Chapter 3

色

3.1 色の原理と表現

これまで色について学んだことのない人にとって，色とは謎に満ちた存在である．色に関して大学初等までに学んだ事項には，光の波長に伴う色の違いと光の 3 原色や色の 3 原色と呼ばれる概念があるだろう．人間が目で見る光は，波としての性質を持っており，様々な波長の光が異なった色として知覚されることは，プリズムや虹などの現象から明らかである．となると，色とは光の波長の性質と言えるのだろうか？　一方で，光の 3 原色とは 3 種類の光の混合によって様々な色が再現できることを意味している．すなわち，プリズムとスリットを用いて分光した任意波長の 2 つの光を合成すると，新たな色の光になる．このことは 2 つの波長の光を混合すると，別の波長の光になることを意味しているのだろうか？

実際には，色は脳の働きによって生み出される視覚的な効果である．さらに，その視覚的な効果は，人間の目の構造に由来している．人間の目は図 3.1 のように，水晶体と呼ばれるレンズによって網膜上に結ばれた映像に対して**視細胞** (visual cell) が反応し，視神経を通して電気的な信号を脳に伝える器官である．視細胞の反応は，光エネルギーの強さだけではなく，波長によっても変化する．たとえば，可視光領域外の赤外光や紫外光などは，どんなに強いエネルギーを持っていたとしても，視細胞は反応しないため，人間には知覚されないのである．

この視細胞には，網膜上に広く荒く分布し弱い光に反応する桿体 (rod) と，黄斑部中心窩付近にのみ高密度に分布し，ある程度強い光に反応する錐体 (cone) がある．人間が対象を注視する場合には，網膜の中心部を用いた中心視が行われ，錐体が重要な役割を果たす．この錐体にはさらに 3 種類あり，それぞれ光の波長に対する反応が異なり，図 3.2 のように短い波長，中程度の波長，長い波長で，それぞれ強く反応する．通常，目で観察される光は複数の波長の光が集まったものであり，3 種類の錐体は，光の波長成分に応じた独立の反応をして，3 つの異なった刺激値を脳に伝えることになる．3 つの刺激値を脳が再構成して「色」として知覚しているのである．

この結果として，仮に波長成分の異なる光であったとしても，3 刺激値が等しくなる可能性がある．脳が受ける情報は，あくまでも 3 刺激値であるため，このような場合には波長の差異を区別することはできない．すなわち，これらの光は人間にとって等しい色として知覚されることになる．このように等しい色として知覚される光とその関係のことを**等色** (isochromatic) と呼ぶ．3 つの光から色を合成できるという光の 3 原色の本質は，この 3 刺激値による等色に由来するのである（物体の色は物体を照らす光源の色に依存するため，光源の性質に依存して 2 つの物体が等色することがある．このような等色を**条件等色** (metamerism) と呼ぶ）．

色の本質が視細胞から脳に伝わる 3 刺激値であることを考えると，色は 3 自由度を持つことがわかる．色を 3 つの値の組によって表現する方法を**混色系**と呼び，色見本など具体的な色の表によって表現する方法を**顕色系**と呼ぶ．顕色系は印刷や紙，絵の具などの素材の色を表す際に，よく用いられる．これに対して，計算機上で色を扱う場合には混色系が利用される．光の 3 原色や色の 3 原色は混色系の一種と考えることもできる．

加法混色は，光を原色とし，3 つの光の組によって色を表す方法である．もっとも利用されている加法混色系は，***RGB* 表現** (Red Green Blue) で光の 3 原色に相当するものである．光の無い状態が「黒」であり，そこに R, G, B の各成分を加えていくことで色が構成される．すべての成分が最大（値域が $[0, 1]$ の場合 1）となると，色は「白」になる．*RGB* 表現では，主な色は表 3.1 のように表現される．

減法混色は，フィルタを原色として，3 つのフィルタの組によって色を表す方法である．***CMY* 表現** (Cyan Magenta Yellow) は色の 3 原色に対応するもので減法混色系の一種である．フィルタの無い状態が「白」であ

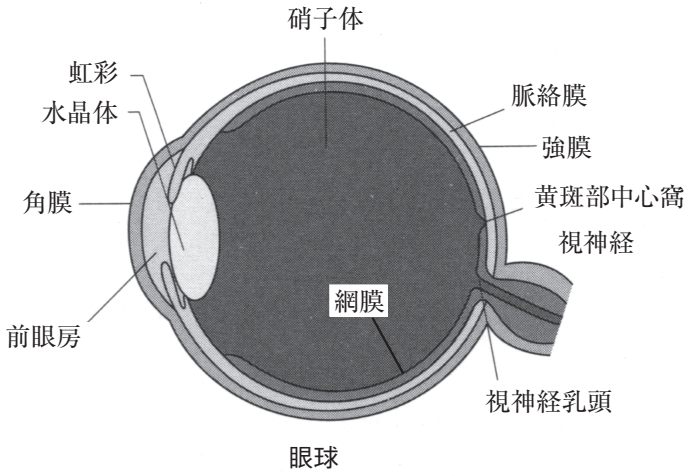

Figure 3.1: 人間の目の構造

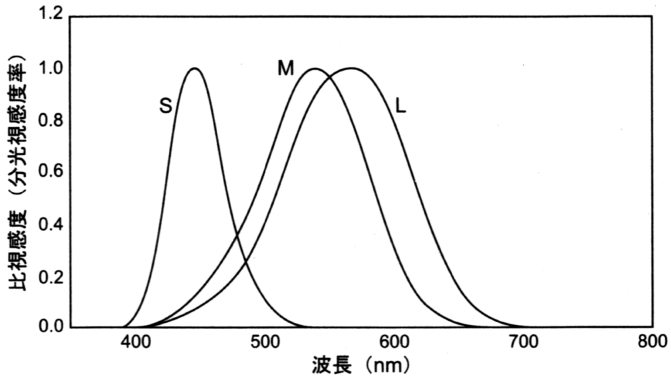

Figure 3.2: 錐体の光感受性

Table 3.1: 色の RGB 表現と CMY 表現

色	Color	(RGB) 成分	(CMY) 成分
赤	Red	(1 0 0)	(0 1 1)
緑	Green	(0 1 0)	(1 0 1)
青	Blue	(0 0 1)	(1 1 0)
シアン	Cyan	(0 1 1)	(1 0 0)
マゼンタ	Magenta	(1 0 1)	(0 1 0)
黄	Yellow	(1 1 0)	(0 0 1)
白	White	(1 1 1)	(0 0 0)
黒	Black	(0 0 0)	(1 1 1)

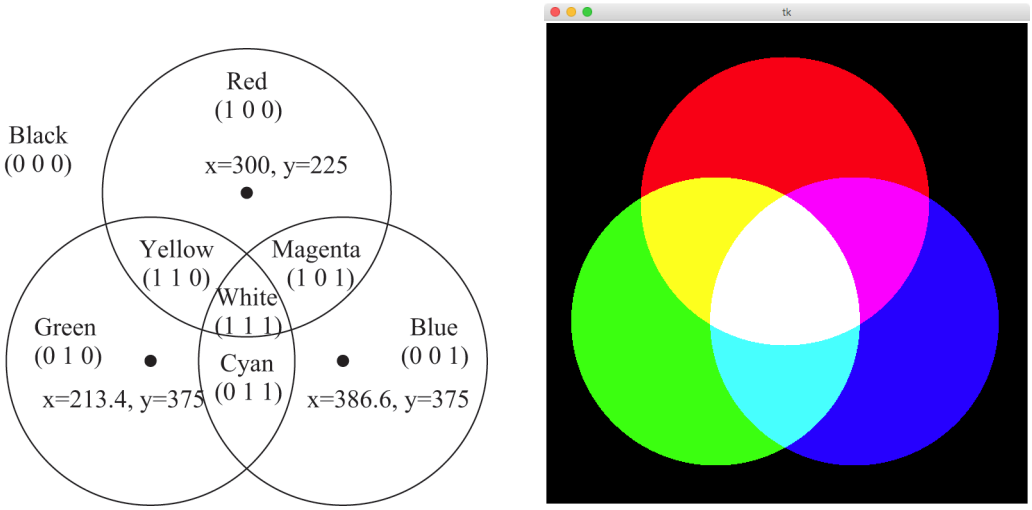

Figure 3.3: 光の 3 原色 (*RGB* 表現)

り，そこに C, M, Y の各成分を混ぜる (光の成分は減ぜられる) ことで色が構成される．すべての成分が最大 (値域が $[0, 1]$ の場合 1) になると，色は「黒」になる．*CMY* 表現では，主な色は表 3.1 のように表現される．*RGB* 表現と *CMY* 表現の間には，以下の関係式が成り立つ．

$$\begin{pmatrix} R & G & B \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} - \begin{pmatrix} C & M & Y \end{pmatrix} \tag{3.1}$$

***HSV*(*HSB*) 表現**は，H, S, V または H, S, B の 3 値によって色を表す方法で，それぞれ**色相** (Hue)，**彩度** (Saturation)，**明度** (Value/Brightness) を表している．*RGB* 表現と *HSV* 表現の変換は簡単な式では表現できない．詳細については，例 2 のプログラムを参照して欲しい．なお，この他にも様々な種類の色の表現形式があるが，Python には *RGB* 表現と 3 種類の色表現（具体的には *HSV*, *HLS*, *YIQ* 表現）とを変換するためのモジュール `colorsys` が用意されている．

3.2 Python による色のプログラム

Python の `tkinter` モジュールにおいて色は，24bit (= 8bit × 3) の *RGB* 表現で，16 進数 6 桁の文字列によって表現される．具体的には，`'#RRGGBB'` という形式の文字列で，`RR`，`GG`，`BB` が，それぞれ 16 進数による *RGB* 成分を表している．すなわち，各成分は 0（16 進数の 00）から 255（16 進数の ff）までの値をとり，黒であれば`'#000000'`，最も明るい赤であれば`'#ff0000'`，白であれば`'#ffffff'` となる．

例１： 光の 3 原色 — `additiveColor.py`

図 3.3 左のように，画面上に 3 つの円 (中心座標が $[300 \quad 225]^{\mathsf{T}}$, $[213.4 \quad 375]^{\mathsf{T}}$, $[386.6 \quad 375]^{\mathsf{T}}$，半径が 180) を考える．それぞれの円の内側に Red (1 0 0), Green (0 1 0), Blue (0 0 1) の 3 色を割り当て，円の重なった部分は単純に色を足し合わせる．すると，図 3.3 右のような，よく見かける「光の 3 原色」の絵が作られる．

```
from tkinter import *                          # tkinter モジュールの import

W, H = (600, 600)                              # canvas の幅と高さ

def display(canvas):                           # (加法混色円の) 描画関数
  '''
  canvas - 描画する canvas
  加法混色の円を描画する
  '''
  radius2 = 180**2                             # 半径の 2 乗
  centers = ((300.0, 225.0), (213.4, 375.0), (386.6, 375)) # 円の中心
```

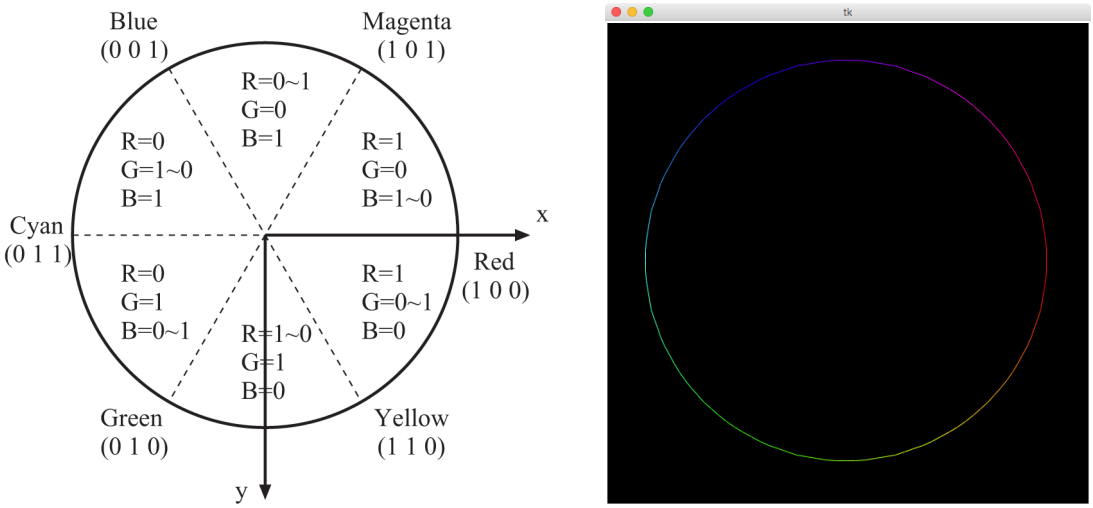

Figure 3.4: 色相と円

```
  for y in range(H):                          # ピクセルの反復（高さ分）
    for x in range(W):                        # ピクセルの反復（幅分）
      color = '#'                             # 色指定文字列の初期化
      for i in range(3):                      # RGB 値の指定
        dist2 = (x-centers[i][0])**2 + (y-centers[i][1])**2 # 中心からの距離の 2 乗
        color += '00' if dist2 > radius2 else 'ff' # 円の外部ならば 00，内部ならば ff
      canvas.create_rectangle((x, y), (x+1, y+1), outline='', fill=color)
                                              # 1 ピクセル (1x1 長方形) の描画
def main():                                   # main 関数
  root = Tk()                                 # ルートフレームの作成
  canvas = Canvas(root, width = W, height = H, bg='black') # canvas の作成
  canvas.pack()                               # canvas の配置確定
  display(canvas)                             # 描画関数（display）の呼出
  root.mainloop()                             # ルートフレームの実行ループ開始

if __name__ == '__main__':                    # 起動の確認（コマンドラインからの起動）
  main()                                      # main 関数の呼出
```

例２： 色相円 — `colorRingRGB.py`

第 2 章の例 3 の円を色相を変えて表示する．図 3.4 のように，円全体を 6 つの区間に区切って，徐々に色を変えていく．なお，このプログラムは第 2 章で作成した `circle` モジュール (`circle.py`) を利用している．

$$
\begin{array}{lcll}
\theta \in [0,\ \frac{\pi}{3}] & : & \text{Red}\ (\ 1\quad 0\quad 0\ ) & \rightarrow\ \text{Yellow}\ (\ 1\quad 1\quad 0\ ) \\
\theta \in (\frac{\pi}{3},\ \frac{2\pi}{3}] & : & \text{Yellow}\ (\ 1\quad 1\quad 0\ ) & \rightarrow\ \text{Green}\ (\ 0\quad 1\quad 0\ ) \\
\theta \in (\frac{2\pi}{3},\ \pi] & : & \text{Green}\ (\ 0\quad 1\quad 0\ ) & \rightarrow\ \text{Cyan}\ (\ 0\quad 1\quad 1\ ) \\
\theta \in (\pi,\ \frac{4\pi}{3}] & : & \text{Cyan}\ (\ 0\quad 1\quad 1\ ) & \rightarrow\ \text{Blue}\ (\ 0\quad 0\quad 1\ ) \\
\theta \in (\frac{4\pi}{3},\ \frac{5\pi}{3}] & : & \text{Blue}\ (\ 0\quad 0\quad 1\ ) & \rightarrow\ \text{Magenta}\ (\ 1\quad 0\quad 1\ ) \\
\theta \in (\frac{5\pi}{3},\ 2\pi] & : & \text{Magenta}\ (\ 1\quad 0\quad 1\ ) & \rightarrow\ \text{Red}\ (\ 1\quad 0\quad 0\ )
\end{array}
$$

```
from tkinter import *                         # tkinter モジュールの import
import circle                                 # circle モジュールの import

def f2hex(x):                                 # 16 進数文字列への変換関数
  '''
  x - [0, 1] の実数値
  00-ff の 16 進数 2 桁の文字列にして返す
  '''
```

```
  return '{:02X}'.format(int(x*0xff))    # 0xff 倍して 16 進数 2 桁の文字列にして返す

def string(r, g, b):                     # 色指定文字列の作成関数
  '''
  r, g, b - RGB の値, [0, 1] の範囲
  RGB 値を 16 進数 2 桁に対応した文字列にして返す
  '''
  return '#' + f2hex(r) + f2hex(g) + f2hex(b) # RGB 値を 16 進数 2 桁にして結合して返す

def color(n, i):                         # 色指定関数
  '''
  n - 頂点数
  i - 順番
  円周上で i/n の色指定文字列を作成して返す
  '''
  oneSixth, twoSixth, threeSixth, fourSixth, fiveSixth, six = \
            (1.0/6.0, 2.0/6.0, 3.0/6.0, 4.0/6.0, 5.0/6.0, 6.0)
                                         # 定数の定義
  ratio = i / n                          # 全体の比
  if ratio <= oneSixth:                  # 0 <= ratio <= 1/6
    return string(1.0, ratio*six, 0.0)
  elif ratio <= twoSixth:                # 1/6 < ratio <= 2/6
    return string((twoSixth-ratio)*six, 1.0, 0.0)
  elif ratio <= threeSixth:              # 2/6 < ratio <= 3/6
    return string(0.0, 1.0, (ratio-twoSixth)*six)
  elif ratio <= fourSixth:               # 3/6 < ratio <= 4/6
    return string(0.0, (fourSixth-ratio)*six, 1.0)
  elif ratio <= fiveSixth:               # 4/6 < ratio <= 5/6
    return string((ratio-fourSixth)*six, 0.0, 1.0)
  else:                                  # 5/6 < ratio <= 1
    return string(1.0, 0.0, (1.0-ratio)*six)

def display(canvas, points):             # (円の) 描画関数
  '''
  canvas - 描画する canvas
  points - 円周上の頂点列
  線分の色を変えながら円を描画する
  '''
  for i in range(len(points)):           # 線分描画の反復 (頂点数分)
    j = (i+1) % len(points)              # 次の頂点, 最終頂点の場合は最初の頂点
    canvas.create_line(points[i], points[j], fill=color(len(points), i))
                                         # 線分の描画 (color 関数による色指定)
def main():                              # main 関数
  root = Tk()                            # ルートフレームの作成
  canvas = Canvas(root, width = circle.W, height = circle.H, bg='black')
                                         # canvas の作成
  canvas.pack()                          # canvas の配置確定
  points = circle.circle()               # 頂点作成関数 (circle) の呼出
  display(canvas, points)                # 描画関数 (display) の呼出
  root.mainloop()                        # ルートフレームの実行ループ開始

if __name__ == '__main__':               # 起動の確認 (コマンドラインからの起動)
  main()                                 # main 関数の呼出
```

例3： *HSV* **表現と色円盤** — colorDisk.py

例2の円の内部も含めて色円盤を描画する．図3.5のように円の中心に近付くにしたがって彩度が落ちて白っぽくなり，中心では白色になる．*HSV* 表現を用いることで，色相と彩度の変化を簡単に計算できる．なお，*HSV* 表現から *RGB* 表現への変換には colorsys モジュールの hsv_to_rgb 関数を利用している．

```
from tkinter import *                    # tkinter モジュールの import
import math                              # math モジュールの import
import colorsys                          # colorsys モジュールの import
import circle                            # circle モジュールの import
from colorRingRGB import string          # colorRingRGB から string 関数の import
```

Figure 3.5: 色相と彩度

```
def display(canvas):                        # (彩度変化の色円盤の) 描画関数
  '''
  canvas - 描画する canvas
  色円盤を描画する
  '''
  center = (circle.W/2, circle.H/2)         # 中心
  radius = circle.R                         # 半径 (250)
  for y in range(circle.H):                 # ピクセルの反復 (高さ分)
    for x in range(circle.W):               # ピクセルの反復 (幅分)
      dx = x - center[0]                    # 中心からの x 座標の差分
      dy = y - center[1]                    # 中心からの y 座標の差分
      h = math.atan2(dy, dx) / (2 * math.pi) # 色相の計算
      h = h if h >= 0.0 else h + 1.0        # 色相を [0, 1] の範囲に収める
      s = (dx**2 + dy**2)**0.5 / radius     # 彩度の計算 (1 以上の場合あり)
      v = 0.0 if s > 1.0 else 1.0           # 明度の指定, 円の内部 1, 外部 0
      s = 0.0 if s > 1.0 else s             # 彩度を [0, 1] の範囲に収める
      r, g, b = colorsys.hsv_to_rgb(h, s, v) # HSV 値から RGB 値
      color = string(r, g, b)               # 色指定文字列の作成
      canvas.create_rectangle((x, y), (x+1, y+1), outline='', fill=color)
                                            # 1 ピクセル (1x1 長方形) の描画
def main():                                 # main 関数
  root = Tk()                               # ルートフレームの作成
  canvas = Canvas(root, width = circle.W, height = circle.H, bg='black')
                                            # canvas の作成
  canvas.pack()                             # canvas の配置確定
  display(canvas)                           # 描画関数 (display) の呼出
  root.mainloop()                           # ルートフレームの実行ループ開始

if __name__ == '__main__':                  # 起動の確認 (コマンドラインからの起動)
  main()                                    # main 関数の呼出
```

章末課題

色の 3 原色 (減法混色/CMY 表現)

例 1「光の 3 原色」のプログラムを参考に,「色の 3 原色」のプログラムを作成せよ．表示結果は，図 3.6 のようになる．この絵は「光の 3 原色 (RGB 表現)」の図 3.3 に酷似している．

ヒント： (3.1) 式を参考に，プログラムの変更は 5 行未満で済むはずである．

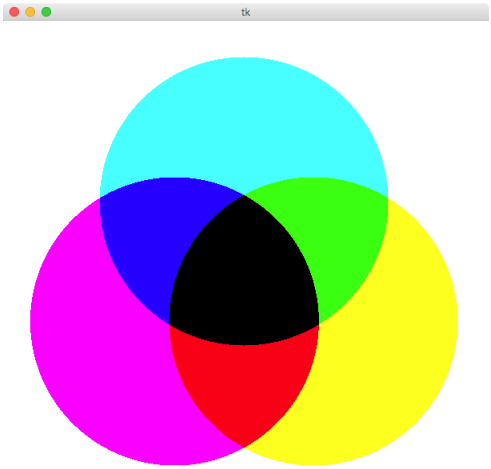

Figure 3.6: 色の 3 原色 (CMY 表現)

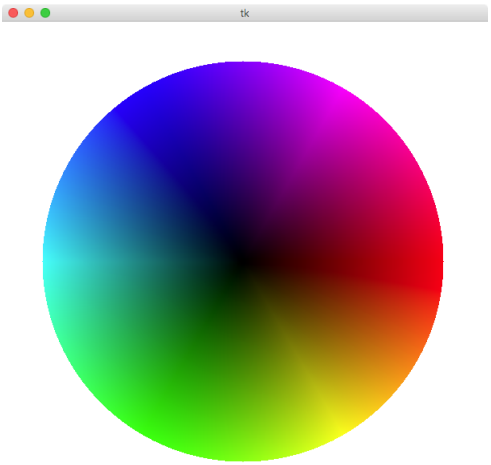

Figure 3.7: 明度を変化させた色円盤

HSV 表現を用いた色相円

色相円の円周上での色の変化は，色相の変化に他ならない．HSV 表現を用いる ($S = 1, V = 1$ に固定する) ことで，図 3.4 の色相円を簡単に描けるはずである．例 2「色相円」と例 3「HSV 表現と色円盤」のプログラムを参考に，HSV 表現を利用した色相円のプログラムを作成せよ．例 2「色相円」のプログラムである `colorRingRGB.py` の `color` 関数を書き換えるだけでよい．

明度を変化させた色円盤

例 3「HSV 表現と色円盤」のプログラムでは，内側に向かって彩度が落ちていき，中心で白色になっていた．ここでは図 3.7 のように，内側に向かって明度が落ちていき，中心で黒色になる色円盤のプログラムを作成せよ．