An Application for Color Blind People
-
Animal Vision

Chiara Orvati
Ihsan Aquil
Saba Imran
Yannick Grimault

Course "Computational Photography"

Two-third report

1 Color Blind

1.1 Description of the algorithm (Yannick Grimault & Saba Imran)

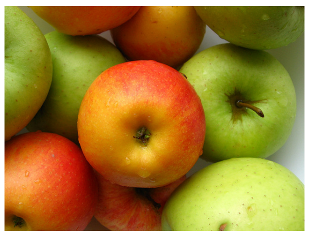
In order to simulate the vision of Color Blind people, we simply used the algorithm described by Brettel et al.[1].

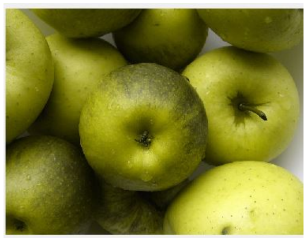
The key point of this algorithm is that we need to work in the LMS color space, which corresponds to the cones of the human eye (that way, working on diseases that render one type of cone inefficient is easier). Thus, the first step is to go from the original color space (RGB space in Matlab), and the last step will be to go back to this color space. This is easily done by multiplying each 3D-dimensional pixel by the corresponding transformation matrix.

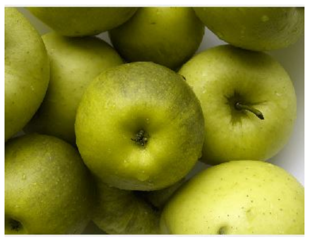
Once this is done, we simply need to apply the correct transformation in the LMS space. In order to find the correct transformation, one needs to understand the different kinds of color blindness: protanopes, deuteranopes and tritanopes. Basically, they just "do" a projection in the LMS space, with coefficients depending on the type of color blindness.

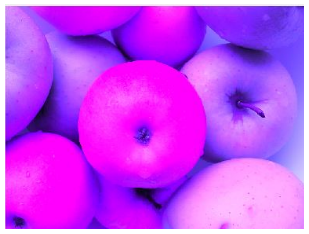
A particular obstacle with the Brettel algorithm is that it hinges on a projection onto one of the normal planes using the OE axis, which is the line between the Origin and the point E, the equal energy stimulus metamer for White. This is supposedly the 'normal axis' which can be seen normally by all three deficiencies.

Because we don't have a direct translation for this concept in the context of our application, I used William Woods's general transformations for each dichromacy.[2]. This paper finds normalized projections into LMS space for the different vision deficiencies. Our results are presented here. It uses the general mechanism for image modification, but finds relevant constant values for it rather than defining a new projection with each pixel.


Original


Protonopia


Deuteranopia


Tritanopia

Now that the basic algorithm is written, we need to create an algorithm for modifying the projection so as to manipulate the severity of the colorblindness. Future plans include writing this script in Java and optimizing it for continuous display in video playback.

2 Animal Vision (Chiara Orvati)

2.1 Description of the algorithm

In order to simulate animal vision, we use the algorithm described hereafter, which makes use of the spectrum representation of an RGB triplet.

[1]H. Brettel, F. Viénot and J. D. Mollon, "Computerized simulation of color appearance for dichromats", JOSA A 14?10 (1997): 2647-2655

[2]Woods, William, "Modifying Images for Color Blind Viewers", Stanford University Electrical Engineering (2012)

Let us first specify the transformation applied to a single pixel of the input image, which is described by an RGB triplet: The first step is to convert the RGB triplet to a spectral representation ranging from 380 - 720 nm using 10 equally sized bins. To obtain this representation we use the RGB values to linearly combine the known spectra of white, yellow, magenta, cyan, red, green and blue, making use of the method and spectral data described in [3].

In order to simulate the vision of a specific animal, we need to specify its cone sensitivities: this is done by defining a certain number of wavelengths which correspond to the peak cone sensitivities of the respective animal. As an example, the white-tailed deer has peak cone sensitivities at approximately 455 nm and 537 nm [4]. Having the spectral representation of the RGB triplet and the wavelengths to which the animal is most sensitive, we can now look up the values of the spectrum at these wavelengths. For the case of the deer this gives us two values, let's say 0.4 and 0.6. Using the inner product of the wavelengths with the spectrum values, we obtain the wavelength of the colour perceived by the animal, i.e., 0.4*455 + 0.6*537 = 504 nm for the example of the deer. However, before applying the inner product we need to normalize the spectral values, so as to make sure to stay in the range 380 - 720 nm. The final step is to map the obtained wavelength back to an RGB triplet, which is done using the values provided by [5] and using a discretization of 10 nm per RGB value.

In order to see the input image through the eyes of an animal we apply the above transformation to every pixel. To make the image appear more natural, we would like to preserve the luminance of the input image. Hence, we transform both input and simulated image to CIE 1976 L*a*b space using the Matlab built-in function `rgb2lab`. To assemble the final image, we use the luminance values of the original image and the a*b* values of the simulated image and transform it back to RGB.

2.2 Matlab Code

The following scripts/functions are provided in the Matlab Code:

- `main.m`: reads an image and displays it like an animal would see it (using the luminance of the original image). In the variable sensitivities one can specify the cone sensitivities of the animal we wish to simulate (ex: [420, 530, 560]. These wavelengths must be in ascending order and in the range 380-720 nm.)
- `img2Animal(in, sensitivities)`: takes an image and transforms every pixel to the corresponding animal vision RGB triplet using the sensitivities specified above.
- `rgb2Animal(R, G, B, sensitivities)`: transforms a single RGB triplet to the corresponding animal vision RGB triplet.
- `rgb2spectrum(red, green, blue)`: converts an RGB triplet to a spectrum with 10 bins, ranging from 380nm to 720 nm.
- `wavelength2rgb(lambda)`: takes a wavelength in the range 380-720 nm and outputs a corresponding RGB triplet using a discretization of 10 nm per RGB value.

[3]Smits, Brian. "An RGB to Spectrum Conversion for Reflectances." (2000).

[4]VerCauteren, Kurt C. and Pipas, Michael J., "A review of color vision in white-tailed deer" (2003). USDA National Wildlife Research Center - Sta Publications. Paper 284.

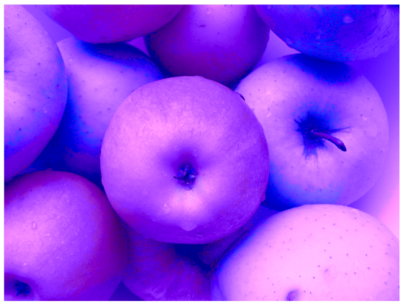
[5]`https://academo.org/demos/wavelength-to-colour-relationship/` . Accessed April 2017.

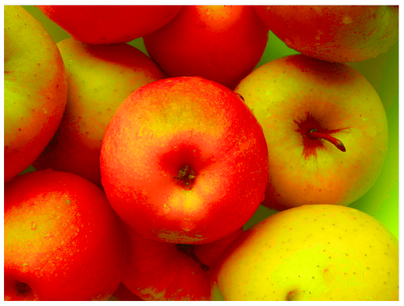
- `getInterpolated(samplePoints, spectrum, x)`: returns the value of the spectrum — which is only known at positions samplePoints — at wavelength x using interpolation.

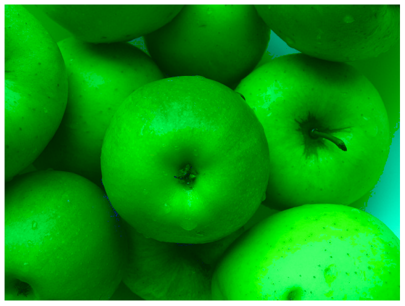
2.3 Results

Below we show example outputs for:

1. the made up cone sensitivities [390, 450],
2. the sensitivities [435, 546, 700] corresponding to the wavelength of blue, green and red
3. the sensitivities [455, 537] corresponding to the cone peak sensitivities of the white-tailed deer


1


2


3

For comparison, the input image is the following[6]:

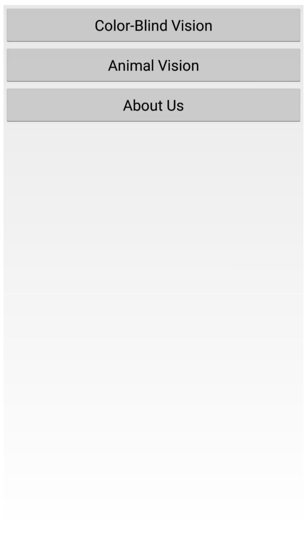
Apples.jpg

3 Our Application

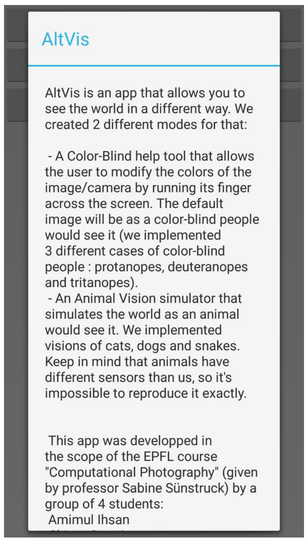
3.1 Basic Application (Yannick Grimault)

For now, our application doesn't do much. We simply have a menu that can redirect the user towards one of the features which are not yet implemented. We also put an "About us" section that gives information to the user about this project. Because it is just a text, we can add a part explaining how the application works too, but this is not our immediate goal.

[6]Taken from Flickr user zaveqna, "Apples," (2008). Accessed March 2017, `https://www.flickr.com/photos/zaveqna/2872120203/`



Main Menu



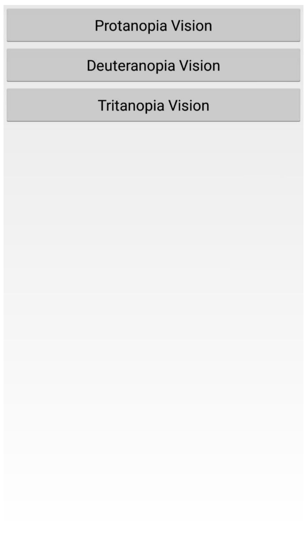
AltVis

AltVis is an app that allows you to see the world in a different way. We created 2 different modes for that:

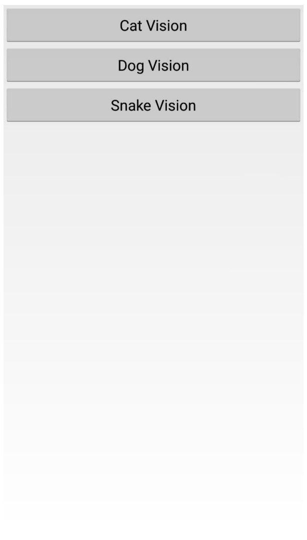
- A Color-Blind help tool that allows the user to modify the colors of the image/camera by running its finger across the screen. The default image will be as a color-blind people would see it (we implemented 3 different cases of color-blind people : protanopes, deuteranopes and tritanopes).
- An Animal Vision simulator that simulates the world as an animal would see it. We implemented visions of cats, dogs and snakes. Keep in mind that animals have different sensors than us, so it's impossible to reproduce it exactly.

This app was developped in the scope of the EPFL course "Computational Photography" (given by professor Sabine Sünstruck) by a group of 4 students:
Amimul Ihsan

About Us



Color Blind Menu



Animal Vision Menu

3.2 Using the camera in Android, with OpenCV 3.2 (Ihsan Aquil)

We tried to implement camera operations in parallel to the main app, so we could work faster. We'll need to merge everything later on though. For now, this is what we've implemented so far that will be used for the camera:

- `MainActivity`: Sets camera functions and controls the camera based on user choice
- `onCreate(Bundle savedInstanceState)`: Makes the camera visible on android phone
- `onPause()`: Pauses the camera based on user's choice
- `onDestroy()`: Stops the camera feed
- `onResume()`: Resumes the camera
- `onCameraViewStarted()`: Gets the RGB value and convert it to Grey level