Real-time color blending of rendered and captured video

Erik Reinhard, Ahmet Oguz Akyuz,
Mark Colbert, Charles E Hughes
School of Computer Science
University of Central Florida
Orlando FL
reinhard@cs.ucf.edu

Matthew O'Connor

Institute for Simulation and Training
University of Central Florida
Orlando FL

ABSTRACT

Augmented reality involves mixing captured video with rendered elements in real-time. For augmented reality to be effective in training and simulation applications, the computer generated components need to blend in well with the captured video. Straightforward compositing is not sufficient, since the chromatic content of video and rendered data may be very different such that it is immediately obvious which parts of the composited image were rendered and which were captured.

We propose a simple and effective method to color-correct the computer generated imagery. The method relies on the computation of simple statistics such as mean and variance, but does so in an appropriately chosen color space - which is key to the effectiveness of our approach. By shifting and scaling the pixel data in the rendered stream to take on the mean and variance of the captured video stream, the rendered elements blend in very well.

Our implementation currently reads, color-corrects and composites video and rendered streams at a rate of more than 22 frames per second for a 720x480 pixel format. Without color correction, our implementation generates around 30 frames per second, indicating that our approach comes at a reasonably small computational cost.

ABOUT THE AUTHORS

Erik Reinhard is assistant professor at the University of Central Florida and has an interest in the fields of visual perception and parallel graphics. He has a B.Sc. and a TWAIO diploma in computer science from Delft University of Technology and a Ph.D. in computer science from the University of Bristol. He was a post-doctoral researcher at the University of Utah. He is founder and co-editor-in-chief of ACM Transactions on Applied Perception and is currently writing a book on High Dynamic Range Imaging.

Ahmet Oguz Akyuz is a Ph.D. student at the University of Central Florida. He studied computer engineering and in 2003 he received his B.Sc. from the Middle East Technical University in Ankara, Turkey. His current interests are in computer graphics, color appearance modeling and high dynamic range imaging.

Mark Colbert is currently a senior undergraduate student at the University of Central Florida majoring in Computer Science. He currently works as an undergraduate research assistant in UCF's Graphics Laboratory. In addition to his academic studies, Mark has worked in the computer animation industry for over 3 years designing graphics for clientele such as the Tampa Bay Buccaneers and the Anaheim Angels.

Matthew O'Connor is the Programming Team Leader for the Media Convergence Laboratory at the Institute for Simulation and Training. He received his B.S. in Computer Science from the University of Central Florida in 2002, and is currently pursuing his Ph.D. in Computer Science. His research interests include mixed reality, interacting agents, and distributed computing.

Charles E. Hughes is Professor of Computer Science in the School of Computer Science at the University of Central Florida. After receiving his PhD in computer Science in 1970, he served on the faculties of Computer Science at Penn State University and the University of Tennessee prior to joining UCF in 1980. His research interests are in mixed reality, distributed interactive simulation and models of distributed computation. He has authored or co-authored over 100 refereed research papers and six books. He is a senior member of the IEEE, and a member of the ACM and the IEEE Computer Society.

Real-time color blending of rendered and captured video

Erik Reinhard, Ahmet Oguz Akyuz, Mark Colbert, Charles E Hughes
School of Computer Science
University of Central Florida
Orlando FL
reinhard@cs.ucf.edu

Matthew O'Connor
Institute for Simulation and Training
University of Central Florida
Orlando FL

INTRODUCTION

Modern simulation and training applications frequently employ a mixture of computer generated content and live video. Thus multiple streams of data need to be composited in real-time before the result may be presented to the viewer. This is easily achieved with the alpha blending operations available on current graphics hardware. For each frame, parts of the captured video are replaced by rendered data. This approach works well, but ignores several issues which may hamper the effectiveness of training and simulation applications.

In particular, the rendered data may have a visually different appearance from the captured video. Consider a computer generated vehicle being superimposed over filmed terrain. This vehicle will only blend in well with the background if the modeling of the vehicle is carefully done, the colors of the vehicle are carefully chosen, and the lighting conditions under which the vehicle is rendered matches the lighting conditions of the terrain. The latter requirement is particularly difficult to maintain.

We present a fast and automatic image analysis and correction technique which allows the rendered content to be post-processed such that it blends in well with the video captured data. The work is directly applicable to simulation and training applications as outlined above, as well as several other fields.

For instance in the film industry, a common task performed by colorists is ensuring that scenes that follow one another appear without unnatural transitions. Since scenes may be shot with different cameras, settings and lighting, frequently this involves chromatic adjustment of film. During post-processing colorists also perform a creative task, namely instilling a particular atmosphere or dramatic effect by adjusting film color.

Similar tasks occur in digital matting applications where separate video streams are composited to form a new scene with each frame containing pixels from multiple sources. Here, usually a foreground is filmed against a blue screen, and the blue is later replaced with a separately filmed background. The filming of foreground and background are usually separated both in time and location.

In augmented reality, video from different sources is also composited. However, here one of the sources is usually computer generated in real time, and live video is the other source. To ensure that the computer generated video blends in well with the live video, it would normally be necessary to carefully match the light sources found in the real scene with the light sources used for rendering the computer generated video stream. In addition the appearance of materials and geometry in the rendered scene (including the direction of shadows), as well as the rendered content should be matched for a believable blend of real-time rendered data and live video.

Matching high level features such as the position of light sources and other geometry is a complex task which requires considerable computational complexity in terms of analysis of the live video. While such an analysis and subsequent matching of 3D geometry with live video is a worthwhile goal which may considerably enhance the visual experience of the presented material, we address the arguably simpler problem of matching the chromatic content in a hands-off manner. Our solution therefore has applications in augmented reality, as well as more general video matting applications, and could be extended to automatically match subsequent scenes in film.

Our work extends a previous idea which allows the colors of a single photograph to be semi-automatically matched with the colors of another photograph (Reinhard et al., 2001). We introduce two set-and-forget user parameters which allow control over the amount of blending. Whenever the environment which is filmed changes considerably, or if the rendered data changes, these parameters may need to be readjusted. However, under normal circumstances the method is robust and parameter adjustment is rarely necessary.

By following the keep-it-simple principle the number of computations our algorithm requires per frame is kept low. Further speed-up is afforded by the fact that the analysis of each frame does not necessarily involve all pixels in both video and rendered streams. The chromatic adjustment may be merged with the compositing step to minimize unnecessary memory accesses. This leads to a simple, effective and highly efficient algorithm which runs at interactive rates.

PREVIOUS WORK

While manipulation of color content of images takes many different forms, we are interested in changing an image by means of analysis of a second image. To date only three such algorithms are known. The first is called Image Analogies and allows many different characteristics of an image to be imposed upon a second image, including such attributes as color and brush-strokes (Hertzmann et al., 2001). This approach is powerful, but perhaps too computationally expensive for real-time operation.

The second algorithm is statistical in nature and is called Color Transfer (Reinhard et al., 2001). Here, a decorrelated color space is used to analyze both example and target images - the statistical measures being mean and variance. The pixels in the target image are shifted and scaled so that this image has the same mean and variance as the example image. This method is straightforward to implement and computationally efficient. It was later extended to allow colorization of grey-scale images (Welsh et al., 2002).

A weaknesses of this method is that the composition of example and target images needs to be similar to obtain plausible results. An ad-hoc solution is to allow the user to specify pairs of swatches which will then guide the color transfer process. While this extends the use of the method, manual intervention is not desirable for real-time applications.

The third algorithm improves the color transfer method considerably by considering color names (Chang et al., 2004). Colors can be classified according to name, which is remarkably consistent across languages and cultures. Rather than employing swatches, each pixel in both example and target images is categorized into one of eleven color groups. The color adjustment is then applied such that colors stay within their category. This produces very plausible results, albeit at the cost of extra computations. If computation time is not at a premium, we would advocate the use of this color naming scheme.

However, in this paper we are interested in real-time color transfer to allow computer generated data to be blended with video. We therefore opt to use the original color transfer method. We apply this algorithm in essentially its basic form, but provide simple extensions to overcome the need for using manually selected pairs of swatches. We also present experiments which show how many pixels per frame are required to compute statistics - thus allowing performance to be optimized without introducing artifacts such as flicker between frames. Finally, we employ a slightly different color space which further reduces the number of computations.

COLOR TRANSFER

In RGB color space, the values of one channel are almost completely correlated with the values in the other two channels. This means that if a large value for the red channel is found, the probability of also finding large values in the green and blue channels is high. Color transfer between images in RGB color space would therefore constitute a complex 3-dimensional problem with no obvious solution.

To understand the color opponent space employed by the human visual system, Ruderman et al converted a set of spectral images of natural scenes to LMS cone space and applied Principle Components Analysis (PCA) to this ensemble (Ruderman et al., 1998). The LMS color space roughly corresponds to the peak sensitivities of the three cone types found in the human retina. The letters stand for Long, Medium and Short wavelengths and may be thought of as red, green and blue color axes.

The effect of applying PCA to an ensemble of images, is that the axes are rotated such that the first channel represents luminance, and the other channels carry red-green and yellow-blue color opponent information. Color opponent channels have a different interpretation than channels in more familiar color spaces. For example, a value specified for red in RGB determines how much red is required to mix a given color. The values usually range from 0, i.e. no red, to 255 indicating maximum red.

In a color opponent channel, values can be both positive and negative. For instance, in the yellow-blue color opponent channel a large positive value indicates ’very yellow’, whereas a large negative value indicates ’very blue’. Values around zero indicate the absence of yellow and blue, i.e. a neutral grey. The same is true for the red-green channel. A consequence of this representation that it is impossible to represent a color that is blue with a tinge of yellow. This is in agreement with human visual experience — humans are incapable of seeing yellowish blue or greenish red. On the other hand color opponent spaces predict that we should be able to see bluish red (purple) or greenish yellow.

Since the PCA algorithm decorrelates the data through rotation, it is evident that one of the first steps of the human visual system is to decorrelate the data. Ruderman et al found that while theoretically the data is only decorrelated, in practice the data becomes nearly independent. In addition, by taking the logarithm the data becomes symmetric and well-conditioned. The resulting color opponent space is termed $L\alpha\beta$.

The implications for color transfer are that our complex 3-dimensional problem may be broken into three 1-dimensional problems with simple solutions (Reinhard et al., 2001). The color transfer algorithm thus converts example and target images to LMS cone space. Then the logarithm is taken, followed by a rotation to color opponent space. Here, the mean and variance of all pixels in both images are computed. This results in three means and three variances for each of the images. The target image is then shifted and scaled so that in color opponent space the data along each of the three axes has the same mean and variance as the example image. The result is then converted back to RGB space.

ALGORITHM

For real-time operation, we would like to apply the color transfer algorithm on a frame-by-frame basis. For this approach to be successful, it is paramount that the number of computations executed for each frame be minimized.

To reduce the number of computations, we are interested to know if executing the color transfer algorithm in logarithmic space is absolutely necessary. Our results show that this is not the case. We can therefore omit taking the logarithm for each pixel.

Since the conversion from RGB color space to LMS cone space is achieved by multiplying each RGB-triplet by a 3x3 matrix, and the conversion between LMS and $L\alpha\beta$ color opponent space is also achieved by matrix multiplication, these two color space conversions may be merged into a single matrix multiplication which needs to be executed for each pixel in the source and target frames. This optimization saves 3 logarithms and a 3x3 matrix multiplication for each pixel in both images.

By omitting the logarithm, we effectively no longer use $L\alpha\beta$ space, but convert to an approximation of the CIE Lab color space instead (Wyszecki and Stiles, 1982):

$$\begin{bmatrix} L \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0.3475 & 0.8231 & 0.5559 \\ 0.2162 & 0.4316 & -0.6411 \\ 0.1304 & -0.1033 & -0.0269 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

The inverse transform to go from Lab to RGB is given here for convenience:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 0.5773 & 0.2621 & 5.6947 \\ 0.5774 & 0.6072 & -2.5444 \\ 0.5832 & -1.0627 & 0.2073 \end{bmatrix} \begin{bmatrix} L \\ \alpha \\ \beta \end{bmatrix} \quad (2)$$

Within $L\alpha\beta$ space we compute the mean and standard deviations in each of the L, α and β channels for both the rendered image and the captured frame. A further opportunity to reduce the number of computations may be exploited by computing these statistics on only a subset of all pixels. We may for instance choose to compute these statistics on only every tenth pixel, thus reducing the number of computations tenfold for this step.

For the rendered data we also omit any pixels which are transparent since we will substitute video data for these pixels during the compositing step.

As in the original color transfer algorithm. the means and variances are used to adjust the pixels of the rendered frame. However, we have found that if we shift and scale the pixel data such that the means and variances are precisely the same as the mean of the video capture, the results are too dramatic. For instance, if the video depicts a red brick wall, adjusting the rendered data to be displayed in front of the wall to have the same means and variances will result in any data to become fully brick-colored. Rather than rely on manually specified swatches to combat this problem, we introduce two user parameters to allow the adjustment to be incomplete. These parameters do not usually need to be adjusted for each frame, but are of the set-and-forget variety.

The adjustment of rendered pixels according to the means and variances of the video data as well as the newly introduced user parameters is then given by:

$$\begin{aligned} L_{\text{target}} &= s_L \left(L_{\text{target}} - \bar{L}_{\text{target}} \right) + \bar{L}_{\text{target}} + d_L & (3) \\ \alpha_{\text{target}} &= s_\alpha \left(\alpha_{\text{target}} - \bar{\alpha}_{\text{target}} \right) + \bar{\alpha}_{\text{target}} + d_\alpha & (4) \\ \beta_{\text{target}} &= s_\beta \left(\beta_{\text{target}} - \bar{\beta}_{\text{target}} \right) + \bar{\beta}_{\text{target}} + d_\beta & (5) \end{aligned}$$

In these equations, $\bar{L}_{\text{target}}$ is the mean luminance of the rendered frame and s_L and d_L are two constants that are computed once for every frame. The α and β channels are computed similarly.

The scale factors s_L, s_α and s_β adjust each pixel such that the variance of the rendered frame becomes equal to the variance of the captured video (the example frame) and includes user parameter s which allows the adjustment to be incomplete:

$$s_L = 1 - s + s\,\frac{\sigma_{L,\text{example}}}{\sigma_{L,\text{target}}} \quad (6)$$

$$s_\alpha = 1 - s + s\,\frac{\sigma_{\alpha,\text{example}}}{\sigma_{\alpha,\text{target}}} \quad (7)$$

$$s_\beta = 1 - s + s\,\frac{\sigma_{\beta,\text{example}}}{\sigma_{\beta,\text{target}}} \quad (8)$$

Here, the variance observed in a particular channel for example and target images is indicated with σ. The offsets d_L, d_α and d_β steer the mean of the target image towards the mean of the example image to the extent dictated by user parameter d:

$$d_L = d\,\left(\bar{L}_{\text{example}} - \bar{L}_{\text{target}}\right) \quad (9)$$

$$d_\alpha = d\,\left(\bar{\alpha}_{\text{example}} - \bar{\alpha}_{\text{target}}\right) \quad (10)$$

$$d_\beta = d\,\left(\bar{\beta}_{\text{example}} - \bar{\beta}_{\text{target}}\right) \quad (11)$$

We combine this adjustment with the actual compositing step so that a pixel is either copied from the video capture, or it is a rendered pixel which is adjusted by the above formula. Once each pixel has either been copied or adjusted, the resulting image is converted back to RGB space for display.

VISUAL QUALITY

The visual quality depends on the number of pixels in each frame that is used in the computation of the means and variances, as well as the user parameters s and d. We demonstrate the behavior of our algorithm by means of a short sequence of captured video — several frames are shown in Figure 1 — and a rendered sequence of a rotating multi-colored sphereflake, as shown in Figure 2. All frames were created at a resolution of 720 by 480 pixels and then run-length encoded into two separate movies. All compositing and color correction is then applied to the two movies without further referring to the original separate frames, or indeed the 3D geometry.

Simply compositing the results would cause the color content of the sphereflake to stand out and not mesh very well with the background (Figure 3). Full application of our algorithm results in the frames shown in Figure 4. Here, the algorithm has been too effective. In particular, the fourth frame which shows predominantly a brick wall, has caused the sphereflake to take on too much red. By adjusting the s user parameter to a value of 0.4 we obtain the sequence of frames shown in Figure 5. We believe that this parameter setting is a reasonable default value between too much and too little adjustment.

We have also found that partial adjustment of the user parameter d did not appreciably change the result. We therefore conclude that this parameter may be removed to save a small number of computations, leaving only s as a user parameter.

With these default settings we combined three different sequences of captured video with three different sets of rendered data. From each of the resulting nine composited sequences we captured a representative frame, and show the results in uncorrected form in Figure 6 and show the corrected images in Figure 7. We have found that for our test scenes further parameter tuning was unnecessary. We always set s to 0.4 and obtained plausible results. This does not mean that this approach will always yield acceptable results, but we have not found a combination of video and rendered content that required adjustments to these parameter settings.

PERFORMANCE

A substantial performance gain was recorded by unrolling loops and reordering memory access by hand. Our sequence rendered at a frame rate of around 13 fps before our low-level code optimization, and improved to around 22 fps after this optimization. A simple compositing algorithm which does not apply any color correction, but uses the same procedures to display the resulting frames, runs under the same configuration at 30 fps. We therefore conclude that for color correcting our test sequence incurs a performance penalty of around 28%.

This optimization includes a scheme whereby the current frame is adjusted on the basis of means and variances computed from the previous frame. In addition the variances computed during the previous frame make use of means computed two frames back. This allows us to simplify and streamline the implementation such that for each frame only a single pass over all pixels is required.

The performance of our implementation depends to some extent on the number of pixels that are rendered in each frame. It is the rendered pixels that are adjusted, whereas the video pixels are only analyzed. This dependency is evidenced by plotting the frame-rate over time, as shown in Figure 8. We see that the part of the sequence around frame 30 is slower than the first and last parts. This correlates with the number of rendered pixels, which we plot in black in Figure 8. The time per frame closely follows the high

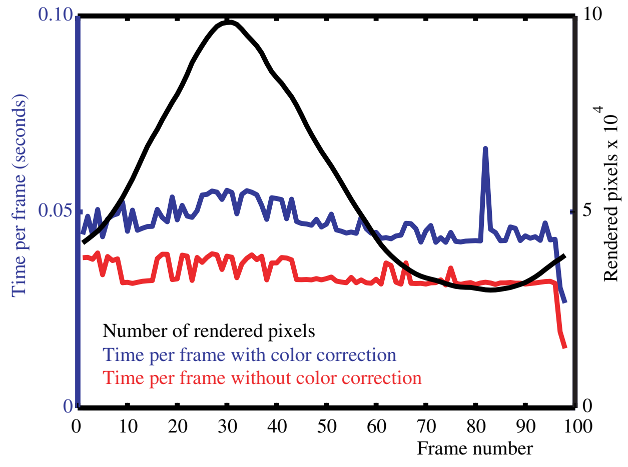

Figure 8: *Frame times and rendered pixels as function of frame-number.*

frequency fluctuations present in the version where we composite without color correction.

Our experiments with skipping frames for computing statistics, or omitting pixels from consideration did not prove to be successful. For instance, the extra if-statement necessary to determine if the current frame should be used to recompute means and variances introduces a branch in the control flow which causes the floating point units of the CPU to become partially unfilled. The introduction of this single if-statement causes the frame-rate to drop by 3 fps.

In addition, the bottleneck in our computations is the adjustment of pixels, and not the computation of statistics. Unfortunately, it is not possible to skip pixel adjustment for any rendered pixels, since this would introduce flickering. For the purpose of comparison, we omitted the computation of new statistics for each frame, while still correcting rendered pixels, albeit with pre-cooked values. The frame-rate increased by around 1.5 fps. Compared with the 3 fps that an extra if-statement would cost, the overhead of computing new averages and means in brute-force manner for every frame is relatively inexpensive.

CONCLUSIONS

A simple and effective method to glean the overall color scheme of one image, and impose it upon another may be extended for use in real-time mixed-reality applications. Low level optimizations proved to be more effective in gaining performance than high level schemes such as computing statistics for only a subset of the frames, or computing statistics on a subset of pixels within each frame. The branching introduced by such high level optimizations cause the floating point pipelines to be used less efficiently, which has a negative impact on performance. The overall effect is that performance is better with a brute-force approach than with these optimizations.

We introduced user parameters which make the transfer of colors between the rendered stream and the video stream partial, which improves the quality of the results. For hands-off approaches such as required in mixed-reality applications, such set-and-forget user parameters are more effective than the manual specification of swatches that the original method employed as a means of user control.

While we demonstrated our algorithm in terms of a mixed-reality scenario, it is equally applicable to the work of colorists. Scenes may be adjusted such that the chromatic content of subsequent scenes does not exhibit unpleasant jumps. Creative applications, whereby scenes are colored for a specific mood or artistic effect, is also catered for. We envisage that scenes may be recolored according to existing footage which already has the desired artistic effect.

ACKNOWLEDGMENTS

This work is partially supported by the Office of Naval Research and the Army's Research, Development & Engineering Command.

REFERENCES

Chang, Y., Saito, S., Uchikawa, K., and Nakajima, M. 2004. Example-based color stylization based on categorical perception. In *First Symposium on Applied Perception in Graphics and Visualization*.

Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., and Salesin, D. H. 2001. Image analogies. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 327–340.

Reinhard, E., Ashikhmin, M., Gooch, B., and Shirley, P. 2001. Color transfer between images. *IEEE Computer Graphics and Applications 21*, 34–41.

Ruderman, D. L., Cronin, T. W., and Chiao, C. 1998. Statistics of cone responses to natural images: Implications for visual coding. *J. Opt. Soc. Am. A 15*, 8, 2036–2045.

Welsh, T., Ashikhmin, M., and Mueller, K. 2002. Transferring color to greyscale images. In *Proc. of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, 277–280.

Wyszecki, G., and Stiles, W. S. 1982. Color Science: Concepts and Methods, Quantitative Data and Formulae. *John Wiley and Sons*.

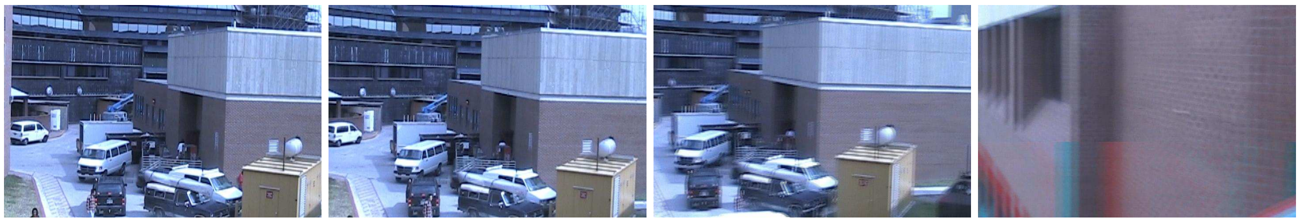

Figure 1: *Four frames from a sequence of video.*

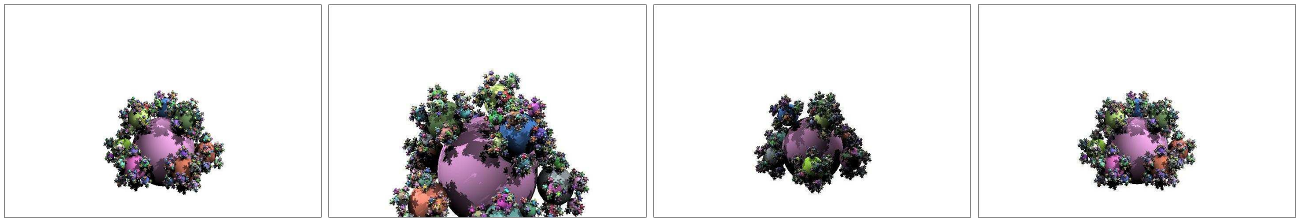

Figure 2: *Four frames from a computer generated sequence.*

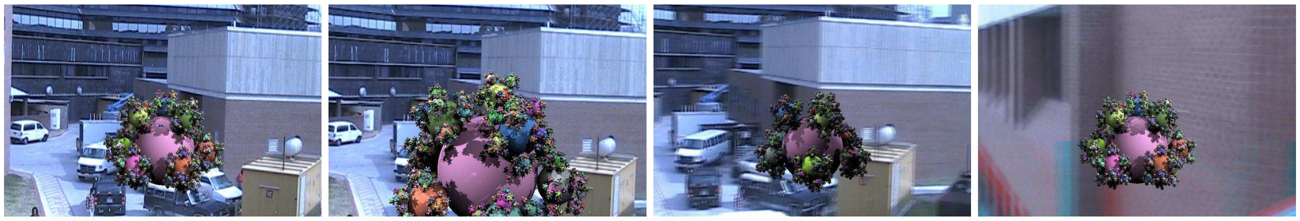

Figure 3: *Compositing without color correction.*

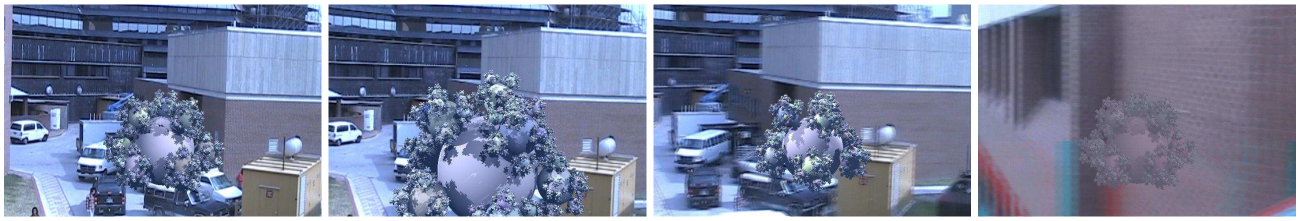

Figure 4: *Full adjustment.*

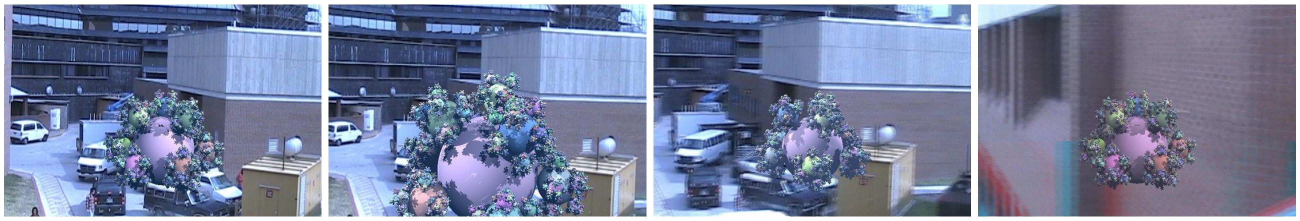

Figure 5: *Partial adjustment with a value of* $s = 0.4$.

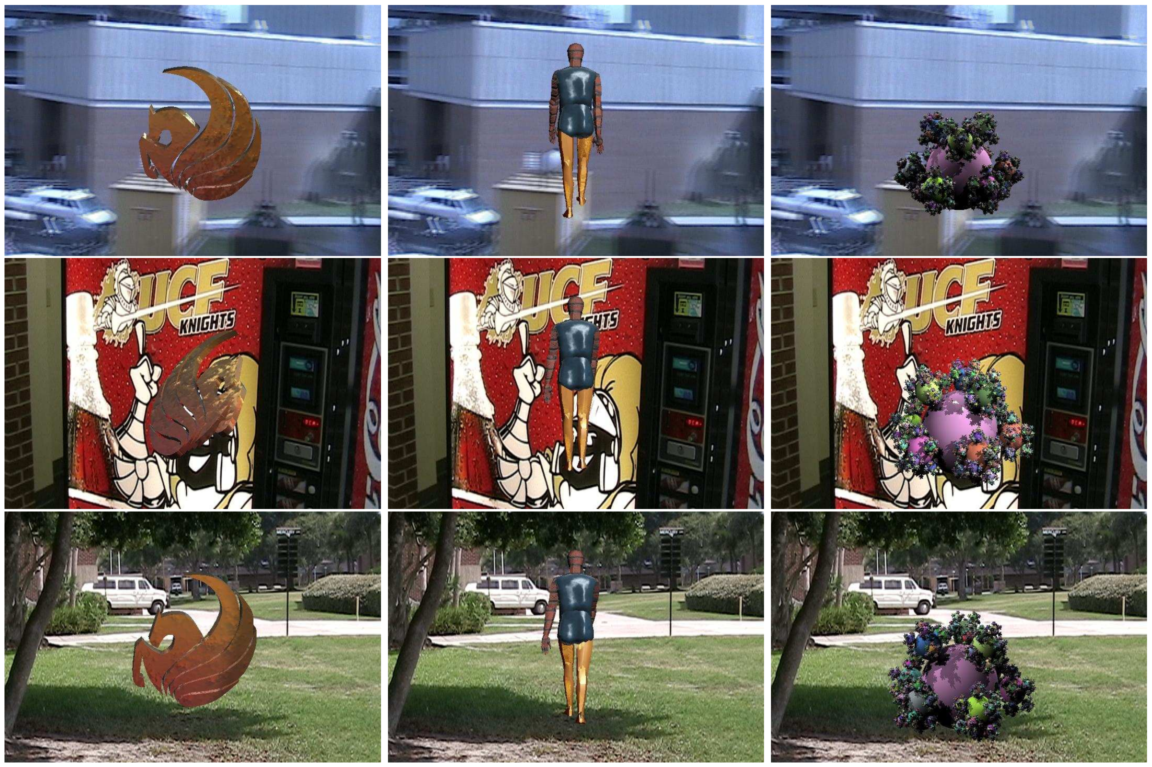

Figure 6: *Uncorrected frames.*

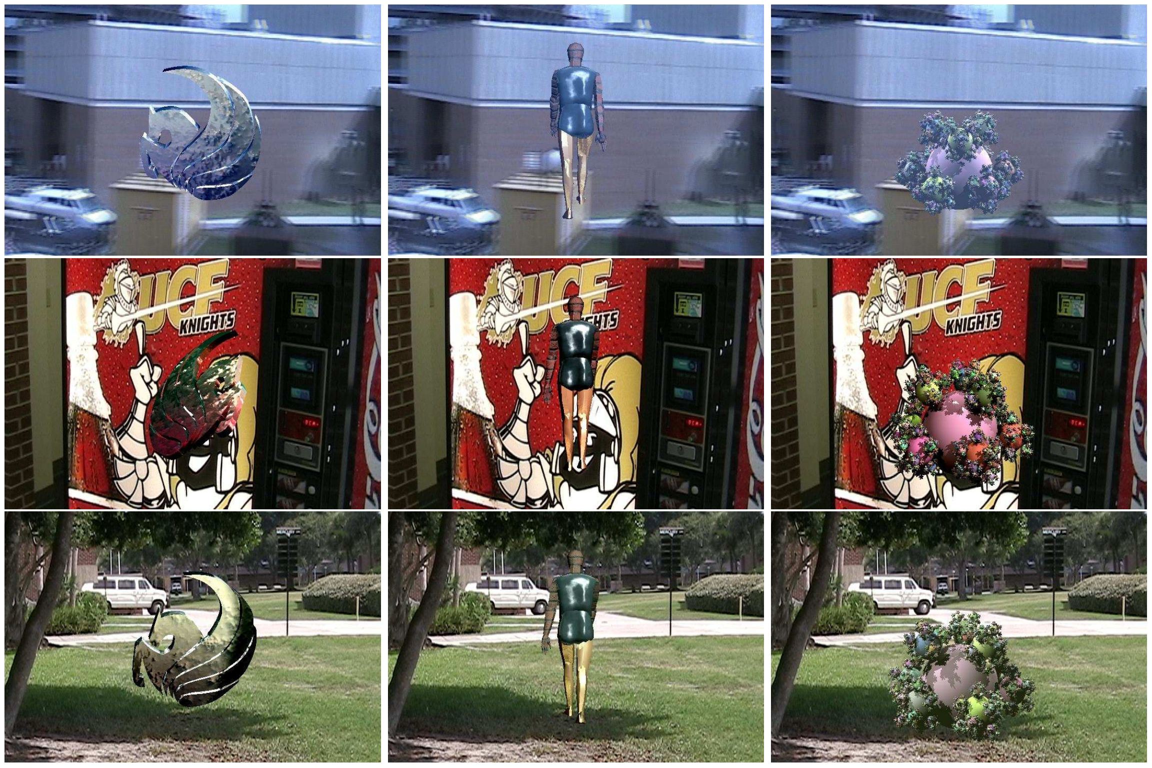

Figure 7: *Corrected frames.*