Color Model Manipulation Using Python Imaging Library

Team Members:

Casey Kuball

Glenn Craver

Steve Carr

Wesley Rose

David Rux

**Abstract**:

Many programs exist today that allow simple image modifications but few allow the manipulation of an image’s color model. This project was designed around the main goal of editing an image’s color data and observing the effects that ensue. This was to be accomplished using a third party image editing API in any language of choice. Using the Python Imaging Library API and the documentation provided to the group, the program was implemented in Python and performs all of the necessary functions as dictated in the assignment’s requirements. By completing this project, one can directly observe the effects of editing an image’s color model and pixel color data.

**Keywords**:

ImageMagick

Wand API

Python 2.7.3

Python Imaging Library (PIL)

RGB

XYZ

Lab

YUV

YCbCr

YIQ

HSL

Color model

Color space

Median-Cut Algorithm

**Introduction**

* Terminology

Throughout this report, the various color models will be referenced using their acronym, which traditionally describes the major components in the color model. When the term ‘API’ is referenced, it is used to denote Application Programming Interface. The only API used in this project is the Python Imaging Library, which provide similar functionality to the features built into the ImageMagick API’s, except PIL provided a superior set of documentation, as well as a larger library of imaging modules.

* Goal

The goal of this project is to implement a program which utilizes a 3rd party imaging API and implements the following functionality when supplied an image:

* + Divide the image into a 6x6 grid
  + Print average color instance values for each cell in the grid for each of the color space listed in the Keywords portion above
  + Increase or reduce the saturation of a row of cells while still maintaining the overall energy of the cell
  + Find a cell with the most similar average color given a color space and reference cell
  + Create seven different versions of a chosen cell, convert each version to a different color space, and reduce the number of colors in each cell to a number provided by the user
  + Create seven different versions of a chosen cell, convert each version to a different color space, and highlight the value of the third component where the value is in the 80th percentile
* Assumptions

The program assumes that the user is aware of the functions that are

implemented and has previous knowledge of the various color models as well as terms such as “saturation”.

**Proposed Solution**

The solution to the goals and requirements was to implement a program utilizing the Python Imaging Library. Given an image provided by the user, the program will perform the functions described in the ‘Goal’ portion above and produce a series of images as output. The program is run locally and the pathname to the image is supplied by the user at runtime, as well as any additional arguments that are needed (such as the number of colors to reduce the image to).

When initializing the program it will automatically partition the supplied image into a 6x6 grid and store the subsequent images using row-major order. e.g.,\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \end{bmatrix}

All five of the user selectable options are saved into their own respective classes with a sixth class storing the color model conversion equations, which are listed in appendix A1.

In order to preserve the overall energy in the image while increasing or decreasing the saturation the following steps were taken in a linear lock-step fashion:

1. Convert the image to HSL
2. Increase luminance without losing energy
   1. multiply saturation component by 1.1 (increase) or 0.9 (decrease)
   2. preserve overall energy
      1. energy = sqrt(H^2 + S^2 + L^2)
   3. alter luminance to keep energy consistent
3. Convert the image back to RHB

To reduce the number of color instances in an image to N the median-cut algorithm was utilized; listed in appendix A2. By doing so a user can specify a value, N, in which the median-cut algorithm will find the N colors that best match the original number of color instances most accurately, thereby reducing the color instances in the image to N.

The printed average color instance values for each of images in the 6x6 grid is displayed as a fractional portion of the max RGB color space. For example, (1, 1, 1) is white and (0, 0 , 0) is black. Once the average RGB of the image is found it is then converted to the user specified color model and listed for each image in the 6x6 grid; example shown in appendix A3.

Depending on the user option selected the program will store any resulting modified images to a newly created folder labeled “Output” located in the same directory as the supplied image passed to the main.py program.

**Interface Specifications**

Initially, our team was using the Wand API binding to implement the ImageMagick image processing libraries, however after discovering the severe lack of documentation we decided to switch to the Python Imaging Library. By doing so we were able to implement several of the Wand functionalities with superior performance and with a better understanding of the PIL capabilities. Aside from the larger support base that PIL provided it also contained a substantial amount of functions that Wand did not contain nor support.

**System Requirements**

The system requirements are as follows:

* + Python 2.7.3 (This is the latest stable release)
  + Python Imaging Library 1.1.7
  + Mac OS X 10.6+, Windows XP or Later (any OS with Python is suitable but installation instructions are only given for Windows)
  + Root permission level to change Path variables

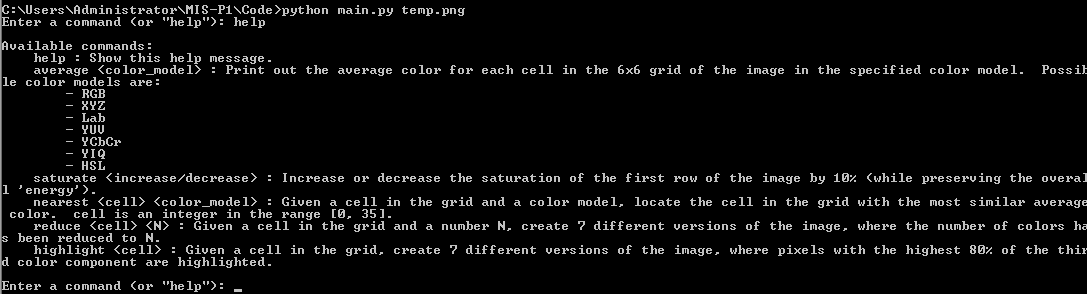
**Installation Instructions**

1. Download and install the x86 (32-bit) version of python 2.7 (<http://www.python.org/download/>).
2. Add the python installation directory (e.g. C:\Python27) to your $PATH

environmental variable.

1. Download and install the latest version of PIL for Python 2.7 (<http://www.pythonware.com/products/pil/>).

**Execution Instructions**

1. Open up a DOS prompt and navigate to the folder where the program is located
2. Type ‘python main.py <Image File>’ to run the program
3. 
4. After starting the program the user can enter 5 different commands with there respective arguments.
5. Entering “help” will list the commands and the arguments they take.

**Related Work**

The color models used in this project consisted of RGB, XYZ, Lab, YUV, YCbCr,

YIQ and HSL. In order to transform one color model into another we relied on the textbook “*Fundamentals of Multimedia*”, which defined the algorithms needed in order to convert between color spaces. In order to best visualize the differences in the various color models the following image displays the color space of RGB compared to CMYK:

The Python Imaging Library that we used to implement this program has been used extensively in the open source community on a variety of projects, several of which can be found through a quick query on Google or stack overflow.

The median cut algorithm implemented in our program is a popular method used for color quantization. There are several different applications which use median cut one example being the Perl module Image-Pngslimmer.

**Conclusions**

By doing this project, the group has learned the intricate differences between the various color models presented in the requirements. While our program only addresses some of the features of the Python Imaging Library, there are many more functions that can be implemented through future work. This project was beneficial in that it exposed how important multimedia objects and methodologies are in regards to the study of computer graphics, image processing and computer vision.

**Bibliography**

"Convert, Edit, and Compose Images." *ImageMagick: Convert, Edit, Or Compose Bitmap Images*. ImageMagick Studio LLC, n.d. Web. 20 Sept. 2012. <http://www.imagemagick.org/script/index.php>.

Drake, Fred L., Jr. "Python V2.7.3 Documentation." *Overview â”*. Python Software Foundation, 19 Sept. 2012. Web. 20 Sept. 2012. <http://docs.python.org/>.

"Python Imaging Library Handbook." *Python Imaging Library Handbook*. Pythonware.com,

6 May 2005. Web. 20 Sept. 2012. <http://www.pythonware.com/library/pil/handbook/index.htm>.

Li, Ze-Nian, and Mark S. Drew. *Fundamentals of Multimedia*. Upper Saddle River, NJ: Pearson Prentice Hall, 2004. Print.

**Appendix**:

**A1**: Color Model Transform Equations

RGB\_to\_XYZ(r, g, b):

x = 0.607 \* r + 0.174 \* g + 0.200 \* b

y = 0.299 \* r + 0.587 \* g + 0.114 \* b

z = 0.066 \* g + 1.116 \* b

return (x, y, z)

RGB\_to\_CMY(r, g, b):

return (1 - r, 1 - g, 1 - b)

RGB\_to\_YUV(r, g, b):

y = 0.299 \* r + 0.587 \* g + 0.144 \* b

u = -0.299 \* r + -0.587 \* g + 0.886 \* b

v = 0.701 \* r + -0.587 \* g + -0.114 \* b

return (y, u, v)

RGB\_to\_YIQ(r, g, b):

y = 0.299 \* r + 0.587 \* g + 0.144 \* b

i = 0.595879 \* r + -0.274133 \* g + -0.321746 \* b

q = 0.211205 \* r + -0.523083 \* g + 0.311878 \* b

return (y, i, q)

RGB\_to\_YCbCr(r, g, b):

y = 0.299 \* r + 0.587 \* g + 0.144 \* b

cb = -0.168736 \* r + -0.331264 \* g + 0.5 \* b + 0.5

cr = 0.5 \* r + -0.418688 \* g + -0.081312 \* b + 0.5

return (y, cb, cr)

DEFAULT\_XYZ = 50

RGB\_to\_LAB(r, g, b, xn=DEFAULT\_XYZ, yn=DEFAULT\_XYZ, zn=DEFAULT\_XYZ):

x = 0.3935 \* r + 0.3653 \* g + 0.1916 \* b

y = 0.2124 \* r + 0.7011 \* g + 0.0866 \* b

z = 0.0187 \* r + 0.1119 \* g + 0.9582 \* b

rel\_x = (x / xn) \*\* (1 / 3)

rel\_y = (y / yn) \*\* (1 / 3)

rel\_z = (z / zn) \*\* (1 / 3)

L = 116 \* (rel\_y) - 16

a = 500 \* (rel\_x - rel\_y)

b = 500 \* (rel\_y - rel\_z)

return (L, a, b)

RGB\_to\_HSL(r, g, b):

max\_ = max(r, g, b)

min\_ = min(r, g, b)

chroma = max\_ - min\_

if chroma == 0:

H = 0 # Hue is undefined

else:

if max\_ == r:

H = ((g - b) / chroma) % 6

elif max\_ == g:

H = ((b - r) / chroma) + 2

else:

H = ((r - g) / chroma) + 4

H \*= 60

L = (.5) \* (max\_ + min\_)

if chroma == 0:

S = 0

else:

S = chroma / (1 - abs(2 \* L - 1))

return (H, S, L)

HSL\_to\_RGB(hue, sat, lum):

chroma = (1 - abs(2 \* lum - 1) ) \* sat

hue\_prime = hue / 60

x = chroma \* (1 - abs( (hue\_prime %2) - 1) )

if hue\_prime >= 0 and hue\_prime < 1:

r1,g1,b1 = (chroma, x, 0)

elif hue\_prime >= 1 and hue\_prime < 2:

r1,g1,b1 = (x, chroma, 0)

elif hue\_prime >= 2 and hue\_prime < 3:

r1,g1,b1 = (0, chroma, x)

elif hue\_prime >= 3 and hue\_prime < 4:

r1,g1,b1 = (0, x, chroma)

elif hue\_prime >= 4 and hue\_prime < 5:

r1,g1,b1 = (x, 0, chroma)

elif hue\_prime >= 5 and hue\_prime < 6:

r1,g1,b1 = (chroma, 0, x)

else:

r1,g1,b1 = (0,0,0) #H was undefined.

li = lum - (.5\*chroma)

return (r1 + li, g1 + li, b1 + li)

XYZ\_to\_RGB (X,Y,Z):

var\_X = X/100

var\_Y = Y/100

var\_Z = Z/100

var\_R = var\_X\*3.2406 + var\_Y \*-1.5372 + var\_Z\*-.4986

var\_G = var\_X \* -0.9689 + var\_Y \* 1.8758 + var\_Z \* 0.0415

var\_B = var\_X \* 0.0557 + var\_Y \* -0.2040 + var\_Z \* 1.0570

if var\_R > 0.0031308:

var\_R = 1.055 \* ( var\_R \*\* ( 1 / 2.4 ) ) - 0.055

else:

var\_R = 12.92 \* var\_R

if var\_G > 0.0031308:

var\_G = 1.055 \* ( var\_G \*\* ( 1 / 2.4 ) ) - 0.055

else:

var\_G = 12.92 \* var\_G

if var\_B > 0.0031308:

var\_B = 1.055 \* ( var\_B \*\* ( 1 / 2.4 ) ) - 0.055

else:

var\_B = 12.92\*var\_B

R = var\_R \*255

G = var\_G \*255

B = var\_B \*255

return (R,G,B)

LAB\_to\_XYZ(L,A,B):

var\_Y = ( L + 16 )/116

var\_X = A/500 + var\_Y

var\_Z = var\_Y - B/200

if var\_Y\*\*3 > 0.008856:

var\_Y = var\_Y\*\*3

else:

var\_Y = ( var\_Y - 16 / 116 ) / 7.787

if var\_X\*\*3 > 0.008856:

var\_X = var\_X\*\*3

else:

var\_X = ( var\_X - 16 / 116 ) / 7.787

if var\_Z\*\*3 > 0.008856:

var\_Z = var\_Z\*\*3

else:

var\_Z = ( var\_Z - 16 / 116 ) / 7.787

ref\_X = 95.047

ref\_Y = 100.000

ref\_Z = 108.883

X = ref\_X \* var\_X

Y = ref\_Y \* var\_Y

Z = ref\_Z \* var\_Z

return (X,Y,Z)

LAB\_to\_RGB(L,A,B):

X,Y,Z = LAB\_to\_XYZ(L,A,B)

R,G,B = XYZ\_to\_RGB(X,Y,Z)

return (R,G,B)

YIQ\_to\_RGB(Y,I,Q):

R = 1\*Y + .9563\*I + .6210\*Q

G = 1\*Y + -.2721\*I + -.6474\*Q

B = 1\*Y + -1.1070\*I + 1.7064\*Q

return (R,G,B)

CMY\_to\_RGB(C,M,Y):

R = 1-C

G = 1-M

B = 1-Y

return (R,G,B)

YUV\_to\_RGB(Y,U,V):

R = 1\*Y + 0\*U + 1.14\*V

G = 1\*Y + -.394\*U + -.581\*U

B = 1\*Y + 2.028\*U + 0\*V

return (R,G,B)

YCbCr\_to\_RGB(Y,Cb,Cr):

R = Y + 1.402\*(Cr-128)

G = Y - .034414\*(Cb-128) - .71414\*(Cr-128)

B = Y + 1.772\*(Cb-128)

return (R,G,B)

**A2**: Median-Cut Algorithm

median\_cut(boxes,n):

while len(boxes)<n:

big\_box = largest\_box(boxes)

range\_list = [range\_r(big\_box),range\_g(big\_box),range\_b(big\_box)]

max\_range = max(range\_list)

if max\_range == range\_list[0]: # red has biggest range

box1,box2 = split\_r(big\_box)

elif max\_range == range\_list[1]:

box1,box2 = split\_g(big\_box)

elif max\_range == range\_list[2]:

box1,box2 = split\_b(big\_box)

i = boxes.index(big\_box)

del boxes[i]

boxes.insert(i,box2)

boxes.insert(i,box1)

return boxes

**A3:** RGB averages



Contributions

*David Rux* - Documentation

*Steven Carr* - Documentation, code review and testing.

Casey Kuball – README/Insallation instructions, documentation, Main interface/command loop, color conversion functions, 6x6 image splicing function, color average printing function, highlighting function.

*Glenn Craver* - compare energy function, convert hsl function, saturate and desaturate functions

*Wesley Rose* - find average color function, compare average color function, display nearest image function, reduce color instances function, median cut function