Real-time Image Correction for Colorblind Users

Tyler D. Howard, *MIT Candidate*, and Jani Kajala, *Supervisor/Advisor*

[1]

***Abstract*— This paper aims to implement a real-time solution to correct images for colorblind users and determine if the absence of such a technique in video games is due to a prohibitive runtime cost or another factor, such as lack of knowledge or awareness. The algorithms used originate from colorblind correction filters for still images and photographs. We implement these filters in real-time as a post-process effect in a graphics engine and measure the framerate cost. The results show that the performance impact is not significant. This work provides an alternative solution to hard asset swaps when creating a game's colorblind mode.**

***Index Terms*— colorblindness, protanopia, deuteranopia, tritanopia, post-process, color, color space**

I. Introduction

PEOPLE with colorblindness suffer a disadvantage when playing games. Color usually serves as an important piece of information relevant to the game's rules, especially so in most puzzle games. While developers aware of the issue do take measures to minimize the negative influence of colorblindness on one's ability to play (such as the use of distinct shapes in addition to color), most others still fail to design with colorblind people in mind.

This study explores and implements methods to improve a colorblind player's experience. More specifically, it aims to complete a series of post-process effects: one adjusts the display to a spectrum of colors so that a non-colorblind player can see what a colorblind player normally does, and another that adjusts the display to a spectrum of colors that a colorblind player can see more clearly. The former assists developers and artists in double-checking that their game does not exclude colorblind players, and the latter corrects already-published games that fail to design around the problem. Offline solutions that correct color to a colorblind-visible spectrum exist, but real-time solutions are not prevalent. These tools have the additional caveat that they must not have a significant negative impact on the frame rate of a given application.

II. Research Review

Current solutions to colorblindness in published games consist of offline solutions that require extra production time. Artists either make a conscious effort to avoid creating colorblind-confusing assets or manually produce extra colorblind-friendly versions of existing colorblind-confusing assets. Examples include Ronimo Games' *Awesomenauts* [1], Invisible Handlebar's *Audiosurf* [2], and Riot Games' *League of Legends* [3]. More often than not, studios take neither approach, such as the case of Capcom's *Puzzle Fighter* [4].

One notable (and very recent, released less than a month before this paper's first draft) game utilizes a dynamic approach: Maxis' *SimCity (2013)* [5] contains a set of post-process effects similar to those described in the introduction.

Two main kinds of color theory exist: trichromatic theory and opponent theory [6]. Maxwell, Young, and Helmholtz, originally proposed trichromatic theory, which states that the human eye has three sets of receptors. These receptors each form monochrome red, green, and blue images. The brain receives input from these receptors and reassembles them to form a final composite image. In practice, the existence of three receptors that roughly correspond to red, green, and blue holds true, but the reassembly part of the theory fails to explain certain visual phenomena, such as the impossibility of perceiving hues described as "reddish-green" or "bluish-yellow." [6]

Opponent theory better describes the events that occur in the neural stage of perception. Hering proposed opponent theory, which also states the existence of three receptors, but terms them as pairs of stimuli: red-green, blue-yellow, and light-dark. Hering's theory explains the observable phenomena of not being able to perceive a hue as "reddish-green" or "bluish-yellow," as equal portions of paired stimuli oppose each other and cancel out. In addition, this explains why colorblindness manifests as defects in either red-green or blue-yellow perception. However, Hering's theory does not match up with physiological evidence and initially failed to gain acceptance. [6]

Initially thought to conflict, in truth, these two theories describe separate parts of a greater whole. Modern opponent-colors theory states that perception of color occurs in multiple stages. Three kinds of receptors do exist. They perceive different parts of the light wavelength spectrum. Long cones (known as L-cones) perceive wavelengths around 420nm (blue), Medium cones (known as M-cones) perceive wavelengths around 530 nm (green), and Short cones (known as S-cones) perceive wavelengths around 560 nm (red).

[1]T. Howard, *MIT Candidate,* is with The Guildhall at SMU, Plano, TX 75024 USA (214-385-3114; e-mail: tylerdavidhoward@gmail.com).

J. Kajala, *Supervisor/Advisor,* is a Lecturer in Software Development at The Guildhall at SMU, Plano, TX 75024 USA (972-473-3532; e-mail: jkajala@smu.edu).

B. Eiserloh, *Reader,* is a Lecturer in Software Development at The Guildhall at SMU, Plano, TX 75024 USA (972-473-3560; e-mail: beiserloh@smu.edu).

T. Cuevas, *Reader,* is the Deputy Director of Research and Curriculum at The Guildhall at SMU at SMU, Plano, TX 75024 USA (972-473-3528; e-mail: acuevas@smu.edu).

These receptors do not transmit information directly to the brain. Instead, neurons encode LMS signals to opponent signals along the pathway to the brain's final reassembly. The sum of all receptors results in the information for the light-dark stimulus pair A (L + M + S = A). The red-green pair R-G consists of the sum of the L and S receptors opposed to the M receptors (L + S - M = R-G). The yellow-blue pair Y-B consists of the sum of the L and M receptors opposed to the S receptors (L + M – S = Y-B). [6] The diagrams below illustrate the encoding from cone signals into opponent signals.

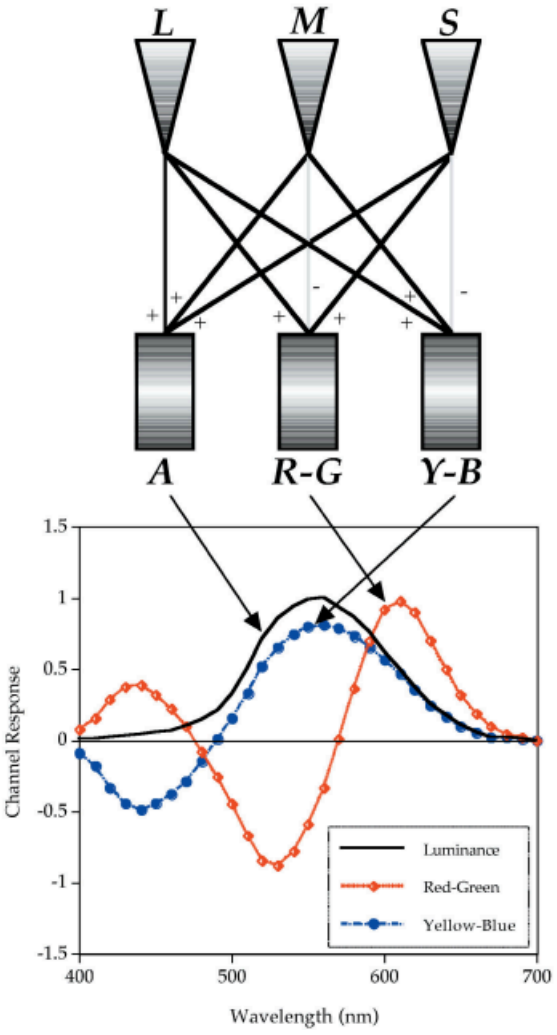


Figure 1.13 from Fairchild: Schematic illustration of the encoding of cone signals into opponent-color signals in the human visual system

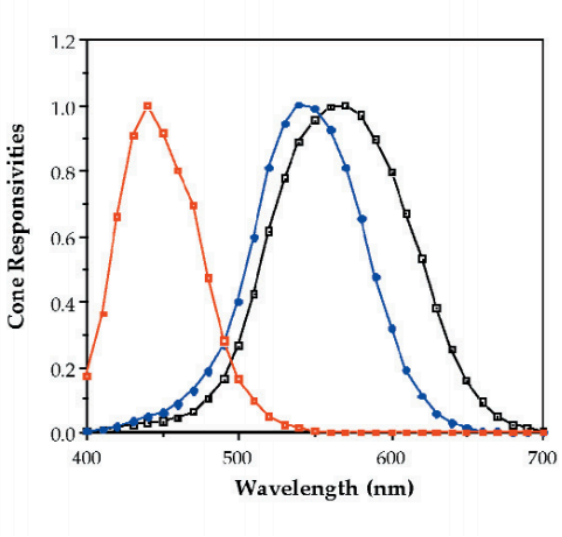
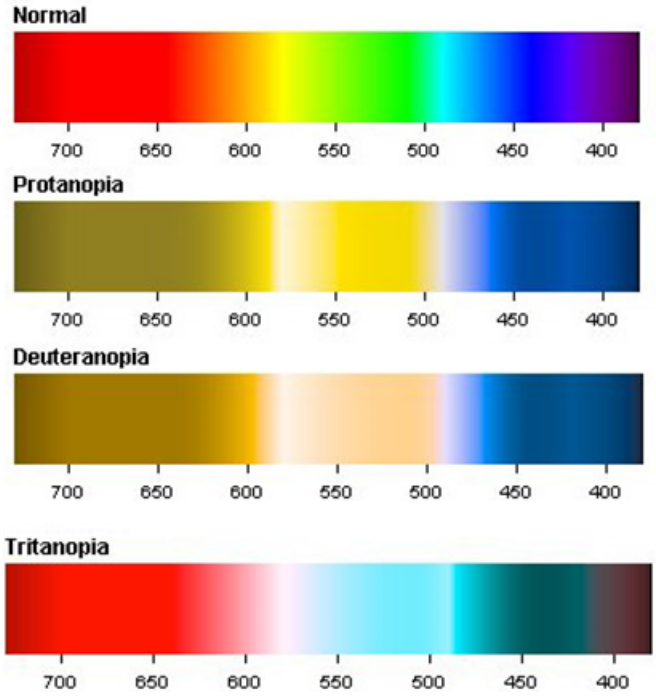


Figure 1.13 from Fairchild: Normalized responsivity spectra of human cone cells, L, M, and S types

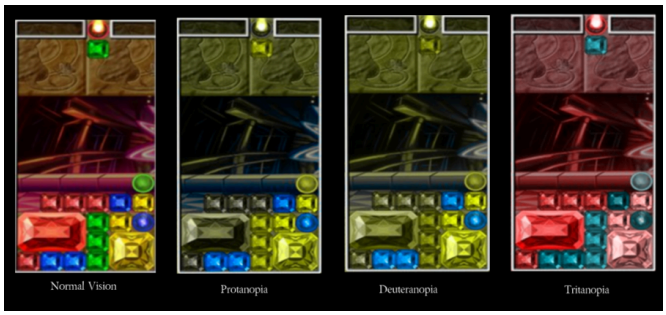
The word "colorblind" does not fully describe the phenomenon of a human eye's defective cones. Damage to an eye's cones rarely reduces their set of visible hues to monochrome – rather, it only impairs their set of visible hues to a smaller range varying from individual to individual. Colorblindness occurs strictly within the physiological side of the color perception process and only results from damage to the L-, M-, or S-cones. [6]

Varieties of colorblindness consist of protanopia, deuteranopia, tritanopia, and anomalous trichromacy. Protanopia occurs when damage to the S-cones is present. Deuteranopia occurs when damage to the M-cones is present. Tritanopia occurs when damage to the L-cones is present. Protanopia and deuteranopia are both forms of red-green colorblindness, and tritanopia is the form of yellow-blue colorblindness. [6]

Normal vision with no damage to cones is termed trichromacy, as all three cones have no defects or damage. Protanopia, deuteranopia, and tritanopia are forms of dichromacy, where two cones function normally, but one does not. Anomalous trichromacy occurs when partial damage to multiple cones exists, resulting in a combination of types of colorblindness. [6]



Diagrams from Colblindor [14]: Visible spectrums for normal and dichromatic varieties of vision



Screenshots from Puzzle Fighter [4], simulating normal vision and different varieties of colorblind vision

Brettel, Viénot, and Mollon [7] describe an algorithm that simulates dichromatic vision. The input consists of a pixel in RGB color space format and a type of dichromatic vision. They transform the pixel **V** in RGB color space to equivalent pixel **Q** in LMS color space with the following matrix **T**:

$$\begin{bmatrix} .1992 & 0.4112 & 0.0742 \\ .0353 & 0.2226 & 0.0574 \\ .0185 & 0.1231 & 1.3550 \end{bmatrix}$$

*Matrix **T** of Brettel et al. to convert from RGB space to LMS space*

$$\mathbf{Q = TV,\ V = T^{-1}\ Q}$$

*Eq. 4 of Brettel et al. to convert RGB pixel **Q** into LMS pixel **V** and vice-versa*

They then replace the LMS component marked as defective (e.g. the M component for an individual with deuteranopia) with a value derived from the other two components, resulting in a new pixel, **Q'**. The simulation algorithm (**Q→Q'**) uses a neutral stimulus **A** as an anchor point to determine the components **a, b, c** of a linear combination.

$$\mathbf{aL_{Q'} + bM_{Q'} + cS_{Q'} = 0}$$

Eq. 7 of Brettel et al.

After obtaining **a, b,** and **c**, (**Q→Q'**) replaces only the defective component.

$$\mathbf{L_{Q'} = -(bM_Q + cS_Q) / a}$$
$$\mathbf{M_{Q'} = -(aL_Q + cS_Q) / b}$$
$$\mathbf{S_{Q'} = -(aL_Q + bM_Q) / c}$$

*Eq. 9 of Brettel et al; protanopia, deuteranopia, and tritanopia **Q→Q'** transforms- only apply one, not all three*

$\mathbf{T^{-1}}$ likewise serves to transform a pixel from LMS color space back to RGB color space, which results in the final output: **V'**, the original pixel transformed as a dichromat perceives it.

$$\begin{bmatrix} 7.4645 & -13.8882 & 0.1796 \\ 1.1852 & 6.8053 & -0.2234 \\ 0.0058 & -0.4286 & 0.7558 \end{bmatrix}$$

*Matrix **T**$^{-1}$ of Brettel* et al.

$$\mathbf{Q = TV}$$
$$\mathbf{Q \rightarrow Q'}$$
$$\mathbf{V' = T^{-1}Q'}$$

Full summary of Brettel et al *simulation*

Dougherty and Wade, the owners of the website *Vischeck* [8], go as far to adjust the image to improve the contrast for people with protanopia or deuteranopia so they can perceive details they previously could not. Their implementation uses the CIELAB color space. The CIELAB color space represents the entire spectrum of human-visible light and consists of three components: **L** ranges between 0.0 and 1.0 and represents luminance, **A** ranges between -1.0 and 1.0 and

represents a spectrum between fully red and green, and **B** ranges between
-1.0 and 1.0 and represents a spectrum between fully blue and yellow. In order to compensate for users with red-green colorblindness (i.e. protanopia and deuteranopia), *Vischeck* alters the image's L and B stimulus channels. They do not have the exact details of their algorithm published. *Vischeck* also lacks a working implementation for users with tritanopia.

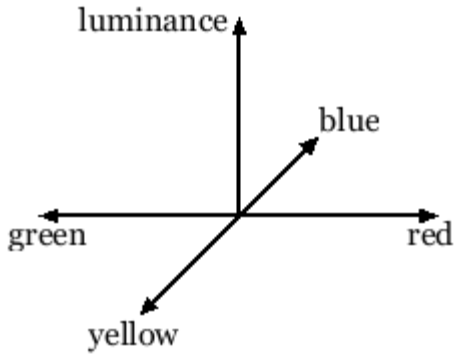


Fig. 3 of Visolve*: geometric representation of CIELAB*

Visolve [9] (similar to *Vischeck* in name, but distinct from it) serves a similar purpose to *Vischeck*: offline correction of photographs and still images. Unlike *Vischeck*, *Visolve* explains its algorithm in detail and has a working implementation for users with tritanopia. While *Vischeck* runs entirely online on a remote computer, *Visolve* has a tool available for Windows and Macintosh computers. The tool captures a screenshot and displays a corrected version – while *Visolve* performs this locally, it does not do so in real time.

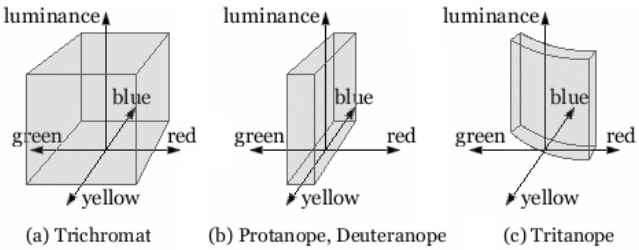


Fig. 8 of Visolve: *geometric representations of trichromatic and dichromatic vision*

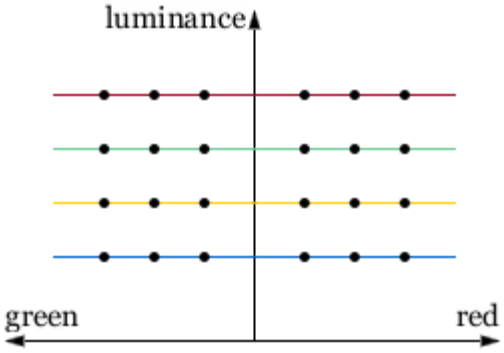


Fig. 9.1 of Visolve: *location of original color before* Visolve *correction*

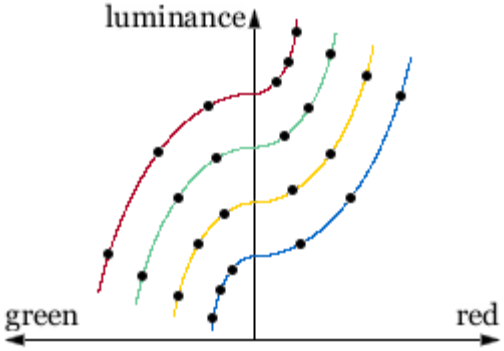


Fig. 9.2 of Visolve: *location of color after* Visolve *correction*

Given figures 9.1 and 9.2, *Visolve*'s implementation provides a more clear explanation of how they correct for colorblind users: the new luminance of the pixel is the old luminance plus a cubic function using the appropriate axis as input (green-red for protanopia and deuteranopia, blue-yellow for tritanopia).

$$\mathbf{L'} = \mathbf{L} + (\mathbf{A}^3) / 2$$
$$\mathbf{L'} = \mathbf{L} + (\mathbf{B}^3) / 2$$

Approximation of Visolve *transformation for green-red and blue-yellow, respectively*

The post-process effects in *SimCity (2013)* utilize a different approach from *Vischeck* and *Visolve*'s CIELAB-oriented approach. *PC Gamer's* Tyler Wilde asked Ocean Quigley, the creative director of *SimCity (2013),* about the game's post-process effects [10]. Quigley replied that he used the *Daltonize* algorithm, an improvement to *Vischeck* published in detail by Fidaner, Lin, and Ozguven [11]. *Daltonize* works entirely in LMS color space and scales the correction effect based on how far off the pixel lies past the visible end of the spectrum.

Daltonize begins by simulating the pixel according to the Brettel *et al* equation and calculates the difference between **V** and **V'**. *Daltonize* transforms **V** again by adding the difference scaled by an error-transform matrix. This linearly maps the portion of information lost to colorblind users to fit within their visible spectrum.

$$\mathbf{V}_{error} = \mathbf{V} - \mathbf{V'}$$
$$\mathbf{V}_{corrected} = \mathbf{V} + \mathbf{E}\mathbf{V}_{error}$$

Correction portion of Daltonize algorithm

$$\begin{bmatrix} 0000 & 0.0000 & 0.0000 \\ 7000 & 1.0000 & 0.0000 \\ .7000 & 0.0000 & 1.0000 \end{bmatrix}$$

*Example error-transform matrix **E***

The full version of the *Daltonize* algorithm from start to finish is as follows, given original input pixel **V**:

$$\mathbf{Q} = \mathbf{TV}$$
Transform from RGB space to LMS space

$$\mathbf{Q} \rightarrow \mathbf{Q'}$$
Reduce from normal vision to dichromatic vision (protanopia, deuteranopia, or tritanopia)

$$\mathbf{V'} = \mathbf{T}^{-1}\mathbf{Q'}$$
Transform from LMS space to RGB space

$$\mathbf{V}_{error} = \mathbf{V} - \mathbf{V'}$$
Calculate difference between normal and dichromatic vision

$$\mathbf{V}_{corrected} = \mathbf{V} + \mathbf{E}\mathbf{V}_{error}$$
Linearly map original pixel from normal vision spectrum to dichromatic vision spectrum

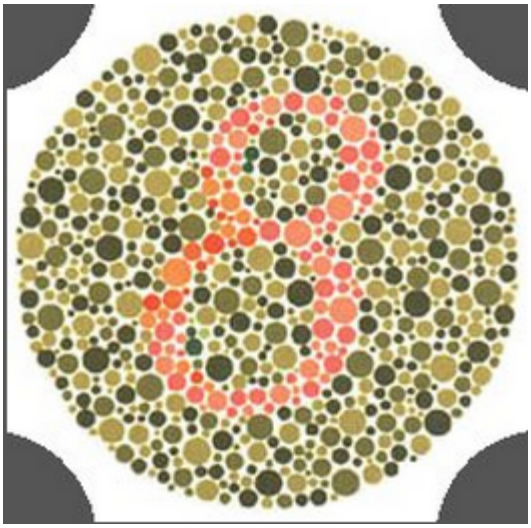
Full summary of Brettel et al *and* Daltonize *algorithms; refer to previous equations for* **Q→Q'** *transforms and values of **T** and **E***

III. Methodology

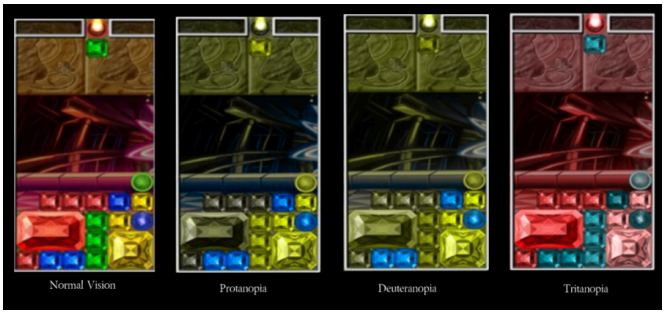
This study seeks to explore rendering techniques to make games easier to play for colorblind people. Doing so would expand the gaming industry's potential audience by approximately five percent. Prior research in the field has found several design-based solutions (e.g. using shapes instead of colors to convey game-critical information). Likewise, several color correction solutions exist, but not at a rate acceptable for real-time games. This study seeks to find a solution that works at the rendering level instead.

The study seeks to find adjustments a post-process technique can make to the final render of a game's frame to correct the colors to within a spectrum that a person with dichromatic vision can see. (Because Brettel et al. covered only dichromatic vision simulation, correction for anomalous trichromacy remains as a topic to pursue.) The intended color correction algorithms utilize either LMS color space to correct the colors from a physiological level or CIELAB color space to correct the color space from a neurological level. The algorithms work as described earlier -

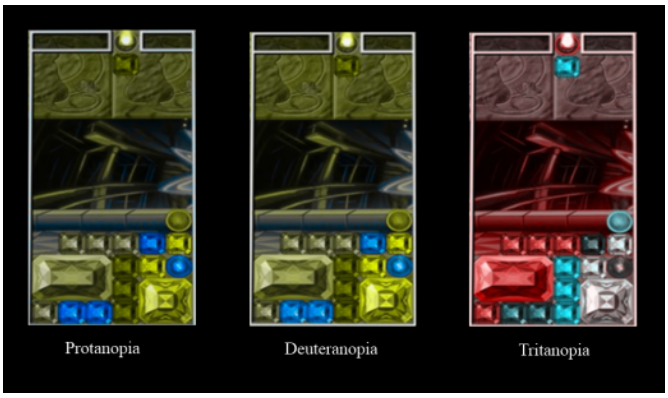
The first artifact for this study consists of a DirectX 11 fragment/pixel shader implementation of the algorithm provided by Brettel, Viénot, and Mollon and another fragment/pixel shader implementation of a color correction algorithm utilizing LMS color space. To demonstrate its correctness, the only input consists of a single texture. Test textures consist of screenshots from existing games [4] and Ishihara plates [12] meant to test for each kind of colorblindness. The algorithm passes inspection if the details of the image obscured by the Brettel *et al* colorblind simulation reappear in the output of the color correction algorithm, and if the artifact's implementation continually performs its color correction task with a refresh rate within five percent of the same texture's refresh rate without correction applied.



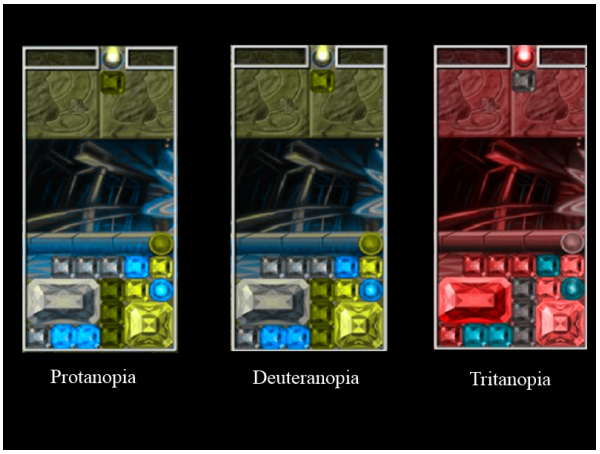
Sample Ishihara plate texture [12] used to detect protanopia and deuteranopia (red-green)



Screenshots from Puzzle Fighter [4], simulating normal vision and different varieties of colorblind vision



Screenshots from Puzzle Fighter [4], post CIELAB-based correction (Visolve implementation)



Screenshots from Puzzle Fighter [4], post LMS-based correction (Daltonize implementation)

The second artifact for this study consists of the same implementation with a full three-dimensional scene with a movable camera as its input instead of a static texture. The algorithm passes inspection if it continually performs its color correction task with a refresh rate within five percent of the same scene's refresh rate without correction applied.

The third artifact for this study consists of a video capture tool which overlays another game in progress with two modes. The first mode helps developers double-check that their game does not impede those with colorblind vision. This mode simulates the visual display that a colorblind person receives – it subtracts colors outside of the spectrum that a colorblind person can see. The other mode helps colorblind people and automatically adjusts the color of the frame so it retains the contrast intended (e.g. correcting a red-brown adjacent to a red-green to some other configuration with sharper contrast). Each of these modes has three sub-modes that individually simulate/correct for protanopia, deuteranopia, and tritanopia. This artifact passes inspection if with a refresh rate within five percent of the same texture's refresh rate without correction applied.

Tools used in the creation of the artifacts consist of the Visual Studio 2010 C++ compiler, the DirectX11 API, and the GDI screen capture API [13]. Test input for the artifacts consist of screenshots from already existing games and three-dimensional scenes created in 3ds Max which cause color confusion in colorblind individuals.

IV. Results

The first part of the artifact's construction consisted of the Brettel *et al* simulation. Creation of the HLSL shader to simulate colorblind vision proved simple to implement, as the rendering engine of choice already had the architecture to add post-process shaders. The in-engine colorblindness simulation output mirrored offline versions of colorblindness simulation output and passed inspection.

Initially, the artifact's correction shaders utilized an approach similar to the *Daltonize* algorithm, but with no success – the error in this first approach was that it did its error detection and correction scaling while still in LMS space instead of first converting back to RGB space. This made finding a proper **E** difficult, especially by experimentation only. While the results for tritanopia were passable, protanopia and deuteranopia lost too much color information in the process: portions of test textures that were previously clear to both normal and colorblind vision alike became indistinguishable.

Upon further research, the correction algorithm was changed to use CIELAB instead to mirror the approach taken by *Vischeck* and *Visolve*. Results for all three kinds of colorblindness proved satisfactory and struck a balance between correction and natural color retention. For the *Puzzle Fighter* test texture, normal vision perceived four distinct block colors, simulated vision perceived only three distinct colors, and correction applied to the simulation restored that to four distinct colors again. One further improvement was made to better fit the needs of users with tritanopia:

$$\mathbf{L'} = \mathbf{L} + ((3 \times \mathbf{B}^3) + (\mathbf{A}^3)) / 8$$

Improvement made to Visolve *blue-yellow filter*

Because the visible spectrum for tritanopia curves, the new equation uses an axis that intersects the **A** and **B** axes at an angle, roughly perpendicular to the tritanopia visible spectrum. While a rough approximation, it suffices and provides more clarity than the original blue-yellow filter.

After the completion of the *Visolve* version of correction shaders, work began on bringing the correction effect out of engine and to pre-existing games. The artifact continually took screenshots of the user's desktop via the Windows GDI API and displayed it as a texture applied to a fullscreen quad.

Screen capture proved to be a slow process, substantially reducing framerate, but the process successfully applied to existing games.

After the completion of the desktop-wide correction feature – quite late into the artifact's life cycle – Maxis released *SimCity (2013)*, and Quigley spoke of the *Daltonize* algorithm. After some research, the algorithm was implemented alongside the *Visolve* for comparison purposes. As expected, the *Daltonize* algorithm also provided results similar in quality to *Visolve*, even if it differs by approach. Where *Visolve* changes value, *Daltonize* changes hue. *Daltonize* correction applied to Brettel *et al* simulation restored the distinction of all four colors of blocks on the *Puzzle Fighter* test texture.

Framerates of each effect are shown in tables 1 and 2:

TABLE I

FRAMERATE OF IN-ENGINE REALTIME IMPLEMENTATIONS OF COLOR CORRECTION POST-PROCESS EFFECTS

Post process	Framerate	Difference
No effect:	52.7 FPS	N/A
Simulation:	52.4 FPS	0.6%
Visolve Correction:	52.5 FPS	0.4%
Daltonize Correction:	52.3 FPS	0.8%

TABLE 2

FRAMERATE OF SCREEN CAPTURE REALTIME IMPLEMENTATIONS OF COLOR CORRECTION POST-PROCESS EFFECTS

Post process	Framerate	Difference
No effect:	15.0 FPS	N/A
Simulation:	14.7 FPS	2.0%
Visolve Correction:	14.8 FPS	1.3%
Daltonize Correction:	14.6 FPS	2.6%

V. CONCLUSIONS

Given the minimal framerate loss from the implementation of the *Visolve* or *Daltonize* algorithms as a post-process effect, a real-time correction solution serves as a viable alternative to production time spent per asset. While the best solution to avoid a performance cost is to design the game to be colorblind-friendly from the ground up, if a studio lacks the time or foresight to do so, implementation of a post-process effect does not scale with the number of assets that need to be designed or changed. In short, it is a flat production cost and a flat performance cost – neither cost scales.

Implementation of tools that apply the effect to anything not native to application with the shader proves to remain a difficult challenge. The GDI API, while a good proof of concept, incurs too much of a framerate loss to be considered viable. In the future, perhaps a driver-level or OS-level implementation would serve as better alternatives to research: manufacturers could add an accessibility option to graphics cards or Windows settings, for instance – one such library already exists for Debian builds of Linux. Future iterations of the HLSL shaders could include uniforms that correspond to the severity of colorblind deficiency that the user has: Currently, the shaders assume that a user has either perfect color vision or a perfectly nonfunctional set of L-, M-, or S-cones, but a setting to interpolate between functional and nonfunctional could exist.

The purpose of the study was to observe the framerate loss incurred by a post-process effect that matches existing correction algorithms. Future research could also investigate which algorithm proves more effective at quickly relaying information (i.e. "in a time-sensitive setting, does it take longer for colorblind users to perceive detail *Visolve* than it does *Daltonize*?").

VI. REFERENCES

[1] *Awesomenauts,* Ronimo Games, May 2012 – Present (ongoing development)

[2] *Audiosurf,* Invisible Handlebar, February 2008

[3] *League of Legends*. Riot Games. July 2009 - Present (ongoing development)

[4] *Super Puzzle Fighter II Turbo* (aka *Puzzle Fighter*). Capcom. May 1996

[5] *SimCity.* Maxis. March 2013

[6] Fairchild, Mark. *Color Appearance Models, 2nd Edition*. Chichester, West Sussex, England: Wiley, 2005.

[7] Brettel, Hans, Mollon and Vienot. "Computerized Simulation of Color Appearance for Dichromats." *Journal of the Optical Society of America A* **14** (1997): 2647 - 2655. http://vision.psychol.cam.ac.uk/jdmollon/papers/Dichromatsimulation.pdf (accessed August 2012).

[8] Dougherty, Bob and Wade. "FAQ - How does it work?" Vischeck. 2008, http://www.vischeck.com/faq/ (accessed August 2012).

[9] Ryobi System Solutions. "Color transformation concept." Visolve. http://www.ryobi-sol.co.jp/visolve/en/concept.html (accessed January 20, 2013).

[10] Wilde, Tyler. "Why games need color blind modes." PC Gamer.

http://www.pcgamer.com/2013/02/01/why-games-need-color-blind-modes-see-simcity-with-simulated-color-blindness/. February 2013 (accessed March 2013).

[11] Fidanel, Lin, and Ozguven. "Analysis of Color Blindness." http://scien.stanford.edu/pages/labsite/2005/psych221/projects/05/ofidaner/colorblindness_project.htm. *Information Systems Laboratory* at *Stanford University,* 2005 (accessed March 2013).

[12] *Ishihara 38 Plates CVD Test.* Colblindor. http://www.colblindor.com/ishihara-38-plates-cvd-test/#prettyPhoto. November 2012 (accessed January 2013).

[13] *About GDI+*. MSDN Library. http://msdn.microsoft.com/en-us/library/ms533797.aspx. March 2012 (accessed March 2013).