Tele-Graffiti

Naoya Takao, Jianbo Shi, and Simon Baker

CMU-RI-TR-02-10

Abstract

Many ideas are best communicated by drawing simple sketches. Good examples include the explanation of computer algorithms to undergraduate students and the sketched preliminary designs of architecture. If two people are located remotely from each other a "remote sketching system" can help them communicate better. One good way to build a remote sketching system is to use a video camera to image what one user draws at one site and display the video at the second site using an LCD projector. Such camera-projector based remote sketching systems date back to the original proposal of Paul Wellner's Xerox Double DigitalDesk. To make such a system usable, the users have to be able to move the paper on which they are drawing during operation of the system, they have to be able to interact with the system in a straightforward manner, and the feedback through the two-way system has to be controlled. In view of this we recently developed Tele-Graffiti, a remote sketching system which provides solutions for these three problems. The advantages of Tele-Graffiti include: (1) real-time paper tracking to allow the users to move their paper during the operation of the system, (2) a hand based user interface, and (3) feedback control to maximize the quality of the shared images based on photometric analysis of the feedback loop. In this report we describe the design and implementation of Tele-Graffiti.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivating Example: Remote Education

Remote education is becoming more and more prevalent; many universities already offer a variety of courses to off-campus students. Typically the lectures are video-taped and shipped to the remote students who view the tapes in their own time. Live video feeds are sometimes provided for large corporate clients. Homework and handouts are distributed and returned either using email or an express shipping service.

Instead of physically attending office hours, if a remote student has a question about class or requires help with a homework assignment, etc, they contact the professor or one of the teaching assistants. If the question is a simple one, the student might ask it in an email. If the question is more involved, the remote student might telephone the professor or one of the teaching assistants during their office hours.

Some questions are very difficult to answer over the telephone. In particular, explaining many technical concepts is often almost impossible without the help of a diagram or figure. An example of such a question might occur in a graph theory class. Perhaps the student failed to understand Dijkstra's shortest-path algorithm [Aho *et al.*, 1974] and wants the professor or teaching assistant to explain it again in more detail.

The easiest way to explain Dijkstra's algorithm is to draw an example graph and a table containing the estimates of the shortest distance to each vertex (See [Aho *et al.*, 1974] Figures 5.25 and 5.26 on pages 208–209.) Having drawn the figure, the professor might run through a couple of iterations of the algorithm explaining the steps taken. Explaining even this simple algorithm would be very hard without the diagram; a purely verbal explanation would be virtually impossible to follow.

If the student visits the professor in their office, the professor can easily draw the figure on a piece of paper or on the whiteboard. If the student is a remote student who

has telephoned the professor, some way is needed for the student to be able to see what the professor is drawing, and possibly vice versa.

1.2 Camera-Projector Based Remote Sketching Systems

There are several ways of building a remote sketching system. One way is to use a tablet and a stylus to input the sketch, and a computer monitor to display the sketch at the remote site. Such systems have a number of disadvantages. Writing with a stylus on a glass tablet is unnatural compared to sketching with a regular pen and paper. Shading and other effects are harder to achieve. Changing color means using the computer to select a new color. Incorporating existing hard-copy documents such as a graded exam is impossible.

Another way of building a remote sketching system is to use a *video camera* to image the sketch at one end, transmit the captured video to the other end, and display it there using an *LCD projector*. See Figure 1 for a schematic diagram of how such a system might operate. The first such camera-projector based remote sketching system was Pierre Wellner's Xerox "Double DigitalDesk" [Wellner, 1993].

Since 1993, systems combining video cameras and LCD projectors have become more and more prevalent. Besides the Xerox "DigitalDesk", other such systems include the University of North Carolina's "Office of the Future" [Raskar *et al.*, 1998], INRIA Grenoble's "MagicBoard" [Hall *et al.*, 1999], and Yoichi Sato's "Augmented Desk" [Sato *et al.*, 2000]. A related LCD projector system is Wolfgang Krueger's "Responsive Workbench" [Krueger *et al.*, 1995], used in Stanford University's "Responsive Workbench" project [Agrawala *et al.*, 1997] and in Georgia Tech's "Perceptive Workbench" [Leibe *et al.*, 2000].

1.3 Other Applications of Remote Sketching

Camera-projector based remote sketching systems have a number of applications besides remote education. Such a system could possibly aid tele-collaboration, for example between

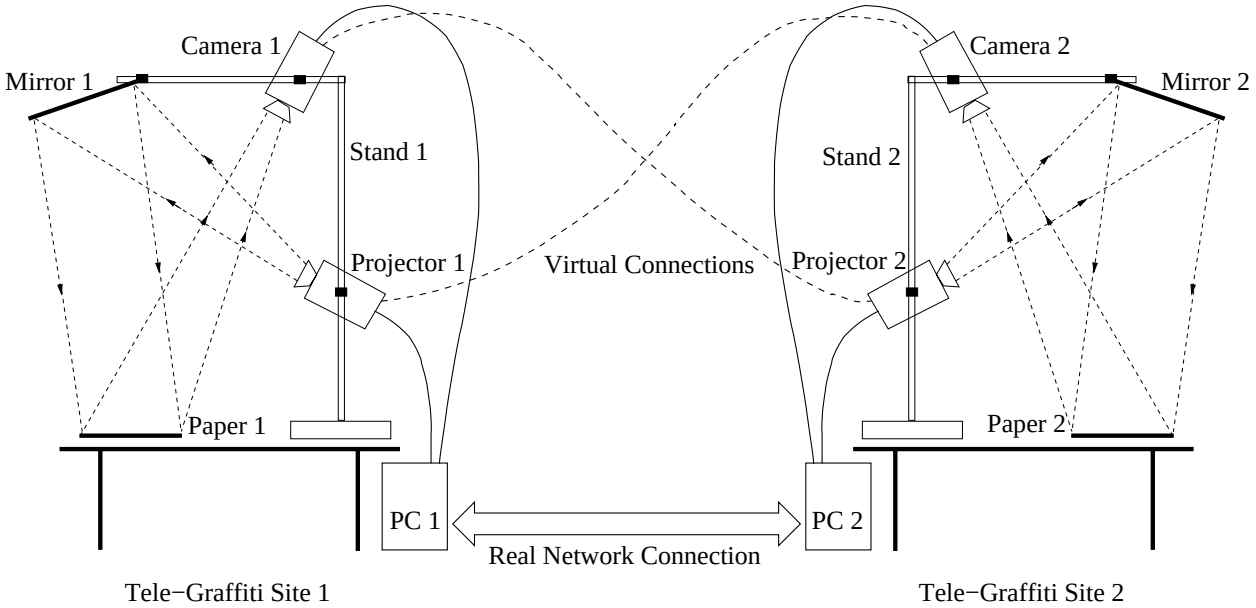


Figure 1: A schematic diagram of 2 Tele-Graffiti sites connected by a computer network [Takao *et al.*, 2001]. The cameras at both sites continuously capture video of what is written on the paper. The video is digitized, compressed, and transmitted to the other site where it is projected onto the paper. Since the pieces of paper can be moved, they have to be tracked in real time at both sites. The projected video can then be warped appropriately to correctly align it with the paper.

a pair of architects working in different cities. While talking, the architects may want to sketch potential ideas they have for the design of a building. They may want to start with an existing plan, placing it under the camera in the sketching system. (Incorporating such an existing hard-copy document into a tablet-based system would be impossible.) They may also want to archive (a summarized version) of the video of their exchange to refer to later.

1.4 Tele-Graffiti

Tele-Graffiti is a remote sketching system that we have recently developed [Takao *et al.*, 2001] [Baker and Shi, 2002] [Takao *et al.*, 2002]. Although the Xerox "Double DigitalDesk" [Wellner, 1993] is an elegant idea, there are a number of technical problems that need to be solved to make it practical and usable:

Real-Time Paper Tracking and Video Transmission: It is natural that the users will

want to move the pieces of paper on which they are writing *during operation of the system*. To allow this capability a camera-projector based remote sketching system must track the paper in real-time. Besides being efficient, paper tracking must also be robust to the occlusions caused by the user's hands, and to the distractions caused by printed or hand-written material on the paper. As the paper may move during the time lag between image projection and capture, we also need to disambiguate the paper from the projected image at the previous time-step. Finally, the four-fold ambiguity in the paper orientation has to be resolved. Figure 16(a) illustrates some of the difficulties encountered in paper tracking.

To solve these problems we have developed a paper tracking algorithm that takes advantage of 3 facts: (1) the paper is bounded by 4 long edges roughly perpendicular to each other, (2) we know the approximate location of the image projected at the previous time step and therefore its edges can be removed, and (3) if we place the paper on a clipboard, the asymmetry between the top and the bottom of the clipboard can be used to resolve the ambiguity in the paper orientation.

To allow real-time interaction between two people, it is also necessary that the video transmission between the remote sites be fast enough for smooth communication. We have implemented an efficient communication system with real-time video compression.

Providing a Suitable User Interface: While remote sketching systems help distant users to communicate through their drawings, such a system would be much more useful if it has functions for sketch summarization, recording, and replay. Moreover, these functions should be able to be controlled in a straightforward manner: i.e. without special devices such as keyboards or mice, and without special (infra-red) cameras.

We have added a hand based user interface to Tele-Graffiti which enables users to invoke miscellaneous functions using only their hands. In parallel with real-time paper tracking, Tele-Graffiti tracks the user's hands in real-time, determines how the hands are interacting with the paper or "user interface objects" such as buttons and sliders

that are projected onto the desktop. When the user selects buttons or moves a slider certain user interface functions are activated.

Minimizing System Feedback and Image Separation: There is the potential for feedback or "visual echoing" in camera-projector systems because the projector radiates light that is then imaged by the camera, and the resulting image then passed back to the projector. It is possible that the projected image may get brighter and brighter until the camera saturates resulting in an unusable system. Alternatively, the system might oscillate between two states.

Based on photometric models of the camera and the projector, we have analyzed the feedback loop in Tele-Graffiti and we derived the "optimal gain" for the system to maximize the image quality (minimizing feedback while ensuring the image from the other side is as visible as possible). Another common problem with LCD projector-based desktop applications is that the ambient light is generally very weak compared to the projector. We propose a way of using the projector to augment the ambient light in the scene, and derive appropriate settings for this scheme.

According to our analysis of the feedback loop, the final "steady state" image viewed by the users of Tele-Graffiti is a weighted combination of the two images that would have been imaged without the Tele-Graffiti feedback loop. In many scenarios it is desirable to estimate what would have been imaged without the feedback. We propose an algorithm to decompose the final viewed images into these "ambient light images."

1.5 Tele-Graffiti Interaction Demo

To illustrate the interactive capabilities of Tele-Graffiti, Figure 2 contains a collection of frames from a video of 2 people playing "tic tac toe" for a demo we gave at ICCV'01 [Takao *et al.*, 2001]. In Figure 2(c)(f)(h), the hand of the user at the remote site can be seen drawing an "O" or a line on the board. Note that the two drawings are well aligned. The complete

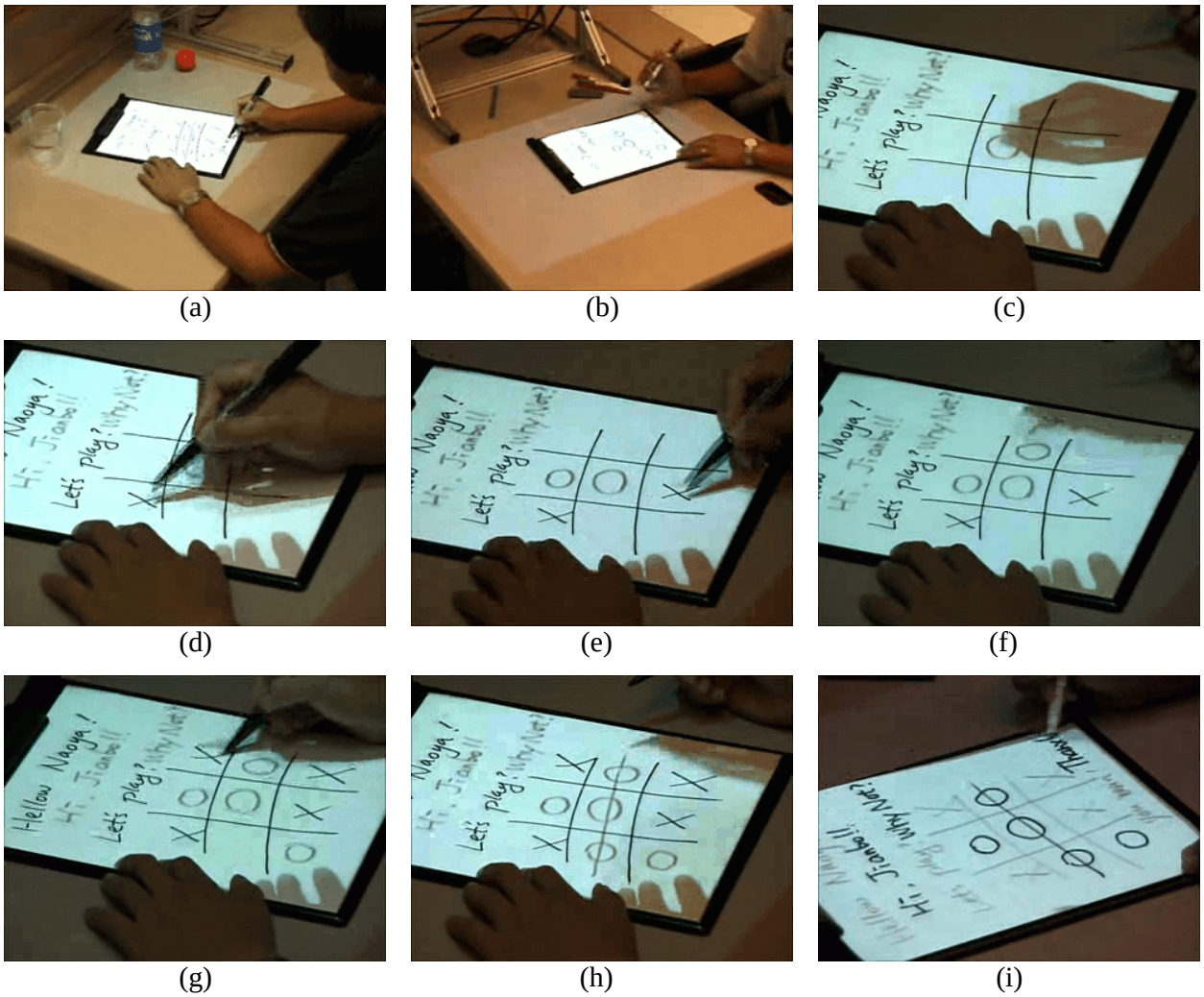


Figure 2: A collection of frames from a video of a demo which we gave at ICCV'01 [Takao *et al.*, 2001]. Two people are playing "tic tac toe," one at site (a) the other at site (b). Images (c)-(h) are of site (a). Image (i) is from site (b) at the end of the game (h). In (c)(f)(h), the hand of the user at the remote site (b) can be seen drawing an "O" or a line on the board. Note that the two drawings at the two sites are well aligned.

video is available on the Tele-Graffiti website at the URL http://www.cs.cmu.edu/~simonb-/telegraffiti/movies/interaction.mpg.

1.6 Overview

In this report we describe the design and implementation of Tele-Graffiti. Section 2 describes the system hardware and its set-up. Section 3 describes the system software, including details of the real-time paper tracking algorithm, the real-time transmission of video, and timing results. In Section 4 we describe our hand detection algorithm, the user interface, the user interface system timing results, and a sample application of the user interface. We proceed to analyze the Tele-Graffiti feedback loop, and derive the "optimal gain" and describe our image separation algorithm in Section 5. Finally, we end with a conclusion in Section 6.

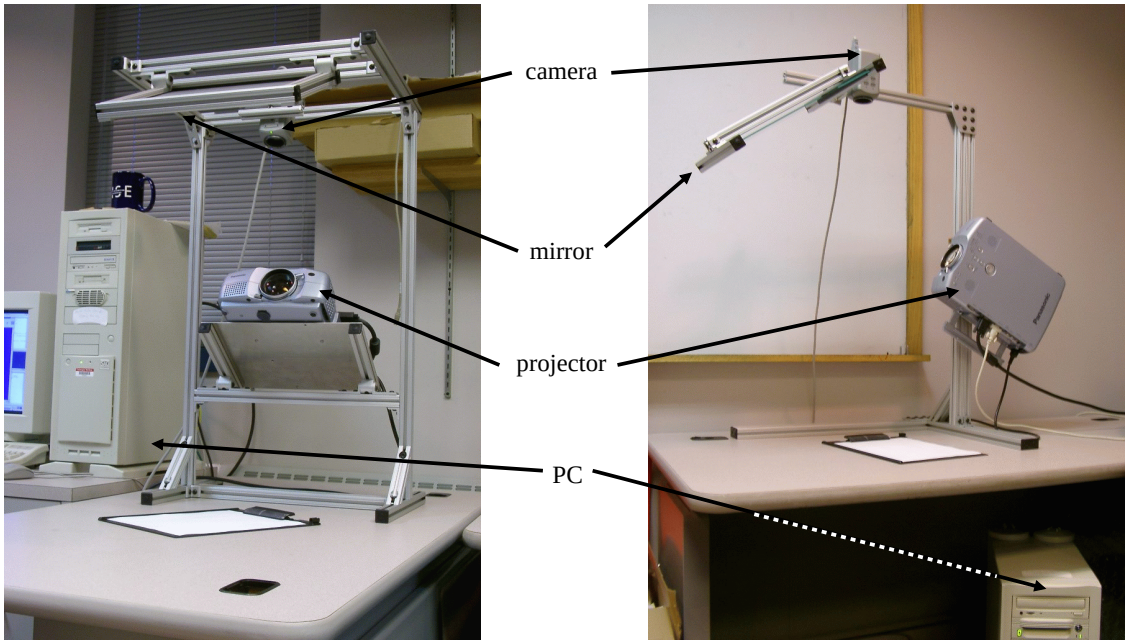
2 Tele-Graffiti Hardware Specifications and Setup

In this section we first describe the Tele-Graffiti hardware (Section 2.1): the stand, the camera, the projector, the PCs, and the network. We then describe how to set-up the system, first how to adjust the stand (Section 2.2), then how to set-up the projector (Section 2.3), next how to set-up the camera (Section 2.4), and finally how to geometrically calibrate the camera-projector relationship (Section 2.5).

2.1 Hardware

A schematic diagram of the Tele-Graffiti hardware design is contained in Figure 1. Figure 3 shows photo images of two real Tele-Graffiti systems. As can be seen, each Tele-Graffiti site contains the following components.

PC: Each Tele-Graffiti site has a PC with dual 450MHz PentiumII processors, an "NVidia GeForce2 GTS" video card, and an OrangeLink Firewire (IEEE1394) card. We use these cards because they have good XWindow/OpenGL support under Linux.



(a) Prototype 1 (b) Prototype 2

Figure 3: Example Tele-Graffiti systems. We have created 2 different prototypes. (a) The projector is mounted horizontally on a supporting plate. (b) The projector is mounted vertically on the pillar of the stand. Prototype 2 was largely designed by Iain Matthews.

Projector: We use a Panasonic PT-L701U LCD projector [Panasonic, 2000] which we run at XGA resolution (1024×768 pixels).

Camera: We use a Firewire (IEEE1394) Sony DFW-VL500 camera [Sony Corporation, 1999], which we run at VGA (640×480 pixels) resolution.

Stand: We have constructed 2 different stands to hold the camera and projector in a compact configuration. In Figure 3(a) the projector is mounted horizontally on a supporting plate, while in Figure 3(b) it is mounted vertically on the pillar. We made these stands out of "80/20 Aluminum Industrial Erector Set" [80/20 Inc., 2002].

Network: The two Tele-Graffiti sites are connected by a local-area network. We have experimented running the system over both 100Base-T and 10Base-T networks.

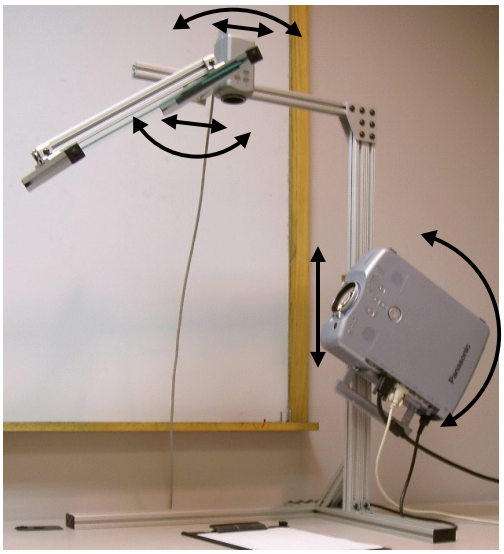


Figure 4: Degrees of freedom of the mirror, camera, and projector in Tele-Graffiti Prototype 2. Each component has 2 DOF: it can be rotated and moved linearly.

2.2 Adjusting the Stand

The Tele-Graffiti stands are highly configurable. The mirror, camera, and projector each have 2 degrees of freedom. The mirror can be rotated and moved inwards and outwards. The camera can be rotated and also moved inwards and outwards. The projector can be rotated and moved up and down. See Figure 4 for an illustration of the degrees of freedom.

To set up the stand, the following two operations have to be performed:

Adjust the Projected Area: Set up the projector so that it is projecting a constant color, typically blue. Adjust the mirror and the projector so that the projector projects light entirely onto the mirror, and the light is reflected approximately vertically downwards to create an approximately rectangular region on the desktop. Keystone correction can be used if so desired to make the working area more rectangular.

Adjust the Viewable Area: Move and rotate the camera so that the center of the captured image is approximately the same as the center of the image projected by the projector. To help in this task, we use a program that displays a "cross" in the center

of the projected image. See Figures 5(a) and (b). Figure 5(a) shows an image captured before the camera was adjusted, Figure 5(b) after the camera was adjusted.

2.3 Setting Up the Projector

To set up the projector, the following steps need to be performed.

Adjust the Video Signal: Make sure that the projector is projecting the whole video signal, and if not, adjust the projector parameters appropriately. With the factory settings, the projector may cut off the extremity of the signal. For the Panasonic PT-L701U projector, adjust the H-POSI/V-POSI/DOT CLK/CLK PHASE parameters in the POSITION menu.

Adjust the Video Orientation: Adjust the projector parameters to change the orientation of the projection so that the user can see the PC screen without a vertical or horizontal flip. Though this is not necessary for the paper tracking and video projection onto the paper, setting up the projector in this way makes it much easier to display letters in the user interface (see Section 4). Letters can be displayed without being reflected, something that requires the details of the font to perform. For the Panasonic projector, a combination of FRONT/REAR and DESK/CEILING in the OPTION menu need to be set appropriately.

Set the Zoom: The zoom on the projector must be adjusted to set the size of the working area on the desktop. Zooming in decreases the working area. Zooming out increases the working area. When zooming out to increase the working area, it is important to make sure that the projector continues to project entirely onto the mirror. If the projector is zoomed out too far, the locations of the mirror and the projector (and the camera) may need to be re-adjusted. See Section 2.2 for more details.

Focus the Projector: The focus setting of the projector must be adjusted (normally this is only possible by hand) to focus the projected image on the desktop. To do this, it is generally better to display a textured image of some sort.

2.4 Setting the Camera Parameters

Following the setup of the stand and the projector, we then adjust the camera parameters. The Sony DFW-VL500 camera allows almost all of the parameters to be changed over the FireWire interface. See Table 1 for a summary of the parameters and our policy for adjusting them. The parameters that are denoted "fixed" are set to a fixed value at system initialization and not changed. Those set to "auto-camera" (like the iris) can be automatically adjusted by the camera. Those set to "auto-tg" are automatically set by Tele-Graffiti during system startup using three programs we developed: "autozoom", "autofocus," and "autocolor' which adjust the zoom, focus, and color parameters respectively.

2.4.1 Image Format

The Sony DFW-VL500 camera can produce images in any of the following formats:

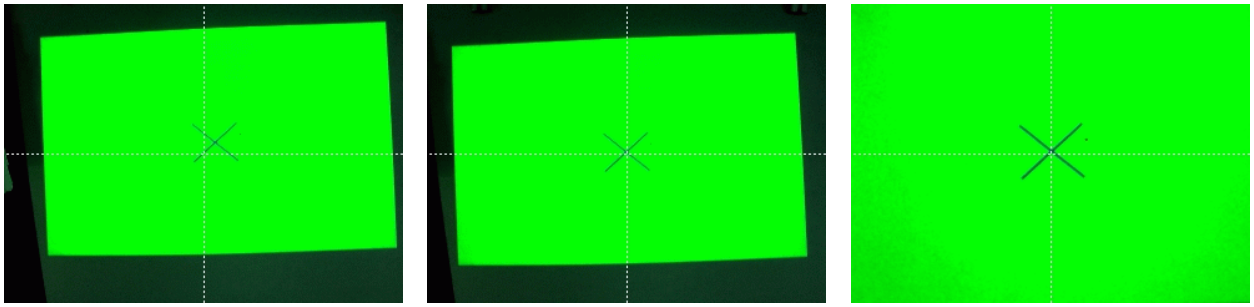
1. YUV 4:4:4 $\times 160 \times 120$
2. YUV 4:2:2 $\times 320 \times 240$
3. YUV 4:1:1 $\times 640 \times 480$
4. YUV 4:2:2 $\times 640 \times 480$.

We chose 4 (YUV 4:2:2 $\times 640 \times 480$ pixels) for the highest resolution and the richest color.

We can also select one of 4 different frame rates: 3.75Hz, 7.5Hz, 15Hz, and 30Hz. We chose 30Hz to maximize the video transmission rate and to make the paper tracking as smooth as possible. See Section 3.3 for the details of the paper tracking algorithm.

Table 1: Sony DFW-VL500 camera parameters and our policy for adjusting them. The parameters denoted "fixed" are set to a fixed value at system initialization and are then not changed. Those set to "auto-camera" can be automatically adjusted by the camera. Those set to "auto-tg" are automatically set by Tele-Graffiti during system startup.

Parameter	Policy	Value
Image Format (See Section 2.4.1)		
image format	fixed	YUV 4:2:2, 640 × 480
frame rate	fixed	30Hz
Correction (See Section 2.4.2)		
gamma	fixed	130 (OFF2)
gain	fixed	0 (No Gain)
brightness	fixed	0
optical filter	fixed	OFF
Zoom (See Section 2.4.3)		
zoom	auto-tg	set by "autozoom"
Focus (See Section 2.4.4)		
focus	auto-tg	set by "autofocus"
sharpness	fixed	62
Exposure (See Section 2.4.5)		
exposure	fixed	96
shutter	fixed	2311 (33.3msec, 30Hz)
iris	auto-camera	N/A
Color (See Section 2.4.6)		
white balance	auto-tg	set by "autocolor"
hue	auto-tg	set by "autocolor"
saturation	auto-tg	set by "autocolor"



(a) (b) (c)

Figure 5: Adjusting the optical axis and the zoom with "autozoom." (a) Autozoom displays a green rectangle filling the screen with a big X in the center, zooms-out the camera as far as possible, and provides the user with the camera view on another PC screen. (b) The user adjusts the camera angles so that the camera axis (represented by an intersection of the two lines) points to the center of the projected area (represented by a large X). (c) Autozoom automatically zooms-in the camera until the camera view is just filled with green texture.

2.4.2 Correction

The Sony DFW-VL500 can perform a number of different intensity corrections to the captured image. We chose to turn all of these corrections off. First the image can be gamma corrected. We chose to turn gamma correction off to ensure a linear relationship between the intensity and the amount of light captured. This linear response is required in the analysis in Section 5. See Section 5.1.5 for an empirical validation of the linearity.

Second, the image can be gain and bias (brightness) corrected. We turned off this correction also. Finally, the camera has an optical filter which we also turned off.

2.4.3 Zoom

Later in this section, *italic* fonts are used to denote the camera parameters.

When Tele-Graffiti is started, a program called "autozoom" can be used to automatically set the *zoom* of the camera so that the viewing area of the camera matches the projection area of the projector. Specifically, *zoom* is set to give the largest viewing area possible that is entirely contained within the projection area. Autozoom operates as follows:

1. Assistance to adjust the camera axis (See also Section 2.2)

Autozoom changes *zoom* to the most zoomed-out state, displays (projects) a green rectangle filling the screen with a large cross in the center, and repeatedly captures an image from the camera and display it. The user then changes the camera angles by hand until the cross comes to the center of the captured image. Figure 5(a) shows an image before the user has adjusted the camera. Figure 5(b) shows an image afterwards.

2. Set the *zoom* automatically

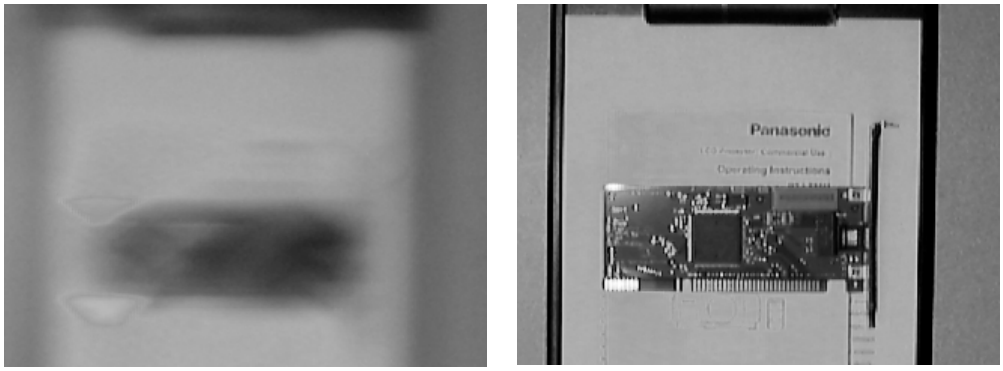
   Autozoom starts from the most zoomed-out state. It then zooms in step by step until the camera view is filled with the projected green color. See Figure 5(c).

2.4.4 Focus

Once the *zoom* of the camera is set, the camera is focused using an "autofocus" program. Since defocusing has the effect of suppressing the high-frequency components, any high-pass filter can be used as a measure of how focused the image is. Generally the Laplacian is used [Nayar and Nakagawa, 1994, Nayar *et al.*, 1996]. We use the following definition of the Laplacian:

$$\begin{aligned} \nabla^2 I(x,y) \quad = \quad & -I(x+1,y-1) \quad -I(x+1,y) \quad -I(x+1,y+1) \\ & -I(x,y-1) \quad +8 \times I(x,y) \quad -I(x,y+1) \\ & -I(x-1,y-1) \quad -I(x-1,y) \quad -I(x-1,y+1) \ . \end{aligned}$$

The average magnitude of the Laplacian is computed over a window in the center of the image. This average value is then maximized using a binary search over the *focus* setting. The resulting *focus* setting gives the most focused image. An example of the result of running this algorithm is shown in Figure 6. Before running the algorithm the user places a collection of textured objects in the center of the field of view of the camera. The un-focused image in Figure 6(a) is an image captured with the camera before the algorithm is run. Figure 6(b) contains an image with the result of the auto-focus algorithm. As can be seen, the auto-focus algorithm produces a perfectly focused image.



(a) (b)

Figure 6: Adjusting the focus with "autofocus." (a) Captured image before adjusting focus. (b) Captured image after adjusting focus with "autofocus."

The Sony DFW-VL500 also has electronics to artificially sharpen (or deblur) the image. We turn this feature off by setting the sharpness to be 62.

2.4.5 Exposure

There are 3 camera settings that control the exposure of the camera. The *shutter* controls the shutter speed and the *iris* controls the aperture of the iris. Either both of these settings can be fixed to a "manual" setting, or one of them can be set to "automatic." If one of the two settings is set to "automatic," the automatic gain control (AGC) is activated and the camera dynamically adjusts whichever of the *shutter* and the *iris* is set to automatic. The setting is adjusted to keep the average intensity of the image a fixed value. This value is specified by the third camera setting: the *exposure*.

Since Tele-Graffiti must operate in a relatively uncontrolled environment where lights may be switched on or off at any time, we found it best to use the automatic gain control. Typically, offices have fairly poor illumination so we set the *shutter* to 30Hz (2311 for the Sony DFW-VL500's) because we found that the automatic gain control was unable to give sufficiently bright images at higher shutter speeds. We set the *iris* to automatic. We found that a suitable value for the *exposure* that gives adequate dynamic range was 96.

In Section 5 we analyze the "gain" through Tele-Graffiti and the effects of feedback. To

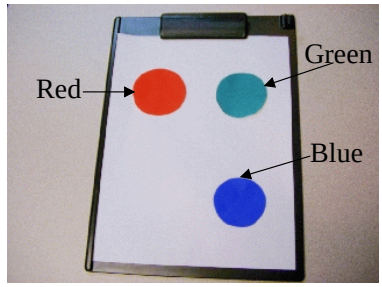


Figure 7: The color calibration chart used to calibrate the color parameters of the camera. 3 colored patches are placed on a blank paper: one red, one green, and one blue.

validate our analysis we experimented with fixed *iris* and *shutter* to set the system "gain" to any fixed value that we desired. In normal operation, however, it is best to use the automatic gain control. The analysis with fixed gain is performed simply to validate our understanding of the feedback, rather than because we feel it is practical to use a fixed gain.

2.4.6 Color Calibration

There are 4 camera parameters that affect the color calibration of the camera, the *white balance U*, the *white balance V*, the *hue* and the *saturation*. We calibrate these 4 parameters using the color calibration chart shown in Figure 7. The calibration chart consists of a white piece of paper with 3 colored disks placed on it, one red, one green, and one blue. Our "autocolor" calibration program first locates these colored disks and then performs the following two steps.

First, the camera is white balanced. We minimize the value of:

$$\sum_{(x,y)\in A_w} (u(x,y)^2 + v(x,y)^2) \tag{1}$$

where A_w is the small region of the paper which is known to be white (no color disks), and $u(x,y)$ and $v(x,y)$ are the observed values of the U and V components (in the YUV color representation) respectively, at pixel (x,y). A pure white image should have zero U value and zero V value. We minimized the expression in Equation (1) using a 2-dimensional binary search over the parameters *white balance U* and *white balance V*.

Second, we adjust the *hue* and *saturation*. We minimize the value of:

$$\begin{aligned} & \textstyle\sum_{(x,y)\in A_r} \left[\frac{\mathrm{C}_{(x,y)} \cdot \mathrm{R}}{|\mathrm{C}_{(x,y)}||\mathrm{R}|}\right]^2 + \textstyle\sum_{(x,y)\in A_g} \left[\frac{\mathrm{C}_{(x,y)} \cdot \mathrm{G}}{|\mathrm{C}_{(x,y)}||\mathrm{G}|}\right]^2 \\ + \quad & \textstyle\sum_{(x,y)\in A_b} \left[\frac{\mathrm{C}_{(x,y)} \cdot \mathrm{B}}{|\mathrm{C}_{(x,y)}||\mathrm{B}|}\right]^2 + \textstyle\sum_{(x,y)\in A_w} \left[\frac{\mathrm{C}_{(x,y)} \cdot \mathrm{W}}{|\mathrm{C}_{(x,y)}||\mathrm{W}|}\right]^2 \end{aligned} \tag{2}$$

where $A_r/A_g/A_b$ are the the small regions of the paper which are known to be red/green/blue respectively, $\mathrm{C}_{(x,y)}$ is the vector $(red, green, blue)$ of the observed color components at pixel (x, y), and R/G/B/W are the target color vectors for red/green/blue/white respectively (i.e. $\mathrm{R} = (255, 0, 0), \mathrm{G} = (0, 255, 0), \mathrm{B} = (0, 0, 255), \mathrm{W} = (255, 255, 255)$). Similarly, Equation (2) is minimized using a 2-dimensional binary search over the *hue* and *saturation* parameters.

The results obtained using "autocolor" are shown in Figure 8(a) and Figure 8(b). We found it relatively hard to color calibrate the Sony DFW-VL500's when the ambient lighting level was low, as in a normal office environment. In particular, the result displayed in Figure 8(b) was obtained with the use of an additional spot light. The result in Figure 8(c) is the result which we obtained with "autocolor" in a normal office environment. As can be seen, the improvement in the color reproduction is far less. We conclude that the Sony DFW-VL500's are best color calibrated in strong ambient light.

2.5 Geometric Calibration of the Camera-Projector Relationship

While we track the location of the paper in the camera (see Section 3.3), we need to warp the video so that when it is displayed by the projector, it appears correctly aligned with the paper. We therefore need to know the relationship between camera coordinates (x_c, y_c) and projector coordinates (x_p, y_p). We assume that the projector follows the same pin-hole model as the camera (with the light rays in the reverse direction). Assuming the paper is planar, the relationship between camera and projector coordinates is:

$$\begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} \equiv H_{pc} \begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} \equiv {H_{pc}}^{-1} \begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} \tag{3}$$

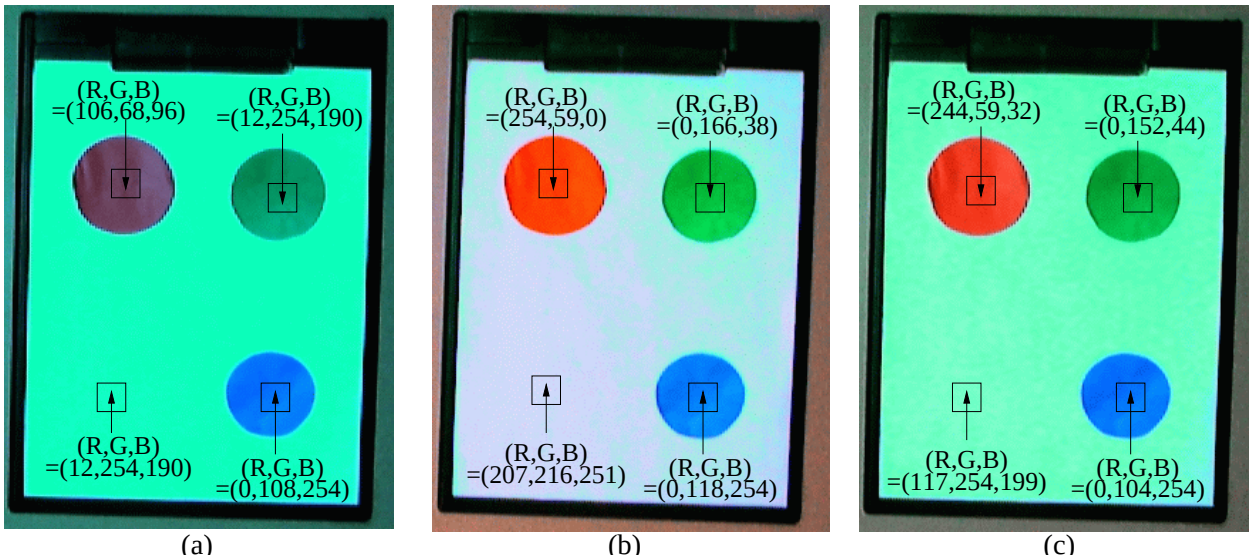


Figure 8: Results of color calibration. (a) Captured image before color calibration. The four color parameters are set to their factory settings (all 128). (b) After color calibration using "autocolor" with an additional spot light. (c) After color calibration in a normal office environment. The average values of red, green and blue components over the four small regions (A_r, A_g, A_b and A_w) are also shown. While (b) shows a great improvement in color reproduction, there is far less improvement in (c).

where H_{pc} is a 3×3 homography or colineation, and $\equiv$ denotes equality up to scale [Faugeras, 1993]. Since H_{pc} doesn't change if the paper remains in the same plane (i.e. the paper stays on the desktop), H_{pc} can be computed at system startup. This constant value of H_{pc} is pre-computed by: (1) projecting a rectangular image with known corner locations onto the desktop, (2) capturing an image of this calibration image, (3) locating the vertices in the captured image using the paper tracking algorithm described in Section 3.3 (since there is no clipboard, we give the system enough prior orientation information to break the four-fold ambiguity in the orientation of the projected image), and (4) solving for H_{pc} using Equation (3) and the 4 pairs of projector-camera coordinates. See Figure 9 for an illustration of this calibration process, which for advertising purposes is conducted with a "Tele-Graffiti" calibration image.

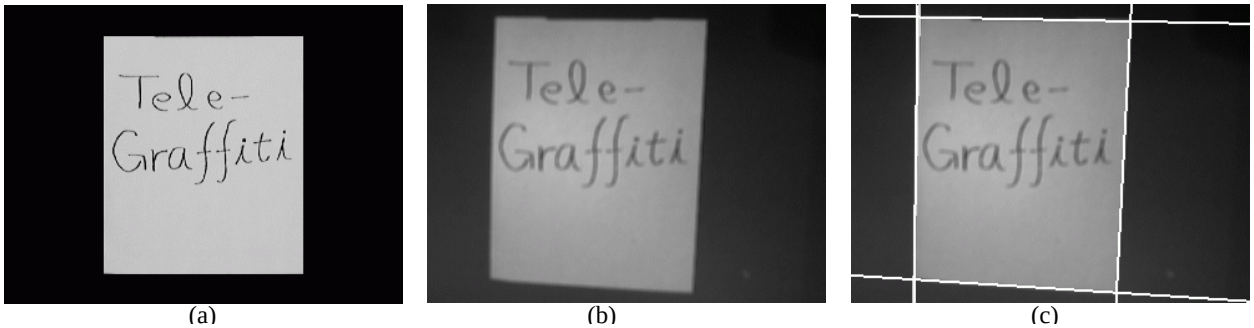


Figure 9: Geometric calibration. A calibration image of a rectangle with known vertices (a) is displayed using the projector and captured by the camera (b). The vertices of the quadrangle in (b) are estimated (c) using the paper tracking algorithm described in Section 3.3. The homography H_{pc} is estimated from the 4 corresponding pairs of camera and projector coordinates and Equation (3).

3 Tele-Graffiti System Software

Each Tele-Graffiti site needs to continuously capture video from its camera, track the paper in the video, warp the image to display so that it is aligned with the paper, and communicate with the other sites. In this section we describe the system software that performs all of these tasks. We begin by describing the system architecture and the 4 threads that it consists of. We then detail each of the 4 threads in turn. We conclude this section by describing the run-time performance of the system.

3.1 System Architecture

Tele-Graffiti runs under Linux (RedHat 7.1) and consists of 4 threads: the Drawing thread, the Paper Detection thread, the Sending thread (the Sending thread is actually implemented in a second process due to TCP/IP constraints), and the Receiving thread. See Figure 10 for a diagram of the 4 threads. The 4 threads share the following data:

Image to Display The latest image received from the remote site. It is primarily shared between the Receiving thread and the Drawing thread.

Remote Paper Vertices The estimated paper vertices in the image to display.

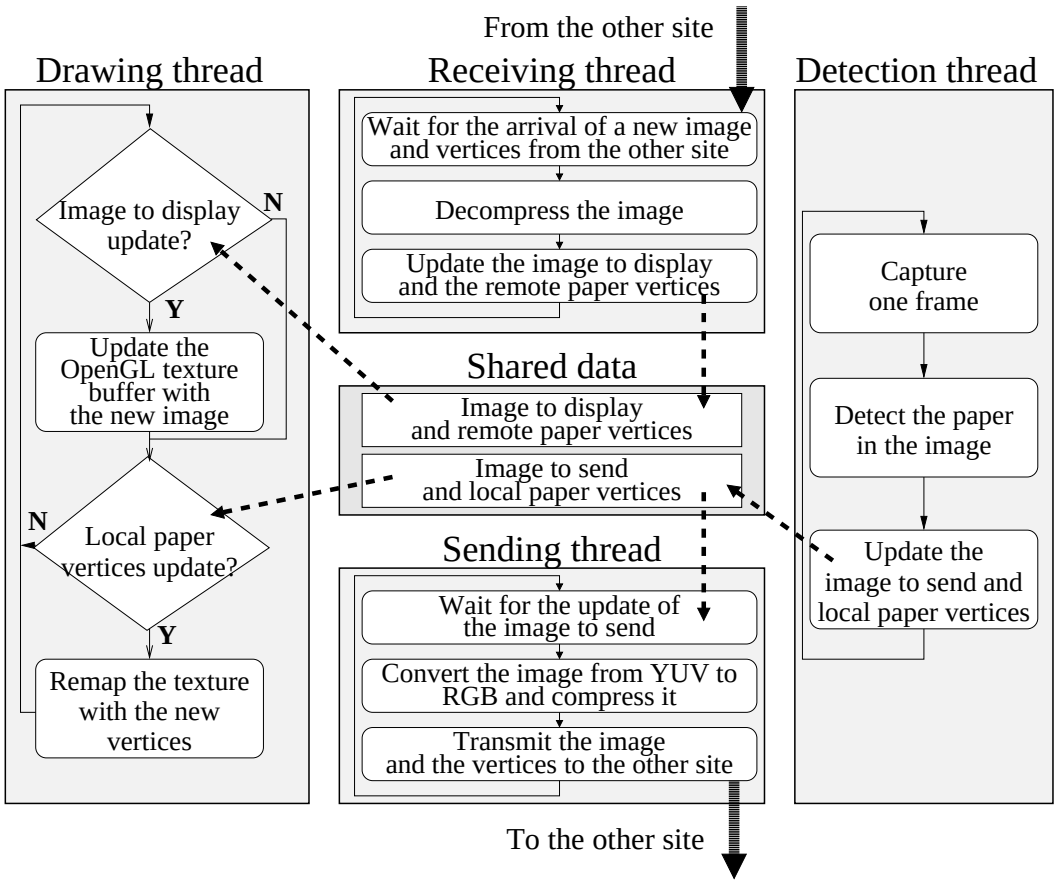


Figure 10: Tele-Graffiti software system architecture. The system consists of 4 threads sharing a common data-structure. The Drawing thread continuously displays the latest image received from the other site (the "image to display") using the latest estimate of the paper vertices (the "local paper vertices"). The Paper Detection thread continuously grabs images, tracks the paper location, and updates the "local paper vertices" and the "image to send." The Receiving thread continuously waits for images and the "remote paper vertices" to arrive from the other site. It then decompresses them and updates the "image to display". The Sending thread (actually implemented as a separate process due to TCP/IP constraints) continuously converts the "image to send" from YUV to RGB, compresses it, and sends it to the other site with the "local paper vertices."

Image to Send The image to send is a YUV image which is primarily shared between the Paper Detection thread and the Sending thread. It is a sub-image of the captured 640×480 image which is just large enough to include the detected paper. Typically, the width and height of the image to send are around $200 \sim 400$ pixels.

Local Paper Vertices The estimated paper vertices in the captured 640×480 image.

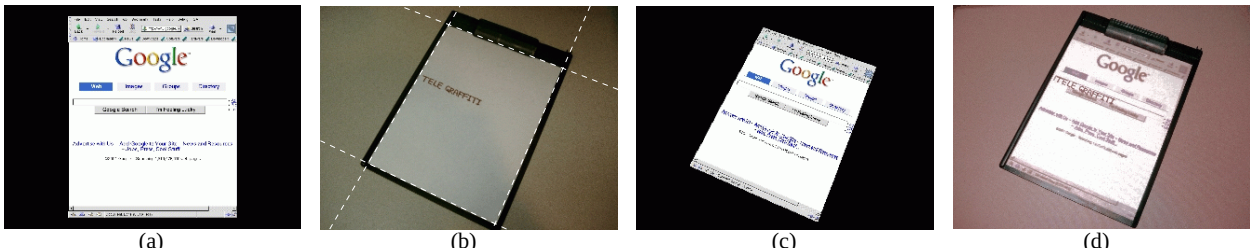


Figure 11: Image warping. The image to display (a). The location of the paper estimated by the paper tracking algorithm of Section 3.3 (b). The image (a) warped using OpenGL becomes the image to project (c). Projecting image (c) onto the paper results in perfect alignment (d).

The first of two these data structures are implemented in global variables with access control, and the last two are stored in "shared memory" utilizing the Linux kernel's mechanism for data sharing between processes.

In the following sections, we describe each thread in detail, starting with the Drawing thread in Section 3.2, then the Paper Detection thread in Section 3.3, and finally the Sending thread and the Receiving thread in Section 3.4.

3.2 Drawing Thread

This thread continuously warps and draws the image to display. The drawn image is output to the projector simply by plugging the monitor output of the PC into the projector. Dual headed video card could be used instead. The Drawing thread waits for updates to the image to display and upon update copies the new image into the OpenGL texture buffer. This thread also waits for changes to the local paper vertices. Whenever this occurs, the Drawing thread redraws (re-maps) the texture on the screen with the new local paper vertices.

3.2.1 Image Warping

To project the image to display correctly so that it appears aligned with the paper at this Tele-Graffiti site, we need two sets of vertices: the vertices of the paper in the image to display and vertices in the projector to which they must be mapped. While the former are

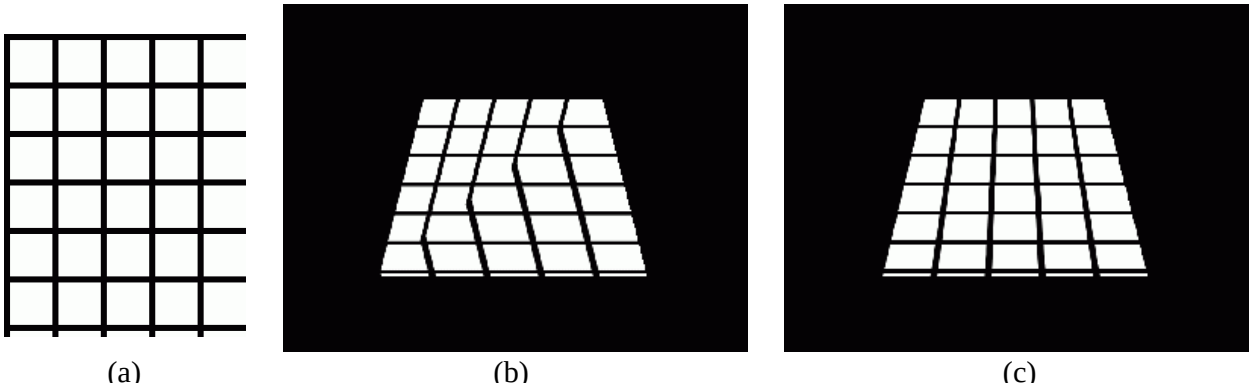


Figure 12: OpenGL texture mapping. (a) Original image. (b) Result image warped with four vertices (one quadrangle). (c) Improved warped image by dividing into a large number of quadrangles.

the remote paper vertices which are received from the other site, for the latter, we need the local paper vertices in the projector coordinate frame (x_p, y_p). Using the right-hand side of Equation (3), these vertices can be computed from the local paper vertices in the camera coordinates (x_c, y_c), which are estimated by the Paper Detection thread with the paper tracking algorithm described in Section 3.3. OpenGL texture mapping is then used to warp the image. See Figure 11 for an illustration of this procedure.

3.2.2 OpenGL Texture Mapping

The OpenGL API provides a very convenient method for image warping, where you only need to specify two pairs of four vertices each of the source and destination quadrangles. However, this function sometimes results in an image with artifacts due to the triangle-based warping algorithm that OpenGL uses. (This may depend upon the specific graphics card and driver you are using.) When warping a quadrangle, OpenGL divides the quadrangle into two triangles and warps them independently, each with a separate affine warp. Thus a discontinuity can sometimes be observed along the diagonal line in the destination image. See Figure 12(b) for an example. We subdivide the paper into a large number of small quadrangles to avoid this effect. We divide the paper into 16×16 quadrangles, calculate the coordinates of the 17×17 vertices for both source and destination images, and tell OpenGL

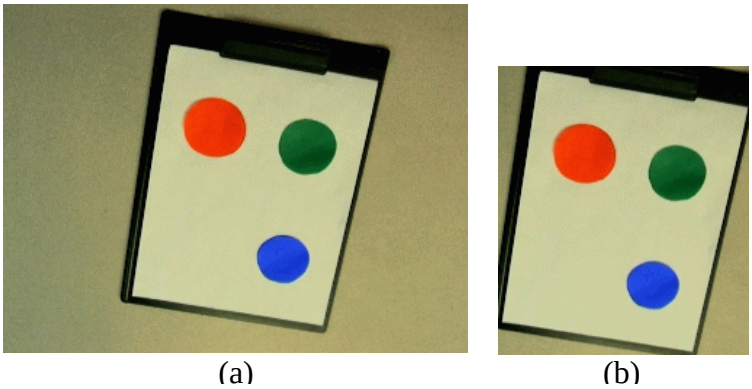


Figure 13: Creating the image to send. The Paper Detection thread detects the paper in the captured image (a) and estimates the paper vertices. Then, using this information, the image to send (b) is created by cropping the captured image so that it is just large enough to include the detected paper image. In this example, the size of the image to send is 346×400 pixels compared to 640×480 for the captured image, a reduction of 55%.

to warp these quadrangles all at once. See Figure 12(c) for an example of the improved result. Since OpenGL warping is usually done in hardware and is very fast (typically only a few milliseconds), the lowering of the warping speed caused by using a large number of quadrangles is negligible.

3.3 Paper Detection Thread: Paper Tracking

The Paper Detection thread continuously does the following:

1. Grabs an image from the camera.

2. Detects or tracks the paper.

   See below for the details of the paper tracking algorithm.

3. Updates the image to send and the local paper vertices.

   Updating the image to send is done by cropping the grabbed image according to the estimated paper vertices. See Figure 13 for an example of cropped image.

4. Notifies the Drawing thread of the update.

Tele-Graffiti Paper Tracking Algorithm

(1) Down-sample and smooth the captured image

(2) Detect edges in the image

(3) Remove edges which come from the projection of previous time steps

(4) Group edge pixels with respect to their angles

(5) Divide edge pixels into paper-clipboard and clipboard-desk edges

(6) Fit lines for each edge group

(7) Estimate the paper orientation: Break the four-fold ambiguity

Figure 14: An overview of the Tele-Graffiti paper tracking algorithm. First edge pixels are detected in the down-sampled and smoothed image and projected edges removed. Then, edges are grouped into four groups each of which are divided into paper-clipboard and clipboard-desk edges. Finally, a line is fit to each of the paper-clipboard edge groups and the paper orientation is estimated.

An overview of our paper tracking algorithm is shown in Figure 14. We discuss the details of each of the steps in the following sections. Our paper tracking algorithm is both efficient (it operates in 20msec on a 450MHz PC) and robust to occlusions, fast motion of the paper, and even the removal and replacement of the paper. The timing results of our algorithm is given in Table 4. Figure 15 contains a collection of frames from a movie illustrating the robustness of our tracking algorithm. The complete video is available on the Tele-Graffiti website at the URL http://www.cs.cmu.edu/~simonb/telegraffiti/movies/papertracking.mpg.

3.3.1 Image Down-Sampling and Smoothing

First the captured image (640×480 YUV 4:2:2) is converted to grayscale and down-sampled to a 320×240 image. Then it is smoothed in both the horizontal and vertical directions. (The down-sampling step is not necessary with a faster CPU.) See Appendix A.2 for an efficient MMX implementation of YUV to Grayscale conversion and down-sampling.

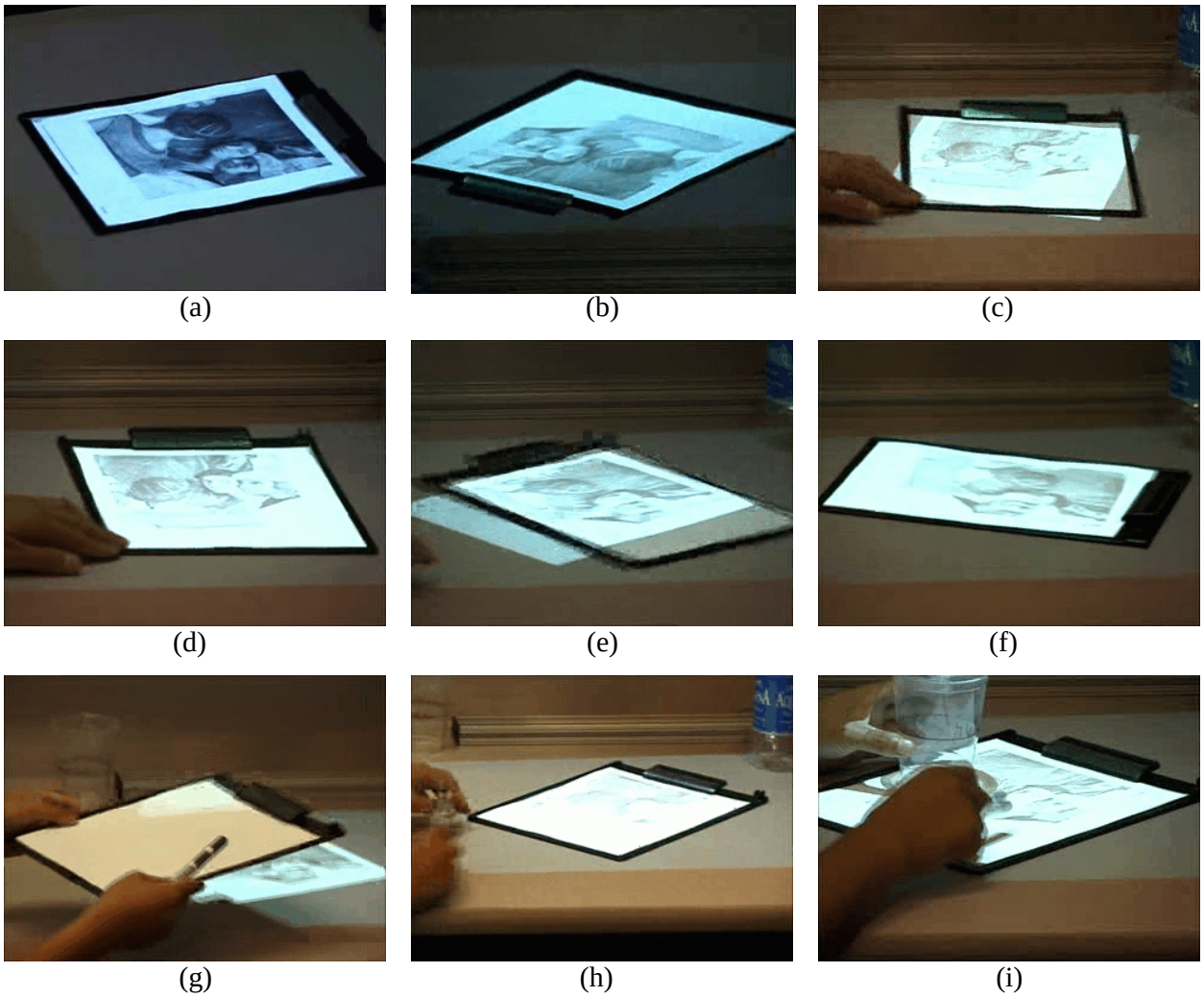


Figure 15: Results of paper tracking. Continuous video of the paper at one site (a) is captured, transmitted, and projected accurately onto the paper at the other site (b). When the paper is moved at the second site (c), it is tracked robustly (d), even if the paper is moved very quickly (e)(f). The paper can be removed temporarily from the system (g). Paper tracking resumes as soon as the paper is returned to the desktop (h). Paper tracking is efficient and robust to occlusions (i). The complete video is available on the Tele-Graffiti website at the URL http://www.cs.cmu.edu/~simonb/telegraffiti/movies/papertracking.mpg.

3.3.2 Edge Detection

Edge pixels are detected based on a constant threshold on the magnitude of the gradient, and are recorded along with their angles. For every pixel in the captured image, we calculate simple pixel difference estimates of the intensity gradients in both the X and Y directions. If

$I(x, y)$ denotes the intensity value at pixel (x, y) in the image we estimate the image gradient $G(x, y)$ at pixel (x, y) as $G(x, y) = \left(\frac{dI}{dx}, \frac{dI}{dy}\right) = (I(x+1,\ y) - I(x,\ y),\ I(x,\ y+1) - I(x,\ y))$. Thresholding $G(x, y)$ with the following condition

$$\left(\frac{dI}{dx}\right)^2 + \left(\frac{dI}{dy}\right)^2 \geq C$$

where C is a constant, we get the edge pixels E_i:

$$\begin{aligned} E_i &= (x_i, y_i, \theta_i) \quad (i = 1, 2, ...n) \\ \theta_i &= \arctan\left[\frac{\left(\frac{dI}{dy}\right)_i}{\left(\frac{dI}{dx}\right)_i}\right]. \end{aligned}$$

See Figure 16(b) for an example of the detected edges in Figure 16(a).

3.3.3 Removal of Any Projected Edges

To keep the previously projected image from confusing the tracking algorithm (see Figure 16(a)), we remove any edges that could come from the projected image at the previous time steps. All detected edges are removed that lie within a fixed distance from any of the lines detected 3 frames before. (There is a 3 frame lag between image capture and image projection, one transferring the image to memory, one in processing, and one in the graphics buffer). See Figure 16(c) for an example. Notice how most of the edges from the projected paper in Figure 16(b) have disappeared in Figure 16(c).

3.3.4 Edge Grouping

We make a histogram of the edge pixels with respect to their angles $\theta_i \in [-\pi, \pi]$, and find 4 peaks that are separated from each other by roughly $\frac{\pi}{2}$. Then we collect all the edges for each of the 4 peaks whose angles are within a fixed distance from the peak. Figure 16(d) contains the histogram of the edge angles, and Figure 16(e) shows the edges that are close to the 4 peaks in the histogram. The colors in Figure 16(e) correspond to the peak to which the edges belong to: red = group 1, green = group 2, blue = group 3, and cyan = group 4.

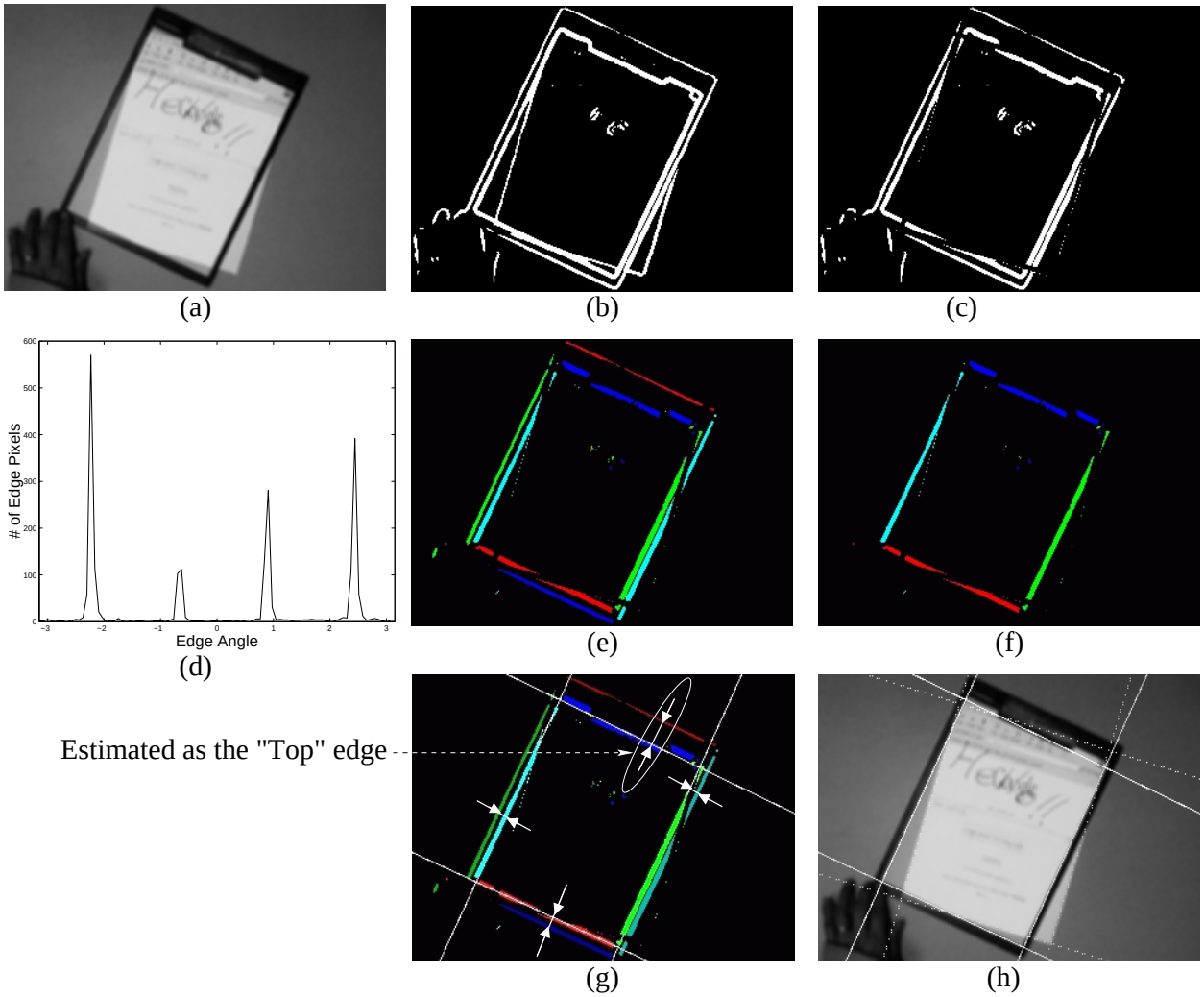


Figure 16: An illustration of the Tele-Graffiti paper tracking algorithm. First edge pixels are detected (b) in the captured image (a). They come from the paper, clipboard, hand, and the projection of the paper in the previous time step. Edge pixels that lie within a fixed distance from the previously detected lines are removed (c). Then a histogram of edge pixels is created with respect to their angles (d). Note that there are four peaks which correspond to the edges of both the paper and the clipboard. The edge pixels are then divided into four color coded groups (e). Most of the edges from the hand or writing on the paper are removed in this step. Next the edge pixels in each group are divided into two, the ones from the paper-clipboard edge (f) and the ones from clipboard-desk edge (omitted in (f)). Then, lines are fit to each group of paper-clipboard edge pixels and the paper orientation is estimated based on the average distance between the resulting line and its corresponding clipboard-desk edge pixels (g). (h) shows that the detected lines are well aligned to the real paper-clipboard edges. Shown in dashed lines are the previously detected lines which are used in (c) to remove the edges of the paper projected from the previous time step.

Tele-Graffiti's Robust Line Fitting Algorithm

(0) Set initial a, b in Equation (4) from the average angle of the edges so that the line is approximately perpendicular to the average angle.

Repeat (1)-(5) up to 5 times.

(1) If there are not enough edges, exit with an error.

(2) Calculate average X and Y coordinates of the edges.

(3) Set c so that the Equation (4) is satisfied with the average X and Y.

(4) Calculate the distance between each edge and the line.

(5) Exit the loop if the average of the distance is smaller than a constant threshold.

(6) Find 20% of most distant edges from the line and remove them.

And then:

(7) If the average distance is still larger than the threshold, result in an error.

(8) Calculate a, b and c from the remaining edges with linear least-squares.

Figure 17: An overview of Tele-Graffiti's robust line fitting algorithm.

3.3.5 Division into Paper-Clipboard and Clipboard-Desk Edges

Each group of edge pixels contains some edges between the paper and the clipboard and some edges between the clipboard and the desk (see Figure 16(c)). They are distinguished at this point by testing each edge to see whether it is directed towards or away from the center of the paper (which is estimated as the mean of the edge locations.) See Figures 16(e) and (f) for an example. Figure 16(f) contains the paper-clipboard edges.

3.3.6 Line Fitting

A robust line fitting algorithm is applied to each group of paper-clipboard edges. The overview of the algorithm is shown in Figure 17. Let us denote the line as Equation (4).

$$aX + bY + c = 0 \tag{4}$$

To avoid repeating costly least-square computations, the algorithm first estimates the slope of the line (a, b) from the average of the edge angles so that the line is approximately perpendicular to the edge angles.

$$\theta_{avg} = \frac{\sum_{i=1}^{n} \theta_i}{n}, \ a = \cos(\theta_{avg}), \ b = \sin(\theta_{avg})$$

Next, the algorithm iteratively performs outlier removal up to 5 times. It repeats the following: (1) c is calculated from Equation (4) setting X, Y to the average of the respective coordinate of the edge pixels, (2) all the distances between each edge and the line are calculated, (3) 20% of most distant edges are removed. This iteration stops if the average distance from the edges and the line becomes smaller than a constant threshold (success), or if there are not enough number of remaining edges to reliably fit a line (failure), or the number of iteration has passed the limit (failure). Finally, if the above iteration stopped successfully, least-square fitting is applied to the remaining edges to re-calculate the line parameters (a, b, c). See Figure 16(g) for the result of line fitting for the 4 groups of paper-clipboard edges.

3.3.7 Estimation of Paper Orientation

The four-fold ambiguity in the paper orientation is broken by computing the average distance of the clipboard-desk edges from the corresponding paper-clipboard line. This information, combined with the distances between opposite pairs of lines, is enough to break the symmetry. The "top" edge is the one with the larger average distance between the clipboard-desk edges and the paper-clipboard line (in the pair of lines separated by the larger distance.) See Figure 16(g) for an example of the ambiguity breaking algorithm.

3.3.8 Additional Background Light

As we depend on the clipboard-desk edges for determining the paper orientation (See Section 3.3.7), they must always be detected. In normal office illumination, however, the desk is often hardly visible because the ambient light is too dark compared to the projected light.

As we are using a black clipboard, the intensity difference of the clipboard-desk edge tends to be too small for the edge to be detected in the edge detection step (See Section 3.3.2), which makes the paper orientation determination difficult.

To avoid this problem, we project additional light using the projector onto the desk over the work area. The intensity of additional projection light is determined automatically at the system startup, by conducting a binary search over the range $[0, 255]$ to find the smallest intensity value which makes the intensity difference of the clipboard-desk edges larger than the constant threshold used in edge detection (See Section 3.3.2).

3.4 Communication between Tele-Graffiti Sites

3.4.1 The Sending and Receiving Threads

Sending and receiving is conducted simultaneously. Each Tele-Graffiti site opens two TCP/IP sockets, one of which is dedicated to sending, and the other to receiving. For this purpose, we have two communications threads, the Sending thread and the Receiving thread. Moreover, since it appears that Linux doesn't allow one process to both receive and send on TCP/IP sockets at the same time (even in different threads), we implemented the Sending thread as a separate process, rather than just as another thread. The details of the communications threads are as follows:

Sending Thread This thread continuously converts the most recent image of the paper at this site (the image to send) from YUV to RGB, compresses it, and transmits it along with the estimated local paper vertices in the image. As the paper detection cycle (30Hz) is faster than the typical image sending cycle and multiple updates to the image to send can occur during one image transmission session, the Sending thread just transmits the most recent image when it starts the transmission. This image is copied to a buffer to avoid being overwritten by the Paper Detection thread. See Appendix A.1 for our fast YUV to RGB conversion algorithm and implementation.

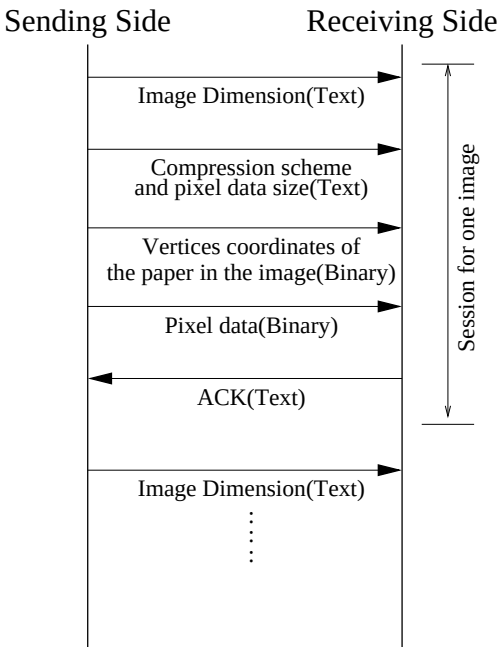


Figure 18: The Tele-Graffiti communication protocol is used with all three compression schemes. The sending side takes the initiative. First the sending side sends the image dimension (width $\times$ height $\times$ color bands), the compression scheme ('No compression' or 'JPEG' or 'Run-length'), and the number of bytes of pixel data to follow, in text format. It then sends the 17×17 quadrangle vertices in binary format (CPU native integer representation, 2.3 Kbytes). Finally the sending side sends the raw binary pixel data and waits for an ACK message from the receiving side. The receiving side replies with an ACK message each after it has successfully received (and decompressed) the whole image. After receiving the ACK message, the sending side starts another session.

Receiving Thread This thread waits for the arrival of images and paper vertices from the other Tele-Graffiti site. Upon arrival, the Receiving thread decompresses the image, updates the image to display and the remote paper vertices, and notifies the Drawing thread of the update.

See Section 3.4.3 for the details of image compression and decompression.

3.4.2 Communication Protocol

The communication protocol used in Tele-Graffiti is a very simple and efficient one. See Figure 18 for the details of the protocol.

3.4.3 Image Compression

As mentioned in Section 3.1, the image to send is typically around 40K $\sim$ 160K pixels: i.e. around 120K $\sim$ 480Kbytes for an RGB image. Assuming that the average is 240K bytes, it theoretically takes about 19msecs to transmit one raw image over a 100Base-T network. However, if we take into account the fact that the effective bit rate of a 100Base-T network is about 60Mbps (60%) and that two systems are communicating in both directions simultaneously on the same physical network (moreover, other systems may be using the same network), it would take longer than the 33msecs required for the desirable frame rate of 30Hz. Moreover, if the network speed is slower than 100Base-T, image transmission time would become significantly slower. Hence, it is necessary to compress the image to send to reduce the transmission time.

We tried a simple run-length compression algorithm whose typical compression rate ($\frac{\text{compressed size}}{\text{original size}}$) was around 20-40%, and JPEG compression with around a 5-15% compression rate. (We used JPEG library v6.2 on Linux, which is distributed by the Independent JPEG Group.) These compression rates suggest that the transmission time would be as short as a few msecs over 100Base-T. We confirmed that JPEG compression doesn't affect the image quality: the compressed image is not distinguishable from the original one by human eyes. See Figure 19 for a comparison of an image before and after JPEG compression.

Table 2 shows timing results for image transmission with various compression schemes (no compression, run length and JPEG compression), over both 100Base-T and 10Base-T networks. Note that the "Communication" time is measured as the time it took to call the system call "sendto()," not the actual time in which the data is sent over the network. Comparing these results, we observe the following facts.

1. Transmission without compression is greatly affected by the network speed, while compressed transmission is less affected.

   Transmission without compression over 10Base-T takes more than 10 times (actu-

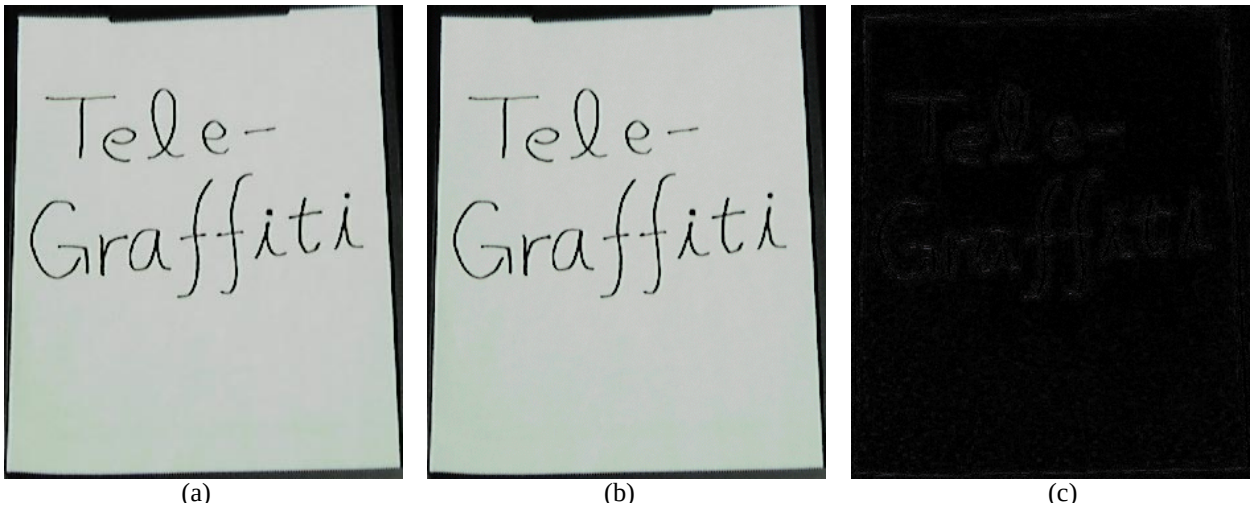


Figure 19: Comparison of a JPEG-compressed image and the original image. (a) Original image. (b) Image after being compressed and decompressed using IDG's JPEG library with "quality=95." The compression rate for this image is 8.4%. (c) Difference between (a) and (b), enlarged by two times.

Table 2: Image transmission timing results with various compression schemes over 100Base-T and 10Base-T networks, measured in milliseconds.

Compression Scheme (typical compression rate)		no compression		run length (20-40%)		JPEG (5-15%)	
Network		100Mbps	10Mbps	100Mbps	10 Mbps	100Mbps	10Mbps
Image transmission		36.2	769.3	59.2	200.2	64.6	60.4
	Communication	36.2	769.3	7.9	149.0	2.0	1.7
	Compression	—	—	51.3	51.2	62.6	58.7
Wait for the ACK		8.7	128.4	7.9	108.2	14.0	16.1
Vertices transmission etc.		7.0	32.3	16.9	28.9	6.5	9.7
Total time per one image		51.9	930.0	84.0	337.3	85.1	86.2

ally about 20 times) longer than over 100Base-T. This is best explained by "network congestion." As two independent data streams are flowing in opposite directions the collision of packets from both sites is very likely to occur, which significantly reduces the network throughput when the network is congested.

2. Most of the time in transmission with JPEG compression is dedicated to CPU compression/decompression time; 58.7+16.1msec out of 86.2msec for 10Base-T.

   The compression time can be reduced if we use faster CPUs. For example, if we assume that we could make the compression/decompression 4 times faster, "Compression" and "Wait for the ACK" time would be reduced by a quarter (most of the "Wait for the ACK" time with the JPEG compression scheme is spent waiting for the receiver to finish decompressing), and the total image transmission time can be estimated to be only 27.7msec compared to 86.2msec. It is obvious that we can get a faster transmission rate with JPEG compression rather than no-compression if we use the most recently available CPUs.

In summary, we can conclude that we can choose the best compression scheme from no compression to JPEG compression, depending on the relative CPU power and network speed.

3.4.4 Suspension of Image Transmission

Users may move the paper during run-time operation, which causes the camera to capture a blurred image. They may also even remove the paper temporarily. It makes no sense in these cases to send the most recent paper image to the other side. Therefore, the Paper Detection thread updates the image to send only when: (1) the paper vertices were successfully detected, (2) they are within the camera view, and (3) the paper has not moved for a while. Otherwise, image transmission is suppressed to avoid the transmission of poor quality images.

3.5 Software System Timing Performance

Timing results for the Drawing thread and the Paper Detection thread are shown in Tables 3 and 4. On dual 450MHz Pentium PCs, these threads operate (comfortably) at 30Hz.

Table 3: Timing results for the Drawing thread. The Drawing thread operates at over 30Hz and even so most of the cycle is spent idling, waiting for the update to the shared data.

Steps	Timing (msec)	Occurrence per loop	Average per loop
Update OpenGL texture buffer	9.8	24.4%	2.4
Image re-mapping (drawing)	0.8	75.6%	0.6
Wait for the update*	–	–	26.2
Overall Drawing thread loop	–	–	29.2

Timing results for the communications threads are shown in Table 5 (Sending thread) and Table 6 (Receiving thread). As we discussed in Section 3.4.3, the computational cost of compressing and decompressing the image is a substantial proportion of this time. On faster PCs, the transmission threads could run far faster. In these tables, all timing results are measured in real time, not a CPU time. The steps marked with "*" (asterisk) spend most of their time idling (i.e. waiting for events or resources) rather than actually computing.

4 A Hand-Based User Interface

For a remote sketching system like Tele-Graffiti, it would be very useful to have functions to record the current session, summarize it, and browse through previously summarized and archived sessions. For example, the remote students described in Section 1.1 may want to go over a Tele-Graffiti session that they had with the professor as a study aid before the final exam. Moreover, they would probably want to step through the session at their own pace, selecting a frame at a time. To enable this, we need to know:

1. Whether the user wants the session to be summarized and saved or not, and whether they are in "normal session mode" or "playback mode."

Table 4: Timing results for Paper Detection thread. Paper detection operates comfortably at 30Hz. Again 10msec per loop is spent idling waiting for image capture.

Steps		Timing (msec)	Operations per Pixel
Image capture* (Wait for the next frame)		10.6	—
Paper detection		19.7	115.4
	Down-sampling (YUV to Grayscale conversion is included)	(2.2)	(12.9)
	Smoothing	(4.0)	(23.4)
	Edge detection	(5.4)	(31.6)
	Removal of projection edges	(1.1)	(6.4)
	Edge grouping	(1.4)	(8.2)
	Edge division	(2.0)	(11.7)
	Line fitting	(0.7)	(4.0)
	Estimating paper orientation	(0.2)	(1.1)
	Other	(2.7)	(15.8)
Update of the image to send		3.0	17.6
	Image cropping	(2.6)	(15.2)
	Other	(0.4)	(2.4)
Overall Paper Detection loop		33.3	—

The user must have a way to turn on and off the various system functions such as whether to record or not, and whether to playback or not.

2. When to record an image.

   As Tele-Graffiti's paper detection (capture) loop runs at 30Hz (see Section 3.5), it is not realistic to record all of the captured frames during the session. Thus we have to determine which frames to record. One appropriate candidate is to record a frame

Table 5: Timing results for the Sending thread (with JPEG compression over a 100Base-T network). Overall the Sending thread operates at 8Hz. With JPEG compression, we get roughly the same results for over 10Base-T network. See also Table 2.

Steps		Timing (msec)
Memory copy etc.		2.0
YUV to RGB conversion		7.1
Image transmission		85.1
	Communication	(2.0)
	Compression	(62.6)
	Wait for the ACK*	(14.0)
	Vertices transmission and others	(6.5)
Wait for update*		31.0
Overall Sending thread loop		125.2

Table 6: Timing results for the Receiving thread (with JPEG compression over a 100Base-T network). The Receiving thread also operates at 8Hz. With JPEG compression, we get roughly the same results for over 10Base-T network. See also Table 2.

Steps	Timing (msec)
Wait for transmission start*	45.8
Receiving*	32.8
Decompression	39.6
Memory Copy	4.0
Others	3.0
Overall Receiving thread loop	125.2

whenever the user's hand which has been over the paper moves away from the paper. It is very likely that this occurs when the user has finished a drawing segment.

3. When the system is in playback mode, which recorded frame to playback.

The user must have a way of selecting the one frame that they want to playback from the multiple recorded frames.

We have designed and implemented a user interface for Tele-Graffiti in accordance with these requirements. We use two kinds of triggers to activate user interface functions: "hand over paper" events and interactions of the hand with user interface objects (UI objects). Hand over paper events occur when the system determines that the user places their hand over the paper or when they retract their hand from over the paper. UI objects are rectangle shaped projections in the work area which enable users to invoke pre-defined commands using their hands. Currently we have implemented two types of UI objects: "Button" and "Slider." Button objects are used to toggle a "mode". Slider objects are used to select a value from a range of values.

The three requirements described above that we need to build a summarization and playback system are then satisfied as follows. Requirement 1 is satisfied using Button objects. Requirement 2 is satisfied using hand-over-paper events. Finally, Slider objects are used to meet requirement 3. Properties of the UI objects such as type, geometry, color, and associated commands are defined in the "UI objects definition file" which is read at system startup. (See Appendix B for the details of the definition of UI objects.)

We now describe our hand tracking algorithm. Afterwards we describe our hand-over-paper detection algorithm and the details of Button and Slider objects. Next, we describe the complete user interface by means of using it to implement a record, summarization, and playback system. Finally we present timing results for the user interface system.

4.1 Hand Tracking

There are several possible ways of tracking the user's hands. One possibility is to use a color-based hand tracking algorithm. We chose not to use a color-based algorithm because such algorithms are generally not sufficiently robust to the variation in the color of the user's

Tele-Graffiti Hand Tracking Algorithm	
(1)	Background subtraction and thresholding
(2)	Compute connected components
(3)	Determine the hand component
(4)	Localize the hand/finger "tip"

Figure 20: An overview of the Tele-Graffiti hand tracking algorithm.

hand and to the lighting conditions, as pointed out in [Sato *et al.*, 2000]. An additional complication is that some LCD projectors such as the PLUS U2-1080 [Plus Corporation, 2000] time-multiplex color. Without synchronizing the shutters of the projector and the camera, the color of the hands varies from frame to frame.

Another way of detecting the user's hands is to use an infra-red camera [Sato *et al.*, 2000]. The color projected by the projector is then not important because the camera primarily only "sees" the heat radiated by the user's hands.

We chose to base our hand detection algorithm on background subtraction because it is fast, robust, and does not require any special hardware such as an infra-red camera. Figure 20 contains an overview of our hand tracking algorithm. We discuss each step in turn in detail in the following sections.

4.1.1 Background Subtraction

We have to consider the following problems during background subtraction.

1. The paper and clipboard are on the desktop and they may move from frame to frame.

2. Because we are using the camera's auto gain control (AGC), the background intensity varies depending on what else is placed on the desktop. For example, when the paper is placed on the desktop the average intensity increases. As a result, the AGC reduces the gain of the camera and so the intensity of the background is reduced.

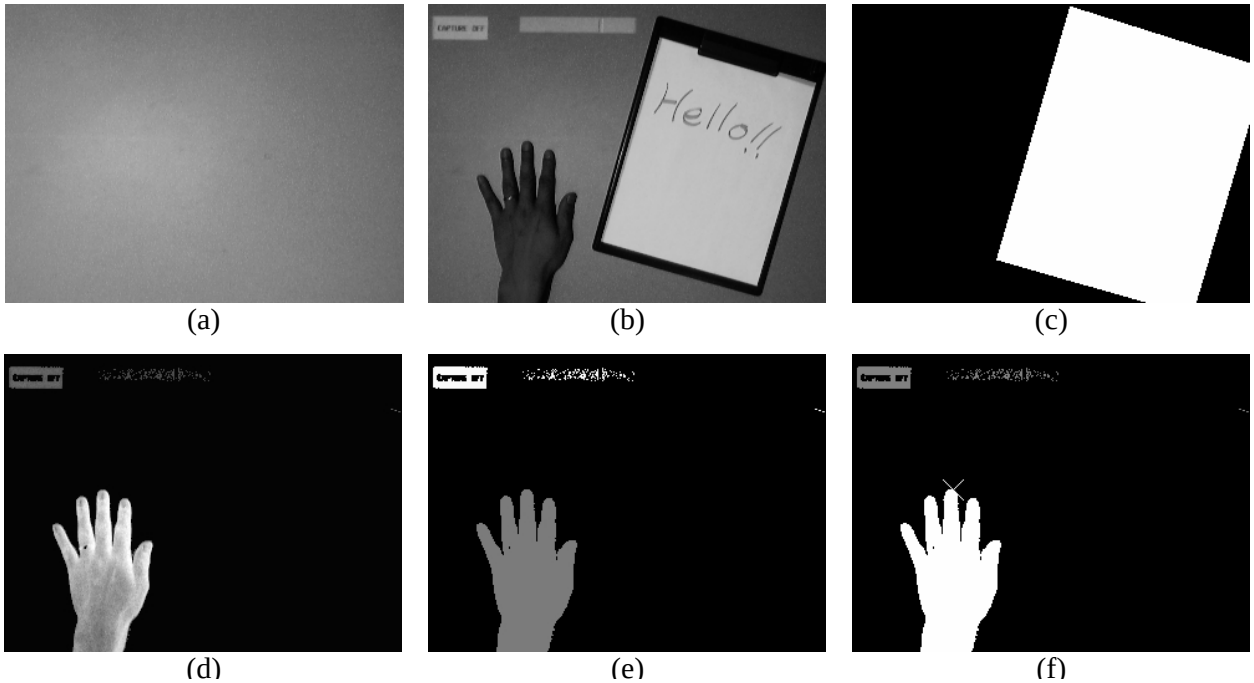


Figure 21: An illustration of the Tele-Graffiti hand tracking algorithm. (a) A background image captured at system startup. (b) An image captured while the system is running, which includes the paper, clipboard, user's hand, and projected UI objects. Note the difference in brightness of the background between (a) and (b) caused by the camera's AGC. (c) A mask image which covers the paper and clipboard, created based on the paper vertices detected in (b). (d) Result of background subtraction with intensity difference compensation. (e) Connected components found in (d). Different components are shown with different intensities. (f) Result of hand component determination. The detected hand is shown as the brightest object, and its tip denoted with an "X."

To solve these problems, our background subtraction algorithm includes the following steps.

Prepare the Background Image: A grayscale background image $I^0(x, y)$ is captured at system startup before anything has been placed on the desktop. The average intensity I^0_{avg} is calculated from the background image. See Figure 21(a) for an example background image.

Create a Mask for the Paper and Clipboard: Create a mask which covers the paper and clipboard using the detected paper vertices. This mask is used in background subtraction. See Figure 21(c) for an example mask image.

Calculate the Average Background Intensity: In order to compensate for the differ-

ence in background intensity caused by the camera's AGC, calculate the average intensity I^i_{avg} of the current frame image $I^i(x, y)$, excluding the paper and clipboard area. Either the mean or the median could be used. We found the mean to be sufficiently robust.

Pixel by Pixel Subtraction: The pixel by pixel binary difference image $D(x, y)$ is computed from the current image $I^i(x, y)$ and the background image $I^0(x, y)$ with predefined threshold C and average difference compensation, as is shown in the following equation.

$$\begin{aligned} d(x,y) &= I^i(x,y) - I^0(x,y) - (I^i_{avg} - I^0_{avg}) \\ D(x,y) &= \begin{cases} 0 & (\text{if } d(x,y) \leq C) \\ d(x,y) & (\text{if } d(x,y) > C) \end{cases} \end{aligned}$$

See Figure 21(d) for an example result.

In our implementation, $D(x, y)$ is divided by two and shifted by 128 so that the result can be represented in an 8 bit grayscale image. Thus, pixel intensity 128 means no difference. We also utilize MMX functions to optimize the speed of computing the average pixel intensity and computing the difference image. Both of these steps also require taking into account the mask image. See Appendix A.4 for more details.

4.1.2 Computing Connected Components

Next, we find the connected components in $D(x, y)$, the result of background subtraction. We use 4-connectedness [Horn, 1996]. Each connected component is given a unique label $[1, 255]$ and the result is represented as another grayscale image where the pixel intensity is the label of the component to which the pixel belongs. Figure 21(e) shows an example.

4.1.3 Determining the Hand Component

We determine which of the components found in the previous step is the hand component using knowledge that the user's hand:

- should always touch the image boundary, and
- is larger than most of the other objects on the desktop (except for the paper and clipboard, which are eliminated using the paper mask.)

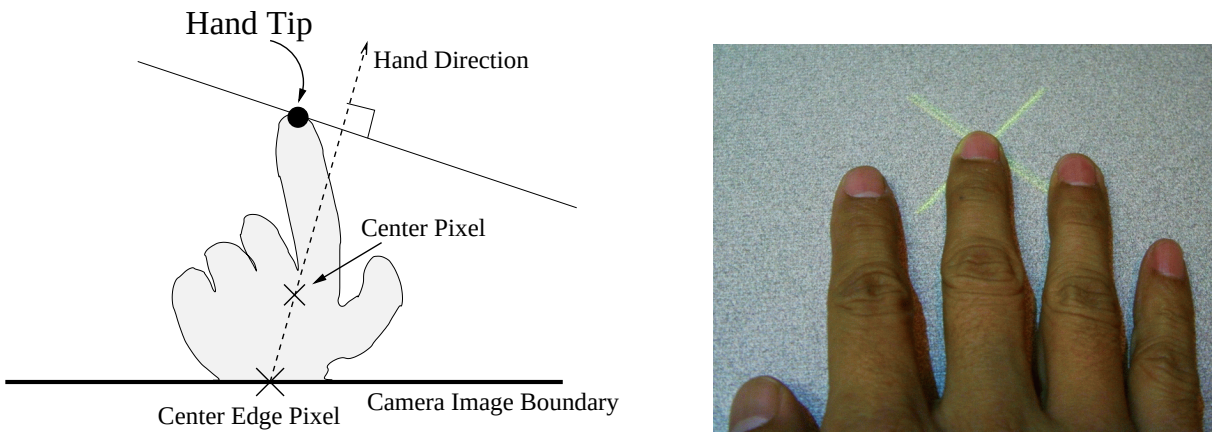
Thus, the algorithm to determine the hand component is to find the largest component that intersects the camera image boundary. See Figure 21(f) for an example.

4.1.4 Hand Tip Localization

Once the hand component has been determined, we then determine its "tip": i.e. where it is pointing. An "X" is displayed on the work area at the detected tip location to show the user that the hand has been detected correctly. The hand tip is located with the algorithm below. See also Figure 22 for an illustration of the algorithm.

1. Compute the major direction of the hand from the following two points:
   (a) the "center edge pixel", i.e. the mean of the points where the hand component touches the camera image boundary
   (b) the "center pixel", i.e. the mean of the XY coordinates of the pixels in the hand component
2. Find the farthest pixel from the edge pixel in the hand component in the hand direction.

See Figure 21(f) for an example of hand tip localization.



(a) (b)

Figure 22: Hand tip localization. (a) First compute the orientation of the hand component from the "center edge pixel" and the "center pixel." Then find the farthest pixel from the edge pixel in the hand direction. (b) The detected hand tip is displayed on the work area with a yellow "X."

4.2 Hand Over Paper Detection

In general, it is very hard to determine whether the user is drawing or not. We assume that if they have their hand over the paper they are drawing. If their hand is away from the paper, they are clearly not drawing. Once we have detected the paper and found the hand component, it is then easy to determine whether the hand is over the paper or not. Our algorithm for hand-over-paper detection is as follows:

1. Create a "hand sensitive" mask around the paper.

   The hand sensitive mask is created based on the estimated paper vertices, just like the paper mask for the background subtraction step (see Section 4.1.1). The hand sensitive mask is slightly larger than the paper mask, however. Since no connected components exist within the paper mask because the background subtraction is fixed at 0 there, the effective "hand sensitive" area is the area between the larger mask and the paper mask. See Figure 23(b) for an illustration.

2. Intersect the hand component with the hand sensitive mask.

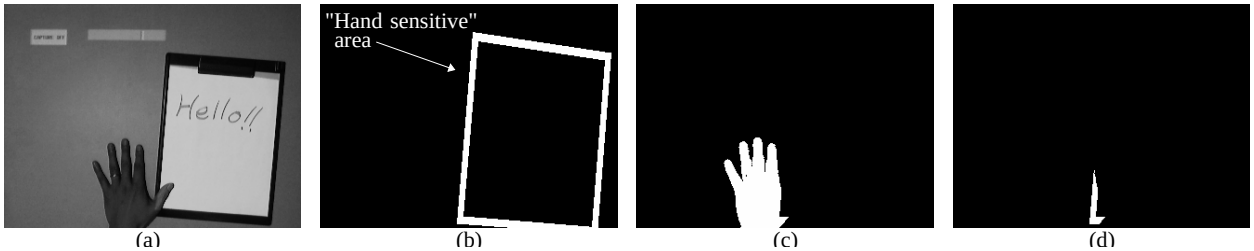


Figure 23: Hand-over-paper detection. (a) Captured image. (b) "Hand sensitive" area. A mask is created based on the paper vertices which is slightly larger than the paper mask for background subtraction (see Figure 21(c)). The area between these two masks is the hand-sensitive area. (c) Detected hand component. Note that the hand is partly masked by the paper mask in the background subtraction step. (d) The result of AND'ing or intersecting (b) and (c). If there are any pixels in the result, the hand is over the paper, and vice versa.

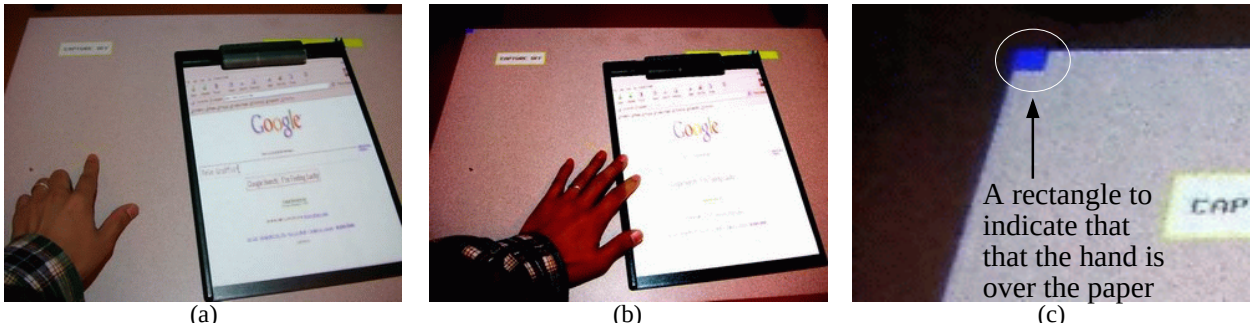


Figure 24: Results of Hand-over-paper detection. (a) Hand detection works simultaneously with paper tracking. (b)(c) When a hand is detected over the paper, a small blue rectangle is displayed in the top left corner of the work area so that the user knows the system has detected the event.

> If there is at least one pixel from the hand component within "hand sensitive" area, the hand is over the paper. See Figure 23(d) for an illustration.

Figure 23 illustrates our hand-over-paper detection algorithm. In our implementation, when the hand is over the paper, the user is notified with a small blue rectangle displayed in the top-left corner of the work area. See Figure 24 for an example.

4.3 Button Object

Button objects are typically used to invoke a one-time action or toggle a system mode. Each Button object has one of the following states: *normal*, *focused*, and *selected*. The state of

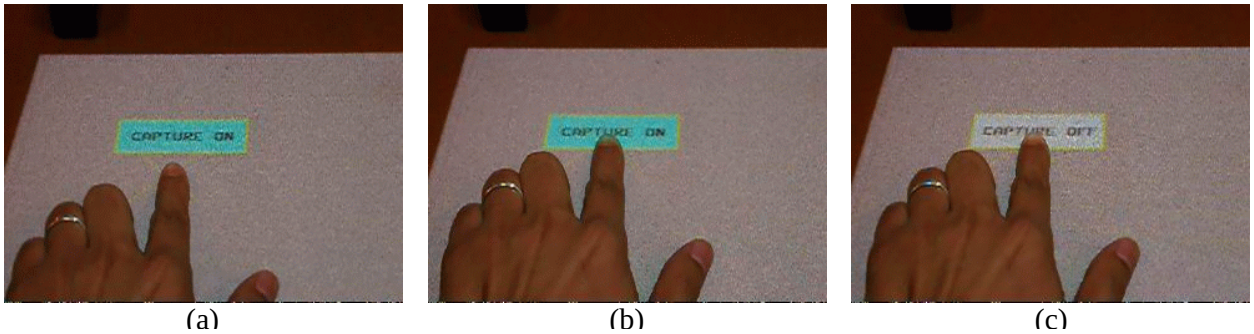


Figure 25: Interaction with a Button. (a) The Button in its "normal" state. When the hand is detected over the Button object (b), its state becomes "selected"; changes its appearance (color and text label) (c), and invokes a pre-defined command (not shown).

a Button object is updated in real-time depending on the interaction with the user's hand. Each Button object's state starts as "normal" and becomes "focused" when the hand moves over the Button object. After the hand remains over the Button for a certain period of time it then becomes "selected" and the command that is associated with the object is invoked. Users can specify separate "text labels" to be drawn in the center of the Button object, for the normal/focused state and for the selected state. It is also possible to specify the behavior of Button objects when the hand leaves the Button after it has been selected: either (1) the Button returns to the "normal" state, or (2) the Button keeps the selected state. See Appendix B for more details of how Button objects are defined. Behavior (1) is useful for invoking a one-time action, while behavior (2) is useful for mode-toggling. For case (2), the object returns to the normal state when it is selected again. See Figure 25 for example images of a user interacting with a Button object.

Hand over Button detection operates in a similar way to hand-over-paper detection: the hand component from background subtraction (see Section 4.1.2) is examined and it is determined if there is at least one pixel in the "hand sensitive" area, a rectangular area slightly larger than the object itself. Figure 26 illustrates hand-over-button detection.

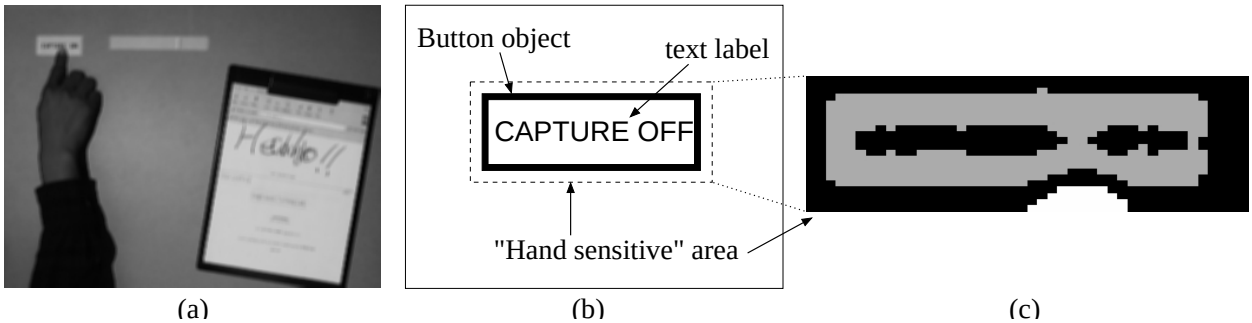


Figure 26: Hand over Button detection. (a) Captured image. One finger of the hand is placed over the Button. (b) An illustration of the "hand sensitive" area of the Button. The hand sensitive area is a rectangular area slightly larger than the object itself. (c) The hand component in the "hand sensitive" area. As there are pixels from the hand component (the brighter pixels) in the hand sensitive area, it is determined that the hand is over the Button. Note that the hand component does not extend into the object area (the darker pixels) even when the hand tip is placed near the center of the object because the projected light needed to draw the object brightens the hand tip. This does not affect hand over Button detection. All that is needed for robust hand over Button detection is that the hand sensitive area be slightly larger than the Button itself.

4.4 Slider Object

Sliders are used to vary a numerical parameter within a range. Each Slider object holds its current value within the range $[0, 100]$ although any other range could be used. A "bar" is drawn in the slider object to denote the current value. The leftmost end of the Slider denotes 0 (the minimum value), the rightmost 100 (the maximum value). See Figure 27 for an example of a user interacting with a Slider.

The Slider object has a state mechanism just like the Button object. When the Slider is in the "selected" state it continues to:

1. estimate the position of the hand,
2. compute the value of the Slider parameter from the position,
3. update the location of the Slider bar according to the parameter value, and
4. notify the system of the parameter value.

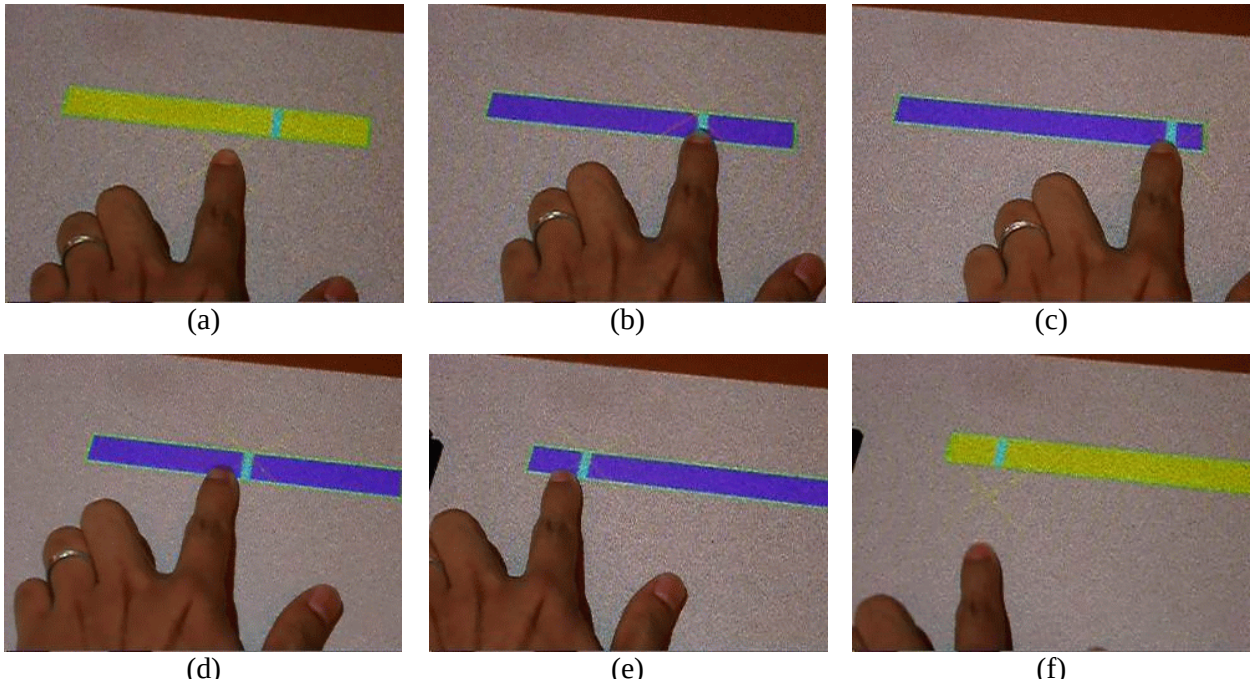


Figure 27: Interaction with a Slider. (a) The Slider in normal state. (b) When the user's hand is detected over the Slider, it is activated and changes its appearance (color). The Slider continues to change the value of its parameter depending on how the finger moves (c)(d)(e). The system is continuously notified of the change to the Slider parameter until the hand leaves the Slider (f).

The user interface not only detects the hand component within the Slider's "hand sensitive" area in the same way as for Button objects, but also estimates the horizontal position of the hand x_h. This value is computed by averaging the X coordinates of the pixels from the hand component within the "hand sensitive" area:

$$x_h = \frac{\sum_{i=1}^{N_h} x_h^i}{N_h}$$

where N_h is the number of pixels from the hand component in the hand sensitive area, and x_h^i is the X coordinate of each such pixel. Note that we do not include any pixels within the object area here because the image of the hand within the projected object area is not stable enough to rely on. See Figure 28(d). The value of the Slider parameter then computed as:

$$\frac{x_h}{w} \times 100$$

where w denotes the width of the hand sensitive area in pixels. Figure 28 illustrates hand

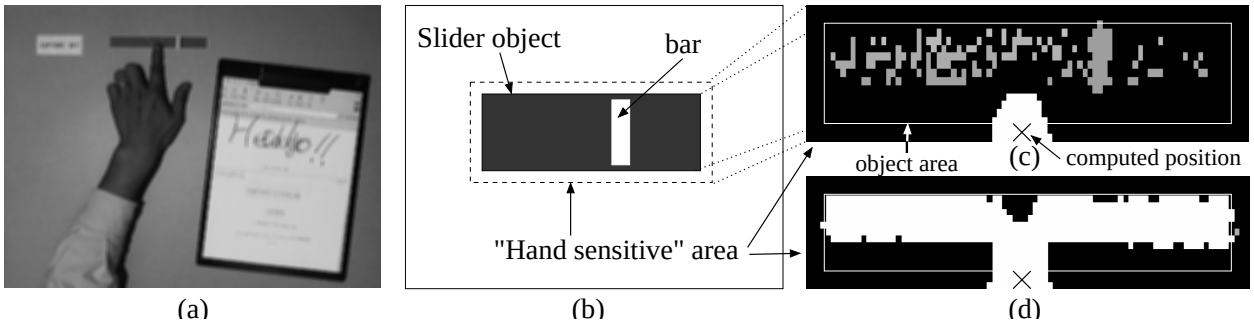


Figure 28: Hand over Slider detection and computation of the hand position. (a) Captured image. One finger of the hand is placed over the Slider object. (b) An illustration of the "hand sensitive" area of the Slider, the same as for the Button. (c) The hand component in the "hand sensitive" area. As pixels from the hand component (the brighter pixels) are found in the hand sensitive area, it is determined that the hand is over the Slider. In addition, the hand position is computed by averaging the X coordinates of the pixels from the hand component between the object area and the "hand sensitive" area, which is shown by a small "X." We don't use the pixels in the object area to compute the hand position because the image of the hand within the object projection area is unstable. In some cases the hand and the Slider are recognized as a single connected component (d), while in (c) the components of the hand and the Slider are separated.

over Slider detection and the computation of the hand position.

4.5 Application: Session Summarization and Replay

We have developed an application to demonstrate the Tele-Graffiti user interface system: session summarization and replay. Using this application, users can record summarized frames during a session and replay desired frames later. In addition, they can control all the functions using only their hands. In this section we describe the details of the application.

4.5.1 Session Summarization Using Hand Over Paper Detection

As we described in Section 4, we know when to record a frame based on hand over paper detection. When there are two Tele-Graffiti sites, we have to take into account both hands at the two sites: we record a frame only when one of the user's hand moves away from the paper, while the other user's hand is away from the paper. Thus each Tele-Graffiti site maintains the state of both hands at both sites. See Figure 29. For this purpose, the Tele-Graffiti sites

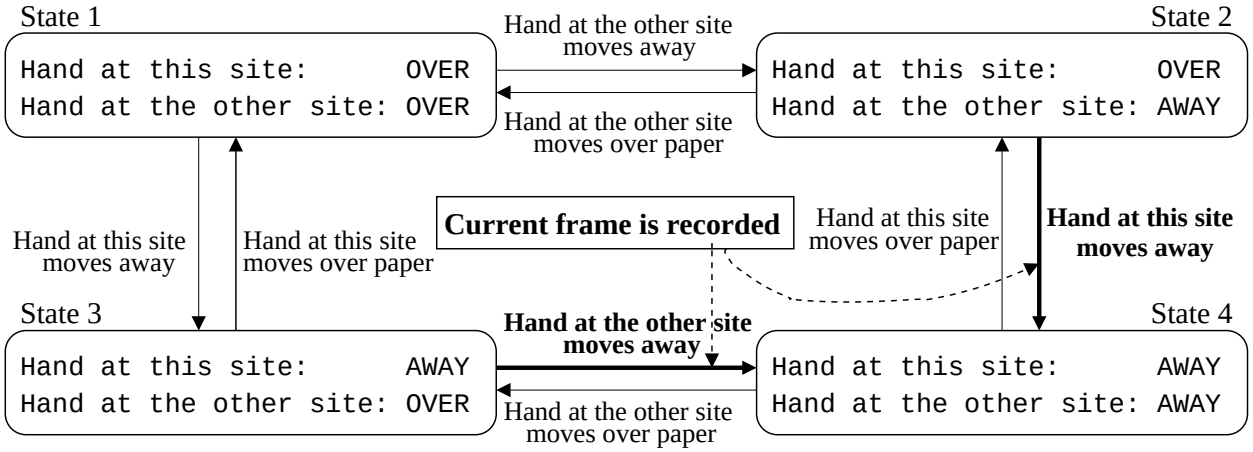


Figure 29: A state diagram of the users' hands at the two sites. The user interface system maintains the state of the hands at both sites while the "record mode" is on. The current frame is recorded whenever one of the hands moves away from the paper and the state becomes "State 4": i.e. a frame is recorded only when there are no hands over the paper.

send to each other the result of hand-over-paper detection along with the paper images. In Figure 29, the current frame is recorded whenever the state changes from State 2 to State 4 or from State 3 to State 4.

In our application, the users turn on the "record mode" with a Button before the session which they want to be summarized (see Figure 30(b)). Then the Tele-Graffiti user interface system saves summarized frames automatically, and the number of saved frames is shown in another Button (see Figure 30(d)). See Figure 33 for the 12 images which are saved during the 90 second session illustrated in Figures 31(g-u).

4.5.2 User Controlled Replay Using a Slider

After the recorded session, the users can browse through the saved frames with a Slider to review the session. Whenever the value of the Slider's parameter is changed, Tele-Graffiti projects the corresponding frame onto the paper. The corresponding frame is chosen from the saved frames based on the Slider's value and the number of the saved frames. For example, if there are 17 saved frames and the Slider's value is 39 in $[0, 100]$, the 7th frame is chosen since $1 + 16 * 39/100 = 7.24$.

In our application, the Slider appears on the desktop when the users turn on the "playback mode" with a Button (see Figures 30(c) and (e)). In the "playback mode," the projector suspends projecting the paper image transmitted from the other site and instead projects the saved frame that the user has chosen. The user can then change the frame to review by moving the Slider bar with their hand (see Figure 30(f)). Figures 32 illustrates how the summarized frames can be browsed using the Slider.

4.5.3 Complete System Demo

The UI objects used in the session summarization and replay application are shown in Figures 30(a-f). There are 4 UI objects: 3 Buttons and 1 Slider. The Buttons are used to: toggle the "record mode," toggle the "playback mode," and indicate the current number of saved frames. The last one doesn't interact with the user's hand because no pre-defined commands are associated with it. See Appendix B for defining user interface objects. The Slider is invisible at system startup and appears on the desktop when the user has turned on the "playback mode." Then it is used to choose the frame to display out of the saved frames.

Figures 31, 32 and 33 contain a collection of frames from a movie demonstrating our session summarization and replay application. Figure 31 shows how the session is recorded and Figure 32 shows how the user browses through the summarized session. Figure 33 contains the summarized (saved) frames during the session. The complete video is available on the Tele-Graffiti website at the URL http://www.cs.cmu.edu/~simonb/telegraffiti-/movies/uidemo.mpg. In the demonstration, the session lasting 90 seconds (2700 frames) was summarized in 12 frames, a compression ratio of $\frac{1}{225}$.

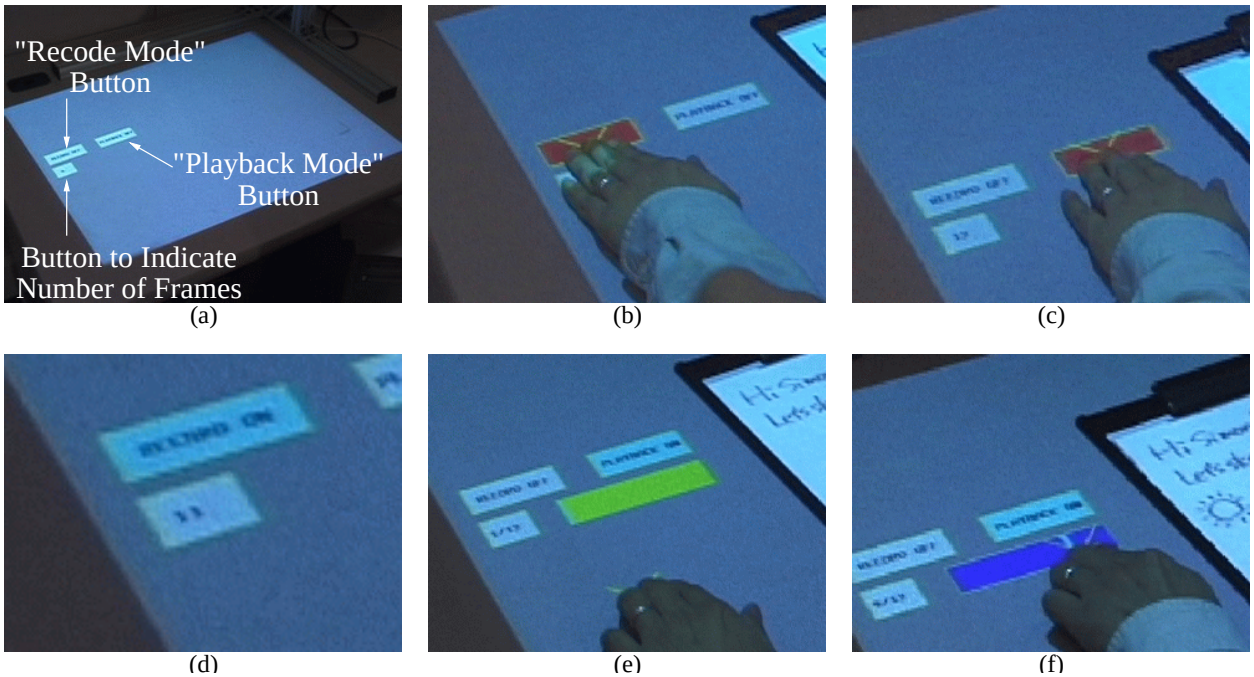


Figure 30: The UI Objects used in the session summarization and replay application. (a) At system startup 3 Buttons are displayed: the one on the top-left is used to toggle the "record mode," the second one on the top-right to toggle the "playback mode," and the third one on the bottom to indicate the number of currently saved frames. (b) The user turns on the "record mode" with the top-left Button. (c) The user turns on the "playback mode" with the top-right Button. (d) The number of frames (11 in this image) is indicated by the bottom Button. (e) When the user turns on the "playback mode," a Slider appears on the desktop. (f) The user changes the frame to display by moving the Slider bar with their hand.

4.6 User Interface Timing Results

Table 7 shows timing results for the Tele-Graffiti hand based user interface on 320×240 grayscale images. The user interface is implemented in the same thread as paper tracking (the Paper Detection thread). The total calculation time of 9.2msec is within the idle time waiting for the next frame (see Table 4): i.e. adding user interface processing to the Paper Detection thread doesn't affect the total system performance. Overall, paper tracking and UI processing operate at 30Hz on a dual 450MHz Pentium-II machine.

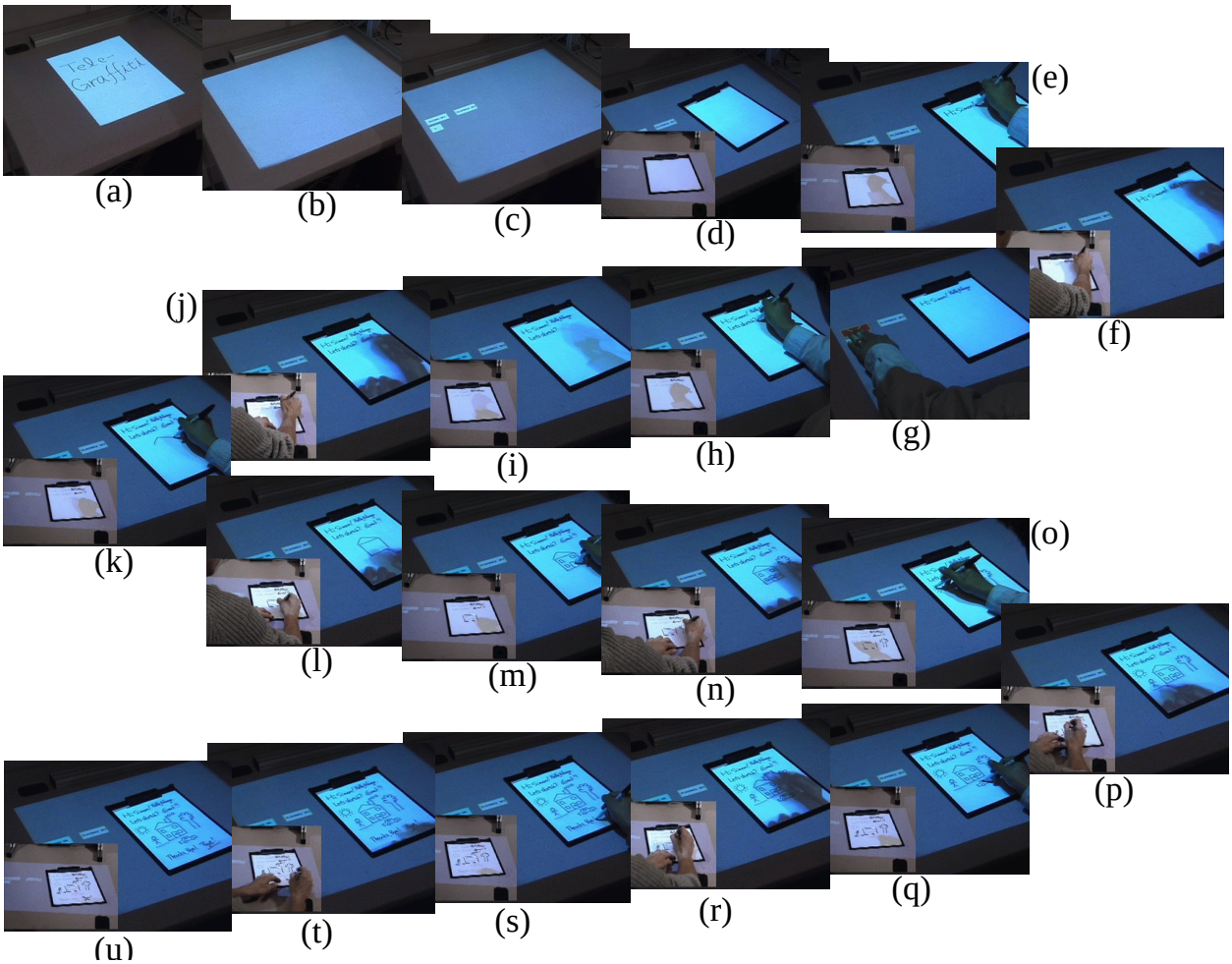


Figure 31: Session summarization and replay. (part 1 of 2, continued in Figure 32). (a) The Tele-Graffiti logo image which is used for geometric calibration (see Section 2.5) is displayed at system startup. (b) Before anything is placed or projected on the desktop, the background image is captured (see Section 4.1.1). (c) After the background image is captured, UI objects are displayed. There are three Buttons. The top left Button is used to toggle the "record mode" and the top right Button to toggle the "playback mode." These modes are initially set to "off." The small Button on the bottom does nothing but indicate the number of saved frames. (d) Paper is placed on the desktop at both sites. The scene of the other site is superimposed on Figures (d-f) and (h-u). (e)(f) Two users greet each other before starting a sketch session. (g) One user turned on the "record mode" with a Button. (h-t) The users sketch with the "record mode" on. Whenever the system detects that both hands are moved away from the paper, the current frame is saved. (u) The final sketched image after the session. The complete video is available on the Tele-Graffiti website at the URL http://www.cs.cmu.edu/~simonb/telegraffiti/movies/uidemo.mpg.

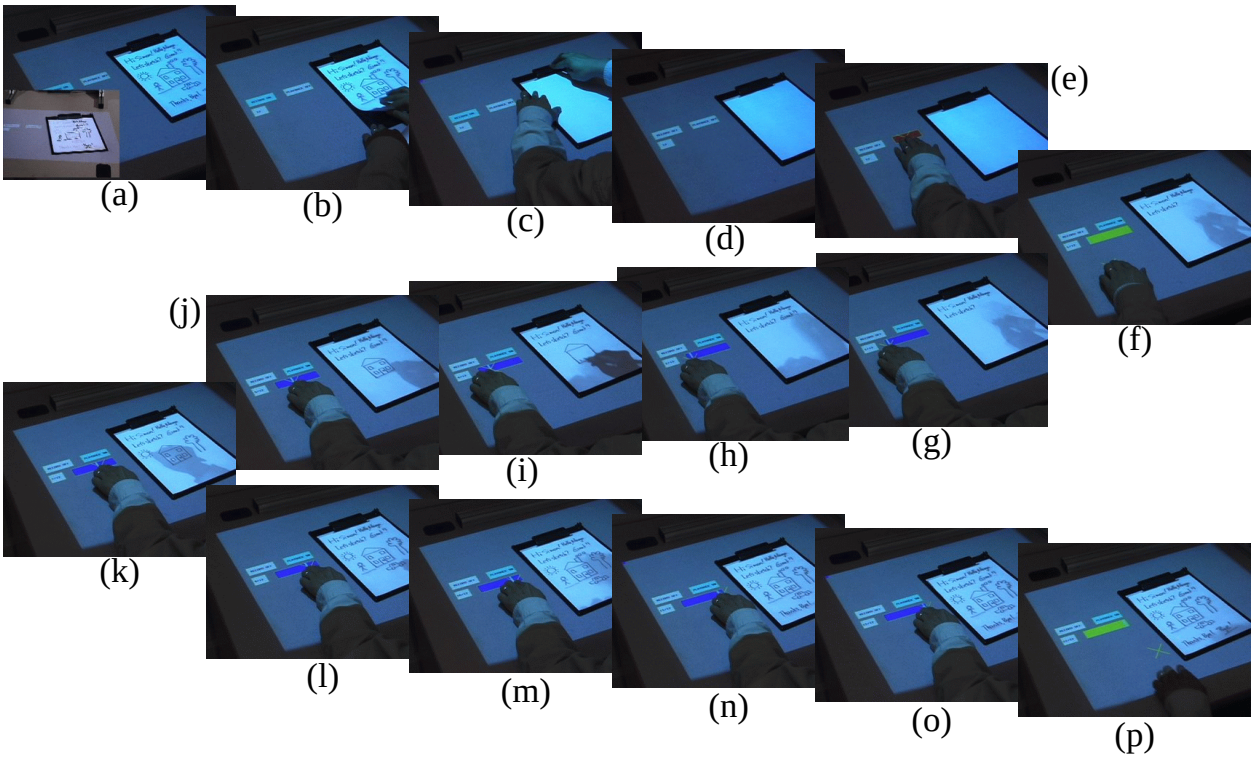


Figure 32: Session summarization and replay (part 2 of 2, continued from Figure 31). (a) The final sketched image after the session. (b-d) The user replaces the sketched paper with a blank paper in preparation to review the session. (e) The "playback mode" is turned on with a Button. (f) A Slider is projected onto the desktop. (g-o) The user browses through the saved frames with the Slider. As he moves the slider bar, the projected frame changes accordingly. (p) The final sketched image is projected onto the paper after browsing.

5 Feedback Analysis and Image Separation

Tele-Graffiti contains an obvious feedback loop, as do similar two-way communication systems such as the Xerox "Double DigitalDesk" [Wellner, 1993] and INRIA's "MagicBoard" [Hall *et al.*, 1999]. The projectors radiate light that is imaged by the cameras. The resulting images are passed on to the projectors at the other sites, where they are imaged again. This feedback loop may or may not be stable. It is possible that the projected image may get brighter and brighter until the camera saturates resulting in an unusable system. Alternatively the system might oscillate. Finally, there is the possibility of "visual echoing."

In this section we analyse the Tele-Graffiti feedback loop. Much of our analysis is

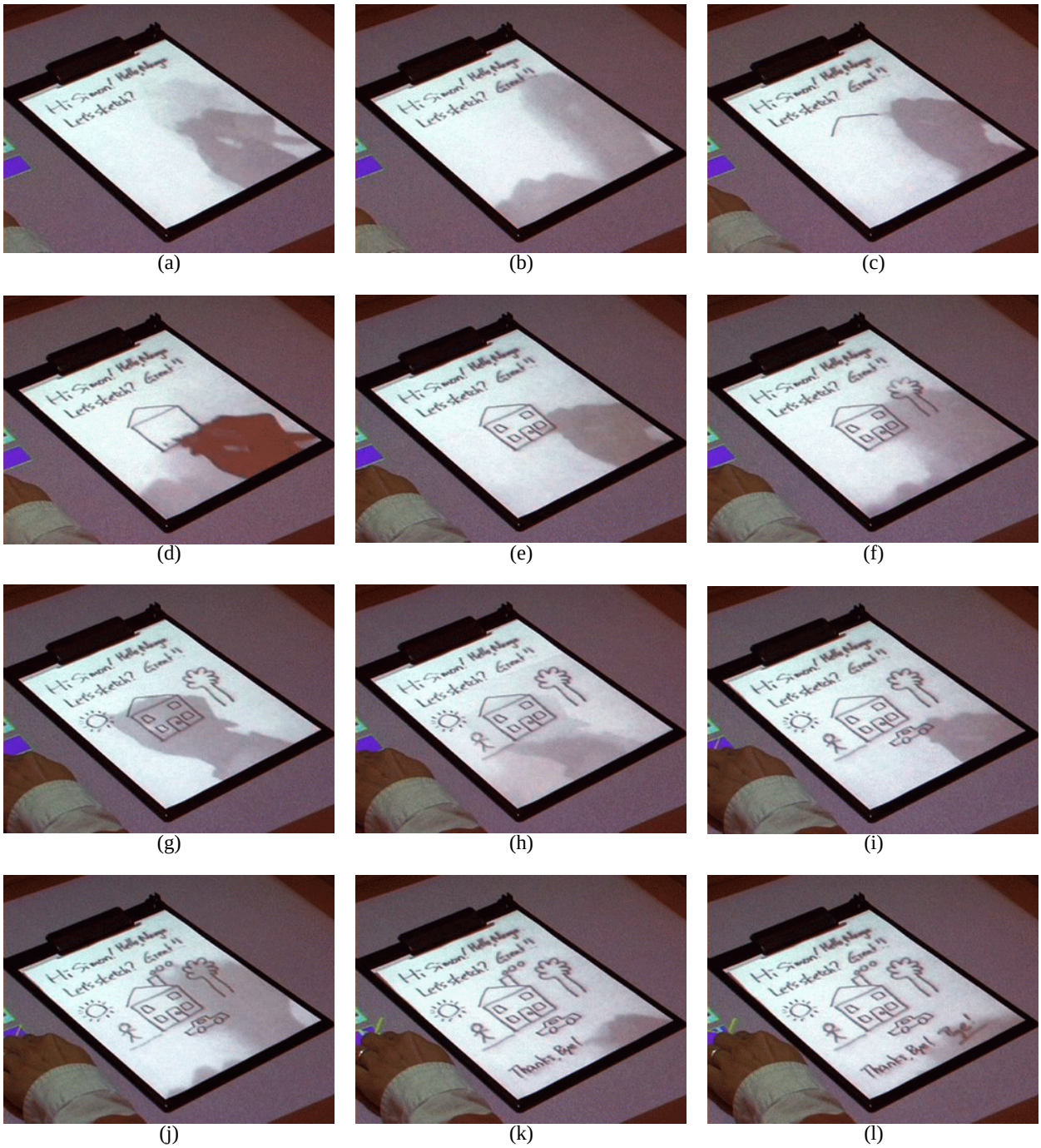


Figure 33: Session summarization results. The session lasting 90 seconds (2700 frames) was summarized in 12 frames, compression ratio of $\frac{1}{225}$. From (a) to (l), the drawing of each site is added step by step alternately. Note that although the feedback ("visual echoing") of the hand (see Section 5) can be observed in the summarized frames, no frames contain the real user's hand except for frame (d) which captured a real hand in error.

Table 7: Timing results for the Tele-Graffiti hand based user interface on a dual 450MHz Pentium-II machine. As the total calculation time of 9.2msec is within the idle time waiting for the next frame in the Paper Detection thread (see Table 4), adding user interface processing doesn't affect the performance of the paper tracking and the video transmission. Paper tracking and UI processing operate together at 30Hz.

Component	Measured Time	Operations per Pixel
Background Subtraction	1.6msec	9.4
Connected Component Search	4.2msec	24.6
Hand and Tip Detection	1.2msec	7.0
UI Objects Handling	2.2msec	12.9
Overall	9.2msec	53.9

applicable to similar projector-camera communication systems such as the Xerox "Double DigitalDesk" and INRIA's "MagicBoard." Starting with photometric models of the projector, the camera, and the paper used to sketch on, we derive the overall gain of the system, the final state that the system converges to, and the final image that will be viewed by the users of the system. We validate our image formation model and its predictions empirically.

Based on our analysis we derive the "optimal gain" as the gain that results in the final viewed image being as close as possible to the sum of the two original, non-feedback, images. Setting the gain to be the optimal gain eliminates visual echoing and results in the best overall image quality. Because the ambient light is usually very weak compared to the projector, however, setting up the system with the optimal gain in normal office conditions results in images that are often very dark. To alleviate this difficulty we propose a way of using the projector to augment the ambient light in the scene and obtain bright images that don't suffer from visual echoing. We also derive appropriate settings for this scheme.

The final "steady state" image viewed by the users of Tele-Graffiti is a weighted combination of the two images that would have been imaged without the Tele-Graffiti feedback loop. In many scenarios it is desirable to estimate what would have been imaged without

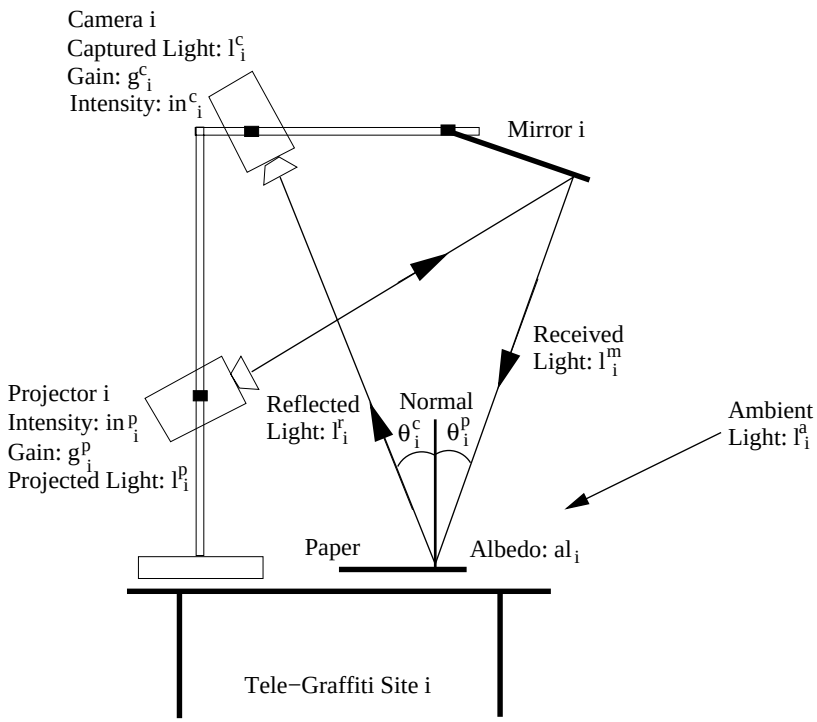


Figure 34: The image formation model. The projector at site i projects a pixel of intensity in_i^p with gain g_i^p resulting in projected light radiance l_i^p. The irradiance of the projected light reaching the paper after being reflected by the mirror is l_i^m and the angle that the projected light makes with the paper normal is θ_i^p. The total irradiance of the ambient light in the scene that reaches the paper is l_i^a and the albedo of the paper is al_i. The radiance of the reflected light is l_i^r, the angle the reflected light makes with the normal is θ_i^c, the irradiance of the light reaching the camera sensor plane is l_i^c, the gain of the camera is g_i^c, and the intensity of the pixel in the camera is in_i^c.

the feedback. For example it would be nice to determine what was drawn on each of the two pieces of paper from the weighted combination of them that is actually imaged. We propose an algorithm to estimate these "ambient light images" from the final viewed images, under the simplifying assumption that the ambient illumination is constant across the paper.

We complete this section by discussing how our analysis is affected: (1) when automatic gain control (AGC) is used in the video cameras and (2) by color imagery.

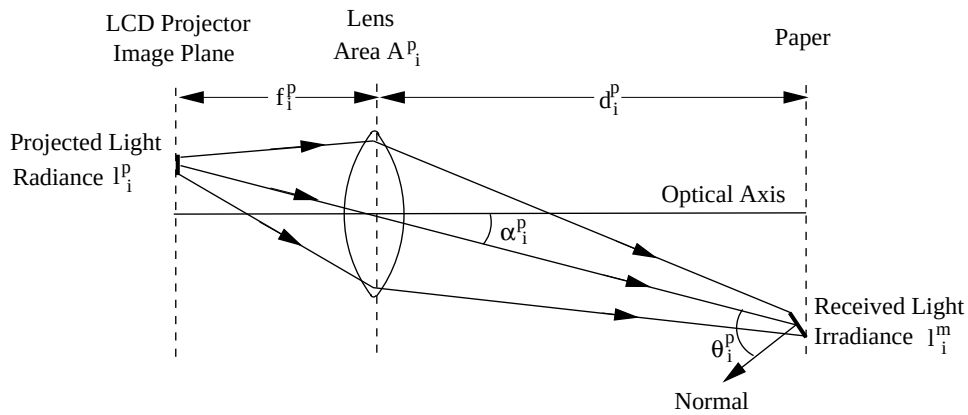


Figure 35: The projector model. Light radiance l_i^p is transmitted by the LCD, focused by the lens onto the paper where the irradiance is l_i^m. See the text for the relationship between l_i^p and l_i^m.

5.1 Image Formation Model

We present our image formation model in Figures 34, 35, and 36. Light is projected by the projector, reflected by the mirror onto the paper where it is combined with the ambient light in the scene, and reflected again to be captured by the camera. We now describe each component of the model in turn; the projector, the paper, and the camera.

5.1.1 Projector Model

We assume that if a pixel in the LCD projector is set to project intensity $\mathrm{in}_i^p \in [0, 255]$, the radiance of the light transmitted by the LCD image plane is:

$$l_i^p \;=\; g_i^p \cdot \mathrm{in}_i^p \tag{5}$$

where g_i^p is the gain of the projector; i.e. we assume that the response of the projector is linear. See Section 5.1.5 for an empirical validation. The relationship between the radiance of the light transmitted by the LCD and the irradiance of the light that is finally received by the paper is illustrated in Figure 35. The derivation is similar to the derivation of the relationship between scene radiance and image irradiance for a camera (see [Horn, 1996] Section 10.3) except that the plane of the paper that the light is projected onto is not frontal, unlike the

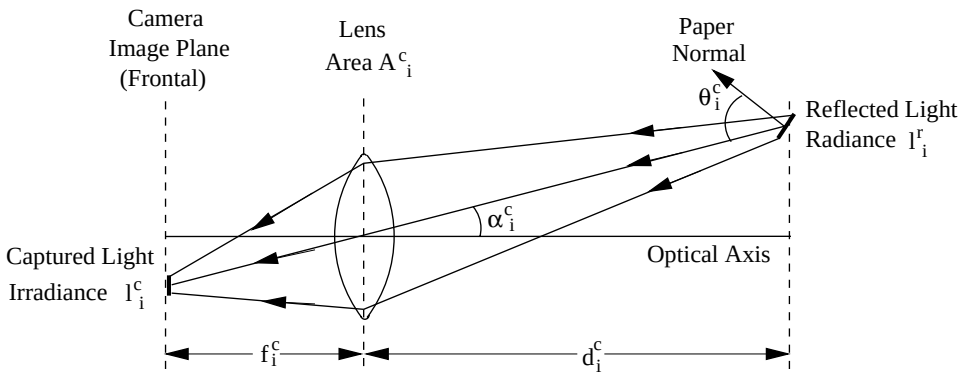


Figure 36: The camera model. Light with radiance l_i^r is reflected by the paper, focused by the lens onto the image plane of the camera, where the irradiance is l_i^c. See body of text for more details.

camera image plane. Taking into account this foreshortening, the relationship becomes:

$$l_i^m \;=\; l_i^p \cdot \frac{A_i^p}{(d_i^p)^2} \cdot \cos^3 \alpha_i^p \cdot \cos \theta_i^p \tag{6}$$

where l_i^m is the irradiance of the light received at the paper, A_i^p is the area of the projector lens, d_i^p is the distance from the lens to the paper (via the planar mirror, which can be ignored otherwise), α_i^p is the angle that the principal ray makes with the optical axis, and θ_i^p is the angle that the principal ray makes with the normal of the paper.

5.1.2 Paper Model

We assume that the paper can be modeled as Lambertian. If the total ambient incoming irradiance is l_i^a and the albedo of the paper is al_i, the radiance of the reflected light is:

$$l_i^r \;=\; \mathrm{al}_i \cdot [l_i^m + l_i^a]. \tag{7}$$

5.1.3 Camera Model

The model of the camera [Horn, 1996] is very similar to the model of the projector, and is shown in Figure 36. If the radiance of the paper is l_i^r, the irradiance of the light captured

on the image plane of the camera is:

$$l_i^c \;=\; l_i^r \cdot \frac{A_i^c}{(f_i^c)^2} \cdot \cos^4 \alpha_i^c \tag{8}$$

where A_i^c is the area of the lens, f_i^c is the distance between the lens and the image plane, and α_i^c is the angle between the principal ray and the optical axis. We also assume that the response of the camera is linear; i.e. the intensity of the corresponding pixel is:

$$\mathrm{in}_i^c \;=\; g_i^c \cdot l_i^c \tag{9}$$

where g_i^c is the gain of the camera and $\mathrm{in}_i^c \in [0, 255]$. Again, see Section 5.1.5 for an empirical validation of the linearity of the Sony DFW-VL500 cameras that we use.

5.1.4 Complete Imaging Model

Putting together Equations (5)–(9), we obtain a relationship between the intensity of the pixel in the projector in_i^p and the intensity of the pixel in the camera in_i^c:

$$\mathrm{in}_i^c \;=\; g_i^c \cdot \frac{A_i^c}{(f_i^c)^2} \cdot \cos^4 \alpha_i^c \cdot \mathrm{al}_i \cdot \left[\mathrm{in}_i^p \cdot g_i^p \cdot \frac{A_i^p}{(d_i^p)^2} \cdot \cos^3 \alpha_i^p \cdot \cos \theta_i^p + l_i^a \right]. \tag{10}$$

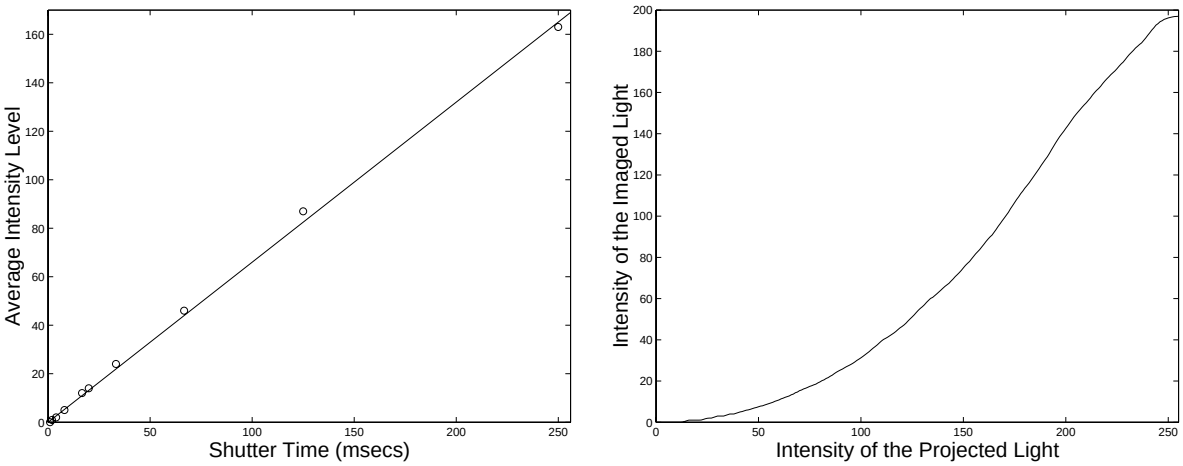
Denoting $K_i^c = g_i^c \cdot \frac{A_i^c}{(f_i^c)^2} \cdot \cos^4 \alpha_i^c$ and $K_i^p = g_i^p \cdot \frac{A_i^p}{(d_i^p)^2} \cdot \cos^3 \alpha_i^p \cdot \cos \theta_i^p$ we obtain:

$$\mathrm{in}_i^c \;=\; K_i^c \cdot \mathrm{al}_i \cdot \left[l_i^a + K_i^p \cdot \mathrm{in}_i^p \right]. \tag{11}$$

Equation (11) says that the image captured by the ith camera in_i^c is the sum of two terms. The first term $K_i^c \cdot \mathrm{al}_i \cdot l_i^a$ is the image of the paper in the ambient light only; it is the image that would have been captured if the projector was not there. The second term $K_i^c \cdot \mathrm{al}_i \cdot K_i^p \cdot \mathrm{in}_i^p$ is the image created by the light originating from the projector.

5.1.5 Photometrically Calibrating the Cameras and Projectors

The response functions of cameras and projectors are frequently not linear, as assumed in Equations (5) and (9). We first estimated the response function of the Sony DFW-VL500



(a) Sony DFW-VL500 Response Function (b) Panasonic PT-L701U Response Function

Figure 37: Empirically measured response functions of the Sony DFW-VL500 cameras and the Panasonic PT-L701U projectors. (a) We measured the response function of the cameras by varying the shutter time. We plot the shutter time (which is proportional to the amount of light entering the camera) against the average intensity level over a white piece of paper. The results show that the response function of the Sony DFW-VL500's is linear, at least when the gamma setting is set to OFF2. (b) We measured the response function of the projectors by projecting different intensities and measuring the amount of light recorded by the (linear) cameras. We plot the intensity of the projected light against the intensity of the imaged light. We find the relationship to be non-linear.

cameras. We set the gamma correction to "OFF2" (value 130) which the manufacture claims sets up the camera with a linear response function. See Section 2.4.2 for more details of the intensity corrections that can be applied to the Sony DFW-VL500's. We then measured the response function of the camera by varying the shutter timing, similarly to how it is varied in the photometric calibration algorithm proposed in [Mitsunaga and Nayar, 1999].

In Figure 37(a) we plot the measured response function of the Sony DFW-VL500's (with gamma setting OFF2). We plot the shutter time against the intensity recorded (averaged over a white piece of paper.) Since the shutter time is proportional to the amount of light entering the camera and the graph in Figure 37(a) is linear, we conclude that the Sony DFW-VL500's are indeed linear. Once we know that the cameras are linear, we then use them to measure the response function of the projectors. We project a variety of different intensities from the projector and measure the intensity of light imaged by the camera (after

it has been reflected by the mirror and paper.) Again, we compute the average value across a sub-rectangle of the paper. In Figure 37(b) we plot the intensity of the projected light against the intensity of the captured light. The results show the projector to be non-linear.

It is easy to correct the non-linear response of the projectors in software. Essentially we just need to apply the inverse of the function in Figure 37(b) to the signal sent to the projector. Suppose that the function in Figure 37(b) is denoted by $\mathrm{in}^c = \mathrm{R}(\mathrm{in}^p)$. We simply add a correction:

$$\mathrm{in}^p \longleftarrow \mathrm{R}^{-1}\left(\frac{\mathrm{R}(255) \times \mathrm{in}^p}{255}\right) \tag{12}$$

to the signal in^p sent to the projector. The resulting response of the projector is then:

$$\mathrm{in}^c = \mathrm{R}\left(\mathrm{R}^{-1}\left(\frac{\mathrm{R}(255) \times \mathrm{in}^p}{255}\right)\right) = \frac{\mathrm{R}(255) \times \mathrm{in}^p}{255} \tag{13}$$

which is linear in in^p. Note that we have to apply the inverse of R to $\frac{\mathrm{R}(255)\times \mathrm{in}^p}{255}$ rather that just in^p to ensure that the argument to R^{-1} is in the domain of the function.

The correction in Equation (12) can be implemented efficiently using a 1D table lookup, similarly to the fast YUV to RGB conversion described in Appendix A.1. It can also be combined with the offset table lookup used to provide extra ambient light from the projector. See Section 5.4 and Appendix A.3 for more details. The two table lookup operations can be composed into a single table lookup improving the efficiency even further. In general, it is desirable not to combine the YUV to RGB lookup with the projector correction lookup. The first of these simply corrects the format. We naturally want to keep the captured RGB images otherwise uncorrected. The uncorrected RGB image should then be corrected just before it is sent to the projector to compensate for the projector's non-linearity.

5.2 The Final Viewed Image

If we set up a 2 site Tele-Graffiti system as in Figure 1, the projected image in_i^p is set to equal the image captured at the other site in_{3-i}^c. If either $i = 1$ or $i = 2$, the other site is

$3-i$. If we now suppose time proceeds in a sequence of steps and denote the image captured by the camera in the ith Tele-Graffiti site at time t by $\mathrm{in}_i^c(t)$, Equation (11) becomes:

$$\mathrm{in}_i^c(t) \;=\; \mathrm{in}_i^c(\mathrm{A}) + K_i^c \cdot \mathrm{al}_i \cdot K_i^p \cdot \mathrm{in}_{3-i}^c(t-1) \tag{14}$$

where $\mathrm{in}_i^c(\mathrm{A}) = K_i^c \cdot \mathrm{al}_i \cdot l_i^a$ is the image of the paper in the ambient light only. Expanding the recursive term $\mathrm{in}_{3-i}^c(t-1)$ yields:

$$\mathrm{in}_i^c(t) \;=\; \mathrm{in}_i^c(\mathrm{A}) + K_i^c \cdot \mathrm{al}_i \cdot K_i^p \cdot \mathrm{in}_{3-i}^c(\mathrm{A}) + K_i^c \cdot \mathrm{al}_i \cdot K_i^p \cdot K_{3-i}^c \cdot \mathrm{al}_{3-i} \cdot K_{3-i}^p \cdot \mathrm{in}_i^c(t-2). \tag{15}$$

Denote $G_i = K_i^c \cdot \mathrm{al}_i \cdot K_i^p$, the total gain through the ith Tele-Graffiti site; i.e. through the three steps of projector, paper, and camera. Equation (15) becomes:

$$\mathrm{in}_i^c(t) \;=\; \mathrm{in}_i^c(\mathrm{A}) + G_i \cdot \mathrm{in}_{3-i}^c(\mathrm{A}) + G_i \cdot G_{3-i} \cdot \mathrm{in}_i^c(t-2). \tag{16}$$

From Equation (11) half of the initial conditions of this recurrence relation are:

$$\mathrm{in}_i^c(0) \;=\; \mathrm{in}_i^c(\mathrm{A}) + G_i \cdot \mathrm{in}_i^p(0) \tag{17}$$

where $\mathrm{in}_i^p(0)$ is the image projected by the projector in the ith site at system startup. The other half of the initial conditions are:

$$\mathrm{in}_i^c(1) \;=\; \mathrm{in}_i^c(\mathrm{A}) + G_i \cdot \mathrm{in}_{3-i}^c(0) \;=\; \mathrm{in}_i^c(\mathrm{A}) + G_i \cdot \mathrm{in}_{3-i}^c(\mathrm{A}) + G_i \cdot G_{3-i} \mathrm{in}_{3-i}^p(0). \tag{18}$$

The solution of the recurrence relation in Equation (16) is therefore:

$$\mathrm{in}_i^c(t) \;=\; \left[\mathrm{in}_i^c(\mathrm{A}) + G_i \cdot \mathrm{in}_{3-i}^c(\mathrm{A})\right] \frac{1-(G_i \cdot G_{3-i})^{t/2}}{1-G_i \cdot G_{3-i}} + \mathrm{in}_i^c(0) \cdot [G_i \cdot G_{3-i}]^{t/2} \tag{19}$$

for $t \geq 2$ and even, and:

$$\mathrm{in}_i^c(t) \;=\; \left[\mathrm{in}_i^c(\mathrm{A}) + G_i \cdot \mathrm{in}_{3-i}^c(\mathrm{A})\right] \frac{1-(G_i \cdot G_{3-i})^{(t-1)/2}}{1-G_i \cdot G_{3-i}} + \mathrm{in}_i^c(1) \cdot [G_i \cdot G_{3-i}]^{(t-1)/2} \tag{20}$$

for $t \geq 3$ and odd. In either case, in the limit $t \to \infty$ we have:

$$\mathrm{in}_i^c(\infty) \;=\; \frac{\mathrm{in}_i^c(\mathrm{A}) + G_i \cdot \mathrm{in}_{3-i}^c(\mathrm{A})}{1-G_i \cdot G_{3-i}} \tag{21}$$

assuming that $G_i \cdot G_{3-1} < 1$. If $G_i \cdot G_{3-1} \geq 1$ the value of $\text{in}_i^c(t)$ keeps increasing until it saturates the camera and the projector and $\text{in}_i^p(t) = \text{in}_i^c(t) = 255$.

Note that the gain $G_i = K_i^c \cdot \text{al}_i \cdot K_i^p$ is not actually a constant, but depends on the paper albedo. The gain will therefore be different for points on the paper that have been written on compared to points that have not been written on. The terms K_i^c and K_i^p also vary in general across the surface of the paper because the angles α_i^c, α_i^p, and θ_i^p vary, albeit very slowly, across the paper. Because the variation is so slow, for simplicity we assume that K_i^c and K_i^p are constant. We do model the variation in the paper albedo, however.

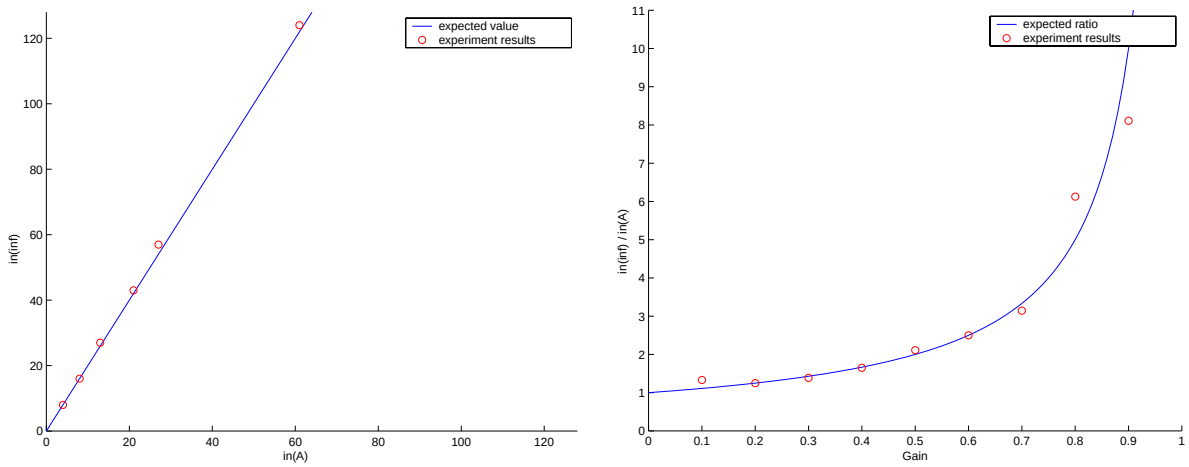
5.2.1 Empirical Validation of the Final Viewed Image

In our empirical evaluation we work with a blank piece of paper. This is sufficient to validate Equation (21). The albedos al_i and gains G_i are therefore treated as constant in this section.

The overall gain G_i depends on the camera and projector parameters through K_i^c and K_i^p. It is generally easier to change the camera parameters, especially with the Sony DFW-VL500's since their parameters can be changed over the Firewire interface. Note that until Section 5.6 we assume that the camera Automatic Gain Control is switched off. See Section 2.4.5 for more discussion of how the cameras are set up.

The gain G_i is set in the following way. We place a blank piece of paper under the camera and grab an image with the projector switched off. We then set the projector to project intensity $\text{in}_i^p = 128$ and grab a second image. We subtract the first image from the second and estimate the average intensity in a small region in the center of the paper. We then perform a binary search over the camera parameters until this average value equals $128 \times G_i$. (Either the shutter or aperture can be varied. We use the aperture.) Naturally, for each trial with new camera parameters we grab a new image with the projector switched off.

We present the results of our experiments in Figure 38. We set the gains $G_1 = G_2$



(a) Varying the Intensity $\text{in}_1^c(\text{A}) = \text{in}_2^c(\text{A})$

(b) Varying the Gain $G_1 = G_2$

Figure 38: Empirical validation of Equation (21). (a) We plot $\text{in}_1^c(\infty)$ for various different values of $\text{in}_1^c(\text{A}) = \text{in}_2^c(\text{A})$ for fixed $G_1 = G_2 = 0.5$. (b) We plot the value of $\text{in}_1^c(\infty)/\text{in}_1^c(\text{A})$ for various settings of $G_1 = G_2$ and $\text{in}_1^c(\text{A}) = \text{in}_2^c(\text{A})$. In both graphs we compare the empirically measured results with the values predicted by Equation (21). We find close agreement in both cases.

and intensities $\text{in}_1^c(\text{A}) = \text{in}_2^c(\text{A})$ to be various values and compare the final measured image intensity $\text{in}_i^c(\infty)$ with that predicted by Equation (21). The ambient light image $\text{in}_i^c(\text{A})$ is varied independently of the gain in our experiments by varying the amount of ambient light in the room using an array of spot-lights. In Figure 38(a) we present results varying the ambient light images, keeping the gains fixed. In Figure 38(b) we present results varying the gains, and implicitly the ambient light images also. In Figure 38(b) we therefore plot the ratio of the final viewed image to the ambient light image, a quantity that should be constant if Equation (21) is correct. As can be seen from the graphs in Figure 38, the predicted values closely match those measured empirically, thereby validating Equation (21) and our derivation of the final viewed images $\text{in}_i^c(\infty)$ presented in the previous section.

5.3 Choosing the Optimal Gain

Equation (21) describes the final state that a 2 site Tele-Graffiti system will settle into. It also describes the final image captured by the cameras and viewed by the users of the system.

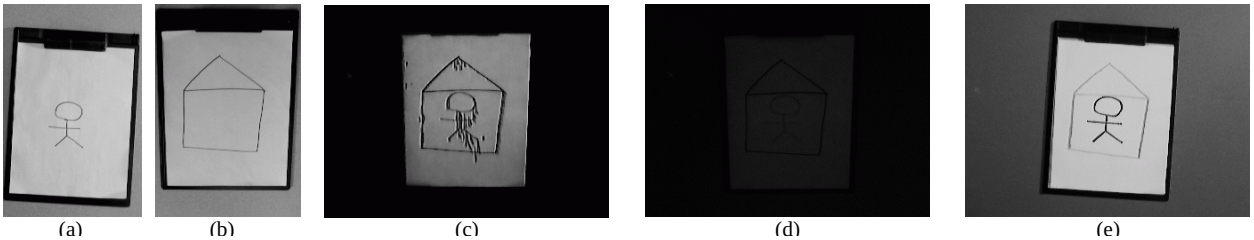


Figure 39: (a-c) Visual echoing caused by using too large a gain $G_i = 0.95$, combined with the inevitable errors in tracking the paper and calibrating H_{pc}: (a) $\mathrm{in}_1^c(\mathrm{A})$, (b) $\mathrm{in}_2^c(\mathrm{A})$, (c) $\mathrm{in}_1^c(\infty)$. (d) The final viewed image $\mathrm{in}_1^c(\infty)$ with optimal gain $G_i^{\mathrm{pap}} = 0.5$ and under normal office illumination is very dark. (e) The final viewed image $\mathrm{in}_1^c(\infty)$ with optimal gain $G_i^{\mathrm{pap}} = 0.5$ and under strong ambient illumination provided by spot lights has excellent contrast and minimal echoing.

How can we set the gains G_i and G_{3-i} to maximize the image quality of $\mathrm{in}_i^c(\infty)$?

Equation (21) implies that if we want $\mathrm{in}_i^c(\mathrm{A})$ and $\mathrm{in}_{3-i}^c(\mathrm{A})$ to contribute equal weight to $\mathrm{in}_i^c(\infty)$ we should use a gain $G_i \approx 1$; if $G_i \ll 1$ the user will not see the image $\mathrm{in}_{3-i}^c(\mathrm{A})$ from the other site. On the other hand, in the limit $G_i, G_{3-1} \to 1$ the final viewed image $\mathrm{in}_i^c(\infty) \to \infty$ and the cameras saturate. Too large a gain also leads to "visual echoing." When one of the users' waves their hand across the paper they see multiple "echoes" of it, transmitted back from the image projected at the other site. Another side effect of visual echoing is that, because of inevitable errors in geometric calibration and paper tracking, sketches "run." See Figure 39(a-c) for an example of such visual echoing.

In practice we need to find a compromise between these two extremes. One way to choose the gains is to choose them to make $\mathrm{in}_i^c(\infty)$ as close as possible to $\mathrm{in}_i^c(\mathrm{A}) + \mathrm{in}_{3-i}^c(\mathrm{A})$. It turns out that is best to use the relative error between these two quantities. If we add in a similar expression for $\mathrm{in}_{3-i}^c(\infty)$, the other site, we aim to minimize:

$$\frac{1}{\left[\mathrm{in}_i^c(\mathrm{A}) + \mathrm{in}_{3-i}^c(\mathrm{A})\right]^2} \left(\left[\frac{\mathrm{in}_i^c(\mathrm{A}) + G_i \cdot \mathrm{in}_{3-i}^c(\mathrm{A})}{1 - G_i \cdot G_{3-i}} - \mathrm{in}_i^c(\mathrm{A}) - \mathrm{in}_{3-i}^c(\mathrm{A}) \right]^2 + \right.$$
$$\left. \left[\frac{\mathrm{in}_{3-i}^c(\mathrm{A}) + G_{3-i} \cdot \mathrm{in}_i^c(\mathrm{A})}{1 - G_i \cdot G_{3-i}} - \mathrm{in}_i^c(\mathrm{A}) - \mathrm{in}_{3-i}^c(\mathrm{A}) \right]^2 \right). \tag{22}$$

In this expression we have ignored the fact that the gains G_i and G_{3-i} are variables and depend upon the albedo of the paper. Assume that $w(\mathrm{al}_i)$ denotes the probability that the

albedo of the paper is al_i. We can then generalize the expression in Equation (22) to:

$$\int_0^{1/\pi}\int_0^{1/\pi}\frac{w(\mathrm{al}_i)w(\mathrm{al}_{3-i})}{\left[\mathrm{in}_i^c(\mathrm{A})+\mathrm{in}_{3-i}^c(\mathrm{A})\right]^2}\left(\left[\frac{\mathrm{in}_i^c(\mathrm{A})+G_i\cdot\mathrm{in}_{3-i}^c(\mathrm{A})}{1-G_i\cdot G_{3-i}}-\mathrm{in}_i^c(\mathrm{A})-\mathrm{in}_{3-i}^c(\mathrm{A})\right]^2\right)$$
$$+\left[\frac{\mathrm{in}_{3-i}^c(\mathrm{A})+G_{3-i}\cdot\mathrm{in}_i^c(\mathrm{A})}{1-G_i\cdot G_{3-i}}-\mathrm{in}_i^c(\mathrm{A})-\mathrm{in}_{3-i}^c(\mathrm{A})\right]^2\Bigg)\ \mathrm{d}\,\mathrm{al}_i\ \mathrm{d}\,\mathrm{al}_{3-i}. \qquad (23)$$

The ranges of the integrals are $[0, 1/\pi]$ because the maximum value that albedos can take is $1/\pi$ [Horn, 1996]. Equation (23) then simplifies to:

$$\int_0^{1/\pi}\int_0^{1/\pi}\frac{w(\mathrm{al}_i)w(\mathrm{al}_{3-i})}{\left[\mathrm{in}_i^c(\mathrm{A})+\mathrm{in}_{3-i}^c(\mathrm{A})\right]^2}\left(\left[\frac{G_i\cdot G_{3-i}\cdot\mathrm{in}_i^c(\mathrm{A})+(G_i\cdot G_{3-i}+G_i-1)\cdot\mathrm{in}_{3-i}^c(\mathrm{A})}{1-G_i\cdot G_{3-i}}\right]^2\right)$$
$$+\left[\frac{G_i\cdot G_{3-i}\cdot\mathrm{in}_{3-i}^c(\mathrm{A})+(G_i\cdot G_{3-i}+G_{3-i}-1)\cdot\mathrm{in}_i^c(\mathrm{A})}{1-G_i\cdot G_{3-i}}\right]^2\Bigg)\ \mathrm{d}\,\mathrm{al}_i\ \mathrm{d}\,\mathrm{al}_{3-i}. \qquad (24)$$

Equation (24) contains two types of terms, gains G_i and ambient light images $\mathrm{in}_i^c(\mathrm{A})$. Because $G_i = K_i^c\cdot\mathrm{al}_i\cdot K_i^p$ and $\mathrm{in}_i^c(\mathrm{A}) = K_i^c\cdot\mathrm{al}_i\cdot l_i^a$ these two quantities are related by:

$$\mathrm{in}_i^c(\mathrm{A}) \;=\; G_i\cdot\frac{l_i^a}{K_i^p} \qquad (25)$$

In order to proceed we now make the following three simplifying assumptions: (1) we assume that l_i^a is constant across the paper, (2) we assume that K_i^p is constant across the paper, and (3) we assume that $\frac{l_i^a}{K_i^p}$ is the same at the two sites. The first of these three assumptions just means that the ambient light is constant across the paper which is very reasonable. The second assumption just means that the various angles between the projected and reflected light, the paper, the camera the projector are all constant across the paper. This is also a reasonable approximation. The third assumption can easily be generalized, however. The details are omitted but it is straightforward to estimate $\frac{l_i^a}{K_i^p}$ for each of the two sites separately from Equation (25). The constant relative values of these expressions can then included in the following analysis and different optimal gains derived for the two sites.

After making these assumptions, Equation (24) simplifies to:

$$\int_0^{1/\pi}\int_0^{1/\pi}\frac{w(\mathrm{al}_i)w(\mathrm{al}_{3-i})}{\left[G_i+G_{3-1}\right]^2}\left(\left[\frac{G_i^2\cdot G_{3-i}+(G_i\cdot G_{3-i}+G_i-1)\cdot G_{3-i}}{1-G_i\cdot G_{3-i}}\right]^2\right)$$

$$+\left[\frac{G_i \cdot G_{3-i}^2 + (G_i \cdot G_{3-i} + G_{3-i} - 1) \cdot G_i}{1 - G_i \cdot G_{3-i}}\right]^2\Bigg) \; \mathrm{d}\,\mathrm{al}_i \; \mathrm{d}\,\mathrm{al}_{3-i}. \tag{26}$$

The optimal value of Equation (26) clearly depends on the weighting function $w(\mathrm{al}_i)$. The sketches we work with are mostly simple "black on white" line drawings. The albedo is therefore nearly always either $\mathrm{al}_i^{\mathrm{pap}}$, the paper albedo, or 0. Let us therefore assume that:

$$w(\mathrm{al}_i) \;=\; \frac{1}{2}\delta(\mathrm{al}_i) + \frac{1}{2}\delta(\mathrm{al}_i - \mathrm{al}_i^{\mathrm{pap}}) \tag{27}$$

where $\delta(\cdot)$ is the Dirac delta function; i.e. a unit impulse. It is possible to continue the analysis with other choices of the weighting function $w(\mathrm{al}_i)$ and derive the optimal gain in other scenarios. We just need to make a concrete choice to proceed. Let us denote $G_i^{\mathrm{pap}} = K_i^c \cdot \mathrm{al}_i^{\mathrm{pap}} \cdot K_i^p$, the gain of the paper. Equation (26) then simplifies to:

$$\frac{1}{[G_i^{\mathrm{pap}} + G_{3-1}^{\mathrm{pap}}]^2}\Bigg(\left[\frac{(G_i^{\mathrm{pap}})^2 \cdot G_{3-i}^{\mathrm{pap}} + (G_i^{\mathrm{pap}} \cdot G_{3-i}^{\mathrm{pap}} + G_i^{\mathrm{pap}} - 1) \cdot G_{3-i}^{\mathrm{pap}}}{1 - G_i^{\mathrm{pap}} \cdot G_{3-i}^{\mathrm{pap}}}\right]^2 +$$
$$\left[\frac{G_i^{\mathrm{pap}} \cdot (G_{3-i}^{\mathrm{pap}})^2 + (G_i^{\mathrm{pap}} \cdot G_{3-i}^{\mathrm{pap}} + G_{3-i}^{\mathrm{pap}} - 1) \cdot G_i^{\mathrm{pap}}}{1 - G_i^{\mathrm{pap}} \cdot G_{3-i}^{\mathrm{pap}}}\right]^2 + (G_i^{\mathrm{pap}})^2 + (G_{3-i}^{\mathrm{pap}})^2\Bigg). \tag{28}$$

The minimum value of the expression in Equation (28) is easily found numerically to be $G_i^{\mathrm{pap}} = G_{3-i}^{\mathrm{pap}} = 0.5$. Equation (28) is plotted in Figure 40(a) for $G_i^{\mathrm{pap}} = G_{3-i}^{\mathrm{pap}}$. The optimal gain of the paper with which to set up Tele-Graffiti is therefore 0.5. The gain of the paper G_i^{pap} can be set using the method described in Section 5.2.1.

Note that in our derivation of the optimal gain we made two assumptions that can be generalized: (1) that $\frac{l_i^a}{K_i^p}$ is the same for the two Tele-Graffiti sites and (2) that the weighting function $w(\mathrm{al}_i)$ is given by Equation (27). It is possible to measure the value of $\frac{l_i^a}{K_i^p}$ for the two Tele-Graffiti sites separately and then derive different optimal gains where in general $G_i^{\mathrm{pap}} \neq G_{3-i}^{\mathrm{pap}}$. It is also possible to use other albedo weighting functions and derive other optimal gains for different proportions of black and white in Equation (27).

5.4 Compensating for Insufficient Ambient Illumination

Most LCD projectors are designed to be very bright because of the quadratic falloff in brightness with the distance d_i^p between the projector and the projection surface. See Equation (6). In Tele-Graffiti the value of d_i^p is smaller than the value that most projectors are designed for. As a result K_i^p is often relatively large. To obtain $G_i^{\text{pap}} = 0.5$, we have to use a small aperture (or a fast shutter) to make K_i^c small enough. The result is that $\text{in}_i^c(\text{A}) = K_i^c \cdot \text{al}_i \cdot l_i^a$, the image of the paper in ambient light, is very dark. As a result, the final image viewed $\text{in}_i^c(\infty)$ is also very dark. See Figure 39(d) for an example. Ideally we would like $\text{in}_i^c(\infty)$ to have its brightest pixels to have intensity close to $\text{in}_i^{\max} \approx 255$ to maximize the dynamic range of the camera. Assume that the paper is the brightest object. Then, to obtain:

$$\text{in}_i^c(\infty) \;=\; \frac{\text{in}_i^c(\text{A}) + G_i^{\text{pap}} \cdot \text{in}_{3-i}^c(\text{A})}{1 - G_i^{\text{pap}} \cdot G_{3-i}^{\text{pap}}} \;=\; \text{in}_i^{\max} \;\approx\; 255 \tag{29}$$

when $\text{in}_i^c(\text{A}) = \text{in}_{3-i}^c(\text{A})$ and $G_i^{\text{pap}} = G_{3-i}^{\text{pap}} = 0.5$, we need:

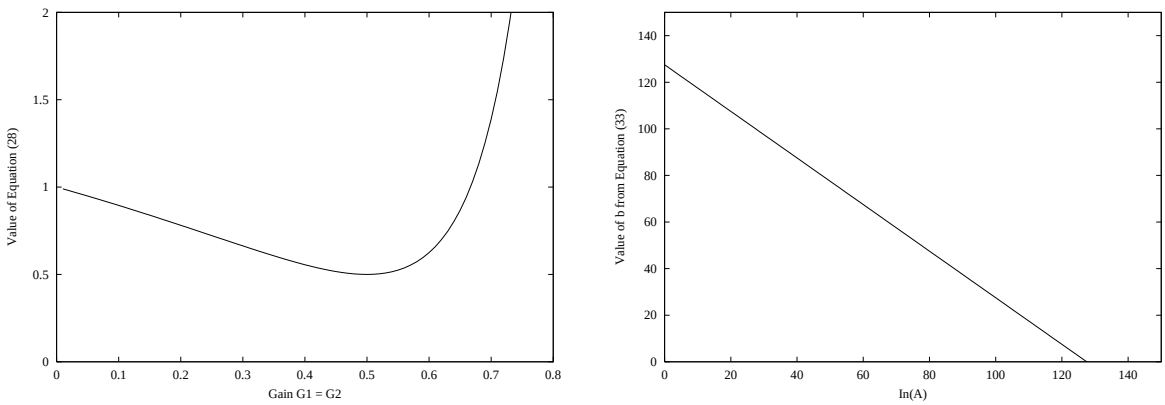
$$\text{in}_i^c(\text{A}) \;=\; \frac{1 - (G_i^{\text{pap}})^2}{1 + G_i^{\text{pap}}} \times \text{in}_i^{\max} \;=\; (1 - G_i^{\text{pap}}) \times \text{in}_i^{\max} \;\approx\; 127.5. \tag{30}$$

If the brightest pixel in $\text{in}_i^c(\text{A}) \ll 127$, the brightest pixel in $\text{in}_i^c(\infty)$ will be $\ll 255$ and a large fraction of the dynamic range of the camera (and projector) will be wasted.

The only way to increase $\text{in}_i^c(\text{A}) = K_i^c \cdot \text{al}_i \cdot l_i^a$ is to increase the ambient light l_i^a because the albedo al_i is fixed and K_i^c must to be set to ensure that the gain $G_i^{\text{pap}} = 0.5$. It is often impossible or inconvenient to increase the ambient illumination in the room. Fortunately, as we now show how, it is possible to use the projector to provide the additional illumination. Suppose that instead of setting the projected image in_i^p to be the captured image in_{3-i}^c we set it to be the image:

$$\text{in}_i^p \;=\; \text{b} + \frac{\text{in}_i^{\max} - \text{b}}{\text{in}_i^{\max}} \times \text{in}_{3-i}^c; \tag{31}$$

i.e. we effectively add constant additional illumination of intensity b. In making this change, we effectively change the gain of the projector to be smaller by the factor $(\text{in}_i^{\max} - \text{b})/\text{in}_i^{\max}$. We therefore have to increase the gain of the camera by the factor $\text{in}_i^{\max}/(\text{in}_i^{\max} - \text{b})$ to keep the



(a) Derivation of the Optimal Gain

(b) The Base Intensity b to Project

Figure 40: (a) Choosing the Gain. We choose the optimal gains G_i^{pap} and G_{3-i}^{pap} to make $\text{in}_i^c(\infty)$ as close as possible to $\text{in}_i^c(\text{A}) + \text{in}_{3-i}^c(\text{A})$; i.e. to minimize the expression in Equation (28). The minimum value is achieved when $G_i^{\text{pap}} = G_{3-i}^{\text{pap}} = 0.5$. Here we plot Equation (28) for $G_i^{\text{pap}} = G_{3-i}^{\text{pap}}$. (b) Choosing the additional ambient light b to be projected. Given the brightest pixel in $\text{in}_i^c(\text{A})$ and assuming $G_i^{\text{pap}} = 0.5$, the graph above (b) of Equation (33) gives the value of b to be projected using Equation (31) to obtain the brightest pixel in $\text{in}_i^c(\infty)$ to be $\text{in}_i^{\max} = 255$.

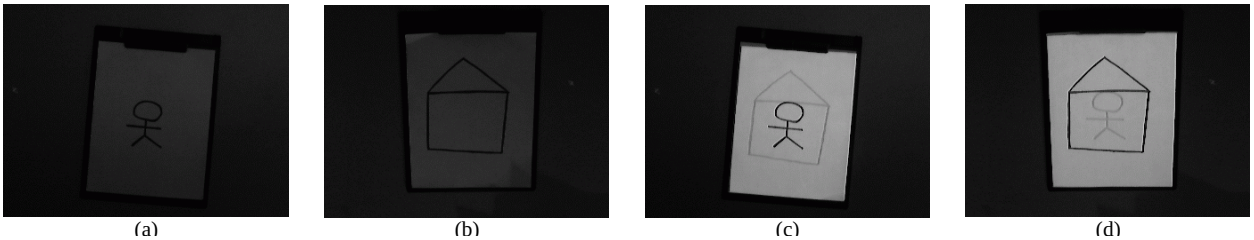
gain of the overall system constant. The value of $\text{in}_i^c(\text{A})$ will therefore increase by this factor because of the change in the gain of the camera and by the amount $\text{in}_i^{\max} \cdot G_i^{\text{pap}} \cdot \text{b}/(\text{in}_i^{\max} - \text{b})$ because of the additional ambient light. To choose b to optimize the dynamic range of the camera we need to solve:

$$\frac{\text{in}_i^{\max}}{\text{in}_i^{\max} - \text{b}} \times [\text{in}_i^c(\text{A}) + G_i^{\text{pap}} \times \text{b}] \;=\; (1 - G_i^{\text{pap}}) \times \text{in}_i^{\max} \tag{32}$$

The solution of this equation is:

$$\text{b} \;=\; (1 - G_i^{\text{pap}}) \times \text{in}_i^{\max} - \text{in}_i^c(\text{A}). \tag{33}$$

Equation (33), plotted in Figure 40(b) for $G_i^{\text{pap}} = 0.5$ and $\text{in}_i^{\max} = 255$, can be used to estimate a suitable base intensity b to project using Equation (31) to ensure that the overall system does not saturate, and yet the maximum possible dynamic range of the camera and projector are used. Example images of the system running using the value of b computed using Equation (33) are included in Figure 41. Note that the images have the image quality



(a) (b) (c) (d)

Figure 41: A 2 site Tele-Graffiti system operating with optimal gain $G_i^{\text{pap}} = 0.5$ and additional ambient light b provided by the projector: (a) $= \text{in}_1^c(\text{A})$, (b) $= \text{in}_2^c(\text{A})$, (c) $= \text{in}_1^c(\infty)$, (d) $= \text{in}_2^c(\infty)$. Note the superior image quality of the images compared to Figure 39(c-d); their brightness, contrast, minimal visual echoing, and visibility of the image from the other site. The final images are very close to Figure 39(e). The results in (c-d) however are captured under normal office illumination, whereas Figure 39(e) was captured under strong ambient illumination provided by spot-lights.

that we have been striving for; they are bright, have excellent contrast, minimal visual echoing, and the image from the other site is clearly visible.

The intensity correction in Equation (31) can easily be implemented as a table lookup, similarly to how the fast YUV to RGB conversion is implemented. See Appendix A.1 for the details of YUV to RGB conversion. It can also be combined with the photometric calibration of the projectors described in Section 5.1.5. See Appendix A.3 for more details.

5.5 Image Separation

When we switch on Tele-Graffiti the system quickly converges to the steady state:

$$\text{in}_i^c(\infty) \;=\; \frac{\text{in}_i^c(\text{A}) + G_i \cdot \text{in}_{3-i}^c(\text{A})}{1 - G_i \cdot G_{3-i}}. \tag{34}$$

For example the ambient images in Figures 41(a) and (b) are converted to the final viewed images in Figures 41(c) and (d). In the final viewed images the two separate sketches are combined into a single sketch. In Figure 41(c) and (d) the person and the house are combined.

Is it possible to reverse this process? Can we recover Figures 41(a) and (b) from Figures 41(c) and (d)? Doing this corresponds to inverting Equation (34). As in Section 5.3 the difficulty in doing this is the fact that G_i is not constant, but is related to $\text{in}_i^c(\text{A})$ through:

$$\text{in}_i^c(\text{A}) \;=\; G_i \cdot \frac{l_i^a}{K_i^p}. \tag{35}$$

If the ambient light l_i^a can vary completely independently across the paper, Equation (34) is not invertible. In Section 5.3 we assumed that l_i^a and K_i^p are constant across the paper. To invert Equation (34) we now make the same assumption. We also assume that the value of $\frac{l_i^a}{K_i^p}$ has been estimated at system startup. This can easily be performed. We set the gain of the paper G_i^{pap} to a fixed value. We can then measure the average ambient light image of the paper and use the results to compute $\frac{l_i^a}{K_i^p}$ from Equation (35).

Denote the ratio $R_i = \frac{K_i^p}{l_i^a}$. Eliminating G_i from Equation (34) yields:

$$\text{in}_i^c(\infty) \;=\; \frac{\text{in}_i^c(\text{A}) + R_i \cdot \text{in}_i^c(\text{A}) \cdot \text{in}_{3-i}^c(\text{A})}{1 - R_i \cdot R_{3-i} \cdot \text{in}_i^c(\text{A}) \cdot \text{in}_{3-i}^c(\text{A})}. \tag{36}$$

Rearranging this equation gives:

$$R_i \cdot [1 + R_{3-i} \cdot \text{in}_i^c(\infty)] \cdot \text{in}_i^c(\text{A}) \cdot \text{in}_{3-i}^c(\text{A}) + \text{in}_i^c(\text{A}) - \text{in}_i^c(\infty) \;=\; 0 \tag{37}$$

and similarly switching the roles of i and $3-i$:

$$R_{3-i} \cdot [1 + R_i \cdot \text{in}_{3-i}^c(\infty)] \cdot \text{in}_i^c(\text{A}) \cdot \text{in}_{3-i}^c(\text{A}) + \text{in}_{3-i}^c(\text{A}) - \text{in}_{3-i}^c(\infty) \;=\; 0 \tag{38}$$

Equations (37) and (38) are a pair of simultaneous quadratic equations for $\text{in}_i^c(\text{A})$ and $\text{in}_{3-i}^c(\text{A})$ in terms of $\text{in}_i^c(\infty)$ and $\text{in}_{3-i}^c(\infty)$. These equations take the form:

$$A \cdot x \cdot y + x - B \;=\; 0 \qquad \text{and} \qquad C \cdot x \cdot y + y - D \;=\; 0 \tag{39}$$

where $A = R_i \cdot [1 + R_{3-i} \cdot \text{in}_i^c(\infty)]$, $B = \text{in}_i^c(\infty)$, $C = R_{3-i} \cdot [1 + R_i \cdot \text{in}_{3-i}^c(\infty)]$, and $D = \text{in}_{3-i}^c(\infty)$ are all positive constants. The variables are $x = \text{in}_i^c(\text{A})$ and $y = \text{in}_{3-i}^c(\text{A})$. Combining these two equations gives:

$$C \cdot x - C \cdot B \;=\; A \cdot y - A \cdot D. \tag{40}$$

Substituting back into the second half of Equation (39) gives:

$$A \cdot y^2 + (C \cdot B - A \cdot D + 1) \cdot y - D \;=\; 0 \tag{41}$$

the solution of which is:

$$y \;=\; \frac{A \cdot D - C \cdot B - 1 \pm \sqrt{(A \cdot D - C \cdot B - 1)^2 + 4 \cdot A \cdot D}}{2 \cdot A}. \tag{42}$$

Because A and D are positive, the magnitude of $\sqrt{(A \cdot D - C \cdot B - 1)^2 + 4 \cdot A \cdot D}$ is larger than the magnitude of $A \cdot D - C \cdot B - 1$. Hence, one of the solutions is positive and the other negative. Since $y = \mathrm{in}^c_{3-i}(\mathrm{A})$ can only be positive, the positive solution must be taken. Back substituting into Equation (40) gives the solution for $x = \mathrm{in}^c_i(\mathrm{A})$ to be:

$$x = \frac{C \cdot B - A \cdot D - 1 \pm \sqrt{(A \cdot D - C \cdot B - 1)^2 + 4 \cdot A \cdot D}}{2 \cdot C}. \tag{43}$$

Again, the one and only positive solution is the one that should be taken.

Equations(42) and (43) are a closed-form solution to the image separation problem. They provide expressions for the ambient light images $\mathrm{in}^c_i(\mathrm{A})$ and $\mathrm{in}^c_{3-i}(\mathrm{A})$ in terms of the final viewed images $\mathrm{in}^c_i(\infty)$ and $\mathrm{in}^c_{3-i}(\infty)$.

We have implemented this solution to the image separation problem, the results of which are shown in Figures 42 and 43. We found that where the intensity is roughly constant the original "drawn" image is restored very well. At points with high gradient, however, there are significant artifacts caused by the "defocusing" effect of several components in the system: the camera, the projector, and the image warping. Defocus effects are not incorporated in our image formation model. Potentially a more sophisticated image formation model could be derived. In the meantime we process the results in Figures 42 with a simple algorithm to remove most of the artifacts. We: (1) detect edges in the final viewed images, (2) compare the magnitudes of the edges at the two sites, and (3) smooth the edges at the "weaker" site based on the assumption that the edges originate from the side at which they are stronger. The results of applying this algorithm to the results in Figure 42 are included in Figure 43. Overall the separated images match the actual drawn images very well.

5.6 Automatic Gain Control

We have analyzed the Tele-Graffiti feedback loop assuming the cameras are set up with fixed apertures and shutter speeds to give a constant gain. Tele-Graffiti is normally operated with

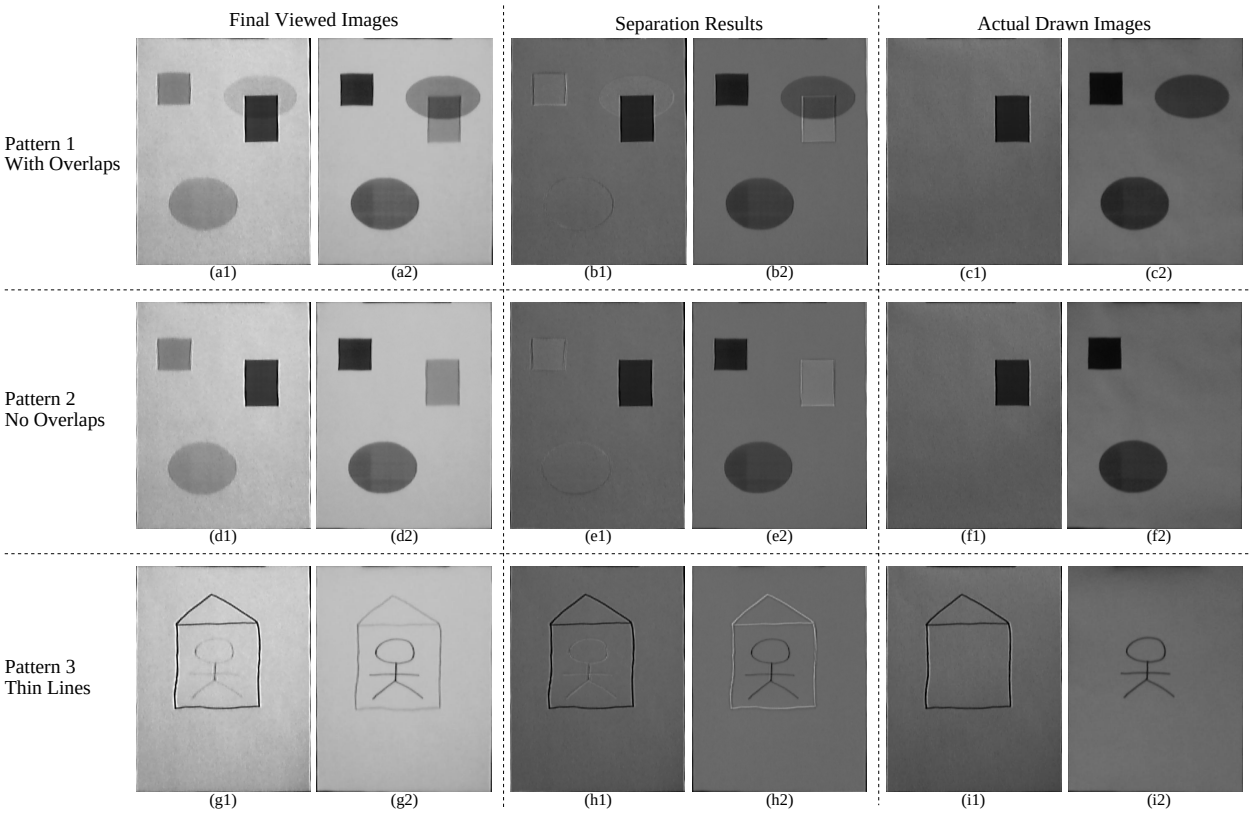


Figure 42: The results of image separation with three different sketches: (1) a sketch with overlaps, (2) a sketch without overlaps, and (3) a sketch with thin lines. The first column of each row contains the two final viewed images of the two sites from which we compute the separated images shown in the second column. The third column contains the actual drawn images at the two sites. The images are all warped to a normalized shape for ease of comparison. Overall our separation algorithm works very well, however, due to the "defocusing" effect of the whole system, the edges where the intensity difference in the drawn image changes too quickly cause artifacts in the separated images, particularly in sketch 3. To alleviate this effect, the images can be processed with a simple edge-smoothing algorithm, the results of which are shown in Figure 43.

the automatic gain control (AGC) switched on, however. See Section 2.4.5 for more details of the camera set up. How does using automatic gain control affect our analysis?

The analysis itself is not affected. The only thing that is affected is our ability to set the gain independently. Instead, AGC operates by adjusting the aperture (or shutter speed) to enforce the requirement that the average intensity in a certain area of the image (normally the center) is a fixed value, the exposure. When AGC is switched on, the camera gain is adjusted until Equation (21) is satisfied, with $\mathrm{in}_i^c(\infty)$ equal to the exposure. If the ambient

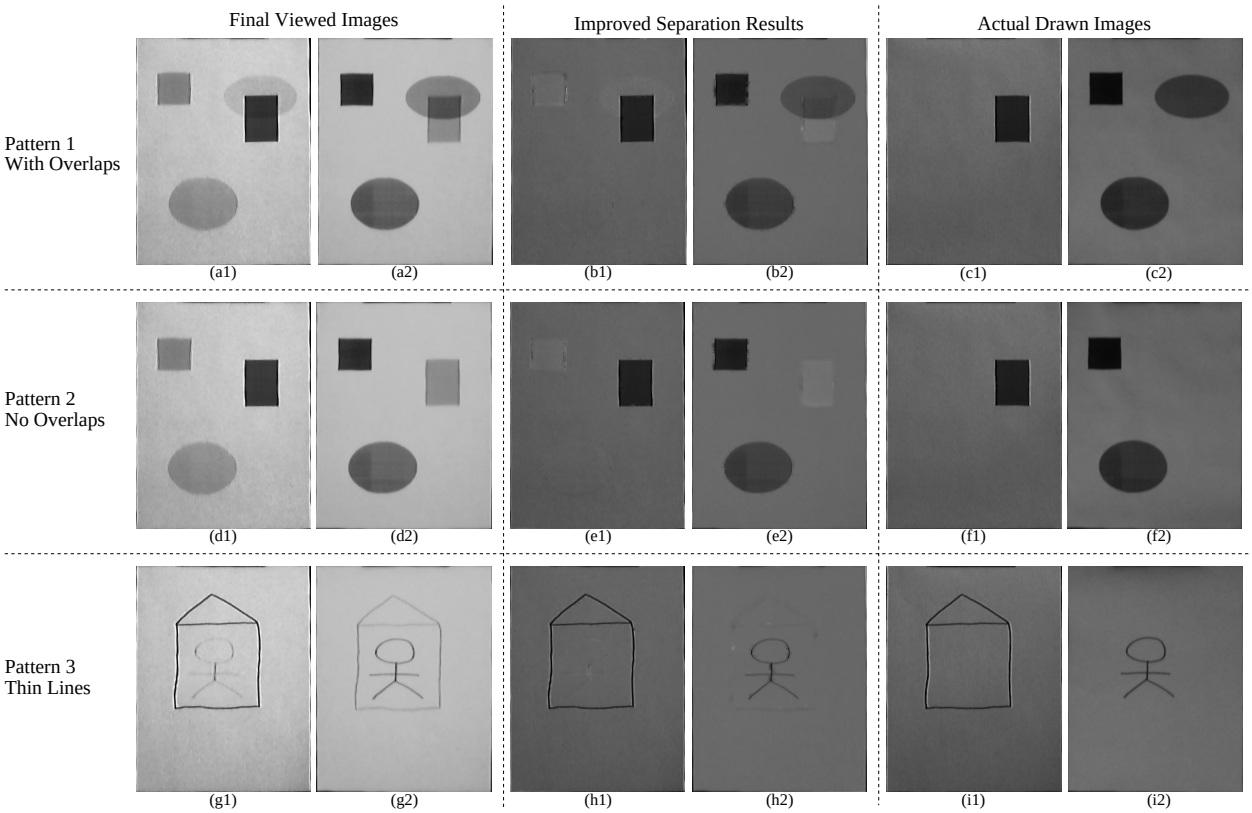


Figure 43: The results of image separation in Figure 42 after being processed by a simple algorithm to remove the artifacts caused by the system defocus. We: (1) detect edges in final viewed images, (2) compare the magnitudes of the edges at the two sites, and (3) smooth the edges at the "weaker" site based on the assumption that the edges originate from the side at which they are stronger.

light level is low, this results in a large gain $G_i \approx 1$ and the system has substantial visual echoing. If the ambient light level is very high, the result is a small gain $G_i \approx 0$ and it is hard to see what is written at the other site. See Figure 39 for examples of these phenomena.

When using automatic gain control we give up the freedom to set the gain independently. We cannot adjust it directly to obtain the desired image quality. Instead, the only parameter we can use to indirectly control the gain (and hence the image quality) is the amount of additional ambient light provided by the projector. In particular, the offset b is the one parameter we can use to change the gain and thereby the image quality. As we increase b the gain of the cameras will be reduced and there will be less visual echoing. On the other hand, as we increase b, the image from the other site will be less visible.

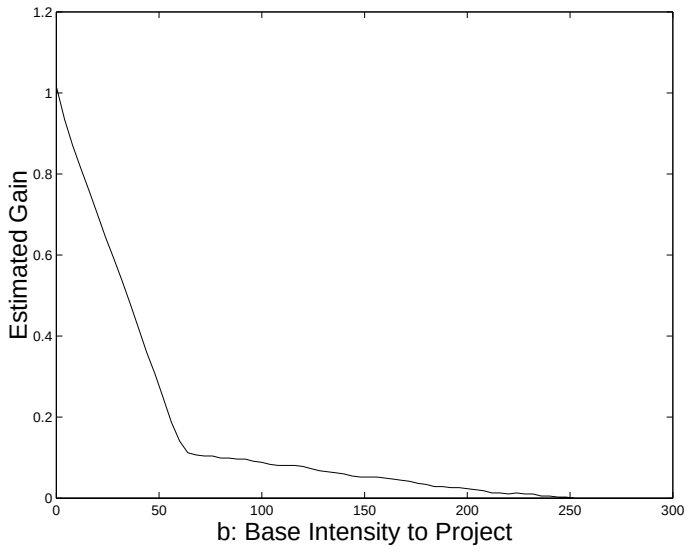


Figure 44: Estimated gain of an AGC camera with various base intensities b. We capture two images, one with a small black square ($\text{in}^p = 0$) projected in the middle of the paper, the other with a small grey square $\text{in}^p = 128$. We then estimate the average intensity of the difference of these two images. The gain of the system is then the average intensity of the difference divided by 128. The gains estimated with this algorithm are plotted for the base intensity offset b.

When AGC is switched on we can still (approximately) estimate the gain. We repeat the procedure outlined in Section 5.2.1. We capture two images, one with a small black square ($\text{in}^p = 0$) projected in the middle of the paper, the other with a small grey square $\text{in}^p = 128$. We then estimate the average intensity of the difference of these two images. The gain of the system is then the average intensity of the difference divided by 128. Figure 44 contains the graph of the estimated gain of the AGC camera with various base intensities b.

In Section 5.2.1 we set the fixed gain of the system by performing a binary search over the aperture of the camera until the gain is the desired value. When automatic gain control is switched on we can do the same. We just perform a binary search over b rather than over the aperture of the camera. The results of doing this are shown in Figure 45.

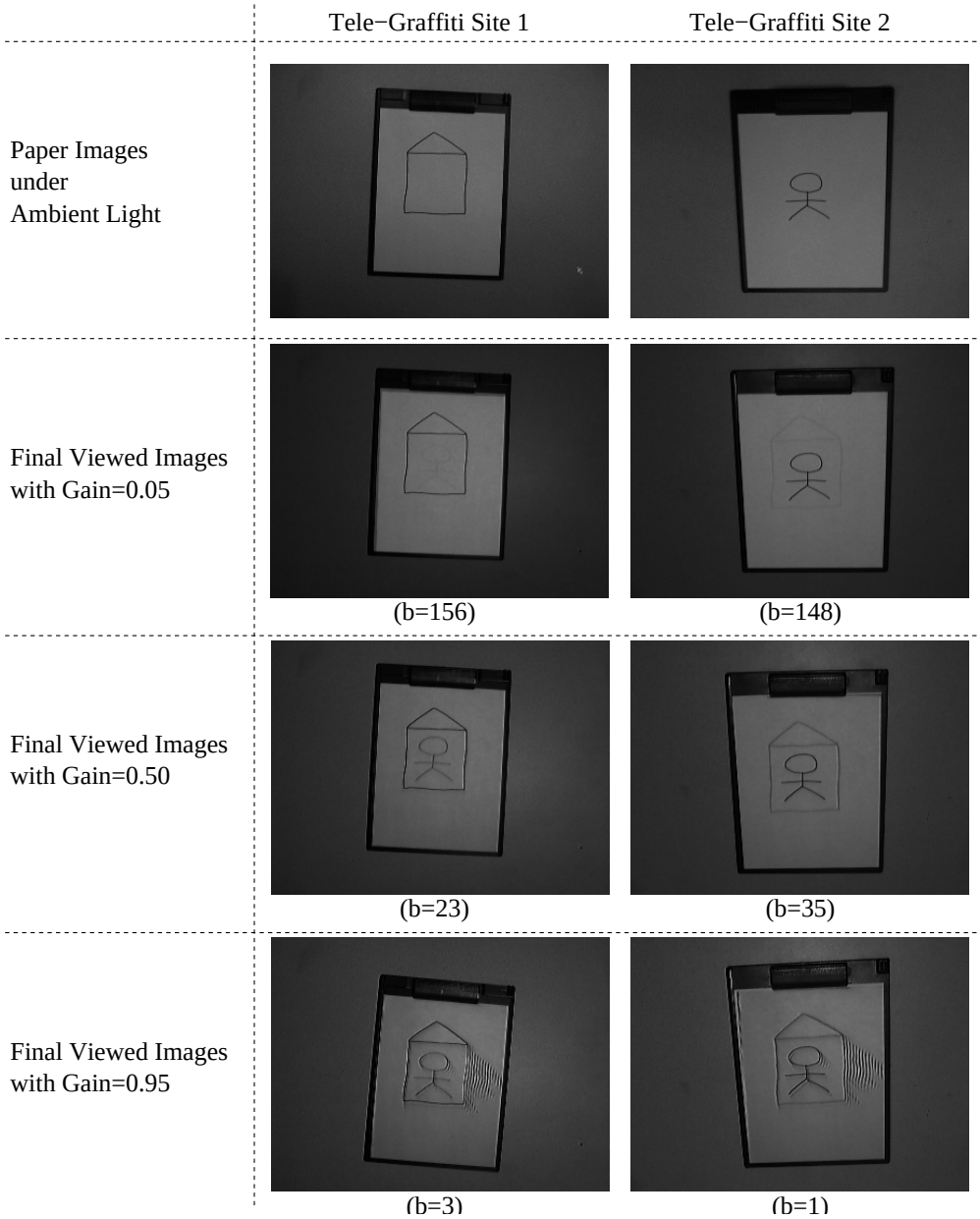


Figure 45: Final viewed images with automatic gain control turned on in the cameras. The images in the first row are the paper images of both sites captured under the ambient light only ($\mathrm{in}_i^c(\mathrm{A})$). Note that these images are not much darker than the final viewed images because of the automatic gain control. The final viewed images ($\mathrm{in}_i^c(\infty)$) with gain=0.05, 0.50, 0.95 are shown in the second, third and fourth rows respectively. To obtain these gains, we computed the base intensity to project b from Figure 44. The value of b used is shown under each pair of final images. With the small gain of 0.05 (the second row), the sketches at the other sites are hard to see. On the other hand, with the large gain of 0.95 (the fourth row), the "visual echoes" ruin the image quality. With the gain of 0.50 (the third row), we get the final images of best quality among these gains: i.e. we can see drawings of both sites clearly while there are no visual echoes.

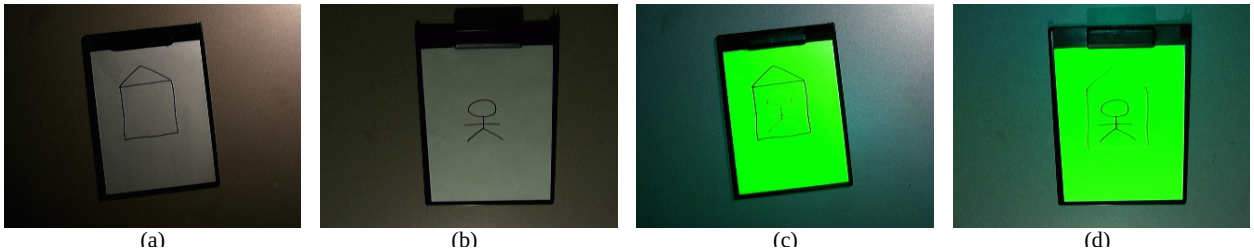


Figure 46: Color saturation caused by color-unbalanced cameras. In this case the green channel has by far the largest gain. The green channel quickly saturates and the paper is filled with green.

5.7 Color Imagery

Working with color images simply means that there is a separate gain for each color channel. Assuming the camera and projector are reasonably color balanced (see Section 2.4.6) these gains are fairly similar and the analysis above can be performed with the average of the three gains. If there is substantial difference in the gains, obtaining good image quality can be difficult however. If we set up the system so that the average gain (across color) of the paper is 0.5, the gain of one of the channels may be much larger; in extreme cases the gain may be ≈ 1.0. In such cases, one of the color channels may saturate. An example of such a situation is shown in Figure 46. In this figure we set up the system (without AGC) so that the average gain is 0.5. However, the cameras are poorly color balanced. The green channel is dominant, having by far the largest gain. When we run Tele-Graffiti the green channel quickly saturates resulting in the images in Figure 46(c) and (d). The solution to this problem is of course simply to color balance the cameras and projectors.

6 Conclusion

6.1 Summary

We have described Tele-Graffiti, a camera-projector based remote sketching system. The major contributions of Tele-Graffiti over existing camera-projector systems are:

Real-Time Paper Tracking and Video Transmission: We have developed a real-time paper tracking algorithm that allows the users of Tele-Graffiti to move the paper during operation of the system. The result is a much more versatile system. We have also developed a software architecture for Tele-Graffiti and implemented real-time video transmission over IP networks on standard 450MHz PCs.

Hand-Based User Interface and Automatic Summarization: We have added a user interface to Tele-Graffiti based on hand-tracking. The system requires no extra hardware, and operates with the Tele-Graffiti cameras. Infra-red cameras are not required. The user interface detects when the user has their hand over the paper and processes their interaction with user interface objects such as buttons and sliders. We have also developed an automatic summarization system for Tele-Graffiti based on detecting when the users have their hands over the paper. Such a system can automatically summarize a several minute long sketching session into a few 10's of frames.

Feedback Analysis for Gain Selection and Image Separation: We have analyzed the Tele-Graffiti feedback loop and derived the final steady state that the system converges to. We use this analysis to derive the optimal gain with which to set up the system to maximize the image quality; to minimize visual echoing, but to maximize how well the sketch from the other site is visible. We extended this analysis to Tele-Graffiti running with automatic gain control (AGC) and color images. We have also derived an algorithm to separate the two final viewed images (which are a weighted combination of the two sketches at the two sites) into the two component sketches.

6.2 Future Work

There are a wide variety of possible directions for future work. Three possibilities are:

3D Paper Tracking: Paper tracking is currently restricted to the plane of the desktop. Using structure-from-motion techniques such as [Faugeras and Lustman, 1988] it should

be possible to extend paper tracking to 3D. Other issues may arise, however, when implementing such a technique. For example, as the paper is brought closer to the camera and projector, the focus settings may need to be dynamically adjusted.

Sketch Interpretation: We have shown how to break the sketch down into a sequence of drawing steps by detecting when the hands move away from the paper. Far more could be done in terms of sketch interpretation. Interpreting sketches has a variety of applications from compression and summarization to retrieval and 3D figure input.

Techniques for Enhancing Video Quality: One goal of analyzing the feedback loop was to derive optimal settings for Tele-Graffiti to maximize the video quality. Much more could be done to improve image quality. Non-linear analysis of the feedback loop may allow better visibility of the image at the remote site, yet without increasing visual echoing. Also, super-resolution techniques such as "hallucination" could be used to improve the image resolution [Baker and Kanade, 2000a, Baker and Kanade, 2000b].

Acknowledgments

We would like thank Iain Matthews and Bart Nabbe for helping us design and build the Tele-Graffiti stands. See Figure 3. Iain also helped with the software architecture and Bart gave us big help controlling the Sony cameras under Linux. Panasonic donated the two projectors.

References

[80/20 Inc., 2002] 80/20 Inc. *Home Page of 80/20 Inc.* http://www.8020.net, 2002.

[Agrawala *et al.*, 1997] M. Agrawala, A. Beers, B. Froehlich, P. Hanrahan, I. MacDowall, and M. Bolas. The two-user responsive workbench: Support for collaboration through

individual views of a shared space. In *Proceedings of SIGGRAPH '97*, 1997. URL: http://www-graphics.stanford.edu/projects/RWB/.

[Aho *et al.*, 1974] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Baker and Kanade, 2000a] S. Baker and T. Kanade. Hallucinating faces. In *Proceedings of the Fourth International Conference on Automatic Face and Gesture Recognition*, Grenoble, France, March 2000.

[Baker and Kanade, 2000b] S. Baker and T. Kanade. Limits on super-resolution and how to break them. In *Proceedings of the 2000 IEEE Conference on Computer Vision and Pattern Recognition*, Hilton Head, South Carolina, June 2000.

[Baker and Shi, 2002] Simon Baker and Jianbo Shi. *Home Page of Tele-Graffiti project*. http://www.ri.cmu.edu/projects/project_423.html, 2002.

[Faugeras and Lustman, 1988] O.D. Faugeras and F. Lustman. Motion and structure from motion in a piecewise planar environment. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(3):485–508, 1988.

[Faugeras, 1993] O.D. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.

[Hall *et al.*, 1999] D. Hall, C. Le Gal, J. Martin, T. Kapuscinski, and J.L. Crowley. MagicBoard: A contribution to an intelligent office environment. In *Proceedings of the 7th International Symposium on Intelligent Robotic Systems*, 1999. http://iihm.imag.fr/demos-/magicboard/.

[Horn, 1996] B.K.P. Horn. *Robot Vision*. McGraw Hill, 1996.

[Intel Corporation, 1996] Intel Corporation. *Intel Architecture MMX Technology Programmer's Reference Manual*. http://cedar.intel.com/cgi-bin/ids.dll/content/content.jsp?-cntKey=Legacy::irtmp_PRM_10660&cntType=IDS_EDITORIAL, 1996.

[Krueger *et al.*, 1995] W. Krueger, C.A. Bohn, B. Froehlich, H. Schueth, and W. Strauss. The responsive workbench: A virtual work environment. *IEEE Computer*, 28(7):42–28, July 1995.

[Leibe *et al.*, 2000] B. Leibe, T. Starner, W. Ribarsky, Z. Wartell, D. Krum, B. Singletary, and L. Hodges. The perceptive workbench: Towards spontaneous and natural interaction in semi-immersive virtual environments. In *Proceedings of the IEEE Virtual Reality 2000 Conference*, 2000. http://www.gvu.gatech.edu/ccg/projects/perceptive_cga-/perceptive_cga.html.

[Mitsunaga and Nayar, 1999] T. Mitsunaga and S.K. Nayar. Radiometric self calibration. In *Proceedings of the IEEE Conference on Computer Vision and Paper Recognition*, 1999.

[Nayar and Nakagawa, 1994] S.K. Nayar and Y. Nakagawa. Shape from focus. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(8):824–831, August 1994.

[Nayar *et al.*, 1996] S.K. Nayar, W. Watanabe, and M. Noguchi. Real-time focus range sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(12):1186–1198, December 1996.

[Panasonic, 2000] Matsushita Electric Company of America, Panasonic. *SPEC FILE: Portable LCD Projector PT-L701U.* ftp://ftp.panasonic.com/pub/Panasonic/Drivers-/PBTS/brochures/S_PT-L701U.pdf, 2000.

[Plus Corporation, 2000] Plus Corporation. *PROJECTOR: U2-1080/870.* http://www.plus-vision.com/en/product/projector/u2-1080/index.html, 2000.

[Poynton, 1996] C.A. Poynton. *A Technical Introduction to Digital Video.* John Wiley & Sons, Inc., 1996.

[Raskar *et al.*, 1998] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive

displays. In *Proceedings of SIGGRAPH '98*, 1998. URL: http://www.cs.unc.edu/Research-/stc/office/.

[Sato *et al.*, 2000] Y. Sato, Y. Kobayashi, and H. Koike. Fast tracking of hands and fingertips in infrared images for augmented desk interface. In *Proceedings of the Fourth International Conference on Automatic Face and Gesture Recognition*, 2000.

[Sony Corporation, 1999] Sony Corporation. *SONY Digital Camera Module DFW-V500-/VL500*. http://www.sony.co.jp/en/Products/ISP/pdf/catalog/DFWV500E.pdf, 1999.

[Takao *et al.*, 2001] N. Takao, J. Shi, S. Baker, I. Matthews, and B. Nabbe. Tele-Graffiti: A paper-based remote sketching system. In *Proceedings of the 8th International Conference on Computer Vision*, Demo Session, 2001.

[Takao *et al.*, 2002] Naoya Takao, Jianbo Shi, and Simon Baker. *Unofficial Home Page of Tele-Graffiti*. http://www.cs.cmu.edu/~simonb/telegraffiti/, 2002.

[Trolltech AS, 2001] Trolltech AS. *Qt XML Module*. http://doc.trolltech.com/3.0/xml.html, 2001.

[Wellner, 1993] P. Wellner. Interacting with paper on the DigitalDesk. *Communications of the ACM*, 36(7):86–96, July 1993.

A Fast Image Processing

A.1 YUV to RGB Conversion

The Sending thread (see Section 3.4.1) converts the image to send from YUV to RGB before compressing and transmitting it. The formula for converting YUV to RGB [Poynton, 1996]

is as follows:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} \simeq \begin{pmatrix} 1.164 & 0 & 1.596 \\ 1.164 & -0.813 & -0.392 \\ 1.164 & 2.017 & 0 \end{pmatrix} \begin{pmatrix} Y-16 \\ U-128 \\ V-128 \end{pmatrix} \tag{44}$$

where $\simeq$ denotes the rounding operation to the range of $[0, 255]$. To avoid time-consuming floating point and rounding (conditional) operations during the run-time computation, we implemented the YUV-RGB conversion using table lookup: i.e. we build conversion tables off-line and look up the result at run-time. For the R component, a two-dimensional 256 $\times$ 256 array for Y and V is pre-computed consisting of the value:

$$1.164 \times (Y-16) + 1.596 \times (V-128).$$

Similarly, for the B component, another two-dimensional 256 $\times$ 256 array is pre-computed for Y and U consisting of the value:

$$1.164 \times (Y-16) + 2.017 \times (U-128).$$

To avoid a three-dimensional array for the G component computation, we pre-compute a two-dimensional array for U and V consisting of the value of:

$$\text{uv\_term} = 0.813 \times (U-128) + 0.392 \times (V-128).$$

The result of this is then used to index into a second pre-computed table for Y and uv_term consisting of the value of:

$$1.164 \times (Y-16) - \text{uv\_term}.$$

Table 8 shows a performance comparison of 3 implementations of YUV to RGB conversion: table lookup, computing with integer approximation, and floating point computation. Table lookup runs more than 3 times faster than the integer approximation, and 11 times faster than the floating point computation. Figure 47 contains the source code to build the lookup table, and Figure 48 contains the run-time YUV to RGB conversion function.

```
unsigned char m_vy2r[256][256], m_uy2b[256][256], m_yuv2g[640][256];
unsigned short m_uv[256][256];
#include <math.h>
#define ROUND(x) ((int)rint(x))

void SetTable(void)
{
  // build (V, Y) to R table
  for(int v = 0 ; v < 256 ; v++) for(int y = 0; y < 256 ; y++) {
    int r = ROUND(1.164 * (float)(y - 16) + 1.596 * (float)(v-128));
    if (r < 0) r = 0; if (r > 255) r = 255;
    m_vy2r[v][y] = r;
  }
  // build (U, Y) to B table
  for(int u = 0 ; u < 256 ; u++) for(int y = 0; y < 256 ; y++) {
    int b = ROUND(1.164 * (float)(y - 16) + 2.017 * (float)(u-128));
    if (b < 0) b = 0; if (b > 255) b = 255;
    m_uy2b[u][y] = b;
  }
  // build a table from (U,V) to intermediate value
  short uv_min = 32767, uv_max = -32768;
  short uv[256][256];
  for(int u = 0; u < 256 ; u++) for(int v = 0 ; v < 256 ; v++) {
    uv[u][v] = ROUND(0.813 * (float)(v-128) + 0.392 * (float)(u-128));
    if (uv_min > uv[u][v]) uv_min = uv[u][v];
    if (uv_max < uv[u][v]) uv_max = uv[u][v];
  }
  // shift the intermediate value and make the minimum value 0
  for(int u = 0 ; u < 256 ; u++) for(int v = 0 ; v < 256; v++) {
    m_uv[u][v] = uv[u][v] - uv_min;
  }
  //assert(uv_max - uv_min < 640);
  // build (intermediate-value, Y) to G table
  for(int y = 0 ; y < 256 ; y++) {
    for(int uv = uv_min ; uv <= uv_max ; uv++) {
      int g = ROUND(1.164 * (float)(y - 16) - (float)uv);
      if (g < 0) g = 0; if (g > 255) g = 255;
      m_yuv2g[uv - uv_min][y] = g;
    }
  }
}
```

Figure 47: Source code of `SetTable()` which pre-computes the lookup tables for YUV to RGB conversion. See Figure 48 for the online YUV to RGB conversion function.

Table 8: Timing results for three YUV to RGB conversion implementations. The numbers denote the time it took to convert an image of VGA (640 × 480) resolution on a 450MHz PentiumII PC.

	Table Lookup	Integer Approximation	Floating Point
Time	20 msec	72 msec	229 msec
Operations per Pixel	29.3	105.5	335

```
void yuv2rgb(unsigned char *YUV, unsigned char *RGB, int NumPixels)
{
  // YUV: YUV buffer(in), RGB: RGB buffer(out), NumPixels: number of pixels
  unsigned char *end = YUV + (NumPixels << 1);
  int u, v, y;

  while(YUV < end) {
    u = *YUV++;
    y = *YUV++;
    v = *YUV++;
    const unsigned char *y2r = m_vy2r[v];
    *RGB++ = y2r[y];

    const unsigned short uv_index = m_uv[u][v];
    *RGB++ = m_yuv2g[uv_index][y];

    const unsigned char *y2b = m_uy2b[u];
    *RGB++ = y2b[y];

    y = *YUV++;

    *RGB++ = y2r[y];
    *RGB++ = m_yuv2g[uv_index][y];
    *RGB++ = y2b[y];
  }
}
```

Figure 48: Source code of `yuv2rgb()` which converts YUV to RGB by looking up the pre-computed tables. See Figure 47 for the offline table building function.

A.2 YUV to Grayscale Conversion

As the first step in our paper detection algorithm, the captured image (640 × 480 YUV 4:2:2) is converted and down-sampled to a 320 × 240 grayscale image (Section 3.3.1). On

Table 9: Timing results for YUV to Grayscale conversion for a VGA (640×480) image and down-sampling to 320×240 pixels, with and without MMX support. The values of operations per pixel are shown in parentheses.

Operation(s)	with MMX	without MMX
YUV to Grayscale conversion	2.9 msec (4.2)	12.6 msec (18.5)
YUV to Grayscale conversion with down-sampling	1.7 msec (2.5)	14.3 msec (20.9)

YUV 4:2:2 mode, the Sony DFW-VL500's return the 4 bytes for two consecutive pixels in the following order: "U, Y1, V, Y2." Thus, for YUV to Grayscale conversion we only have to extract every other byte: "Y1, Y2." We developed an efficient implementation of this which makes use of the MMX functionality of the Pentium II processor [Intel Corporation, 1996]. Table 9 shows timing results for YUV to Grayscale conversion with and without MMX support. With MMX support, the conversion runs 4 times faster than without MMX. The source code for YUV to grayscale conversion utilizing MMX is shown in Figure 49.

We also developed a function which does both down-sampling and YUV to grayscale conversion at the same time with a small modification to this code: simply extracting the Y value for every other pixel in every other line instead of picking up every other byte. Table 9 shows the combined time for both operations. Utilizing MMX makes the combined operations 8 times faster.

A.3 Projector Linearization and Offset Addition by Table Lookup

In Section 5.1.5 we described how the non-linearity of the projector can be corrected by applying the transformation:

$$\mathrm{in}^p \longleftarrow \mathrm{R}^{-1}\left(\frac{\mathrm{R}(255) \times \mathrm{in}^p}{255}\right) \tag{45}$$

```
unsigned char Ymask[8] = { 0x00, 0xff, 0x00, 0xff, 0x00, 0xff, 0x00, 0xff };
                          // U,    Y1,   V,    Y2,   U,    Y3,   V,    Y4

void
yuv2gray(unsigned char *YUV, unsigned char *GRAY, int NumOfPixels)
{// YUV: pointer to YUV buffer, GRAY: grayscale buffer
 // NumofPixels: number of pixels to be converted::must be a multiple of 8.
  unsigned char *end = YUV + (NumOfPixels << 1);

  // this is a trick to avoid an optimizer bug in gcc 2.96.
  char dummy[10]; sprintf(dummy, "%x", (int)GRAY);

  // MM1 = 0xFF00FF00FF00FF00;
  __asm__("MOV %0, %%eax" : : "m"(Ymask) : "eax");
  __asm__("MOVQ (%eax), %mm1");

  while(YUV < end) {
    // load 8 bytes from YUV to MM0
    __asm__("MOVQ %0, %%mm0" : : "g" (*YUV));
    YUV += 8;
    // MM0 &= 0xFF00FF00FF00FF00; leave Ys only
    __asm__("PAND  %mm1, %mm0");
    // MM0 >>= 8;
    __asm__("PSRLQ   $8, %mm0");
    // now MM0 contains 4 bytes as (H)0 Y4 0 Y3 0 Y2 0 Y1(L)

    // again, load 8 bytes from YUV to MM2
    __asm__("MOVQ %0, %%mm2" : : "g" (*YUV));
    YUV += 8;
    // MM2 &= 0xFF00FF00FF00FF00;
    __asm__("PAND  %mm1, %mm2");
    // MM2 >>= 8;
    __asm__("PSRLQ   $8, %mm2");
    // now MM2 contains next 4 bytes as (H)0 Y8 0 Y7 0 Y6 0 Y5(L)

    // merge MM0 and MM2 into MM0(pack 16bitx8 into 8bitx8)
    __asm__("PACKUSWB %mm2, %mm0");
    // now MM0 contains the 8 bytes:(H)Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1(L)
    // store MM0 into GRAY
    __asm__("MOVQ %%mm0, %0" : "=m"(*GRAY));

    GRAY += 8;
  }
  __asm__("EMMS");// end of MMX utilization
}
```

Figure 49: Source code of `yuv2gray()` which utilizes MMX functions. Assumes gcc version 2.96 with GNU assembler version 2.10 or higher.

to the projected image in^p, where R is the response function of the projector. See Section 5.1.5 for more details. In Section 5.4 we described how the projected image can also be corrected:

$$\mathrm{in}_i^p \longleftarrow \mathrm{b} + \frac{\mathrm{in}_i^{\max} - \mathrm{b}}{\mathrm{in}_i^{\max}} \times \mathrm{in}_i^p \tag{46}$$

to add additional ambient light with "offset" b to the projected image to compensate for the lack of ambient light in the scene. These two transformations can be composed to give:

$$\mathrm{in}^p \longleftarrow \mathrm{R}^{-1}\left(\frac{\mathrm{R}(255) \times \mathrm{b} + \frac{\mathrm{in}_i^{\max} - \mathrm{b}}{\mathrm{in}_i^{\max}} \times \mathrm{in}_i^p}{255}\right). \tag{47}$$

This complex transformation can be implemented efficiently as a 1D table lookup similarly to how YUV to RGB conversion was performed in Appendix A.1. As an offline step we create a 1D array of 256 unsigned characters. For each value of in_i^p in [0,255] we pre-compute the results of the transformation in Equation (47) and store the result in the appropriate location in the array. During online processing to apply the transformation to an image in_i^p, each pixel in in_i^p is transformed simply by indexing into the array. This table lookup can be performed efficiently using code similar to that for YUV to RGB conversion in Figure 48.

A.4 Background Subtraction with Masking and Thresholding

As the first step of our hand tracking algorithm detailed in Section 4.1.1, background subtraction with paper masking, compensation for intensity change, and thresholding is performed. This set of computations can also be implemented efficiently by taking advantage of the Intel MMX instruction set [Intel Corporation, 1996]. In particular, we developed two functions which utilizes MMX: `PixelAverage()` and `BackgroundSubtraction()`.

`PixelAverage()` computes the average intensity of a grayscale image taking into account only the pixels that are not masked out. Figures 50 and 51 contains the source code.

Given three images (a grayscale image, a background image, and a mask image) and the average intensity of the background image, `BackgroundSubtraction()` does the following computations (in this example, the threshold intensity level is set to 32):

1. Compute the difference I_{dif} between the average intensity of the grayscale image and the average intensity of the background image by calling `PixelAverage()`.

2. Load 8 bytes each from the buffers of the grayscale image, the background image, and the mask image into the MMX registers. Denote each 8 bytes with G_i, B_i, and M_i respectively ($i = [0, 7]$).

3. Divide G_i and B_i by 2 to assure that the subtraction result will fit in 8 bits.

4. Compute $G_i = G_i - B_i - I_{dif}$. (Compensation for intensity change)

5. Compute $G_i = G_i$ NAND M_i. (Masking out)

6. Compute $P_i = G_i + 16$ with unsigned character saturation, then compare P_i with `0xff`, and replace P_i with `0xff` on match and with `0` otherwise.

7. Compute $Q_i = G_i - 16$ with unsigned character saturation, then compare Q_i with `0x00`, and replace Q_i with `0xff` on match and with `0` otherwise.

8. Compute $G_i = G_i$ NAND P_i then $G_i = G_i$ NAND Q_i. In these 3 steps, the bytes in G_i whose value is between 0xf0(-16) and 0xff(-1) or between 0x00(0) and 0x10(16) are thresholded to 0.

9. Store G_i in the result buffer.

See Figures 52, 53 and 54 for the source code.

Table 10 contains timing results for both intensity averaging and background subtraction, with and without MMX support. With MMX support, overall both functions run around 4 times faster than without MMX.

```
#define PIXEL_AVERAGE_SKIP 4//number of pixels to skip in average computation+1
static unsigned char all0001[8] = { 0x1,0x0,0x1,0x0,0x1,0x0,0x1,0x0 };

int
PixelAverage(unsigned char *m_buf, int m_bytes,
             unsigned char *maskbuf, int num_masked_pixels)
{// m_buf: image buffer to compute average
 // m_bytes: size of the image in bytes
 // maskbuf: image buffer of mask image
 // num_masked_pixels: number of non-zero (mask) pixels in mask image
  unsigned char *buf = m_buf;
  unsigned char *end = buf + m_bytes;

  // MM7 = 0;
  __asm__("PXOR %mm7, %mm7");
  // MM6 = 0x0001000100010001;
  __asm__("MOV %0, %%eax" : : "m"(all0001) : "eax");
  __asm__("MOVQ (%eax), %mm6");
  // MM5 = 0; // sum of the intensity
  __asm__("PXOR %mm5, %mm5");

  while(buf < end) {
    // load 8 bytes from buf to MM3
    __asm__("MOVQ %0, %%mm3" : : "g" (*buf));
    buf += 8 * PIXEL_AVERAGE_SKIP;
    // load 8 bytes from maskbuf to MM0
    __asm__("MOVQ %0, %%mm0" : : "g" (*maskbuf));
    maskbuf += 8 * PIXEL_AVERAGE_SKIP;

    // load another 8 bytes from buf to MM4
    __asm__("MOVQ %0, %%mm4" : : "g" (*buf));
    buf += 8 * PIXEL_AVERAGE_SKIP;

    // MM0 = MM3 NAND MM0 (mask out)
    __asm__("PANDN %mm3, %mm0");
    // convert a byte to a word in MM0 : MM0(lower 4 words) MM1(higher 4words)
    __asm__("MOVQ %mm0, %mm1");
    __asm__("PUNPCKLBW %mm7, %mm0");
    __asm__("PUNPCKHBW %mm7, %mm1");

    // load another 8 bytes from maskbuf to MM3
    __asm__("MOVQ %0, %%mm3" : : "g" (*maskbuf));
    maskbuf += 8 * PIXEL_AVERAGE_SKIP;

    // add up words in MM0 and MM1 into MM0
    __asm__("PADDUSW %mm1, %mm0");
```

Figure 50: Source code of `PixelAverage()` which computes the average intensity of a grayscale image taking into account only the pixels that are not masked out. Part 1 of 2, continued in Figure 51.

Table 10: Timing results for intensity averaging and background subtraction for a 320×240 grayscale image, with and without MMX support. The values of operations per pixel are shown in parentheses.

Operations	with MMX	without MMX
Intensity Averaging	0.3 msec (1.8)	0.8 msec (4.7)
Background Subtraction	1.3 msec (7.6)	5.4 msec (31.6)
Overall	1.6 msec (9.4)	6.2 msec (36.3)

```
    // MM3 = MM4 NAND MM3 (mask out)
    __asm__("PANDN %mm4, %mm3");
    // convert a byte to a word in MM3 : MM2(lower 4 words) MM3(higher 4words)
    __asm__("MOVQ %mm3, %mm2");
    __asm__("PUNPCKLBW %mm7, %mm2");
    __asm__("PUNPCKHBW %mm7, %mm3");
    // add up words in MM2 and MM3 into MM2
    __asm__("PADDUSW %mm3, %mm2");

    // add up words in MM0 and MM2 into MM0
    __asm__("PADDUSW %mm2, %mm0");

    // add up words in MM0
    __asm__("PMADDWD %mm6, %mm0");
    __asm__("PACKSSDW %mm7, %mm0");
    __asm__("PMADDWD %mm6, %mm0");
    // add MM0 (sum for 16 pixels) to MM5 (total sum)
    __asm__("PADDD %mm0, %mm5");
  }

  int avg[1];

  // save MM5(total sum) to avg[0]
  __asm__("MOVD %%mm5, %0" : "=m"(avg));
  __asm__("EMMS");

  avg[0] /= ((m_bytes - num_masked_pixels) / PIXEL_AVERAGE_SKIP);

  return avg[0];
}
```

Figure 51: Source code of `PixelAverage()` part 2 of 2, continued from Figure 50.

```
static unsigned char all10[8] = { 0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10 };
static unsigned char allFF[8] = { 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff };

void
BackgroundSubtract(int m_width, int m_height, unsigned char *m_buf,
                   unsigned char *bgbuf, int bg_avg,
                   unsigned char *maskbuf, int num_mask_pixels,
                   unsigned char *destbuf)
{
  // m_buf/bgbuf/maskbuf/destbuf: buffers of grayscale images
  //                              target/background/mask/destination
  // bg_avg: average intensity of the background image
  // num_mask_pixels: number of mask pixels in the mask image
  int m_bytes = m_width * m_height;
  int avgdif = PixelAverage(m_buf, m_bytes, mask, num_mask_pixels) - bg_avg;
  static char avgdifs[8];
  for(int i = 0 ; i < 8 ; i++) {
    avgdifs[i] = avgdif/2;
  }

  unsigned char *end = m_buf + m_bytes;

  // MM3 = 0
  __asm__("PXOR %mm3, %mm3");
  // MM5 = avgdifs[0..7]
  __asm__("MOV %0, %%eax" : : "m"(avgs));
  __asm__("MOVQ (%eax), %mm5");
  // MM6 = 0x1010101010101010
  __asm__("MOV %0, %%eax" : : "m"(all10));
  __asm__("MOVQ (%eax), %mm6");
  // MM7 = 0xffffffffffffffff
  __asm__("MOV %0, %%eax" : : "m"(allFF));
  __asm__("MOVQ (%eax), %mm7");
  // ESI = destbuf
  __asm__("MOV %0, %%esi" : : "m"(destbuf) : "esi");
  // EDI = maskbuf
  __asm__("MOV %0, %%edi" : : "m"(maskbuf) : "edi");
  // EDX = bgbuf
  __asm__("MOV %0, %%edx" : : "m"(bgbuf) : "edx");
  // ECX = end
  __asm__("MOV %0, %%ecx" : : "m"(end) : "ecx");
  // EAX = m_buf
  __asm__("MOV %0, %%eax" : : "m"(m_buf) : "eax");
```

Figure 52: Source code of `BackgroundSubtraction()` part 1 of 3, continued in Figure 53. In this part, the registers are initialized.

```
__asm__("_loop_start:");

__asm__("CMP %ecx, %eax"); // m_buf < end ?
__asm__("JGE _exit_loop");

// load 8 bytes from this image to MM0 and shift each byte right by 1bit
__asm__("MOVQ (%eax), %mm0");
__asm__("ADD $8, %eax"); // m_buf += 8;

__asm__("MOVQ %mm0, %mm2");
__asm__("PUNPCKLBW %mm3, %mm0");
__asm__("PUNPCKHBW %mm3, %mm2");
__asm__("PSRLW $1, %mm0");
__asm__("PSRLW $1, %mm2");
__asm__("PACKUSWB %mm2, %mm0");

// load 8 bytes from background to MM1 and shift each byte right by 1bit
__asm__("MOVQ (%edx), %mm1");
__asm__("ADD $8, %edx"); // bgbuf += 8;

__asm__("MOVQ %mm1, %mm2");
__asm__("PUNPCKLBW %mm3, %mm1");
__asm__("PUNPCKHBW %mm3, %mm2");
__asm__("PSRLW $1, %mm1");
__asm__("PSRLW $1, %mm2");
__asm__("PACKUSWB %mm2, %mm1");

// load 8 bytes from maskbuf to MM2
__asm__("MOVQ (%edi), %mm2");
__asm__("ADD $8, %edi");// maskbuf += 8;

// MM0(this image) -= MM1(background) (signed with saturation)
__asm__("PSUBSB %mm1, %mm0");

// MM2 -= MM5 (compensation for average difference)
__asm__("PSUBSB %mm5, %mm0");

// MM2 = MM2 NAND MM0 (masking out)
__asm__("PANDN %mm0, %mm2");
```

Figure 53: Source code of `BackgroundSubtraction()` part 2 of 3, continued from Figure 52 and in 54. In this part the following computations are performed. (1) Load 8 bytes each from the target grayscale image, the background image, and the mask image into the MMX registers. (2) Divide the bytes of grayscale image and background image by 2. (3) Subtract the background image from the target image. (4) Subtract the average difference between the target and the background from the result of (3) (average difference compensation). (5) Mask out the result with the mask image.

```
  // thresholding: eliminate bytes from -0x10 to 0x10
  // add 0x10 to unsigned bytes in MM2 with unsigned char saturation
  // (eliminates 0xF0(-16)..0xFF(-1))
  __asm__("MOVQ %mm2, %mm0");
  __asm__("PADDUSB %mm6, %mm0");
  // save bytes pattern of 0xFF to MM0
  __asm__("PCMPEQB %mm7, %mm0");

  // sub 0x10 from unsigned bytes in MM2 with unsigned char saturation
  // (eliminates 0x00(0)..0x10(16))
  __asm__("MOVQ %mm2, %mm1");
  __asm__("PSUBUSB %mm6, %mm1");
  // save bytes pattern of 0x00 to MM0
  __asm__("PCMPEQB %mm3, %mm1");

  // MM0 = MM2 NAND MM0
  __asm__("PANDN %mm2, %mm0");
  // MM1 = MM0 NAND MM1
  __asm__("PANDN %mm0, %mm1");

  // save 8 bytes in MM1 to destbuf
  __asm__("MOVQ %mm1, (%esi)");
  __asm__("ADD $8, %esi");// destbuf += 8;

  __asm__("JMP _loop_start");

  // end of the loop
  __asm__("_exit_loop:");
  __asm__("EMMS");// end of MMX utilization
}
```

Figure 54: Source code of `BackgroundSubtraction()` part 3 of 3, continued from Figure 53. In this part, constant value thresholding is performed with an addition and a subtraction with unsigned character saturation. Finally, the result is stored in the destination buffer.

B Defining User Interface Objects

Users define UI objects in a UI objects definition file. The definition includes: what UI objects to use (Button or Slider), the number of objects, their geometry, appearance, text to draw, pre-defined command to invoke, etc. In this appendix we describe more details of how the user interface is defined.

B.1 Pre-Defined Commands

The names of the available commands are defined at compile time. Currently the following commands are implemented.

- "RecordStart"

  Set the system parameter "record mode" to ON: i.e. tell the system to save the next frame after the hand moves away from the paper.

- "RecordStop"

  Set the system parameter "record mode" to OFF.

- "TogglePlaybackMode"

  Toggle the system parameter "playback mode."

- "SliderMove"

  Calls the pre-defined function with the slider parameter value (0-100) when it is changed by the hand.

B.2 UI Objects Definition File

The UI objects definition file is an XML-based text file [Trolltech AS, 2001]. Its format is exemplified by the example contained in the Figure 55, which is used in the session summarization and replay application of Tele-Graffiti's UI system (see Section 4.5). The syntax of the UI objects definition file is as follows.

- The object definitions are included in a tag pair `<uielements>` and `</uielements>`.
- A button object is defined by a tag `<button` and `>`, with its properties inside the tag.
- A slider object is defined by a tag `<slider` and `>`, with its properties inside the tag.

- Up to 32 buttons and sliders in total can be defined.
- Currently implemented properties for each object is shown in Table 11.
- All geometry parameters (top, left, width, height) are specified in projector coordinates.
- All the color parameters are specified with the format `#RRGGBB`, where `RR`, `GG` and `BB` are the hexadecimal representation of color components $[0, 255]$ red, green and blue.
- Comments can be written between `<!--` and `-->`.

```
<uielements>
<!-- A Button to toggle the "record mode." -->
<button
        top="288" left="32" width="128" height="48"
        bgcolor="#ffffff" fgcolor="#000000"
        borderwidth="2" bordercolor="#ffff00"
        visible="1" sticktoselect="1"
        onselect="RecordStart" onunselect="RecordStop"
        label="RECORD OFF" selectedlabel="RECORD ON"
        focusedcolor="#ff0000" selectedcolor="#00ffff"
/>
<!--
 The following button does nothing but indicate the number of recorded frames.
-->
<button
        top="352" left="32" width="64" height="48"
        bgcolor="#ffffff" fgcolor="#000000"
        borderwidth="2" bordercolor="#ffff00"
        visible="1" sticktoselect="0"
        label="0" selectedlabel="0"
        focusedcolor="#ffffff" selectedcolor="#ffffff"
/>
<!-- A button to toggle the "playback mode." -->
<button
        top="288" left="208" width="128" height="48"
        bgcolor="#ffffff" fgcolor="#000000"
        borderwidth="2" bordercolor="#ffff00"
        visible="1" sticktoselect="1"
        onselect="TogglePlayback" onunselect="TogglePlayback"
        label="PLAYBACK OFF" selectedlabel="PLAYBACK ON"
        focusedcolor="#ff0000" selectedcolor="#00ffff"
/>
<!--
 A slider to browse through the recorded frames.
 It initially is invisible and appears when the playback mode is ON.
-->
<slider
        top="352" left="136" width="200" height="48"
        bgcolor="#ffff00" fgcolor="#00ffff"
        borderwidth="2" bordercolor="#00ff00"
        visible="0" initvalue="0"
        onselect="SliderMove"
        focusedcolor="#ff0000" selectedcolor="#0000ff"
>
</uielements>
```

Figure 55: The UI objects definition file used in the session summarization and replay application.

Table 11: Implemented properties of UI objects.

Object	Property name	Description
(Common)	top	Y coordinate of top edge of the rectangle.
	left	X coordinate of the left edge of the rectangle.
	width	Width in pixels.
	height	Height in pixels.
	bgcolor	Background (filled) color in normal state.
	fgcolor	Foreground color. It is used to draw text label for button objects, and bar for slider objects.
	borderwidth	Width of the border which is drawn around the rectangle of the object.
	bordercolor	Color of the border.
	focusedcolor	Background color in focused state.
	selectedcolor	Background color in selected state.
	visible	Visibility ("0" or "1")
	onselect	Name of a pre-defined command which is executed when this object is selected.
	onunselect	Name of a pre-defined command which is executed when this object is unselected.
Button	label	Text to draw in the center of the button object in normal and focused state.
	selectedlabel	Text to draw in selected state.
	sticktoselect	Boolean value to decide the behavior of the button object when the hand moves away the object after it has been selected. True ("1") means that the object keeps its selected state until the hand moves over it again, while false ("0") means that the object returns to normal state as soon as the hand leaves it.
Slider	initvalue	Initial value between 0 and 100.