

## **Tele-Graffiti: A Camera-Projector Based Remote Sketching System with Hand-Based User Interface and Automatic Session Summarization**

**Naoya Takao** <sup>†</sup>

Matsushita Research  
Matsushita Corporation  
Osaka, Japan

**Jianbo Shi** <sup>†</sup>

The GRASP Laboratory  
University of Pennsylvania  
Philadelphia, PA 19104

**Simon Baker**

The Robotics Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

### **Abstract**

One way to build a remote sketching system is to use a video camera to image what each user draws at their site, transmit the video to the other sites, and display it there using an LCD projector. Such camera-projector based remote sketching systems date back to Paul Wellner's (largely unimplemented) Xerox Double DigitalDesk. To make such a system usable, however, the users have to be able to move the paper on which they are drawing, they have to be able to interact with the system using a convenient interface, and sketching sessions must be stored in a compact format so that they can be replayed later. We have recently developed Tele-Graffiti, a remote sketching system with the following three features: (1) real-time paper tracking to allow the users to move their paper during system operation, (2) a hand based user interface, and (3) automatic session summarization and playback. In this paper, we describe the design, implementation, and performance of Tele-Graffiti.

**Keywords:** Camera-projector based remote sketching systems, remote communication and collaboration, video compression and transmission, paper detection and tracking, hand-based user interfaces, automatic summarization, archiving, and playback.

<sup>†</sup> Research conducted while at The Robotics Institute, Carnegie Mellon University.



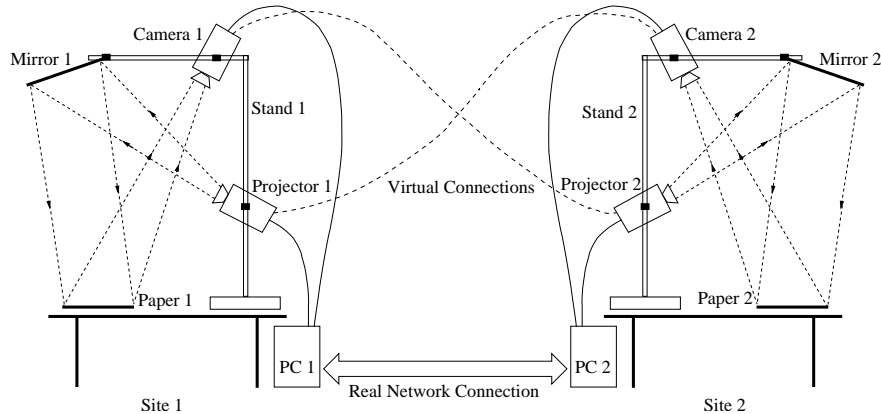


Figure 1: A schematic diagram of a 2 site camera-projector based remote sketching system [Wellner, 1993, Takao *et al.*, 2001]. The cameras at both sites continuously capture video of what is written on the paper. The video is digitized, compressed, and transmitted to the other site where it is projected onto the paper. Note that the projected video must be warped appropriately to align it correctly with the paper.

## 1 Introduction

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 rather than using a different pen. 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 video to the other end, and display it there using an *projector*. See Figure 1 for a schematic diagram of such a system. 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 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 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]. Recently, cameras and projectors have also been combined to develop smart displays [Sukthankar *et al.*, 2001] with automatic keystone correction, laser pointer control, automatic alignment of multiple displays, and shadow elimination. Although this list is by no means comprehensive, it clearly demonstrates the growing interest in such systems.

Tele-Graffiti [Takao *et al.*, 2001, Takao *et al.*, 2002] is a camera-projector based remote sketching system that we have recently designed and implemented. Although the Xerox “Double Digital-Desk” [Wellner, 1993] is an elegant idea, there are a number of technical problems that need to be solved to implement such a real-time system and to make it both practical and usable:

**Real-Time Paper Tracking:** The users of a remote sketching system 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. Such functionality is not available in current systems such as [WolfVision, 2002].

**Video Transmission:** 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.

**Providing a Suitable User Interface:** A remote sketching system would be much more useful if it could be controlled without using the keyboard or mouse. Instead it would be best if the user could control the system just using their hands.

**Sketch Summarization:** While remote sketching systems help users to communicate through their sketches, such a system would be much more useful if it has functions for sketch summarization, recording, and replay.

We have implemented all of these functions in Tele-Graffiti [Takao *et al.*, 2002]. Overall, the system software, including paper tracking, user interface processing, and summarization, runs at 30Hz on a PC with dual 450MHz Pentium-II processors. The video quality depends on the network bandwidth. With typical network parameters a video rate of around 10 frames per second is achieved. In this paper we describe the design and implementation of Tele-Graffiti.

## 2 Tele-Graffiti Hardware and Calibration

In this section we briefly describe the Tele-Graffiti hardware and the geometric calibration of the camera and projector. More details of the set-up of the system are included in [Takao *et al.*, 2002].

### 2.1 Hardware

A schematic diagram of a 2-site Tele-Graffiti system is contained in Figure 1. Figure 2 shows photos of 2 real Tele-Graffiti systems. Each Tele-Graffiti site contains the following components:

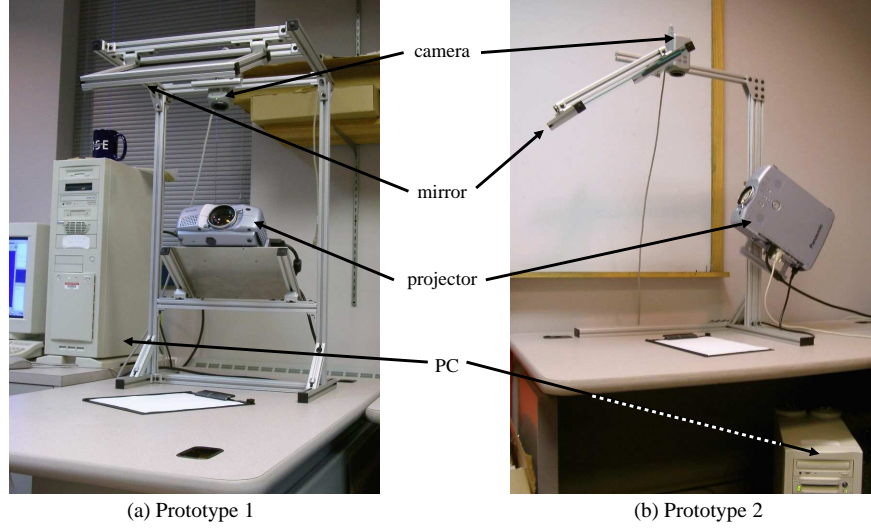


Figure 2: Two prototype Tele-Graffiti systems. (a) Prototype 1: The projector is mounted horizontally on a supporting plate. (b) Prototype 2: The projector is mounted vertically on the pillar of the stand.

**Computer:** Each Tele-Graffiti site has a PC with dual 450MHz Pentium-II processors, an “NVIDIA GeForce2 GTS” video card, and an OrangeLink Firewire (IEEE1394) card.

**Projector:** We use an XGA resolution ( $1024 \times 768$  pixels) Panasonic PT-L701U LCD projector.

**Camera:** We use a Firewire (IEEE1394) Sony DFW-VL500 camera, which we run at VGA ( $640 \times 480$  pixels) resolution.

**Stand:** We constructed 2 different prototype stands to hold the camera and projector in a compact configuration. In Figure 2(a) the projector is mounted horizontally on a supporting plate, while in Figure 2(b) it is mounted vertically on the pillar.

**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.

## 2.2 Geometric Calibration of the Camera-Projector Relationship

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 perspective imaging model as the camera (with the light rays in the reverse direction). Assuming the paper is



Figure 3: 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 (1).

planar, the relationship between camera and projector coordinates is:

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

where  $H_{pc}$  is a  $3 \times 3$  homography, 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 information to break the four-fold orientation ambiguity), and (4) solving for  $H_{pc}$  using Equation (1) and the 4 pairs of projector-camera coordinates. See Figure 3 for an illustration of this calibration process.

### 3 Tele-Graffiti System Software

Each Tele-Graffiti site must continuously capture video from the camera, track the paper in the video, warp the video from the other site so that it is aligned with the paper, and communicate with the other site. We now describe the system software that performs all of these tasks.

#### 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 4 for a diagram of the 4 threads. The 4 threads share the following data:

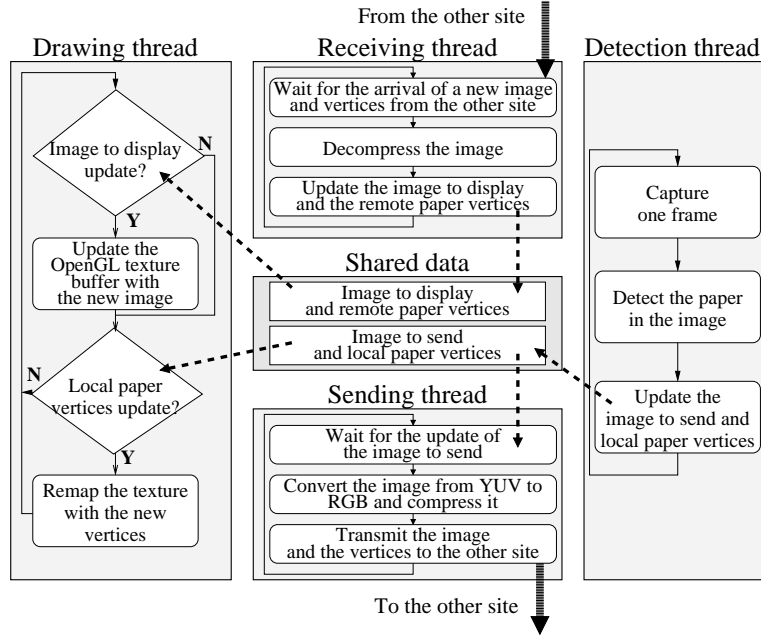


Figure 4: Tele-Graffiti software architecture. The system consists of 4 threads sharing a common data-structure. The Drawing thread continuously displays the latest image received from the other site (the ‘image to display’) using the latest estimate of the paper vertices (the ‘local paper vertices’). 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 decompresses them and updates the ‘image to display’. The Sending thread (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 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.

**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  $640 \times 480$  image which is just large enough to include the detected paper.

**Local Paper Vertices:** The estimated paper vertices in the captured  $640 \times 480$  image.

The first two of 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.



Figure 5: Image warping. The image to display (a). The location of the paper estimated by the paper tracking algorithm described in 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).

## 3.2 Drawing Thread

The Drawing 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. A 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 the vertices in the projector to which they must be mapped. While the former are 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 (1), 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 5 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 sets of four vertices each for 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 being used.) When warping a quadrangle, OpenGL divides the quadrangle into two triangles and warps them



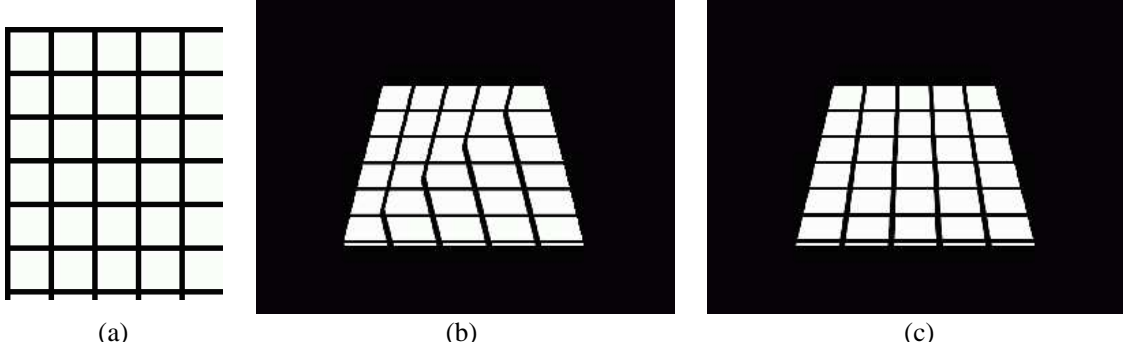


Figure 6: OpenGL texture mapping. (a) Original image. (b) Result image warped with four vertices (one quadrangle). (c) Improved warped image by dividing into a large number of quadrangles.

independently, each with a separate affine warp. Thus a discontinuity can sometimes be observed along the diagonal line in the destination image. See Figure 6(b) for an example.

To avoid this effect, we subdivide the paper into  $16 \times 16$  quadrangles. We calculate the coordinates of the  $17 \times 17$  vertices for both source and destination images, and tell OpenGL to warp these quadrangles all at once. See Figure 6(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 7 for an example of this image cropping.

4. Notifies the Drawing thread of the update.

An overview of our paper tracking algorithm is shown in Figure 8. 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. A breakdown of the timing of the algorithm is given in Table 3.

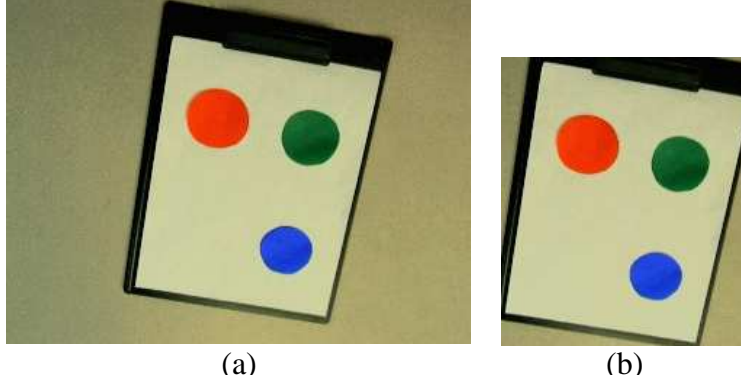


Figure 7: Creating the image to send. The Paper Detection thread detects the paper in the captured image (a) and estimates the paper vertices. The image to send (b) is created by cropping the captured image so that it is just large enough to include the detected paper image. In this example, the image to send is  $346 \times 400 = 138\text{K}$  pixels compared to  $640 \times 480 = 307\text{K}$  pixels for the captured image.

### 3.3.1 Image Down-Sampling and Smoothing

First the captured image ( $640 \times 480$  YUV 4:2:2) is converted to grayscale and down-sampled to a  $320 \times 240$  image. (The down-sampling step is not necessary with a faster CPU.) Then it is smoothed in both the horizontal and vertical directions. See [Takao *et al.*, 2002] for an efficient MMX implementation of YUV to Grayscale conversion and down-sampling.

### 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 the magnitude of  $G(x, y)$  using the following condition

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

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

$$E_i = (x_i, y_i, \theta_i) \quad (i = 1, 2, \dots, n) \quad (3)$$

$$\theta_i = \arctan \left[ \frac{\left(\frac{dI}{dy}\right)_i}{\left(\frac{dI}{dx}\right)_i} \right]. \quad (4)$$

### Tele-Graffiti Paper Tracking Algorithm

- (1) Down-sample and smooth the captured image
- (2) Detect edges in the downsampled 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) Robustly fit lines to each edge group
- (7) Estimate the paper orientation: Break the four-fold ambiguity

Figure 8: An overview of the Tele-Graffiti paper tracking algorithm. First, edge pixels are detected in the down-sampled and smoothed image and any projected edges removed. Next, the edges are split into four groups each of which are divided into paper-clipboard and clipboard-desk edges. Finally, a line is robustly fit to each of the paper-clipboard edge groups and the paper orientation ambiguity is resolved.

See Figure 9(b) for an example of the detected edges in Figure 9(a).

#### 3.3.3 Removal of Any Projected Edges

To keep the previously projected image from confusing the tracking algorithm (see Figure 9(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 9(c) for an example. Notice how most of the edges from the projected paper in Figure 9(b) have disappeared.

#### 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 9(d) contains the histogram of the edge angles, and Figure 9(e) shows the edges that are close to the 4 peaks in the histogram. The colors in Figure 9(e) correspond to the peak to which the edges belong to.

#### 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 9(e)). 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

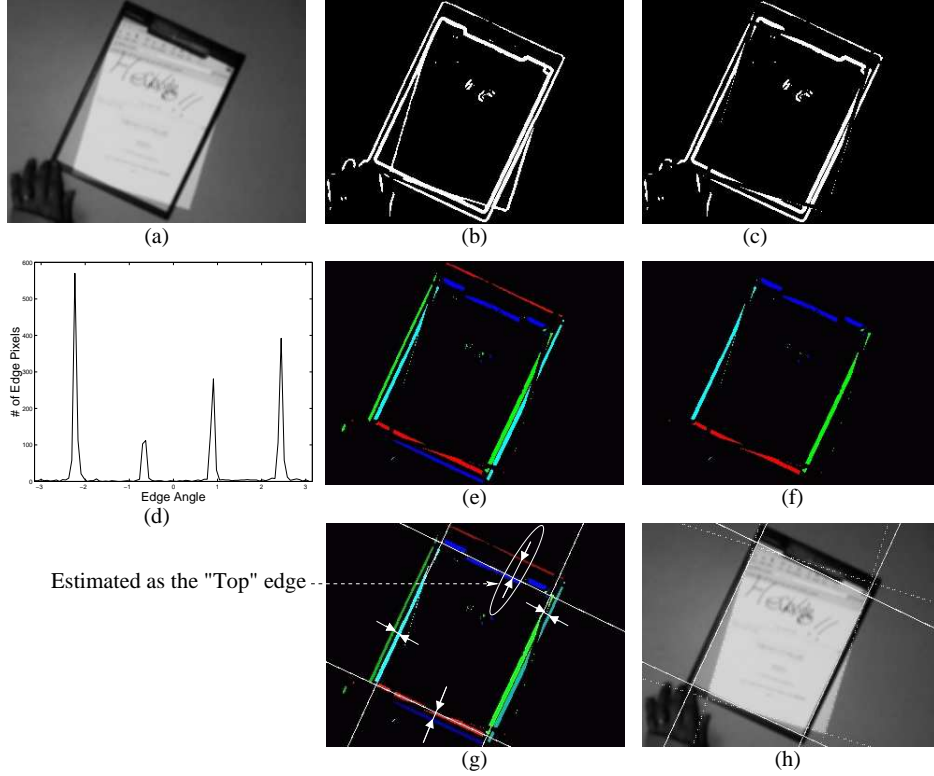


Figure 9: An illustration of the Tele-Graffiti paper tracking algorithm. First edge pixels are detected (b) in the captured image (a). They come from the paper, clipboard, hand, and the projection of the paper in the previous time step. Edge pixels that lie within a fixed distance from the previously 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.

is estimated as the mean of the edge locations.) See Figures 9(e) and (f) for an example. Figure 9(f) shows only the paper-clipboard edges after the clipboard-desk edges have been removed.

### 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 10. Let us denote the line as Equation (5).

$$aX + bY + c = 0 \quad (5)$$

### Tele-Graffiti's Robust Line Fitting Algorithm

- (0) Initialize  $a$  and  $b$  in Equation (5) from the average angle of the edges in Equation 6.
- Repeat (1)-(6) up to 5 times.
- (1) If there are not enough edges, exit with an error.
  - (2) Calculate the average X and Y coordinates of the edges.
  - (3) Set  $c$  so that the Equation (5) 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 10: An overview of Tele-Graffiti's robust line fitting algorithm.

To avoid repeating costly least-squares computations, the algorithm first estimates the slope of the line ( $a$ ,  $b$ ) as the average of the edge angles:

$$\theta_{avg} = \frac{\sum_{i=1}^n \theta_i}{n}, \quad a = \cos(\theta_{avg}), \quad b = \sin(\theta_{avg}) \quad (6)$$

Next, the algorithm iteratively performs outlier removal up to 5 times. It repeats the following: (1)  $c$  is calculated from Equation (5) 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 remaining edges to reliably fit a line (failure), or the number of iterations has passed the limit (failure). Finally, if the above iteration stopped successfully, least-squares fitting is applied to the remaining edges to re-calculate the line parameters ( $a$ ,  $b$ ,  $c$ ). See Figure 9(g) for the results of line fitting.

#### 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 ambiguity. The “top” edge is the one with the larger average distance between the clipboard-desk edges and the paper-clipboard line (in the pair separated by the larger distance.) See Figure 9(g) for an example.

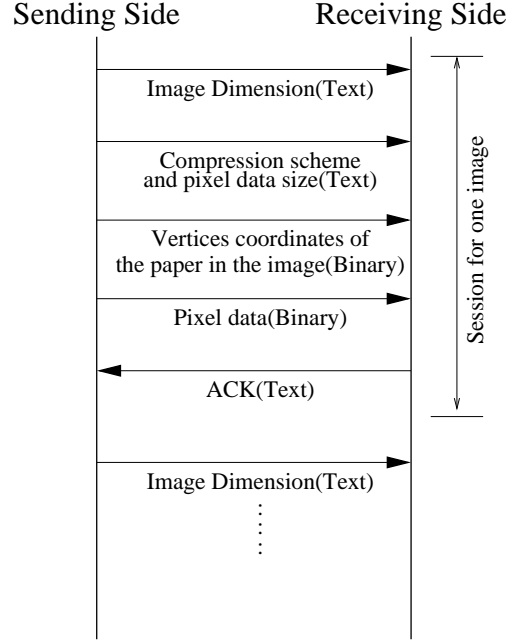


Figure 11: The Tele-Graffiti communication protocol. The sending side takes the initiative by sending the image dimension (width  $\times$  height  $\times$  color bands), the compression scheme ('No compression' or 'JPEG' or 'Run-length'), and the number of bytes of pixel data to follow, in text format. It then sends the  $17 \times 17$  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 after it has successfully received (and decompressed) the whole image.

## 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 reason, 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:** The Sending thread continuously converts the most recent image at this site (the image to send) from YUV to RGB, compresses it, and transmits it to the other site along with the estimated local paper vertices. (See [Takao *et al.*, 2002] for the details of our fast YUV to RGB conversion algorithm.) 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 it being overwritten.

**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.

The communication protocol is very simple. See Figure 11 for the details.

### 3.4.2 Image Compression

As mentioned in Section 3.1, the image to send is typically around  $40K \sim 160K$  pixels: i.e. around  $120K \sim 480K$ bytes 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, the 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 is 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.

Table 1 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: 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. We observe the following points:

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 (actually about 20 times) longer than over 100Base-T. The fact that the performance drop-off is 20 times rather than 10 times is explained by dramatically increased “network congestion.” As the two data streams flow in opposite directions the collision of packets from both sites is very likely to occur, significantly reducing the network throughput.

Table 1: Image transmission timing results with various compression schemes over 100Base-T and 10Base-T networks, measured in milliseconds. With no compression, the transmission time is largely determined by the network bandwidth. The drop-off in performance from 100Base-T to 10Base-T is about a factor of 20 rather than the factor of 10 that might be expected because of increased network congestion. With JPEG compression, the transmission time is largely determined by the compression time. If the CPU's were significantly faster, the overall communication time would be substantially less.

Compression Scheme (Typical compression rate)		No Compression (100%)		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 for one image		51.9	930.0	84.0	337.3	85.1	86.2

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.

### 3.5 Software System Timing Performance

Timing results for the Drawing thread and the Paper Detection thread are shown in Tables 2 and 3. On dual 450MHz Pentium PCs, these threads operate (comfortably) at 30Hz. Timing results for the communications threads are shown in Table 4 (Sending thread) and Table 5 (Receiving thread). As we discussed in Section 3.4.2, 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 in CPU time. Also note that the steps marked with “\*” (asterisk) spend most of their time idling (i.e. waiting for events or resources) rather than actually computing.



Table 2: 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

Table 3: Timing results for the Paper Detection thread. Paper detection operates comfortably at 30Hz. Around 10msec per loop is spent idling waiting for image capture. We also include an estimate of the average number of CPU operations spent for each of the  $640 \times 480 = 307K$  pixels in the captured image.

Steps		Timing (msec)	Operations per Pixel
Image capture* (Wait for the next frame)		10.6	—
Paper detection		19.7	115.4
	Down-sampling (including YUV to Grayscale conversion)	(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	—

## 4 A Hand-Based User Interface

We have implemented a user interface for Tele-Graffiti. 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 has just placed their hand over the paper or when they have just retracted their hand from over the paper. UI objects are rectangular 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:

Table 4: Timing results for the Sending thread (with JPEG compression over a 100Base-T network). Overall the Sending thread operates at 8Hz. Approximately the same results are obtained over 10Base-T networks.

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 5: Timing results for the Receiving thread (with JPEG compression over a 100Base-T network). The Receiving thread operates at 8Hz. Approximately the same results are obtained over 10Base-T networks.

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

“Button” and “Slider.” Button objects are used to toggle a “mode”. Slider objects are used to select a value from a range. See [Freeman *et al.*, 1998] for another example of a vision based slider.

## 4.1 Hand Tracking

There are several possible ways of tracking the user’s hands. (See [Crowley and Coutaz, 1995, Pavlovic *et al.*, 1997, Chipolla and Pentland, 1998] for surveys.) One possibility is to use a color-based hand tracking algorithm [Lien and Huang, 1998, Quek *et al.*, 1995]. 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 hand and to the lighting conditions, as pointed out in [Sato *et al.*, 2000, von Hardenburgh and Bernard, 2001]. An additional complication is that some LCD projectors time-multiplex color. Without synchronizing 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

### 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 12: The Tele-Graffiti hand tracking algorithm. We base our algorithm on background subtraction because it is fast, robust, and does not require any special hardware such as an infra-red camera.

*et al.*, 2000, Oka *et al.*, 2002]. The color projected by the projector is then not important because the camera primarily only images the “heat” radiated by the users hands. Like [von Hardenburgh and Bernard, 2001], we base our hand tracking algorithm on background subtraction because it is fast, robust, and does not require any special hardware. Figure 12 contains an overview of our algorithm. We now discuss each step in turn.

#### 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.

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  $I_{avg}^0$  is calculated from the background image. See Figure 13(a) for an example background image.

**Create a Mask for the Paper and Clipboard:** Create a mask which covers the paper and clipboard. This mask is used in background subtraction. See Figure 13(c) for an example.

**Calculate the Average Background Intensity:** In order to compensate for the difference in background intensity caused by the camera’s AGC, calculate the average intensity  $I_{avg}^i$  of the current 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 and so use it because it can be computed more efficiently using MMX instructions. See [Takao *et al.*, 2002] for the details.

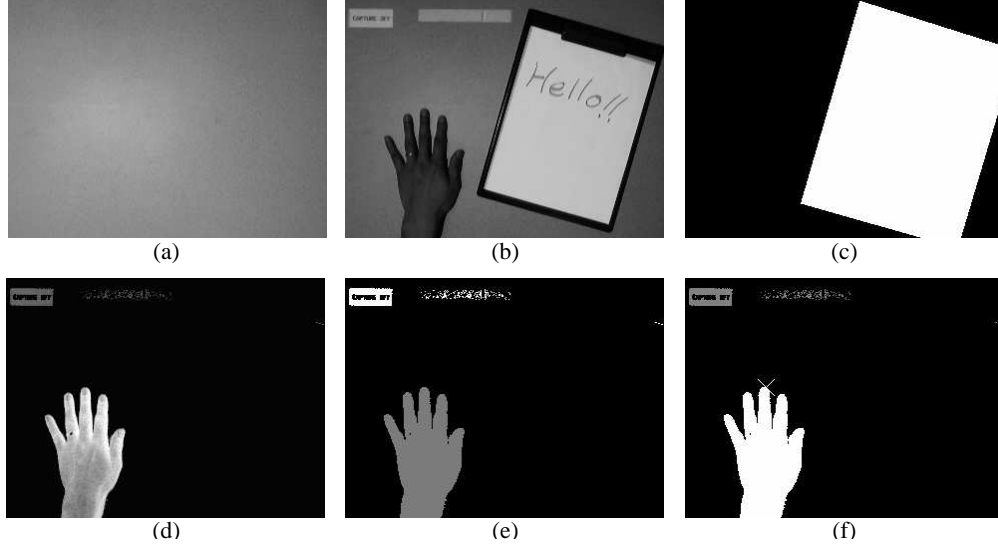


Figure 13: An illustration of our hand tracking algorithm. (a) A background image captured at system startup. (b) An image captured while the system is running, which includes the paper, clipboard, user's hand, and projected UI objects. Note the difference in brightness of the background between (a) and (b) caused by the camera's AGC. (c) A mask image which covers the paper and clipboard, created from 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) The result of hand component determination. The detected hand is shown as the brightest object.

**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 pre-defined threshold  $C$  and average difference compensation, as is shown in the following equation.

$$d(x, y) = I^i(x, y) - I^0(x, y) - (I_{avg}^i - I_{avg}^0) \quad (7)$$

$$D(x, y) = \begin{cases} 0 & (\text{if } d(x, y) \leq C) \\ d(x, y) & (\text{if } d(x, y) > C) \end{cases} \quad (8)$$

See Figure 13(d) for an example background subtraction result.

In our implementation, the difference image  $D(x, y)$  is divided by two and shifted by 128 so that the result can be represented in an 8 bit grayscale image. Thus, intensity 128 means no difference. We also utilize the Pentium-II MMX instruction set to optimize the speed of computing the average pixel intensity and computing the difference image. Both of these steps must take into account the mask image. See [Takao *et al.*, 2002] for the details of our implementation.

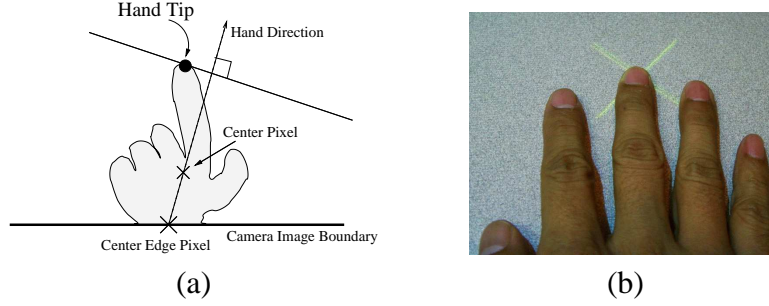


Figure 14: Hand tip localization. (a) First compute the orientation of the hand component from the “center edge pixel” and the “center pixel.” Then find the farthest pixel from the edge pixel in the hand direction. (b) The detected hand tip is displayed on the work area with a yellow “X.”

#### 4.1.2 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 13(e) shows an example result.

#### 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 13(f) for an example result.

#### 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 in 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 14 for an illustration of the algorithm.

1. Compute the direction of the hand from the following two points:

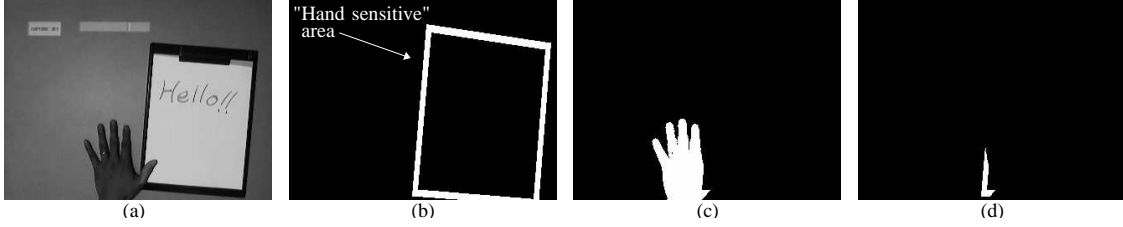


Figure 15: Hand-over-paper detection. (a) Captured image. (b) “Hand sensitive” area. A mask is created based on the paper vertices which is slightly larger than the paper mask used for background subtraction (see Figure 13(c)). The area between these two masks is the hand-sensitive area. (c) Detected hand component. Note that the hand is partly masked by the paper mask. (d) The result of intersecting (b) and (c). If there are any pixels in the result, the hand is over the paper, and vice versa.

- (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 hand component pixels.

2. Find the farthest pixel from the edge pixel in the hand component in the hand direction.

See Figures 13(f) and 14(b) for examples of hand tip localization.

## 4.2 Hand Over Paper Detection

In general, it is hard to determine whether the user is drawing or not. We assume that if the user has their hand over the paper they are drawing. If their hand is away from the paper, they are definitely not drawing. Once we have detected the paper and found the hand component, we then determine whether the hand is over the paper. 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 15(b).

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

If there is at least one pixel from the hand component within “hand sensitive” area, the hand is over the paper. See Figure 15(d) for an illustration of this step.

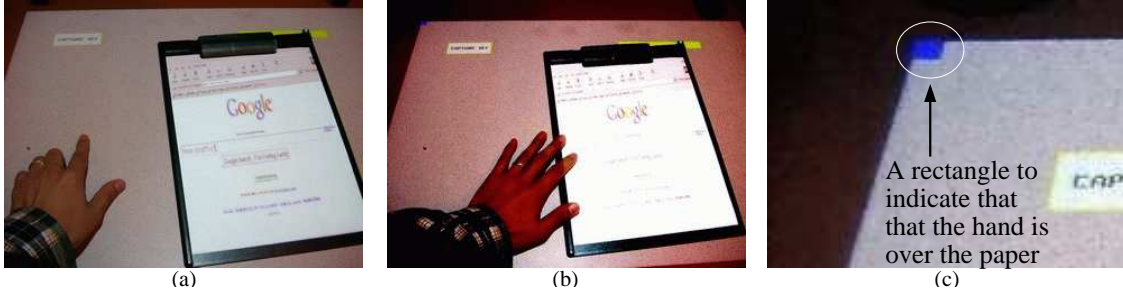


Figure 16: Results of Hand-over-paper detection. (a) Hand detection works simultaneously with paper tracking. (b)(c) When a hand is detected over the paper, a small blue rectangle is displayed in the top left corner of the work area so that the user knows the system has detected the event.

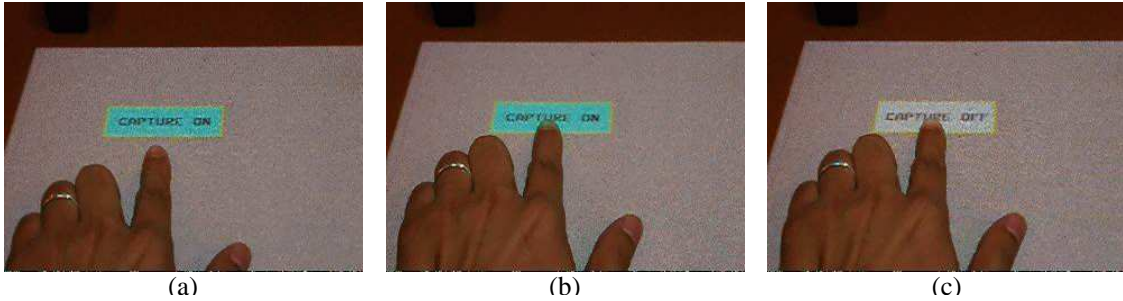


Figure 17: Interaction with a Button. (a) The Button in its ‘normal’ state. When the hand is detected over the Button object (b), its state becomes ‘selected’; changes its appearance (color and text label) (c), and invokes a pre-defined command (not shown, but in this case to turn of the summarization mode.)

Figure 15 illustrates our hand-over-paper detection algorithm. 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 16 for an example of the algorithm running and the user feedback.

### 4.3 Button Objects

Button objects are used to invoke a one-time action or toggle a system mode. Each Button object has one of three states: *normal*, *focused*, and *selected*. 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 becomes “selected” and the command that is associated with the button is invoked. See Figure 17 for an example 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 button’s “hand sensitive” area, a rectangular area slightly larger than the object itself. Figure 18 illustrates our hand-over-button detection algorithm. Also note that a similar “hand over button” system is used in a commercial system from [Tegrity Inc., 2002].

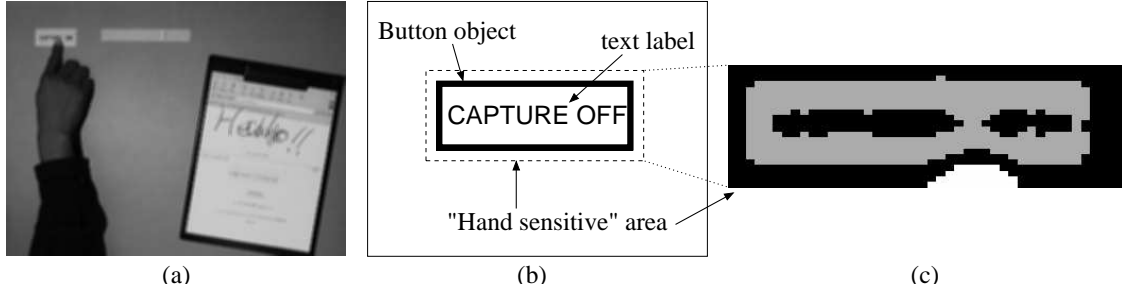


Figure 18: Hand over Button detection. (a) Captured image. One finger of the hand is placed over the Button. (b) An illustration of the ‘hand sensitive’ area of the Button. The hand sensitive area is a rectangular area slightly larger than the object itself. (c) The hand component in the ‘hand sensitive’ 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.

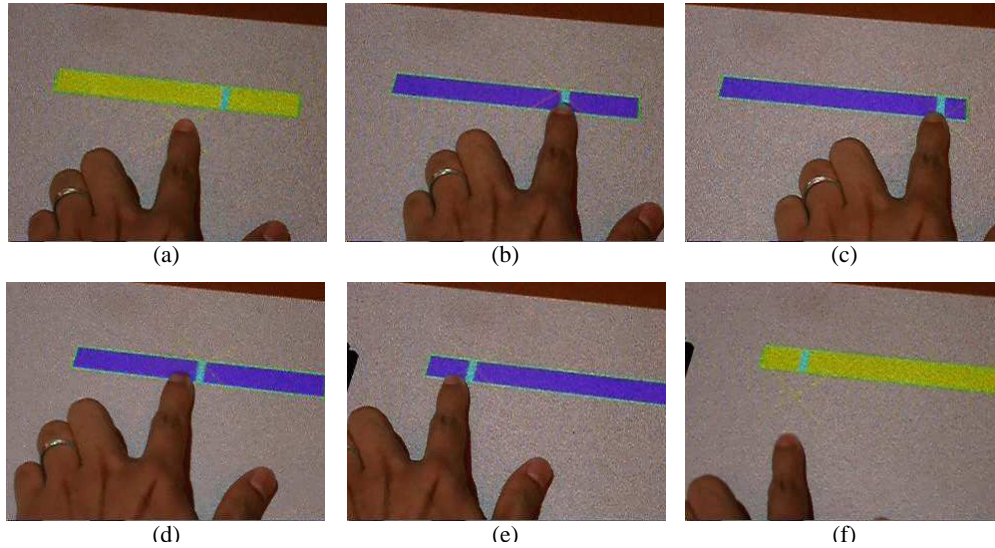


Figure 19: A user interacting with a Slider. (a) The Slider in the ‘normal’ state. (b) When the user’s hand is detected over the Slider, it is activated and changes its appearance (color). The Slider continues to change the value of its parameter depending on how the finger moves (c)(d)(e). The system is continuously notified of the change to the Slider parameter until the hand leaves the Slider (f).

## 4.4 Slider Objects

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 19 for an example.

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 hand position,
3. update the location of the Slider bar according to the parameter value, and
4. notify the system of the parameter value.

The user interface detects the hand component within the Slider’s “hand sensitive” area just like for Button objects, and also estimates the horizontal position of the hand. This value is computed by averaging the x-coordinates of the pixels in the hand component in 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 20(d) for an example. 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 20 illustrates hand over Slider detection and the computation of the hand position.

## 4.5 User Interface Timing Results

Table 6 shows timing results for the user interface on  $320 \times 240$  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 3): i.e. adding user interface processing to the Paper Detection thread doesn’t affect the overall system performance. Paper tracking and UI processing operate at 30Hz on a dual 450MHz Pentium-II machine.

## 5 Complete System Demonstration

Figures 21 and 22 contain a collection of frames from a movie demonstrating the complete Tele-Graffiti system. Figure 21 shows how the session is recorded and Figure 22 shows how the user

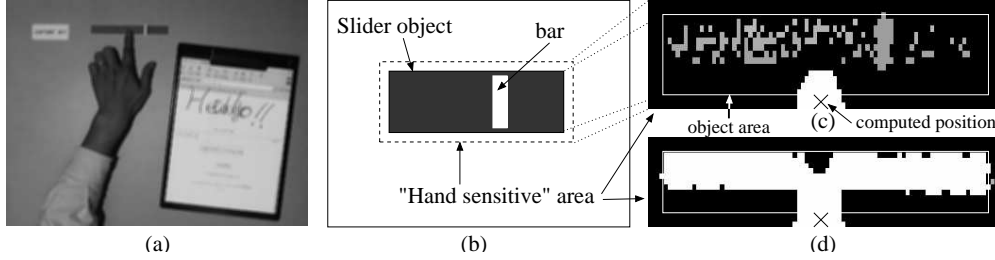


Figure 20: Hand over Slider detection and computation of the hand position. (a) Captured image. One finger of the hand is placed over the Slider object. (b) An illustration of the “hand sensitive” area of the Slider, the same as for the Button. (c) The hand component in the ‘hand sensitive’ area. As 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.

Table 6: 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 3), adding user interface processing doesn’t affect the performance of the paper tracking and 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

browses through the summarized session. The complete video accompanies this paper and is available at the URL <http://www.cs.cmu.edu/~simonb/telegraffiti/movies/uidemo.mpg>.

## 5.1 Session Summarization Using Hand Over Paper Detection

In the demo we record a frame based on hand over paper detection and use the recorded frames to summarize the session analogously to the approaches in [Rekimoto, 1999, Mynatt *et al.*, 1999]. When there are two Tele-Graffiti sites, however, 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 already away from the paper. In order to implement this algorithm, each Tele-Graffiti site maintains the state of both hands at both sites. Each site sends the other the result

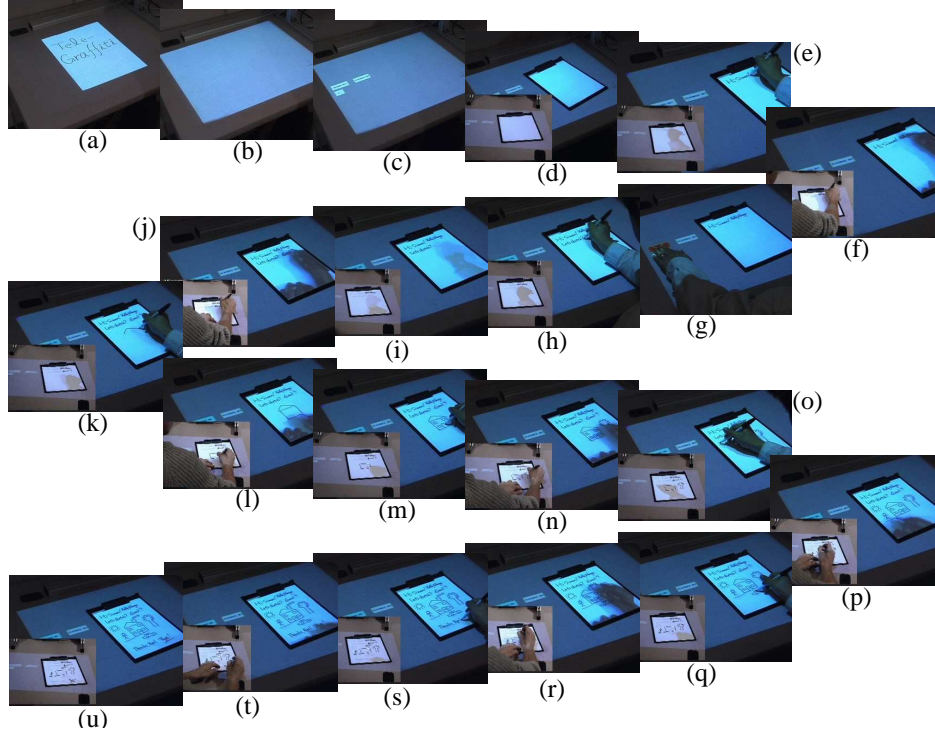


Figure 21: Complete System Demonstration (part 1 of 2, continued in Figure 22). (a) The Tele-Graffiti logo image which is used for geometric calibration (see Section 2.2) is displayed at system startup. (b) Before anything is placed or projected on the desktop, the background image is captured (see Section 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 turn 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 of this interaction which accompanies this paper is available on the Tele-Graffiti website at URL <http://www.cs.cmu.edu/~simonb/telegraffiti/movies/uidemo.mpg>.

of hand-over-paper detection along with the paper images. See [Takao *et al.*, 2002] for the details.

## 5.2 User Controlled Playback 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  $1 + 16 * 39 / 100 = 7.24$ . Figure 22 illustrates how the summarized frames can be browsed using the Slider.

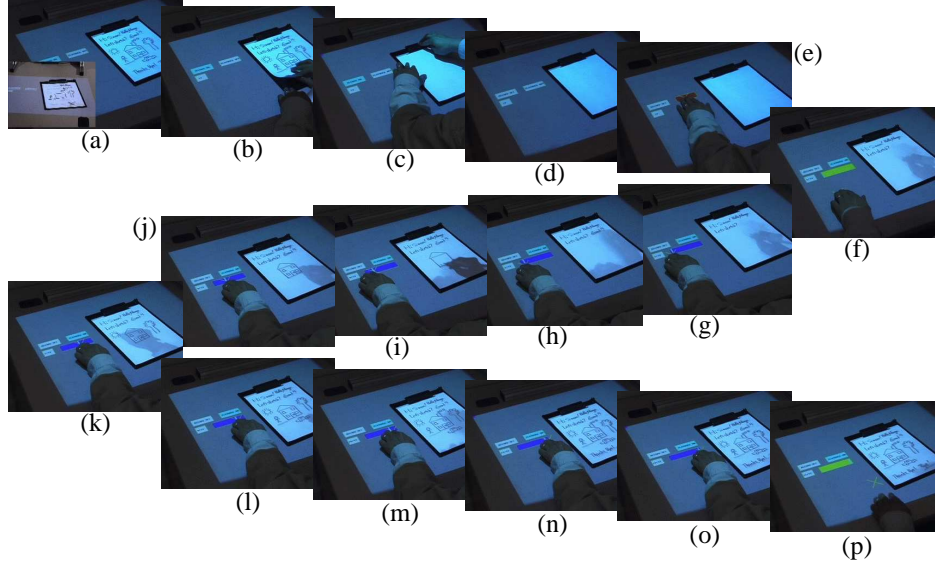


Figure 22: Complete System Demonstration (part 2 of 2, continued from Figure 21). (a) The final sketched image after the session. (b-d) The user replaces the sketched paper with a blank paper in preparation to review the session. (e) The “playback mode” is turned on with a Button. (f) A Slider is projected onto the desktop. (g-o) The user browses through the saved frames with the Slider. As he moves the slider, the projected frame changes accordingly. (p) The final sketched image is projected onto the paper.

## 6 Conclusion

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

**Real-Time Paper Tracking:** 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.

**Real-Time Video Transmission:** We have developed a software architecture for Tele-Graffiti and implemented real-time video transmission over IP networks on standard 450MHz PCs.

**Hand-Based User Interface:** 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 UI objects such as buttons and sliders.

**Automatic Summarization:** We have 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 frames.

The complete Tele-Graffiti system operates at 30Hz on a PC with dual 450MHz Pentium-II processors. With typical network parameters a video rate of around 10 frames per second is achieved.

## Acknowledgments

We would like thank Iain Matthews and Bart Nabbe for helping us design and build the Tele-Graffiti stands. See Figure 2. 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

- [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/>.
- [Chipolla and Pentland, 1998] R. Chipolla and A.P. Pentland. *Computer Vision for Human-Machine Interaction*. Cambridge University Press, 1998.
- [Crowley and Coutaz, 1995] J. Crowley and J. Coutaz. Vision for man machine interaction. In *Proceedings of Engineering Human Computer Interaction*, 1995.
- [Faugeras, 1993] O.D. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.
- [Freeman *et al.*, 1998] W.T. Freeman, D.B. Anderson, P.A. Beardsley, C.N. Dodge, M. Roth, C.D. Weissman, W.S. Yerazunis, H. Kage, K. Kyuma, Y. Miyake, and K. Tanaka. Computer vision for interactive computer graphics. *IEEE Computer Graphics & Applications*, 18(3):42–53, 1998.
- [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.
- [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](http://www.gvu.gatech.edu/ccg/projects/perceptive_cga/perceptive_cga.html).
- [Lien and Huang, 1998] C. Lien and C. Huang. Model-based articulated hand motion tracking for gesture recognition. *Image and Vision Computing*, 16(2):121–134, February 1998.

- [Mynatt *et al.*, 1999] E.D. Mynatt, T. Igarashi, W.K. Edwards, and A. LaMarca. Flatland: New dimensions in office whiteboards. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 1999.
- [Oka *et al.*, 2002] K. Oka, Y. Sato, and H. Koike. Real-time tracking of multiple fingertips and gesture recognition for augmented desk interface. In *Proceedings of the Fifth International Conference on Automatic Face and Gesture Recognition*, 2002.
- [Pavlovic *et al.*, 1997] V. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997.
- [Quek *et al.*, 1995] F. Quek, T. Mysliwiec, and M. Zhao. Finger mouse: A freehand pointing interface. In *Proceedings of the International Workshop on Automatic Face and Gesture Recognition*, 1995.
- [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/>.
- [Rekimoto, 1999] J. Rekimoto. Time-machine computing: A time-centric approach for the information environment. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1999.
- [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.
- [Sukthankar *et al.*, 2001] R. Sukthankar, T.J. Cham, and G. Sukthankar. Dynamic shadow elimination for multi-projector displays. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001. <http://crl.research.compaq.com/vision/interfaces/projector/>.
- [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] N. Takao, J. Shi, and S. Baker. Tele-Graffiti. Technical Report CMU-RI-TR-02-10, Carnegie Mellon Robotics Institute, March 2002.
- [Tegrit Inc., 2002] Tegrit Inc. *Home Page of Tegrit Inc.* <http://www.tegrity.com>, 2002.
- [von Hardenburgh and Bernard, 2001] C. von Hardenburgh and Francois Bernard. Bare-hand human-computer interaction. In *Proceedings of the ACM Workshop on Perceptive User Interfaces*, 2001.
- [Wellner, 1993] P. Wellner. Interacting with paper on the DigitalDesk. *Communications of the ACM*, 36(7):86–96, July 1993.
- [WolfVision, 2002] WolfVision. WolfVision visualizer, 2002.