Color Distribution Transfer for Mixed-Reality Applications

Stefan Spelitz*
Supervised by: Reinhold Preiner†

The Institute of Computer Graphics and Algorithms - Computer Graphics Group
Vienna University of Technology
Vienna / Austria

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. The aim of this work is to integrate an existing method of statistics-based color mapping into mixed-reality applications. This allows us to simulate current luminance conditions of the scene as well as changes in the camera driver settings and apply them onto virtual objects. This paper contains a fast-running approach to provide a color mapping between virtual objects and the real scene, which can be used in real-time applications. The results show that this method increases the immersion of virtual objects in a real scene.

Keywords: Mixed Reality, Augmented Reality, Color Transfer, Differential Rendering, Tone Mapping

1 Introduction

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 such that virtual objects cannot be distinguished from real, existing objects. These techniques may be used in (but are not limited to) edutainment systems, architectural and urban visualizations or for marketing reasons.

To create such an illusion, different approaches already exist. Klein and Murray [4] 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 [6] are taking direct and indirect illumination effects into account to simulate mutual lighting effects between real and virtual objects.

Although these methods create accurate results, they don't consider matching the actual colors between the virtual objects and the camera scene.

(a) *Source* **(b)** *Target* **(c)** *Result*

Figure 1: *Color matching between two images. Source image (as in the one whose colors were changing) on the left. Target image (as in the one whose colors we want to match to) in the middle. Result after using color transfer [12] in CIELab color space on the right.*

Cameras map the radiance of the real world to an image with RGB information, by using a camera-specific transfer function. Virtual objects which are merged with a camera image are easily categorized as artificial, because their colors don't match those in the scene (see Figure 5, 'Tonemapping' column for examples). The colors in the image typically depend on the global illumination conditions as well as the hue, saturation or white balance settings of the camera. With a stable color mapping function the behavior of the camera can be simulated and it is possible to adapt the colors of virtual objects to better suit the colors available in the camera image. Virtual representations of real objects registered within the system are desirable, but not necessary for the algorithm to work.

The work in this paper is based on Differential Instant Radiosity [6], which is used for the basis framework. My method attempts to improve the previous work of 'adaptive camera-based color mapping' [7] by using a global, statistics-based mapping, known as 'color transfer' [12] in the domain of mixed-reality.

This paper's main contributions are:

- An analysis of suitable color spaces in the domain of statistics-based color matching (Section 3)
- A novel approach of using 'color transfer' [12] to globally adapt the colors of virtual objects to that of a camera image (Section 6)
- A fast-running implementation of the method as GPU shader code (Section 6.1)

*stefan.spelitz@tuwien.ac.at
†preiner@cg.tuwien.ac.at

2 Related Work

This paper is based on my bachelor thesis [15] which contains additional information and an additional color mapping function. In the remainder of this section, the related work on histogram matching between two images and existing applications in augmented reality are discussed.

Statistics and Histogram Matching: 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. Based on this method Kim et al. [3] 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. The method of Reinhard et al. produces convincing results, but 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 and a post-processing step to preserve the gradients of the source image.

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 [2]. Similarly, not depending on a fixed decorrelated color space, Xiao and Ma [16] 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 [8] 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.

Color Matching in Augmented Reality: Knecht et al. [7] proposed an algorithm 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.

In the recent work of Oskam et al. [9], they provided a color balancing technique while tracking a marker in the real scene. They use the marker to compare its colors with a virtual representation and build up correspondences. To find corresponding points they use a RANSAC-based algorithm. A radial basis function interpolation is used to propagate the correspondences to the remaining color space. The algorithm creates plausible results, but needs initial correspondences.

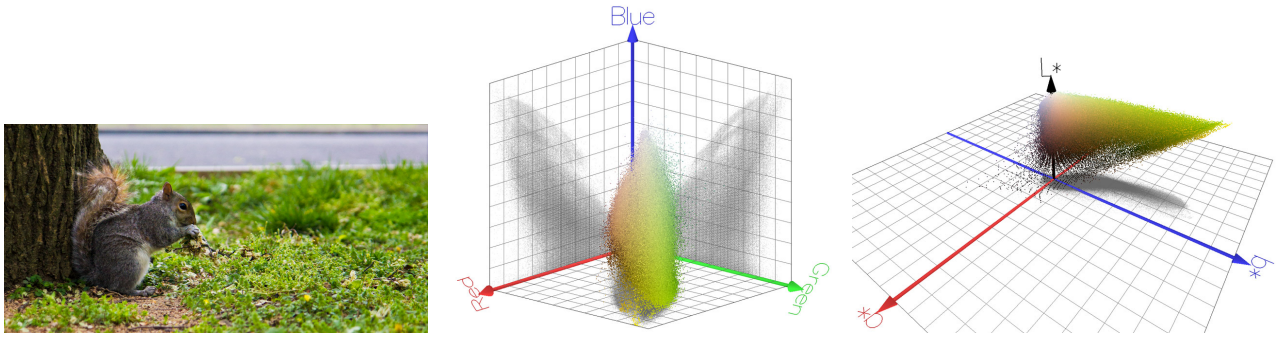
3 Color Spaces

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 algorithm presented in this paper is 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.

The RGB color space tends to have axes, which are strongly correlated. An example is displayed in Figure 2. In this example, the values are typically small in each channel for dark colors. 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. Thus, for this paper, the RGB color space will not be used to perform color mapping.

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.

Reinhard and Pouli [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



(a) *Input image* **(b)** *RGB correlation* **(c)** *CIELab correlation*

Figure 2: *Decorrelation properties of color spaces. Color distribution of image (a) is plotted in RGB space (b) and in CIELab color space (c). RGB shows an almost diagonal distribution on each pair of axes. CIELab distribution is along L* and b* axes. Plots created with ColorSpace software [1].*

environment settings (e.g. indoor, day, night). They 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.'*

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

4 Differential Rendering

Knecht et al. [6] developed a method called 'Differential Instant Radiosity' (DIR), which is the core of the framework used in this paper. They 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 from 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 for the local scene containing virtual and real objects (LS_{rv})
- One GI solution containing only the geometric representation of real objects (LS_r)

Illumination is captured using a fish eye camera to adapt the scene to environment lighting. For details on how the global illumination solutions are obtained, see [6]. 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.

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 is necessary. The chosen global tone mapping operator is based on the work of Reinhard et al. [14]. 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.

5 Application Flow

Figure 3 shows the abstract application flow 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 four stages, which will be explained next.

Stage 1: After LS_r and LS_{rv} have been tone mapped, they are converted together with the camera image 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.

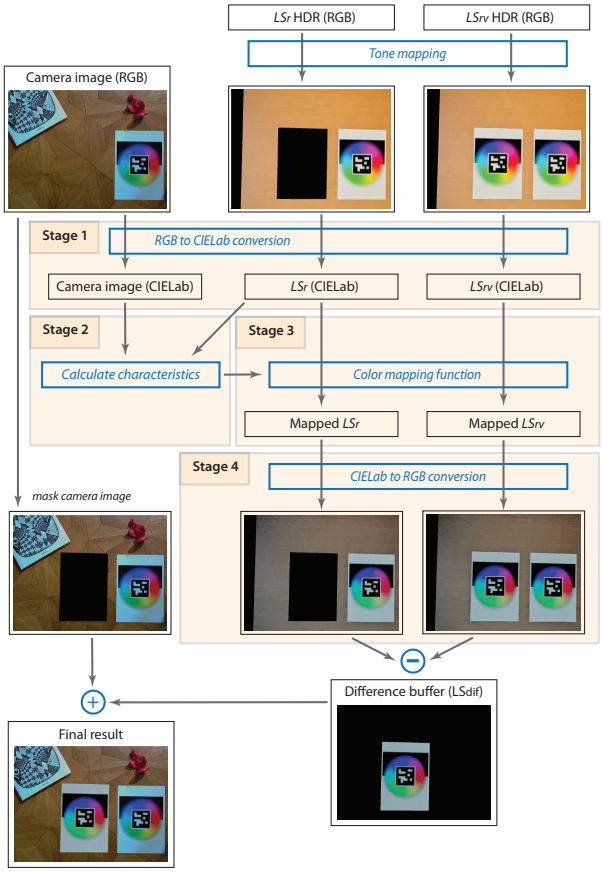


Figure 3: *The application workflow. By calculating the characteristics of the real-world in comparison to LS_r, the color mapping function is applied to the LS_r and LS_{rv} solutions. The color mapping operates in the decorrelated CIELab color space. The difference between the buffers is then merged with the camera image to create the final result.*

Stage 2: For the actual color mapping function we need to calculate the differences between the representation of the real-world (i.e. LS_r) and the actual real-world itself (i.e. the camera image). By determining the characteristics of these two images, the resulting values can be used by the 'color mapping function' (CMF).

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. This will be explained in detail in Section 6.

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 then 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.

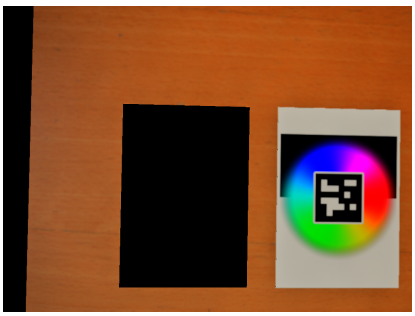


Figure 4: *This figure shows the LS_r solution with two virtual representations of real objects (table, color-chart). The black area on the left indicates the end of the virtual table. The black area in the center is the place for the virtual object to be rendered.*

6 Color Transfer

The color mapping function is based on the work of Reinhard et al. [12]. Although its primary purpose is to transfer the colors between two images (see Figure 1), in this paper three images will be involved.

The first step is to calculate the color characteristics of the source (i.e. LS_r) and the target (i.e. CI) images. The characteristics are the mean and the standard deviation of the respective color distributions. Denoted by μ_s, μ_t and σ_s, σ_t. The next step is to convert each data point (x_i) of the LS_r and LS_{rv} solutions:

$$x_i^* = (x_i - \mu_s)\frac{\sigma_t}{\sigma_s} + \mu_t$$

So we move the data points by the source mean, scale them by using the standard deviations and move them again by adding the target mean. Please note, that the same transformation is applied to LS_{rv} and to LS_r. Therefore colors which are the same in both solutions will remain equal after the mapping. This is an important feature, necessary for differential rendering.

6.1 Color Characteristics on the GPU

One aspect when using the color transfer method is to calculate the color characteristics in an efficient and fast way. We will concentrate on doing that in the following section with respect to GPU shader considerations.

Calculating the mean: The arithmetic mean is defined as:

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

where x_i is a data point of an image with dimensions $[w,h]$. With n being the number of data points, which is $n = w \cdot h$.

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 (see Figure 4). These black areas must not have any influence on the calculated mean value.

The count of pixels in the black areas is denoted as n_{zero}. The mean (μ), obtained by the mipmapping operation, can be modified to exclude the black areas by using:

$$\mu_{correct} = \frac{\mu \cdot n}{n - n_{zero}}$$

Note 1: This works only because the data points we want to exclude from the mean calculation have a value of zero.

Note 2: 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 (and $n_{zero} = n$), there won't be any information available from the camera image 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:

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

The variance can easily be calculated. 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 shader pass. 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.

When calculating the variance a similar problem occurs as when calculating the mean for LS_r. Because some data points (count is n_{zero}) shall be excluded from the calculation, we need to correct the variance calculation. Excluded data points have values of zero, so we can rewrite the variance Eq. 1 to be:

$$var = \frac{1}{n} \cdot (n_{zero} \cdot (0 - \mu)^2 + \sum_{i \in R} (x_i - \mu)^2) \quad (2)$$

with R being the remaining set of data points and having $|R| = n - n_{zero}$. The corrected variance only contains the remaining data points and is therefore, according to Eq. 1:

$$var_{correct} = \frac{1}{|R|} \cdot \sum_{i \in R} (x_i - \mu)^2 \quad (3)$$

Which is equal to (using Eq. 2):

$$var_{correct} = \frac{1}{|R|} \cdot (var \cdot n - n_{zero} \cdot \mu^2) \quad (4)$$

This shows that it is possible to exclude zero valued data points by using the mipmap-calculated variance and applying Eq. 4 to get the corrected variance.

7 Results

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. The operating system was a Microsoft Windows 7, 64 bit. The framework was developed in C# using the DirectX 10 API in conjunction with the SlimDX library. The used shader language was HLSL.

To see the capabilities of the 'color transfer' method, a test setup was created and evaluated under different contrast and saturation settings in the camera driver.

As seen in Figure 5(a), 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.

The goal is to render a virtual color chart object (placed to the 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 without losing its overall color appearance.

Broadly speaking there are three different environments, in which the methods operate. These are

- the 'default state' without tweaks of the camera driver settings (Figure 5(a))
- the scene with changed camera driver settings (i.e. contrast, saturation) (Figure 5(b)-(e))
- the scene with obstacles occluding the color chart (Figure 5(f)-(g))

By occluding the real-existing color chart with obstacles it is impossible to find a direct mapping between the colors of the virtual and the real color chart. The color mapping is still expected to deliver good results even if there is no real object for a virtual representation available in the scene.

7.1 Comparison

The following methods have been compared:

- Color transfer (see Section 6)
- Adaptive camera-based color mapping by Knecht et al. [7]
- 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. In Figure 5 it is used as a comparison of how the virtual object would look like without any color adaptation.

The method of Knecht et al. is based on a heuristic which creates color sample pairs. These pairs are used to define a color mapping function. The heuristic works well as long as there is a virtual and a real representation of the same object in the scene (Figure 5(a)-(e)). It fails to find a suitable color mapping with the real-existing color checker board (Gretagmacbeth - ColorChecker Digital SG) occluding the color chart (Figure 5(f)). In Figure 5(g) with only some real-existing colored paper spread out, the method nearly completely adapts to the existing colors in the scene, which typically is not the desired result. 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 (Figure 5(b)-(e)), but some color intensity is lost in the yellow, green and cyan areas of the virtual object. Especially when using a high contrast level (Figure 5(b)) or the color checker board (Figure 5(f)). In the scene with only some colored paper spread out (Figure 5(g)), the method does not adapt to the new colored environment but only attempts to darken the colors, when compared to the 'tonemapping' 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 better choice for color mapping in mixed-reality applications.

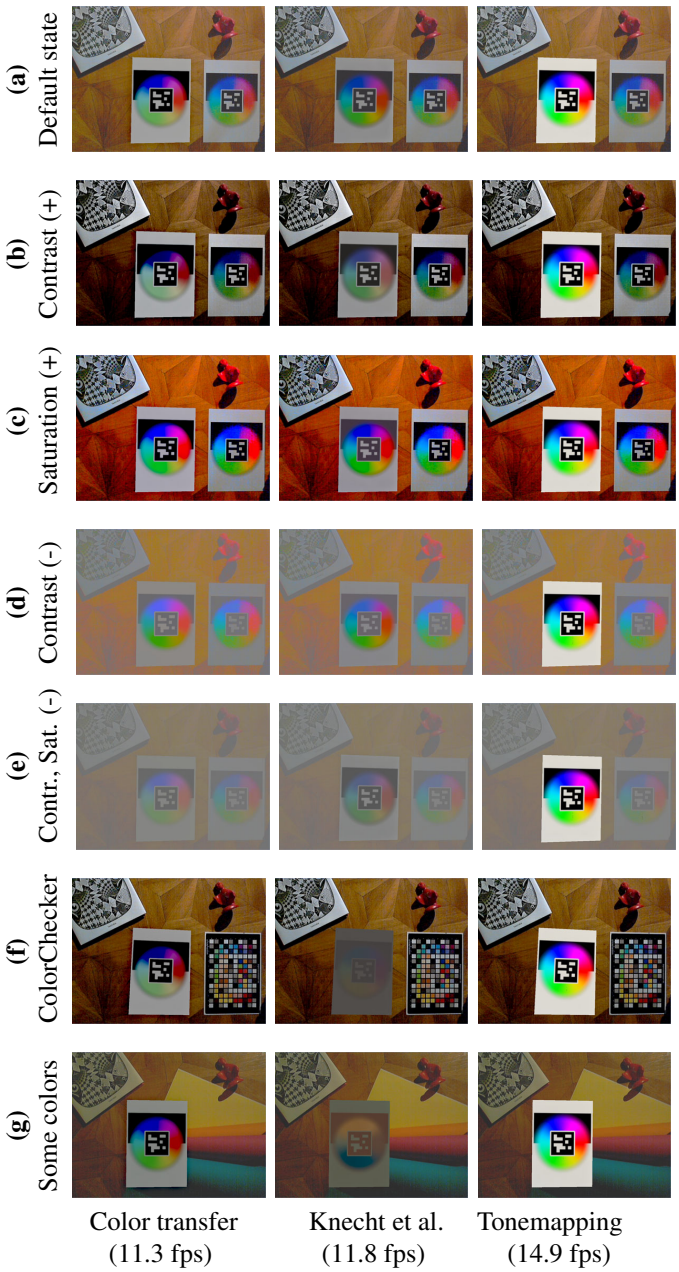


Figure 5: *Comparison of different algorithms with different settings and environments.*

7.2 User Study

The 'color transfer' method, presented in this paper, has been evaluated by Knecht [5] as part of a web survey. In this evaluation the participants were able to choose between two mixed-reality scenes in each trial. The task was to choose the image where the virtual objects fit the scene subjectively better than in the other image. Overall there were 65 trials available to the users. The survey tested multiple combinations of different rendering modes (global illumination, color mapping, camera artifacts).

It showed that the 'color transfer' overall has a negative impact on perceived quality. As a possible explanation, Knecht concluded that the color mapping changes the saturation levels of the virtual objects in a way noticeable to the user. All test images were used with standard camera settings. Therefore the possible strength of the algorithm of adapting to highly changed camera settings (e.g. saturation, contrast) was not tested.

8 Conclusions, Limitations and Future Work

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. A color mapping technique has been presented, which dynamically adapts 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 [4] (like distortion and blur) a high quality illusion could be created, resulting in virtual objects, which may be undistinguishable from real ones.

8.1 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 are 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 is the color mapping function. This function is working with statistical data of the whole scene and tries to adapt colors of virtual objects to the color average 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.

As with other statistics and histogram matching methods, the 'color transfer' algorithm is good for automatic mapping. On the other hand it does not provide direct control over the color mapping process and thus may not be suited for any situation.

8.2 Future Work

The user study of Knecht [5] showed that the overall impression of color-mapped virtual objects is worse than without color mapping when using standard camera settings. Knecht also stated that a separate study should be performed, investigating the impact of changed camera settings on the perceived quality of color-mapped mixed-reality scenes. Nonetheless, the survey showed that there is the need for improvement for the default case, when the standard camera settings are used.

References

[1] Philippe Colantoni. Colorspace software. http://www.couleur.org. Accessed: 2014-03-15.

[2] 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.

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

[4] 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.

[5] Martin Knecht. *Reciprocal Shading for Mixed Reality*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, December 2013.

[6] 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.

[7] 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.

[8] 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.

[9] Thomas Oskam, Alexander Hornung, Robert W. Sumner, and Markus Gross. Fast and stable color balancing for images and augmented reality. In *3D Imaging, Modeling, Processing, Visualization*

and Transmission (3DIMPVT), 2012 Second International Conference on, pages 49–56, Oct 2012.

[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] Erik Reinhard, Michael Adhikhmin, Bruce Gooch, and Peter 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] Stefan Spelitz. Color distribution transfer for mixed-reality applications. Bachelor thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, October 2012.

[16] 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.

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