Abstract

Individuals suffering from a colour vision deficiency (or CVD) have trouble distinguishing between certain colours. For a person suffering from a form of CVD, everyday tasks such as watching the weather forecast or even viewing a webpage become difficult. With the advance of the Internet and e-commerce which rely heavily on visual stimuli, it is becoming increasingly important that designers ensure their work is as accessible as possible to those suffering from a CVD. Research in the field of CVD and computer science has led to the development of CVD simulation, allowing an individual to view the world from the perspective of an individual suffering from a CVD. The overall objective of this project is to produce an educational tool that can be used to promote the awareness of CVD through visual representation by applying various filters to images.

1. Introduction

Computer imagery has developed significantly over the past twenty years; users are no longer limited to black and white command prompts. However as advances have been made in computing and computer graphics, the problem of colour vision deficiency has been largely ignored.

According to Hoffman designerm user interfaces will have a predisposition to “color-code”:

Designers will frequently color-code individual words, buttons, or areas of the screen to differentiate functions or to group similar items.

In order to promote awareness of colour vision deficiency, we proposed to develop software that will allow the user to view the world from the perspective of those with this condition. Thus we have developed Vis-It; an educational tool which filters images to represent the three kinds of dichromatic colour vision deficiency.

2. Background

Colour vision deficiency (henceforth referred to as CVD) was first presented as a formal scientific observation by the chemist John Dalton around two hundred years ago. Dalton theorised that his own condition of CVD was caused by the presence of tinted humour within his eyes. Though his reasoning for the cause of colour vision deficiency was discredited by many such as Hunt et al who noted a post-mortem autopsy presented there were no additional liquids within his eyes, his initial observation was a catalyst for anecdotal observations of the condition which would later cause many papers to be published; papers that would not only recognise CVD’s existence but also present the cause of its occurrence.

One of those papers was written by Deeb et al, the authors state within their text that CVD can be either be genetic or an acquired condition. In the case that the condition is genetic; the condition is sex-linked and thus it is more common among males, since CVD is a recessive quality that is carried within the X chromosome. Thus for a female to be colour vision deficient both her X chromosomes must carry the recessive CVD quality. On the other hand CVD can be acquired after birth from damage to the brain and/or eyes.

Deeb et al continue to elaborate about CVD by explaining that the cause of the condition is the absence or malfunction of photoreceptors in the eye. The cause of the condition is concurred by a text by Bowmaker who explains that photoreceptors are divided up into two classes; rods and cones. While cones detect colour, rods detect light (rods will not be investigated since they are not within the scope of this paper).

Cones are categorised according to their wavelength detection function; each base colour (red, green and blue) gives off a wavelength
which is received by a respective cone within the human eye. In summary there are three types of cone photoreceptors; red cones for receiving long wavelengths, green cones for receiving medium wavelengths and blue cones for receiving short wavelengths. The severity of CVD is dependent upon the amount and type of photoreceptors which are absent or dysfunctional, which according to Judd\(^6\) can be divided up in the following broad categories:

\[\text{§} \]

**Monochromacy:** Also known as total colour deficiency; all types of photoreceptors are absent or dysfunctional, thus the individual has no ability to distinguish long, medium or short wavelengths. According to Cassin \textit{et al}\(^7\) this form of CVD is considered very rare occurring in 0.00001% of all men and women within the United States.

\[\text{§} \]

**Dichromacy:** One of the types of photoreceptors is absent or dysfunctional, thus the individual has the ability to perfectly distinguish one colour but the individual tends to confuse two base colours. According to Cassin \textit{et al}\(^7\) there is 2.4% occurrence within men and 0.03% within women within the United States.

\[\text{§} \]

**Anomalous Trichromacy:** According to Judd\(^6\) trichromacy is a case when an individual can perfectly recognise three different colours, therefore normal vision is a form of trichromacy. However there is a form of CVD referred to as anomalous trichromacy, where the anomaly exists because according to Byrne \textit{et al}\(^8\) an individual has three fully functioning types of photoreceptors cones but still has slight trouble when it comes to distinguishing certain “hues”\(^2\) of colours. According to Cassin \textit{et al}\(^7\) this is the most common form of CVD affecting around 6.3% of all males and 0.37% of all females within the United States.

The notation of the three generalised segments of CVD has proven that CVD is a very large area of study; approaching it will require some scope, thus a specific form of CVD must be chosen before progressing any further. We decided to investigate dichromacy since anomalous trichromacy is not a condition which is extreme enough to impair an individual, and conversely monochromacy is extremely rare.

According to Kaiser \textit{et al}(1996)\(^9\) there are three different types of dichromatic colour vision deficiencies:

\[\text{§} \]

**Protanopia:** A condition where cone photoreceptors which usually detect long wavelengths (the perception of the colour red) are malfunctioning or absent. The effect on an individual’s perception is the compensation of long wavelengths with medium (or green) wavelengths resulting in the confusion of the colour green and red (figure 1). Cassin \textit{et al}\(^7\) state that this form of dichromacy affects around 1% of all males and 0.02% of all females within the United States.

\[\text{§} \]

**Deutanopia:** A condition where cone photoreceptors which usually detect medium wavelengths (the perception of the colour green) are malfunctioning or absent. The effect on an individual’s perception is the compensation of medium wavelengths with long wavelengths resulting in the confusion of the colour red and green (as with the case of protanopia). Cassin \textit{et al}\(^7\) have presented statistics about the occurrence of this condition within the United State; deuteranopia affects around 1% of all males and 0.01% of all females.

Figure 1.
• Tritanopia: A condition where cone photoreceptors which usually detect short wavelengths (the perception of the colour blue and yellow) are malfunction or absent. Unlike deuteranopia and protanopia, the short wavelength is not compensated by other wavelengths, instead the effect on an individual’s perception is the inability to perceive blue or yellow resulting in the individual’s inability to distinguish the unperceivable colours. According to Cassin et al. this is the least common form of dichromacy affecting around 0.001% of all males and 0.03% of all females within the United States.

The observations by John Dalton and the subsequent extensions made by other scientists after him removed the shroud of ignorance about the condition. As a result of scientific experiment subtle mitigations to CVD have been applied in situations where colour has a critical role. Key examples are road signs; stop signs are in red but are always octagons, the positioning of bulbs within a traffic light (the top bulb which is red is always stop, the middle bulb which is yellow is always yield, and the bottom bulb which is green is always go). The technique implemented within road signs to override the importance of colour is called “visual cue”.

However visual cues are only effective when colours have associative meanings, but one could easily ask: how can CVD be mitigated in situations where colours do not have associative meanings? Such a question easily applies to images when colours are utilised to give a direct representation of an object. A text by Hoffman suggests the avoidance of use of colours which can be confused by those who are colour vision deficient. On the other hand not everyone is an expert in CVD and thus a technique or tool must be developed to visually present the perspective of those who suffer from CVD. Thus we have an objective but no means by which to reach that objective.

3. Related Work

Before we were able to start work on the design of our application and further our ideas it was essential that we conducted some research into related work in the field of colour vision deficiencies. This consisted of two parts, researching solutions that were already available and reading academic papers related to colour vision deficiencies.

3.1. Current Solutions

Vischeck is a web based application which provides an interactive online demo to demonstrate how images look from the perspective of someone suffering from a colour vision deficiency. The website allows users to upload images from their own computer and select a filter to apply to the image in order to view the image from the perspective of someone suffering from deuteranopia, protanopia or tritanopia. Vischeck also provides a facility which allows users to enter a URL and apply one of the given filters to an HTML document. The services that Vischeck provide are of a similar nature to the application that we planned to implement.

Colour vision deficiency is not a new problem in the world, and hence people have already calculated ways in which to simulate it digitally. We discovered a paper by Hans Brettel which describes an algorithm he had developed to correct images and display them as a colour vision deficient person would see them.

The algorithm he uses alters RGB pixel values according to the type of colour vision deficiency present. The paper deals with 3 types of colour vision deficiency, protanopia, deuteranopia and tritanopia, by removing the wavelengths of the damaged cone types in the eye from the RGB values. How this is carried out will be explained later in this report.

4. Aims

Our primary aim was to produce an educational tool that could be used in order to
view images from the perspective of an individual suffering from a colour vision deficiency. We aimed to provide a tool that could be used by designers as a means of assessing how accessible their work is to people suffering from a colour vision deficiency. As well as providing a means of assessment, we aimed to produce a tool which educated and promoted the general awareness of colour vision deficiencies to its users. In order to achieve such goals, we proposed to provide an intuitive GUI that made the viewing of such images both an enjoyable and easy process for the user.

5. Design specification

Based on the research we conducted into already available current solutions and our original aims, the following requirement specification was devised:

5.1. Essential Features

The essential features were regarded as the fundamental minimal requirements to be implemented. It was decided that these features must be fully functioning before any extended features were considered for implementation.

- Uploading of images from the computer.
- Library of sample images.
- Application of deuteranopia, protanopia and tritanopia filters.
- Scaling of images.
- Saving of filtered images.
- Printing of filtered images.
- Displaying descriptions of the filters

5.2. Extended Features

The following features were considered as additional features and were to be implemented after the successful implementation of the essential features in the following order of priority:

- HTML Filter System
- ClearView
- Help Feature

6. System Design

6.1. Graphical User Interface

Building a well-designed intuitive GUI was considered a high priority as it provided the user with a means of uploading and viewing images. The GUI was designed with usability in mind and we aimed to provide an intuitive interface with a logical layout. The initial GUI design is displayed in figure one. The GUI was designed as a JFrame containing a JPanel which in turn contained three other components. The largest component at the centre of the GUI was designed to display the images. The component at (e) was designed with the intention of holding a JProgressBar to display filtering information to the user and the component (d) as a main menu from which various tasks could be invoked. When designing the GUI we had both first time and experienced users in mind. It was decided that the components should be laid out in the logical order that they would be used during the filtering process, beginning with opening an image and ending with saving and printing functions. In order to guide the first time user through the filtering process, components are disabled and enabled at the relevant points. For the more experienced user the GUI provides a standard file menu from which tasks can be invoked. These tasks may also be called using shortcuts consequently aiding the experienced user and increasing usability.

Figure 1.
Simulating Images for Dichromatic Colour Vision Deficiency with Vis-IT

...aiding accessibility. In order for users to truly appreciate the differences between a filtered image and an unfiltered image it was important that users were able to view the original unfiltered images alongside the new filtered images. We decided to place a JSplitPane in (c) to contain the unfiltered image and filtered image as this allowed the two images to be displayed along side each other at the same time. The JSplitpane’s divider also provided a means for the user to increase or decrease the size of the filtered or unfiltered image by dragging the divider to the desired position. We felt this would benefit those who may suffer other visual impairments and require larger images to view them properly.

Vis-IT was created with the intention of being able to run on several different operating systems. We therefore thought it appropriate that the GUI would adjust its look and feel depending on the operating system the program was running on, thus providing a familiar setting for the user. As Vis-IT has been designed to teach individuals about colour vision deficiencies and help designers provide accessible interfaces it was important that the colours used in the GUI were accessible to those suffering from a form of colour vision deficiency. The colour scheme for the Vis-IT GUI was therefore kept very simple, using a combination of black, white and greys. This did not only increase the accessibility of the program to a variety of users, but by keeping the colour scheme as simple as possible it also ensured that attention was not drawn away from the images being filtered.

7. Vis-IT Technical Details

Implementation of Vis-IT went according to plan although we did encounter a few challenges along the way, as expected in any software development process.

7.1 GUI Development

The implementation of the GUI turned out to be quite a complex process. In order to achieve the desired GUI design several different panels needed to be placed inside each other, each with their own layout manager. During the implementation of the GUI several sketches were drawn in order to keep track of which panels contained each other and the layout manager associated with each panel (figure 2). These sketches were vital to the implementation process as they kept track of the GUI progress.

![Figure 2.](image)

The main window (see Appendix 1) of the GUI Figure 1 (a) was achieved by creating a JPanel that used a BorderLayout manager. A second JPanel Figure 2 (b) was placed in the centre to contain a JSplitPane (d). The JSplitPane was implemented to hold the unfiltered image (c) and the filtered image (e) alongside each other. As we had no experience of using JSplitPane’s the implementation of the JSplitPane was conducted with reference to the Java Swing Tutorial12. A property change listener was added to the JSplitPane, which made a call to a scale function whenever the component was changed. The scale function calculated the width and height of the components in the JSplitPane and scaled the images accordingly to fill the component.

During implementation it was decided that it would be useful if the program could handle multiple images. This would enable a user to have several images open at the same time in order to examine the differences between the different filters. It was therefore decided that a JTabbedPane should be implemented in the centre of the GUI (c) to hold the JSplitPane. The GUI was implemented in such a way that each time a new image was opened a new tab was created and a new JSplitPane added to the tab.

7.2. Developing the filter

The algorithm developed by Brettel et al11 is fairly straight forward, and relatively easy to encode. The only problem was obtaining the correct values. The algorithm consists of 3 main
Simulating Images for Dichromatic Colour Vision Deficiency with Vis-IT

steps and would resemble the following pseudo code

\[
\text{While pixels remaining} \{
\quad \text{Get next pixel}
\quad \text{Extract RGB values}
\quad \text{Step 1: Transform RGB values into LMS values}
\quad \text{Step 2: Correct the appropriate value}
\quad \text{Step 3: Transform LMS back into RGB}
\quad \text{Generate new colour from RGB values}
\}
\text{Display resulting picture}
\]

Extracting pixels and getting their RGB values was a simple case of using a PixelGrabber object supplied in the Java.image libraries to grab pixels in an image and then applying some bit shifting and masking to get the individual red, green and blue components of each pixel.

In step 1 there is some matrix multiplication, and as Java has no built in matrix functions, an external library was used. Transforming the RGB values to LMS values (LMS values being the responses to light of the long, medium and short wavelength cone types described earlier in this report) involved multiplication by a transformation matrix. The values for this matrix were not taken from the Brettel paper, however, as the values he used were based on readings from a CRT monitor using specialised equipment. We discovered another paper by Reinhard et al that contained a method for generating a device-independent RGB to LMS transformation matrix by first transforming RGB values into XYZ values in CIE space and then transforming the XYZ values to LMS co-ordinates. This double transformation could be rewritten as a single transformation from RGB to LMS values as matrices are associative.

With these LMS values we could then correct them according to the type of colour vision deficiency chosen by the user in the GUI. Following Brettel's calculations, for protanopia, the long wave-length component is corrected, for deuteranopia, the medium wavelength component is corrected and for tritanopia the short wavelength component is corrected.

This correction process relies on two other attributes; a neutral colour line from the origin of the LMS space graph to a point called the Equal Energy Stimulus, which according to Brettel is “the brightest possible metamer of an equal-energy stimulus”. The points that lie between the origin and this point should remain unchanged for all images after filtering as they are perceived the same by everyone. The second attribute is referred to as an Anchor Stimulus. This can be one of a few values, and depends on the position of the initial LMS co-ordinate and the type of colour vision deficiency being dealt with. It decides how a value is corrected by choosing which side of the origin to equal energy stimulus line to project the missing component. The other components would remain unchanged.

With the corrected LMS values, we then had to return to RGB space so the image could be displayed, hopefully filtered correctly for the user. This was done by multiplying the new LMS co-ordinates by the inverse of the initial transformation matrix. We then wrote the new RGB pixel into a new image at the same position as the original pixel (see Appendix 2).

8. Challenges

Implementation of the project ran fairly smoothly. We did, however face the following challenges during implementation.

8.1. Using a JTabbedPane

The JTabbedPane was not part of the original GUI design and its implementation had an unforeseen impact on the rest of the system architecture. In order to handle multiple images the images being loaded into the program needed to be stored somewhere so that filters could be reapplied to images previously loaded. In order to overcome this problem two arrays were implemented, one to hold the original images and one to hold the filtered images. This meant that each time an image needed to be accessed the currently selected tab needed to be determined and the corresponding panel retrieved from it. The JSplitPane would then be extracted from the panel and the desired image could be extracted from the JSplitPane.

8.2. Memory Management

Images files can take up a lot of memory, but thankfully with image compression techniques the file sizes can be reduced quite dramatically. Being an image processing project, we had to be able to deal with the amount of data a larger image may contain. With the miniaturisation of flash memory and improvements in digital photography, very high resolution images are now easily available. With high resolution comes high pixel count, with a typical digital
camera having upwards of 4 million pixels per image. As the filtering algorithm we are using runs over each pixel, the larger the image, the longer the filtering will take. Although we could have divided all images up into small chunks to filter separately, this caused the performance of the application to suffer.

The PixelGrabber class we used initially grabbed the whole picture and split all the pixels into an array with a length of the width of the image multiplied by the height of the image. For a large image, say 3000 pixels in width by 2000 pixels in height, an array of 6 million integers would be generated. The default Java heap space could not cope with this and so we had to attempt to work around it.

When the array of pixels for an image was created by the PixelGrabber, an OutOfMemoryError would be thrown if the array was too large. For our application to continue running this error would need to be caught and the pixel array nullified to clear out the memory it was attempting to use. The pixels in the large image would then be grabbed and filtered in two halves, assuming that no further OutOfMemoryErrors were thrown. If further OutOfMemoryErrors were thrown, the image would be filtered in smaller and smaller pieces until the whole image was filtered.

This method also helped when there were large multiple images open at the same time. Multiple images would take up memory and reduce what would be available to the filtering process, which if unchecked would cause further memory errors.

9. Testing

Testing was a continuous process that was carried out throughout the development of Vis-IT.

Each time a new feature was implemented the system was tested and any bugs found recorded. As a result of continuous testing, debugging and bug fixing took place on a day to day basis throughout the implementation process.

After the majority of implementation had taken place we embarked on further testing regimes. Since the majority of our code was heavily based on GUI interaction we decided to focus are testing strategies in this area. Testing of the GUI was not a straightforward procedure as it heavily relied on visual stimuli that could not be verified by merely running a program. A number of test cases were devised to test various routes that the user could take through the program. The expected outcome of a test case was then compared to the actual outcome that the program produced. If the expected outcome correctly corresponded with the actual outcome then the test case passed. If however, the excepted outcome and the actual outcome differed then the test case failed and the relevant bug was recorded. Any bugs found were then fixed and the relevant test cases were rerun.

In order to test how successful the implemented filters were we tested both the performance and the output of the program. The performance was measured by timing how long it took an image to be filtered. The test was then repeated with a number of different images of different sizes to ensure the filtering procedure was efficient. The outputs that the filter produced were then compared to example filtered images found on the Internet and a visual comparison was made. The results of these tests showed that our filtering process did not produce as accurate results as we had hoped.

The final stage of testing took the form of usability testing. We asked a number of individuals unfamiliar with the program to take part in a form of acceptance testing. The individuals were asked to follow a number of scenarios devised to test the functionality of the program and record any observations they made during their experience. This helped us evaluate how successful we were in achieving our aim to produce an intuitive and logical GUI.

10. Future Enhancements

Having completed the basic aims of the project, we looked at implementing some of the extended features that we discussed at the beginning of the project.

10.1. Dealing with HTML

Since an aim of our project was to promote designer awareness to CVD, we felt that an HTML Filter as a future extension would be a logical idea. The implementation of an HTML parser within Vis-IT is based on a two-step process, they are:

- Building a simple web browser
- Filtering the web pages.
The initial step is quite simple thanks to the class “JEditorPane” within the Java package. JEditorPane is a class which is utilized for the display and modification of text files, including HTML. The web browser would be based on a JEditorPane (Figure 3 (d)). For user interaction the browser would have to include a JPanel at the top of the window (c) which holds a JTextArea for URL input and a JButton to open the URL inputted in the JTextArea. A further JPanel (e) would be needed to hold the various components required for filtering, such as a JComboBox containing a list of the filters and a JButton to apply a filter.

We have designed a simple web browser with very limited functionality. While web pages can be loaded from the browser’s address bar, the web browser cannot handle redirection. The only way to load web pages is through usage of the address bar, users are unable to access web pages through links within a loaded page. We have discovered a way to solve this problem through the addition of a HyperlinkListener to the JEditorPane which displays the HTML file.

Even though we were unable to implement a filtering feature in the web browser we did have ideas about how the filtering process might work. The simplest solution that comes to mind is the conversion of HTML files into images and then applying the filtering code to the newly created image. Another possible solution would be the implementation of a HTML parser.

10.1.1 Parsing:

Parsing is a technique where a specific document’s tags for colour attributes. Implementation of a parser can take place in two ways, the usage of third party parsing packages like (HTMLParser) or Swing’s package for HTML editing, HTMLEditorKit. We will investigate using Swing’s package since it creates less dependency on third party packages. Creating a customized parser within Java is quite simple; all that is required is the creation of a subclass of ParserCallBack (within the HTMLEditorKit), since the ParserCallBack class is one which determines what will happen when a certain tag is found within an HTML document.

In order to parse a document the methods of the (superclass) ParserCallBack must be overridden by defining exactly which HTML tags to handle. Java gives us the ability to handle each kind of tag, however investigating each type of tag is only part of the process, we still need to extract colour attributes of the tags. Java has functionality for extracting attributes of a given tag by accessing HTML.Attribute.

In conclusion Java’s HTMLEditorKit gives us all the functionality we need for extracting relevant tags and querying each tag’s colour.

10.2. ClearView

Another possible way in which Vis-IT can be extended is through the implementation of “ClearView”. The idea behind ClearView is to provide the user with a transparent window which can be moved around the desktop filtering anything placed underneath it. This would allow users to view the operating system and other applications from the perspective of an individual suffering from a colour vision deficiency.

The simplest way in which ClearView could be implemented would be to create a transparent window with an ActionListener which is activated when the user is not moving the window. When this ActionListener is activated a screen shot of anything placed under the window is taken and saved as a JPEG. A filter can then be applied to the JPEG and the filtered image displayed to the user.

11. Conclusions

In conclusion, we achieved the main aims laid out at the start of the project. The GUI is simple yet efficient, there is code built in to stop the user doing anything untoward and the
filtering process is fast and gives good user feedback. The code has been written with extendibility in mind and given a few more weeks, multiple features could be implemented. Our application also gives educational information about the forms of colour vision deficiency without going into too much detail.

Acknowledgements

We would like to thank Richard Jones for his support and supervision, David Barnes for allowing us to use his OImage and ImageFileManager classes and Peter Kenny for assisting us with the filtering algorithm.

Bibliography
