FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Color Distribution Transfer For Mixed-Reality Applications

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software and Information Engineering

eingereicht von

Stefan Spelitz
Matrikelnummer 0925601

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Mag.rer.soc.oec. Martin Knecht, Bakk.techn.

Wien, 01.08.2012

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Color Distribution Transfer For Mixed-Reality Applications

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Stefan Spelitz
Registration Number 0925601

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Mag.rer.soc.oec. Martin Knecht, Bakk.techn.

Vienna, 01.08.2012

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Stefan Spelitz
Rotenlöwengasse 8/12, 1090 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

In mixed-reality environments it is essential to integrate virtual objects seamlessly into a real scene. Virtual objects should have similar appearances to those of real objects captured by a video camera. This is useful for many real-life application scenarios, including product advertising and visualization, edutainment systems or for enhancing cultural heritage sites.

Typical problems in this domain are to match the current 'color mood' of the video camera scene with the colors of virtual (rendered) objects. The color mood depends on the global illumination conditions as well as the hue, saturation or white balance settings of the camera.

The aim of this paper is to integrate existing methods of histogram transfers used in the domain of computational photography into mixed-reality environments. These methods allow us to simulate current luminance conditions in the scene and changes in the camera driver settings to apply them onto virtual objects.

This thesis contains two fast-running approaches to provide a color mapping between virtual objects and the real scene, which can be used in real-time applications. The results show that these methods increase the immersion of virtual objects in a real scene.

Contents

CHAPTER 1

Introduction

1.1 Mixed-reality applications

Mixed-reality attempts to embed virtual objects into the real world. This is typically done by using a video stream of a camera, merging rendered objects into the video input and presenting the result on an output device, like a PC monitor or a mobile device. One goal might be to create a perfect illusion so that virtual objects cannot be distinguished from real, existing objects. These techniques may be used in (but not limited to) product designs, architectural and urban visualizations or for marketing reasons. To create such an illusion, different approaches already exist. Klein and Murray [6] introduced methods to simulate camera artifacts, e.g. distortion, chromatic aberrations and blur on virtual objects. These artifacts are applied onto the virtual objects, before they are merged with the video stream. Other methods [7] are taking direct and indirect illumination effects into account to simulate mutual lighting effects between real and virtual objects.

1.2 Problem statement

Although existing methods [6], [7] create accurate results, these methods don't consider matching the actual colors between the virtual objects and the camera scene. Therefore virtual objects are still easily distinguishable from the real scene by the user.

Knecht et al. [8] developed a method to match the colors between virtual objects and the camera scene. This is done by creating color sample pairs based on matching similar colors as well as through a heuristic function. A color mapping function is then derived from the color sample pairs. The method performs well if there are similar colors on virtual and real objects. If there is not enough matching information in the camera scene or the differences between the colors of virtual and real objects are too extreme, a correct mapping will likely fail and lead to incorrect colors in the final result.

To resolve these issues, the primary goal is to find a stable color mapping function to transfer the 'color mood' of the camera scene onto the virtual objects. This is done by converting the colors of the virtual objects to match those of the camera scene. In addition the function must perform reasonably fast, because of its use in real-time applications.

CHAPTER 2

Related Work

2.1 Definition and concepts

Differential Instant Radiosity

Knecht et al. [7] developed a method called 'Differential Instant Radiosity' (DIR), which is the core of the framework used in this paper. Its purpose is to combine differential rendering (DR) and instant radiosity to be used in mixed-reality applications. By doing so it is possible to calculate effects like shadow casting and indirect illumination between real and virtual objects. The main aspect used in this paper is the work about differential rendering. To use DR the following information is needed:

- The camera image (CI)
- One global illumination (GI) solution containing virtual and real objects (LS_{rv})
- One GI solution containing only the real objects (LS_r)

The actual DR process is done by creating the difference between LS_{rv} and LS_r after both solutions have been tone and color mapped. The difference (i.e. $LS_{rv} - LS_v$) is then applied to a masked CI to obtain the final result.

By using a virtual representation of real objects (i.e. LS_r) it is possible to measure the difference between LS_r and the CI. This measurement can then be used to do the actual mapping between virtual objects and the real scene. This will be discussed in more detail in Chapter 4.

Tone mapping

Tone mapping (also known as tone reproduction) is the operation of converting high dynamic range (HDR) images to low dynamic range (LDR). The problem is known from analog photography where it's a common task to convert the HDR of real world luminances to the LDR of print media. In computer graphics the same problem occurs when trying to map a virtual scene

as an output to a LDR screen. Luminances in the real world are spanning a wide range, while in LDR media, the range of light is much narrower. To resolve this discrepancy, the range of luminances are scaled down, while still maintaining the hue values. For the method in this paper, tone mapping is necessary to map virtual objects to the LDR of the camera input, in order to calculate the differences between virtual and real scene and to finally merge the virtual objects into the real scene. Because only the luminances are changed, a color mapping is still necessary (which will be discussed in Section 2.2).

The tone mapping operator chosen in this paper is based on the work of Reinhard et al. [14]. Only the global tone mapping operator is used, which consists of the following steps. First, by calculating the log-average luminance, which is an approximation for the scene lighting (known as 'key') of the image to be mapped:

$$L_{avg} = \exp(\frac{1}{N} \sum \log(\delta + L_w(x, y))) \tag{2.1}$$

where N is the total number of pixels in the image, δ is a small factor to avoid problems with zero values and $L_w(x, y)$ is the 'world' luminance of pixel (x, y). The new luminance, using a 'key' value a to match, is calculated by using:

$$L(x, y) = \frac{a}{L_{avg}} L_w(x, y) \tag{2.2}$$

Which must be scaled to be within displayable range:

$$L_s(x, y) = \frac{L(x, y)}{1 + L(x, y)} \tag{2.3}$$

2.2 Transfer of color statistics

The first popular method by Reinhard et al. [12] matches the mean and standard deviation of a source image to that of a target image. This is done separately for each color channel in L$\alpha\beta$ color space (see Section 3.3). Based on this method Kim et al. [5] proposed a method which also works in L$\alpha\beta$ color space but is using an additional pre-processing of the source image's colors and transforms only the α and β channels. Although the method of Reinhard et al. produces convincing results, it works with statistical data of the whole image and thus can create new colors in the result by mixing up two or more colors of the target image. Xiao and Ma [17] tried to solve this problem with histogram matching (sometimes called histogram specification) and a post-processing step to preserve the gradients of the source image. Histogram matching (HM) is the transformation of one image's color histogram to another histogram. By using HM, the tonal distribution is changed, allowing to control the relative frequency of each tonal value.

Another way of performing histogram matching is to create an image dependent color space instead of using a fixed one. This is done by eliminating the coherence between the color channels, also with the idea to perform color mapping on each color channel separately. A non-linear mapping with this approach has been proposed by Grundland and Dodgson [3]. Similarly, not depending on a fixed decorrelated color space, Xiao and Ma [18] decompose the source and

target image data into its principal components (with singular value decomposition) to perform a one-dimensional color mapping.

Besides these methods which perform color mapping on each color channel, there are those which are trying to solve the color mapping in N-dimensions. Neumann and Neumann [9] are using a computationally simple, permissive, or optionally strict 3D histogram matching. Instead of using opponent color channels they are using a cylindric color space to map hue, lightness and saturation as their main attributes. Another N-dimensional mapping has been proposed by Pitié et al. [10]. In their work they use a N-dimensional probability density function transformation with an involved post-processing step, which matches the gradient field of the output image to the source image.

The requirements of a color mapping function, for usage in real-time mixed-reality applications are to transfer the colors without additional user interaction and to allow real-time framerates. Methods which allow the user to control the amount of transformation (like [11]) are useful for a manual matching of arbitrary images, but in real-time applications simple, fast-running methods are necessary, which must be well suited for generic automated tasks. Additionally by working with a virtual representation of the real world (i.e. LS_r) we have a somewhat less complicated environment for color mapping. This is because LS_r and the real world (i.e. the camera image) typically are similar to each other, both in shape and color appearance.

CHAPTER 3

Color Spaces

3.1 Introduction

As stated in the previous section, color mapping algorithms can typically be categorized into two classes. There are those which operate in a N-dimensional color space and those which operate on each color axis separately. The mapping algorithms presented in this paper are working on a per-color-axis basis. It has been observed that if the channels can be made strongly decorrelated then image processing can be done in each channel independently [12]. So it is assumed, that the choice of the color space is important for algorithms which perform one-dimensional matching.

Reinhard et al. [13] compared the quality of color mapping in the domain of different color spaces (e.g. CIELab, L$\alpha\beta$, HSV, XYZ) in combination with several environment settings (e.g. indoor, day, night). Although it seems plausible to see more indoor-specific mixed-reality applications, color mapping in mixed-reality environments cannot make assumptions about the environment it is used in. Therefore it is necessary to choose a color space with overall good performance results. Reinhard et al. [13] concluded in their work:

> *'Surprisingly, we find that CIELAB, if used with illuminant E as the white point leads on average to the best performance, yielding a plausible colour transfer in 77 % of all cases tested.'*

Because CIELab (E) performs well in all tested environments and especially in indoor areas, it is the color space of choice in this paper.

The color space L$\alpha\beta$ is used in several color mapping algorithms [5, 12, 16, 17], including the original color transfer method of Reinhard et al. It is mentioned in this paper as well.

3.2 RGB color space

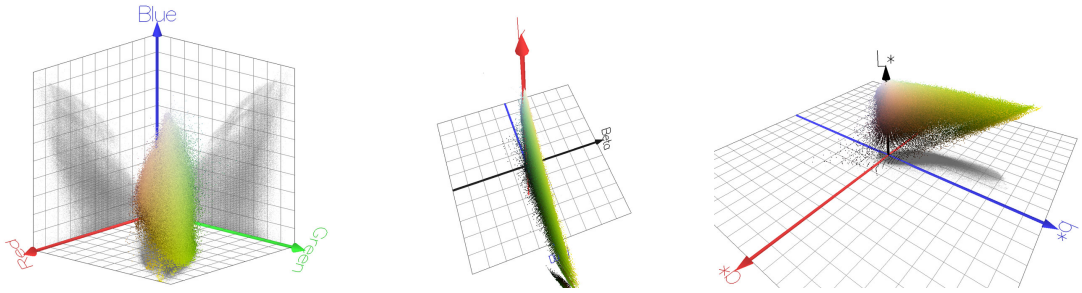
The device dependent RGB color space consists of red, green and blue chromatic axes. It is device dependent, because the actual displayed colors, represented by RGB values, vary from

device to device or even when altering settings (like brightness or contrast).

This color space tends to have axes, which are strongly correlated. An example is displayed in Figure 3.1. In this example, the values are typically small in each channel, if the pixels are dark. The values are getting larger as the luminance rises. If the blue channel's values are large, then most values in the red and green channels are getting larger, too. This results in an almost diagonal distribution between the axes, which signals a strong correlation.

Therefore, when changing the color of a pixel to match another one, it is necessary to change all color channels simultaneously. This results in more complex color matching techniques, especially when trying to compute each color channel separately. Thus, in this paper, the RGB color space will not be used to perform color mapping.

(a) *Input image*



(b) *RGB correlation* **(c)** *L$\alpha\beta$ correlation* **(d)** *CIELab correlation*

Figure 3.1: *Decorrelation properties of color spaces. The bottom row shows pixel values plotted in three-dimensional space for different color spaces. The top image is used as input. RGB shows an almost diagonal distribution on each pair of axes. L$\alpha\beta$ distribution is along L and α axes. CIELab distribution is along L* and b* axes. Plots created with ColorSpace [2]*

3.3 L$\alpha\beta$ color space

Ruderman et al. [15] discovered in their work the logarithmic color space L$\alpha\beta$. It is a transformation of the LMS cone space [1]. Its axes are strongly decorrelated for natural images. The first axis represents an achromatic channel. It is a measurement for equal logarithmic fluctuations in all three LMS channels (see Eq. 3.4). Such a fluctuation would occur if a scene would change in radiance. Thus the first axis is generally referred as the 'radiance' (or luminance). The other two

[1]The LMS color space is based on human's perception of colors. It is named after the sensitivity of the human eye to long (L), medium (M) and short (S) wavelengths of light.

axes are representing the blue-yellow and red-green chromatic opponent channels. Because of the decorrelation between the axes of this color space, it is suitable for operating on each color axis separately.

L$\alpha\beta$ color space was primarily intended for natural images, which need not be the main focus of mixed-reality. Therefore in this paper CIELab (E) has been chosen instead of L$\alpha\beta$, which also shows better performance results (see Section 3.1).

RGB to L$\alpha\beta$

Transforming images from RGB to the L$\alpha\beta$ color space is done by first converting them to the device independent XYZ space. The conversion depends on settings (like white point and gamma correction) of the device, where the images are intended to be displayed on. This information is typically not available. Therefore we will map white in the CIE 1931 xy chromaticity diagram to white in RGB space and vice versa. Illuminant E is used as white point definition, which's chromaticity coordinates are $(x, y) = (\frac{1}{3}, \frac{1}{3})$. Reinhard et al. [12] provided a matrix for conversion, which is a modified version of the International Telecommunications Union (ITU) standard conversion matrix. The original matrix (CIE XYZitu601-1) is based on illuminant D65. The modified conversion matrix for illuminant E is:

$$M_E = \begin{bmatrix} 0.5141364 & 0.32387860 & 0.16036376 \\ 0.2650680 & 0.67023428 & 0.06409157 \\ 0.0241188 & 0.12281780 & 0.84442666 \end{bmatrix} \tag{3.1}$$

The conversion from RGB to XYZ color space is done through:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M_E \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.2}$$

After which the values are converted to LMS space:

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3897 & 0.6890 & -0.0787 \\ -0.2298 & 1.1834 & 0.0464 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.3}$$

And finally converted to L$\alpha\beta$:

$$\begin{bmatrix} L \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \log L \\ \log M \\ \log S \end{bmatrix} \tag{3.4}$$

L$\alpha\beta$ to RGB

These are the inverse steps to transform images back to RGB color space. First step is to get back to logarithmic LMS space:

$$\begin{bmatrix} \log L \\ \log M \\ \log S \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{3} & 0 & 0 \\ 0 & \frac{\sqrt{6}}{6} & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} L \\ \alpha \\ \beta \end{bmatrix} \tag{3.5}$$

Then, after raising ten to the power of the logarithmic LMS values to go back to linear color space, the transformation from LMS to RGB can be achieved by using:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 4.4679 & -3.5873 & 0.1193 \\ -1.2186 & 2.3809 & -0.1624 \\ 0.0497 & -0.2439 & 1.2045 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix} \tag{3.6}$$

3.4 CIELab color space

CIELab (also CIE L*a*b*) is a device independent color space with three axes. 'L' represents the lightness of the color with a range from 0 (black) to 100 (white). The other two axes are representing the blue-yellow (channel 'b') and red-green (channel 'a') chromatic opponent channels with an unbounded range. It is a non-linear transformation of the CIE XYZ color space, while still remaining reversible. It is considered to be perceptually uniform. This means, that the euclidean distance of two colors in CIELab are reflected as equally distant in perception. The color space is based on the XYZ tristimulus values of the reference white point. In this paper the values of standard illuminant 'E' will be used for conversion.

As already stated in the introduction (Section 3.1) this color space has been chosen because of its good performance results in the domain of color mapping.

RGB to CIELab

Transforming images from RGB to CIELab (E) color space is done by first converting them to the device independent XYZ space (see Eq. 3.2). After which the values are converted to CIELab:

$$\begin{bmatrix} L^* \\ a^* \\ b^* \end{bmatrix} = \begin{bmatrix} 0 & 116 & 0 & -16 \\ 500 & -500 & 0 & 0 \\ 0 & 200 & -200 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix} \tag{3.7}$$

Where x_i is defined as:

$$\forall i \in \{1,2,3\} : x_i = \begin{cases} 7.787 a_i + \frac{16}{116} & \text{if } a_i \leq 0.008856 \\ \sqrt[3]{a_i} & \text{otherwise} \end{cases} \tag{3.8}$$

and

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \mathrm{diag}(E)^{-1} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.9}$$

With standard illuminant E as the white point, which is $[X_n, Y_n, Z_n] = [100, 100, 100]$.

CIELab to RGB

The inverse transformation from CIELab to RGB is done through the following steps. First by transforming CIELab to device independent XYZ color space:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathrm{diag}(E) \cdot \begin{bmatrix} f(b_1) \\ f(b_2) \\ f(b_3) \end{bmatrix} \tag{3.10}$$

again with illuminant E as the white point and

$$f(b) = \begin{cases} (b + \frac{16}{116})^3 & \text{if } L^* > 7.9996 \\ \frac{b}{7.787} & \text{otherwise} \end{cases} \tag{3.11}$$

Where b_i is calculated through:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{116} & \frac{1}{500} & 0 \\ \frac{1}{116} & 0 & 0 \\ \frac{1}{116} & 0 & -\frac{1}{200} \end{bmatrix} \begin{bmatrix} L^* \\ a^* \\ b^* \end{bmatrix} \tag{3.12}$$

The final transformation from XYZ to RGB color space is done by:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 2.5651 & -1.1665 & -0.3986 \\ -1.0217 & 1.9777 & 0.0439 \\ 0.0753 & -0.2543 & 1.1892 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.13}$$

CHAPTER 4

Method

4.1 Introduction

The method is based on Differential Instant Radiosity (see Section 2.1). It is assumed that geometric representations of real world objects (at least some objects) are available. The LS_r and LS_{rv} solutions contain geometric models of high dynamic range (HDR). Because the captured camera image is only in low dynamic range (LDR) a tone mapping operation (see Section 2.1) is necessary. After the tone mapping has been applied, all information is in a common LDR color space.

As a result the method creates a merged image which consists of the camera image, virtual objects, shadows and reflections. By using a suitable color mapping function it will merge virtual objects seamlessly into the surrounding environment, defined by the camera image.

4.2 Application flow

Figure 4.1 shows the abstract application flow of this method with the help of an example. In the camera image four real existing objects are available. The wooden surface and the color chart have similar geometric representations in the application. The red figure and the book have no virtual representation. Therefore LS_r contains the wooden surface and the virtual representation of the color chart. LS_{rv} contains in addition to the content of LS_r, the object to be rendered into the real scene, which is another color chart.

The actual color mapping process is divided into stages, which will be explained next.

Stage 1

After LS_r and LS_{rv} have been tone mapped, they are converted together with the CI to the CIELab color space. This is done to minimize correlation between the color axes, so that manipulations of one color axis don't affect the other axes as well.

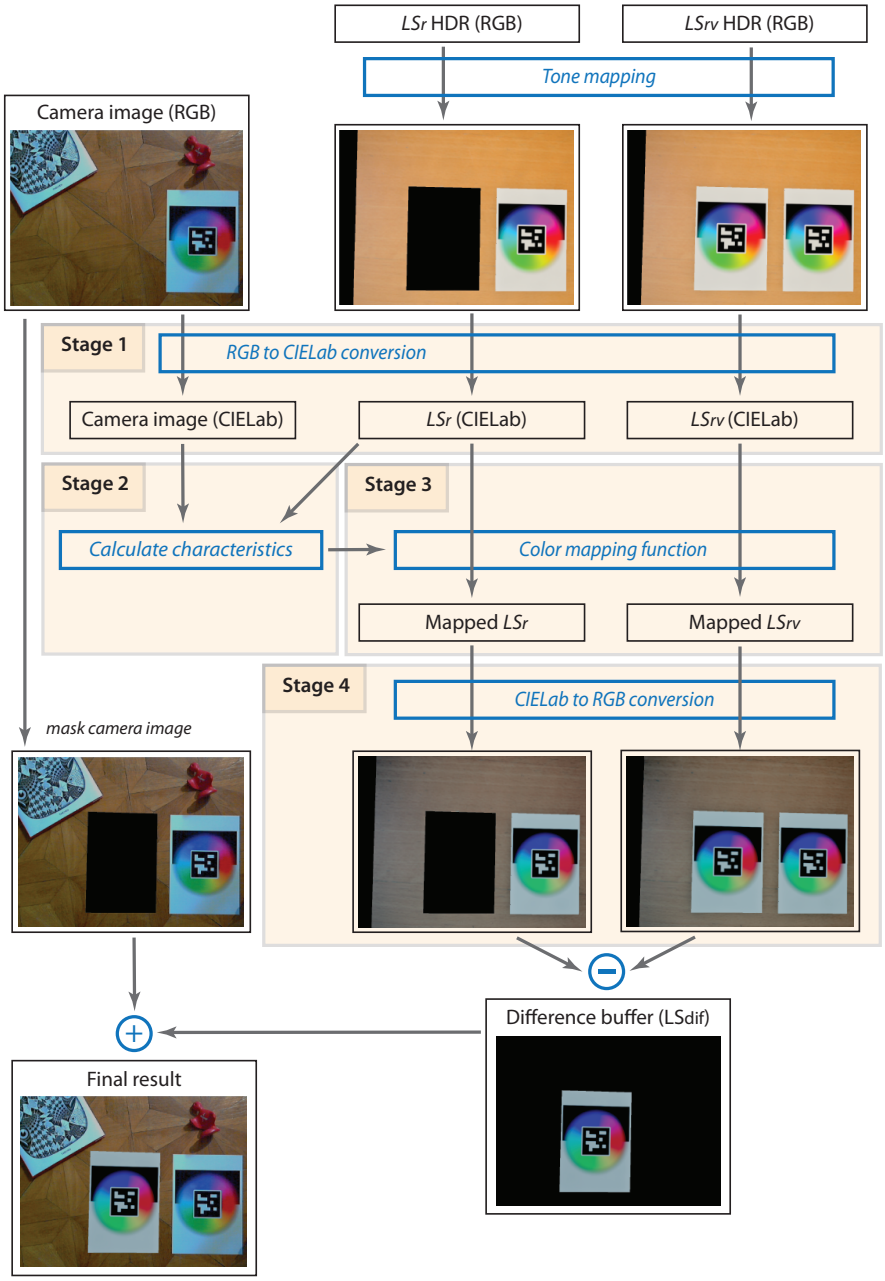


Figure 4.1: *This figure shows the application workflow. By calculating the characteristics of the real-world in comparison to LS_r, the color mapping function is applied to the LS_r and LS_{rv} solutions. The color mapping operates in the decorrelated CIELab color space. The difference between the buffers is then merged with the camera image to create the final result.*

Stage 2

For the actual color mapping function we need to calculate the differences between our representation of the real-world (i.e. LS_r) and the actual real-world itself (i.e. the camera image). By determining the characteristics between these two images, the resulting values can be used by the 'color mapping function' (CMF). Depending on what CMF algorithm is used, different characteristics are collected. This will be explained together with the CMF in Section 4.3.

Stage 3

The CMF applies the characteristics to both, LS_{rv} and LS_r, in order to convert their colors to match those of the camera image. Possible algorithms of the CMF will be explained in depth in Section 4.3.

Stage 4

The CIELab conversion was only necessary to calculate the characteristics and perform the color mapping. So after the color mapping is done, LS_{rv} and LS_r will be converted back to RGB color space.

Obtaining the result

We get the final result by calculating the difference

$$LS_{dif} = LS_{rv} - LS_r$$

and adding the difference buffer to a masked version of the camera image.

4.3 Color mapping function

Color transfer

This mapping function is based on the work of Reinhard et al. [12]. Although its primary purpose is to transfer the colors between two images, in this paper three images will be involved. This is done by imposing the color characteristics between LS_r and the camera image to use them not only for LS_r, but also for LS_{rv}. The algorithm works on each color axis separately and therefore needs a decorrelated color space.

The first step is to calculate the mean and standard deviation of the source (i.e. LS_r) and the target (camera) images. Denoted by μ_s, μ_t and σ_s, σ_t. The next step is to subtract the source mean from each data point (dp) of the LS_r and LS_{rv} solutions:

$$dp' = dp - \mu_s$$

After which the data points are scaled by factors determined by the standard deviations:

$$dp'' = dp' \frac{\sigma_t}{\sigma_s}$$

And finally the data points are moved by adding the target mean:

$$dp^* = dp'' + \mu_t$$

Because the same transformation is applied to LS_{rv} and to LS_r, colors which are the same in both solutions will remain equal after the mapping. This is an important feature, necessary for differential rendering.

CDF matching

This color mapping function has been chosen as an alternative approach to the 'color transfer' algorithm. Similar to the 'color transfer' method, the cumulative distribution function (CDF) is working with statistical characteristics in a decorrelated color space. It is based on the work of Pitié et al. [10] and Bourke's [1] 'histogram matching'.

By calculating cumulative distribution functions (F_s, F_t) from histograms for the source (i.e. LS_r) and target (camera) images, it is possible to reshape the histograms and therefore the distribution of color values of the LS_r and LS_{rv} solutions. To map a color value c from source to target it is necessary to lookup the probability p of a color to be less than or equal to c in the source CDF:

$$p = F_s(c)$$

Which is then used in the inverse target CDF to lookup the new target color:

$$c' = F_t^{-1}(p)$$

where

$$F_t^{-1}(x) = \inf\{y | F_t(y) \geq x\}$$

Again, as with the 'color transfer', the same transformation is applied to values of LS_{rv} and LS_r and therefore the results can be used for differential rendering.

4.4 Limitations

Because of using differential rendering, the method presented in this paper needs virtual representations of real object's geometry. This is necessary to determine the differences between the representation and the real scene captured by the camera. These differences are then applied by using the color mapping function onto the virtual objects. Although the algorithms also work without a virtual representation of the environment, the results would be less precise because of the lack of mapping information. Therefore this approach should only be used in mixed-reality systems which support the representation of the real scene.

Another obvious limitation are the color mapping functions. These functions are working with statistical data of the whole scene and try to adapt colors of virtual objects to the color average or histograms of the scene. This doesn't need to be correct in every possible scenario. Especially if there are multiple areas in the scene with huge differences in luminance or color setting, the average of the scene might not be the correct mapping target. A possible solution to this would be to divide the scene into sections and perform a color mapping for each section.

The reliability of the ‘cdf matching’ algorithm greatly depends on the given mapping source and target. Because of matching colors by their relative frequencies, colors which are similar to each other before the color mapping, could be quite distinguishable after the mapping operation. Therefore this method may introduce visible artifacts at color transitions.

CHAPTER 5

Implementation

5.1 Introduction

The color mapping functions were realized on a PC running Microsoft Windows 7 64 bit. The framework was developed in C# using the DirectX 10 API in conjunction with the SlimDX library. The goal was to implement as much as possible in shader code, to allow a fast running real-time-application. The used shader language was HLSL.

5.2 Color Transfer

The main aspect when using the color transfer method of Reinhard et al. is to calculate the color characteristics in an efficient and fast way. We will concentrate on doing that in the following paragraphs.

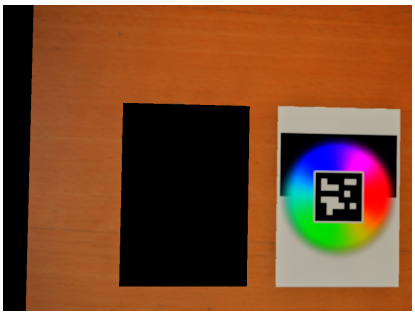


Figure 5.1: *This figure shows the LS_r solution with two virtual representations of real objects (table, color-chart). The black area on the left indicates the end of the virtual table. The black area in the center is the place for the virtual object to be rendered.*

Calculating the mean

The arithmetic mean is defined as:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

where n is the number of data points (x_i), which is $n = w \cdot h$ for an image with the dimensions $[w, h]$.

Calculating the arithmetic mean is easily done by creating mipmaps. Mipmapping will typically only work on quadratic textures. Creating a quadratic texture from a rectangular one is done by bilinear point sampling. Because of using linear interpolation the mean does not get affected.

The mean must be calculated for the camera image as well as for LS_r. Although calculating the mean for the CI is straight forward, this is not the case for LS_r. It contains some areas with no information where there is either no virtual representation for a real object available or where a virtual object will be placed at. See Figure 5.1 for an example. These black areas must not have any influence on the calculated mean value. Excluding these areas from the mean calculation is done by counting the data points to be excluded dpe and correcting the mean μ calculated by the mipmapping operation. To get the correct mean of an image with the dimensions $[w, h]$ we use:

$$\mu_{correct} = \frac{\mu \cdot w \cdot h}{w \cdot h - dpe}$$

Please note that this works only if the data points we want to exclude from the mean calculation have a value of zero.

Note: The corrected mean calculation could have been applied to the masked version of the camera image, too. The masked version contains black areas at each point where a virtual object will be placed at. Although it is possible, there are some drawbacks. The black areas are 'lost information'. These areas won't be included in the calculation, so we have less information about the target environment we want to map to. Therefore we lose precision in the color mapping. In addition, if a virtual object covers the whole scene, there won't be any information available from the CI and thus the mapping would fail.

Calculating the standard deviation

Because the standard deviation is the square root of the variance, we will concentrate on calculating the variance. The variance for discrete values is defined as:

$$var = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{5.1}$$

The variance can easily be calculated. Given a texture with the dimensions $[w, h]$ and $n = w \cdot h$, we first calculate the squared deviation from the mean for each data point in the texture (i.e. $(x_i - \mu)^2$). This operation can be executed in one pass in a shader. The next step is to make the texture quadratic and execute mipmapping to get the arithmetic mean of the sum of squared

deviations. The result of mipmapping is the variance. Please note again that by creating a quadratic texture by using bilinear point sampling, the arithmetic mean does not get affected.

When calculating the variance a similar problem occurs as when calculating the mean for LS_r. Because some data points (count is dpe) shall be excluded from the calculation, we need to correct the variance calculation. Excluded data points have values of zero, so we can rewrite our variance Eq. 5.1 to be:

$$var = \frac{1}{n} \cdot (dpe \cdot (0 - \mu)^2 + \sum_{i \in R} (x_i - \mu)^2) \tag{5.2}$$

with R being our remaining set of data points and having $|R| = n - dpe$. Equation 5.2 can be rewritten to:

$$var = \frac{1}{n} \cdot (dpe \cdot \mu^2 + \sum_{i \in R} (x_i - \mu)^2) \tag{5.3}$$

The corrected variance must only contain the remaining data points and is therefore, according to Eq. 5.1:

$$var_{correct} = \frac{1}{|R|} \cdot \sum_{i \in R} (x_i - \mu)^2$$

Which is equal to (using Eq. 5.3):

$$var_{correct} = \frac{1}{|R|} \cdot (var \cdot n - dpe \cdot \mu^2) \tag{5.4}$$

This shows that it is possible to exclude zero valued data points by using the mipmap-calculated variance and applying Eq. 5.4 to get the corrected variance.

5.3 CDF matching

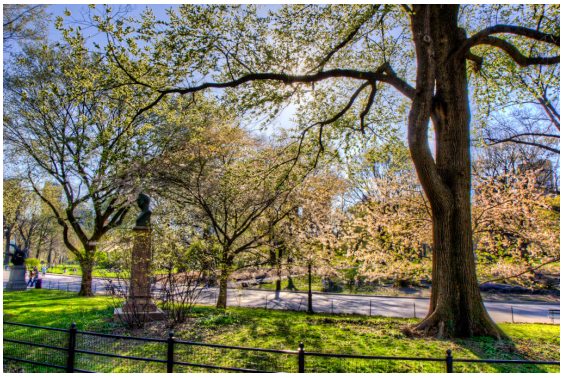
Creating a cumulative distribution function (CDF) from an image is done by first creating histograms on a per color channel basis. Histograms contain for each color the according color count. By normalizing these color counts we get the relative frequencies which can be accumulated to get the resulting CDF.

In practical environments not each color will be counted, but the histogram and the CDF will be divided into sections (called bins). These bins will comprise multiple adjacent colors and for each bin, a color count will be calculated.

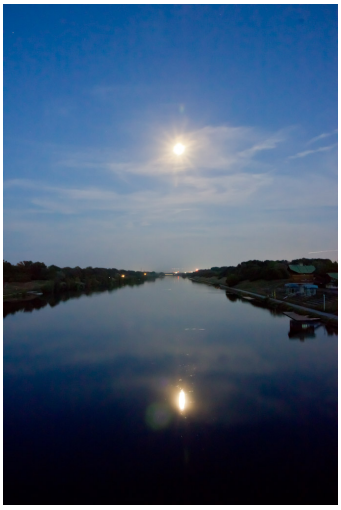
A bin of a CDF is centered around one color value and contains the cumulative sum of relative frequency of the color counts (i.e. the probability of a color to be less than or equal to the bin-center-color). To get smooth results, the lookup of the probability of one color is done by using linear interpolation between the probabilities of two adjacent bins. The reverse lookup (i.e. the lookup from a quantile to a color value) is also done by linear interpolation of the color values of two adjacent bins. Without linear interpolation the result image would only contain as many colors as the bin count.

Bin count

Calculating the color count for one bin is done in two shader passes. First by determining if a color is in the corresponding bin (i.e. 1 to be in this bin and 0 if not). And second by summing up these values. Thus the number of bins greatly affects the speed of the CDF matching algorithm. Choosing a lower bin count increases performance but decreases the matching quality. Figure 5.3 demonstrates the influence of choosing a bin count on the final result. This example shows that choosing a bin count of 500 almost perfectly matches the 'target' CDF. Choosing a bin count between 10 and 500 leads to smoother color transitions, especially around the highlights. Whereas a bin count of 5 does not match the images in a pleasant way.

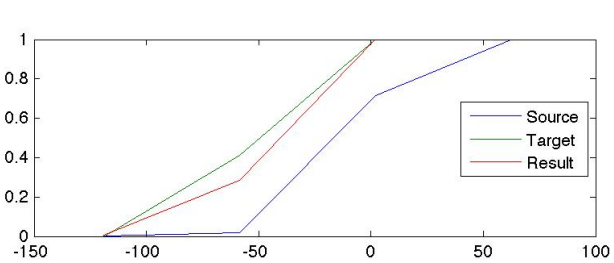


(a) *Source*

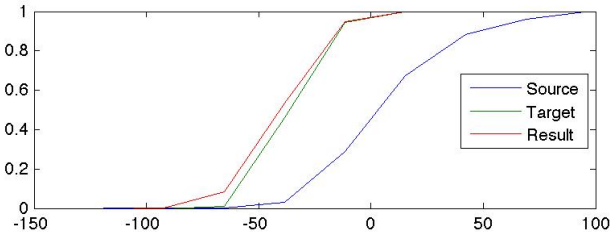


(b) *Target*

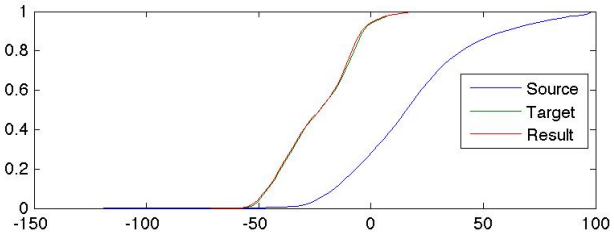
Figure 5.2: *Source image (as in the one whose colors we're changing) on the left. Target image (as in the one whose colors we want to match to) on the right.*

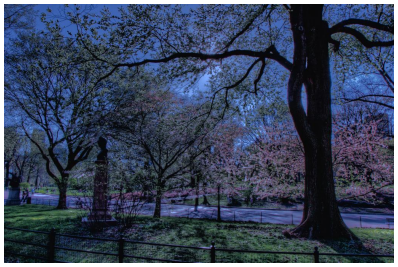


(a) *5 bins*

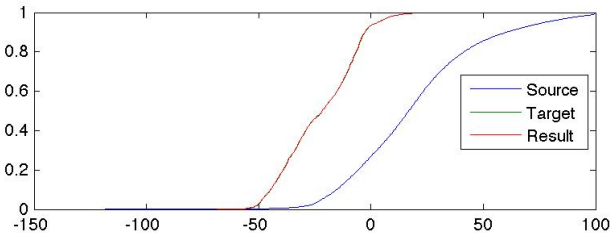


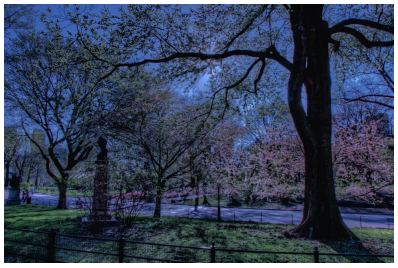
(c) *10 bins*





(e) *100 bins*





(g) *500 bins*

Figure 5.3: *CDF matching by using the images from Figure 5.2 with different bin counts. On the left side is the CDF of channel 'b*' of the CIELab color space with the corresponding result on the right.*

CHAPTER 6

Results

6.1 Test setup

The PC used for the test results has an Intel Core i7-950 Quad 3.06 GHz CPU with 6 GB RAM and a nVIDIA GeForce 9800 GTX+ graphics card. Details about the developing environment and the operating system are mentioned in the implementation Chapter 5.

As seen in Figure 6.1 the test setup contains multiple real-existing objects. A wooden surface, a book, a red figure and a color chart to the right. The application has only two registered virtual objects, which is the wooden surface and the color chart. Broadly speaking there are three different environments, in which the methods operate. These are

- the 'default state' without tweaks of the camera driver settings
- scenes with changed camera settings (contrast, saturation)
- environments with obstacles hiding the color chart

The goal is to render a virtual color chart object (placed left of the real-existing one) which matches the color settings of the surrounding environment. The virtual object representation should match the real object's appearance. If there is no real object for comparison available, the virtual object should fit into the environment in a harmonic way.

6.2 Comparison

The following methods have been compared:

- Color transfer (see Section 4.3)
- CDF matching (see Section 4.3)

- Adaptive camera-based color mapping by Knecht et al. [8]
- Photographic tone reproduction (tone mapping) by Reinhard et al. [14]

The tone mapping operation by Reinhard et al. is influenced by the global illumination solution and does not react on changes in the scene or camera settings. It is a tone mapping operator and not a color mapping function. Thus it can be seen as a comparison of how the virtual object would look like without any color mapping.

The CDF matching algorithm has been configured with a sufficient bin count of 50. A higher bin count didn't change the result in a relevant way. The algorithm does not adapt well to low saturation or contrast scenes leaving black areas of the color chart untouched. In addition there are visible artifacts in the color gradient without smooth color transitions in most of the test scenes. Another serious drawback is the performance. It is by far the slowest of the tested color mapping operations. Changing the implementation of the CDF matching algorithm could result in significantly improved performance results, although visible artifacts would still remain.

The method of Knecht et al. adapts well to changes in the camera settings but the built in heuristic function fails to find a color mapping with the real-existing color checker board (Gretagmacbeth - ColorChecker Digital SG). In the last scene with only some real-existing colored paper spread out, the method nearly completely adapts to the existing colors in the scene, which typically isn't the desired result. It outperforms the CDF matching algorithm and compared to the tone mapping operation, it is a reasonably fast color mapping technique.

Using the 'color transfer' method results in a good adaptation to changed camera settings, but some color intensity is lost in the yellow, green and cyan areas of the virtual object. Especially when using a high contrast level or the color checker board. In the scene with only some colored paper spread out, the method does not adapt to the new colored environment but only attempts to darken the colors, when compared to the 'tonempaping' operator. The performance results are quite similar to the results of Knecht et al.

Essentially, the conclusion of the tests is that the 'color transfer' method is the best choice for color mapping in mixed-reality applications.

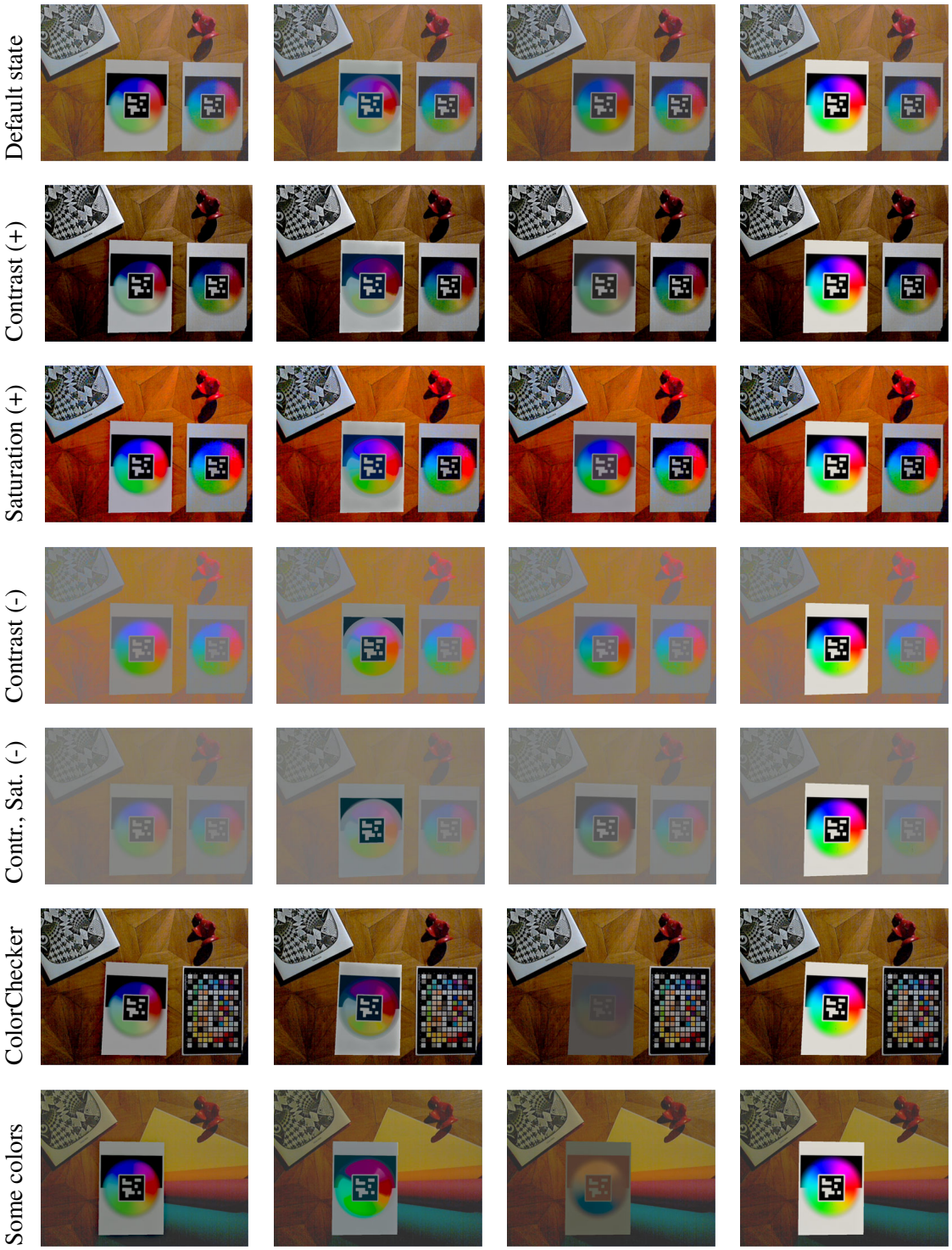


Figure 6.1: *Comparison of different algorithms with different settings and environments.*

CHAPTER 7

Summary

7.1 Conclusion

Existing methods, known from computational photography, for transferring the color distribution from one image to another have been combined with differential rendering to a novel approach, usable in the field of mixed-reality applications. Two color mapping techniques have been presented, which dynamically adapt in each rendered frame to the internal changes of the camera settings. By using this method, colors of virtual objects are closely related to the colors of the camera image, which results in a better immersion of virtual impressions in a real scene.

It has been shown that the presented 'color transfer' mapping algorithm is superior to existing approaches. Furthermore by combining this color mapping with the simulation of camera artifacts [6] (like distortion and blur) a high quality illusion could be created, resulting in virtual objects, which may be undistinguishable from real ones.

Bibliography

[1] Paul Bourke. Histogram matching. http://paulbourke.net/texture_colour/equalisation/. Accessed: 2012-09-07.

[2] Philippe Colantoni. Colorspace software. http://www.couleur.org. Accessed: 2012-09-25.

[3] Mark Grundland, Neil Dodgson, Reiner Eschbach, and Gabriel Marcu. Color histogram specification by histogram warping. *Color Imaging X: Processing, Hardcopy, and Applications*, 5667(1):610–621, 2005.

[4] Henry R. Kang. *Computational Color Technology*. SPIE Press, 2006.

[5] Jae Hyup Kim, Do Kyung Shin, and Young Shik Moon. Color transfer in images based on separation of chromatic and achromatic colors, 2009.

[6] Georg Klein and David Murray. Compositing for small cameras. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '08, pages 57–60, Washington, DC, USA, 2008. IEEE Computer Society.

[7] Martin Knecht, Christoph Traxler, Oliver Mattausch, Werner Purgathofer, and Michael Wimmer. Differential instant radiosity for mixed reality. In *Proceedings of the 2010 IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2010)*, pages 99–107, October 2010.

[8] Martin Knecht, Christoph Traxler, Werner Purgathofer, and Michael Wimmer. Adaptive camera-based color mapping for mixed-reality applications. In *Proceedings of the 2011 IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2011)*, pages 165–168. IEEE/IET Electronic Library (IEL), IEEE-Wiley eBooks Library, VDE VERLAG Conference Proceedings, October 2011. E-ISBN: 978-1-4577-2184-7.

[9] Laszlo Neumann and Attila Neumann. Color style transfer techniques using hue, lightness and saturation histogram matching. In *Computational Aesthetics in Graphics, Visualization and Imaging 2005*, pages 111–122, 5 2005.

[10] François Pitié, Anil C. Kokaram, and Rozenn Dahyot. Automated colour grading using colour distribution transfer. *Computer Vision and Image Understanding*, 107(1–2):123 – 137, 2007.

[11] Tania Pouli and Erik Reinhard. Progressive color transfer for images of arbitrary dynamic range. *Computers & Graphics*, 35(1):67 – 80, 2011.

[12] E Reinhard, M Adhikhmin, B Gooch, and P Shirley. Color transfer between images. *Computer Graphics and Applications, IEEE*, 21(5):34–41, 2001.

[13] Erik Reinhard and Tania Pouli. Colour spaces for colour transfer. In *Proceedings of the Third international conference on Computational color imaging*, CCIW'11, pages 1–15, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *IN: PROC. OF SIGGRAPH'02*, pages 267–276, 2002.

[15] Daniel Ruderman, Thomas Cronin, and Chuan-Chin Chiao. Statistics of cone responses to natural images: implications for visual coding. *J. Opt. Soc. Am. A*, 15(8):2036–2045, 1998.

[16] Yao Xiang, Beiji Zou, and Hong Li. Selective color transfer with multi-source images. *Pattern Recognition Letters*, 30(7):682 – 689, 2009.

[17] X Xiao and L Ma. Gradient-preserving color transfer. *Computer Graphics Forum*, 28(7):1879–1886, October 2009.

[18] Xuezhong Xiao and Lizhuang Ma. Color transfer in correlated color space. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, VRCIA '06, pages 305–309, New York, NY, USA, 2006. ACM.