Software Chroma Keying in an Immersive Virtual Environment

F. van den Bergh[†], V. Lalioti[‡]

[†]*Department of Computer Science, University of Pretoria, South Africa,* fvdbergh@cs.up.ac.za
[‡]*Department of Computer Science, University of Pretoria, South Africa,* vlalioti@cs.up.ac.za

Abstract

This paper presents a very fast software chroma keying algorithm, requiring at most five integer operations per pixel. An improvement of more than 300% in performance over HLS based algorithms was achieved. The algorithm has been developed for use in an immersive virtual environment, which is also described here.
Keywords: *Virtual Reality, Immersive Telepresence, Chroma Keying*

1 Introduction

Compositing is a technique used when two images are combined so that some regions of the resulting image come from the first source image, while the rest comes from the second source image. A *mask*, in this context, describes which part of the first source image should remain. Several methods of generating such a mask exist; one that is often used in television productions is a technique known as *Chroma Keying*.

With chroma keying the one image is recorded in a studio where the background is a uniform color, usually blue. The actor (weatherman, newsreader, etc.), who is not wearing any blue clothing, can then be distinguished from the background based on color. For television productions a special hardware device then generates a mask corresponding to the blue part of the image, which can be used to combine the image of the actor with a different background, like a computer generated weather map.

For some situations it would be preferable to do chroma keying in software, as it eliminates the need for the special (expensive) hardware. The algorithm presented in this paper is simple, fast and produces very good quality images when compared to dedicated hardware approaches.

Next, Section 2 will give a brief overview of Immersive Telepresence and chroma keying, followed by a description of the new chroma keying algorithm in Section 3. Some results are presented in Section 4, while Section 5 describes an application where the software chroma keying has been used, followed by a discussion of future directions.

2 Background

2.1 Virtual Environments

Projective Display Systems are the state of the art in high end Virtual Environments. They release the user from the heavy load and inconvenience related to head-mounted displays, and capitalise on the increased resolution and rendering speed of the available hardware [7]. A common characteristic of the projection-based VE systems is that they all extend the real space by a virtual space providing a common world co-ordinate system, where the local and the remote participants are part of. Projection-based VR installations allow collaboration between small groups on the same or in different locations, without the need for avatar representation through the use of real-time video of remote participants for Immersive Telepresence.

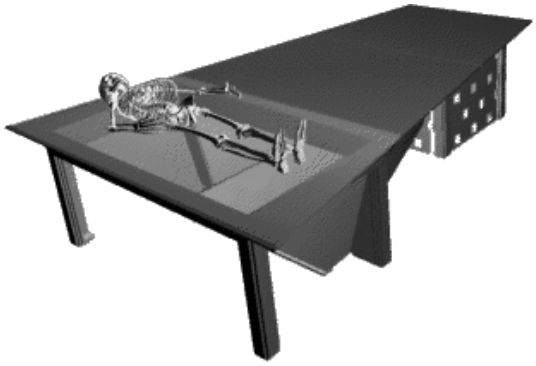

Figure 1: A model of the Responsive Workbench

The mapping of virtual space to real space allows us to characterize projection-based virtual environments as desk or room size installations [3]. Currently desk and room size installations include the Responsive Workbench[1], the Collaborative Responsive Workbench, the CyberStage or the Teleport [1]. The RWB was developed in 1993 at GMD and uses the metaphor of a working table in order to present stereoscopic 3-dimensional virtual data to one or more users and allow a more natural way of interaction [8] [9] Figure 1. The Collaborative Responsive Workbench extends the Responsive Workbench by one vertical screen, thus enlarging the viewing frustum Figure 2. Each of the displays is 1.80x1.20m and the viewing frustum is around 2m by 3m. In this cube the virtual 3d objects can

[1]RWB[TM] is a registered Trademark of the German National Research Center for Information Technology

Figure 2: Collaborative Responsive Workbench

be directly manipulated via a stylus or a data glove or other input devices such as the Cube. The Collaborative Responsive Workbench, apart from a larger viewing frustum, provides a better metaphor for remote collaboration, namely that of working around a table.

CyberStage is a CAVE[2] [2] like four-sided room-size stereo display system installed at GMD, which creates the illusion of immersion within a computer-generated virtual environment. Users see large virtual spaces and hear spatially distributed sound. Projection systems like CyberStage allow direct and body-centered human interaction within virtual worlds as well as teamwork. Three wall size rear projection systems are installed orthogonal to the floor projection, each with a size of 3x3 meters. An SGI 4 pipe Onyx 2 Infinite Reality generates eight user-controlled images. The user position is tracked with Polhemus Fastrak sensors. Crystal Eyes shutter glasses are used for stereo image perception. The display resolution is 1024 x 768 pixels at 120 Hz for each of the four displays. The eight channel-surround-sound system is fed by IRCAM's room acoustic software Spatilisateur and provides support for localised sound sources within the virtual environment [4] [12]. Another innovation of the CyberStage is the acoustic floor, which allows for the generation of a sense of vibrations.

A first approach in Immersive Telepresence was making use of the Collaborative Responsive Workbench and CyberStage to provide a virtual space where remote and local participants can meet and collaborate as if face-to-face. Immersive Telepresence uses live stereo-video, hardware chroma keying and texture mapping in order to integrate the live video into any virtual environment projected in these installations [11]. In addition, an approach for adjusting the left and right images generated from a static stereo-camera has been implemented for accommodating moving of participants within such an installation [10].

Chroma-keying is a technique commonly used in virtual studio productions. Actors are located in a uniformly lit blue room. The hardware device, namely the chroma-keyer, is then used to separate the image of the actor from the blue background. This is done by adding an alpha-channel to the original video frames. The video stream is then merged in real-time with the virtual environment. The camera parameters are tracked and the virtual environment is rendered in real-time according to the perspective of the real camera, thus creating a very realistic impression of the actors being immersed in the virtual environment and interacting with the virtual objects as if there were real [6]. While hardware chroma keying is of the high quality required for virtual studio productions, its high cost prohibits its use for Immersive Telepresence. Additionally, the chroma keying requirements of Immersive Telepresence are not as strict as the ones of virtual studios. Therefore the approach presented in this paper is using software to achieve chroma keying for the purpose of remote participant extraction from a fixed blue background in real-time.

[2]CAVE™ is a registered Trademark of the University of Illinois

2.2 Chroma Keying: Exact Solutions

A thorough mathematical approach to the problem of chroma keying is presented by Smith and Blinn [14]. They define what they call the "Matting Problem" (Matting is roughly the film industry equivalent for chroma keying), which is then used to illustrate their new method. Thus, the matting problem, according to Smith and Blinn:

> Given C_f and C_b at corresponding points, and C_k a known backing color, and assuming $C_f = C_o + (1-\alpha_0)C_k$, determine C_o which then gives the composite color $C = C_o + (1-\alpha_0)C_b$ at the corresponding point, for all points that C_f and C_b share in common.

Note that the C_i are vectors of the form $C_i = [\, R_i\, G_i\, B_i\, \alpha_i \,]$, where the R_i, G_i and B_i components are *premultiplied* by the α_i value, and thus fall in the range $[0..\alpha_i]$, where $0 \leq \alpha_i \leq 1$.

C_o is the color representing the foreground object; this includes the alpha component. This is the value to be determined, given the backing color C_k and the actual color of the image at that point, C_f. This allows for merging the object with an arbitrary background (represented by C_b), again using the linear interpolant $C = C_o + (1-\alpha_o)C_b$

To solve the matting problem, start with a composite image (like one recorded with a camera), and a known backing color (*e.g.* blue). The following system results:

$$\begin{aligned} R_f &= R_o + (1-\alpha_o)R_k \\ G_f &= G_o + (1-\alpha_o)G_k \\ B_f &= B_o + (1-\alpha_o)B_k \end{aligned}$$

where the X_f components represent the values recorded by the camera, and the X_k's represent the background. The X_o's would form the solution to this system. Note that there are four unknowns, and only three equations, resulting in an under-specified system.

Thus it is impossible to obtain a unique solution to the chroma keying problem when using this approach. By

placing constraints on the system it becomes possible to solve it, *e.g.* by requiring the foreground object to contain no blue component, resulting in

$$C_o = \begin{bmatrix} R_f & G_f & 0 & 1 - \frac{B_f}{B_k} \end{bmatrix}$$

Alternatively, restricting the foreground to be shades of gray, they obtained

$$C_o = \begin{bmatrix} R_f & G_f & B_f - B_k + \alpha_o B_k & \frac{G_f - (B_f - B_k)}{B_k} \end{bmatrix}$$

Smith and Blinn show that a unique, general solution does exist, but their method requires that the objects be shot against two different backgrounds which is not a viable method when actors are involved.

2.3 Chroma Keying: Approximations

Much of the work done on chroma keying has been protected by patents, most of which have expired a few years ago. Many of these were discovered by Petro Vlahos, who is also well known for his Ultimatte™ equipment. The approximate solutions to the matting problem needs human (or maybe intelligent software) intervention to fine-tune some parameters until the result looks correct.

One of the earliest approximations (credited to Vlahos, represented in Smith and Blinn's notation) is of the form

$$\alpha_o = 1 - a_1(B_f - a_2 G_f),$$

where the values are clamped to [0..1].

For this approximation to work, several assumption needs to be made, *e.g.* a blue background is used, actors will not be wearing blue *etc.*

The second approach to chroma keying is more *ad hoc*. The first assumption is that the range of background colors C_k are localized in a small region in 3D-RGB space and is disjoint from the colors found in the foreground objects. Around this region in RGB space several concentric polyhedra are constructed, first a small one containing the background color, with all interior points defined to have an α value of 0 (completely transparent). The largest polyhedron must be on or just outside the boundary of the colors found in the foreground objects, so that $\alpha = 1$ for all points outside this polyhedron. In-between these two polyhedra, several other are constructed, each one representing a surface of constant α, where $0 < \alpha < 1$.

The Primatte™ device from Photron Ltd. is based on this approach, using 128-faced polyhedra. An introductory discussion of the operation of this system can be found in the Primatte whitepaper, published on their web site (http://www.photron.com/WHITEPAPER/kanprie.php3). Their algorithm was originally published in [13].

Smith and Blinn [14] point out several problems that remain with this approach. The algorithm presented next falls into this category of approximations, requiring human intervention to adjust the parameters until acceptable results are obtained.

3 The Algorithm

The most important task of a chroma keying algorithm is to classify a pixel as being of 'key color' or not. For comparison a hue-based approach will be discussed first, followed by the improved approach. The difference between these two algorithms lies in the way in which the color classification is done.

In this paper, the key color is assumed to be blue, since it is a popular choice of key color as it complements human skin tones. It has also been found that people prefer working in a blue room, as opposed to a red or green room.

3.1 Hue-based approaches

Although most computer displays use the RGB (Red, Green and Blue) color space, this is by far not the only one in which images can be represented. With a simple transformation the RGB triple can be converted into its HLS representation, which consists of Hue, Lightness and Saturation components. The geometric representation of HLS color space is a double-hexcone (RGB is a cube). If the hexcone is viewed from above, we see a hexagon where the hue is interpreted as the angle along the edge, measured from Red which lies at 0°. In this arrangement, blue lies at 240° (for a more detailed discussion of the HLS model, see [5]). Thus, to identify a blue pixel, we simply target a sector of the hue space, say $230° - 255°$. Figure 3 illustrates such an HLS hexagon, with the shaded sector indicating the range that will be keyed out. The choice of hue angles are arbitrary, and should be selected by the user to match the particular blue screen.

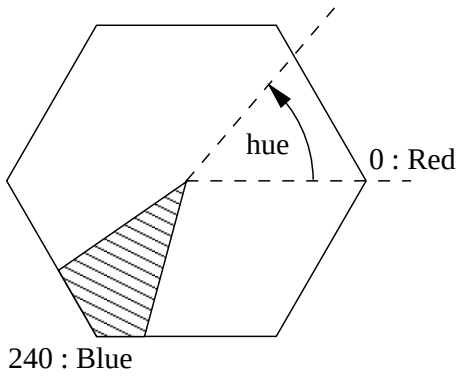

Figure 3: A HLS hexagon

While this approach works well, it presents a difficulty when implemented in software: The image has to be in HLS color space. If not, it has to be converted first, which slows down the algorithm significantly.

For comparison (see Section 4) a software HLS implementation was written. This algorithm was also optimized for keying blue only, which means that a full HLS conversion was avoided. The core of the algorithm can be seen in Figure 4. Note that this implementation only has two alpha levels: opaque and transparent.

```
// s points to the interleaved RGBA image
unsigned char *rp=s,*gp=s+1,*bp=s+2,*ap=s+3;
int h,l,s;
for (int idx = 0; idx < imgsize; idx++)
{
  int max = (*rp > *bp) ? *rp : *bp;
  max = max > *gp ? max : *gp;
  int min = (*rp < *bp) ? *rp : *bp;
  min = min < *gp ? min : *gp;

  if (max == min) h = UNDEFINED; else
  {
    int delta = max - min;
    if (*bp == max) h = 240 +
         (60*(*rp-*gp))/delta;
    else h = 0;
    if (h < 0) h += 360;
  }

  if ((h < hue_max) && (h > hue_min))
   *ap = 0;
  else *ap = 255;
  rp += 4; bp += 4;
  gp += 4; ap += 4;
}
```

Figure 4: C++ code for the HLS algorithm

3.2 An improved approach

If we are provided with an image in the RGB color space, it is possible to implement a fast algorithm requiring a maximum of five operations per pixel. The question that must be answered is "When is a pixel considered to be blue?"

The first set of criteria to be met are that $B > R$ and $B > G$, where R, G, B are simply the separate color components of the image, in the range [0..255] each. In other words, if the pixel is to appear blue, the blue component must be the dominant component. However, this is still too broad, for we could have $B = x$, $R = x - 1$ and $G = x - 1$, which is a shade of gray that will wrongly be classified as blue using the above criteria. One way to narrow the selection criteria is to add a distance constraint. Thus, if

$$d = \sqrt{(B-R)^2 + (B-G)^2} > d_{max} \qquad (1)$$

then we can say that the pixel will definitely appear to be blue, and should be keyed out. Multiplication (squaring) and square root operations are very time consuming, so a simplified distance measure is used instead:

$$d = 2 \times B - R - G \qquad (2)$$

Figure 5 shows an RGB cube containing a skew pyramid *OBTPQ* (in *dashed* lines). This pyramid defines the volume of the cube in which $B \geq R$ *and* $B \geq G$.

Figure 6 shows the skew pyramid *OBTPQ* intersected by a plane *S*. The plane *S* is parallel to the main diagonal line *OP* and represents a surface of constant *d* (as defined in (2)). Note that (2) is only defined for points inside the skew pyramid (blue-dominant colors).

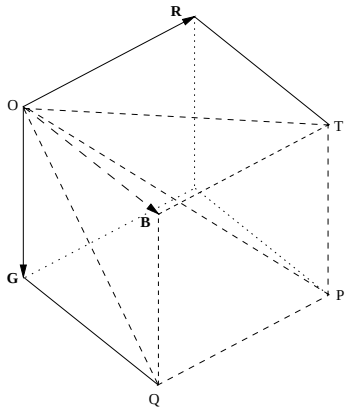

Figure 5: An RGB cube showing the skew pyramid *OBTPQ*

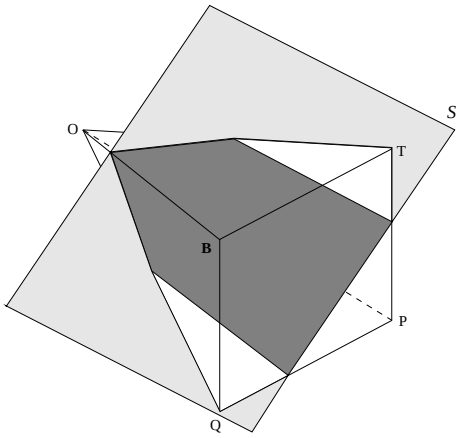

Figure 6: An RGB cube showing a plane *S* intersecting the skew pyramid *OBTPQ*

The dark-gray polygon is the intersection surface of the skew pyramid and the plane *S*. All the points inside this polygon correspond to colors in RGB space with a specific (constant) *d* value, with *d* defined in (2). The diagonal *OP* is the 'gray line' of the cube, as all colors on this line have equal R, G and B components, thus they are all shades of gray. The plane *S* is parallel to the diagonal *OP*, a fact that can also be derived from (2). This implies that all the points in the dark-gray polygon are equally 'blue', or in other words, they are at an approximately equal distance from their gray values with equal intensity. Thus the *d* value of a color inside the skew pyramid can be used to classify it as being blue or not by selecting a threshold in *d*. Note that *d* is not a true distance metric, but rather a computationally efficient approximation.

Visualize the volume *above* the plane *S* before moving on to Figure 7.

Figure 7 again shows an RGB cube. In this cube, the skew pyramid is still visible in dashed lines. Visualize the figure as a cube with the volume of the skew pyramid *above* the plane *S* having been removed. The two light-gray triangles are the side walls of the pyramid (visible above the plane *S*).

As mentioned before, a color (pixel) can thus be clas-

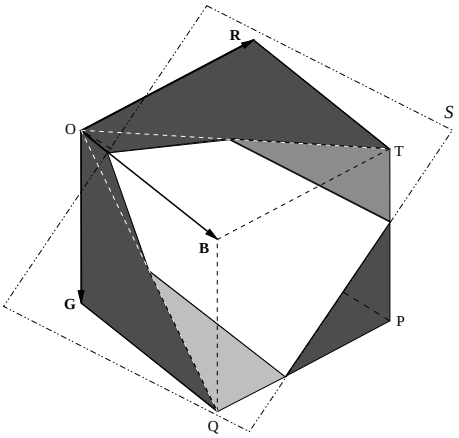

Figure 7: An RGB cube showing a plane S of constant d

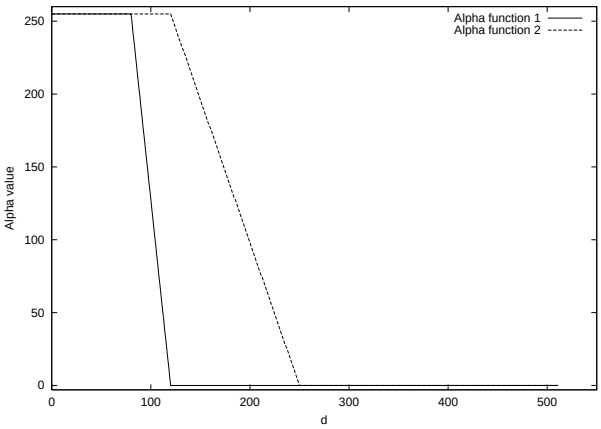

Figure 8: Two sample alpha functions

sified as lying above the plane S (and inside the skew pyramid), or below the plane S, where the exact position of the plane S is set by choosing a value d_{max} as a threshold parameter. For each pixel a value for d is then computed, and a mask is generated according to the outcome of the comparison of this d to d_{max}.

In practice the mask that we generate is not a binary mask, but rather an 8-bit alpha value. When combining the foreground and background images (using the mask), the resulting pixel will be a linear interpolation of the foreground and background pixels with the alpha value acting as the blending parameter. In this sense, the alpha value acts as a transparency value for each pixel.

Simple thresholding of the distance measure d will produce a sharp edge around the person (actor) that we are trying to isolate from the background. Since we have an 8-bit transparency value it is possible to smooth out the edges by using semi-transparent pixels on the edges. The algorithm proposed here solves this problem in an elegant way: The distance d, computed using (2), is used as input to what we call an *alpha function*. The alpha function returns a value between 0 (transparent) and 255 (opaque) based on *how far the pixel is from OP in RGB space*. Equation 2 returns a value between 0 (for pure gray) and 510 (for pure blue), as R, G and B all lie between 0 and 255. Two typical alpha functions are shown in Figure 8.

The reason for showing *two* sample alpha functions is that the algorithm has adjustable parameters that control the shape of the alpha function, which is used to calibrate the algorithm for the particular blue room in which the video is being filmed. In the code, this alpha function is treated as a 512-entry look-up table, thus *any* type of function can be used. In practice, linear ramps like those shown in Figure 8 work very well, producing smooth edges around the actor. The slope of the ramp affects this "edge smoothness", while the starting and ending positions of the ramp selects which shade of blue is keyed out.

Figure 9 lists the core loop of the algorithm. As can be seen in the listing, only two comparisons and possibly one lookup is required for each pixel. A lookup requires an additional three integer operations to calculate the correct index into the table. This means that non-blue pixels require only two operations, while blue pixels require five. Depending on the image, this results in between three and four operations per pixel on average.

```
// s points to the interleaved RGBA image
unsigned char *rp=s,*gp=s+1,
              *bp=s+2,*ap=s+3;
for (int i = 0; i < imgsize; i++)
{
  if (*bp > *rp && *bp > *gp )
    *ap = alpha_map[(*bp<<1) - *rp - *gp];
  else *ap = 255;
  rp += 4; bp += 4;
  gp += 4; ap += 4;
}
```

Figure 9: C++ code for the algorithm

4 Results

The improved algorithm is compared to a hardware chroma keyer in terms of image quality, and in terms of performance to an HLS software algorithm.

4.1 Image quality

Several factors can be considered when comparing the image quality of the software algorithm to that of the dedicated hardware device (an Ultimatte 7, in this case) — for one, the Ultimatte 7 can leave in the shadows, while the current software implementation keys them out. However, for our set-up (See Section 5), the removal of shadows was required. Other than the shadows, the software algorithm produced comparable if not better images. The difficulty here is that both the hardware device and the algorithm have adjustable parameters, so a fair comparison is difficult. In all the test images used it was always possible to obtain comparable results with the software algorithm, within about 2 minutes of fine-tuning.

Given these constraints, a few interesting observations can be made in Figure 11. By looking closely at the surgeon's left arm (at the right-hand side of the image), one

Figure 10: Original image

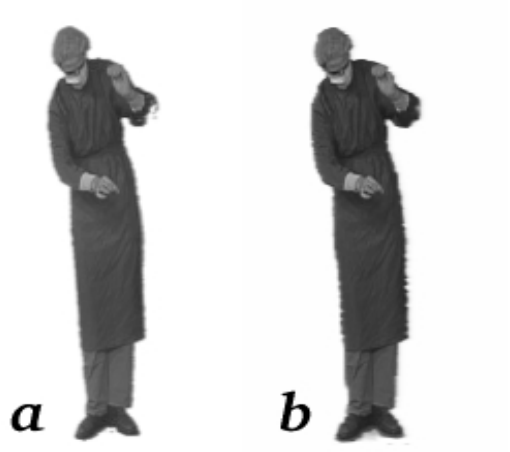

Figure 11: **a**) Hardware keyed, **b**) Software keyed

can see that the hardware keyer has incorrectly keyed out a part of the arm (Figure 11*a*). The software keyer (Figure 11*b*) handled this part slightly better. A blow-up of this region is shown in Figure 12.

In Figure 14 it is once again visible that the software algorithm produces acceptable results. In fact, the software algorithm handled the reflection in the shutter glasses slightly better. Compare the black (solid) glasses in the original (Figure 13) to the holes visible in Figure 14*a*. Notice that the glasses appear much more solid in Figure 14*b*.

Overall, it would be fair to say that the software algorithm presented here produces images that are certainly good enough for the application (see Section 5) for which it is intended.

4.2 Performance

For this section, an optimized HLS algorithm was benchmarked along with the improved algorithm. Testing was done on an Intel Pentium II processor, running at 450MHz. In both tests, the algorithms were applied to an image in main memory, and the Frames Per Second (FPS) rating was computed based on the time it took to key this image one thousand times.

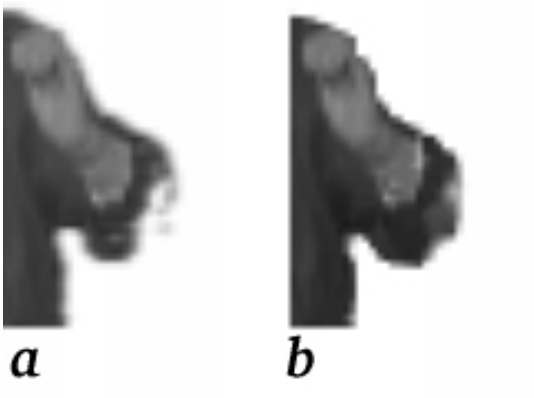

Figure 12: Blow-up of elbow region: **a**) Hardware keyed, **b**) Software keyed

Resolution	HLS Algorithm	Improved Algorithm
720×576	12.02 FPS	43.16 FPS
368×288	50.45 FPS	173.01 FPS
128×256	158.98 FPS	625.00 FPS

Table 1: Software HLS *vs* Improved Algorithm, FPS

Table 1 shows the results of running the algorithm on different size images, while Table 2 shows the same data, but rated in millions of pixels per second. Note that one would expect the Mpixel/s rating to remain more or less constant, but the percentage of blue pixels in the image and the caching efficiency of the machine both influence the performance of the algorithms.

The improved algorithm shows an increase of 340% up to 390% in performance, compared to the HLS algorithm.

5 Application

The improved algorithm presented in Section 3 was implemented and tested in a distributed Virtual Environment system.

Figure 13: Original image

Figure 14: **a**) Hardware keyed, **b**) Software Keyed

Resolution	HLS Algorithm	Improved Algorithm
720×576	4.98 Mpixels/s	17.90 Mpixels/s
368×288	5.15 Mpixels/s	18.34 Mpixels/s
128×256	5.21 Mpixels/s	20.48 Mpixels/s

Table 2: Software HLS *vs* Improved Algorithm, Mp/s

The system consists of two collaborative responsive workbenches (as described in Section 2.1). On each workbench a stereo camera is mounted so that the person using the workbench can be filmed. The resulting video streams are sent to the remote workbench where the images of the person are integrated into the virtual environment that is being displayed on that workbench. By carefully positioning the two video streams (a stereo pair) the illusion of a face-to-face meeting can be achieved, as the workbench has the ability to display stereoscopic images.

Before integrating the image of the person into a virtual environment the background must be removed so that only the person remains in the video stream. If the background was not removed the person *and* the background would appear in the virtual environment, destroying the illusion of the *person* being in the virtual world. The background is removed using chroma keying; initially with the help of a hardware chroma keyer. Because of the simplified domain (*e.g.* stationary cameras, unvarying lighting conditions) it is now possible to use software chroma keying instead, which turns out to be much cheaper.

It is also believed that the improved algorithm presented here is quicker than other approximation methods. The simple calculation used by the algorithm is almost certainly faster than testing whether a point lies inside a 128-faceted polyhedron.

In Section 4.2 it was shown that the software chroma keyer can achieve real-time frame rates (depending on the resolution) on inexpensive CPUs. Further optimization is possible by limiting the region of the image that has to be keyed. This is made possible by the fact that the person working at the workbench will not move around much.

6 Future work

The full range of effects that can be achieved by using different alpha functions is not yet completely understood. It might be possible to preserve shadows by designing the appropriate alpha function, for example by adding a "notch" in the region of the curve where the shadows lie.

Another interesting project would be the automatic determination of the appropriate alpha function. Currently, two parameters control the start and stop points of the "ramp" in the curve, which must be manually adjusted until the desired keying is achieved. Thus, a simplified problem would be the automatic determination of good values for these two parameters as a starting point for interactive calibration.

Lastly, the issue of "blue spill" has not been addressed yet. This occurs when blueish light reflects off the blue screen onto white objects, causing them to appear blue. Currently the object will be partially keyed out if the object appears "blue enough" to the algorithm. Future work will attempt to address this issue.

7 Acknowledgements

Figures 1,2, 10 and 13 were provided courtesy of GMD (German National Research Center for Information Technology).

References

[1] C. Breiteneder, S. Gibbs, and C. Arapis. TELEPORT- an augmented reality teleconferencing environment. In *Proc. 3rd Eurographics Workshop on Virtual Environments Coexistence & Collaboration*, Monte Carlo, Monaco, February 1996.

[2] C. Cruz-Neira, D.J. Sandin, T.A. DeFanti, R. Kenyon, and J.C. Hart. The CAVE, audio visual experience automatic virtual environment. *Communications of the ACM*, June 1992.

[3] P. Dai, G. Eckel, M. Goebel, F. Hasenbrink, V. Lalioti, U. Lechner, J. Strassner, H. Tramberend, and G. Wesche. Virtual spaces - VR projection system technologies and applications. In *Tutorial Notes of the 1997 Eurographics Conference*, Budapest, 1997.

[4] F. Dechelle and M. DeCecco. The IRCAM real-time platform and applications. In *Proc. of the 1995 International Computer Music Conference*, San Francisco, 1995.

[5] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice 2nd Ed.*, chapter 13, pages 590–595. Addison-Wesley, 1992.

[6] S. Gibbs, C. Arapis, C. Breiteneder, V. Lalioti, S. Mostafawy, and J. Speier. Virtual studios: An overview. *IEEE Multimedia*, pages 18–35, January-March 1998.

[7] H. Haase, F. Dai, J. Strassner, and M. Goebel. Immersive investigation of scientific data. *Scientific Visualization, IEEE Press*, 1997.

[8] W. Krueger and B. Froehlich. The responsive workbench. *IEEE Computer Graphics and Applications*, May 1994.

[9] W. Kruger, C. Bohn, B. Froehlich, H. Schueth, W. Strauss, and G. Wesche. The responsive workbench: A virtual work environment. *IEEE Computer*, pages 12–15, May 1994.

[10] V. Lalioti, C. Garcia, and F. Hasenbrink. Virtual meeting in cyberstage. In *ACM Symposium on Virtual Reality Software and Technology, VRST 98*, pages 2–5, Taipei, Taiwan, November 1998.

[11] V. Lalioti, F. Hasenbrink, and C. Garcia. Meet.Me@Cyberstage: towards immersive telepresence. In M Goebel et al, editor, *Virtual Environments'98*, pages 90–102, Springer-Verlag Series, Vienna, Autumn 1998.

[12] E. Lindemann, F. Starkier, and F. Dechelle. The IRCAM musical workstation: Hardware overview and signal processing features. In *Proceedings of the 1990 International Computer Music Conference*, San Francisco, 1990.

[13] Y. Mishima. A software chromakeyer using polyhedric slice. In *Proceedings of NICOGRAPH 92*, pages 44–52, 1992.

[14] Alvy Ray Smith and James F. Blinn. Blue Screen Matting. In *Proceedings of SIGGRAPH'96*, pages 259–268, New Orleans, Louisiana, August 1996.