BrightBoard: A Video-Augmented Environment

Quentin Stafford-Fraser [†*]
[†] Rank Xerox Research Centre (EuroPARC)
Ravenscroft House, 61 Regent Street
Cambridge CB2 1AB, United Kingdom
Tel: +44 1223 341521
E-mail: fraser@europarc.xerox.com

Peter Robinson [*]
[*] University of Cambridge Computer Laboratory
New Museums Site, Pembroke Street
Cambridge CB2 3QG, United Kingdom
Tel: +44 1223 334637
E-mail: pr@cl.cam.ac.uk

ABSTRACT
The goal of 'Computer Augmented Environments' is to bring computational power to everyday objects with which users are already familiar, so that the user interface to this computational power becomes almost invisible. Video is a very important tool in creating Augmented Environments and recent camera-manufacturing techniques make it an economically viable proposition in the general marketplace. BrightBoard is an example system which uses a video camera and audio feedback to enhance the facilities of an ordinary whiteboard, allowing a user to control a computer through simple marks made on the board. We describe its operation in some detail, and discuss how it tackles some of the problems common to these 'Video-Augmented Environments'.

KEYWORDS
Augmented reality, image processing, machine vision, pattern recognition, ubiquitous computing

INTRODUCTION
Video cameras can now be produced, with controlling circuitry, on a single chip. Digital output reduces the need for expensive frame-capture cards, and it is reasonable to assume that simple cameras, being devoid of moving parts, will soon be a cheaper accessory for personal computers than keyboards, microphones, and mice. The possibilities for video-conferencing and other human-human interactions are obvious, but how might the ready availability of video sources enrich the way we interact with the machines themselves?

Augmented Environments
In the office environment the computer has traditionally been thought of as a tool, in the same way that a typewriter or Rolodex is a tool. It is something employed by a user to accomplish a particular task. It is a very flexible tool, and can fulfil the roles of typewriter, Rolodex and calculator simultaneously, but the activity is generally initiated by a 'user', a term which succinctly describes the relationship.

We might hope that the computer of the future would be more like an assistant than like a typewriter, that it would perform mundane tasks *on your behalf* when it sees they are necessary rather than always operating directly *under your control*. The more a secretary knows about your way of life, your preferred modes of work, how cluttered your desk is and when you are in a meeting, the more useful he or she can be to you. The same applies to a computer, but the average PC has no knowledge of the world outside its box and has to be spoon-fed with information through its limited range of input devices.

This is part of the philosophy behind *Computer-Augmented Environments*; the desire to bring the computer 'out of its box' and give it more awareness of the world around, so that it *augments* and *enhances* daily life rather than attempting to replace it. Can we, for example, make the computer understand what we do in our offices rather than putting an impoverished 'office metaphor' on the machine's screen? Can we centre computational power around everyday objects with which users are already so familiar that they don't think of them as a human-computer interface? During work on the DigitalDesk [10,16], for example, we experimented with ways of enabling the computer to recognise an ordinary pencil eraser, and using that as the means of deleting parts of an electronic image projected onto the desk. The motivation was simple: people already know how to use erasers, and will continue to use them for hand-drawn pictures. By augmenting the eraser's capabilities we simply expand its scope of use. We neither detract from its abilities to erase pencil, nor require the user to learn a new tool.

ACHIEVING AUGMENTED ENVIRONMENTS
There is an inherent difficulty in the goals of Computer-Augmented Environments (CAEs). For computers to have much understanding of the world around, they must be equipped with sensors to monitor it. These may be directly connected to the computer, or they may relay information to it via some communication medium. The more sensors, the more complete will be the picture a computer can create of its surroundings and the activities of its users. However, the more sensors we embed in the environment, along with the associated cables, batteries and power supplies, the more investment is needed to bring about our goal and the more intrusive the

technology becomes, so frustrating the aim of an 'invisible' user interface.

The solution is to give the computer a small number of senses which have a broad scope of application and which operate remotely, i.e. without direct contact with the objects being sensed. The obvious candidates are vision and hearing, through the use of video cameras and microphones. Similarly, if the computer is to communicate with humans, then the analogous output systems are audio feedback and perhaps some projection facilities. Not only do cameras and microphones have a broad range of application, they can often be applied to more than one task at the same time. A single camera might monitor a whole room, and detect when the room is occupied, when a meeting is in progress, and when an overhead projector is being used, as well as recording the meeting.

This paper describes the use of video in the creation of a Computer-Augmented Environment called *BrightBoard*, which uses a video camera and audio feedback to augment the facilities of an ordinary whiteboard. Until fairly recently the deployment of such systems in the average workplace would be limited by the cost both of cameras and of the computing power required to process video signals. However, manufacturing developments are making video cameras an economically viable alternative to more conventional sensors, and the typical office workstation is now fast enough for the simple image processing required for many of these 'Video-Augmented Environments' (VAEs).

BRIGHTBOARD: THE WHITEBOARD AS A USER INTERFACE

From prehistoric cave paintings to modern graffiti, mankind has conveyed information by writing on walls. They provide a working surface at a convenient angle for humans to use, with space for expansive self-expression, and they form a natural focus for a presentation or group discussion. The blackboard, the more recent whiteboard, and to some degree the overhead projector, are an extension of this basic theme, with the added facility of being able to erase easily any information which is no longer needed. Whiteboards have become very popular tools in environments ranging from the kindergarten to the company boardroom.

BrightBoard aims to capitalise on this natural means of expression by making use of the whiteboard as a user interface. It is not the first system to explore the whiteboard's potential. Projects such as Tivoli [11] have tried to capitalise on this natural means of expression, and have created note-taking, shared drawing and even remote conferencing systems by emulating and enhancing a whiteboard using a computer. There have been many variations on the whiteboard theme as well. VideoWhiteboard [15] used a translucent drawing screen on which the silhouette of the other party could be seen. Clearboard [8] was a similar system which used the metaphor of a glass whiteboard where the parties in a two-way video conference were on opposite sides of the

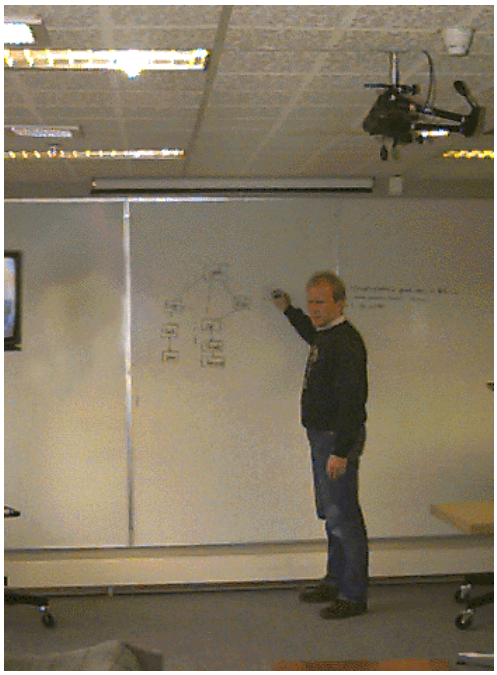

Figure 1: BrightBoard in use

glass, allowing both face-to-face discussion and shared use of a drawing space. Typically, these systems use a large-screen display and electronic pens whose position in the plane of the screen can be sensed and which may have pressure-sensitive tips to control the flow of 'ink', thus giving a more realistic feel to the whiteboard metaphor. The Xerox Liveboard [4] is an example of a complete, self-contained, commercially available unit built on this basis; it has a workstation, Tivoli software, display and pens in one ready-to-install package. The software allows several users (potentially at different locations) to draw on a shared multi-page whiteboard where each page may be larger than the screen and scrollable. The software is controlled by a combination of gestures, buttons and pop-up menus. A more recent commercial offering is *SoftBoard*. This is a special whiteboard which uses pens and erasers similar to conventional which have a reflective sleeve allowing their position to be detected by an infra-red laser scanning device fitted to the board. The movements of the pens and erasers are relayed by the board to a computer, which can then construct an electronic version of the image on the board.

Such systems, while useful, have their failings. Their price is typically quoted in thousands of dollars, they are often delivered by a forklift truck, and are generally installed in board-rooms whose primary users have neither the time nor the inclination to master the software. They also fail to achieve the ease of use of a conventional whiteboard, even for those experienced with them. To quote one user, "The computer never quite gets out of the way".

Examples of whiteboard use observed by the author, which tend not to translate well into the electronic domain, include the following:

- A ‘y’ was written on the board, but looked rather like a ‘g’. The writer used a finger to erase part of the letter before continuing; a process which barely interrupted the flow of the writing.
- During the design of a user interface, screen components such as buttons and scrollbars were drawn on Post-It notes or on pieces of paper affixed to the board with magnets. They could then be added, removed or repositioned easily within the sketches on the board.
- The whiteboard eraser had been temporarily mislaid, and a paper towel was used instead.

A computer-based “whiteboard metaphor” always breaks down somewhere because the computer’s view of the board differs from the user’s.

WHAT IS BRIGHTBOARD?

BrightBoard *is* a whiteboard. Anyone who can use an ordinary whiteboard can use it. In addition, it provides some of the functionality of electronic documents - the images on the board can be saved, printed, faxed and emailed simply by walking up to the board and making particular marks. (See examples in Figure 2, and the Video Figure in the CHI 96 Video Program).

BrightBoard can also operate as a more general input device for any computer-controlled system. An example of its use might be as follows:

> Eric is leading a meeting in a room equipped with a BrightBoard camera. He arrives early to discover that the room is a little chilly, so he writes the temperature he requires on the whiteboard, T21, and the air-conditioning reacts appropriately. When the participants arrive, he makes a mark on the board to start the video-recording of the meeting, V, and then uses the board as normal. During the meeting the participants request a copy of a diagram on the board, so Eric marks that area of the board and prints off six copies P6. As the meeting draws to a close, Eric scribbles a quick note requesting coffee for six, marks out the area as before, and mails it to his secretary M. Finally, he switches off the video-recorder and the air-conditioning by erasing his original marks.

All this control has been achieved without Eric leaving the focal point of the meeting – the whiteboard – and without direct interaction with any machines or controls. It is easy to configure:

- the marks that BrightBoard will recognise, by capturing and labelling samples. (“This mark is an ‘S’.”).
- the relationships between different marks required to constitute a command. (“A *save* command is an ‘S’ inside a box.”).
- the action to be taken when a particular command is specified. (“When you see a *save* command, execute this shell script.”)

Corner marks can be used to delimit an area of the board, and commands can then be made to act only on this area. Inexperienced users may find it useful to have a summary of the available commands posted beside the whiteboard.

BrightBoard uses a video camera pointed at a standard whiteboard (Figure 1), thus eliminating the need for expensive installations and electronic pens. It uses audio or other feedback, and so has no need of a large display. The program, once configured, puts minimal load on the workstation, and requires no manual intervention. The computer can therefore be shut away in a cupboard, or put in a separate room – particularly useful for meeting rooms where the hum of fans and the flicker of screens can be a distraction.

HOW DOES BRIGHTBOARD WORK?

BrightBoard consists of several programs, but the main one has a structure common to many VAE applications. It centres around a loop containing the following steps:

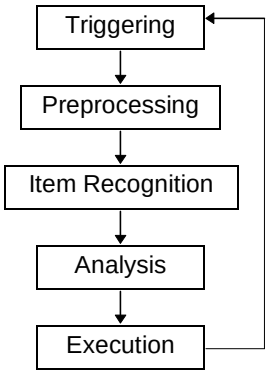

Figure 3: BrightBoard’s main loop

1. Triggering – Wait until a suitable image can be captured, and then do so.
2. Preprocessing of the image. (Thresholding, in the case of BrightBoard)
3. Item Recognition. (Finding & Recognising marks)
4. Analysis of the items found looking for particular situations.

A ‘Print’ command *Selecting an area of the board* *A ‘Fax to Peter’ command*

Figure 2: Some sample BrightBoard commands

5. Execution of some action(s) based on the situations found.

In practice, these stages may not be strictly sequential. The triggering might be accomplished by dedicated hardware inside the camera, for example. In BrightBoard the execution of external commands is done as a separate background process.

We shall look at each of these stages in turn.

Triggering
There are two considerations here:

- Whiteboards suffer badly from obstruction – the interesting things are generally happening when somebody is obscuring the camera's view. We must detect when people are in the image and wait until they have moved out of the way. Humans have a very useful characteristic in that they tend to keep moving, and in many VAEs this can be sufficient to separate them from an inanimate background.
- One of the aims of BrightBoard is that it should be practical to have it running all the time. To start it up whenever a user wished to write on the board would largely defeat the object of the exercise, but, on the other hand, it would not be acceptable for an infrequently used facility to consume too many resources on the user's workstation.

Both problems can be solved by the use of a 'triggering' module. BrightBoard normally operates in a semi-dormant 'standby' mode. Every half-second or so, the triggering module captures a low-resolution image and examines a 40x30 grid of pixels, comparing each with the equivalent pixel in the previous image. It calculates the percentage *P* of these pixels which have changed by more than a certain threshold. This puts very little load on a typical workstation. The threshold for each pixel is based on the statistics of the noise seen at that pixel position in ten calibration frames captured at startup time. This allows the system to compensate for regions of the image which are inherently noisier due to lower light levels, reflections of flickering screens, etc. In addition, these thresholds are adjusted slowly over time, to cope with changes in lighting during the day.

The triggering module can wait either for movement, or for stability. By concatenating these two modes we say, in effect: "Ignore an unchanging whiteboard. Wait until you see movement in front of it, and when this movement has finished, then proceed with your analysis". A full-resolution image can then be captured, to be used in the following stages.

Thresholding
If we are to analyse the writing on the board, we must now distinguish it from the background of the board itself. A simple global threshold will not suffice, for two reasons:

a) The high proportion of white to black pixels means that analysis of the histogram does not generally reveal clear maxima and minima - the 'black peak' tends to get lost amid the noise of the 'white'.[1]
b) the level of illumination on the board often varies dramatically from one side to the other.

There are many sophisticated methods available which attempt to solve the former by examining only pixels near the black/white boundaries, thus giving a better balance of black and white pixels [5], and the latter by dividing the board into tiles, finding local thresholds suitable for the centre of each tile, and then extrapolating suitable values for points between these centres [3].

Unfortunately, these methods are rather too slow for an interactive system, so we use an adaptive thresholding algorithm developed by Wellner for the DigitalDesk [16]. This involves scanning the rows of pixels one at a time, alternating the direction of travel, and maintaining a running average of the pixel values. Any pixel significantly darker than the running average at that point is treated as black, while the others are taken to be white. This simple algorithm works remarkably well in cases where the image is known to have small areas of dark on a light background, such as we find in the typical printed page and on a whiteboard, and it only involves a single examination of each pixel.

Item recognition
There are two distinct operations here; first we find the marks on the image of the board, then we attempt to recognise them.

Finding
Each black 'blob' in the thresholded image is found and analysed. We are not interested in blobs consisting of only a few pixels. With the camera zoom setting chosen to include a reasonable amount of a whiteboard, the symbols we wish to recognise will generally contain between a hundred and a thousand pixels. We need not, therefore, examine every pixel in the image, but can look at every third or fourth pixel in each direction with fair confidence that we will still strike any areas of interest. This has the added benefit of 'missing' much of the noise in the image which appears as blobs of one or two pixels.

Once a black pixel is found, a flood-fill algorithm is used to find all the black pixels directly connected to that one. As the fill proceeds, statistics about the pixels in the blob are gathered which will be used later in the recognition process; for example, the bounding box of the blob, the distribution of the pixels in each axis, the number of pixels which have white above them and the number with white to the right of them. For the sake of speed, an 'upper limit' on the number of pixels is passed to the filling routine. Beyond this limit the blob is unlikely to be anything we wish to recognise, so the flood fill continues, marking the pixels as 'seen', but the statistics are not gathered. When the fill completes, if the number

[1] For simplicity we refer to the marks as being 'black' on a 'white' background. In fact, any coloured pens may be used, though the thresholding is easier with the darker colours.

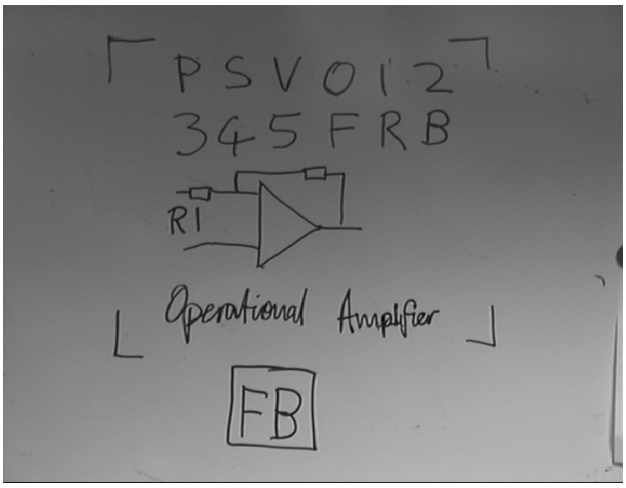

Figure 4: A sample image captured...

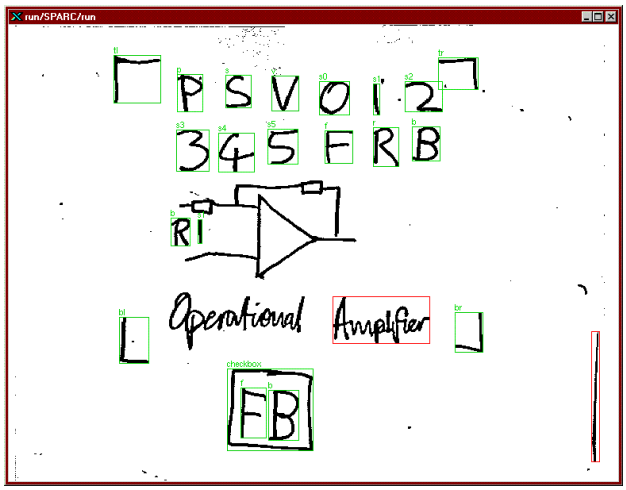

Figure 5: ...and processed by BrightBoard

of pixels is less than this upper limit but more than a specified lower limit, the blob's statistics are added to a list of items for further analysis.

Recognising

In the current system, no further processing of the image is carried out. The problem of off-line handwriting recognition is the subject of a large and active research field, and for a good overview the reader is referred to A.W.Senior [13]. Many things could be done by analysing the topology of each blob, for example, but this would involve a significant amount of pixel processing, which we have so far managed to avoid. In addition, topological analysis can cause problems with such a potentially noisy image. After the thresholding, the majority of white pixels have been ignored and each black one will have been examined maybe twice or three times. Yet, for the limited range of symbols we wish to recognise, the statistics gathered are sufficient. From these we can calculate a *feature vector* - a set of real numbers representing various characteristics of the blob, which can be thought of as coordinates positioning the blob in an n-dimensional space. At the time of writing, 12 dimensions are in use, and the values calculated are chosen to be reasonably independent of the scale of the blob or the thickness of the pen used. For example, one value is the ratio of the number of black pixels with white above them to the number of black pixels with white to the right of them. This gives a rough measure of the ratio of horizontal to vertical lines in the symbol. Another set of characteristics are based on moments about the centroid of the blob. Moments have been found to be very useful in this type of pattern recognition [2,7], and a hardware implementation is possible allowing substantial speed improvements [6].

The recognition is done by comparing the feature vector of each blob found to those of prototype copies of the symbols we wish to match. These are all calculated off-line in advance from a training set of board images. The simplest recogniser might just find the closest prototype in the n-dimensional space to the candidate blob, and identify the blob as being of the same type. This has several limitations, however:

- The scales of the dimensions are not in any way related, and a distance of 0.1 in one dimension may be far more significant than a distance of 1.0 in another. We therefore use the *Mahalanobis distance* instead of the *Euclidean distance*, which simply means that each dimension is scaled by dividing it by the standard deviation of the values found in this dimension amongst the prototypes.
- The second limitation is the absence of a rejection condition. We assume that all blobs can be recognised, which is most unlikely. Using the Mahalanobis distance, however, it makes more sense to reject blobs which are more than a certain distance from even the closest prototype. If this distance is more than $\sqrt{n}D$, where n is the number of dimensions, it means that the candidate blob is on average more than D standard deviations from the prototype in each dimension. By adjusting the value of D the selectiveness of our recogniser can be controlled.
- The third is that, in many of the dimensions, the groups formed by different symbols may overlap to a considerable degree. The capital letters *B* and *R*, for example, are similar in many ways, and it is quite possible that the statistics calculated for a candidate *R* may place it closest to a prototype *B* (see Figure 6). To help avoid this, we use a *k-nearest-neighbours* algorithm which states that X can be classified as being a symbol Y if at least a certain number of its *k* nearest prototypes are symbol Y. For BrightBoard, we classify X as being an *R* if at least 3 of its 5 nearest neighbours are *R*s. The cost of finding the *k* nearest neighbours is approximately proportional to the number of prototypes examined, if *k* is small.

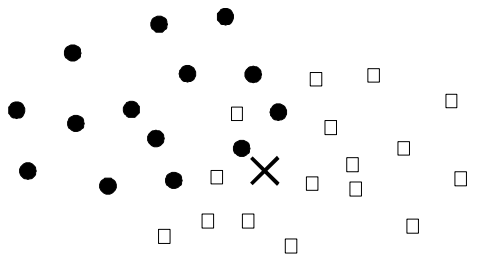

Figure 6: If circles are B prototypes and squares are R prototypes, what is X?

- Lastly, the values used for some dimensions are less reliable than others at distinguishing between symbols. We therefore have a set of dimension weightings which are adjusted by hand on a trial and error basis. All distances in a given dimension are multiplied by the appropriate weighting before being used. A number of methods for selecting weights automatically are described in [2].

Several methods are available to improve the partitioning of the *n*-space. The Karhunen-Loève transform can be used to choose alternative axes which highlight the distribution of the prototypes. Simard et al describe an alternative distance metric which is invariant to small transformations of the input image [14].

Neural Alternatives

This recognition phase is an obvious candidate for neural network-based solutions, and these are being investigated. The use of a neural net could mean improved reliability and constant recognition time, at the expense of less predictability and more time-consuming analysis of the training set. At present the training data consists of about 20 copies of each of the 17 symbols we recognise, and this is really far too small for neural net training. To combat this we are experimenting with the use of 'fake' training data, created by applying to the captured images the sort of variations that might be found in normal handwriting: small amounts of shear and rotation, and variations in scale and aspect ratio. This is less time-consuming than the capture of new training data, and early experiments show an encouraging improvement in both neural- and non-neural-based recognition rates.

The limitations of a very crude recogniser can be overcome to a substantial degree by the choice of command patterns. In practice we find that the current recogniser is capable of distinguishing well between the symbols in its known alphabet, (we measured recognition rates of 93-96%) but it has a tendency to be over-generous, and recognise as valid symbols some things which are not. The chances of these 'false' symbols occurring in such relationships as to constitute a valid command are, however, very small.

Analysing

We now have a system which waits for an opportune moment, captures an image of the board, and finds and recognises the symbols in the image. The next problem is how to describe the commands we wish to have executed in terms of these symbols. We do not wish to hard-code such descriptions into the program, so we need a grammar with which to specify the combination of symbols which constitute, say, a 'print' command.

Fortunately, there are languages available which specialise in the description and analysis of relationships: those designed for Logic Programming, of which Prolog is the most common. If the information from the recogniser is passed to a Prolog engine as a set of facts, we can then write Prolog rules to analyse the contents of the board. For each blob found, BrightBoard assigns a unique number *x* and adds a rule to a Prolog database an assertion of the form:

```
bounds( itemx, w, e, n, s )
```

which specifies that blob *x* has a bounding box delimited by the north-west corner (*w*, *n*) and the south-east corner (*e* , *s*). Simple rules can then be written to determine, for example, whether blob A is inside blob B, or to the right of it, or of a similar size, from these entries. In addition, if the blob has been recognised, a second assertion will be made, of the form:

```
issym( itemx, y )
```

which indicates that item *x* has been recognised as being symbol *y*. A 'Print' command might then be defined as follows:

```
doprint :-
    issym(X, p),
    issym(Y, checkbox),
    inside(X, Y),
    /+ (inside(Z, Y), Z \= X)
```

This can be roughly translated as "there is a print command if we can find blobs X and Y such that X is a 'P' and Y is a 'checkbox' and X is inside Y, and nothing else is inside Y [2]".

Both current and previous states can be passed to Prolog, so that the rules can also specify that printing should only occur if either X or Y or both were not present in the previous analysis. This prevents a command from being accidentally repeated.

On a SPARCstation 2, BrightBoard took 4.5 seconds to capture, threshold, display, analyse and recognise the 'Fax to Bob' command in the 740 x 570 image shown in Figure 4, from the time at which movement was no longer detected. There is much room for optimisation; speed was not a primary issue during BrightBoard's initial development.

Executing

The final stage is to take action based on the analysis. A 'command file' relates Prolog predicates such as 'doprint' to UNIX commands that will actually do the printing. Each line in this file has the form:

---

[2] The final part is more accurately transcribed as: 'The goal "Z is inside Y and Z is not equal to X" cannot be satisfied'

```
<predicate> <filetype> <command>
```

where <predicate> is, for example, 'doprint', <command> is any valid UNIX shell command, and <filetype> is either 'none' or the name of an image format[3]. If it is not 'none' then a temporary file of the specified format is created and its filename can be passed to the UNIX command by using '%s' in the command file. A print command might then be:

```
doprint    pgm     pgmtops %s | lpr
```

though more complicated actions would generally be implemented by a specially written script or another program. The commands currently employed also provide audio feedback to the user through the use of pre-recorded or synthesised speech. The user is informed not only when a print command has been seen, but also when the printing has been completed.

One predicate is given special treatment in the current version of BrightBoard. It is named 'inc_area' and checks for the presence of symbols which mark the bounds of the area to be included in the image file. This allows small areas of the board to be printed, saved, sent as messages etc.

The next version of BrightBoard (currently under development) uses an extended protocol for the interaction between Prolog and the UNIX commands. The number of parameters of the predicate may be specified in the command file, and, if greater than zero, the values of the variables returned by the evaluation are passed to the UNIX command as environment variables. The UNIX command is executed once for each match found in a given image, with a special variable set on the last match. This allows the function of 'inc_area' and similar predicates to be implemented by external programs, giving rise to much greater flexibility. As an example, a print command can consist of a *P* followed by a digit, where the digit represents the number of copies to be printed. The `doprint` predicate can then have a parameter in which it returns the digit found, and this information is passed to the executed command.

Until now, all use of BrightBoard has been by the author and two colleagues for a substantial number of demonstrations, but under fairly controlled lab conditions. The system has almost reached the stage where user testing is possible in a real office environment, and this is the obvious next step in its development.

FUTURE POSSIBILITIES

There are a few aspects of BrightBoard which are worth highlighting, especially in the context of possible future developments.

The first is that there is minimal configuration required to set it up. All that is needed is a camera with a fairly unobstructed and 'straight-on' view of the board, zoomed to a reasonable scale. It would be straightforward, therefore, to make a portable version of BrightBoard. A briefcase containing a laptop computer and a camera with a small tripod could be carried into any meeting room, the camera pointed at a board and the program started, and the whiteboard in the meeting room is immediately endowed with faxing, printing, and recording capabilities.

Secondly, the system is not limited to whiteboards – any white surface will suffice. Thus noticeboards, refrigerator doors, flipcharts, notebooks and papers on a desk can all be used as a simple user interface. The current version of BrightBoard has been switched from monitoring a whiteboard to providing a 'desktop photocopying' device without even stopping and restarting the program. A camera clipped to a bookshelf above the desk and plugged into a PC (which will often be on the desk anyway) enables any document on the desk to be copied, saved, faxed without the user moving from the desk or touching a machine. If the user does not wish to write on the documents themselves, then Post-it notes or cardboard cut-outs with the appropriate symbols drawn on them can be used. Parts of the paper documents can be selected using the area delimiting symbols, and pasted into electronic documents. Resolution is a slight problem here, as a typical frame-grabber capturing half a page will only provide about 100 dots-per-inch; the same resolution as a poor-quality fax. It does, however, capture a grey-scale image, and the anti-aliasing effects make the resolution appear much higher than would be the case with a purely black & white image. In situations where a binary image is definitely needed, the greyscale information can be used to enhance the resolution artificially. We have found the appearance of thresholded images to be greatly improved by inserting extra pixels between the original grey pixels and giving them values based on linear interpolation between the neighbouring originals. A double-resolution image is formed which is then passed to the thresholding module. Since this increases the time required for thresholding by a factor of four, the process is only used in the output of images, and not in BrightBoard's internal processing.

A richer interaction model would be possible by monitoring the position and gestures of a user's hands. The system, before acting on a command, for example, could request confirmation from the user which might be given with a 'thumbs-up' gesture. Conventional hand-tracking systems have generally required the user to wear special gloves and position-sensing devices [1], but systems have been built using video alone. Jakub Segen describes a system which allows the user to control a mouse-style pointer on the screen, or to 'fly' through a Virtual Reality-like environment, using hand movements and gestures watched by a camera [12]. Myron Krueger's *VideoPlace* combines projected graphics with the silhouette of the user to create an artistic medium which responds to full-body gestures [9]. Such a system would be easier to include in a portable BrightBoard, as described above, than one involving extra hardware.

---

[3] The formats 'pgm' (portable greymap) and 'pbm' (portable bitmap) are currently supported.

An interesting challenge would be the creation of a friendlier user interface to the Prolog rules. One of the aims of BrightBoard is that it should be accepted in a normal office environment, but the inhabitants of such an environment will not generally be Prolog programmers. A programming language allows us great flexibilty, however, which can be difficult to duplicate in other ways. Consider the following specification:

> 'A *P* in a box, possibly followed by another symbol representing a digit which is also inside the box, constitutes a print command, where the number of copies is given by the digit, or is one if no digit exists. There must be no other symbol inside the box.'

It is difficult to imagine an easy way of representing this graphically. Indeed, even the concept represented by the words 'followed by' must be explicitly defined. A textual front-end to the Prolog could possibly be created which would more closely resemble natural language, or a programming language with which users were more likely to be familiar. This is only an issue if the users are expected to customise the set of commands provided by BrightBoard.

CONCLUSIONS

The dramatic reduction in cost of video cameras and of the computing power needed to process their output opens up a wide variety of opportunities for novel human-computer interaction methods, especially by augmenting the capabilities of everyday objects and environments. This can be less intrusive than more conventional methods of interaction because of the large amount of data that can be gathered from a camera with only a small amount of wiring, and because the camera is a remote sensor and so does not in any way interfere with the user's normal use of the augmented objects.

BrightBoard is an example of this genre which illustrates some of the problems that will be issues to many video-augmented environments and shows what can be accomplished by combining relatively unsophisticated image processing and pattern recognition techniques with logic-based analysis of their results.

ACKNOWLEDGMENTS

This work has been sponsored by Rank Xerox Research Centre, Cambridge, England (better known as 'EuroPARC'). We are grateful for their support and to Dr Pierre Wellner, Mik Lamming, Mike Flynn and Dr Richard Bentley for advice and assistance.

REFERENCES

1. Baudel, T. and Beaudouin-Lafon, M. "Charade: Remote Control of Objects using Free-Hand Gestures", *Comm. ACM*, Vol. 36 Number 7, July 1993, pp 28-37.
2. Cash, G.L. and Hatamian, M., "Optical Character Recognition by the Method of Moments", *Computer Vision, Graphics, and Image Processing*, Vol. 39, pp. 291-310 (1987)
3. Castleman, K., *Digital Image Processing*, Prentice-Hall Signal Processing Series, 1979. The tile-based thresholding algorithm was developed originally by R.J.Hall.
4. Elrod, S., Bruce,R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pedersen, E., Pier, K., Tang, J., and Welch, B., "Liveboard: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration", *Proceedings of CHI*, ACM, 1992, pp. 599-602.
5. Gonzalez, R.C., and Woods, R.E., *Digital Image Processing*, Addison-Wesley 1992
6. Hatamian, M., "A Real-Time Two-Dimensional Moment Generating Algorithm and Its Single Chip Implementation", *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol ASSP-34, No. 3, June 1986, pp. 546-553
7. Hu, Ming-Kuei, "Visual Pattern Recognition by Moment Invariants", *IRE Trans. on Information Theory*, Vol. IT-8, 1962, pp. 179-187.
8. Ishii, H., Kobayashi, M., and Grudin, J., "Integration of Inter-Personal Space and Shared Workspace: ClearBoard Design and Experiments", *Proceedings of CHI*, ACM, 1992, pp. 33-42.
9. Krueger, M.W., *Artificial Reality II*, Addison Wesley, 1991
10. Newman, W. and Wellner, P., "A Desk Supporting Computer-based Interaction with Paper Documents", *Proceedings of CHI*, ACM, May 1992, pp. 587-592.
11. Pedersen, E., McCall, K., Moran, T.P. and Halasz, F.G., "Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings", *Proceedings of INTERCHI*, ACM, 1993, pp. 391-398.
12. Segen, J., "Controlling Computers with Gloveless Gestures", *Proceedings of Virtual Reality Systems '93 conference*, NYC, March 15, 1993.
13. Senior, A.W, "Off-line Handwriting Recognition: A Review and Experiments". University of Cambridge Engineering Department technical report CUED/F-INFENG/TR105, December 1992. Available by FTP.
14. Simard, P.Y, Le Cun, Y., Denker, J.S, "Memory-Based Character Recognition Using a Transformation Invariant Metric" in *Proc. 12th IAPR International Conference on Pattern Recognition*, Jerusalem, 1994, Vol. II pp. 262-267.
15. Tang, J.C., and Minneman, S.L., "VideoWhiteboard: Video Shadows to Support Remote Collaboration", *Proceedings of CHI*, ACM, 1991, pp. 315-322.
16. Wellner, P., *Interacting with Paper on the DigitalDesk*. University of Cambridge Computer Laboratory Ph.D. Thesis, October 1993. Computer Lab Technical Report 330