UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JÚLIA PEIXOTO VIOLATO

Real-Time Computer Game Recoloring for Dichromats

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Science

Advisor: Prof. Dr. Manuel Menezes de Oliveira Neto

Porto Alegre
October 2022

ABSTRACT

The ability to correctly perceive and differentiate colors is crucial for a wide variety of day-to-day activities. In video games, colors are often used to communicate important information to the players. For instance, green and red often indicate a character's health, and can become essential to the construction of the game's narrative and to ensure coherent playability. Thus, individuals with color vision deficiency are typically constrained in their ability to interact with games in the way intended by the developers. In games with competitive or online aspects, this can introduce unfair disadvantages, as such players might lack precious information, requiring them more time to determine a course of action. The impact of this information loss might even discourage these players entirely from interacting with a game. In this work, we provide a contrast-enhancing, temporal-coherent recoloring plugin for video games to assist dichromat players. For this, we use ReShade, a post-processing injector that exposes color and depth information to enable customized shaders and filters on games. Our solution is applicable to any game based on OpenGL, Vulkan, or DirectX, running on Windows platforms. By reducing color ambiguity for dichromat players, it is expected that our work will improve the experience of individuals with color vision deficiency when playing video games on these platforms.

Keywords: Dichromat. color vision deficiency. recoloring. computer graphics. video games.

Recoloração em Tempo Real de Jogos para Computador para Dicromatas

RESUMO

A capacidade de perceber e diferenciar cores é essencial em uma grande variedade de atividades do dia-a-dia. Em video games, cores são muito utilizadas para comunicar informações importantes ao jogador. Por exemplo, verde e vermelho frequentemente indicam a saúde de um personagem, e podem se tornar essenciais para a construção da narrativa do jogo e para garantir uma jogabilidade coerente. Por isso, indivíduos com daltonismo tipicamente sofrem limitações em sua habilidade de interagir com jogos da forma imaginada pelos desenvolvedores. Em jogos com aspectos competitivos ou online, isso pode introduzir desvantagens, já que informações valiosas podem faltar a esses jogadores, exigindo que eles usem mais tempo para determinar uma ação a ser tomada. O impacto dessa perda de informação pode até mesmo desencorajar esses jogadores de sequer interagir com um jogo. Nesse trabalho, nós providenciamos um plugin de recoloração para video games, que aumenta o contraste e preserva coerência temporal, para auxiliar jogadores dicromatas. Para isso, nós utilizamos o ReShade, um injetor de pós-processamento que expõe informações de cor e profundidade que permitem o desenvolvimento de shaders e filtros personalizados para jogos. Nossa solução é aplicável a qualquer jogo baseado em OpenGL, Vulkan ou DirectX, que seja executado em plataformas Windows. Ao reduzir a ambiguidade de cores para jogadores dicromatas, é esperado que nosso trabalho melhore a experiência de indivíduos daltônicos ao jogar jogos nessas plataformas.

Palavras-chave: dicromata, dicromatopsia, recoloração, computação gráfica, jogos eletrônicos.

LIST OF FIGURES

Figure 1.1 Protanopia filtering in Final Fantasy XIV: A Realm Reborn........................11
Figure 1.2 Deuteranopia filtering in Final Fantasy XIV: A Realm Reborn. 12
Figure 1.3 Protanopia filtering in Final Fantasy XIV: A Realm Reborn........................12

Figure 2.1 Colorblind mode as seen in games in the Call of Duty series. Source: (HARDIN, 2016). 20
Figure 2.2 Deuteranopia colorblind mode in Overwatch (2016). 21
Figure 2.3 Tritanopia colorblind mode in Overwatch (2016). Source: (HARDIN, 2016). 22

Figure 3.1 Planar approximation for the color gamut of dichromats in the CIE L*a*b* color space. (a) Protanope. (b) Deuteranope. (c) Tritanope. Source: (MACHADO; OLIVEIRA, 2010). 24
Figure 3.2 Steps of the recoloring algorithm. Source: (MACHADO; OLIVEIRA, 2010). 26

Figure 4.1 ReShade's user interface overlay in Supergiant's Hades (2018), with no plugins enabled. 28
Figure 4.2 ReShade's user interface overlay in Supergiant's Hades (2018), with our plugin enabled...29
Figure 4.3 Dropdown menu to select dichromacy type in our plugin............................29
Figure 4.4 Projection of p_i onto a plane defined by point $o = (0, 0, 0)$ and the normal vector $b*'$..33
Figure 4.5 Custom portrait selection feature in Pathfinder: Kingmaker (2018) with example image from the original article. 36
Figure 4.6 Example image recolored with our plugin in Pathfinder: Kingmaker (2018). The area outside the custom portrait display is manually set to white by our shader before the recoloring process. 36
Figure 4.7 Medical visualization image of a skull (a) and its recoloring done with the original Real-Time Temporal-Coherent Color Contrast Enhancement for Dichromats technique (b). Source: (MACHADO; OLIVEIRA, 2010)..................36

Figure 5.1 Original colors, protanopia simulation, and recoloring of the menu in Hades (2020)...38
Figure 5.1 Original colors, protanopia simulation, and recoloring of the menu in Hades (2020)...39
Figure 5.2 Original colors, deuteranopia simulation, and recoloring of an arena in Hades (2020)...39
Figure 5.2 Original colors, deuteranopia simulation, and recoloring of an arena in Hades (2020)...40
Figure 5.3 Original colors, tritanopia simulation, and recoloring of scenery in Hades (2020)...40
Figure 5.3 Original colors, tritanopia simulation, and recoloring of scenery in Hades (2020)...41
Figure 5.4 Original colors, deuteranopia simulation, and recoloring of the map in Shadow of the Tomb Raider (2018)..42
Figure 5.5 Original colors, protanopia simulation, and recoloring of an example of the hunting mechanic in Shadow of the Tomb Raider (2018). 43

Figure 5.6 Original colors, tritanopia simulation, and recoloring in Tell Me Why (2020). 44
Figure 5.6 Original colors, tritanopia simulation, and recoloring in Tell Me Why (2020). 45
Figure 5.7 Original colors, protanopia simulation, and recoloring of UI elements in Final Fantasy XIV: A Realm Reborn (2013). 45
Figure 5.8 Original colors, protanopia simulation, and recoloring of ground markers in a Final Fantasy XIV: A Realm Reborn (2013) boss fight arena........ 46
Figure 5.9 Original colors, protanopia simulation, and recoloring of a boss fight arena in Final Fantasy XIV: A Realm Reborn (2013). 46
Figure 5.10 Original colors, deuteranopia simulation, and recoloring of ground markers in a Final Fantasy XIV: A Realm Reborn (2013) boss fight arena. 47
Figure 5.11 Original colors, tritanopia simulation, and recoloring of a ground marker in a Final Fantasy XIV: A Realm Reborn (2013) boss fight arena. 47
Figure 5.12 Original colors, deuteranopia simulation, and recoloring of UI elements in Final Fantasy XIV: A Realm Reborn (2013)........ 48
Figure 5.13 Original colors, protanopia simulation, and recoloring of the player's home in Stardew Valley (2016)........ 48
Figure 5.13 Original colors, protanopia simulation, and recoloring of the player's home in Stardew Valley (2016)........ 49
Figure 5.14 Original colors, deuteranopia simulation, and recoloring of a cave in Stardew Valley (2016)........ 49
Figure 5.14 Original colors, deuteranopia simulation, and recoloring of a cave in Stardew Valley (2016)........ 50
Figure 5.15 Original colors, tritanopia simulation, and recoloring of a town in Stardew Valley (2016)........ 50
Figure 5.15 Original colors, tritanopia simulation, and recoloring of a town in Stardew Valley (2016)........ 51

LIST OF TABLES

Table 5.1 Average frames per second measured. 52
Table 5.2 Average per-frame plugin execution time. 52

LIST OF ABBREVIATIONS AND ACRONYMS

CVD	Color Vision Deficiency
PCA	Principal Component Analysis
UI	User Interface

CONTENTS

1 INTRODUCTION **10**
2 RELATED WORK **14**
2.1 Recoloring Web Pages **14**
2.1.1 ColorBlind Filter Service 14
2.1.2 Wakita and Shimamura 14
2.1.3 Ichikawa et al. (2003) 15
2.2 Recoloring Images **15**
2.2.1 Ichikawa et al. (2004) 15
2.2.2 Daltonize 16
2.2.3 Rasche et al. 16
2.2.4 Kuhn et al. 17
2.2.5 Jefferson and Harvey 17
2.3 Contrast-Preserving Grayscale Mapping **18**
2.3.1 Color2Gray 18
2.3.2 Decolorize 19
2.4 Accessibility **20**
2.5 Summary **21**
3 BACKGROUND **23**
3.1 Real-Time Temporal-Coherent Color Contrast Enhancement for Dichromats **23**
3.1.1 Maximum contrast loss in the CIE *L*a*b** color space 23
3.1.2 Principal Component Analysis 24
3.1.3 Computing results 25
3.1.4 Ensuring temporal coherence 26
3.2 Summary **27**
4 IMPLEMENTATION **28**
4.1 ReShade **28**
4.2 ReShade FX **29**
4.3 Recoloring Plugin **30**
4.3.1 RGB to CIE *L*a*b** convertion 30
4.3.2 Generating random noise 31
4.3.3 Principal Component Analysis 32
4.3.4 Computing final colors 33
4.3.5 Temporal Coherence 35
4.4 Difficulties **35**
4.5 Summary **37**
5 RESULTS **38**
5.1 Recoloring **38**
5.2 Performance Analysis **51**
5.3 Summary **53**
6 CONCLUSION AND FUTURE WORK **54**
REFERENCES **56**

1 INTRODUCTION

Color vision deficiency is a group of conditions that affect an individual's ability to perceive color. Human color vision is possible due to three types of photoreceptors, called cones, present in the retina. The photopigments in the human cones are sensitive to long, medium or short wavelengths. When these cones are lacking in the light-sensitive pigments that make this color perception possible, the individual is unable to perceive different shades of these colors regularly (MACHADO; OLIVEIRA, 2010; NEITZ; NEITZ, 2000; GORDON, 1998). While some illnesses (such as diabetes, glaucoma, and macular degeneration) might lead to color vision deficiency later in life (SAMPLE; WEINREB; BOYNTON, 1986; SCOTT; FEUER; JACKO, 2002; JUDD, 1949), the cause of CVD is essentially genetic. No clinical or surgical treatment is known for CVD.

The most common form of CVD, observed in around 8% of men and 0.5% of women, is a difficulty distinguishing between shades of red and green, which characterizes two of the three types of dichromacy, protanopia and deuteranopia. The third type, tritanopia, affects the individual's ability to differentiate shades of blue and yellow, but is considerably less common, with an incidence of about 0.01% on both men and women (SHARPE et al., 1999). Other types of color vision deficiency observed are anomalous trichromacy, a condition in which no photoreceptor is lacking in the eye, but one type of cone perceives light slightly out of alignment, and monochromacy, or achromatopsia. Individuals with achromatopsia are unable to perceive any colors, and see instead different shades of gray.

CVD has a negative impact on certain aspects of life, from children having issues with color-based learning methods, to identifying whether food is unsafe to eat based on its appearance. For most people, it is natural to use color perception to gather information that is not only essential, such as reading warning signs, but also to make daily life more comfortable, like easily finding a car in a parking lot.

In this work, we focus on the experience that dichromats have when interacting with electronic games, particularly those that rely on color elements to present relevant information to the player. While video games have been a popular form of entertainment ever since their creation, the most recent years have seen a global record rise in that popularity, as is reflected in the growth of the video-game industry: Unity Technologies, for example, reports a 46% growth in their daily active users during the spring of 2020 as compared to that same period in 2019 (IRPAN et al., 2020).

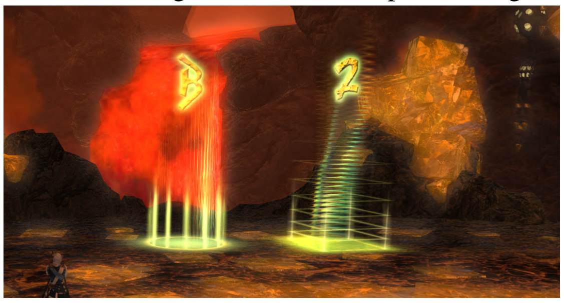
As of 2022, market research and consulting firm DFC Intelligence reports that over 3 billion people worldwide are regular video game users (DFC INTELLIGENCE, 2022). Since this number contemplates around 40% of the world's population, it follows that a significant part of this group can be expected to display some form of color vision deficiency.

With its popularity as a form of entertainment and the economic impact of the video games industry worldwide, electronic games should strive to be accessible to anyone who wants to play them, with as little disadvantages as possible when compared to players with normal color vision.

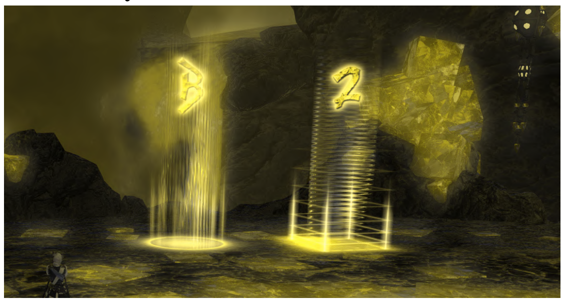
Nowadays, many games offer at least some form of minimizing the effects of color vision deficiency through accessibility settings. However, this cannot be said of all games, nor is the quality of these accessibility settings standard across the games that do offer them.

Very often, color blind filtering options provided by games do not offer reliable results. In the 2013 game Final Fantasy XIV: A Realm Reborn, for instance, the color changes perceived by color deficient players are hardly noticeable. This is an online game who's latest update was released in December of 2021, and yet no updates to its color blind accessibility options have been made in the years since the initial release.

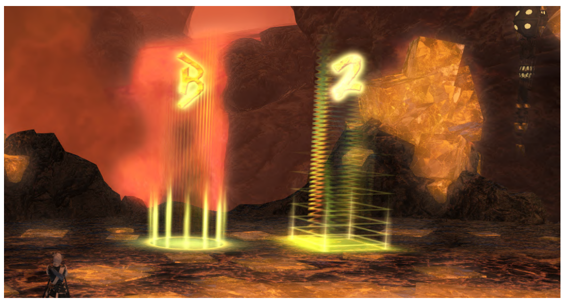
Figure 1.1 – Protanopia filtering in Final Fantasy XIV: A Realm Reborn.



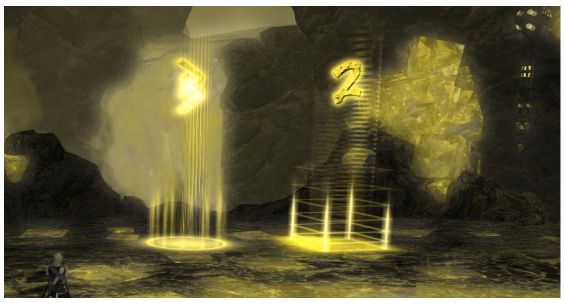
(a) Normal color vision.



(b) Protanopia simulation.



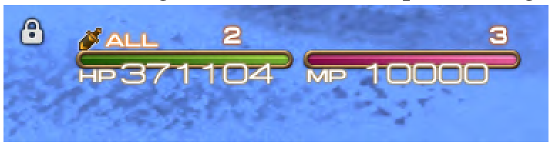
(c) Game's protanopia filtering, as seen by players with normal color vision.



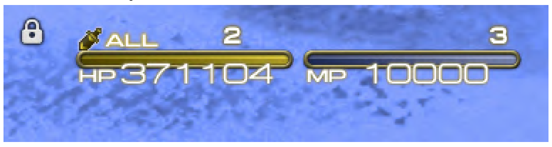
(d) Protanopia filtering in Final Fantasy XIV: A Realm Reborn, as seen by protanopes.

To exemplify some core elements of this game, as they are seen by players with full color vision and dichromats, Figure 1.1 shows one of the game's arenas where a boss fight takes place, with ground markers frequently used by players to coordinate movement around the area. Figures 1.2 and 1.3 show a player's health and magic gauges, which provide essential information in the context of the game, in different colored backgrounds. In all cases, the image shown above displays a portion of the game's screen with no filtering, and the image below displays that same portion with the game's color blind filters turned on. To simulate the vision of a dichromat, we projected the colors onto the plane representing the gamut of colors the dichromat can see in the CIE $L^*a^*b^*$ color space.

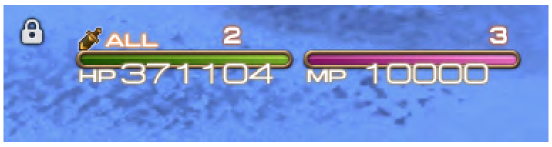
Figure 1.2 – Deuteranopia filtering in Final Fantasy XIV: A Realm Reborn.



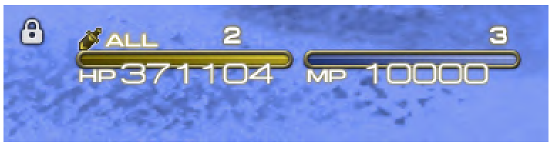
(a) Normal color vision.



(b) Deuteranopia simulation.

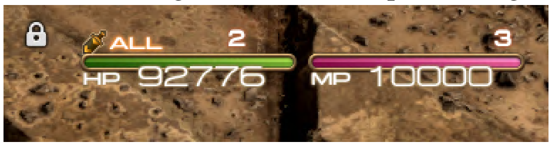


(c) Deuteranopia filtering in Final Fantasy XIV: A Realm Reborn, as seen by players with normal color vision.

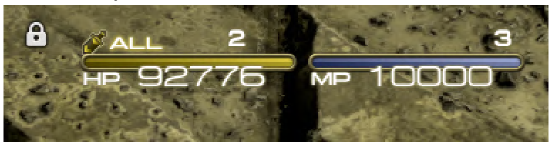


(d) Deuteranopia filtering in Final Fantasy XIV: A Realm Reborn, as seen by deuteranopes.

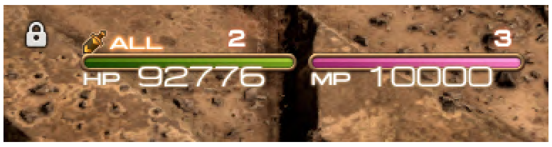
Figure 1.3 – Protanopia filtering in Final Fantasy XIV: A Realm Reborn.



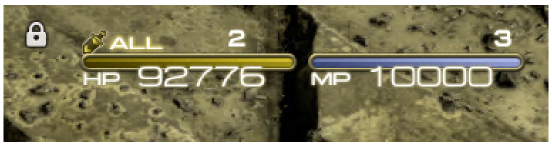
(a) Normal color vision.



(b) Protanopia simulation.



(c) Protanopia filtering in Final Fantasy XIV: A Realm Reborn, as seen by players with normal color vision.



(d) Protanopia filtering in Final Fantasy XIV: A Realm Reborn, as seen by protanopes.

While it might be safe to say that the cases in which these effects can make a game seem unplayable to a player with CVD are a minority, having a character in the game die because their health bar was not clearly visible in the background, for example, can certainly make the player's experience more uncomfortable. In online games where players have to compete with each other, this increased difficulty in perceiving information puts players with CVD at disadvantage.

The main goal of this undergraduate thesis is to implement a simple to use, generic recoloring filter for electronic games for Windows platforms, applying an existing algorithm to offer dichromat players a method to satisfyingly minimize the visual information loss caused by color vision deficiency.

The remaining of this thesis is organized as follows: Chapter 2 describes related work on the topic of image recoloring for dichromats. Chapter 3 presents in detail the algorithm used in the development of our plugin. Chapter 4 details the implementation of our plugin. Chapter 5 presents results obtained. Finally, Chapter 6 presents our conclusions and discusses possibilities for future work.

2 RELATED WORK

This chapter presents the techniques that have inspired of influenced the work discussed in this thesis. The article that serves as basis for this thesis is covered in detail in Chapter 3. The variety of algorithms described here have been developed in an effort to aid individuals with color vision deficiency when interacting with digital images and videos. The existing techniques for image and video recoloring presented have a few different goals: recoloring Web pages, recoloring images and multiframe sequences, or both. In addition, we also present contrast-preserving color to grayscale mapping techniques, and the accessibility options CVD players can find in games.

2.1 Recoloring Web Pages

2.1.1 ColorBlind Filter Service

In 2006, Iaccarino et al. (IACCARINO et al., 2006) proposed a tool for Web applications to improve accessibility for dichromats by modifying the colors in HTML pages and embedded images. Its focus was on protanopes and deuteranopes, as its goal was to enhance the contrast for shades of red and green. This algorithm, however, was designed to be customizable. It operates on the HSL color space and allows users to manually pick the proportions by which the color coordinates are changed. To do so, it relies on the parameters provided to perform the recoloring.

2.1.2 Wakita and Shimamura

Wakita and Shimamura proposed in 2005 the SmartColor (WAKITA; SHIMAMURA, 2005), an algorithm that aims to infer the author's intentions behind the colors used in a Web document, such as emphasizing a text passage. The recoloring process then has the goal of most effectively conveying these intentions to the color blind user.

These "intentions" of the author are described by three desirability functions, all of which take into account the author of the document's desire to keep an element's color unchanged. SmartColor condenses them into a single desirability function, and recolors the image by finding the colors that maximize this function. The search space of this algorithm, however, grows exponentially with the number of colors in the document, and so it is computationally expensive. Although most Web documents have a limited number of colors, it is not possible to use SmartColor in an interactive manner for an amount as small as 10 colors.

2.1.3 Ichikawa et al. (2003)

Ichikawa et al. (ICHIKAWA et al., 2003) presented a method to recolor Web pages for anomalous trichromats using an improved genetic algorithm to generate optimal values for each original color in order to make the HTML page more readable for those users. The algorithm operates on the Luv color space (where L represents the luminance and u and v represent chromaticity values for each color) to evaluate each individual during the optimization process.

Ichikawa et al. define an abstracted image model, where color images are divided in color regions that can be either "even" with each other, or "included" in a parent region. Then, they apply this model to Web pages by considering, for example, the background of the page a parent region in which the other elements are "included", and different colored text to have an "even" relation among them. By doing this, their work doesn't analyze each pixel on a screen, but rather each *element* in the HTML document that has a color associated with it.

2.2 Recoloring Images

2.2.1 Ichikawa et al. (2004)

In 2004, the authors expanded on their abstract image model (ICHIKAWA et al., 2003) from Web pages to still images (ICHIKAWA et al., 2004). In this case, to apply this model to a given image, first it has to be decomposed. All colors in the image are quantized to obtain n representative colors, then each pixel is assigned a region G_k. This is done by finding the representative color C_k with the smallest distance to the pixel color. After this process, the abstract image model can be applied to describe the relationship between these n regions.

This time, instead of using their genetic algorithm, the authors use a random bitclimber to obtain the optimal colors C'_k for each region represented by C_k. Then, the n differences between each C'_k and its corresponding C_k are calculated and used to adjust the color of the individual pixels.

2.2.2 Daltonize

Daltonize (DOUGHERTY; WADE, 2002) combines two image processing techniques to preserve image information for people with CVD, also with focus on protanopes and deuteranopes. The first of these techniques consists of increasing the red and green contrast in the image, since anomalous trichromats can have partial color perception instead of a complete loss. The second technique involves converting red-green variations into blues and yellows, and changes in brightness.

To guide these steps, Daltonize accepts three input parameters. They define how much the red and green contrast should be increased, how those colors should be projected onto the luminance channel, and how they should be projected onto blues and yellows. A game recoloring filter provided by ReShade (CROSIRE, 2015) implements the LMS Daltonization algorithm (DALTONIZE.ORG, 2010). This algorithm is based on the Daltonize process, after which the filter is named, but does not require the user to provide any parameters and encompasses all three types of dichromacy. Instead, the LMS Daltonization technique, as implemented by this filter, consists of first converting the RGB colors in the image into the LMS color space. Then, color blindness is simulated by bringing the corresponding channel values down to zero (the L channel for protanopes, M for deuteranopes, and S for tritanopes), and this loss is partially compensated by adding to the other channels. The resulting image is then obtained by converting the adjusted LMS values back into RGB.

2.2.3 Rasche et al.

Rasche et al. present a recoloring technique based on preserving the perceptual distances (RASCHE; GEIST; WESTALL, 2005a), but for all pairs of colors present in an image. This method is based on contrast-preserving grayscale mappings, and its optimization step consists of minimizing an affine transformation. However, the results of this process do not guarantee that all of the colors used belong to the gamut perceived by the dichromat, and does not take into account color variations in more than one direction.

These limitations were then addressed by the author in a following work (RASCHE; GEIST; WESTALL, 2005b), where the colors are first quantized into a smaller set, and then a constrained, multi-variate optimization process is applied to them. The recolored set is then used to recolor the full image. The results obtained, however, while improved, are still prone to local minima, and the algorithm does not scale well for larger sets of quantized colors.

2.2.4 Kuhn et al.

Kuhn et al. presented a recoloring algorithm based on enhancing color contrast (KUHN; FERNANDES; OLIVEIRA, 2008) using a mass-spring optimization process, with the goal of preserving the naturalness of the image. In the same manner as other algorithms described, this technique is applied on a quantized set of colors, and uses the resulting set to optimize the entire image. To do this, in the CIE $L^*a^*b^*$ space, this algorithm associates each quantized color with a particle initialized with the coordinates of the color perceived by the dichromat. Each pair of particles is then connected by a spring with elasticity coefficient 1 and rest length equal to the Euclidean distance between their colors. To preserve the naturalness in the image, each particle is assigned a mass reciprocal to the perceptual distance between its color and the color perceived by the dichromat. Although this approach is considerably faster than previous ones, it is still not fast enough to be applied in real-time, such as in electronic games. Its use of a subset of the original colors also does not guarantee temporal-coherence for use in interactive environments.

2.2.5 Jefferson and Harvey

Jefferson and Harvey present another algorithm designed both for the recoloring of Web documents and images (JEFFERSON; HARVEY, 2006), based on preserving the perceptual distance between the colors that an individual with normal color vision observes, for the colors that a person with CVD observes. This process is based on minimizing the combination of a set of objective functions to preserve brightness and contrast, use colors present in the dichromat's color gamut, and preserve naturalness in the recolored image or document.

Due to the optimization cost, this is done for a subset of the original colors, and the results are interpolated for the rest of the colors using inverse-distance weighting interpolation (SHEPARD, 1968). The weight associated with a color is the sum of the inverse squares of its distances from the key colors in the subset used. However, the authors still reported the times required for execution as several minutes for any subset larger than 25 colors.

2.3 Contrast-Preserving Grayscale Mapping

In our work, we aim to minimize the information loss experienced by players with CVD. Our recoloring process does this by preserving that information, as much as possible, through contrast. Although we still have the advantage of working with a reduced range of colors, rather than a monochromatic scale, this is a goal shared with contrast-preserving color-to-grayscale algorithms.

The standard grayscale techniques most often used in commercial applications, like Photoshop, simply map each color to its luminance value in a given color space. This keeps achromatic colors unchanged, but maps different colors that have the same luminance to the same shade of gray.

2.3.1 Color2Gray

A technique to work around mapping different colors to the same shade of gray, proposed in 2005 by Gooch et al. (GOOCH et al., 2005), uses a least-squares optimization process to modulate the differences in luminance and chromaticity values. These differences are calculated between each pixel and its given neighborhood, in a perceptually uniform color space, and the changes in luminance and chromaticity in this space are then reflected onto the original image's chrominance.

The Color2Gray algorithm admits three customizable parameters to guide this process: one to control whether chromatic differences are mapped to increases or decreases in luminance, one to determine how much the chromatic variation influences the changes in luminance values, and one to determine the size of the neighborhood used.

The algorithm produces good results, but as shifts luminance values to enhance contrast between regions without checking for pixels that were originally gray, it does not preserve achromatic shades like the traditional techniques do. Additionally, it has a computational cost quadratic in the number of pixels in the image. This makes it unsuitable for use with interactive applications.

According to the authors, they have attempted to use principal component analysis to estimate an ellipsoid that best approximates the image's color distribution. The grayscale image could then be obtained by projecting the colors onto the axis of the ellipsoid with the largest variance. However, they point out, along with Rasche et al. (RASCHE; GEIST; WESTALL, 2005b), that for this method to work for images with variations in multiple directions, it would require an optimization step to combine these components, therefore keeping the high computational cost.

2.3.2 Decolorize

To compensate for the contrast loss, Grundland and Dodgson (GRUNDLAND; DODGSON, 2007) add an amount K_i to the original luminance of each pixel p_i. However, to avoid having a quadratic cost when calculating the K_i values, this technique uses a local sampling method called *Gaussian pairing*.

This Gaussian pairing strategy consists of randomly selecting, for each pixel p_i, a pixel p_j from a circular neighborhood around it. The neighborhood size is computed from the dimensions of the original image, and p_j is selected based on a normal probability distribution function.

All the colors in the image are converted to a non-perceptually uniform color space, YQP, created by the authors, where Y represents the luminance channel, and Q and P represent opponent-color chromatic channels. The contrast loss for each pixel p_i is then calculated by comparing the differences in luminance (Y), and in the original RGB colors, between p_i and its selected neighbor p_j.

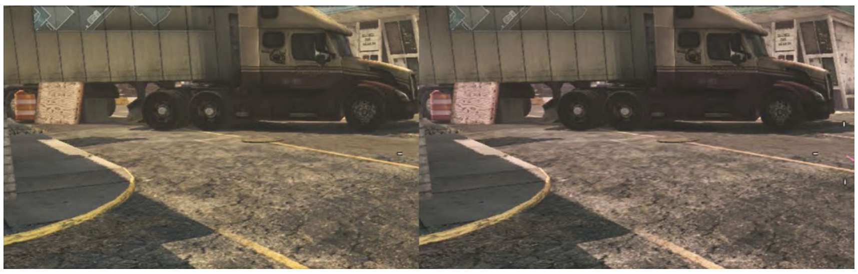
$$Loss_{(p_i,p_j)} = \frac{Y_i - Y_j}{||RGB_i - RGB_j||} \tag{2.1}$$

To estimate the direction of maximum contrast loss in the YQP space, the authors use a technique called *predominant component analysis*. This technique uses a sum of weighted vectors, where $Loss_{(p_i,p_j)}$ is the weight assigned to the $p_i - p_j$ vector, to approximate this direction in the PQ chromaticity plane. The values for K_i are then calculated by projecting the colors onto this vector.

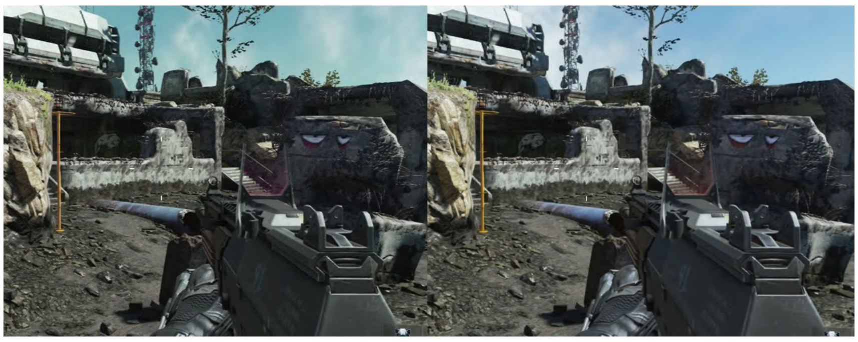
2.4 Accessibility

As previously mentioned, more and more games being released or updated in recent years now offer its player base with a range of different options to improve the experience of players with disabilities. Among those, recoloring filters for players with CVD are very often present.

Figure 2.1 – Colorblind mode as seen in games in the Call of Duty series. Source: (HARDIN, 2016).



(a) Colorblind mode in Call of Duty: Ghosts (2013). Left: original image. Right: recolored image.

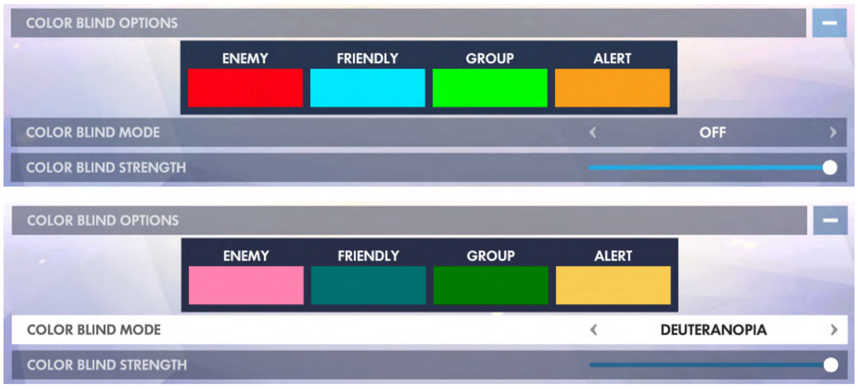


(b) Colorblind mode in Call of Duty: Advanced Warfare (2014). Left: original image. Right: recolored image.

However, these settings have a tendency to simply add an overlay filter to the game's screen, to apply a different hue to or oversaturate the entire color palette used. Figure 2.1 displays an usage of a reddish tint over the image in the Activision game series Call of Duty, in an attempt to make the colors more discernible to a player with CVD. In Figure 2.1b, especially, the effect is just barely perceptible.

In some cases, this can even have the opposite of the intended effect. Figures 2.2 and 2.3 illustrate the results of the tritanopia accessibility option provided by the Blizzard Entertainment game Overwatch (2016) as they are perceived by a player with normal color vision, compared to what deuteranopes and tritanopes see. In Figure 2.2, the deuteranopia mode makes the "group" and "alert" colors more different, but makes the "enemy" and "friendly" colors more similar. For tritanopes (Figure 2.3), the contrast between the "enemy" and "alert" labels, and the "friendly" and "party" labels, are less perceptible with the tritanope filter than without. As of the last update to Overwatch, in September of 2022, these accessibility options received no improvements or changes.

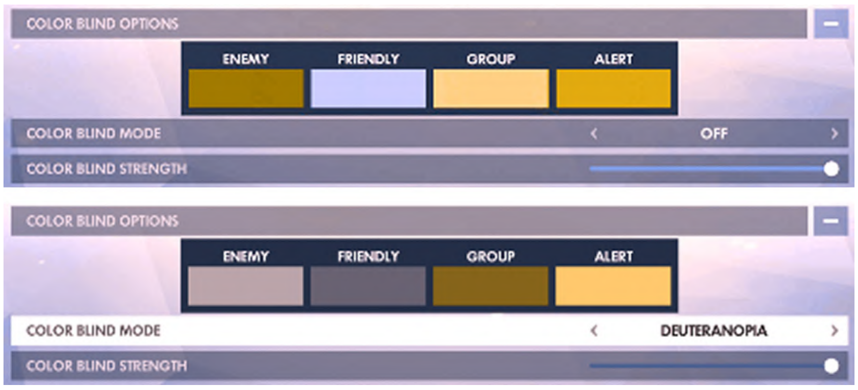
Figure 2.2 – Deuteranopia colorblind mode in Overwatch (2016).



COLOR BLIND MODE OFF

COLOR BLIND MODE DEUTERANOPIA

(a) Deuteranopia mode as seen by people with normal color vision.



COLOR BLIND MODE OFF

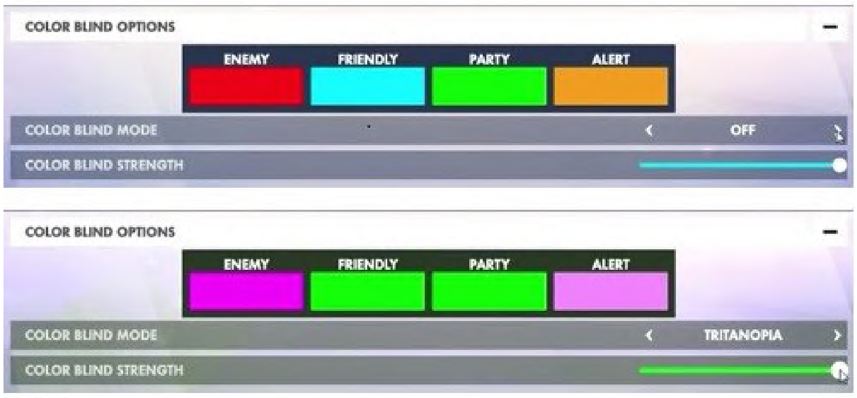
COLOR BLIND MODE DEUTERANOPIA

(b) Deuteranopia mode as seen by deuteranopes.

2.5 Summary

This chapter has reviewed the basics of other works that have addressed similar problems to ours, and that have served as inspiration or basis for the development of both the recoloring method we have chosen to work with, and our plugin itself.

Figure 2.3 – Tritanopia colorblind mode in Overwatch (2016). Source: (HARDIN, 2016).



COLOR BLIND OPTIONS

ENEMY FRIENDLY PARTY ALERT

COLOR BLIND MODE OFF

COLOR BLIND STRENGTH

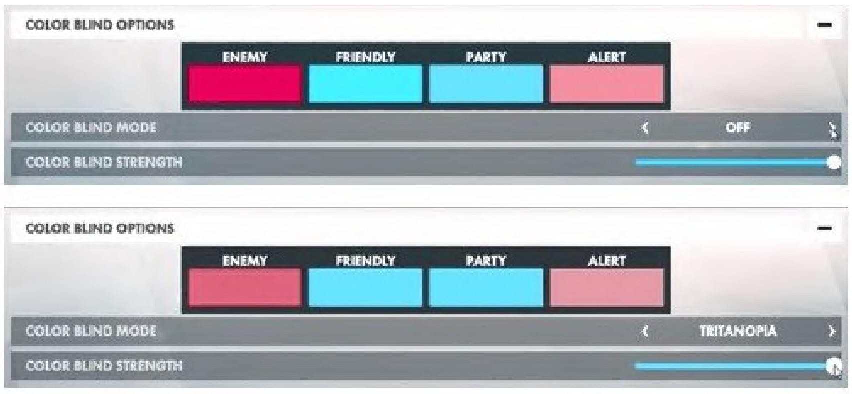
COLOR BLIND OPTIONS

ENEMY FRIENDLY PARTY ALERT

COLOR BLIND MODE TRITANOPIA

COLOR BLIND STRENGTH

(a) Tritanopia mode as seen by people with normal color vision.



COLOR BLIND OPTIONS

ENEMY FRIENDLY PARTY ALERT

COLOR BLIND MODE OFF

COLOR BLIND STRENGTH

COLOR BLIND OPTIONS

ENEMY FRIENDLY PARTY ALERT

COLOR BLIND MODE TRITANOPIA

COLOR BLIND STRENGTH

(b) Tritanopia mode as seen by tritanopes.

3 BACKGROUND

This chapter presents the details of the temporal-coherent color-contrast enhancement algorithm for dichromats by Machado and Oliveira (MACHADO; OLIVEIRA, 2010), which was used for the development of our plugin. Since the focus of our work is on how players with color vision deficiency glean color-based information from video games, which are more often than not time-sensitive in regards to the player's reaction, the contrast-enhancing and time-coherent aspects of this technique, applied in real-time, suit our needs adequately.

3.1 Real-Time Temporal-Coherent Color Contrast Enhancement for Dichromats

This technique uses the perceptually uniform CIE $L^*a^*b^*$ color space to recover most of the color contrast loss experienced by dichromats. To do this, they must find the direction v_{ab} in the a^*b^* chromaticity plane that maximizes this contrast loss, in a least squares sense, and then project the original colors onto the plane aligned with v_{ab}. At the same time, they must check and correct for any sudden changes in the direction of v_{ab} to ensure color consistency in interactive environments and multiframe sequences.

3.1.1 Maximum contrast loss in the CIE *L*a*b** color space

According to Brettel et al. (BRETTEL; VIéNOT; MOLLON, 1997), the range of colors that a dichromat can see can be represented by two half-planes in the LMS color space, each of them anchored to one point representing a hue that stays invariant for the given dichromacy type. These two half-planes can then be satisfactorily approximated by a single plane (VIéNOT; BRETTEL; MOLLON, 1999).

The planes that approximate the color gamuts perceived by each type of dichromat can then be mapped to the CIE $L^*a^*b^*$ color space. In this space, according to Kuhn et al. (KUHN; FERNANDES; OLIVEIRA, 2008), the angles between the protanope, deuteranope and tritanope gamuts and the L^*b^* plane are $\theta_p = -11.48°$, $\theta_d = -8.11°$ and $\theta_t = 46.37°$, respectively. Figure 3.1 illustrates these gamuts, along with the color ranges they represent.

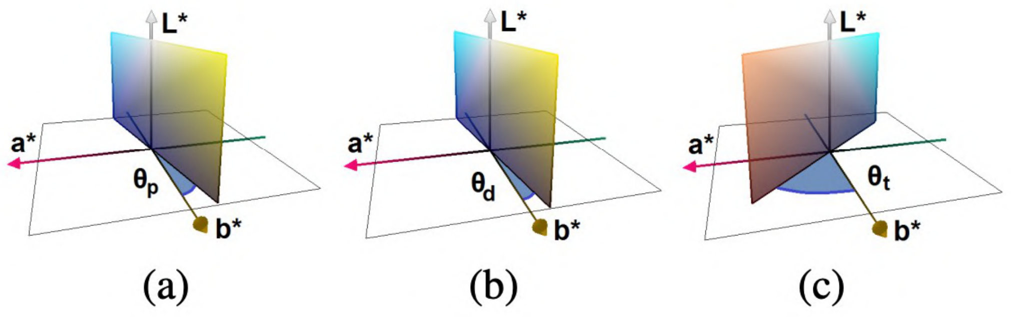


Figure 3.1 – Planar approximation for the color gamut of dichromats in the CIE L*a*b* color space. (a) Protanope. (b) Deuteranope. (c) Tritanope. Source: (MACHADO; OLIVEIRA, 2010).

3.1.2 Principal Component Analysis

Finding the direction v_{ab} of maximum color contrast loss experienced by a dichromat when observing a given image, in the least squares sense, can be a computationally expensive process. It would require computing, for each pixel p_i in the image, the contrast loss between p_i and every other pixel p_j in a neighborhood of size N_i around p_i. However, they can exploit spatial coherence, and the observation that neighboring pixels tend to have similar colors between each other, to simplify this step. To do this, they use the Gaussian pairing technique described by Grundland and Dodgson (GRUNDLAND; DODGSON, 2007). Thus, they pick a single neighbor p_j for each pixel p_i in the image, and use the contrast loss between this pair as the estimate for the contribution of p_i to the loss experienced by the dichromat. The distances from p_i to p_j are randomly defined by a univariate Gaussian distribution, with zero mean and variance $(2/\pi)\sigma^2$, where $\sigma^2 = \sqrt{2min(width, height)}$. Here, $width$ and $height$ are the image dimensions. For better performance with animated or video sequences, all neighbor coordinates are precomputed and stored in a separate texture, and the same (p_i, p_j) pairs are used for the entire sequence.

To obtain the estimated value of the relative contrast loss for a pair of pixels (p_i, p_j), they compare the contrast between the two colors perceived by an individual with normal color vision, with the contrast perceived by the dichromat, which can be done by projecting p_i and p_j onto the dichromat's color gamut. Because the CIE $L^*a^*b^*$ color space is approximately perceptually uniform, they can calculate the estimated loss as

$$l_{(p_i,p_j)} = \frac{||p_i - p_j|| - ||p'_i - p'_j||}{||p_i - p_j||} \tag{3.1}$$

where p'_i and p'_j are the projected values of p_i and p_j, and $||v||$ represents the vector length operation for a given vector v. This contrast loss happens in the direction of the 3D vector $v_{ij} = p_i - p_j$, but, because they must preserve the L^* luminance coordinates of the original colors to avoid reversing the polarities between each pair of colors (KUHN; FERNANDES; OLIVEIRA, 2008), they can discard the L^* coordinate from each v_{ij} and work with the chromaticity plane a^*b^*. This allows us to find the direction that maximizes the loss of local contrast by computing the eigenvectors of a 2×2 matrix instead of a 3×3 one.

Then, they let $w_i = l_{(p_i,p_j)} v_{ij}$ be the contrast loss associated to pixel p_i, and define M as a matrix of dimensions $n \times 2$, where n is the number of pixels in the image, whose rows contain the coordinates of the w_i vectors associated with each pixel.

$$M = \begin{bmatrix} w_1^{a^*} & w_1^{b^*} \\ w_2^{a^*} & w_2^{b^*} \\ \dots & \dots \\ w_n^{a^*} & w_n^{b^*} \end{bmatrix} \tag{3.2}$$

The v_{ab} vector representing the direction that maximizes contrast loss can then be computed as the eigenvector of the $M^T M$ matrix, for the corresponding eigenvalue with largest absolute value. For this step, the value of the b^* coordinate of v_{ab} is arbitrarily set to 1, and the characteristic equation solved for a^*. This is a simple, efficient way to avoid obtaining the trivial solution $v_{ab} = 0$ as a result.

3.1.3 Computing results

To obtain the resulting colors, they first project the original colors onto the plane defined by the L^* vector and v_{ab}, and then rotate these projected colors around the L^* axis to align with the dichromat's color gamut. Figure 3.2 illustrates this process done for protanopes, for two colors c_1 and c_3, whose neighbors are, respectively, c_2 and c_4.

Here, colors c'_1 to c'_4 are the projections of c_1 to c_4, respectively, on the dichromat's color gamut, and represent how the dichromat perceives c_1 to c_4. Colors c''_1 to c''_4 are the projection of the original colors onto the plane aligned with the direction of maximum

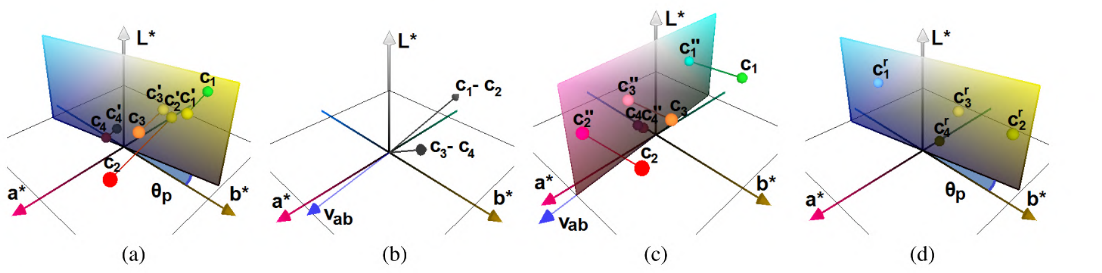


Figure 3.2 – Steps of the recoloring algorithm. Source: (MACHADO; OLIVEIRA, 2010).

contrast loss, and c_1^r to c_4^r are the resulting colors obtained after rotating the projection to align with the dichromat's gamut.

The colors obtained through this process tend to be more spread out than the the projection of the original colors onto the dichromat's gamut, and the final image obtained after converting the colors back to RGB from the CIE $L^*a^*b^*$ space will use only colors that the dichromat can see, and have improved color contrast for the dichromat if compared with the original.

It is possible to further exaggerate the contrast in the final image by rescaling all chromaticity coordinates to have a maximum value of 148, as the maximum length of chromaticity vectors in the CIE $L^*a^*b^*$ color space is 148.47. The authors, however, point out that this is not the intended use for this technique, due to the higher perceptual distortions caused by this exaggeration.

3.1.4 Ensuring temporal coherence

In this technique, if v_{ab} is close to being aligned with the a^* axis, as is the case in Figure 3.2, then minor changes in the original colors of the image might shift the b^* coordinate of v_{ab} from positive to negative, or vice versa. However, as described in Section 3.1.2, they set the value of b^* to 1 when solving the characteristic equation associated with v_{ab}, and have the value of a^* adjust its sign to accommodate any changes in v_{ab}'s direction. This means that b^* will always be positive, and any changes in the input colors that would shift it to a negative value instead cause the v_{ab} vector to reverse directions entirely (*i.e.*, for protanopes and deuteranopes, blues would become yellows, and vice versa).

This is a problem because, for interactive applications, animated sequences, and videos, it is important to keep the recoloring consistent and avoid abrupt changes in the colors of the objects between one frame and another. As a simple but effective solution to avoid the occurrence of such artifacts, they can store the value of v_{ab}, and compare it to the new computed v_{ab} in the next frame. If the angle between the previous and current vectors approaches 180°, they invert the current values of v_{ab} to enforce temporal coherence in the recoloring process.

3.2 Summary

This chapter has detailed the temporal-coherent color-contrast enhancing algorithm we used for the implementation of our plugin, and the theoretical concepts behind its development.

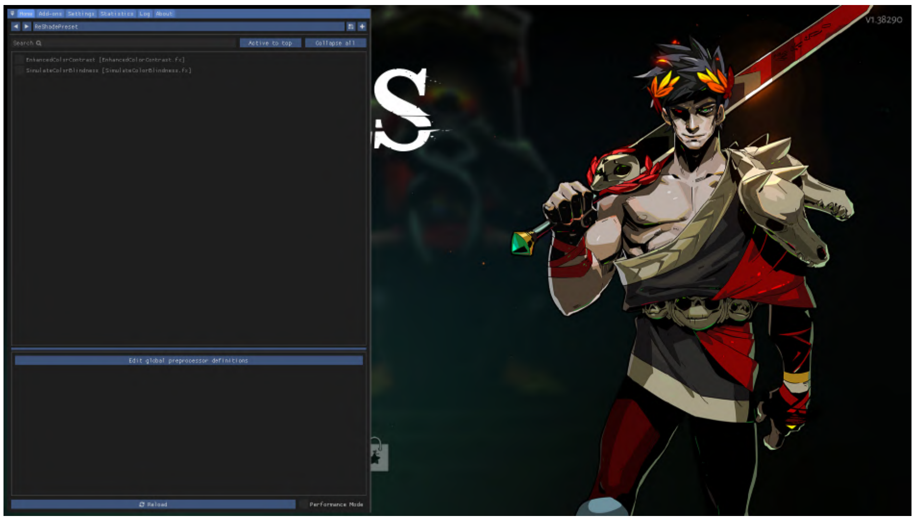
4 IMPLEMENTATION

In this chapter, we describe the tools and environment used to apply the algorithm reviewed in Chapter 3 to electronic games running on Windows platforms, and detail the development of the resulting plugin.

4.1 ReShade

To implement the chosen recoloring technique, this work uses the open-source C++ post-processing injector ReShade (CROSIRE, 2015). ReShade was designed to be run with games and videos, but it is popularly used in many video game communities to apply different filters to a game's screen, commonly for purely aesthetic reasons. ReShade enables the development of said filters by exposing the color and depth information output from the game itself, after the game's own rendering process is done, and applying ReShade's shaders to the frame before displaying the resulting image on the user's screen.

To use ReShade with any given game, the user first has to run the installer and select the desired game's executable, from which ReShade attempts to identify the graphics API the game runs on. In case it is unable to correctly perform this identification, the user has to select the API the application should be installed for. After this, ReShade will start up automatically along with the game's executable.



EnhancedColorContrast [EnhancedColorContrast.fx]
SimulateColorBlindness [SimulateColorBlindness.fx]

Figure 4.1 – ReShade's user interface overlay in Supergiant's Hades (2018), with no plugins enabled.

Figures 4.1 and 4.2 show ReShade's game overlay from where the user can activate and deactivate the available filters. The bottom panel of the overlay displays the enabled filters' settings to the user (Figure 4.3). A short video showing how this interface is used is available by clicking *here*.

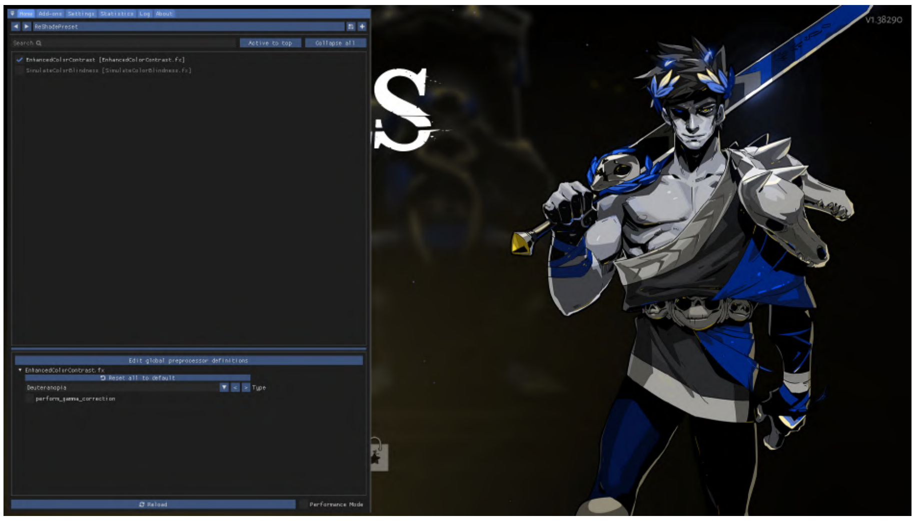


Figure 4.2 – ReShade's user interface overlay in Supergiant's Hades (2018), with our plugin enabled.

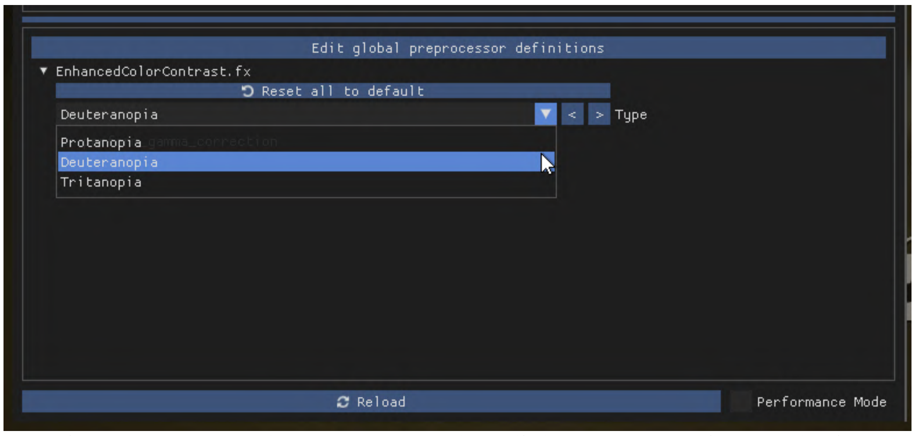


Figure 4.3 – Dropdown menu to select dichromacy type in our plugin.

ReShade was designed to work with computer games, specifically those running on Windows systems; as of the writing of this work, ReShade supports Windows 7 SP1, 8.1, 10 and 11, and requires .NET Framework 4.6.2 or higher to run.

4.2 ReShade FX

In order to support its application on games who run on a greater range of rendering APIs (namely, DirectX, OpenGL and Vulkan), ReShade provides its own shading language, ReShade FX, and compiles its shaders according to the API used by the game, every time the game application is opened. ReShade FX allows for the creation of multiple *techniques*, which, in the context of the language, are a series of user-defined vertex shaders, pixel shaders and/or compute shaders. Every pass, each technique and shader is applied to the processed frame in the order specified in the shader code.

One of the biggest advantages that ReShade offers is the ability to output each shader's result to up to eight different textures at the same time, provided all chosen textures have the same dimensions, and to sample from any declared texture from within any shader without aditional steps. The language also supports most of the HLSL intrinsic functions, plus a few additional methods to access and store values in the textures the shader is working with, and to perform interlocked operations (which in ReShade FX are called atomic operations). When using these interlocked operations, other threads are not allowed to access the variables that are in use (*i.e.* the operation is indivisible).

ReShade filters can expose variables to the user of the shader through the use of uniform variables, which, unless explicitly set to hidden, will be displayed in ReShade's user interface (UI), where their values can be manually changed. Alternatively, because to the shader code itself uniform variables work as constant variables, the only way to change a uniform variable's value as the shader is being executed is through the UI.

Uniform variables allow the use of annotations that can be used to customize their UI appearances, such as *ui_label*, that defines a display name to replace the variable name in the UI, and *hidden*, a flag that lets the variable be hidden from the interface. Additionally, the *source* annotation is used to request special runtime values such as cursor position and movement information, the system date and random values.

4.3 Recoloring Plugin

The ReShade shader implemented uses the UI to provide the user with a drop-down menu from where to select the type of dichromacy the recoloring should be done for. This determines the angle used for most of the rotation matrices applied later. For the technique itself, this work divides each step in the algorithm implemented in different shaders, executed in sequence, which are described in Subsections 4.3.1 to 4.3.5.

4.3.1 RGB to CIE *L*a*b** convertion

First we use a pixel shader to convert the RGB values from the back buffer, from where the frame output by the game's shaders is retrieved, to the CIE $L^*a^*b^*$ color space, and store the resulting colors in a new texture, called *texLab*, that will be kept unchanged as these values will be used for future calculations.

4.3.2 Generating random noise

Second, we randomly generate the coordinates for one neighbor for each pixel in the frame using a Gaussian probability distribution, and store these coordinates in another new texture, *texRandomNoise*. This was originally done in a compute shader with n threads (where n is the number of pixels in the frame), however, since the texture access methods that a compute shader can use are very costly, this decreased the performance of the game significantly and caused the screen to display ripple effects whenever a scene moved. We then changed this process to a pixel shader instead, successfully avoiding this distortion.

With the resources offered by ReShade FX, the only method to retrieve a random number is through the *random* annotation in an uniform variable, which returns an uniformly distributed integer in a chosen range. Therefore, to approximate the desired distribution as best as possible, we instead use the Box-Muller transform (BOX; MULLER, 1958) for mean $\mu = 0$ and standard deviation $\sigma = \sqrt{\frac{2}{\pi} scale}$, as these are the values specified by Grundland and Dodgson (GRUNDLAND; DODGSON, 2007). Here, $scale$ is the spatial scale based on the frame's dimensions that determines the neighborhood size (Equation 4.1). This value is constant for all pixels as it is dependent only on the frame dimensions.

$$scale = \sqrt{2 * \frac{min(buffer\_width, buffer\_height)}{\pi}} \tag{4.1}$$

Let x_i and y_i be the coordinates of pixel p_i, and r_1 and r_2 two pseudorandom values that will be used as seeds. ReShade offers a *timer* floating point uniform variable that returns the time in milliseconds since the game was started. We retrieve this value twice and use their fractional parts as the values of r_1 and r_2 as a better alternative to the *random* uniform variable, which would have the same value for every pixel. Equations 4.2 to 4.4 represent our use of the Box-Muller transform to generate x'_i and y'_i as the Gaussian noise coordinates for the given pixel p_i.

$$mag = \sigma\sqrt{-2 \cdot \log(r_1)} \tag{4.2}$$

$$x'_i = x_i + mag \cdot \cos(2\pi r_2) \tag{4.3}$$

$$y'_i = y_i + mag \cdot \sin(2\pi r_2) \tag{4.4}$$

The resulting noise coordinates that will be stored are clamped between 0 and the dimensions of the frame, then normalized. It is important to note that, unlike in the original algorithm (MACHADO; OLIVEIRA, 2010), our noise texture is generated again with every pass, as there is no way to restrict this at technique level through ReShade.

4.3.3 Principal Component Analysis

For the Principal Component Analysis step of this algorithm, using a pixel shader and the textures with the CIE $L^*a^*b^*$ color values and the random neighbor coordinates, *texLab* and *texRandomNoise* we project each pixel and its neighbor onto the dichromat's color gamut. The gamut itself is computed using the b^* vector, and a rotation matrix calculated using the appropriate angle according to the user's selection between protanopia, deuteranopia and tritanopia. Thus, the rotated b^* vector and the unaltered L^* vector define the gamut plane. From this, the projection is obtained through the operation

$$p'_i = p_i - (p_i \cdot b^{*\prime})b^{*\prime} \tag{4.5}$$

where p_i is the pixel being projected and $b^{*\prime}$ is the rotated $b*$ vector. The $\cdot$ operator describes the dot product operation between two vectors. Figure 4.1 exemplifies this process.

Then, the contrast loss experienced by the dichromat is calculated by comparing the distance between each pixel and its neighbor in their original CIE $L^*a^*b^*$ values, and the distance between those pixels after being projected to the gamut. If we consider p and n as the original values of the pixel and its neighbor, and p' and n' their values after being projected, then

$$Loss_i = 1 - \frac{||p'_i - n'_i||}{||p_i - n_i||} \tag{4.6}$$

defines the relative contrast loss observed for that pair of pixels, which happens in the direction of $\theta_i = p_i - n_i$.

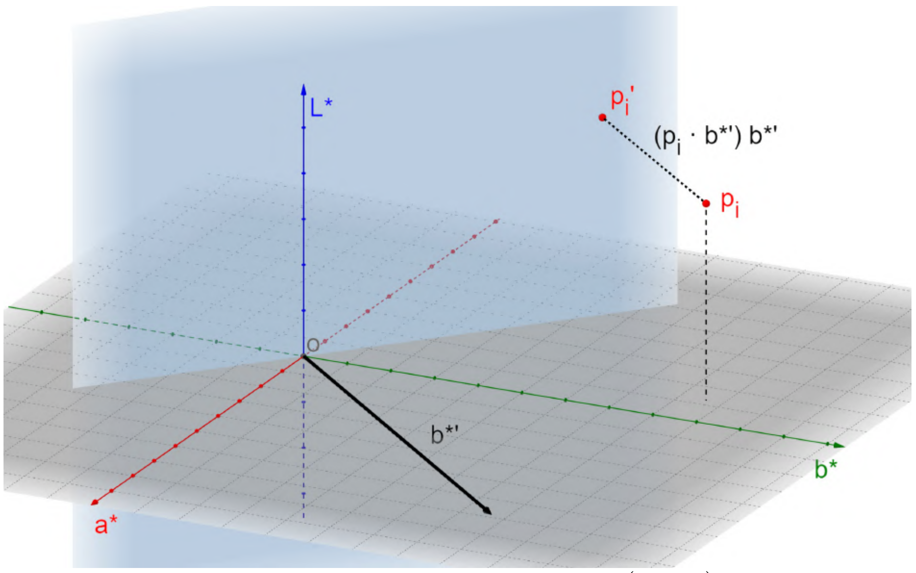


Figure 4.4 – Projection of p_i onto a plane defined by point $o = (0, 0, 0)$ and the normal vector $b*'$.

To simplify the next steps, instead of storing $Loss_i \cdot \theta_i$ in the output *texPCA* texture, we discard the L^* coordinates of the contrast loss here, and instead store the values of a_i^{*2}, $a_i^* b_i^*$, and b_i^{*2} for each pixel p_i.

4.3.4 Computing final colors

As described, we can obtain the direction of maximum contrast loss experienced by the dichromat through a matrix M whose rows store the coordinates a^* and b^* of each vector calculated in the PCA step. This involves finding the eigenvalue with the largest absolute value for the 2×2 matrix $M^T M$, whose format will be, for n as the number of pixels in the frame:

$$\begin{bmatrix} \sum_i^n a_i^{*2} & \sum_i^n a_i^* b_i^* \\ \sum_i^n a_i^* b_i^* & \sum_i^n b_i^{*2} \end{bmatrix}. \tag{4.7}$$

When declaring a texture in ReShade, we can define how many MIP levels it should have, and each mipmap will be automatically generated. For this work, we set this value to 12, the minimum number of MIP levels required to reduce a texture of dimensions 2160×3840, which is the maximum resolution supported by the vast majority of current computer games, to one single texel. Then, since the *texPCA* texture already contains the values of a^{*2}, $a^* b^*$, and b^{*2} for each pixel, we need only use the ReShade method *tex2Dlod* to retrieve the last MIP level, which will contain an average of all values, and multiply it by the number of pixels to obtain their sums.

We find the eigenvalues of this matrix through solving a standard quadratic equation $ax^2 + bx + c = 0$, for $a = 1$, $b = -(\sum_i^n a_i^{*2} + \sum_i^n b_i^{*2})$, and $c = \sum_i^n a_i^{*2} \cdot \sum_i^n b_i^{*2} - (\sum_i^n a_i^* b_i^*)^2$. From the eigenvalue λ with largest absolute value, the coordinates of the eigenvector that represents the direction on the a^*b^* plane in which maximum contrast loss is experience by the dichromat are

$$a^* = (\sum_i^n a_i^{*2})^2 - \lambda \tag{4.8}$$

and

$$b^* = -\sum_i^n a_i^* b_i^*. \tag{4.9}$$

These coordinates describe the $\theta_{ab} = (a^*, b^*)$ vector that, along with the L^* vector, define a plane in the direction of maximum contrast loss, onto which every color stored in the *texPCA* texture will be projected. The projection of these colors is done in the same manner described in Equation 4.6, but replacing $b^{*\prime}$ with $N_{L\theta_{ab}}$ as the new plane's normal:

$$N_{L\theta_{ab}} = \frac{\theta_{ab} \times L}{||\theta_{ab} \times L||} \tag{4.10}$$

and

$$p_i' = p_i - ||p_i \cdot N_{L\theta_{ab}}||N_{L\theta_{ab}} \tag{4.11}$$

where $\times$ described the cross product between two vectors. The projected values are output to a new texture called *texProjectedColors*. Next, in another pixel shader, we find the angle between the $L^*\theta_{ab}$ plane and the dichromat's gamut, and rotate the colors projected onto this maximum contrast loss plane to align with the gamut. We calculate this rotation angle through

$$\alpha = \arccos\left(\frac{\theta_{ab} \cdot b^*}{||\theta_{ab}|| \cdot ||b^*||}\right) \tag{4.12}$$

and then compare the cross product of θ_{ab} and b^*, which produces a vector parallel to L^*, with the L^* vector itself through a dot product. If this dot product is 0, meaning $\theta_{ab} \times b^*$ and L^* have opposite directions, we must flip the sign of the angle α found.

These projected and rotated values are output to a texture called *texResultColors* and, lastly, in a final pixel shader, these results are converted back from the CIE $L^*a^*b^*$ color space to their RGB values, and returned to the back buffer to be displayed to the user on the game's screen.

4.3.5 Temporal Coherence

To check and correct for abrupt changes in the direction of v_{ab}, we store its value in a 1×1 texture named *texPreviousDirection*. For each frame, we compare the current v_{ab} to the one stored in the previous frame. If the angle between them is higher than $175°$, we reverse the direction of the current v_{ab}. We use ReShade's $framecount$ uniform variable to check if we're rendering the first frame and skip this step for the first pass.

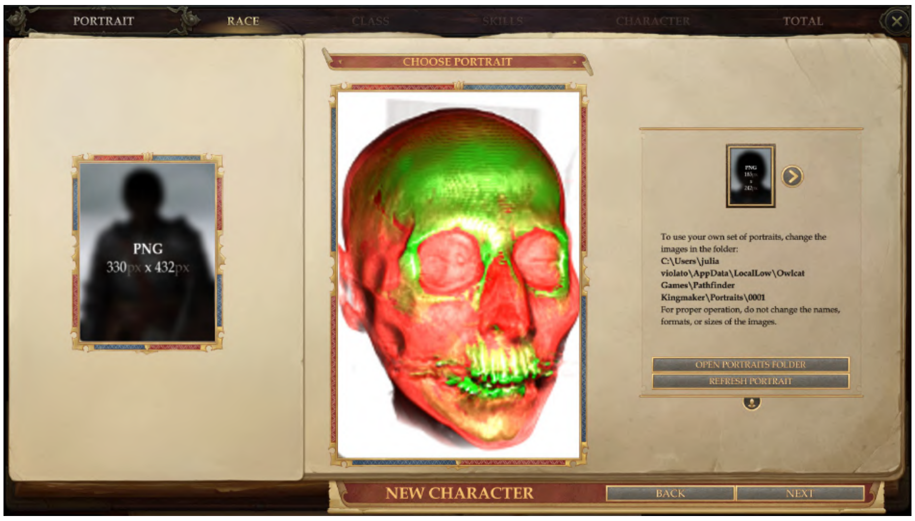
The original algorithm also takes steps to smooth the transition between angles even in cases when the colors are not being flipped. However, in video games we must account for the possibility of the player being able to leave one region for another, with a different color palette, which may cause wide angles between the contrast loss direction of one frame and the next. Therefore, our implementation cannot interfere with this.

4.4 Difficulties

One of the major difficulties of working with ReShade is the fact that the ReShade FX language is limited to shaders only. Thus, any step of the algorithm has to be adapted to be coded in a vertex, pixel or compute shader, and making efficient use of texture coordinates and thread work groups was crucial to avoid game performance loss.

Working exclusively with shaders also meant a lack of debug tools to validate each step of the algorithm as the plugin was being developed. Because of the lack of access to a custom library used, some operations were also difficult to validade with the original code.

Another minor difficulty we faced was the validation of the obtained results. Because ReShade is primarily designed to be used with games, in order to recolor the example images provided by the authors, it was necessary to find a game that allows the player to upload and display custom images, and then limit the plugin's area of operation to the image only, as to make sure the game's other UI elements didn't interfere with the recoloring. We managed to achieve this with the game Pathfinder: Kingmaker (2018), which allows the player to upload custom portraits to their game profile (Figure 4.5). Then, in the first shader, we manually set every pixel outside of the custom portrait's area to white



To use your own set of portraits, change the images in the folder:
C:\Users\julia violato\AppData\LocalLow\Owlcat Games\Pathfinder Kingmaker\Portraits\0001
For proper operation, do not change the names, formats, or sizes of the images.

Figure 4.5 – Custom portrait selection feature in Pathfinder: Kingmaker (2018) with example image from the original article.

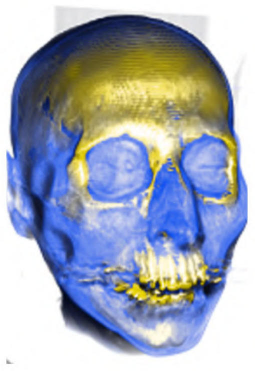
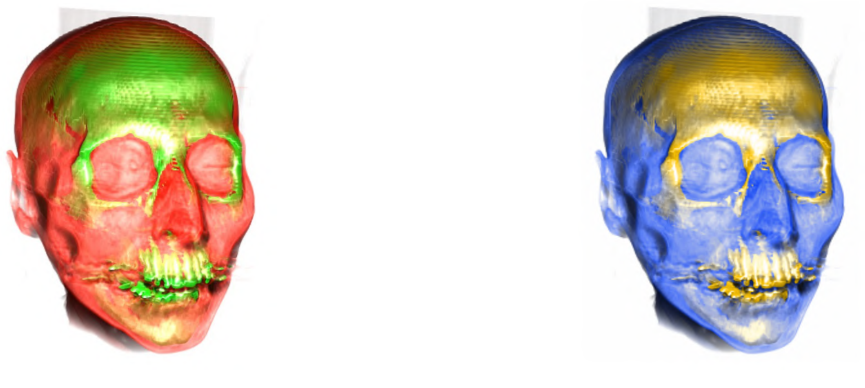


Figure 4.6 – Example image recolored with our plugin in Pathfinder: Kingmaker (2018). The area outside the custom portrait display is manually set to white by our shader before the recoloring process.



(a) (b)

Figure 4.7 – Medical visualization image of a skull (a) and its recoloring done with the original Real-Time Temporal-Coherent Color Contrast Enhancement for Dichromats technique (b). Source: (MACHADO; OLIVEIRA, 2010)

(Figure 4.6). We were then able to verify that our implementation produced correct results, by comparing these resulting images with the original examples provided by the authors of the algorithm (Figure 4.7).

4.5 Summary

In this chapter, we have discussed in detail the development of our plugin. This includes how we used the tools available through ReShade to implement the temporal-coherent color-contrast enhancing algorithm for games, as well as the steps taken to work around the limitations of ReShade to reach our goals.

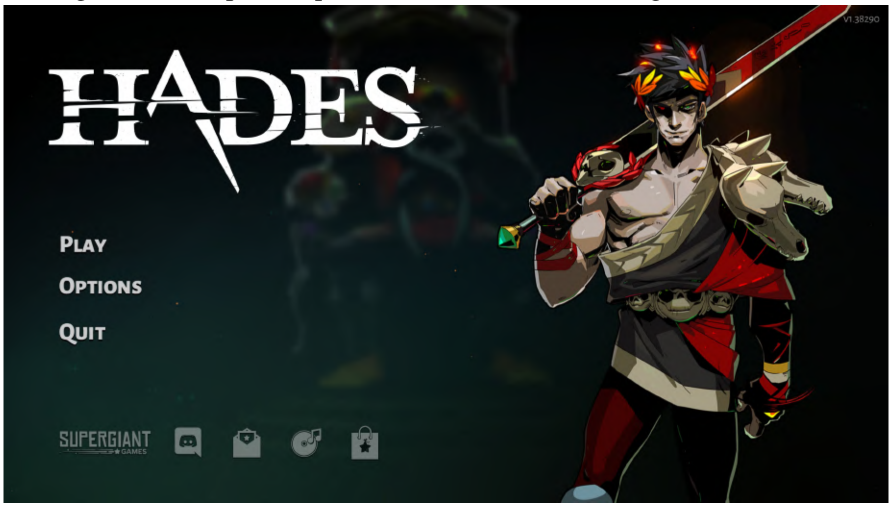
5 RESULTS

This chapter presents the results we obtained with our finished plugin across different games, by comparing the game as it is seen by a player with normal color vision and by a dichromat with and without the recoloring filter we developed. In cases where the game itself offers accessibility options for color blind players, the effects of these options are also presented for comparison. Like the examples provided previously in Section 1, to simulate color blind deficiency, we project the colors onto the dichromat's gamut plane in the CIE $L^*a^*b^*$ color space. We also present performance metrics for our plugin and its impact on game performance.

5.1 Recoloring

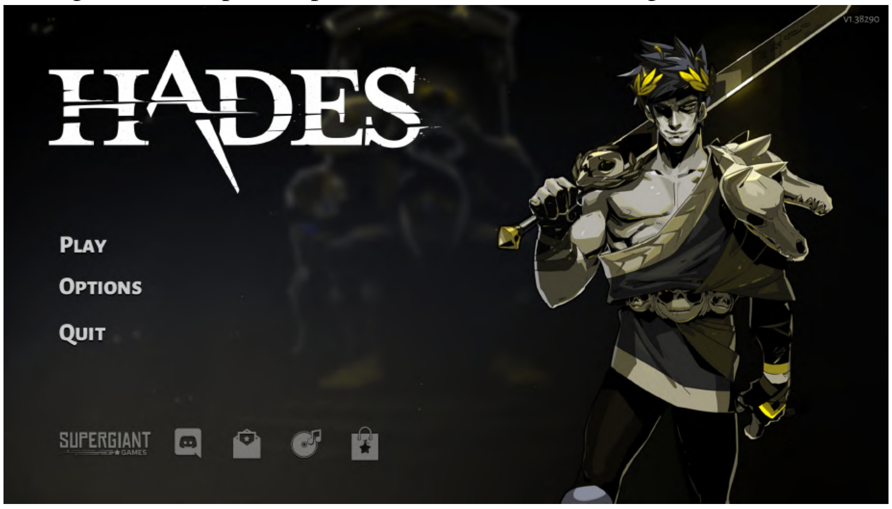
Figures 5.1 to 5.3 show the effects of our plugin on Supergiant's Hades (2020) for the three types of dichromacy. This is a game that makes heavy use of red and green shades, and so color contrast loss impacts protanopes and deuteranopes the most. In Figure 5.2 in particular, we can note how important UI elements, such as the character's health bar and the time remaining to complete the level, become much more distinguishable for the dichromat with the use of our plugin.

Figure 5.1 – Original colors, protanopia simulation, and recoloring of the menu in Hades (2020).

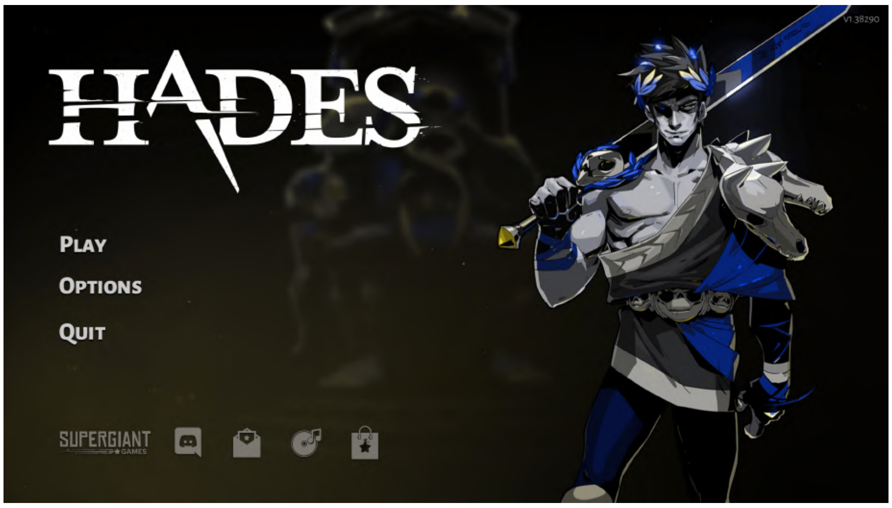


(a) Normal color vision.

Figure 5.1 – Original colors, protanopia simulation, and recoloring of the menu in Hades (2020).

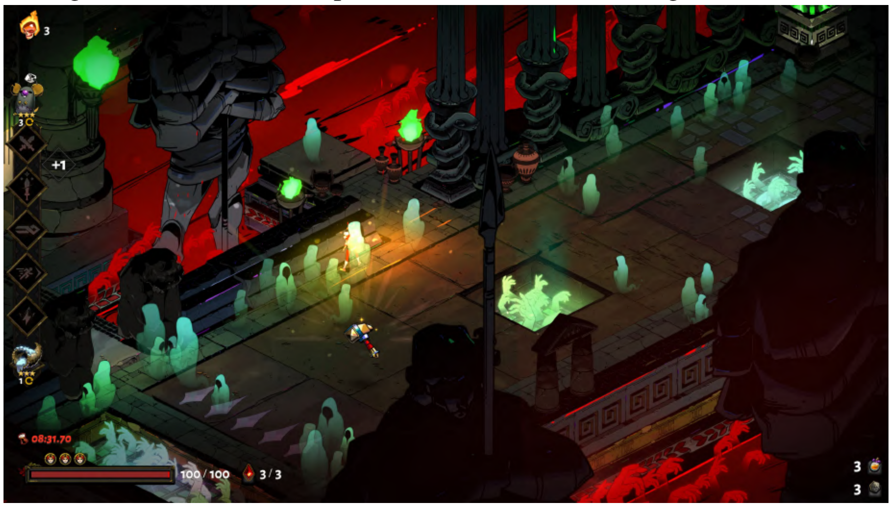


(b) Protanopia simulation.



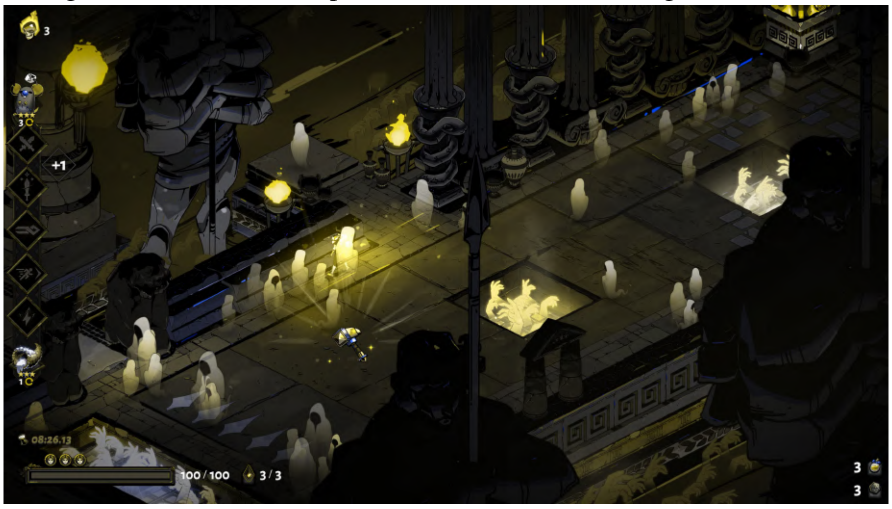
(c) Our recoloring for protanopia.

Figure 5.2 – Original colors, deuteranopia simulation, and recoloring of an arena in Hades (2020).

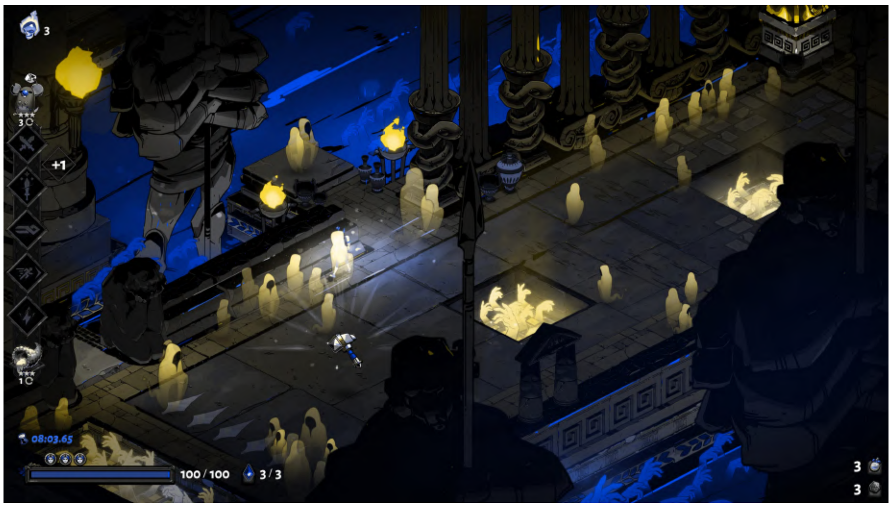


(a) Normal color vision.

Figure 5.2 – Original colors, deuteranopia simulation, and recoloring of an arena in Hades (2020).

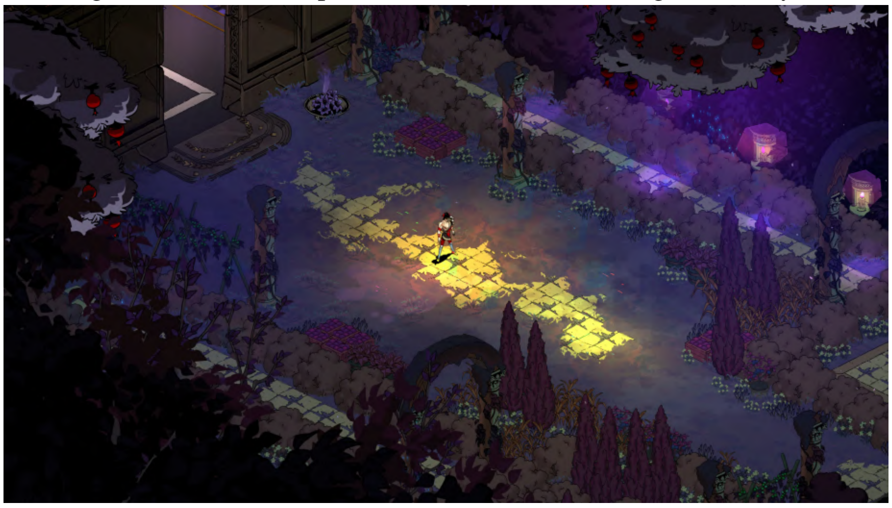


(b) Deuteranopia simulation.



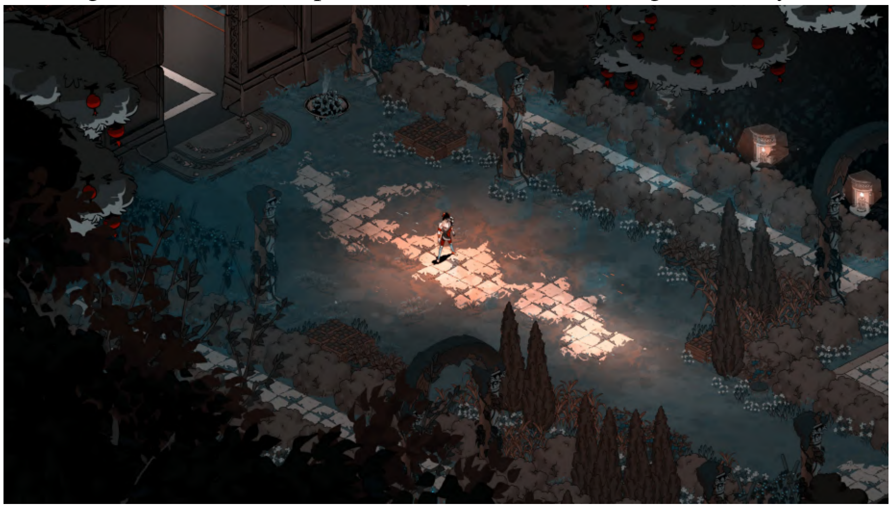
(c) Our recoloring for deuteranopia.

Figure 5.3 – Original colors, tritanopia simulation, and recoloring of scenery in Hades (2020).

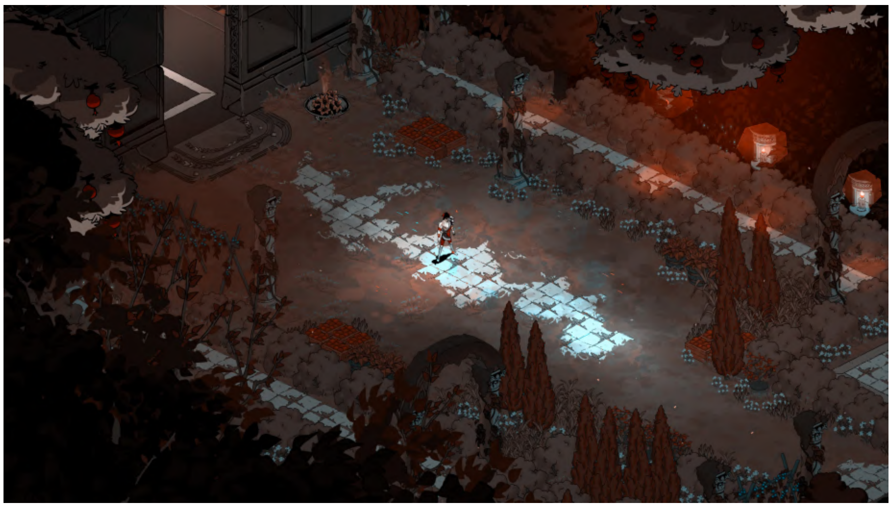


(a) Normal color vision.

Figure 5.3 – Original colors, tritanopia simulation, and recoloring of scenery in Hades (2020).



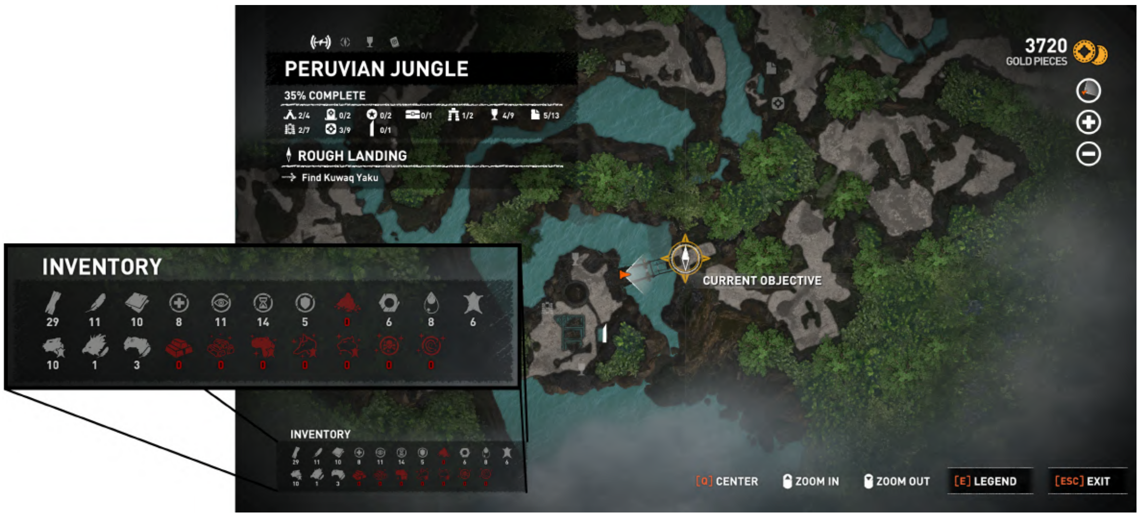
(b) Tritanopia simulation.



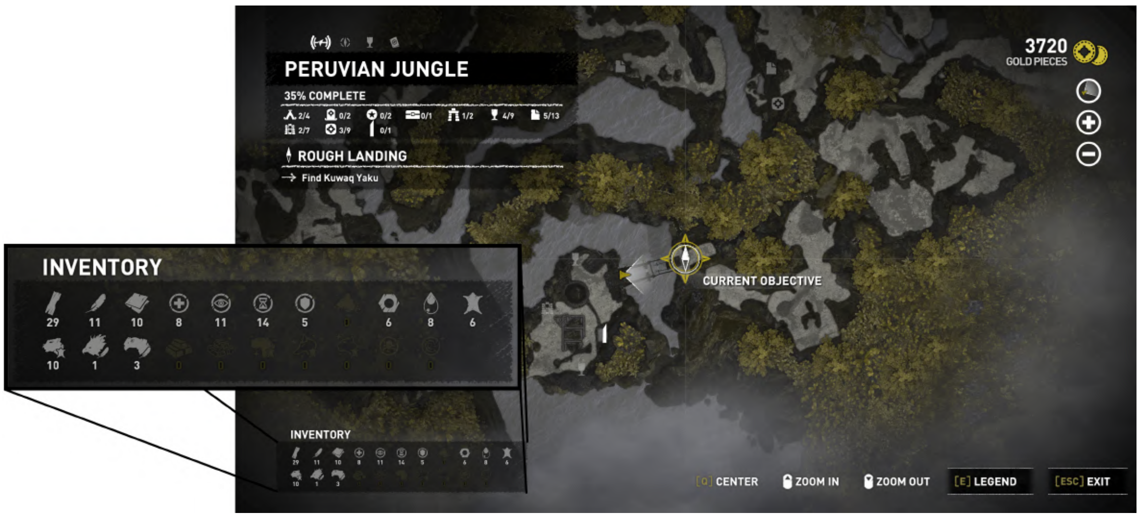
(c) Our recoloring for tritanopia.

In Figure 5.4, we see how our plugin enhances the contrast of inventory icons in Eidos-Montréal's Shadow of the Tomb Raider (2018), which represent important information that should be easily visible to the player, but in the process sacrifices part of the naturalness of the map by changing the color of the river to yellow, and the gold pieces to blue. This is a side effect of being restricted to the range of colors the dichromat can see, which is significantly smaller than the one perceived by a normal trichromat. This is an action-adventure game with many survival aspects, like hunting in wooded environments. For a player with CVD, identifying targets in this situation can be much more difficult, as their colors might make them blend with the background. Figure 5.5 shows that our recoloring can alleviate this.

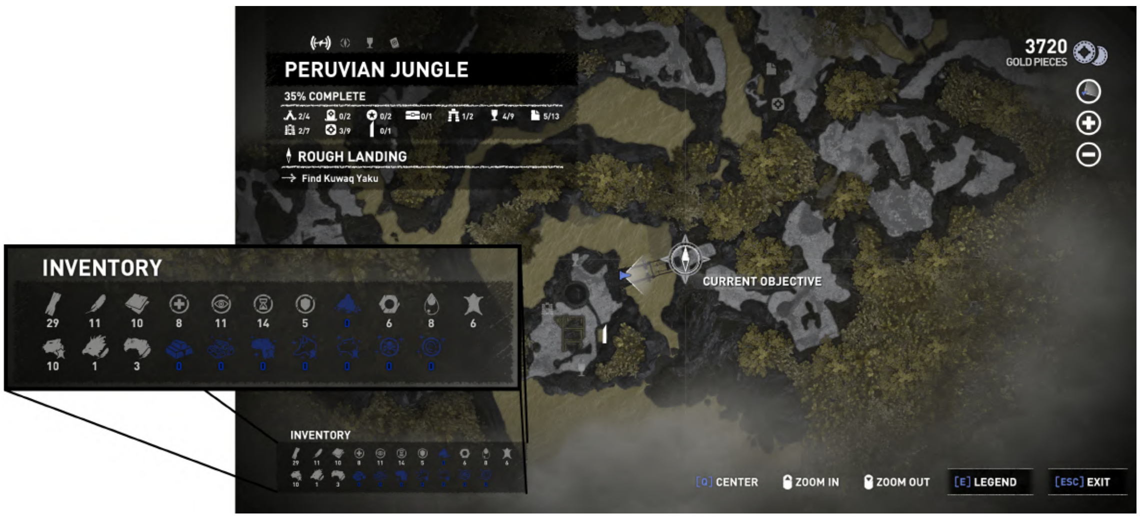
Figure 5.4 – Original colors, deuteranopia simulation, and recoloring of the map in Shadow of the Tomb Raider (2018).



(a) Normal color vision.

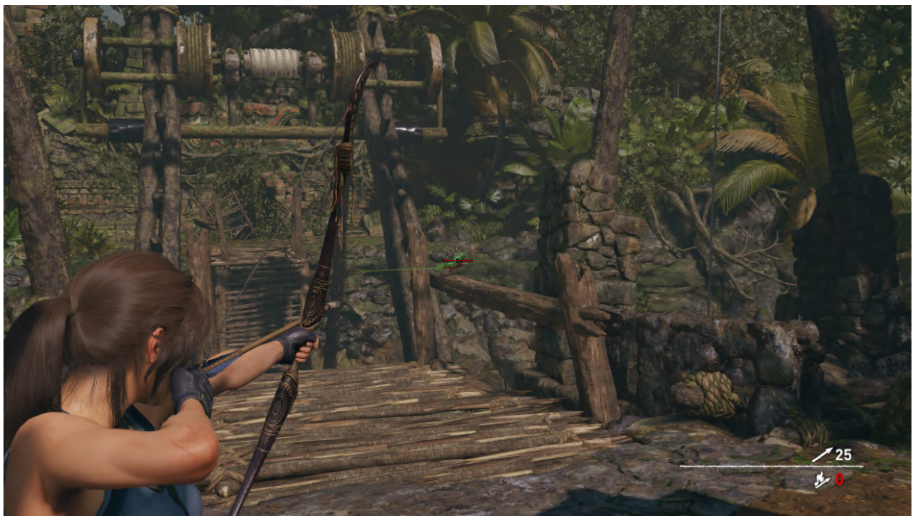


(b) Deuteranopia simulation.

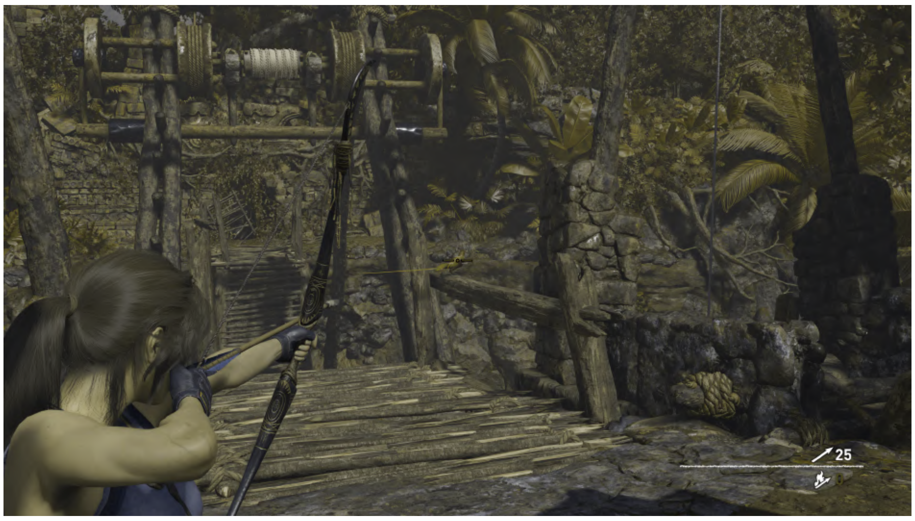


(c) Our recoloring for deuteranopia.

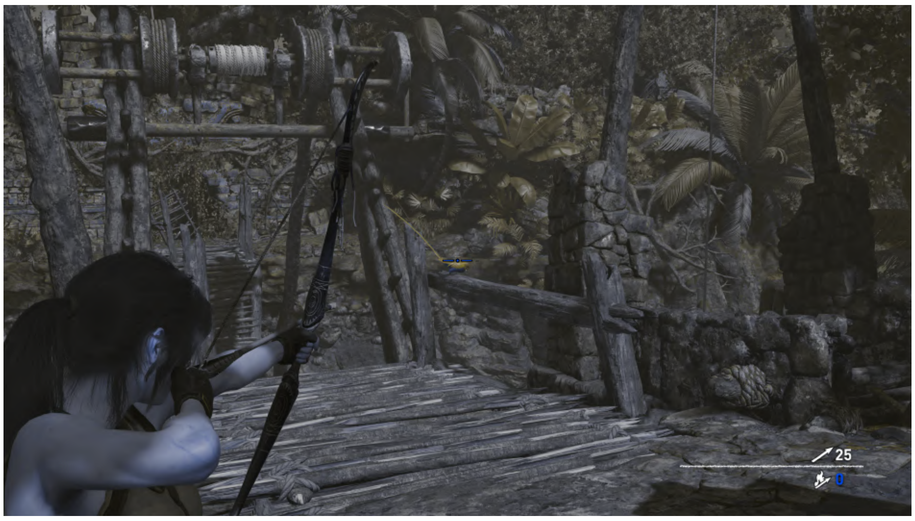
Figure 5.5 – Original colors, protanopia simulation, and recoloring of an example of the hunting mechanic in Shadow of the Tomb Raider (2018).



(a) Normal color vision.



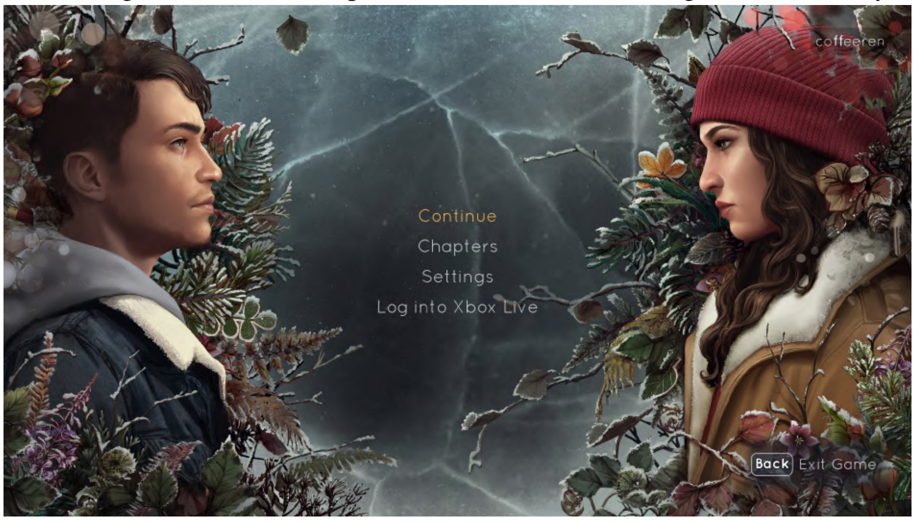
(b) Protanopia simulation.



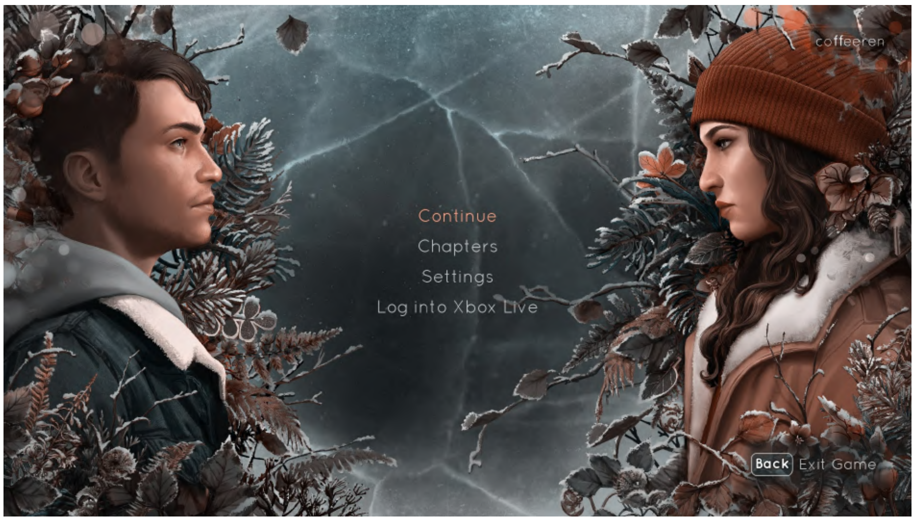
(c) Our recoloring for protanopia.

Similarly to the loss of naturalness seen in Figure 5.4, this process might sometimes shift colors in ways that are less than ideal. In Figure 5.6, for example, we see for Dontnod Entertainment's Tell Me Why (2020) that our plugin spreads the colors in a way that does recover contrast in the background image, but as a consequence makes the selected option in the menu much lighter, and harder to distinguish from the others. This is at the moment not avoidable without changes to the algorithm itself, as it happens in the recovery of contrast between the text and the background it is shown on. This is due to our usage of the Gaussian pairing technique to select a neighbor for each pixel, with a neighborhood of fixed size in relation to the game's resolution. Therefore, if any element, such as the other menu items, falls completely outside of a pixel's neighborhood, it will not be taken into consideration when computing that pixel's new color.

Figure 5.6 – Original colors, tritanopia simulation, and recoloring in Tell Me Why (2020).

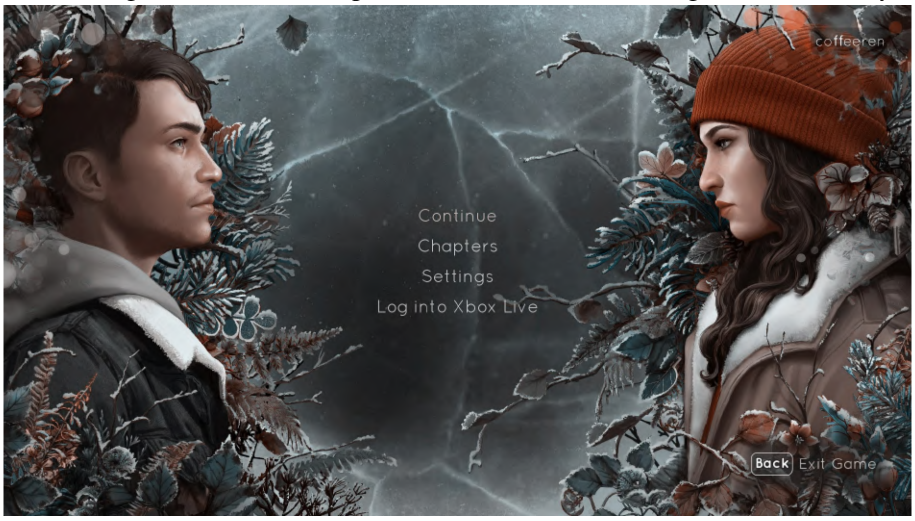


(a) Normal color vision.



(b) Tritanopia simulation.

Figure 5.6 – Original colors, tritanopia simulation, and recoloring in Tell Me Why (2020).

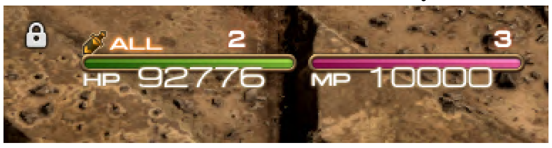


(c) Our recoloring for tritanopia.

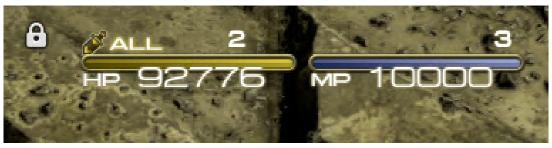
To maintain the contrast between two separate UI elements, we would require a way to separate them from the rest of the image during the recoloring process, either through the user's input, or with a method to identify in code which pixels belong to them. Without these changes, at its current state our solution might not be beneficial in situations where the user wishes to keep elements placed further apart on the screen easily distinguishable from one another.

When applying our plugin to Square Enix's Final Fantasy XIV: A Realm Reborn (2013), we can notice a significant improvement in the color contrast of essential UI elements (Figures 5.7 to 5.12), especially when compared to the native filtering for players with CVD that the game offers. Figures 5.7, 5.8 and 5.12 show the results of applying our plugin to the examples outlined in Chapter 1.

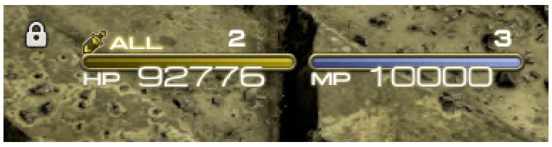
Figure 5.7 – Original colors, protanopia simulation, and recoloring of UI elements in Final Fantasy XIV: A Realm Reborn (2013).



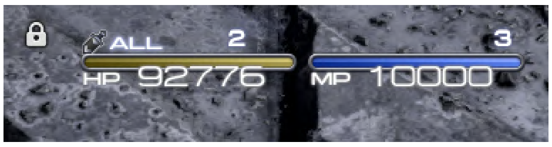
(a) Normal color vision.



(b) Protanopia simulation.

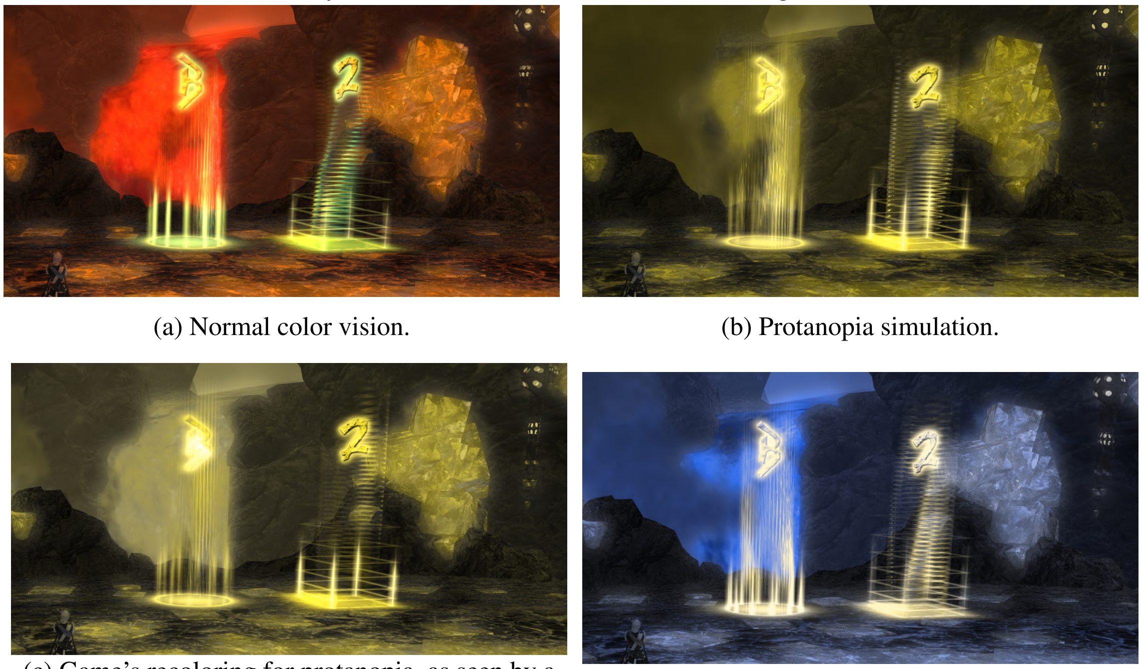


(c) Game's recoloring for protanopia, as seen by a protanope.



(d) Our recoloring for protanopia.

Figure 5.8 – Original colors, protanopia simulation, and recoloring of ground markers in a Final Fantasy XIV: A Realm Reborn (2013) boss fight arena.



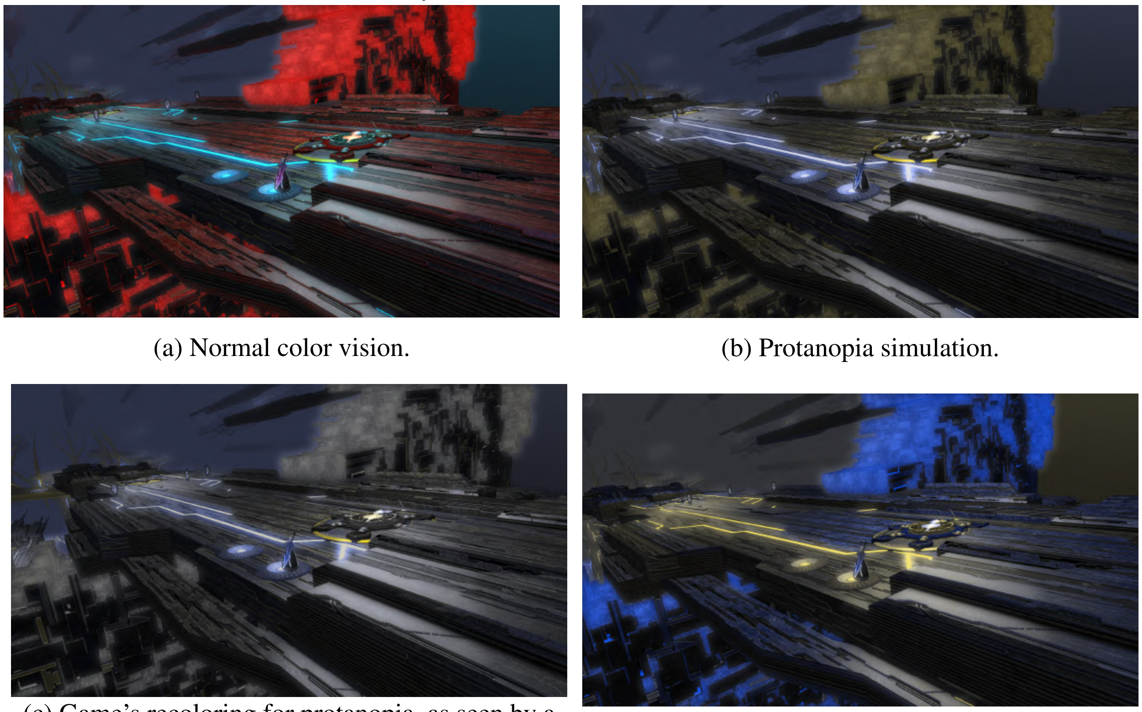
(a) Normal color vision.

(b) Protanopia simulation.

(c) Game's recoloring for protanopia, as seen by a protanope.

(d) Our recoloring for protanopia.

Figure 5.9 – Original colors, protanopia simulation, and recoloring of a boss fight arena in Final Fantasy XIV: A Realm Reborn (2013).



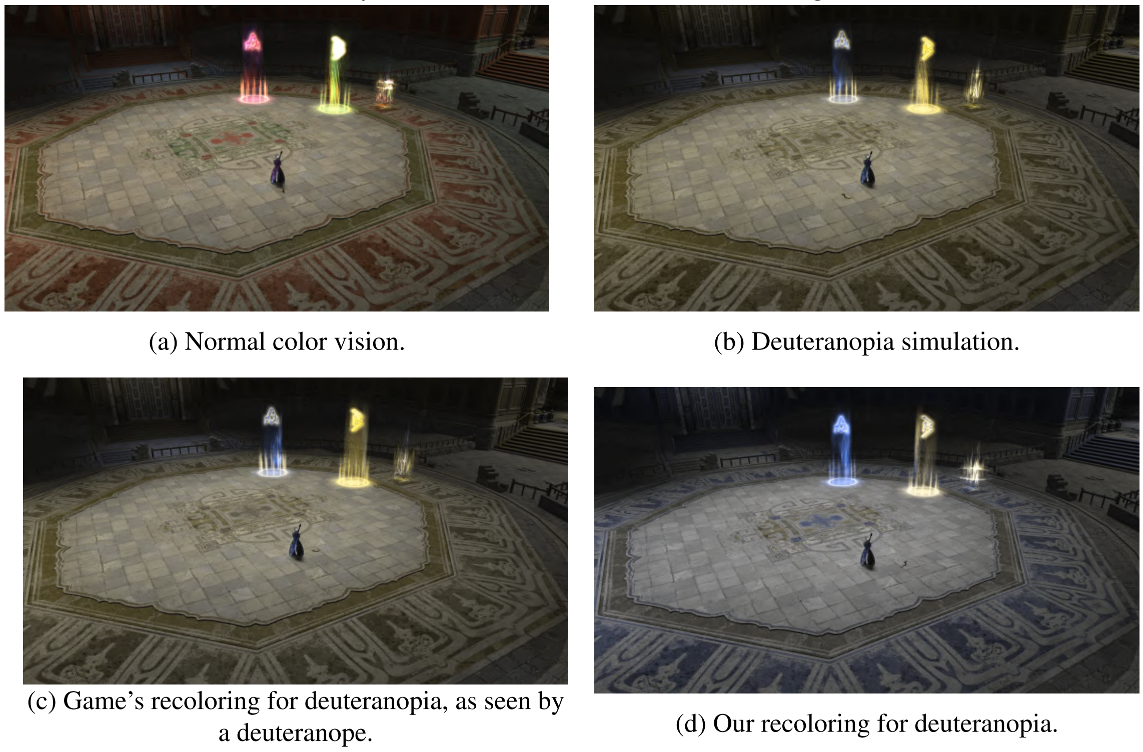
(a) Normal color vision.

(b) Protanopia simulation.

(c) Game's recoloring for protanopia, as seen by a protanope.

(d) Our recoloring for protanopia.

Figure 5.10 – Original colors, deuteranopia simulation, and recoloring of ground markers in a Final Fantasy XIV: A Realm Reborn (2013) boss fight arena.

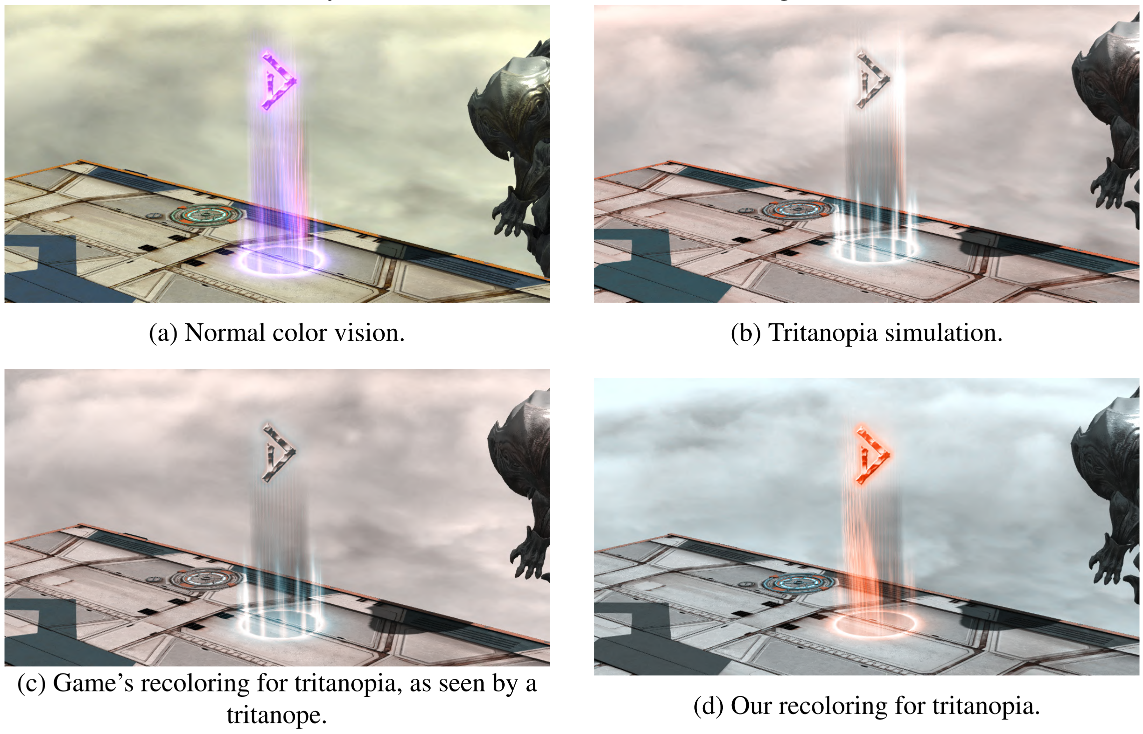

(a) Normal color vision.

(b) Deuteranopia simulation.

(c) Game’s recoloring for deuteranopia, as seen by a deuteranope.

(d) Our recoloring for deuteranopia.

Figure 5.11 – Original colors, tritanopia simulation, and recoloring of a ground marker in a Final Fantasy XIV: A Realm Reborn (2013) boss fight arena.

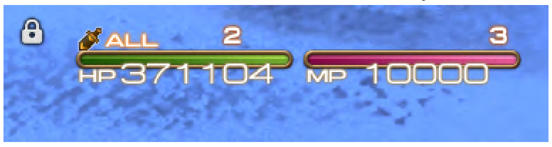

(a) Normal color vision.

(b) Tritanopia simulation.

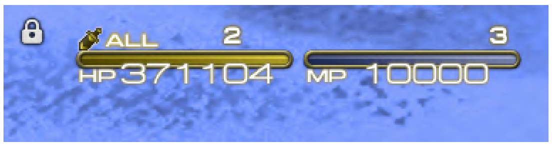
(c) Game’s recoloring for tritanopia, as seen by a tritanope.

(d) Our recoloring for tritanopia.

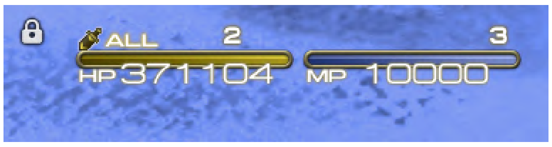
Figure 5.12 – Original colors, deuteranopia simulation, and recoloring of UI elements in Final Fantasy XIV: A Realm Reborn (2013).



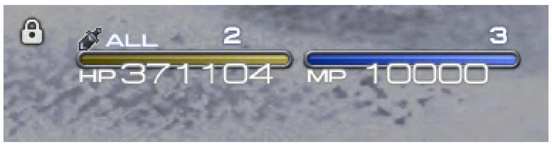
(a) Normal color vision.



(b) Deuteranopia simulation.



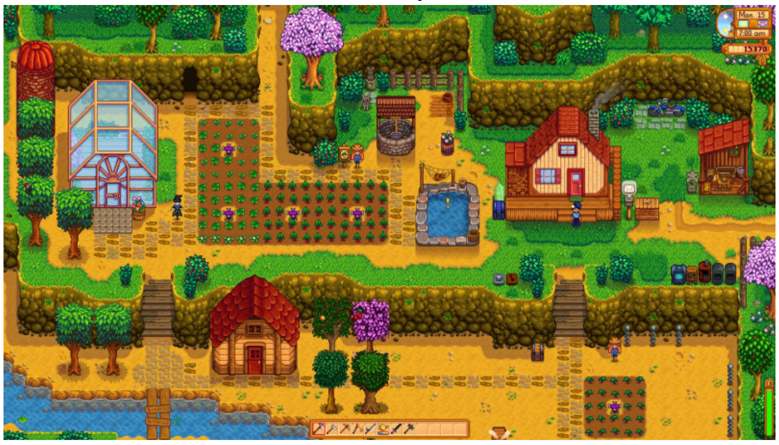
(c) Game's recoloring for deuteranopia, as seen by a deuteranope.



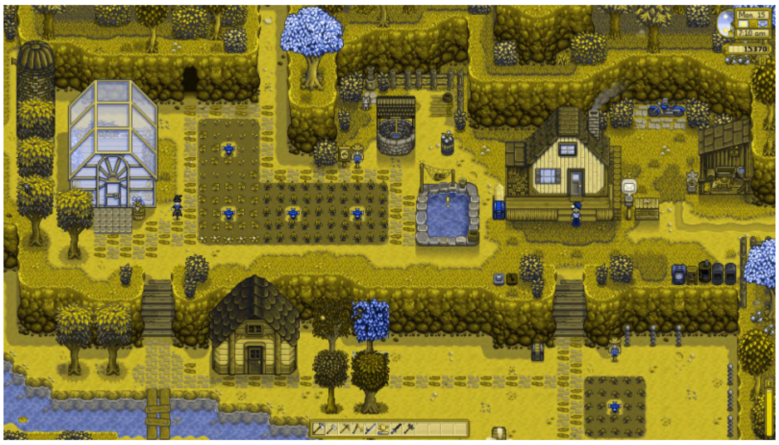
(d) Our recoloring for deuteranopia.

Figures 5.13 to 5.15 show examples of our recoloring that still recover much of the visual effects of the bright color choices for different sceneries in ConcernedApe's Stardew Valley (2016).

Figure 5.13 – Original colors, protanopia simulation, and recoloring of the player's home in Stardew Valley (2016).

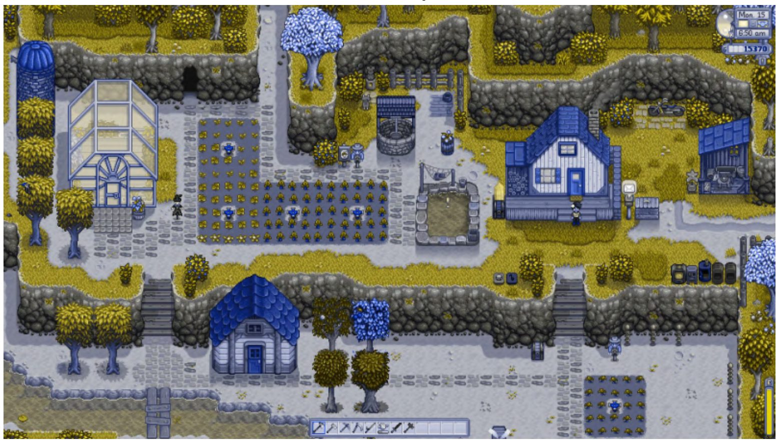


(a) Normal color vision.



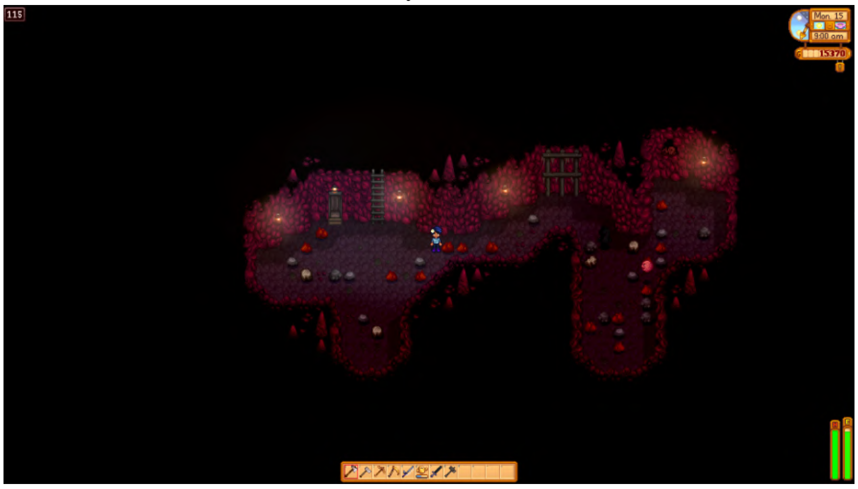
(b) Protanopia simulation.

Figure 5.13 – Original colors, protanopia simulation, and recoloring of the player’s home in Stardew Valley (2016).

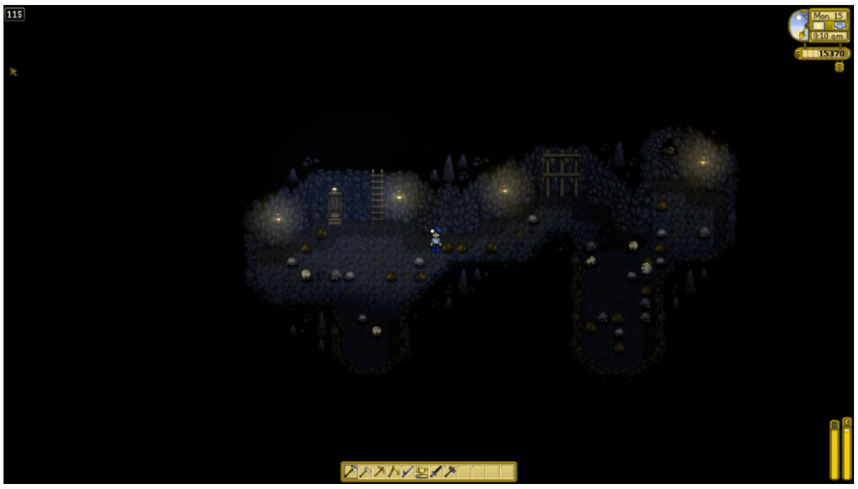


(c) Our recoloring for protanopia.

Figure 5.14 – Original colors, deuteranopia simulation, and recoloring of a cave in Stardew Valley (2016).

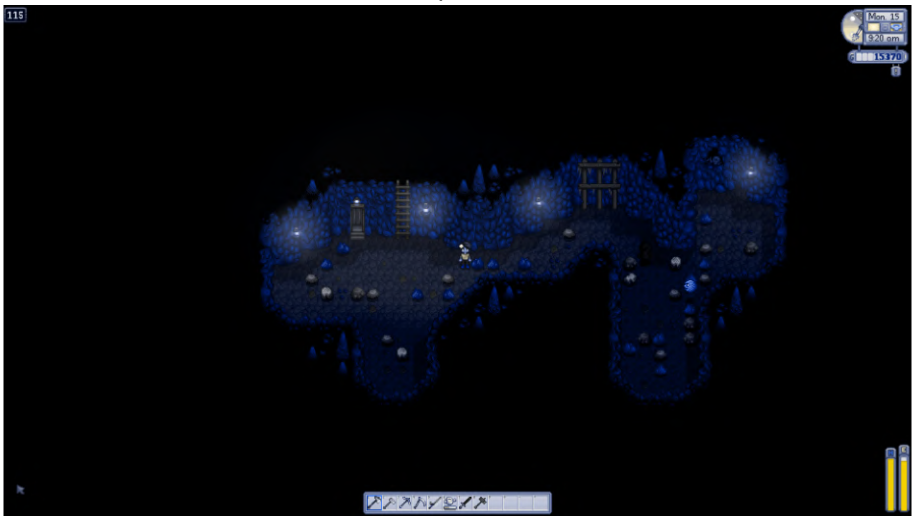


(a) Normal color vision.



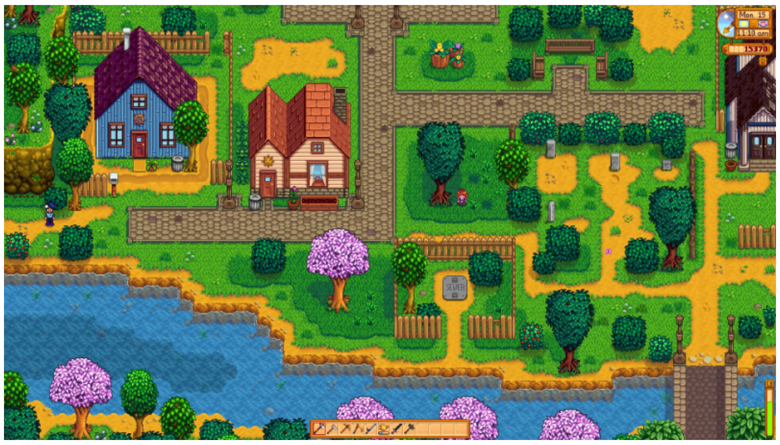
(b) Deuteranopia simulation.

Figure 5.14 – Original colors, deuteranopia simulation, and recoloring of a cave in Stardew Valley (2016).

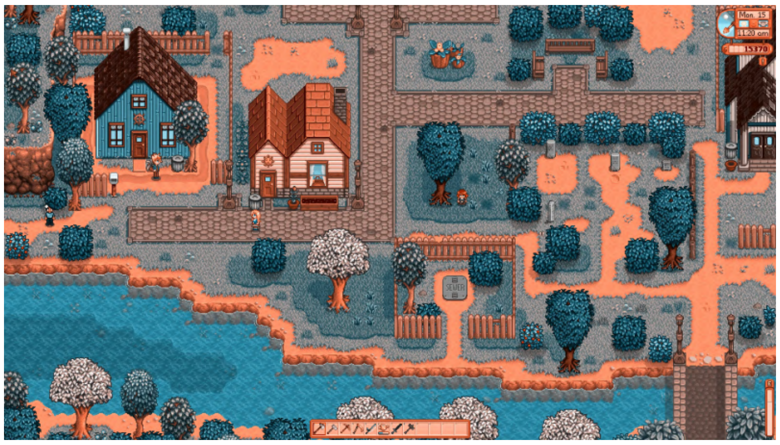


(c) Our recoloring for deuteranopia.

Figure 5.15 – Original colors, tritanopia simulation, and recoloring of a town in Stardew Valley (2016).

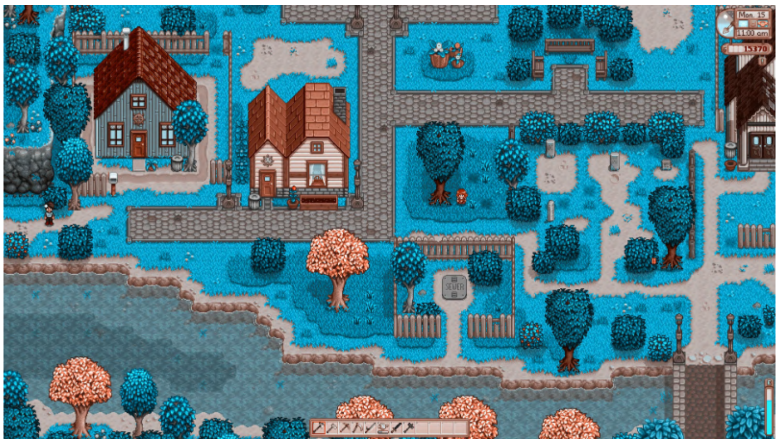


(a) Normal color vision.



(b) Tritanopia simulation.

Figure 5.15 – Original colors, tritanopia simulation, and recoloring of a town in Stardew Valley (2016).



(c) Our recoloring for tritanopia.

To show the recoloring effects of our plugin in real time we have provided, in hyperlinks on the titles, recordings of Hades (2020), Final Fantasy XIV: A Realm Reborn (2013), Shadow of the Tomb Raider (2018), and Stardew Valley (2016), comparing the game's original colors, with a simulation of how a dichromat perceives them, and with our recoloring.

5.2 Performance Analysis

All of the results shown in this undergraduate thesis were obtained using an AMD Ryzen 7 1700 Eight-Core 3.00 GHz processor with 16GB of RAM and an NVIDIA GeForce GTX 1070 GPU, on a monitor with a 3840×2160 resolution. All games were executed at the highest resolution supported by each game: 1920×1080 for Tell Me Why (2020), 2560×1440 for Final Fantasy XIV: A Realm Reborn (2013), and 3840×2160 for the others. We used ReShade's frame counter and timer statistic settings to measure the average frametime of each game with and without our plugin. Measures were taken while the game was actively being played from about two to five minutes. During this time, roughly the same interactions with the game with and without the plugin were performed to maintain the inputs as similar as possible. For example, walking the same path in Shadow of the Tomb Raider (2018) with and without our plugin, and fighting the same group of enemies in Final Fantasy XIV: A Realm Reborn (2013). Table 5.1 shows these measures for the games used in our examples.

ReShade's statistics feature also provides, for each active shader, the time required

Table 5.1 – Average frames per second measured.

Game	API	Avg. Frametime with Plugin	Avg. Frametime without Plugin
Hades	Vulkan	16.69ms	16.88ms
Shadow of the Tomb Raider	DirectX 11	33.05ms	30.78ms
Tell Me Why	DirectX 11	16.66ms	16.66ms
Final Fantasy XIV: A Realm Reborn	DirectX 11	16.69ms	16.69ms
Stardew Valley	OpenGL	16.98ms	16.79ms
Pathfinder: Kingmaker	DirectX 11	10.98ms	10.89ms

in the CPU and GPU to execute it for every frame. Table 5.2 shows these times for each game previously mentioned. To obtain these values, we measured the average execution time for thirty random frames. For the current version of ReShade, there is a bug that prevents this information from being displayed correctly for games based on Vulkan. Because of this, we cannot provide this information for the game Hades (2020). As expected, the running times of our shaders increase with the game's resolution, and this is the only factor that affects the time required to execute them.

Table 5.2 – Average per-frame plugin execution time.

Game	Resolution	Avg. CPU Time	Avg. GPU Time
Shadow of the Tomb Raider	3840×2160	0.164ms	2.898ms
Tell Me Why	1920×1080	0.052ms	3.722ms
Final Fantasy XIV: A Realm Reborn	2560×1440	0.046ms	0.913ms
Stardew Valley	3840×2160	0.205ms	1.898ms
Pathfinder: Kingmaker	3840×2160	0.078ms	6.708ms

For games without heavy graphic requirements, no significant performance loss was noticed in this environment. In earlier stages of development, Stardew Valley (2016) kept displaying a small but noticeable ripple effect whenever the scene moved when using our plugin, even after noise generation improvement described in Section 4.3.2. At the time, our plugin had an average GPU execution time of 16.091ms for this game. With ReShade's latest update on September 16th of 2022, however, improvements for the OpenGL API were made, and this is no longer observed. Shadow of the Tomb Raider (2018) showed an average loss of approximately 2 frames per second, from around 32 frames per second to around 30, while using our plugin, but as this is a much more computationally demanding game, this was not an unexpected result. Even then, the game's performance remains above 30 frames per second, a framerate still considered acceptable.

5.3 Summary

This chapter has presented various results obtained from applying our plugin to various games. We have outlined the situations in which its usage might be more beneficial to players with CVD, and where our plugin can still be improved. Lastly, we presented some performance metrics extracted from ReShade about the execution time required by our plugin, and how it affects the performance of the games.

6 CONCLUSION AND FUTURE WORK

We have presented a real-time method to recover color contrast information in electronic games, porting an efficient recoloring algorithm to an application that allows us to easily apply it to various games in only a few steps. Our solution provides a consistent, reliable way to make a game's visual elements more distinguishable for dichromat players, without significantly impacting its performance.

While accessibility in the world of video games has been a highly discussed topic in more recent years, the vast majority of developers still do not offer effective methods to adapt their games for those with impaired color vision. Our results show that said methods can be easily implemented, and their presence could come to be the norm.

In future steps, the experience of players with CVD with our plugin should be taken into consideration, and their feedback used as basis for any necessary improvements. We would like to analyze how the use of our plugin can reduce the time it takes for a dichromat to discern color information in games, especially in time sensitive situations. While the measures taken by us indicate that, in most cases, our plugin does not significantly impact a game's performance, much more insight will be gained from obtaining these measures in different machines, on a wider variety of games with different system requirements.

At its current state, our solution is capable of reliably and efficiently enhancing contrast for dichromats between adjacent elements in an image, such as distinguishing ground markers from the ground itself in our examples provided for Final Fantasy XIV: A Realm Reborn. As seen for the game Tell Me Why, however, it might not be the best choice where it is important to maintain the contrast between screen elements that are far apart from each other. Thus, if the player wanted to enhance the contrast between two different ground markers that are not adjacent, for example, there would be no guarantee that our technique would produce the desired results. In the future, it would be beneficial to offer the option to keep distant UI elements easily distinguishable from each other, if it is what the user would prefer. Additionally, further steps can be taken to preserve the naturalness in the images to maintain a visually pleasing experience.

The use of the ReShade application to develop our plugin limits its usage to computer games running on a Windows system. However, despite being a popular one, computers aren't the most used platform for games. Even players who favor computers over mobile devices and consoles might choose to use these other platforms from time to time.

Once polished, our work can be expanded to be compatible with a wider range of systems.

It is also important to note that this plugin was developed with dichromats in mind. A method to control the intensity of the recoloring might be an interesting feature for anomalous trichromats who retain a reduced ability to differentiate certain colors, and have a less limited range of available colors to work with.

REFERENCES

BOX, G. E. P.; MULLER, M. E. A note on the generation of random normal deviates. **Annals of Mathematical Statistics**, 1958.

BRETTEL, H.; VIéNOT, F.; MOLLON, J. D. Computerized simulation of color appearance for dichromats. **J. Opt. Soc. Am. A**, Optica Publishing Group, v. 14, n. 10, p. 2647–2655, Oct 1997.

CROSIRE. **ReShade**. [S.l.], 2015. Available from Internet: <https://reshade.me/>.

DALTONIZE.ORG. **LMS Daltonization**. [S.l.], 2010. Available from Internet: <http://www.daltonize.org/2010/05/lms-daltonization-algorithm.html>.

DFC INTELLIGENCE. **Global Video Game Consumer Segmentation**. [S.l.], 2022. Available from Internet: <https://www.dfcint.com/product/video-game-consumer-segmentation-2/>.

DOUGHERTY, B.; WADE, A. **Daltonize**. [S.l.], 2002. Available from Internet: <http://www.vischeck.com/daltonize/>.

GOOCH, A. et al. Color2gray: Salience-preserving color removal. **ACM Trans. Graph.**, v. 24, p. 634–639, 07 2005.

GORDON, N. Colour blindness. **Public Health**, v. 112, n. 2, p. 81–84, mar. 1998.

GRUNDLAND, M.; DODGSON, N. A. Decolorize: Fast, contrast enhancing, color to grayscale conversion. **Pattern Recognition**, v. 40, n. 11, p. 2891–2896, 2007. ISSN 0031-3203.

HARDIN, B. **Colorblind accessibility in video games**. [S.l.], 2016. Available from Internet: <https://www.gamersexperience.com/colorblind-accessibility-in-video-games-is-the-industry-heading-in-the-right-direction/>.

IACCARINO, G. et al. Efficient edge-services for colorblind users. In: **Proceedings of the 15th international conference on World Wide Web**. [S.l.: s.n.], 2006. p. 919–920.

ICHIKAWA, M. et al. Web-page color modification for barrier-free color vision with genetic algorithm. In: **LNCS**. [S.l.: s.n.], 2003. v. 2724, p. 2134–2146. ISBN 978-3-540-40603-7.

ICHIKAWA, M. et al. Preliminary study on color modification for still images to realize barrier-free color vision. In: **2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)**. [S.l.: s.n.], 2004. v. 1, p. 36–41 vol.1.

IRPAN, E. et al. **COVID-19's Impact on the Gaming Industry: 19 Takeaways**. [S.l.], 2020. Available from Internet: <https://create.unity.com/COVID-19s-impact-on-the-gaming-industry>.

JEFFERSON, L.; HARVEY, R. Accommodating color blind computer users. In: **Proceedings of ASSETS 2006. Portland, USA**. [S.l.: s.n.], 2006. v. 2006, p. 40–47.

JUDD, D. B. The color perceptions of deuteranopic and protanopic observers. **J. Opt. Soc. Am.**, Optica Publishing Group, v. 39, n. 3, p. 252–256, Mar 1949. Available from Internet: <https://opg.optica.org/abstract.cfm?URI=josa-39-3-252>.

KUHN, G. R.; FERNANDES, L. F.; OLIVEIRA, M. M. An efficient naturalness-preserving image-recoloring method for dichromats. **IEEE Transactions on Visualization & Computer Graphics**, IEEE Computer Society, Los Alamitos, CA, USA, v. 14, n. 06, p. 1747–1754, nov 2008. ISSN 1941-0506.

MACHADO, G. M.; OLIVEIRA, M. M. Real-time temporal-coherent color contrast enhancement for dichromats. **Computer Graphics Forum**, v. 29, n. 3, p. 933–942, 2010.

NEITZ, M.; NEITZ, J. Molecular Genetics of Color Vision and Color Vision Defects. **Archives of Ophthalmology**, v. 118, n. 5, p. 691–700, 05 2000. ISSN 0003-9950.

RASCHE, K.; GEIST, R.; WESTALL, J. Detail preserving reproduction of color images for monochromats and dichromats. **IEEE computer graphics and applications**, v. 25, p. 22–30, 05 2005.

RASCHE, K.; GEIST, R.; WESTALL, J. Re-coloring images for gamuts of lower dimension. **Computer Graphics Forum**, v. 24, p. 423–432, 09 2005.

SAMPLE, P. A.; WEINREB, R. N.; BOYNTON, R. M. Acquired dyschromatopsia in glaucoma. **Survey of ophthalmology**, v. 31 1, p. 54–64, 1986.

SCOTT, I.; FEUER, W.; JACKO, J. Impact of visual function on computer task accuracy and reaction time in a cohort of patients with age-related macular degeneration. **American Journal of Ophthalmology**, v. 133, p. 350–7, 03 2002.

SHARPE, L. T. et al. Opsin genes, cone photopigments, color vision, and color blindness. In: **Color Vision: From Genes to Perception**. Cambridge, UK: Cambridge University Press, 1999. p. 3–51.

SHEPARD, D. A two-dimensional interpolation function for irregularly-spaced data. In: **Proceedings of the 1968 23rd ACM National Conference**. New York, NY, USA: Association for Computing Machinery, 1968. (ACM ’68), p. 517–524. ISBN 9781450374866.

VIéNOT, F.; BRETTEL, H.; MOLLON, J. D. Digital video colourmaps for checking the legibility of displays by dichromats. **Color Research and Application**, v. 24, p. 243–252, 1999.

WAKITA, K.; SHIMAMURA, K. Smartcolor: Disambiguation framework for the colorblind. In: **ASSETS 2005 - The Seventh International ACM SIGACCESS Conference on Computers and Accessibility**. [S.l.: s.n.], 2005. p. 158–165.