Project Report: Exploring Discrete Structures in Digital PhotographyHao Chen

December 8, 2024

Northeastern University

CS 5002

Contents

Introduction

When I was a junior in high school, I was already a Photoshop user, using it to create some images, and at that time I would get involved in the application of color, but honestly not a lot, I just knew that it was “color”. After graduating from university I got into photography and fell madly in love with it, and this time, “color” was inseparable from me. With the advent of digital cameras, taking a photo is no longer just a matter of adjusting the exposure and pressing the shutter. The coloring in lightroom directly determines the artistic style a photo wants to express. “Color” is no longer just ‘color’ to me, it is hue, it is saturation, it is contrast, it is brightness, it is curve, and also, it is #000000, #ffffff, #2fe963. When we learned about hexadecimal and RGB in the first class, it was familiar and unfamiliar to me because I didn't really know where the string of characters representing the colors came from, so I was extremely interested. So when choosing my research topic, it was natural for me to want to explore the application of discrete structures in the field of photography.

This project explores how mathematical principles like hexadecimal-to-decimal conversion and color model transformations (RGB, HSV, CMYK) are applied in digital photography. By developing a color palette tool, I aimed to better understand the application of hexadecimal in the field of color from the root.

Analysis

Reference object

If I want to explore the connection between hexadecimal and color, it is important to have a reference object for analysis. Since I use Photoshop a lot, I chose Photoshop's color selection function as a reference object, this is also what I want to make through this project, as shown in Figure 1. This interface has a square for selecting saturation (S) and value (V) , a bar for selecting hue, and a parameter area.

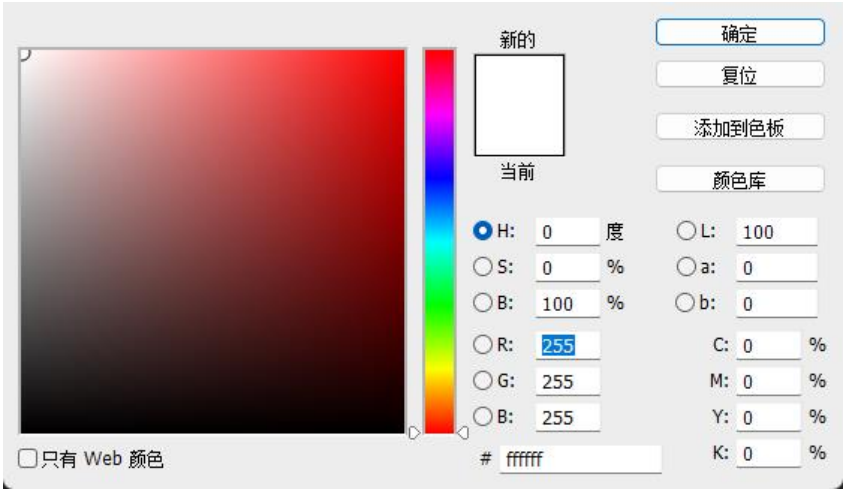

Figure 1. Choose Color in PhotoShop

Theoretical knowledge

First I need to explore hexadecimal and RGB conversions, which we studied in the first lesson. I reviewed how hexadecimal numbers represent RGB colors. For instance, #ffffff represents white, and #000000 represents black. This involved understanding the base-16 numbering system and converting it into decimal values

Second is the understanding of RGB, HSV, CMYK these primary color models. First of all, I looked up the information of these three primary color models, and had a deep understanding of them. RGB is the three primary colors of red, green and blue, and HSV is the first time I heard about it, but after understanding it, I found that I have been using it all the time. H is the hue, which can be understood as choosing a color, S is the saturation, V is the brightness, and some people use B to express it. CMYK is more interesting, this is the color used in the printing industry, when the printer needs to print color content, the printer from the CMYK four ink cartridges to adjust the required color.

Figure 2. CMYK Ink for Printer

RGB to CMYK

$R' = R/255$

$G' = G/255$

$B' = B/255$

$K = 1-\max(R', G', B')$

$C = (1-R'-K) / (1-K)$

$M = (1-G'-K) / (1-K)$

$Y = (1-B'-K) / (1-K)$

CMYK to RGB

$R = 255 \times (1-C) \times (1-K)$

$G = 255 \times (1-M) \times (1-K)$

$B = 255 \times (1-Y) \times (1-K)$

Figure 3. RGB and CMYK convert

I also found the conversion formulas between these color models, Python's colorsys library will help with the conversion between RGB and HSV, but RGB and CMYK are my custom functions based on the formulas

GUI Development with Tkinter

My reference target was the color selection function in PhotoShop, this interface is very convenient and intuitive for me to use, it shows H,S,V very well and can input R,G, B to get colors so I modeled my GUI after it. I created an interactive graphical user interface (GUI) using tkinter. This interface includes:

- A hue bar to select a base color.
- A color square to adjust saturation and value.
- A preview panel displaying the chosen color.
- Fields to manually input RGB values and see corresponding color updates.
- A button to display color model conversions in a popup window.

Hue Bar

Set both S and V to 1 and draw them line by line through the Canvas function of Tkinter to achieve a gradient process. and hue varied from 0° (red) to 360° (back to red).

```
def draw_hue_bar(self):  1 usage
    for y in range(self.hue_bar_height):
        h = y/(self.hue_bar_height - 1)
        # This is a hue bar, so set s and v equal to 1.0
        hex_color = self.hsv_to_hex(h, s: 1.0, v: 1.0)
        # Draw line by line
        self.hue_canvas.create_line(0,y,self.hue_bar_width,y, fill = hex_color)
```

Figure 4. Code to Draw Hue Bar

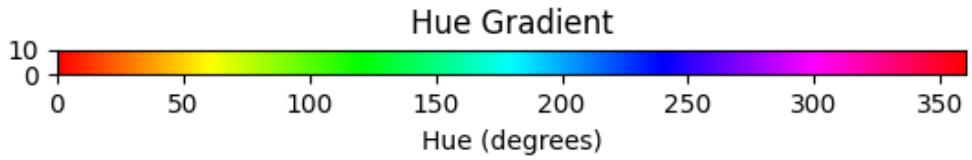

Figure 5. visualization of Hue Bar(by matplotlib)

Color Square

The square adjusts saturation (x-axis) and value (y-axis) for the selected hue, the saturation increases from left to right gradient (0-1), and the brightness increases from bottom to top gradient (0-1). Each pixel is colored using its corresponding HSV value, converted to RGB via colorsys. This gives user a more intuitive view of the difference between different S and different V. This function is also drawn line by line to achieve gradient.

```
def draw_color_square(self):  3 usages
    self.color_canvas.delete("all")
    # In vertical, v decrease from top to bottom
    for y in range(self.square_size):
        v = 1 - (y/(self.square_size-1))
        # In horizon, s increase form left ro right
        for x in range(0, self.square_size):
            s = x / (self.square_size -1)
            hex_color = self.hsv_to_hex(self.hue, s, v)
            self.color_canvas.create_rectangle(x,y,x+1, y+1, outline = hex_color,fill = hex_color)
```

Figure 6. Code to Draw Color Square

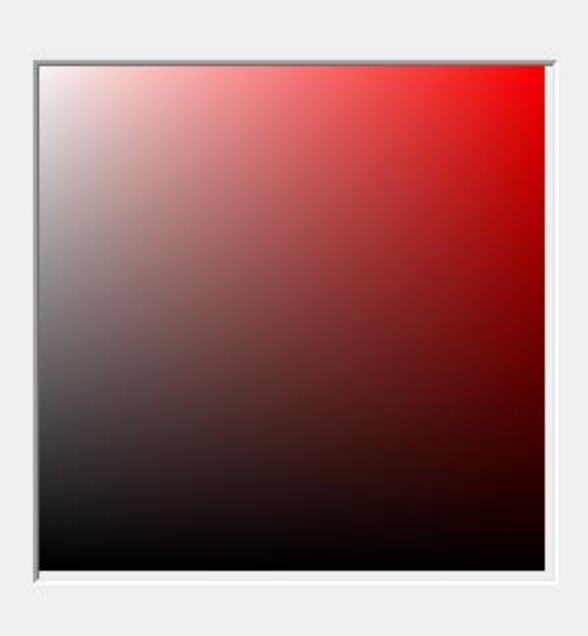

Figure 7. Visualization of Color Square

Color Display

The essence of this area is to display the color selected by the user. There are two ways to obtain the color, one is to click the color on the color palette and the other is to manually input the RGB value. The logic here is to capture the coordinates of the user's mouse click, and then obtain its corresponding color according to the coordinates, and then display it in this area. As for entering RGB to get the color, I set up a button to add functionality to this button to get the color based on the captured value, and then update it in the display area

Convert Button

Through the obtained formula and colorsys built-in function, to achieve the conversion between RGB, HSV, CMYK, and then through the messagebox to pop up a separate window to display these information

Python Libraries Used

tkinter: For GUI development.

colorsys: For RGB and HSV transformations.
math: For general mathematical operations.
matplotlib: For visualizations.

Conclusion

Through this project, I explored the mathematical foundations of digital photography, specifically focusing on color theory and its connection to Discrete Structures. By implementing a tool that visualizes color transformations and enables interaction with different color models, I achieved my goal of understanding how mathematical concepts drive practical applications in photography. The conversion between various color models is really amazing, but no matter what kind of model, the final need to output into color must be converted into hexadecimal numbers, by writing these codes, I undoubtedly have a further understanding of the application of hexadecimal in color. Although my color palette is made, it is only made, and there is still a long way to go from a good use experience. It may be that there are too many pixels to draw, the color square will load slowly, and there is a significant delay when clicking the hue bar, about a second or two or so, it may be that the program is not simple enough, there are too many things to calculate, and there is still a lot of room for optimization.

Secondly, the function of this color palette is not comprehensive, many mature works will provide some commonly used colors as options, so that users do not have to deploy their own, and lack of contrast between old and new colors. These are some of the limitations and disadvantages.

As for the future improvement direction, I found that there are LAB color models on PS that can be used, which I have not learned yet, and LAB can also be added in the future. It's also important to keep the program running smoothly when adding content.

My initial purpose is to explore the application of discrete structure in related aspects of photography. Pure color is a very small aspect, and there are many other aspects that can be studied, such as light and dark tone, sensor sensitivity, image or pixel arrangement and combination and other algorithms, which are all worthy of study.

Personal Reflection

My biggest gain from this project is to understand how hexadecimal is actually applied to color, which is different from just learning theoretical knowledge, and I will be more impressed when I participate in it. There is no doubt that this experience is very valuable to me, not only to explore the aspects that I am interested in, but also no matter what the result is, the color palette I have made has reached the effect I want, and I think it is a great sense of accomplishment to make it. Just as there is a very classic saying in the landscape and ecological photography which very much in line with my evaluation

of my project: “Capturing the moment is always more important than capturing it perfectly.”

Source Code and Result Display

```
import tkinter as tk
from tkinter import messagebox
import colorsys

class ColorPalette(tk.Frame):
    def __init__(self, master = None):
        super().__init__(master)
        self.master.title("Color Palette")

        # Frames, left: Hue, mid: square, right: RGB number
        self.left_frame = tk.Frame(self.master)
        self.mid_frame = tk.Frame(self.master)
        self.right_frame = tk.Frame(self.master)

        self.left_frame.pack(side="left", padx=10, pady=10)
        self.mid_frame.pack(side= "left", padx=10, pady=10)
        self.right_frame.pack(side="right", padx=10, pady=10)

        # Size set, 256 for later use
        self.square_size = 256
        self.hue_bar_height = 256
        self.hue_bar_width = 30

        # HSV values
        self.hue = 0.0
        self.saturation = 1.0
        self.value = 1.0

        # Hue canvas
        self.hue_canvas = tk.Canvas(self.left_frame, width = self.hue_bar_width,
height = self.hue_bar_height, bd= 3)
        self.hue_canvas.pack()
        self.hue_canvas.bind("<Button-1>", self.hue_click)

        # Draw the hue bar
        self.draw_hue_bar()

        # Color square
```

```
        self.color_canvas = tk.Canvas(self.mid_frame, width = self.square_size, height=self.square_size,bd = 3, relief = "sunken")
        self.color_canvas.pack()
        self.color_canvas.bind("<Button-1>", self.canvas_click)

        # Draw the square
        self.draw_color_square()

        # Color Display (user picked)
        self.color_display = tk.Canvas(self.right_frame, width = 50, height = 50, bd = 3, relief = "sunken")
        self.color_display.pack(padx = 10, pady = 10)

        # RGB Entries
        self.r_var = tk.StringVar(value = "255")
        self.g_var = tk.StringVar(value = "0")
        self.b_var = tk.StringVar(value = "0")

        # Label set
        tk.Label(self.right_frame, text = "R").pack(anchor = "w")
        self.r_entry = tk.Entry(self.right_frame, textvariable=self.r_var, width = 5)
        self.r_entry.pack(anchor = "w")

        tk.Label(self.right_frame, text="G").pack(anchor="w")
        self.g_entry = tk.Entry(self.right_frame, textvariable=self.g_var, width=5)
        self.g_entry.pack(anchor="w")

        tk.Label(self.right_frame, text="B").pack(anchor="w")
        self.b_entry = tk.Entry(self.right_frame, textvariable=self.b_var, width=5)
        self.b_entry.pack(anchor="w")

        self.update_button = tk.Button(self.right_frame, text="Update Color", command=self.update_color_from_rgb)
        self.update_button.pack(padx = 10,pady = 10)

        # Conversion button
        self.conversion_button = tk.Button(self.right_frame, text="Conversions", command= self.show_conversion)
        self.conversion_button.pack(padx = 10, pady = 10)

        # Show hex
        self.hex_var = tk.StringVar(value="#ffffff")
        tk.Label(self.right_frame, text="Hex:").pack(anchor="w")
        self.hex_label = tk.Label(self.right_frame, textvariable=self.hex_var,
```

```
width=10)
        self.hex_label.pack(anchor="w", pady=5)

        # Initial color display
        self.update_color_display()


    def hsv_to_hex(self, h, s, v):
        r, g, b = colorsys.hsv_to_rgb(h, s, v)
        return f'#{int(r * 255):02x}{int(g * 255):02x}{int(b * 255):02x}'

    # The left frame
    def draw_hue_bar(self):
        for y in range(self.hue_bar_height):
            h = y/(self.hue_bar_height - 1)
            # This is a hue bar, so set s and v equal to 1.0
            hex_color = self.hsv_to_hex(h,1.0,1.0)
            # Draw line by line
            self.hue_canvas.create_line(0,y,self.hue_bar_width,y, fill = hex_color)

    # The mid frame, a square to change saturation and value for the selected hue
    def draw_color_square(self):
        self.color_canvas.delete("all")
        # In vertical, v decrease from top to bottom
        for y in range(self.square_size):
            v = 1 - (y/(self.square_size-1))
            # In horizon, s increase form left ro right
            for x in range(0, self.square_size):
                s = x / (self.square_size -1)
                hex_color = self.hsv_to_hex(self.hue, s, v)
                self.color_canvas.create_rectangle(x,y,x+1,  y+1,  outline  =
hex_color,fill = hex_color)

    # Click to choose hue
    def hue_click(self, event):
        y = event.y
        if 0 <= y < self.hue_bar_height:
            self.hue = y / (self.hue_bar_height - 1)
            self.draw_color_square()
            self.update_color_display()

    # Click to set s and v for hue
    def canvas_click(self, event):
        x,y = event.x, event.y
```

```
            # Check the area of Square
            if 0 <= x <= self.square_size and 0<= y <= self.square_size:
                self.saturation = x / (self.square_size -1)
                self.value = 1 - y/(self.square_size -1)
                self.update_color_display()

        # Update the color display
        def update_color_display(self):
            r,g,b = colorsys.hsv_to_rgb(self.hue,self.saturation, self.value)
            R, G, B = int(r*255), int(g*255), int(b*255)
            hex_color = f'#{R:02x}{G:02x}{B:02x}'
            self.color_display.delete("all")
            self.color_display.create_rectangle(0,0,50,50, fill = hex_color, outline =
hex_color)

            # Update the number in Entry
            self.r_var.set(str(R))
            self.g_var.set(str(G))
            self.b_var.set(str(B))

            # Update hex label
            self.hex_var.set(hex_color)
            self.hex_label.config()

        # Allow user to entry R G B and update
        def update_color_from_rgb(self):
            try:
                R = int(self.r_var.get())
                G = int(self.g_var.get())
                B = int(self.b_var.get())

                # If user entry over 255 or less than 0, change it to 255 or 0
                R = max(0, min(R, 255))
                G = max(0, min(G, 255))
                B = max(0, min(B, 255))

                self.r_var.set(str(R))
                self.g_var.set(str(G))
                self.b_var.set(str(B))

                r = R / 255.0
                g = G / 255.0
                b = B / 255.0
```

```
                h, s, v = colorsys.rgb_to_hsv(r, g, b)
                self.hue = h
                self.saturation = s
                self.value = v

                # Redraw hue bar selection (no direct indicator, but hue changed)
                # Redraw the square with the new hue
                self.draw_color_square()
                self.update_color_display()
            except ValueError:
                messagebox.showerror("Invalid input", "Please enter integer values for
R, G, and B.")

    # Convert RGB to CMYK (By formula)
    def rgb_to_CMYK(self, R, G, B):
        r = R / 255.0
        g = G / 255.0
        b = B / 255.0

        k = 1 - max(r, g, b)

        c = (1 - r - k) / (1 - k)
        m = (1 - g - k) / (1 - k)
        y = (1 - b - k) / (1 - k)

        return (c, m, y, k)

    # Show HSV and CMYK based on current RGB
    def show_conversion(self):
        #Same as previous
        R = int(self.r_var.get())
        G = int(self.g_var.get())
        B = int(self.b_var.get())

        R = max(0, min(R, 255))
        G = max(0, min(G, 255))
        B = max(0, min(B, 255))

        r = R / 255.0
        g = G / 255.0
        b = B / 255.0

        # RGB TO HSV
        h, s, v = colorsys.rgb_to_hsv(r, g, b)
```

```
            # 0° to 360°  (Red circle back to Red)
            h_deg = h * 360

            # RGB to CMYK
            c, m, y, k = self.rgb_to_CMYK(R, G, B)

            details = (
                f"Current RGB: ({R}, {G}, {B})\n\n"
                f"HSV: (H={h_deg:.2f}°, S={s:.2f}, V={v:.2f})\n\n"
                f"CMYK: (C={c:.2f}, M={m:.2f}, Y={y:.2f}, K={k:.2f})"
            )

            messagebox.showinfo("Color Conversions", details)

if __name__ == "__main__":
    main_window = tk.Tk()
    palette = ColorPalette(master = main_window)
    palette.mainloop()
```

Result Display:

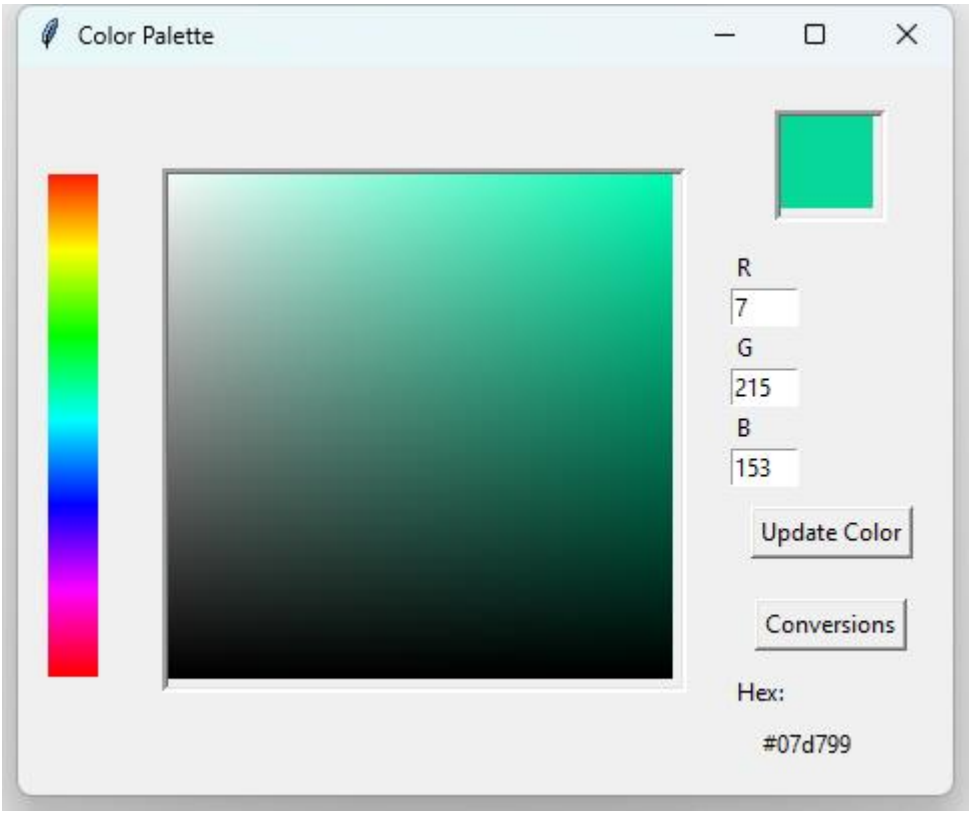

Figure 8. My Color Palette