Digital Image Processing (CS/ECE 545) Lecture 8: Regions in Binary Images (Part 2) and Color (Part 1)

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Recall: Sequential Region Labeling

- 2 steps:
  1. Preliminary labeling of image regions
  2. Resolving cases where more than one label occurs (been previously labeled)
- Even though algorithm is complex (especially 2nd stage), it is preferred because it has lower memory requirements
- First step: preliminary labeling
- Check following pixels depending on if we consider 4-connected or 8-connected neighbors

$$\mathcal{N}_4(u,v) = \begin{array}{|c|c|c|} \hline & N_2 & \\ \hline N_1 & \times & \\ \hline & & \\ \hline \end{array} \quad \text{or} \quad \mathcal{N}_8(u,v) = \begin{array}{|c|c|c|} \hline N_2 & N_3 & N_4 \\ \hline N_1 & \times & \\ \hline & & \\ \hline \end{array}$$

Recall: Preliminary Labeling: Propagating Labels

- First foreground pixel **[1]** is found
- All neighbors in *N(u,v)* are background pixels **[0]**
- Assign pixel the first label **[2]**

(b) only background neighbors

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

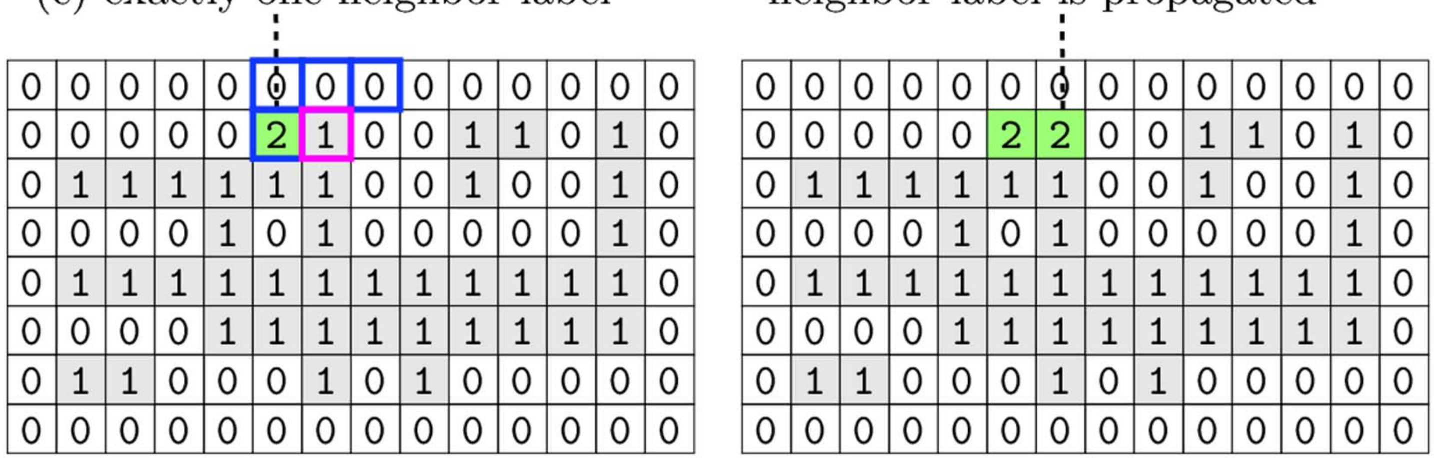
new label (**2**)

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	1	0	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0	1	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1	1	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Recall: Preliminary Labeling: Propagating Labels

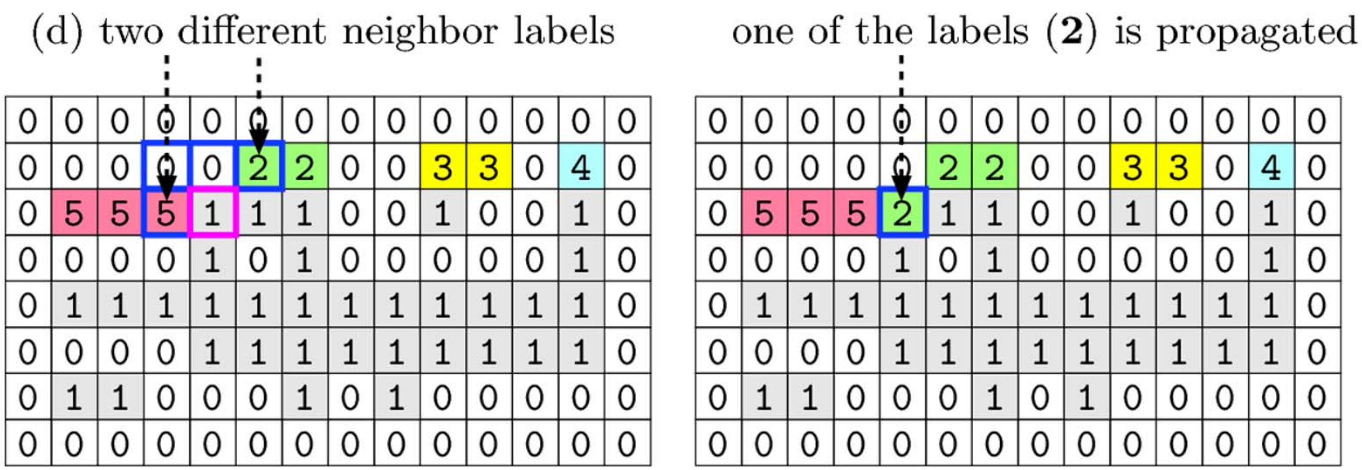
- In next step, exactly on neighbor in *N(u,v)* marked with label 2, so propagate this value **[2]**

(c) exactly one neighbor label — neighbor label is propagated



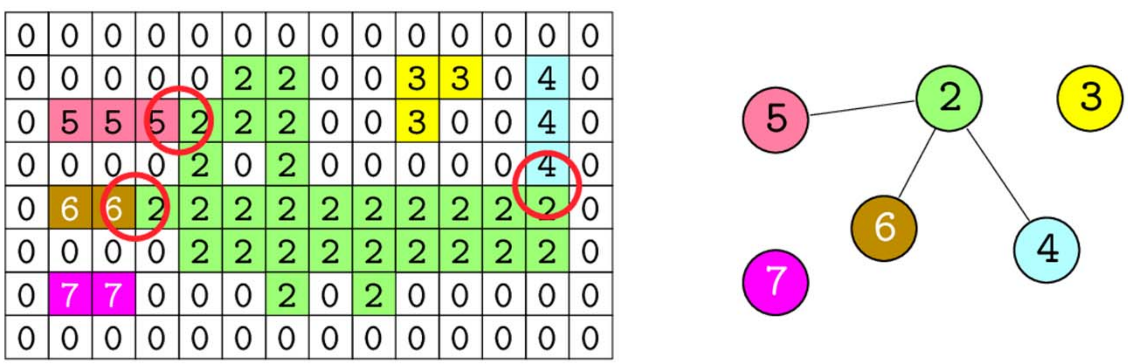
Recall: Preliminary Labeling: Propagating Labels

- Continue checking pixels as above
- At step below, there are **two** neighboring pixels and they have differing labels (**2** and **5**)
- One of these values is propagated (2 in this case), and collision <**2**,**5**> is registered



Recall: Preliminary Labeling: Label Collisions

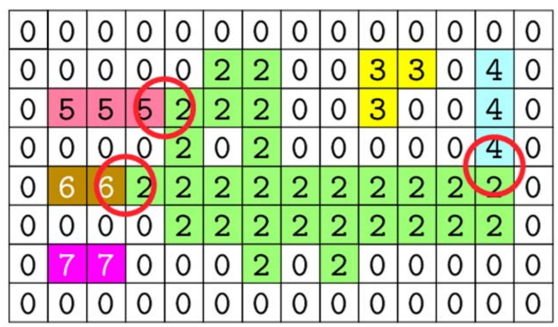
- At the end of labeling step
  - All foreground pixels have been provisionally marked
  - All collisions between labels (red circles) have been registered
  - Labels and collisions correspond to edges of undirected graph



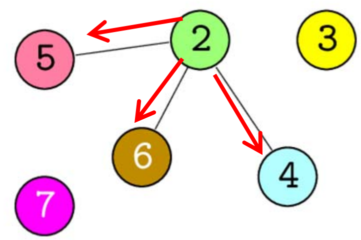
(a) (b)

Recall: Resolving Collisions

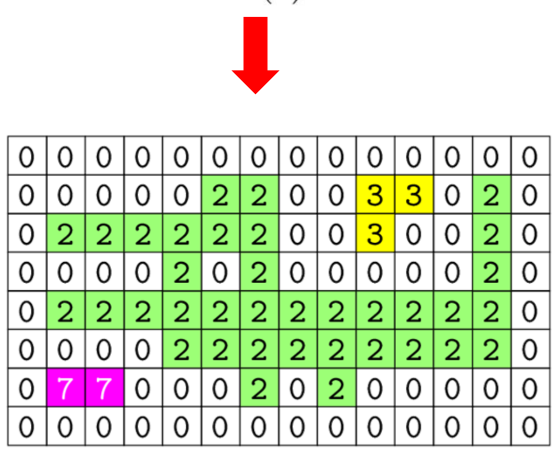
- Once all distinct labels within single region have been collected, assign labels of all pixels in region to be the same (e.g. assign all labels to have the smallest original label. E.g. **[2]**



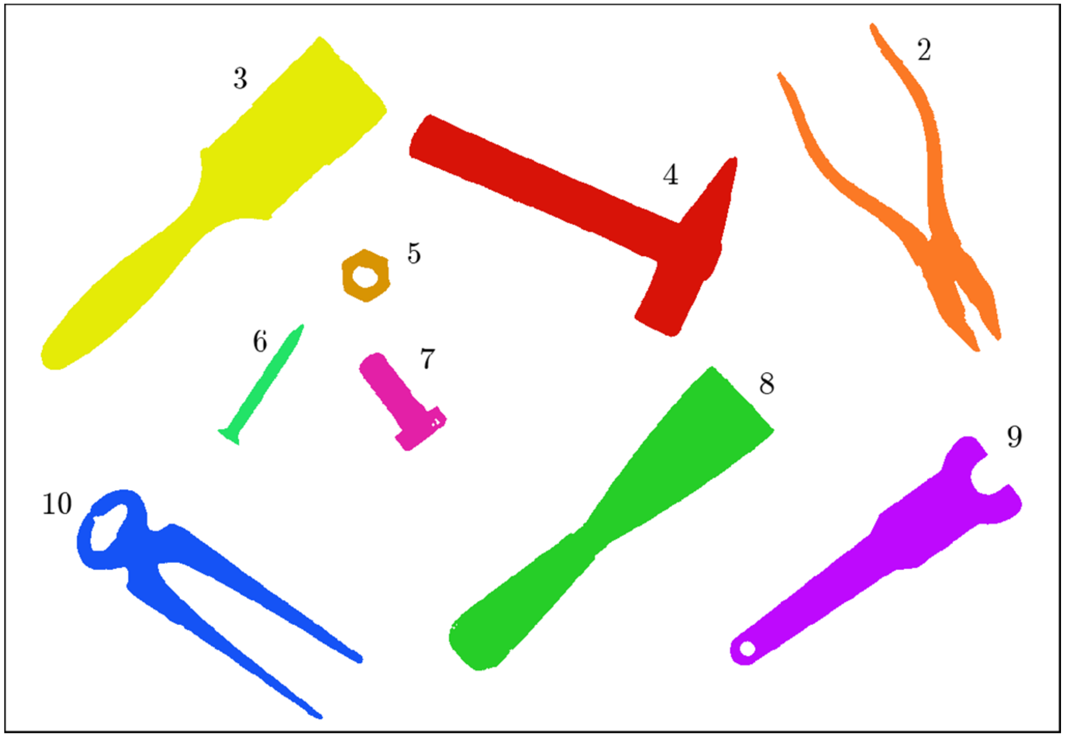
(a)



(b)



Region Labeling: Result



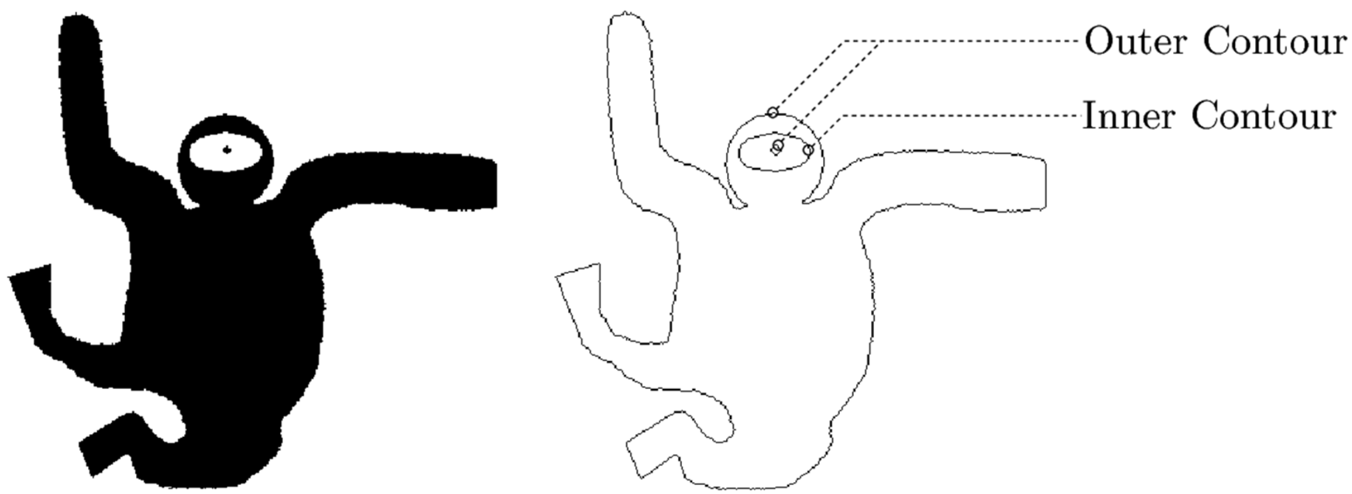
Label	Area (*pixels*)	Bounding Box (*left, top, right, bottom*)	Center (x_c, y_c)
2	14978	(887, 21, 1144, 399)	(1049.7, 242.8)
3	36156	(40, 37, 438, 419)	(261.9, 209.5)
4	25904	(464, 126, 841, 382)	(680.6, 240.6)
5	2024	(387, 281, 442, 341)	(414.2, 310.6)
6	2293	(244, 367, 342, 506)	(294.4, 439.0)
7	4394	(406, 400, 507, 512)	(454.1, 457.3)
8	29777	(510, 416, 883, 765)	(704.9, 583.9)
9	20724	(833, 497, 1168, 759)	(1016.0, 624.1)
10	16566	(82, 558, 411, 821)	(208.7, 661.6)

Region Contours

- After finding regions, find region contours (outlines)
- Sounds easy, but it is non-trivial!
- Morphological operations can be used to find boundary pixels (interior and exterior)
- We want ordered sequence of pixels that traces boundaries

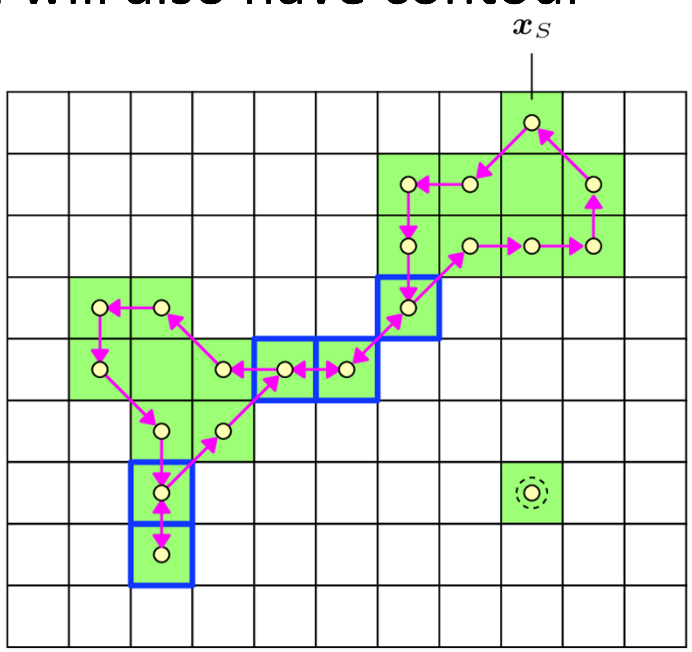
Inner vs Outer Contours

- Outer contour:
  - lies along outside of foreground (dark) region
  - Only 1 exists
- Inner contour:
  - Due to holes, there may be more than 1 inner contour



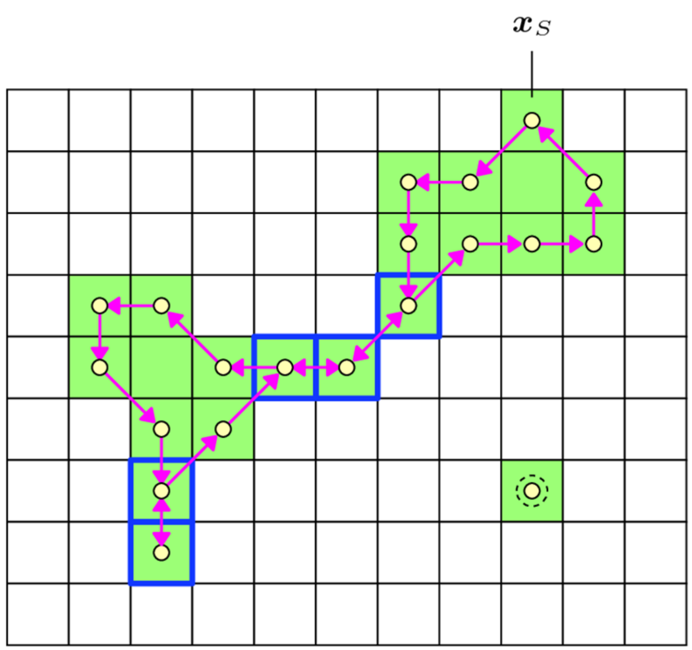
Inner vs Outer Contours

- Complicated by regions connected by thin line 1 pixel wide
- Contour may run through same pixel multiple times, from different directions
- **Implication:** we cannot use return to a starting pixel as condition to terminate contour
- Region with 1 pixel will also have contour



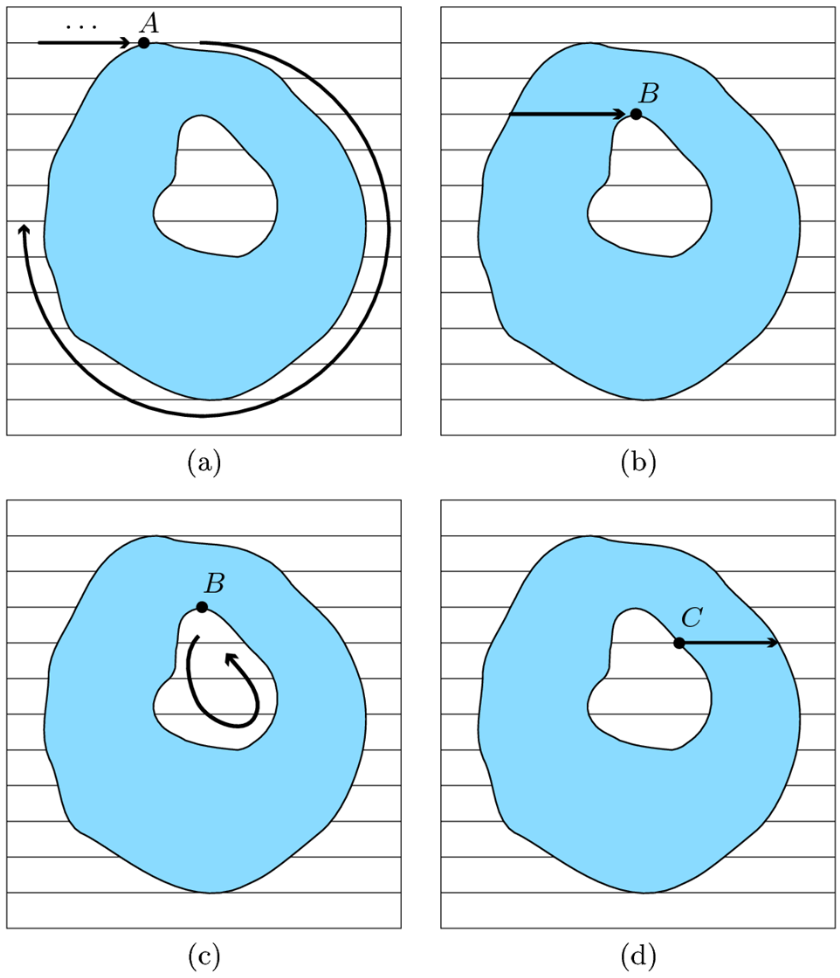
General Strategy for Finding Contours

- Two steps:
  - Find all connected regions in image
  - For each region proceed around it starting from pixel selected from its border
- Works well, but implementation requires good record keeping



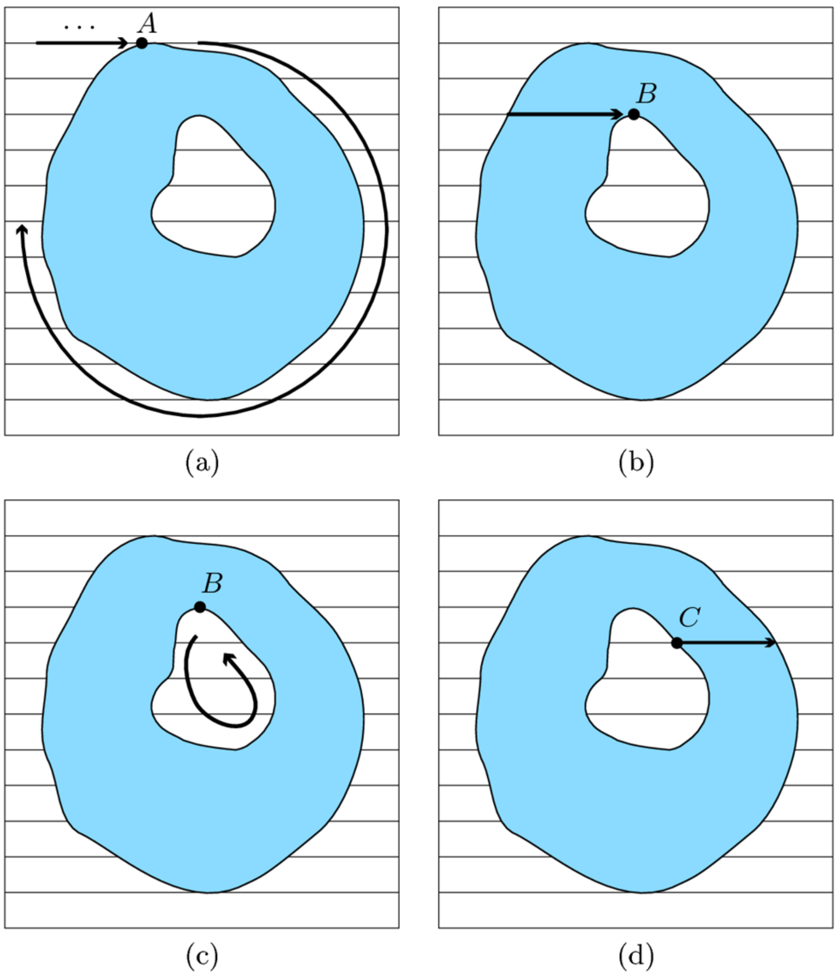
Combining Region Labeling and Contour Finding

- Identifies and labels regions
- Traces inner and outer contours
- **Step 1 (fig (a)):**
  - Image is traversed from top left to lower right.
  - If there's a transition from foreground pixel to previously unmarked foreground pixel (***A***), ***A*** lies on outer edge of a new region
  - A new label is allocated and starting from point ***A***, pixels on the edge along outer contour are visited and labeled until ***A*** is reached again (fig a)
  - Background pixels directly bordering region are labeled -1



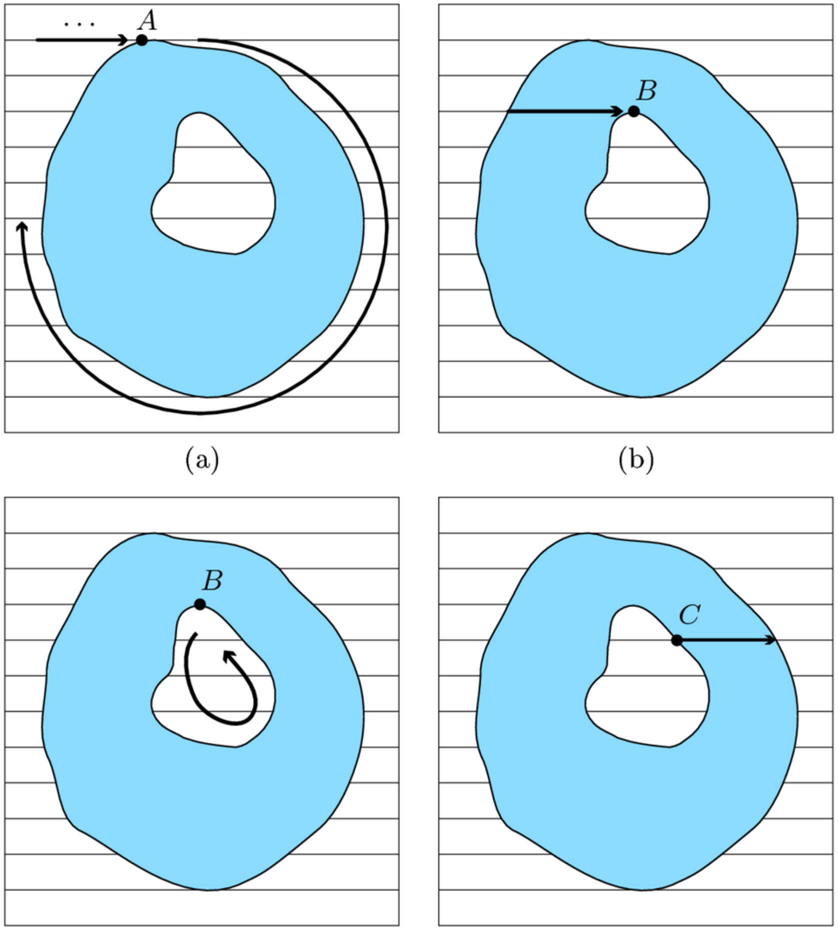
Combining Region Labeling and Contour Finding

- **Step 2 (fig (b) & (c)):**
  - If there’s transition from foreground pixel ***B*** to unmarked background pixel, ***B*** lies on **inner contour**.
  - Starting from point ***B*** inner contour is traversed. Pixels along inner contour are found and labeled with label from surrounding region (fig (c)) till arriving back at ***B***



Combining Region Labeling and Contour Finding

- **Step 3 (fig (d)):**
  - When foreground pixel does not lie on contour (not an edge), this means neighboring pixel to left has already been labeled (fig 11.9(d)) and this label is propagated to current pixel



(a) (b) (c) (d)

Algorithm for Combining Region Labeling and Contour Finding

```
CombinedContourLabeling (I)
        I: binary image
        Returns a set of contours and a label map (labeled image).

    Create an empty set of contours: C ← {}
    Create a label map LM of the same size as I and initialize:
    for all (u, v) do
        LM(u, v) ← 0                                  ▷ label map LM
    R ← 0                                             ▷ region counter R

    Scan the image from left to right and top to bottom:
    for v ← 0 ... N−1 do
        L_k ← 0                                       ▷ current label L_k
        for u ← 0 ... M−1 do
            if I(u, v) is a foreground pixel then
                if (L_k ≠ 0) then                     ▷ continue existing region
                    LM(u, v) ← L
                else
                    L_k ← LM(u, v)
                    if (L_k = 0) then                 ▷ hit new outer contour
                        R ← R + 1
                        L_k ← R
                        x_S ← (u, v)
                        c_outer ← TraceContour(x_S, 0, L_k, I, LM)
                        C ← C ∪ {c_outer}             ▷ collect new contour
                        LM(u, v) ← L_k
            else                                      ▷ I(u, v) is a background pixel
                if (L ≠ 0) then
                    if (LM(u, v) = 0) then            ▷ hit new inner contour
                        x_S ← (u−1, v)
                        c_inner ← TraceContour(x_S, 1, L_k, I, LM)
                        C ← C ∪ {c_inner}             ▷ collect new contour
                    L ← 0
    return (C, LM).          ▷ return the set of contours and the label map
```

continued in Alg. 11.4 ▷▷

Complete code in appendix D of text

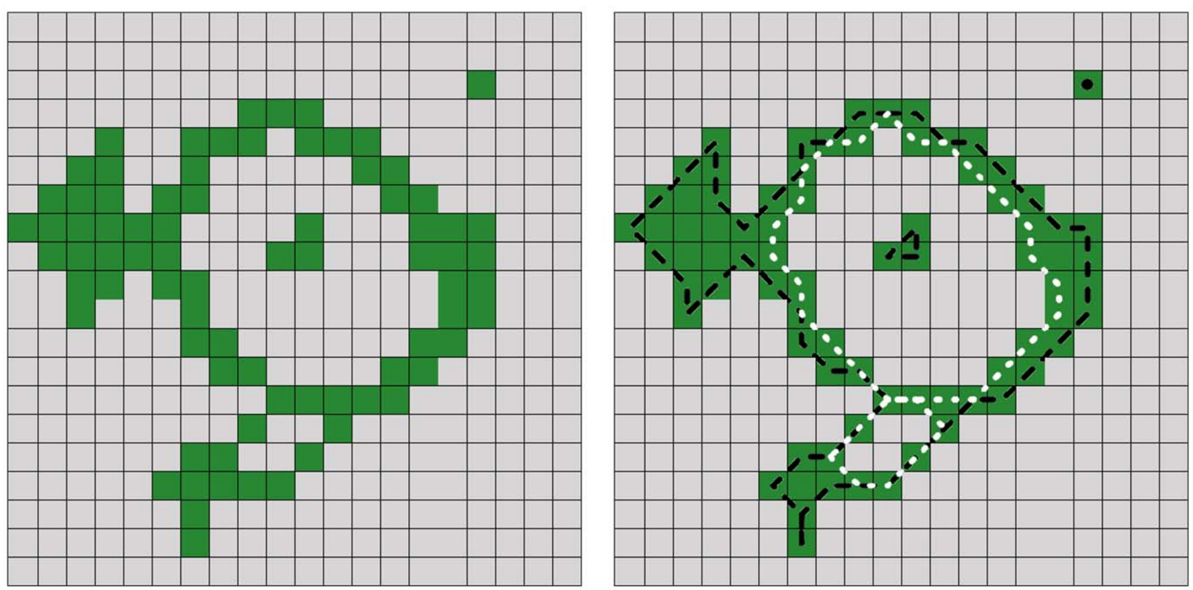
Algorithm for Combining Region Labeling and Contour Finding

```
TraceContour(x_S, d_S, L_k, I, LM)
        x_S: start position, d_S: initial search direction,
        L_c: label for this contour
        I: original image, LM: label map.
        Traces and returns the contour starting at x_S.
    (x_T, d_next) ← FindNextPoint(x_S, d_S, I, LM)
    c ← [x_T]                              ▷ create a contour starting with x_T
    x_p ← x_S                              ▷ previous position x_p = (u_p, v_p)
    x_c ← x_T                              ▷ current position x_c = (u_c, v_c)
    done ← (x_S ≡ x_T)                     ▷ isolated pixel?
    while (¬done) do
        LM(u_c, v_c) ← L_c
        d_search ← (d_next + 6) mod 8
        (x_n, d_next) ← FindNextPoint(x_c, d_search, I, LM)
        x_p ← x_c
        x_c ← x_n
        done ← (x_p ≡ x_S ∧ x_c ≡ x_T)     ▷ back at start point?
        if (¬done) then
            Append(c, x_n)                 ▷ add point x_n to contour c
    return c.                              ▷ return this contour

FindNextPoint(x_c, d, I, LM)
        x_c: start point, d: search direction,
        I: original image, LM: label map.
    for i ← 0...6 do                       ▷ search in 7 directions
        x' ← x_c + Delta(d)                ▷ x' = (u', v')
        if I(u', v') is a background pixel then
            LM(u', v') ← −1                ▷ mark background as visited (−1)
            d ← (d + 1) mod 8
        else                               ▷ found a nonbackground pixel at x'
            return (x', d)
    return (x_c, d).                       ▷ found no next point, return start point

Delta(d) = (Δx, Δy),   with
```

d	0	1	2	3	4	5	6	7
Δx	1	1	0	−1	−1	−1	0	1
Δy	0	1	1	1	0	−1	−1	−1

Result of Combining Region Labeling and Contour Finding

- Outer contours shown as black polygon lines running through centers of contour pixels
- Inner contours drawn in white
- Contours of single pixel regions marked by small circles filled with corresponding color



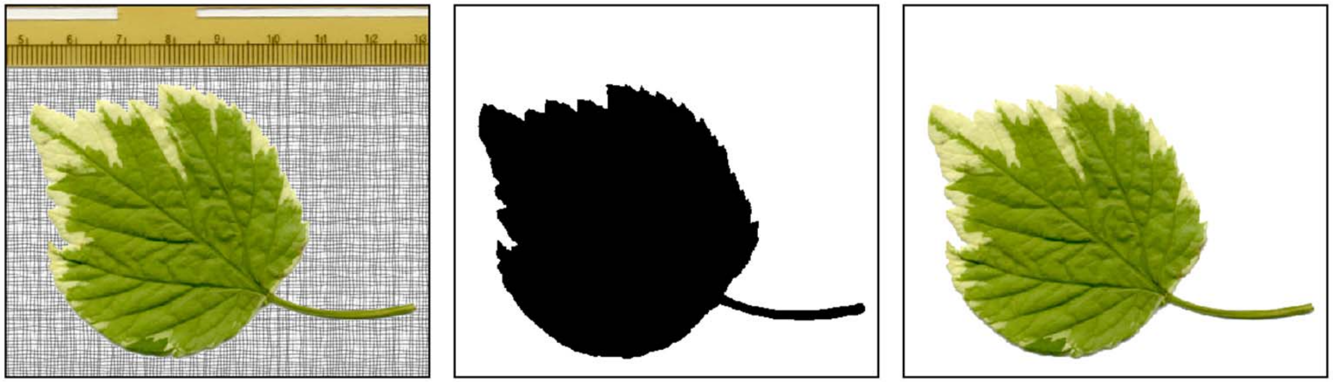
(a) (b)

Result of Combining Region Labeling and Contour Finding (Larger section)

- Outer contours marked in black
- Inner contours drawn in white

Representing Image Regions

- Matrix is useful for storing images
- Matrix representation requires same (large) memory allocation even if image content is small (e.g. 2 lines)
- Regions in image can be represented using logical mask
  - Area within region assigned value **true**
  - Area outside region assigned value **false**
- Called **bitmap** since boolean values can be represented by 1 bit



(a) (b) (c)

Original Image   Logical (big) mask   Masked Image

Run Length Encoding (RLE)

- Sequences of adjacent foreground pixels can be represented compactly as **runs**
- **Run:** Maximal length sequence of adjacent pixels of same type within row or column
- Runs of arbitrary length can be encoded as:

$$Run_i = \langle \text{row}_i, \text{column}_i, \text{length}_i \rangle$$

Starting pixel (row, column)

Example

Bitmap

	0	1	2	3	4	5	6	7	8
0									
1			×	×	×	×	×	×	
2									
3					×	×	×	×	
4		×	×	×		×	×	×	
5	×	×	×	×	×	×	×	×	×
6									

$\longrightarrow$

RLE

$\langle$row, column, length$\rangle$

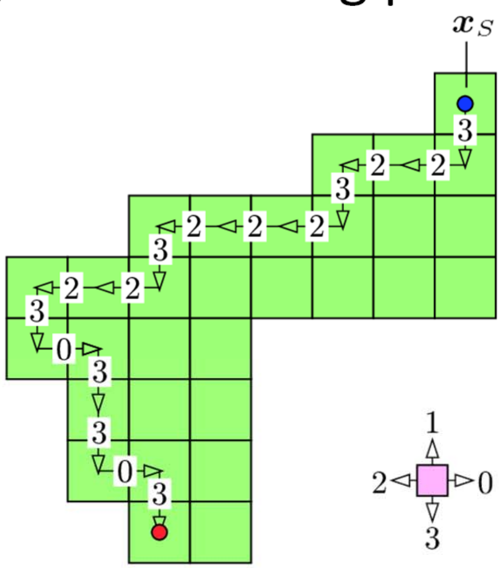
$\langle 1, 2, 6 \rangle$
$\langle 3, 4, 4 \rangle$
$\langle 4, 1, 3 \rangle$
$\langle 4, 5, 3 \rangle$
$\langle 5, 0, 9 \rangle$

Run Length Encoding (RLE)

- RLE used as simple lossless compression method
- Forms foundation for fax transmission
- Used in several codecs including TIFF, GIF and JPEG

Chain Codes

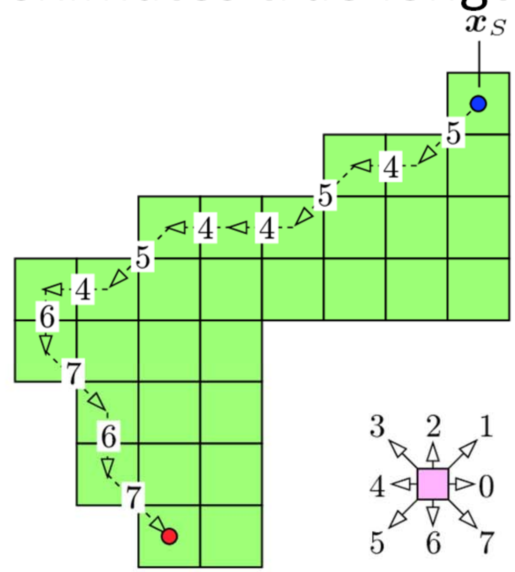
- Region representation using contour encoding
- Contour beginning at start point $\boldsymbol{x}_S$ represented by sequence of directional changes it describes on a discrete raster image
- Essentially, each possible direction is assigned a number
- Length of resulting path approximates true length of contour



4-Chain Code

3223222322303303...111

$length = 28$



8-Chain Code

54544546767...222

$length = 18 + 5\sqrt{2} \approx 25$

Differential Chain Codes

- Contour **R** is defined as sequence of points

$$\boldsymbol{c}_{\mathcal{R}} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots \boldsymbol{x}_{M-1}] \text{ with } \boldsymbol{x}_i = \langle u_i, v_i \rangle$$

- To encode region **R**,
  - Store starting point
  - Instead of storing sequence of point coordinates, store **relative direction** (8 possibilities) each point lies away from the previous point. i.e. create elements of its chain code sequence $\boldsymbol{c}'_{\mathcal{R}} = [c'_0, c'_1, \ldots c'_{M-1}]$ by

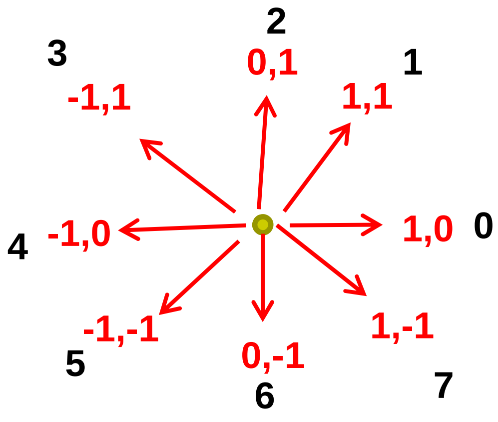
$$c'_i = \text{Code}(\Delta u_i, \Delta v_i)$$

Where

$$(\Delta u_i, \Delta v_i) = \begin{cases} (u_{i+1} - u_i, v_{i+1} - v_i) & \text{for } 0 \leq i < M-1 \\ (u_0 - u_i, v_0 - v_i) & \text{for } i = M-1, \end{cases}$$

  - Code is defined by table below

Δu	1	1	0	−1	−1	−1	0	1
Δv	0	1	1	1	0	−1	−1	−1
$\text{Code}(\Delta u, \Delta v)$	0	1	2	3	4	5	6	7



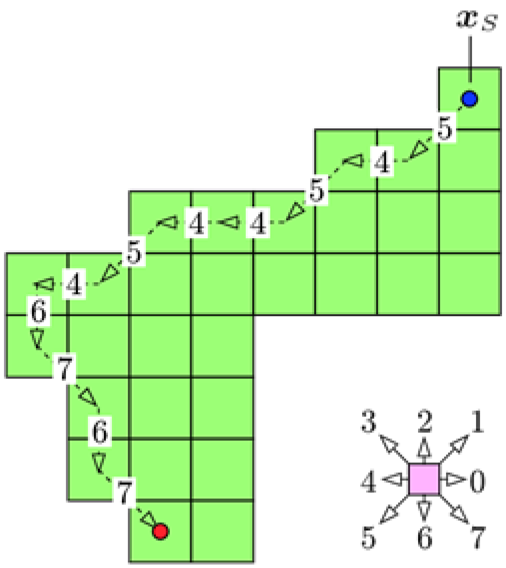
Differential Chain Code

- Comparison of 2 different absolute chain codes is difficult
- **Differential Chain Code:** Encode **change** in direction along discrete contour
- An absolute element chain code $\boldsymbol{c}'_{\mathcal{R}} = [c'_0, c'_1, \ldots c'_{M-1}]$ can be converted element by element to differential chain code with elements given by

$$c''_i = \begin{cases} (c'_{i+1} - c'_i) \bmod 8 & \text{for } \; 0 \le i < M-1 \\ (c'_0 - c'_i) \bmod 8 & \text{for } \; i = M-1 \end{cases}$$

Differential Chain Code Example

- **Differential Chain Code** for the following figure is:



8-Chain Code
54544546767...222
$length = 18 + 5\sqrt{2} \approx 25$

$$c'_{\mathcal{R}} = [5, 4, 5, 4, 4, 5, 4, 6, 7, 6, 7, \ldots 2, 2, 2]$$
$$c''_{\mathcal{R}} = [7, 1, 7, 0, 1, 7, 2, 1, 7, 1, 1, \ldots 0, 0, 3]$$

Example: 7 – 6 = 1

Shape Numbers

- Digits of differential chain code frequently interpreted as number to base b
  - b = 4 for 4-connected contour
  - b = 8 for 8-connected contour

$$\begin{aligned}\mathrm{Val}(\boldsymbol{c}''_{\mathcal{R}}) &= c''_0 \cdot b^0 + c''_1 \cdot b^1 + \ \ldots \ + c''_{M-1} \cdot b^{M-1} \\ &= \sum_{i=0}^{M-1} c''_i \cdot b^i\end{aligned}$$

- We can shift the chain code sequence cyclically
- Example: shifting chain code cyclically by 2 positions gives

$$\begin{aligned}\boldsymbol{c}''_{\mathcal{R}} &= [\,0, 1, 3, 2, \ldots 9, 3, 7, 4\,] \\ \boldsymbol{c}''_{\mathcal{R}} \rhd 2 &= [\,7, 4, 0, 1, 3, 2, \ldots 9, 3\,]\end{aligned}$$

Shape Number

- We can shift the sequence cyclically until the numeric value is maximized denoted as

$$k_{\max} = \arg \max_{0 \leq k < M} \textsc{Val}(\boldsymbol{c}''_{\mathcal{R}} \rhd k)$$

- The resulting code is called the **shape number**
- To compare 2 differential codes, they must have same starting point
- Shape number does not have this requirement
- In general chain codes are not useful for determining similarity between regions because
  - Arbitrary rotations have too great of an impact on them
  - Cannot handle scaling or distortions

Fourier Descriptors

- Interprete 2D contour as a sequence of values [z_0, z_1, ….z_{M-1}] in complex plane, where

$$z_i = (u_i + \mathrm{i} \cdot v_i) \in \mathbb{C}$$

- Coefficients of the 1D **Fourier spectrum** of this function provide a shape description of the contour in frequency space

Properties of Binary Regions

- Human descriptions of regions based on their properties:
  - “a red rectangle on a blue background”
  - “sunset at the beach with two dogs playing in the sand”
- Not yet possible for computers to generate such descriptors
- Alternatively, computers can calculate mathematical properties of image or region to use for classification
- Using features to classify images is fundamental part of **pattern recognition**

Types of Features

- Shape features
- Geometric features
- Statistical shape properties
- Moment-Based Geometrical Properties
- Topological Properties

Shape Features

- **Feature:** numerical or qualitative value computable from values and coordinates of pixels in region
- **Example feature:** One of simplest features is **size** which is the total number of pixels in region
- **Feature vector:**
  - Combination of different features
  - Used as a sort of "signature" for the region for classification or comparison
- Desirable properties of features
  - Simple to calculate
  - Not affected by translation, rotations and scaling

Geometric Features

- Region **R** of binary image = 2D distribution of foreground points within discrete plane
- **Perimeter:** Length of region's outer contour
- Note that the region **R** must be connected
- For 4-neighborhood, measured length of contour is larger than it's actual length
- Good approximation for 8-connected chain code $\boldsymbol{c}'_{\mathcal{R}} = [c'_0, c'_1, \ldots c'_{M-1}]$

$$\text{Perimeter}(\mathcal{R}) = \sum_{i=0}^{M-1} \text{length}(c'_i)$$

$$\text{with} \quad \text{length}(c) = \begin{cases} 1 & \text{for} \quad c = 0, 2, 4, 6 \\ \sqrt{2} & \text{for} \quad c = 1, 3, 5, 7 \end{cases}$$

- Formula leads to overestimation. Good fix: multiply by 0.95

$$P(\mathcal{R}) \approx \text{Perimeter}_{\text{corr}}(\mathcal{R}) = 0.95 \cdot \text{Perimeter}(\mathcal{R})$$

Geometric Features

- **Area:** Simply count image pixels that make up region

$$A(\mathcal{R}) = |\mathcal{R}| = N.$$

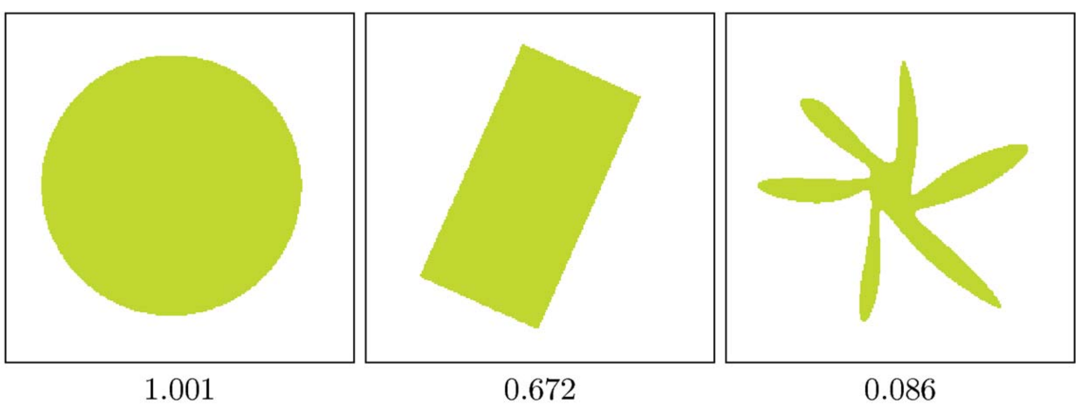
- **Area of connected region (without holes):** that is defined by M coordinate points can be estimated using the Gaussian area formula for polygons as

$$A(\mathcal{R}) \approx \frac{1}{2} \cdot \left| \sum_{i=0}^{M-1} \left(u_i \cdot v_{(i+1) \bmod M} - u_{(i+1) \bmod M} \cdot v_i \right) \right|$$

Geometric Features

- **Compactness and Roundness:** is the relationship between a region's area and its perimeter. i.e. A / P^2
- Invariant to translation, rotation and scaling.
- When applied to circular region ratio has value of $1/ 4\pi$
- Thus, normalizing against filled circle creates feature sensitive to **roundness** or **circularity**

$$\text{Circularity}(\mathcal{R}) = 4\pi \cdot \frac{A(\mathcal{R})}{P^2(\mathcal{R})}$$

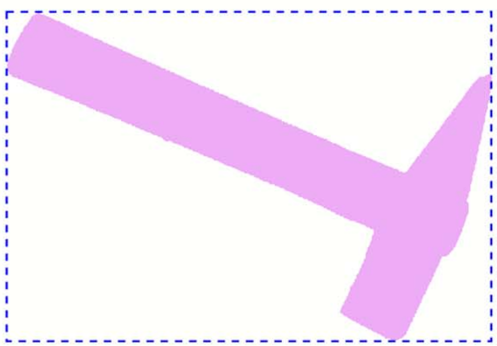


Geometric Features

- **Bounding Box:** mininimal axis-parallel rectangle that encloses all points in **R**

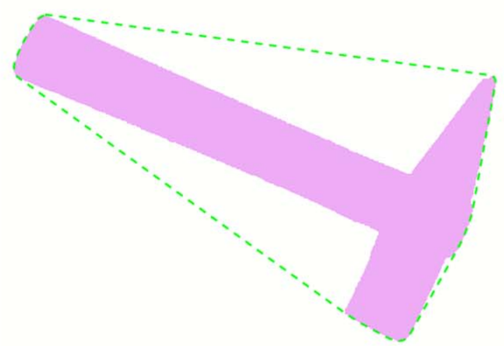
$$\text{BoundingBox}(\mathcal{R}) = \langle u_{\min}, u_{\max}, v_{\min}, v_{\max} \rangle$$

- **Convex Hull:** Smallest polygon that fits all points in **R**
  - **Convexity:** relationship between length of convex hull and perimeter of the region
  - **Density:** the ratio between area of the region and area of the convex hull



(a)

Bounding box



(b)

Convex Hull

Statistical Shape Properties

- View points as being statistically distributed in 2D space
- Can be applied to regions that are **not connected**
- **Central moments** measure characteristic properties with respect to its midpoint or centroid
- **Centroid:** of a binary region is the arithmetic mean of all (x,y) coordinates in the region

$$\bar{x} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(u,v)\in\mathcal{R}} u \qquad \text{and} \qquad \bar{y} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(u,v)\in\mathcal{R}} v$$

Statistical Shape Properties

- **Moments:** Centroid is only specific case of more general concept of moment
- **Ordinary moment** of the order **p,q** for a discrete (image) function **I(u,v)** is

$$m_{pq} = \sum_{(u,v)\in\mathcal{R}} I(u,v) \cdot u^p v^q$$

← **Taking the pth moment in u direction And qth moment in v direction**

- **Area** of a binary region is **zero-order moment**

$$A(\mathcal{R}) = |\mathcal{R}| = \sum_{(u,v)\in\mathcal{R}} 1 = \sum_{(u,v)\in\mathcal{R}} u^0 v^0 = m_{00}(\mathcal{R})$$

Statistical Shape Properties

- Similarly centroid can be expressed as

$$\bar{x} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(u,v)\in\mathcal{R}} u^1 v^0 = \frac{m_{10}(\mathcal{R})}{m_{00}(\mathcal{R})}$$

$$\bar{y} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(u,v)\in\mathcal{R}} u^0 v^1 = \frac{m_{01}(\mathcal{R})}{m_{00}(\mathcal{R})}$$

- Moments are concrete physical properties of a region

Review: Moments of Statistical Data

- Moments used to quantify how skewed data is
- Example: Given the numbers, 3, 2, 3.7, 5, 2.7 and 3 the relative symmetry or skewness can be determined by calculating moments
- Third moment formula: For each point X, calculate:

$$m_3 = \frac{\sum(X - Average)^3}{N}$$

- We can calculate 2^{nd}, 3^{rd}, 4^{th}, etc moments

Statistical Shape Properties

- **Central moments:**
  - Use the region's centroid as reference to calculate translation-invariant region features
  - Shifts the origin to the region's centroid (**Note:** ordinary moment does not)
- **Order p, q central moments** can be calculated as:

$$\mu_{pq}(\mathcal{R}) = \sum_{(u,v)\in\mathcal{R}} I(u,v) \cdot (u-\bar{x})^p \cdot (v-\bar{y})^q$$

- For binary image with *I(u,v)* = 1

$$\mu_{pq}(\mathcal{R}) = \sum_{(u,v)\in\mathcal{R}} (u-\bar{x})^p \cdot (v-\bar{y})^q$$

Statistical Shape Properties

- Values of central moments depends on:
  - Distances of all region points to centroid
  - Absolute size of the region
- **Size-invariant** features can be obtained by scaling central moments uniformly by some factor ***s***

$$s^{(p+q+2)}$$

- **Normalized central moments:**

$$\bar{\mu}_{pq}(\mathcal{R}) = \mu_{pq} \cdot \left(\frac{1}{\mu_{00}(\mathcal{R})}\right)^{(p+q+2)/2}$$

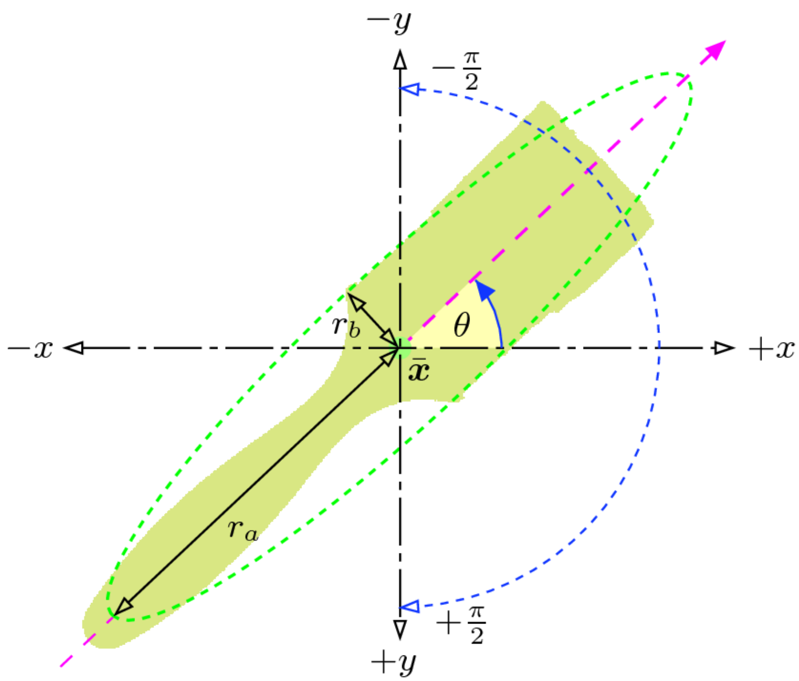
for (p + q) >= 2

Code to Compute Moments

```
import ij.process.ImageProcessor;

public class Moments {
  static final int BACKGROUND = 0;

  static double moment(ImageProcessor ip,int p,int q) {
    double Mpq = 0.0;
    for (int v = 0; v < ip.getHeight(); v++) {
      for (int u = 0; u < ip.getWidth(); u++) {
        if (ip.getPixel(u,v) != BACKGROUND) {
          Mpq += Math.pow(u, p) * Math.pow(v, q);
        }
      }
    }
    return Mpq;
  }
  static double centralMoment(ImageProcessor ip,int p,int q)
    {
    double m00 = moment(ip, 0, 0); // region area
    double xCtr = moment(ip, 1, 0) / m00;
    double yCtr = moment(ip, 0, 1) / m00;
    double cMpq = 0.0;
    for (int v = 0; v < ip.getHeight(); v++) {
      for (int u = 0; u < ip.getWidth(); u++) {
        if (ip.getPixel(u,v) != BACKGROUND) {
          cMpq +=
            Math.pow(u - xCtr, p) *
            Math.pow(v - yCtr, q);
        }
      }
    }
    return cMpq;
  }
  static double normalCentralMoment
                        (ImageProcessor ip,int p,int q) {
    double m00 = moment(ip, 0, 0);
    double norm = Math.pow(m00, (double)(p + q + 2) / 2);
    return centralMoment(ip, p, q) / norm;
  }

} // end of class Moments
```

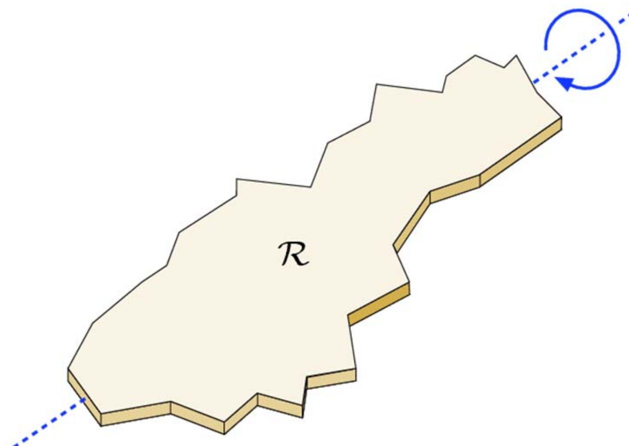
Moment-Based Geometrical Properties

- Several interesting features can be derived from moments
- **Orientation:** describes direction of **major axis** that runs through centroid and along the widest part of the region



Moment-Based Geometrical Properties

- Sometimes called the **major axis of rotation** since rotating the region around the major axis requires the least effort than about any other axis



- Direction of major axis can be calculated from central moments

$$\tan(2\,\theta_{\mathcal{R}}) = \frac{2 \cdot \mu_{11}(\mathcal{R})}{\mu_{20}(\mathcal{R}) - \mu_{02}(\mathcal{R})} \quad \Rightarrow \quad \theta_{\mathcal{R}} = \frac{1}{2}\tan^{-1}\left(\frac{2 \cdot \mu_{11}(\mathcal{R})}{\mu_{20}(\mathcal{R}) - \mu_{02}(\mathcal{R})}\right)$$

- Result is in the range [-90, 90]

Moment-Based Geometrical Properties

- Might want to plot region's orientation as a line or arrow
- Using the parametric equation of a line

$$\boldsymbol{x} = \bar{\boldsymbol{x}} + \lambda \cdot \boldsymbol{x}_d = \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} + \lambda \cdot \begin{pmatrix} \cos(\theta_{\mathcal{R}}) \\ \sin(\theta_{\mathcal{R}}) \end{pmatrix}$$

Start point **Direction vector**

- Region's orientation vector ***x_d*** can be computed as

$$x_d = \cos(\theta_{\mathcal{R}}) = \begin{cases} 0 & \text{for } A = B = 0 \\ \left[\frac{1}{2}\left(1 + \frac{B}{\sqrt{A^2+B^2}}\right)\right]^{\frac{1}{2}} & \text{otherwise,} \end{cases}$$

$$y_d = \sin(\theta_{\mathcal{R}}) = \begin{cases} 0 & \text{for } A = B = 0 \\ \left[\frac{1}{2}\left(1 - \frac{B}{\sqrt{A^2+B^2}}\right)\right]^{\frac{1}{2}} & \text{for } A \geq 0 \\ -\left[\frac{1}{2}\left(1 - \frac{b}{\sqrt{A^2+B^2}}\right)\right]^{\frac{1}{2}} & \text{for } A < 0, \end{cases}$$

where

$$A = 2\mu_{11}(\mathcal{R}) \qquad B = \mu_{20}(\mathcal{R}) - \mu_{02}(\mathcal{R})$$

Moment-Based Geometric Properties

- **Eccentricity:** Ratio of lengths of major axis and minor axis
- Expresses how elongated the region is

$$\text{Ecc}(\mathcal{R}) = \frac{a_1}{a_2} = \frac{\mu_{20} + \mu_{02} + \sqrt{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2}}{\mu_{20} + \mu_{02} - \sqrt{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2}}$$

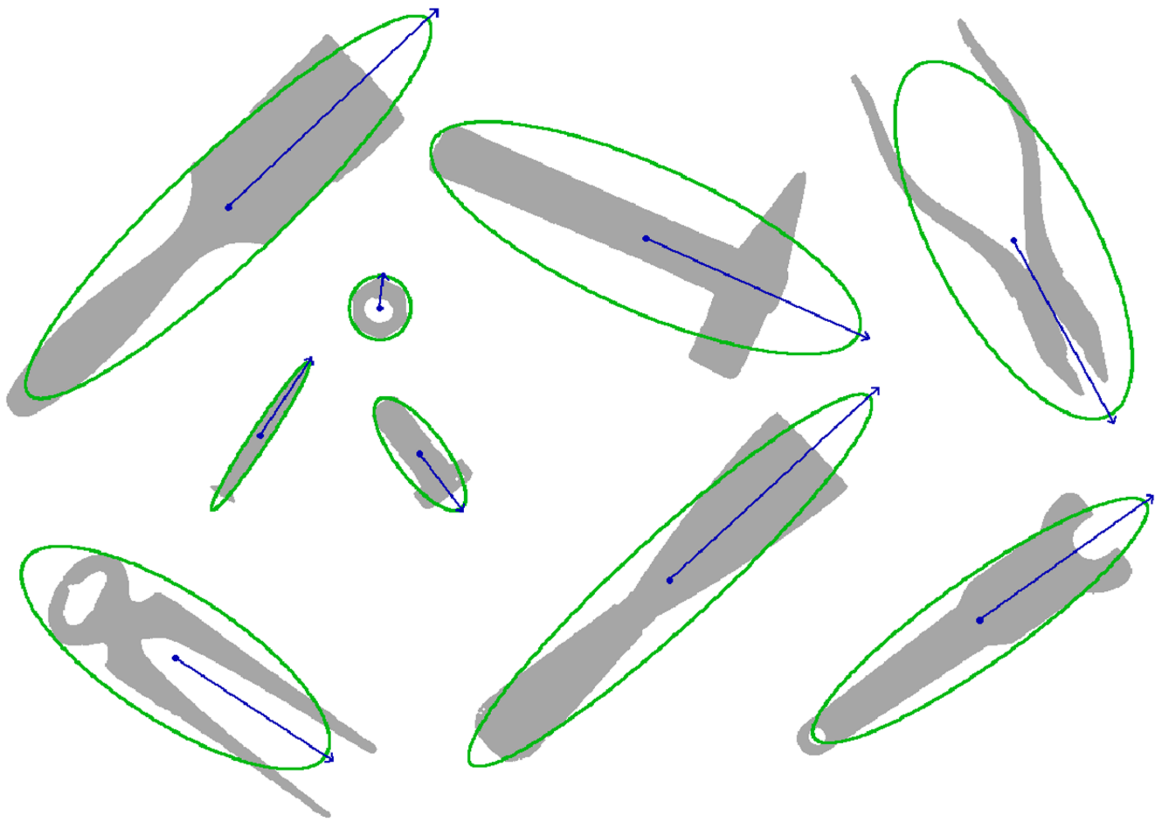
- The lengths of the major and minor axis are

$$r_a = 2 \cdot \left(\frac{\lambda_1}{|\mathcal{R}|}\right)^{\frac{1}{2}} = \left(\frac{2\,a_1}{|\mathcal{R}|}\right)^{\frac{1}{2}}$$

$$r_b = 2 \cdot \left(\frac{\lambda_2}{|\mathcal{R}|}\right)^{\frac{1}{2}} = \left(\frac{2\,a_2}{|\mathcal{R}|}\right)^{\frac{1}{2}}$$

Moment-Based Geometrical Properties

- Example images with orientation and eccentricity overlaid



Moment-Based Geometrical Properties

- Normalized central moments not affected by translation or uniform scaling of a region but changed by rotation
- Moments called **Hu's moments** (seven combinations of normalized central moments) are invariant to translation, scaling and rotation

$$
\begin{aligned}
H_1 &= \bar{\mu}_{20} + \bar{\mu}_{02} \\
H_2 &= (\bar{\mu}_{20} - \bar{\mu}_{02})^2 + 4\,\bar{\mu}_{11}^2 \\
H_3 &= (\bar{\mu}_{30} - 3\,\bar{\mu}_{12})^2 + (3\,\bar{\mu}_{21} - \bar{\mu}_{03})^2 \\
H_4 &= (\bar{\mu}_{30} + \bar{\mu}_{12})^2 + (\bar{\mu}_{21} + \bar{\mu}_{03})^2 \\
H_5 &= (\bar{\mu}_{30} - 3\,\bar{\mu}_{12}) \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot \left[(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - 3(\bar{\mu}_{21} + \bar{\mu}_{03})^2\right] \\
&\quad + (3\,\bar{\mu}_{21} - \bar{\mu}_{03}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03}) \cdot \left[3\,(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2\right] \\
H_6 &= (\bar{\mu}_{20} - \bar{\mu}_{02}) \cdot \left[(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2\right] \\
&\quad + 4\,\bar{\mu}_{11} \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03}) \\
H_7 &= (3\,\bar{\mu}_{21} - \bar{\mu}_{03}) \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot \left[(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - 3\,(\bar{\mu}_{21} + \bar{\mu}_{03})^2\right] \\
&\quad + (3\,\bar{\mu}_{12} - \bar{\mu}_{30}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03}) \cdot \left[3\,(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2\right]
\end{aligned}
$$

Projections

- **Horizontal projection** of row $\mathbf{v_0}$ is sum of pixel intensity values in row $\mathbf{v_0}$
- **Vertical projection** of row $\mathbf{u_0}$ is sum of pixel intensity values in row $\mathbf{u_0}$
- For binary image, projection is count of foreground pixels in corresponding row or column

Projections

- Image projections are 1d representations of image contents
- Vertical and horizontal projections of image *I(u,v)* defined as

$$P_{\text{hor}}(v_0) = \sum_{u=0}^{M-1} I(u, v_0) \qquad \text{for} \ \ 0 < v_0 < N$$

$$P_{\text{ver}}(u_0) = \sum_{v=0}^{N-1} I(u_0, v) \qquad \text{for} \ \ 0 < u_0 < M$$

Code to Compute Vertical and Horizontal Projections

```
public void run(ImageProcessor ip) {
  int M = ip.getWidth();
  int N = ip.getHeight();
  int[] horProj = new int[N];
  int[] verProj = new int[M];
  for (int v = 0; v < N; v++) {
    for (int u = 0; u < M; u++) {
      int p = ip.getPixel(u, v);
      horProj[v] += p;
      verProj[u] += p;
    }
  }
  // use projections horProj, verProj now
  // ...
}
```

Topological Properties

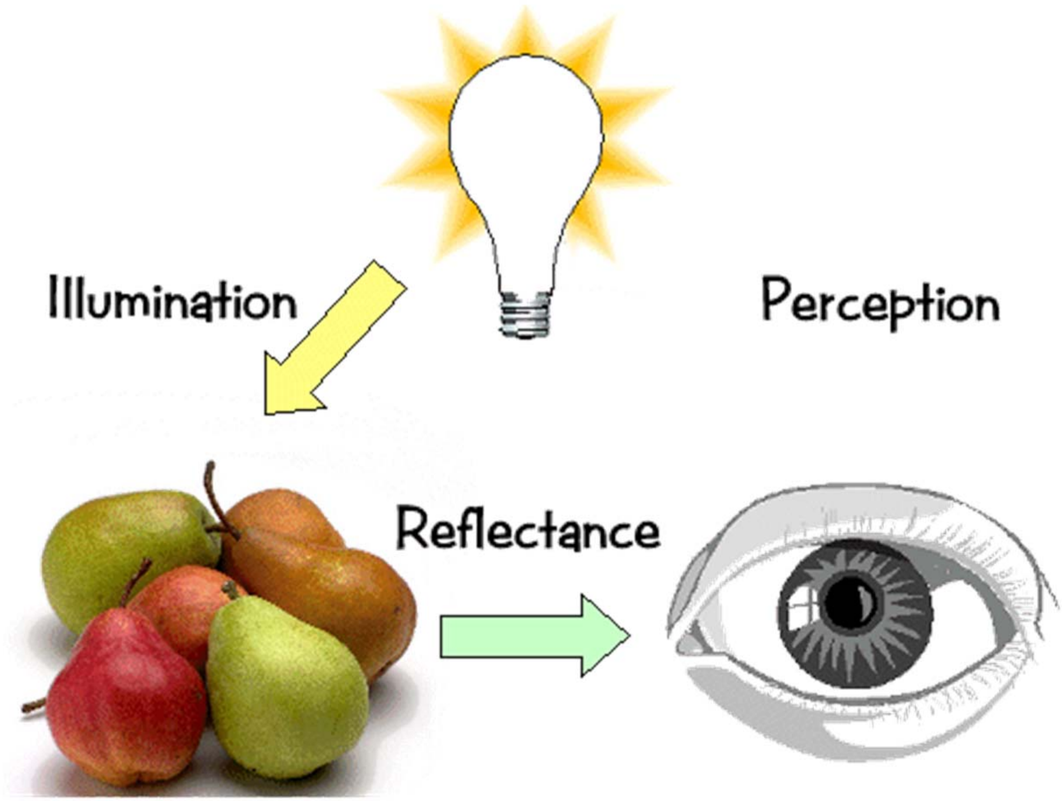
- Capture the structure of a region
- Invariant under strong image transformations
- **Number of holes** is simple, robust feature
- **Euler number:** Number of connected regions – number of holes

$$N_E(\mathcal{R}) = N_R(\mathcal{R}) - N_L(\mathcal{R})$$

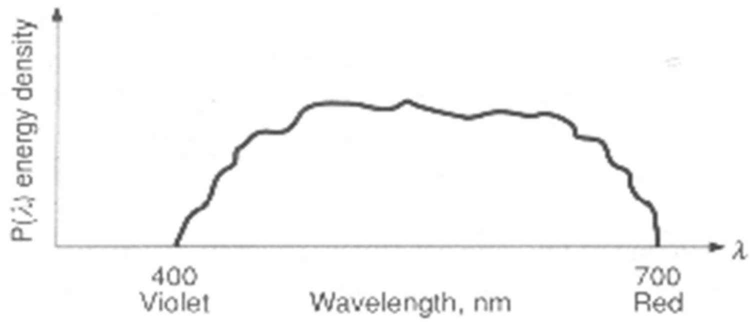
- Topological features often combined with numerical features (e.g. in Optical Character Recognition (OCR))

Basics Of Color

- Elements of color:



What is color?



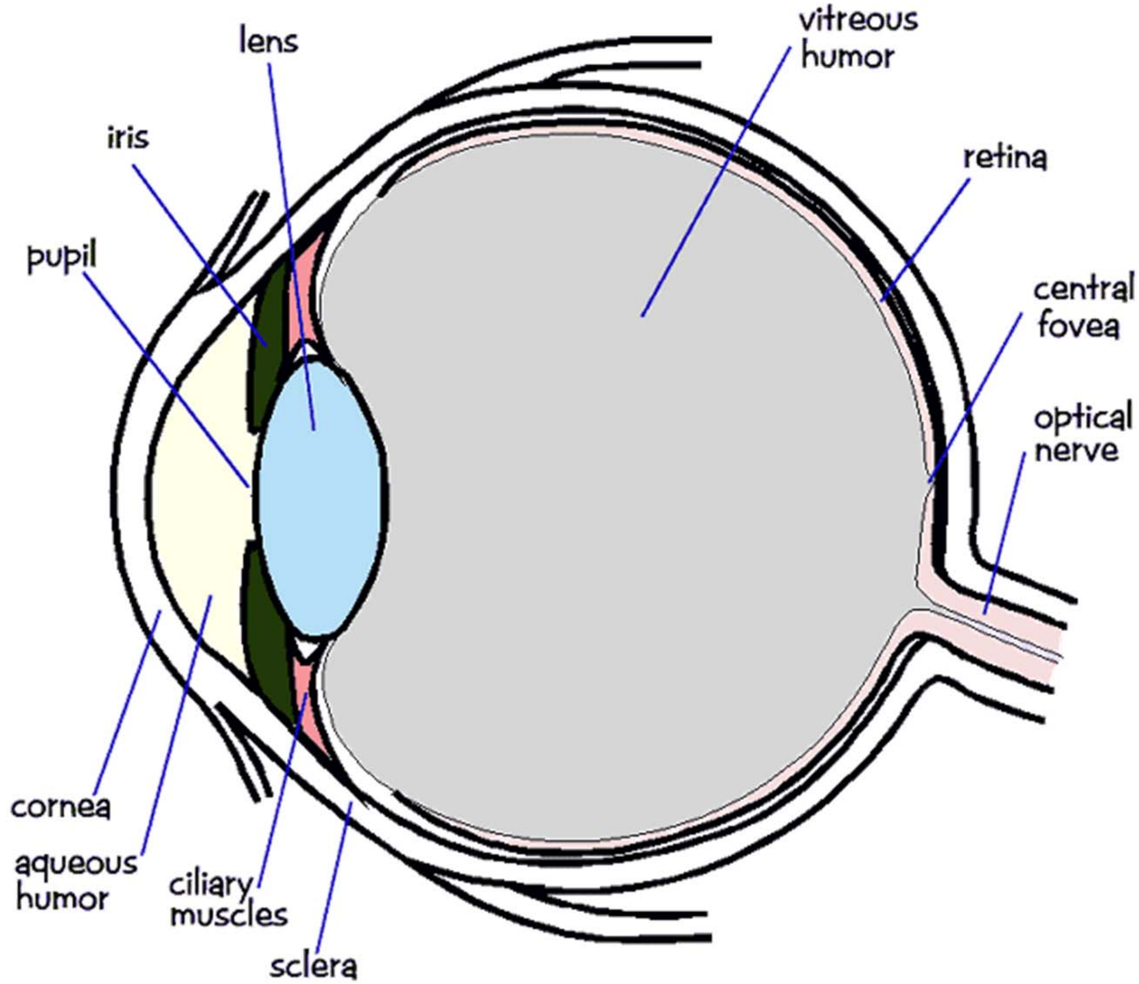
- Color is defined many ways
- Physical definition
  - Wavelength of photons
  - Electromagnetic spectrum: infra-red to ultra-violet
- But so much more than that…
  - Excitation of photosensitive molecules in eye
  - Electrical impulses through optical nerves
  - Interpretation by brain

Introduction

- **Color description:** Red, greyish blue, white, dark green…
- **Computer Scientist:**
  - **Hue:** dominant wavelength, color we see
  - **Saturation**
    - how pure the mixture of wavelength is
    - How far is the color from gray (pink is less saturated than red, sky blue is less saturated than royal blue)
  - **Lightness/brightness:** how intense/bright is the light

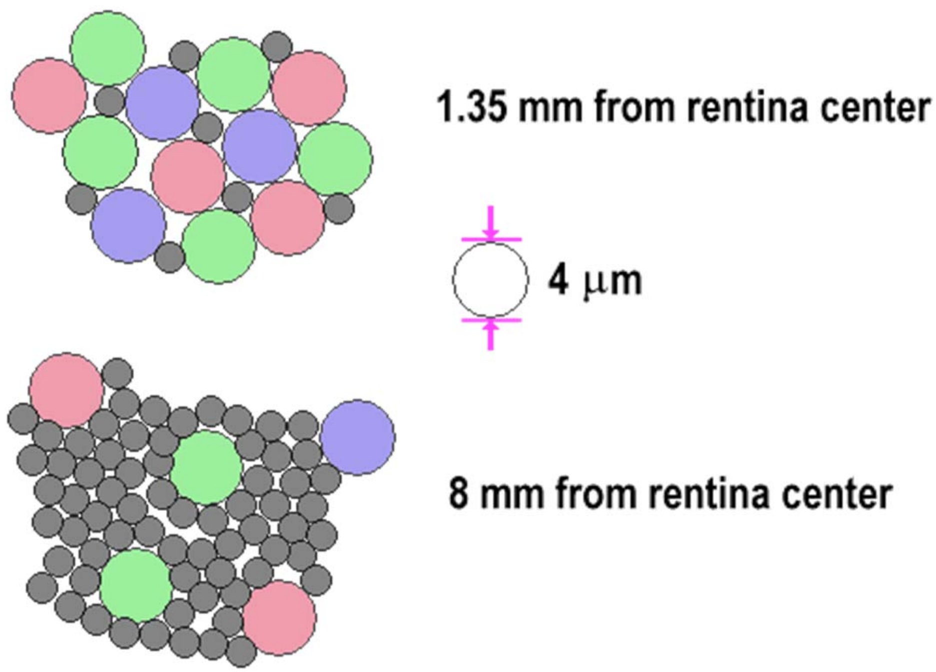
Recall: The Human Eye

- The eye:
- The retina
  - Rods
  - Cones
    - Color!



Recall: The Human Eye

- The center of the retina is a densely packed region called the *fovea*.
  - Eye has about 6- 7 million cones
  - Cones much denser here than the *periphery*

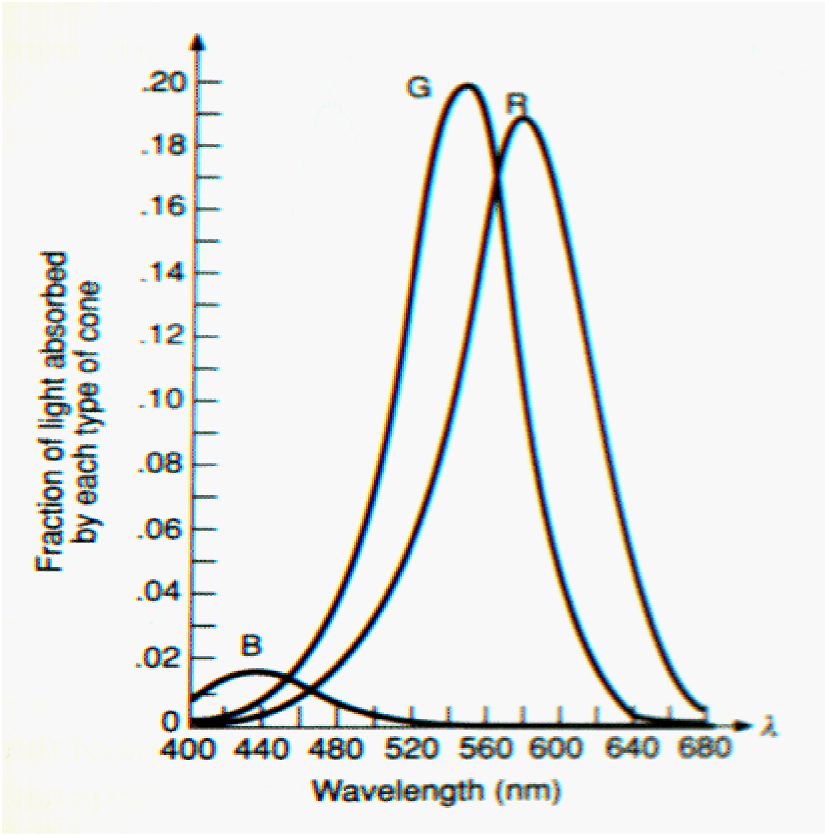


The Human Eye

- Rods:
  - relatively insensitive to color, detail
  - Good at seeing in dim light, general object form
- Human eye can distinguish
  - 128 different hues of color
  - 20 different saturations of a given hue
- **Visible spectrum:** about 380nm to 720nm
- Hue, luminance, saturation useful for describing color
- Given a color, tough to derive HSL though

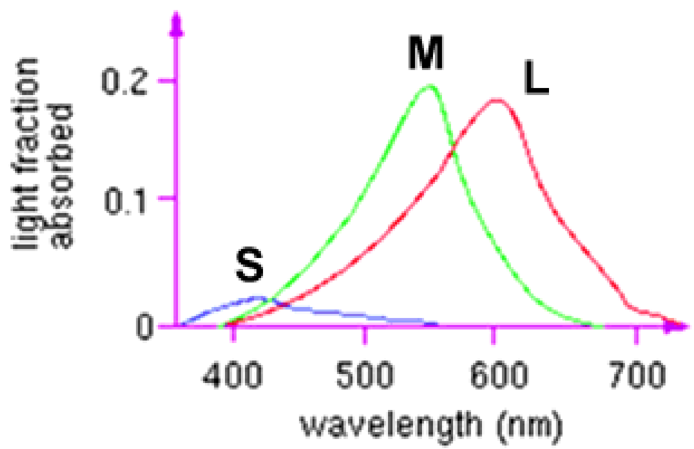
Tristimulus theory

- 3 types of cones
  - Loosely identify as R, G, and B cones
- Each is sensitive to its own spectrum of wavelengths
- Combination of cone cell stimulations give perception of **COLOR**



The Human Eye: Cones

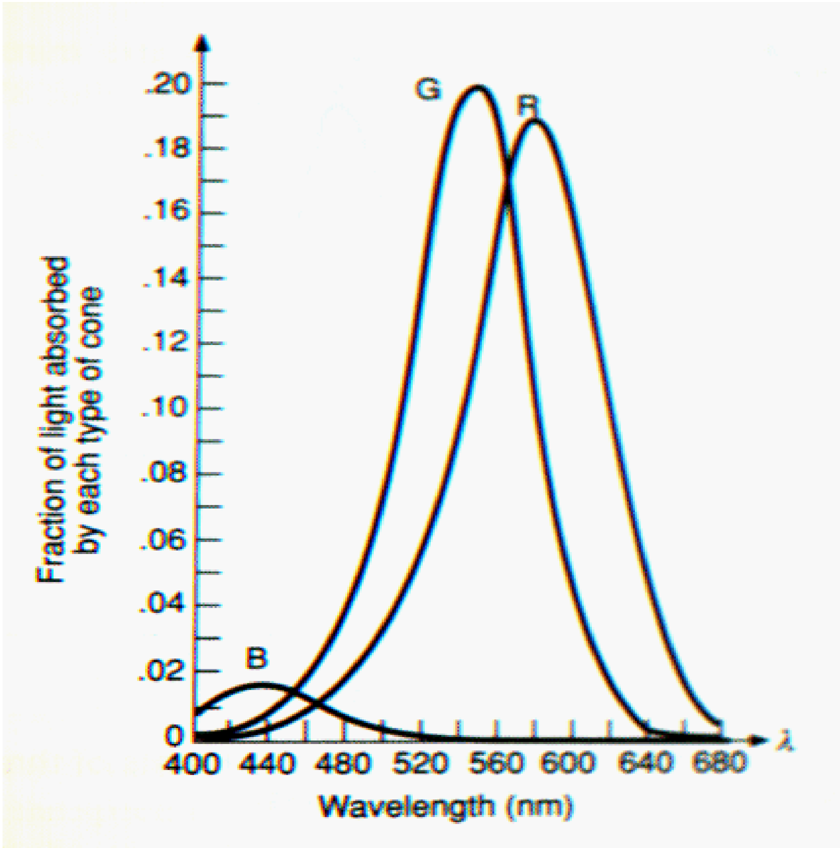
- Three types of cones:
  - **L** or **R**, most sensitive to red light (610 nm)
  - **M** or **G**, most sensitive to green light (560 nm)
  - **S** or **B**, most sensitive to blue light (430 nm)



- Color blindness results from missing cone type(s)

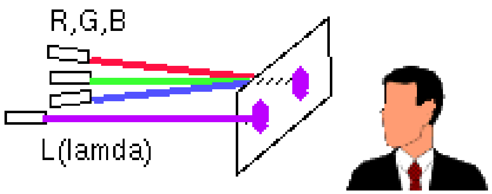
The Human Eye: Seeing Color

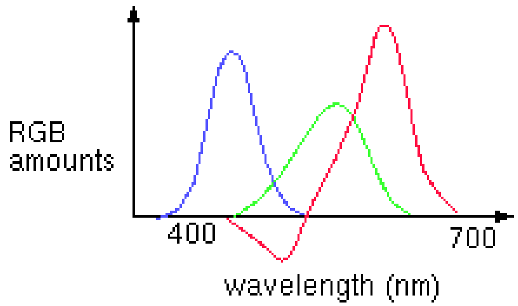
- The tristimulus curve shows overlaps, and different levels of responses
- Eyes more sensitive around 550nm, can distinquish smaller differences
- What color do we see best?
  - Yellow-green at 550 nm
- What color do we see worst?
  - Blue at 440 nm



Color Spaces

- Three types of cones suggests color is a 3D quantity.
- How to define 3D color space?
- Color matching idea:
  - shine given wavelength (λ) on a screen
  - Mix three other wavelengths (R,G,B) on same screen.
  - Have user adjust intensity of RGB until colors are identical:





Color Spaces

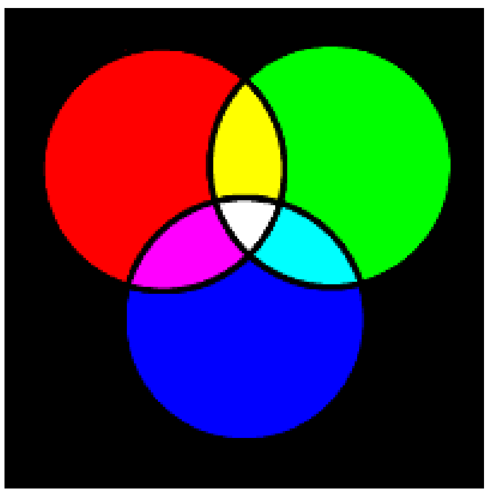
- Alternate lingo may be better for other domains
- **Artists:** tint, tone shade
- **Computer Graphics/Imaging:** Hue, saturation, luminance
- Many different color spaces
  - RGB
  - CMY
  - HLS
  - HSV Color Model
  - And more.....

Combining Colors: Additive and Subtractive

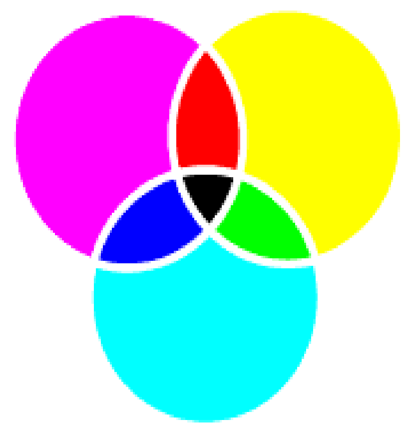
Add components

Remove components from white

Additive (e.g. RGB)



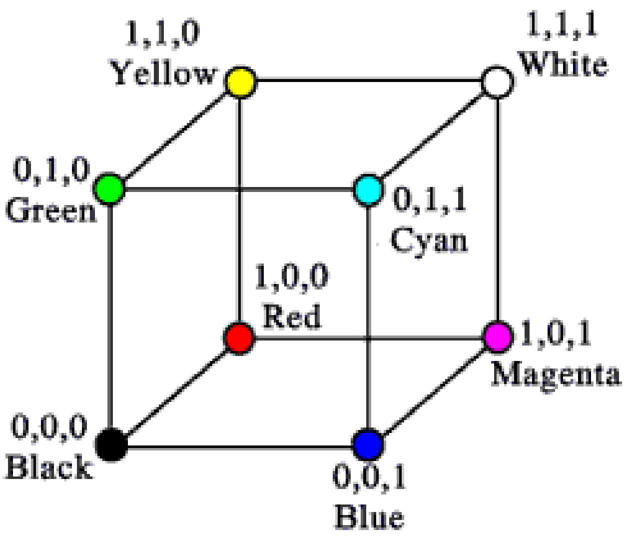
Subtractive (e.g gCMYK)



- Some color spaces are additive, others are subtractive
- **Examples:** Additive (light) and subtractive (paint)

RGB Color Space

- Define colors with (r, g, b) amounts of red, green, blue
- Additive, most popular
- Maximum value = 255 or 1.0 if normalized
- (0,0,0) = black, (1,1,1) = White
- Equal amounts of R,G, B = gray (lies on cube white-black diagonal)



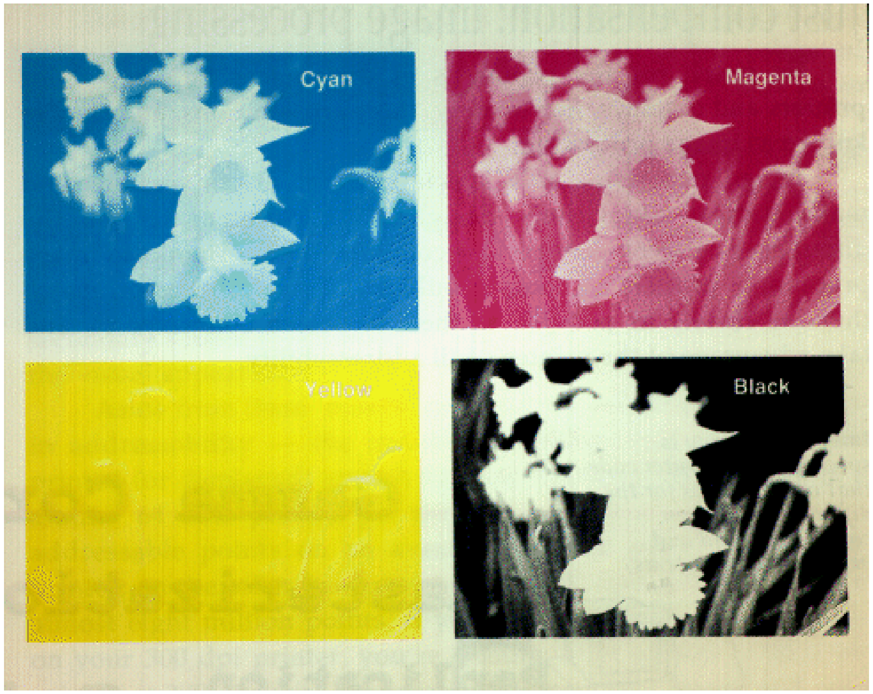
RGB Color Images

- RGB image is shown with corresponding RGB channels
- The fruits are mostly yellow and red hence high values in R and G channels
- Values in B channel are small except for bright highlights on fruit
- Tabletop is violet which contains higher values in B channel
- Most operations we have studied so far in grayscale can work on color images by performing operation on each channel (RGB)

R G B

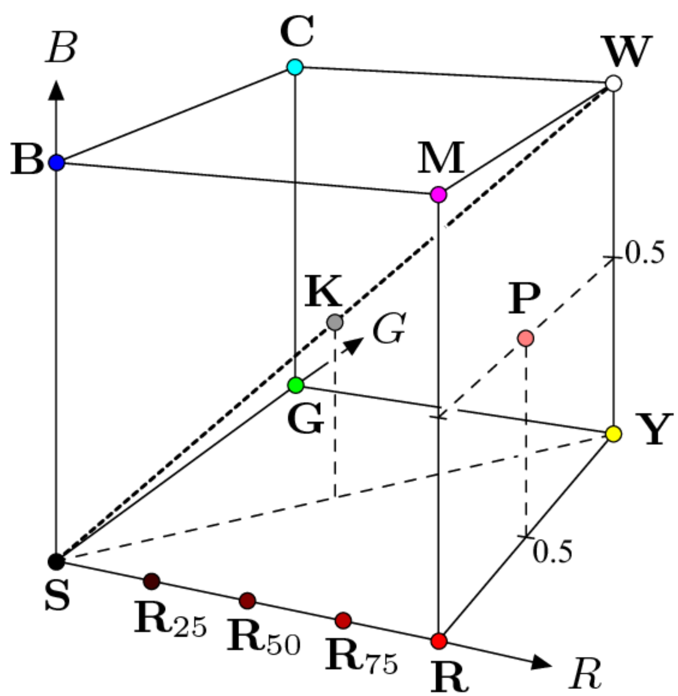
CMY

- Subtractive
- For printing
- Cyan, Magenta, Yellow
- Sometimes black (K) is also used for richer black
- (c, m, y) means subtract the compliments of C (red) M (green) and Y (blue)



RGB – CMY Relationship

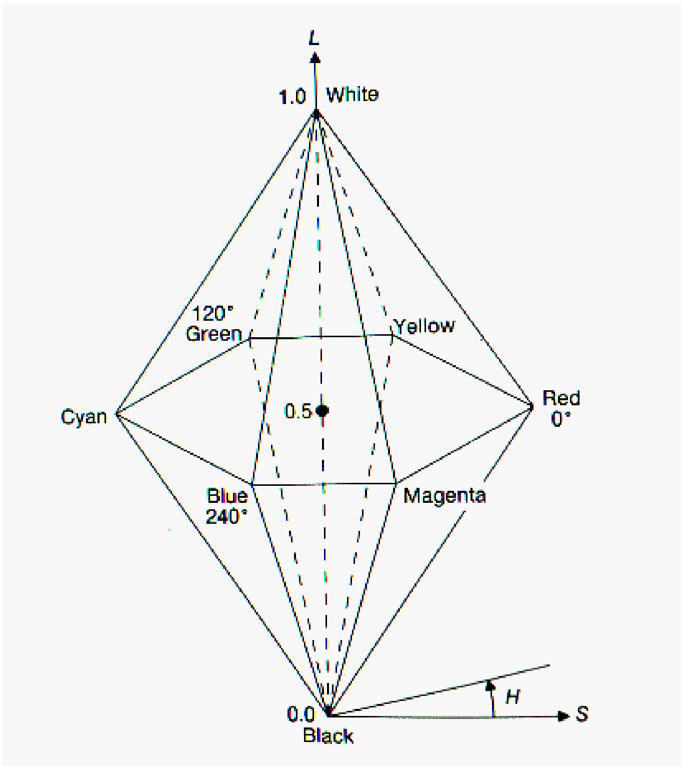
- Interesting to put RGB and CMY in same cube
- R,G,B and C,M,Y lie at vertices
- Perception of RGB may be non-linear



Point	Color	RGB Value R	G	B
S	Black	0.00	0.00	0.00
R	Red	1.00	0.00	0.00
Y	Yellow	1.00	1.00	0.00
G	Green	0.00	1.00	0.00
C	Cyan	0.00	1.00	1.00
B	Blue	0.00	0.00	1.00
M	Magenta	1.00	0.00	1.00
W	White	1.00	1.00	1.00
K	50% Gray	0.50	0.50	0.50
$\mathbf{R}_{75}$	75% Red	0.75	0.00	0.00
$\mathbf{R}_{50}$	50% Red	0.50	0.00	0.00
$\mathbf{R}_{25}$	25% Red	0.25	0.00	0.00
P	Pink	1.00	0.50	0.50

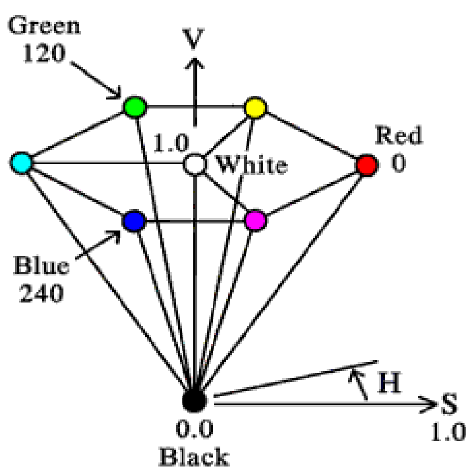
HLS

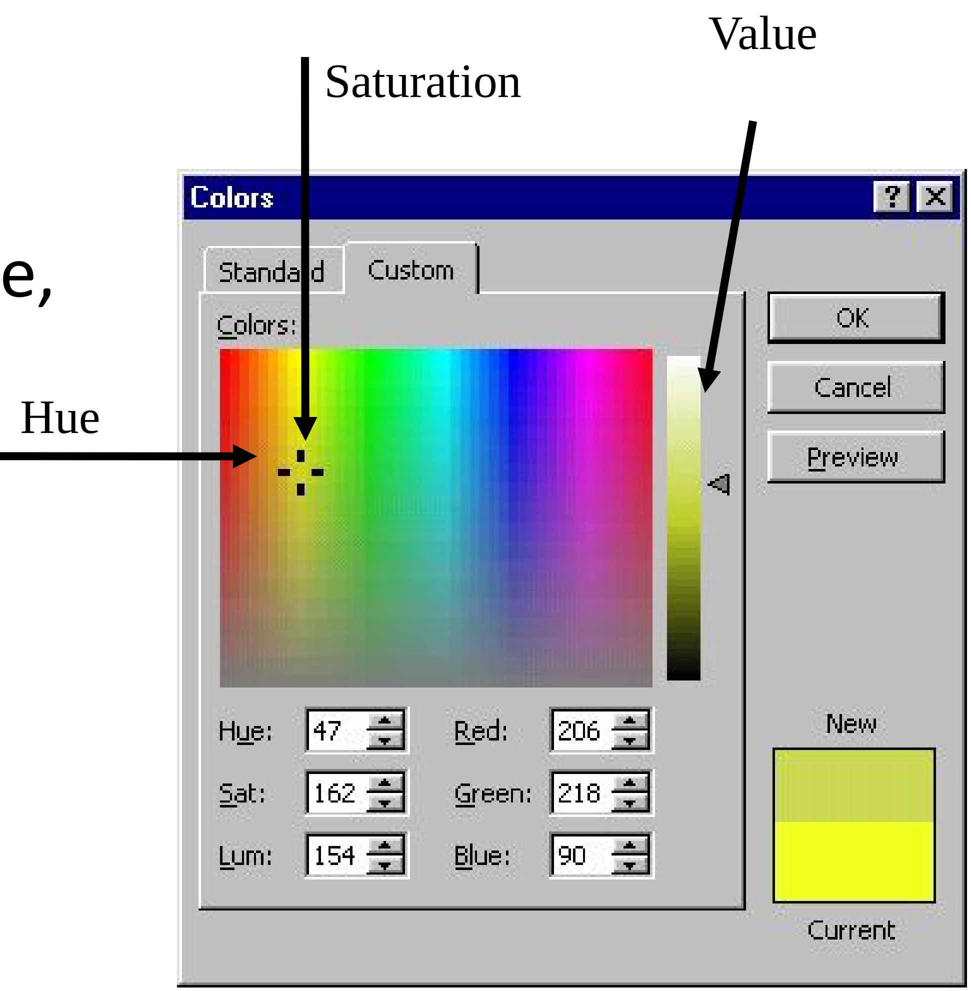
- Hue, Lightness, Saturation
- Based on warped RGB cube
- Look from (1,1,1) to (0,0,0) or RGB cube
- All hues then lie on hexagon
- Express hue as angle in degrees
- 0 degrees: red



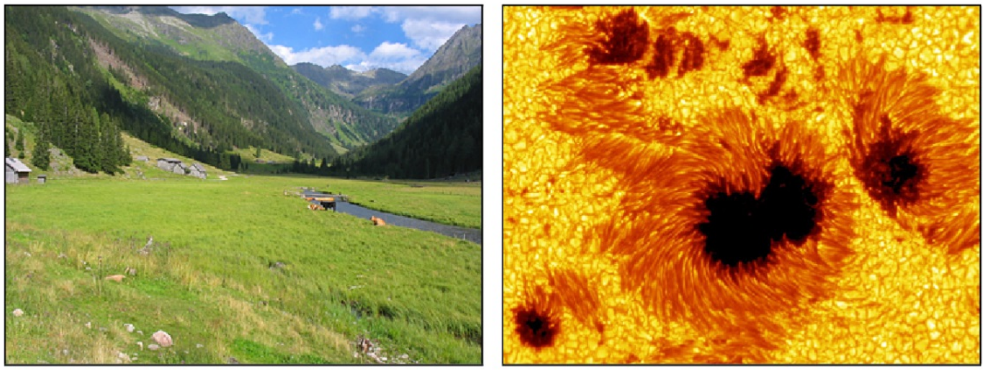
HSV Color Space

- More intuitive color space
  - H = Hue
  - S = Saturation
  - V = Value (or brightness)
- Based on artist Tint, Shade, Tone
- Similar to HLS in concept

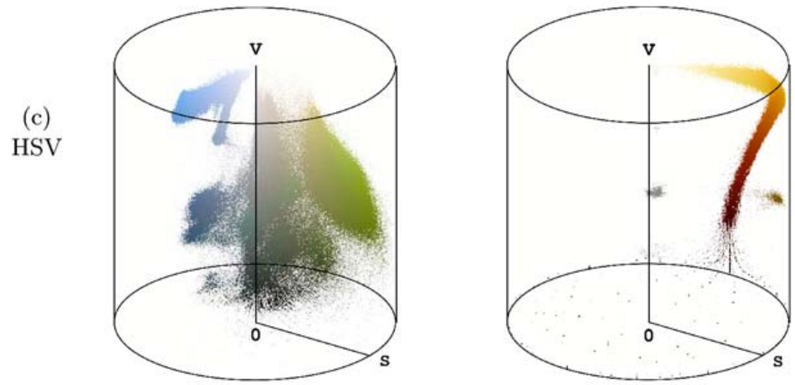




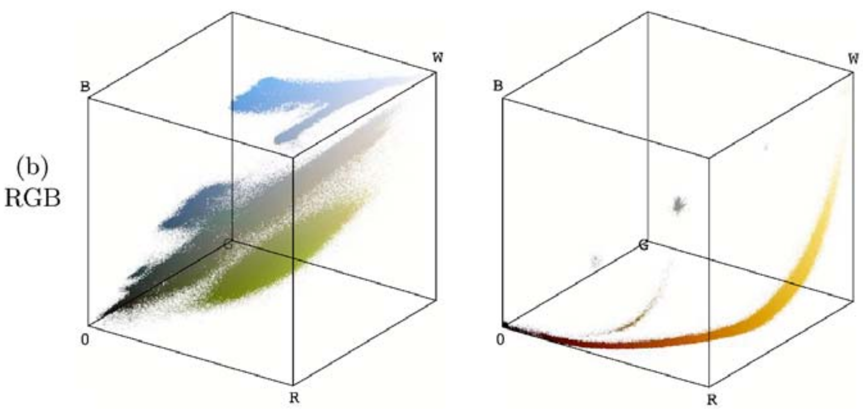
Examples of Color Distribution of Natural Images in 3 Color Spaces



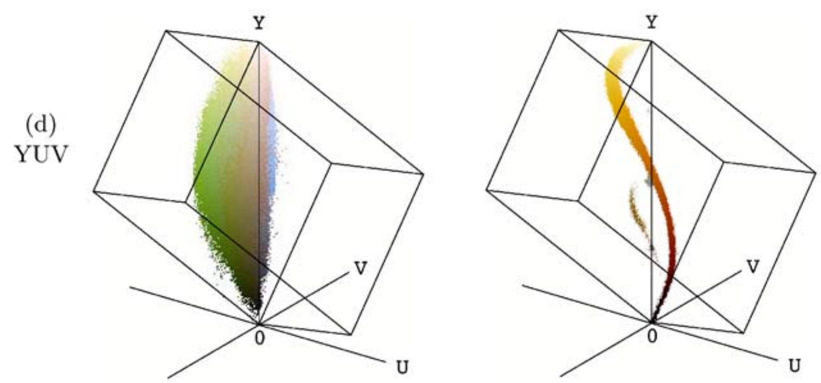
Natural scenes



HSV



RGB



YUV

Converting Color Spaces

- Converting between color models can also be expressed as such a matrix transform:

$$[R \quad G \quad B] = [X \quad Y \quad Z]\begin{bmatrix} 2.739 & -1.110 & 0.138 \\ -1.145 & 2.029 & -0.333 \\ -0.424 & 0.033 & 1.105 \end{bmatrix}$$

Conversion to Grayscale

- Simplest way to convert color to grayscale

$$Y = \mathrm{Avg}(R, G, B) = \frac{R + G + B}{3}$$

- Resulting image will be too dark in red and green areas
- Alternative approach is use a weighted sum of RGB

$$Y = \mathrm{Lum}(R, G, B) = w_R \cdot R + w_G \cdot G + w_B \cdot B$$

- Original weights for analog TV

$$w_R = 0.299 \qquad w_G = 0.587 \qquad w_B = 0.114$$

- Newer ITU weights for digital color encoding

$$w_R = 0.2125 \qquad w_G = 0.7154 \qquad w_B = 0.072$$

Hueless (Gray) Color Images

- An RGB image is hueless or gray when all components equal

$$R = G = B$$

- To remove color from an image
  - Use weighted sum equation to calculate luminance value Y
  - Replace R,G and B components with Y value

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} \leftarrow \begin{pmatrix} Y \\ Y \\ Y \end{pmatrix}$$

- In ImageJ, simplest way to convert RGB color image to grayscale is to use method **`convertToByte( boolean doscaling)`**
- **`convertToByte( boolean doscaling)`**uses default weights
- Can change weights applied using **`setWeightingFactors`**

Desaturating Color Images

- **Desaturation:** uniform reduction in amount of RGB
- **How?** Calculated the desaturated color by linearly interpolating RGB color and the corresponding (Y,Y,Y) gray point in RGB space

$$\begin{pmatrix} R_d \\ G_d \\ B_d \end{pmatrix} \leftarrow \begin{pmatrix} Y \\ Y \\ Y \end{pmatrix} + s_{\text{col}} \cdot \begin{pmatrix} R - Y \\ G - Y \\ B - Y \end{pmatrix}$$

- s_{col} takes values in [0,1] range

ImageJ Desaturation PlugIn

```
// File Desaturate_Rgb.java

import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class Desaturate_Rgb implements PlugInFilter {

  static double sCol = 0.3; // color saturation factor

  public int setup(String arg, ImagePlus im) {
    return DOES_RGB;
  }

  public void run(ImageProcessor ip) {

    // iterate over all pixels
    for (int v = 0; v < ip.getHeight(); v++) {
      for (int u = 0; u < ip.getWidth(); u++) {

        // get int-packed color pixel
        int c = ip.get(u, v);

        // extract RGB components from color pixel
        int r = (c & 0xff0000) >> 16;
        int g = (c & 0x00ff00) >> 8;
        int b = (c & 0x0000ff);

        // compute equivalent gray value
        double y = 0.299 * r + 0.587 * g + 0.114 * b;

        // linearly interpolate (yyy) ↔ (rgb)
        r = (int) (y + sCol * (r - y));
        g = (int) (y + sCol * (g - y));
        b = (int) (y + sCol * (b - y));

        // reassemble color pixel
        c = ((r & 0xff)<<16) | ((g & 0xff)<<8) | b & 0xff;
        ip.set(u, v, c);
      }
    }
  }

} // end of class Desaturate_Rgb
```

References

- Wilhelm Burger and Mark J. Burge, Digital Image Processing, Springer, 2008
- University of Utah, CS 4640: Image Processing Basics, Spring 2012
- Rutgers University, CS 334, Introduction to Imaging and Multimedia, Fall 2012
- Gonzales and Woods, Digital Image Processing (3rd edition), Prentice Hall
- Computer Graphics using OpenGL by F.S Hill Jr, 2nd edition, chapter 12