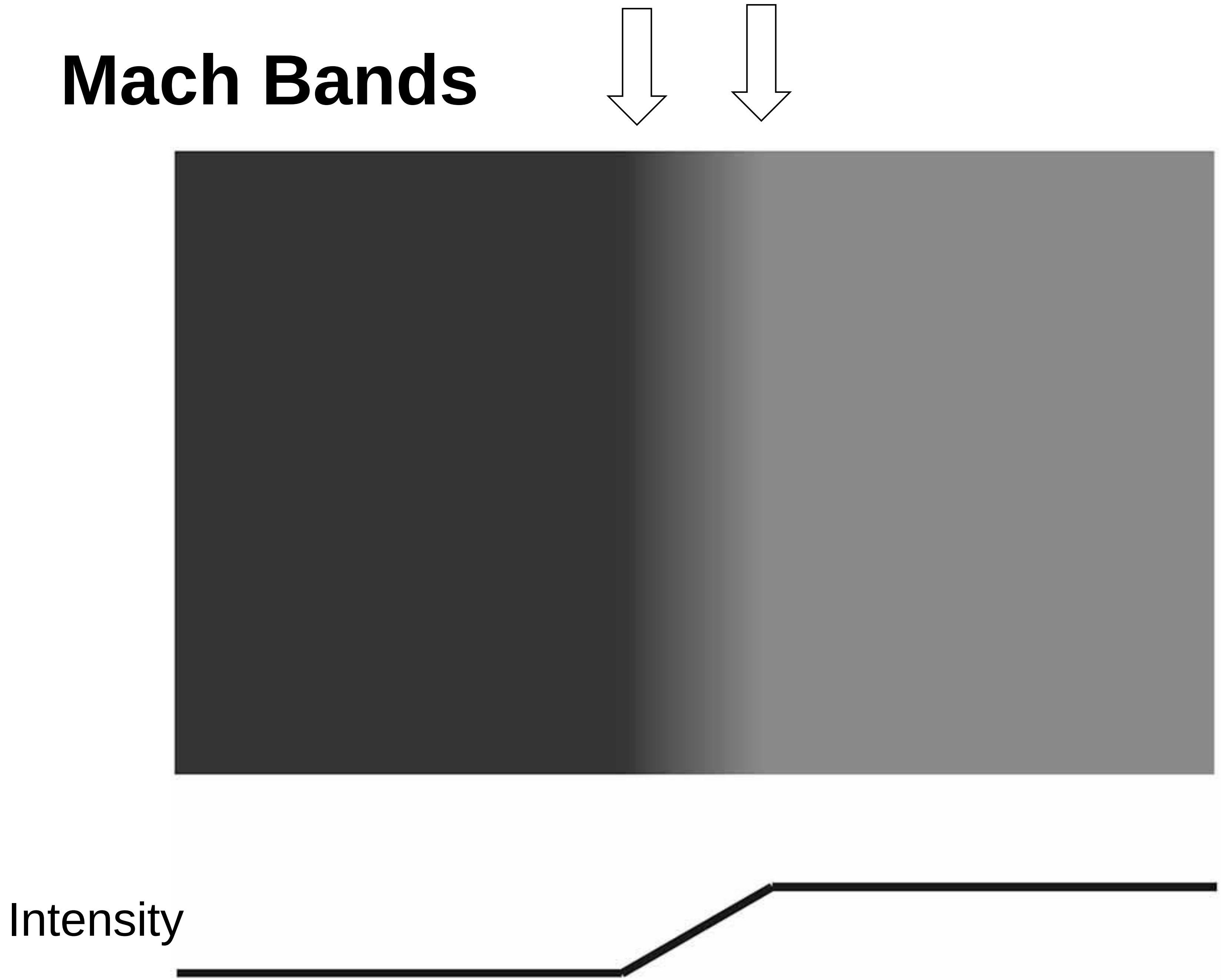
- Humans still perceive the quantization even if the colors are close.



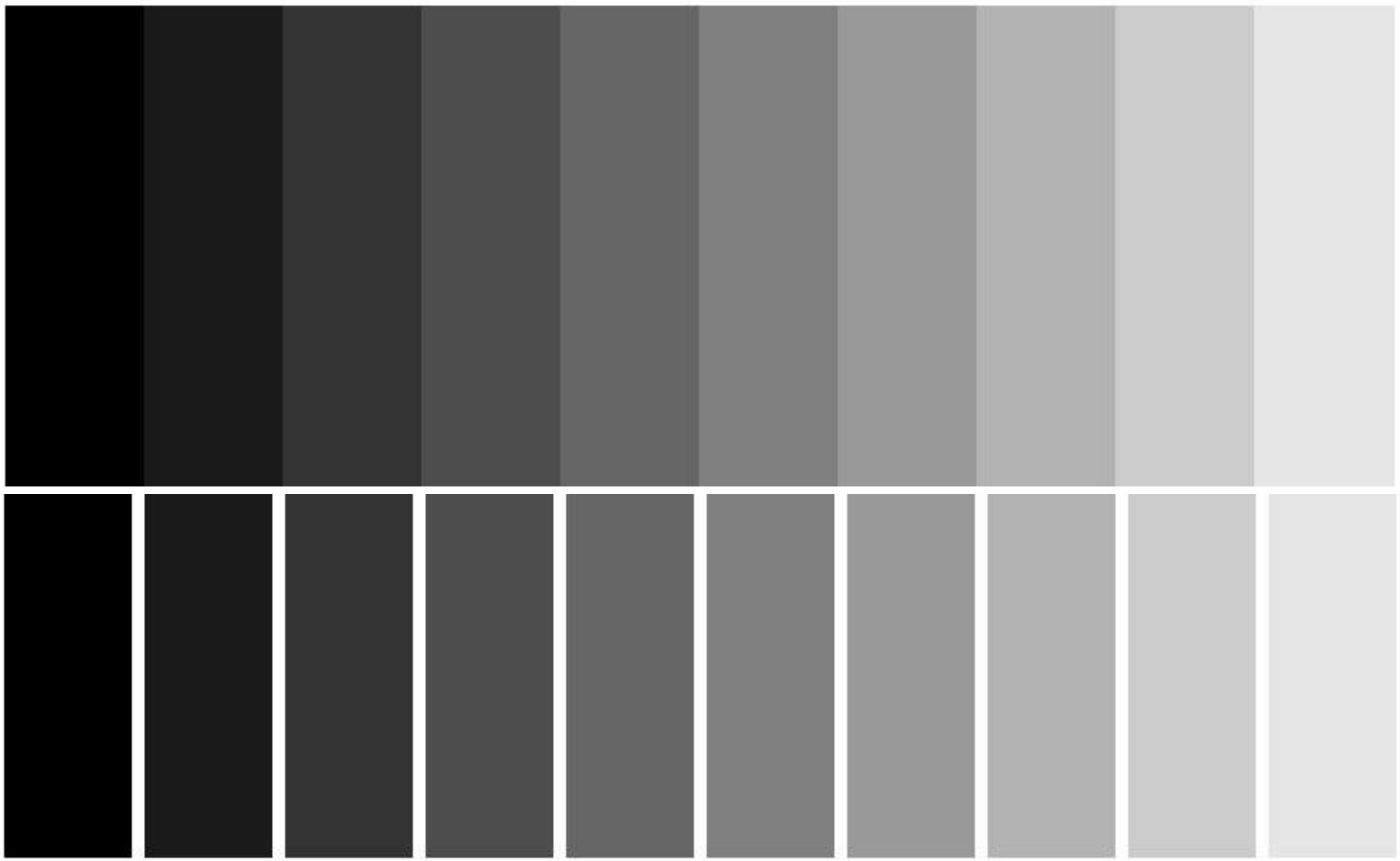
- **Mach bands**
  - Optical illusion
  - Emphasizes boundaries

From: Mach bands explained by response normalization

Mach Bands



Mach Bands



Mach Bands in Reality

Color Dithering

Dithering:

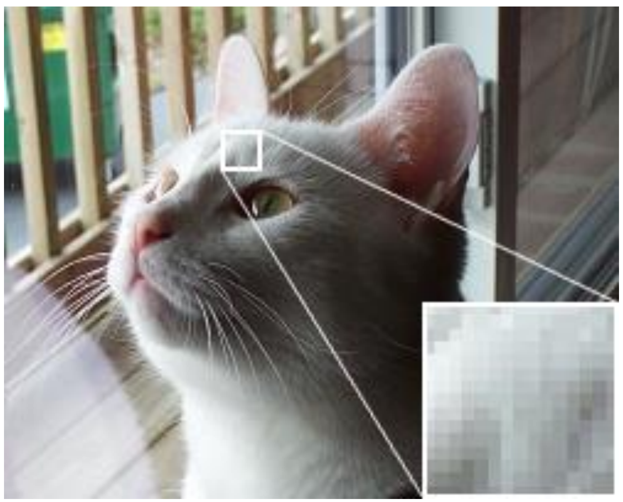
Randomly map to different colors

Error diffusion:

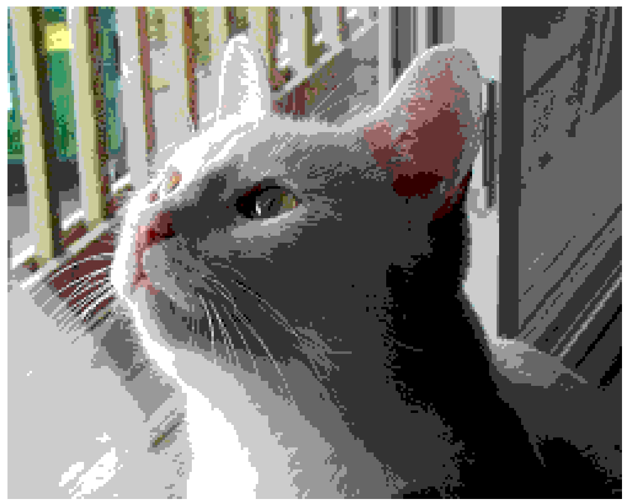
propagate the color error to the nearest

More Result

Original



256 Colors



256 Colors + ED

Wikipedia.org

Chapter 5.3 – Painterly Rendering

Painting

Filter-based Painterly Rendering

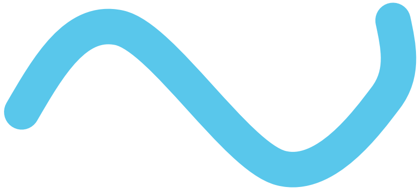
- Edge-preserving filter (e.g., bilateral filter)

Filter-based Painterly Rendering

- Oil Painting:
  – Colors are flat / smooth
  – Strong edges
- Smoothing / averaging the prophage surrounding pixels:
  – with small color different
- No stroke elements

Hertzmann‘s Algorithm

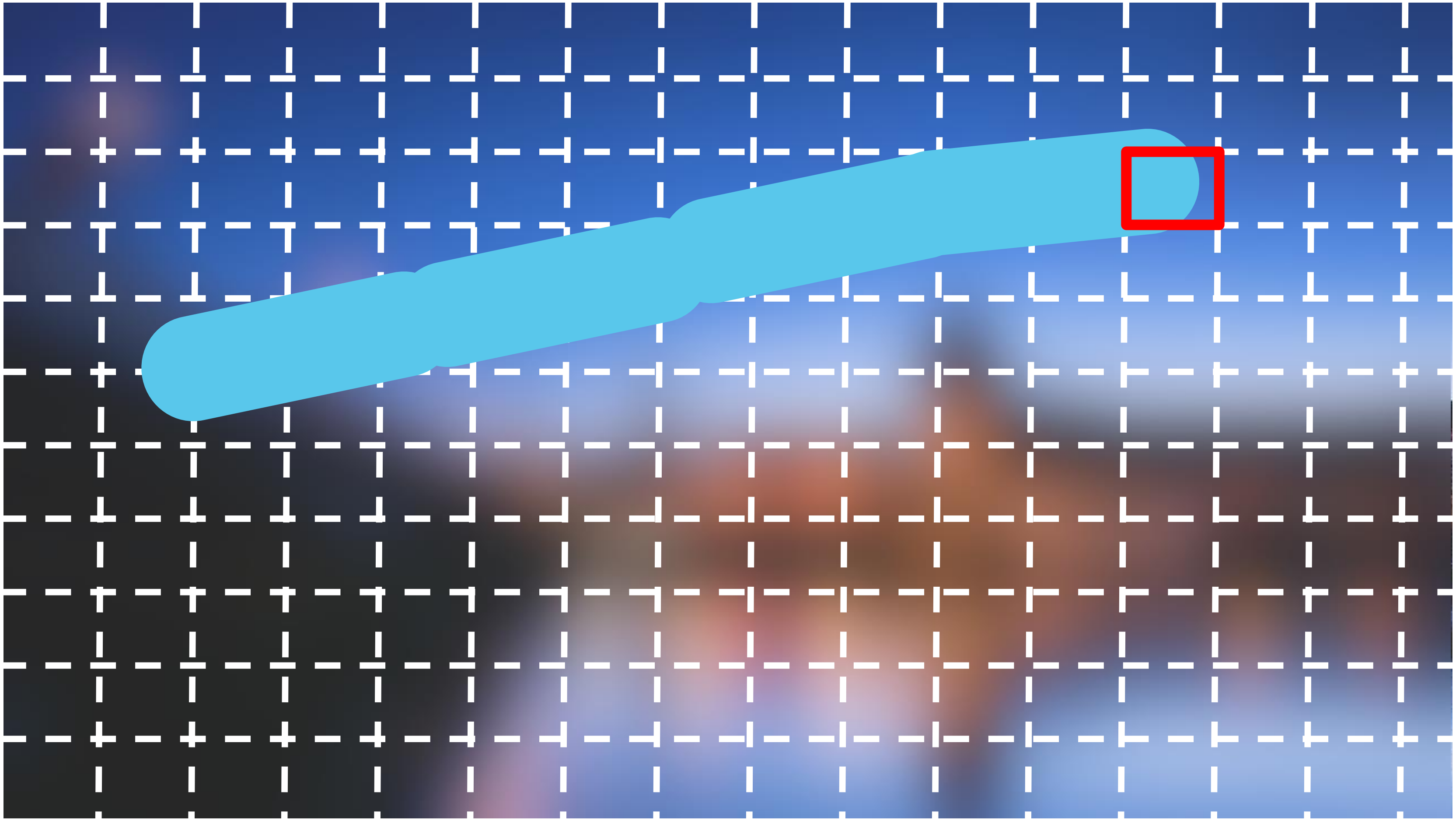
- Approximate an image with a number of strokes
  - Strokes with uniform colors
  - Simple form (line with circular caps)



- Starts with large width stroke first

- Add smaller strokes where it is not well approximated

Drawing One Stroke



Drawing One Stroke

- Start from a pixel with largest error
- Add a line with fixed length *l*
- Repeat until some stopping conditions
- The orientation is based on the image gradient

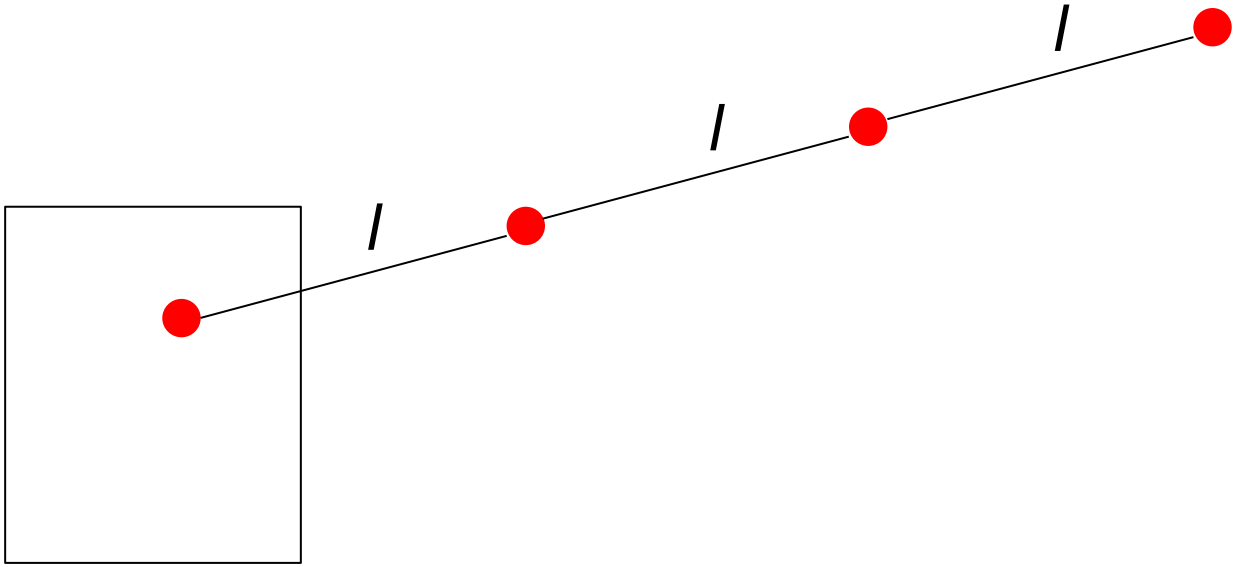
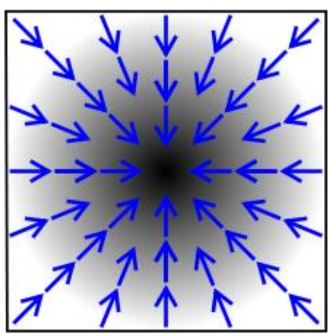
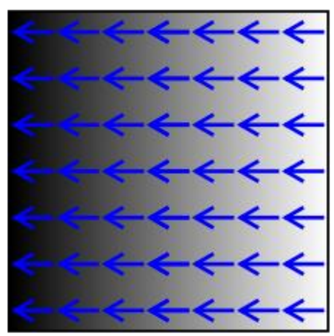
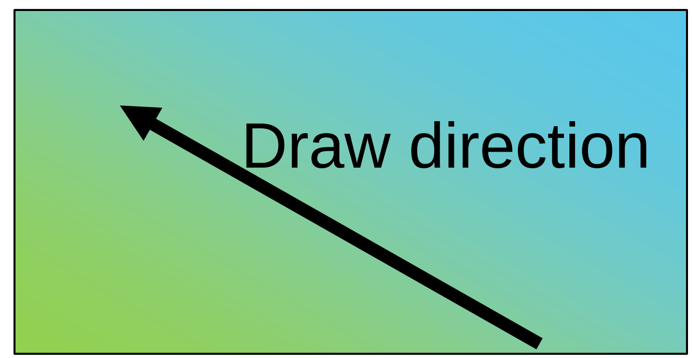


Image Gradient

- Image gradient describes the directional change of intensity
  - Derivative in horizontal and vertical directions
  - Approximation by **Sobel operator**
- Draw along the **normal** of image gradient







$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

$$x' = x\ cos\theta - y\ sin\theta$$
$$y' = x\ sin\theta + y\ cos\theta$$

Hertzmann‘s Algorithm

For each brush size from large to small:

- Blur target image
- Compute image color difference
- Check for each cell, if average error is large, create a new stroke:

  Starting point: the pixel with largest error

  Orientation: perpendicular to the image gradient

  Termination: Until color difference too big or max lenght reached
- Draw the strokes in random order

Further Details:

For each brush size from large to small:

- Blur target image
- **Compute image color difference**

$$ColorDiff = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

Further details:

For each brush size from large to small:

- ...
- **If average error is large, create a new stroke**

  Starting point: the pixel with largest error

  Orientation: perpendicular to the image gradient

  Termination: Until color difference too big or max lenght reached

- Stroke have constant thickness and uniform color
- Multiple straight lines with same length
- Note: create only, not draw

Further details:

For each brush size from large to small:

- ...
- **Draw the strokes in random order**

Usage of Z-buffer

- Each stroke gets a z-coordinate
- Randomizing a large list of brush strokes
- Avoids regular pattern

Painting with Three Brushes

(a)

(b)

(c)

(d)

Aaron Hertzmann: Painterly Rendering with Curved Brush Strokes of Multiple Sizes, Siggraph 1998

```
function makeSplineStroke(x0,y0,R,refImage)
{
   strokeColor = refImage.color(x0,y0)
   K = a new stroke with radius R
          and color strokeColor
   add point (x0,y0) to K
   (x,y) := (x0,y0)
   (lastDx,lastDy) := (0,0)

   for i=1 to maxStrokeLength do
   {
      if (i > minStrokeLength and
        |refImage.color(x,y)-canvas.color(x,y)|<
        |refImage.color(x,y)-strokeColor|) then
          return K

      // detect vanishing gradient
      if (refImage.gradientMag(x,y) == 0) then
       return K

      // get unit vector of gradient
      (gx,gy) := refImage.gradientDirection(x,y)
      // compute a normal direction
      (dx,dy) := (-gy, gx)

      // if necessary, reverse direction
      if (lastDx * dx + lastDy * dy < 0) then
        (dx,dy) := (-dx, -dy)

      // filter the stroke direction
      (dx,dy) :=fc*(dx,dy)+(1-fc)*(lastDx,lastDy)
      (dx,dy) := (dx,dy)/(dx^2 + dy^2)^1/2
      (x,y) := (x+R*dx, y+R*dy)
      (lastDx,lastDy) := (dx,dy)

      add the point (x,y) to K
   }
   return K
}
```

Style Parameters

- *Approximation threshold (T)* When to add strokes

- *Brush sizes* $(R_1, \ldots, R_n)$
- *Grid size* (f_g)

Output size

- *Curvature Filter* (f_c)
- *Blur Factor*(f_σ)
- *Minimum and maximum stroke lengths*
- *Opacity* (α)

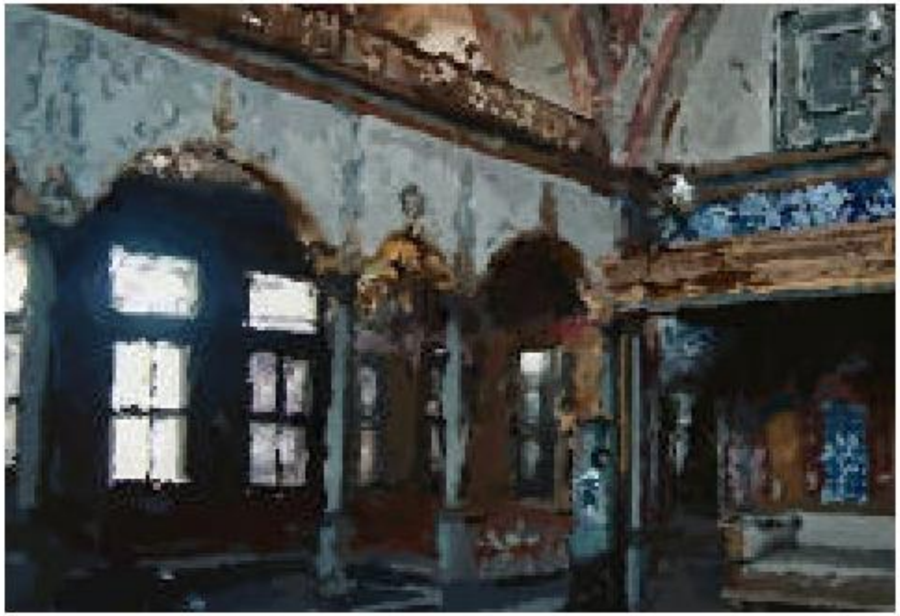
Stroke style

Rendering Style

Styles are defined as follows (in his paper!)

- **"Impressionist"** — A normal painting style, with no curvature filter, and no random color.
- **"Expressionist"** — Elongated brush strokes. Jitter is added to color value.
- **"Colorist Wash" —** Loose, semi-transparent brush strokes. Random jitter is added to R, G, and B color components.
- **"Pointillist"** — Densely-placed circles with random hue and saturation.

“Impressionist“ paintings



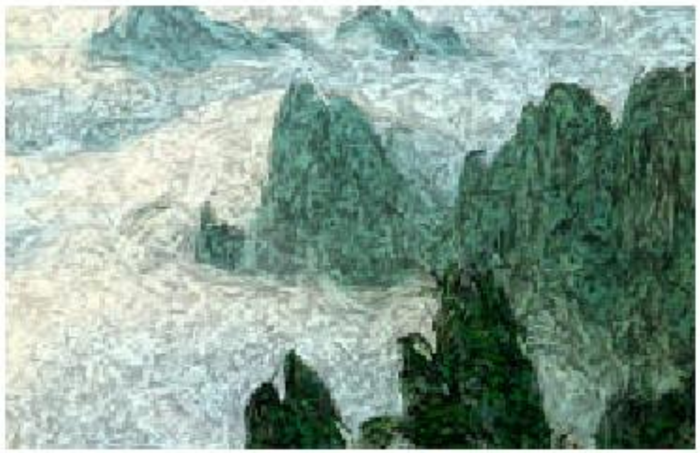
(a)

(b)

Styles

Impressionist

Expressionist



Colorist Wash

Style Transfer

Using AI. Not included in this course.

http://bethgelab.org/research/machine_learning/style_transfer/

Suggested Readings

- https://en.wikipedia.org/wiki/Color_quantization
- https://www.youtube.com/watch?v=LQST9MITKrw

- Aaron Hertzmann: Painterly Rendering with Curved Brush Strokes of Multiple Sizes, Siggraph 1998 Conference Proceedings, Pages 453-46, ACM Press https://www.mrl.nyu.edu/publications/painterly98/hertzmann-siggraph98.pdf

Exercise!

Cheer! ☺

Sketch 6 (Painterly Rendering)

- Implement the Hertzmann's algorithm
  - Three different brush sizes: (powers of two, e.g., 16, 8, 4)
  - Random or grid-based seed point selection
  - Curved strokes guided by vector field -> Sobel operator
  - Color picked from source and stroke length check
  - (Optional) Implement different styles (pointillist, impressionist, expressionist)