The Office of the Past: Document Discovery and Tracking from Video

Jiwon Kim Steven M. Seitz
Department of Computer Science and Engineering
University of Washington
{*jwkim|seitz*}*@cs.washington.edu*

Maneesh Agrawala
Microsoft Research
maneesh@microsoft.com

Abstract

This paper presents an approach for reconstructing the physical state of documents and other objects on a desk over time using an overhead video camera. The history of the desktop is subsequently analyzed to enable a range of interesting queries. A desktop object is discovered when it first moves, and is tracked through the image sequence using change detection and motion segmentation techniques. The space-time structure of the scene is represented by a sequence of graphs where an "event", i.e., a change in the state of the scene, is modeled as a transition between subsequent graphs. We present two prototype applications that are based on these desktop reconstruction techniques. The first one allows the user to browse the structure and history of the desktop, and query documents of interest. The second application is a virtual desktop interface where the user can virtually manipulate the document stacks with mouse interaction.

1. Introduction

Despite the increasing prevalence of electronic documents, the usage of paper still prevails in the office environment due to the unique advantages that physical documents provide [1, 2]. Consequently, it is common for office workers to have large stacks of documents cluttered on their desks, which makes it hard to find documents when they are needed.

The Office of the Future project [3] addressed this problem in part by leveraging display technologies to project onto walls or surfaces in the room, thereby minimizing the need for paper. However, their approach requires a fundamental change to the the physical facilities and the working practice of the users. In this paper, we address the same problem but with the additional goal of designing an unobtrusive system, i.e., a system that does not get in the way of the user. To achieve this goal we simply augment the working environment with a single video camera. The camera is set up over the desk (see Figure 1), and records the changes in the stacks of documents over time. Our software system then analyzes the video stream to discover and track documents as they move around the desk, and automatically indexes them for the user. Our system does not make any assumptions about the initial state of the environment. Each document is discovered after it first moves, so the desk is allowed to be initially cluttered, as is common in real situations.

Figure 1: Experimental setup: A camera recording the desktop.

This analysis gives rise to a number of useful applications for managing paper documents. For example, the user can ask the system to find his tax form that is buried somewhere in the stacks of documents on the desk. Alternatively, the user can interactively browse the content of the document stacks on a remote PC, providing an interface like a virtual desktop. The user can annotate discovered documents, e.g., adding a link to an electronic version or a related web site, and this information remains attached to the document as the desktop changes over time.

To support these queries, we must discover and track the positions of the documents on the desk. In our system, each document is discovered when it first moves. Then it is tracked through the series of input images by detecting temporal changes and estimating the motion. A top-to-bottom

hierarchy of the stack is maintained by reasoning about the occlusion order. A sequence of graphs is used to represent the evolution of the stacks over time. A change in the state of the desktop, or an *event*, corresponds to the transition between subsequent graphs. At each event, the most likely event is selected based on the image analysis results, and is used to update the graph.

The main novelty and contribution of this work is the formulation and approach for reconstructing the state of the desktop over time, as a sequence of graphs. Towards this end, we adopt a number of existing low-level algorithms, e.g., change detection, segmentation and matching as building blocks. We feel that the primary contributions of this paper lie not in these individual steps, but rather in the overall analysis engine and its application to a novel real-world problem domain.

The rest of the paper is organized as follows. First, we discuss related work in the following section. Section 3 then describes in detail how the system discovers and tracks documents. After demonstrating some results in section 4, the paper concludes with a discussion and future work in section 5.

2. Related work

Researchers have explored various ways to enhance the office environment with cameras and projectors [4, 5, 3, 6, 7, 8, 9, 10, 11]. DigitalDesk [4] augments the physical desktop with a camera and a projector, allowing the user to interact with the projected image on the desk. However, the emphasis was more on the interaction than tracking physical objects. Tele-graffiti [5] is a sketch-based remote communication system also utilizing a camera and a projector. Although their system supports tracking a single paper mounted on a clipboard, it cannot track papers in a stack. The Office of the Future project [3] envisioned an entirely new office environment, using a number of cameras and projectors to capture and display information onto any surface in the room. Instead, our goal is to be as unobtrusive as possible, leaving the office as it is, except for the addition of a single camera. ObjectSpaces [6] is a general framework for recognizing human activities based on the human user's interaction with physical objects that must be pre-registered by hand. Our system is specialized for the discovery and tracking of desktop objects, and is able to automatically discover them without manual registration. More recently, Fujii et al. [7] explored an idea for tracking stacked objects using stereo vision. However, they use the physical height of the stack to detect changes in the stack, and therefore their technique is not applicable to the general case of paper documents.

The Self-Organizing Desk [8] attempts to track papers on the desk with a camera, and is most similar to our work. However, their system imposes a few key restrictions on the objects and the scene, e.g., all desktop objects must be standard size papers, and they are only allowed to translate. We overcome these limitations and present more convincing experimental results. Some advantages of their work include text-based document clustering using character recognition techniques and allowing underlying documents to slightly shift during movement. We hope to add these capabilities to our system in the future.

Recently, tracking and tagged ID technologies such as barcodes, RF tags and IR tags, are gaining renewed interest from both industries and academia [12, 13]. These technologies require a specialized reader, and are not suitable for accurate detection of object location in a stack. They also require attaching a physical tag to the object, which must be done beforehand and takes up physical space on the object.

In the computer vision and AI communities, a large body of work exists for object tracking and recognition. In particular, layer extraction ([14] and subsequent papers) is an area relevant to our work, as the document stack is by nature a layered structure. As we focus on documents, we are able to use specialized appearance-based tracking techniques. Also, the layered structure not only represents multiple objects with different motions, but also the complex spatial hierarchy of the documents on the desk.

Most closely related to the approach adopted in this paper is Object Discovery [15]. They propose a method to discover objects over time as they enter and leave the scene, by analyzing the temporal history of each pixel of the image sequence. They attempt to explain the temporal evolution of a scene with relatively infrequent object motions, and provided the major inspiration for this paper. However, our work differs from theirs in a few important aspects. First, in Object Discovery, the scene must satisfy the clean world assumption, i.e., each object must both enter and leave the scene. In contrast, we allow the objects to freely enter and leave the scene without such constraint. We also combine temporal and spatial information instead of doing a pure temporal analysis, and are able to recognize a group of pixels as an object, as well as track its location over time. And while they focus on theory with limited experimental results, our objective is to build a practical system that can reconstruct the state of the desk, and we present two applications built on top of this system.

Finally, our work differs from most research in document analysis and recognition, where the focus is primarily on algorithms and applications for analyzing a given document (e.g., OCR), rather than tracking and discovery of documents. In the future, we may want to integrate our system with such techniques.

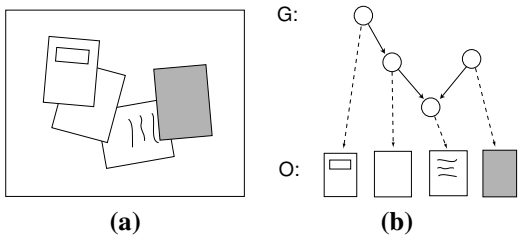

Figure 2: (a) The actual desktop. (b) Representation of the desktop as a scene graph G and an object database O. O contains the position and orientation of each object, and a bitmap image.

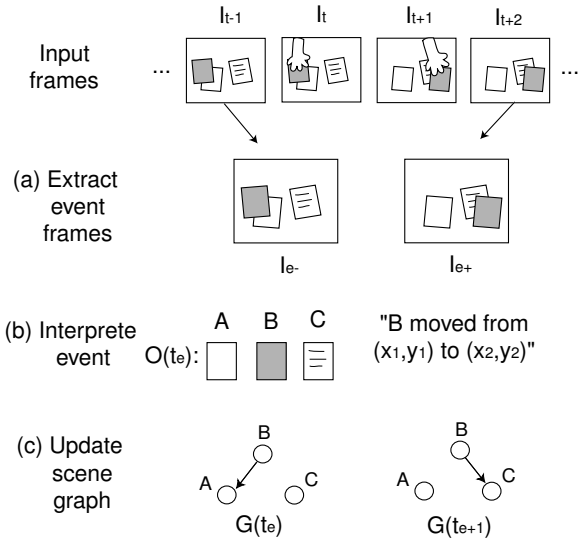

Figure 3: An overview of the document discovery and tracking algorithm. (a) For each event, we extract a pair of images corresponding to the state of the desk before and after the event. (b) Then, these images are analyzed to determine what type of event occurred to which object. (c) Finally, the scene graph is updated accordingly.

3. Document discovery and tracking

In this section, we present a detailed description of how the system discovers and tracks documents from the input video. We first give a problem definition, then explain in detail the actual algorithm used to solve the problem.

3.1. Problem definition

The goal can be briefly stated as follows: given an input video of a desktop where various objects enter, leave and change position on a regular basis, reconstruct the configuration of objects on the desk at each instant in time. Note that the system is not restricted to tracking only documents; documents are modeled as a special case of an object. The input video is recorded from a video camera mounted high above the desk in a straight looking down pose, as illustrated in figure 1. The camera may be mounted on a tripod, or attached to the ceiling.

A more detailed problem definition is as follows. An "event" is defined as a change in the state of the desk, indexed by e, and let t_e denote the time immediately before the event e occurs. Each object o is represented as a bitmap image with position and orientation at each time t. The state of the desktop at each time t is described by a directed acyclic graph (DAG) $G(t)$ that we call the *scene graph*, representing the occlusion order between objects[1], and a set of objects $O(t)$ that have been discovered up to t that we call the *object database*. An object is discovered the first time it moves. Each node n_o of $G(t)$ represents an object o, and an edge $n_o \rightarrow n_p$ exists if o occludes p and there is no other object q that occludes p and is occluded by o. An event is described as a transition from $G(t_e)$ to $G(t_{e+1})$, combined with an update of $O(t_{e+1})$ with $O(t_e) \cup \{o\}$ if a new object o is discovered.

Given this representation, the objective of the system can be restated as (1) detecting events by analyzing the input video, and (2) reconstructing the sequence of DAG's explaining the events. Then the system can answer queries such as "where is my tax form?" by returning its current location stored in the current scene graph and the object database. Figure 2 provides a pictorial description of the representation. We chose DAG's as our representation since they can encode complex overlaps between document stacks such as those shown in the figure. DAG's cannot represent visibility cycles, but we assume that such cases are rarely encountered in practice.

In this paper, the following set of simplifying assumptions are made. It is assumed that at each event exactly one object moves, and only objects that are on top of $G(t_e)$ can move, i.e., objects that are not occluded by any other object. The type of the motion is one of three possible cases: move, entry, or exit. Note that an object is allowed to enter the scene without having to leave, and vice versa, as opposed to the clean world assumption in [15]. It is further assumed that when an object changes position, its motion is a combination of 2D translation and rotation, i.e. there is no change in scale or 3D pose. As the camera is positioned far from the desk, it is reasonable to assume that change in the height of stack causes negligible scale change. In addition, its straight looking down pose ensures that there is minimal change in object shapes. Finally, the objects are assumed to have distinct enough appearances for the camera to tell them apart by comparing color values. However, we make no assumptions about the object shape, e.g., the object does

[1]Note that our use of graphs for the scene representation differs from the scene graphs in computer graphics community that are commonly used to represent the 3D transformation hierarchy of objects in the scene rather than object occlusion as in our system.

not have to be rectangular.

3.2. Discovery and Tracking Algorithm

Given the image frames from the input video, events are first detected by extracting frames representing the state of the desktop before and after the event. For each event, the before and after image pair are analyzed to determine the type (move, entry and exit) and parameters of the event (which object moved and how it moved). Finally, the scene graph is updated according to the event. An overview of the algorithm is illustrated in figure 3. Also note that a number of parameters are used throughout the algorithm whose values were largely determined by experiment.

3.2.1 Event frame extraction

To detect events, we extract a pair of input frames that correspond to the state of the desktop before and after the event by a simple frame differencing technique. More precisely, we take the image frames $\{I_1, I_2, \cdots, I_{n_t}\}$ as input, and produce $\{I_{e-}, I_{e+} | e = 1, \cdots, n_e\}$ as output, where n_t is the number of input frames and n_e is the number of events. We compute the frame difference between each input frame I_t and the following image I_{t+1}. If the difference is above a given threshold, we assume that an event is occurring at time t. For each event, I_{e-} is the input frame immediate before the event starts (i.e., I_{t_e}), and I_{e+} is the input frame immediately after the event finishes.

3.2.2 Event interpretation

Once I_{e-} and I_{e+} are extracted for each event, they are analyzed to interpret the event, to determine the type and parameters of the event as well as the object that moved.

This task is in turn divided into a few substeps. First, we extract the *event region*, the region of the image where a significant change occurred across the event. Then, the event is classified by hypothesizing and testing each type of event in sequence, and the foreground region is segmented accordingly. Finally, we determine the type of event that actually took place, as well as the object that moved.

A. Event region extraction

The event region R_e is identified by frame differencing. I_{e-} and I_{e+} are first blurred to reduce noise. Then, the RGB difference between $I_{e-}(p)$ and $I_{e+}(p)$ is thresholded to produce a binary per-pixel label $event_e(p)$ whose value is 1 if the pixel is within the event region, i.e., $p \in R_e$, and 0 otherwise. R_e is finally post-processed to remove holes and small speckles. An example result of event region detection is shown in figure 4 (c).

B. Hypothesizing events

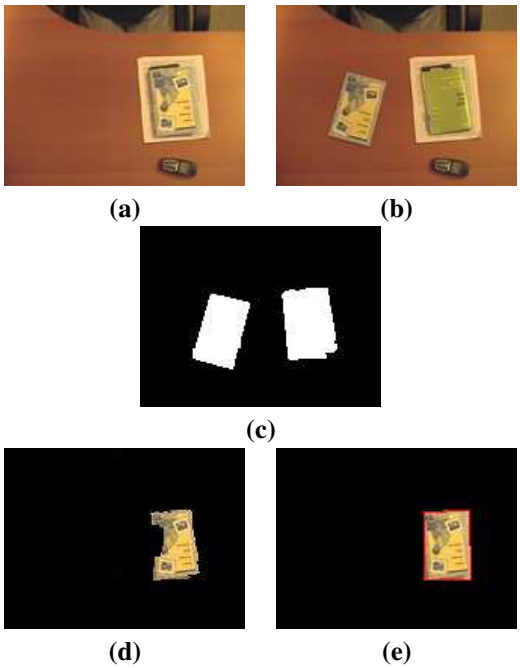

Figure 4: An example event. (a) Image before event. (b) Image after event. (c) Corresponding event region. (d) Foreground region, segmented using graph cuts. (e) Oriented bounding box.

Once the event region is detected, each type of event is hypothesized in order, to determine the type and parameter of the event. By default, we first hypothesize the event as a move event. If the computed score is too small, then the event is considered an entry/exit event, and the algorithm proceeds to determine whether it is an entry or an exit event.

B.1. Move event

In case of a move event, a search is performed to find the optimal transformation $T = (u, v, \theta)$, translation (u, v) followed by rotation θ about the origin, that best explains the foreground motion, by computing a pixel-based matching error between I_{e-} and I_{e+}. The matching error $E(T)$

Figure 5: Example of an incorrect transformation with low matching error, due to a uniform background (computed foreground is outlined in both images). Note that in this solution, most pixels in the event region (figure 4(c)) are not transformed onto the event region.

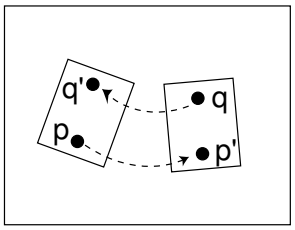

Figure 6: Two points p and q in the event region shown in figure 4 (c). Each pixel in the event region has two possible explanations: a transformation $T(p) = p'$ of a point p in the before image, or an inverse transformation $T^{-1}(q) = q'$ of a point q in the after image.

is defined as follows:

$$E(T) = \frac{1}{|R_e|} \sum_{p \in R_e} e(p, T)$$

$e(p, T)$ is the per-pixel matching error for pixel p under the transformation T, as defined later. We define $overlap(T)$ as the ratio of pixels in the event region that remain within the event region after being transformed by T. If $overlap(T)$ is below a given threshold, we consider T a bad solution and assign a large constant value to penalize such solutions.

$$overlap(T) = \frac{\sum_{p \in R_e} event_e(p) * event_e(Tp)}{|R_e|}$$

In other words, $E(T)$ is the average matching error for a pixel in the event region, under the condition that T transforms most of the pixels within the event region. This condition is necessary to prevent false solutions with low matching error, such as the example shown in figure 5.

Each pixel in the event region belongs to one of two categories: (1) a pixel on the object before the event, and (2) a pixel on the object after the event. Therefore, the per-pixel matching error $e(p, T)$ is defined as the minimum of two cases, (1) difference between $I_{e-}(p)$ and $I_{e+}(Tp)$, and (2) difference between $I_{e-}(T^{-1}p)$ and $I_{e+}(p)$. An illustration is given in figure 6.

$$\begin{aligned} e(p, T) &= min(e_T, e_{T^{-1}}) \\ e_T &= D(I_{e-}(p), I_{e+}(Tp)) \\ e_{T^{-1}} &= D(I_{e-}(T^{-1}p), I_{e+}(p)) \end{aligned}$$

$D(I(p), J(q))$ is a difference operator between two pixels $I(p)$ and $J(q)$ that is defined as follows. Each pixel

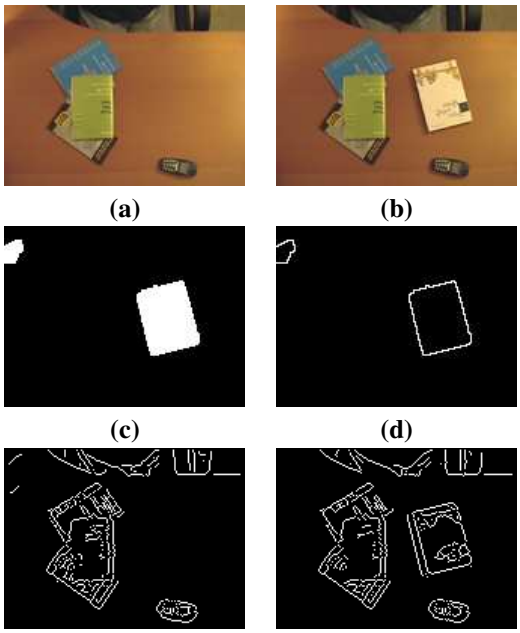

Figure 7: An example of an entry event. (a) Image before event. (b) Image after event. (c) Event region. (d) Border pixels. (e)(f) Edge detection results of (a) and (b). Border pixels in (d) align better with edge pixels in (f).

is defined as a vector (H, S, V, G), where H, S, V corresponds to the color, and $G = 1$ if gradient $>$ threshold, 0 otherwise. We found that HSV is more reliable than RGB under changing illumination. We obtain D by normalizing the difference in color and edge labels with their respective variance and summing them up. If p or q is outside the image frame (e.g., when q is a transformation of p by T that lands outside the image frame), D is assigned a large constant value to penalize such cases.

$$D(I(p), J(q)) = \frac{||I^{HSV}(p) - J^{HSV}(q)||}{\sqrt{Var^{HSV}}} + \frac{|I^G(p) - J^G(q)|}{\sqrt{Var^G}}$$

Instead of doing a brute-force search over all possible transformations $T = (u, v, \theta)$, a hierarchical scheme is adopted where a coarse level search computes matching error at coarse grids of the search space, and a fine level search traverses the "candidate" grids, a small percentage of coarse grids with low matching error. We use an octree to recursively subdivide and search each candidate grid. The fine level search result with the lowest matching error is chosen as the final transformation.

B.2. Entry/exit event

If the matching error as a move event is above a given

threshold, the event is considered an entry/exit event.

If the entry/exit occurred on top of a known stack structure, we can determine the event type by first comparing the event region of I_{e+} with the image of the stack under the document on top. If they match, it means that the top document exited. Otherwise, we decide that it is an entry event.

If the event occurred inside an "empty" area of the desk, i.e., where the stack structure is not discovered yet, we adopt an approach suggested in [15], where the strong edges of I_{e-} and I_{e+} are compared with the object boundary. If it is an entry, the edges of I_{e+} should contain the object boundary, but I_{e-} should not, and vice versa for an exit event. Figure 7 shows an example. We measure the alignment between the edges and object boundary by computing the average distance from a boundary pixel to its closest edge pixel. We denote this alignment measure for I_{e-} and I_{e+} by B_{e-} and B_{e+}, respectively. Object boundary is obtained from the boundary of the event region. The boundary is computed by eroding the event region and taking the difference from the original event region. Canny edge detection is used to preserve strong edges and suppress non-boundary edges.

C. Foreground segmentation

Once the event has been classified, the foreground region is segmented according to the event type.

Given the optimal transformation T for a move event, the foreground is segmented using the graph cuts technique described in [16]. The data term for labeling a pixel (x, y) as foreground is given by $D(I_{e-}(p), I_{e+}(Tp))$. In case of a background pixel, it either remains the same across an event or is occluded by the foreground object. Therefore, the data term is defined as the minimum of two cases, $D(I_{e-}(p), I_{e+}(p))$ and $D(I_{e-}(T^{-1}p), I_{e+}(p))$. The neighborhood term is a small constant across the event region boundary, and a larger constant elsewhere in the image. If there is more than one segment, the largest segment is taken, to satisfy the assumption that only one object moves at each event. An example segmentation result is shown in figure 4 (d).

For an entry or exit event, the foreground is simply given by the event region if there is only one segment. Otherwise, the largest segment is taken.

Lastly, an oriented bounding box is fit to the foreground region by computing 2 major axes of orientation and lengths along these axes, following the technique described in [17]. See figure 4 (e) for an example result. Having an oriented bounding box enables us to match the foreground region against the objects in the database in a rotation-invariant manner.

D. Identifying event type and object

The most likely event type is chosen and the object is identified as follows:

1. Identify event type.
   - If $E(T) < threshold$, then move;
   - Otherwise,
     - If the current scene graph is empty in the event region,
       - If $B_{e-} < B_{e+}$, exit;
       - Otherwise, entry
     - Otherwise,
       - If the event region in I_{e+} matches the object under the top object in the event region, exit;
       - Otherwise, entry
2. Identify object.
   - If move or exit, match foreground against all objects on top of $G(t_e)$;
   - If entry, match against all objects in $O(t_e)$ that are not in $G(t_e)$.
   - If none match, update $O(t_e)$ by adding the new object.
   - Update the object's bitmap image, position and orientation with its current appearance and location after e.

3.2.3 Updating scene graphs

Once the event is interpreted, the final step is to construct the new scene graph $G(t)$, and if a new object has been discovered, update the previous scene graphs $G(1), \cdots, G(t-1)$ accordingly.

$G(t_{e+1})$ is constructed by making a copy of $G(t_e)$ and updating it according to the event e. The move event of an object o requires removing edges $n_o \rightarrow n_p$ for all objects p that have been disoccluded by o, and adding edges $n_o \rightarrow n_q$ for all objects q that are directly under o. For the entry of an object o, new edges $n_o \rightarrow n_p$ are added for all objects p that are directly under o. A similar procedure is applied in the case of an exit. $G(t_e+1), \cdots, G(t_{e+1}-1)$ are constructed by copying $G(t_{e+1})$.

If o is a newly discovered object, all previous scene graphs are also updated. A node representing o, and edges $n_p \rightarrow n_o$ for all objects p that are directly on top of o, are added to each previous scene graph.

Figure 8: Example input sequence. Time progresses from upper left to lower right.

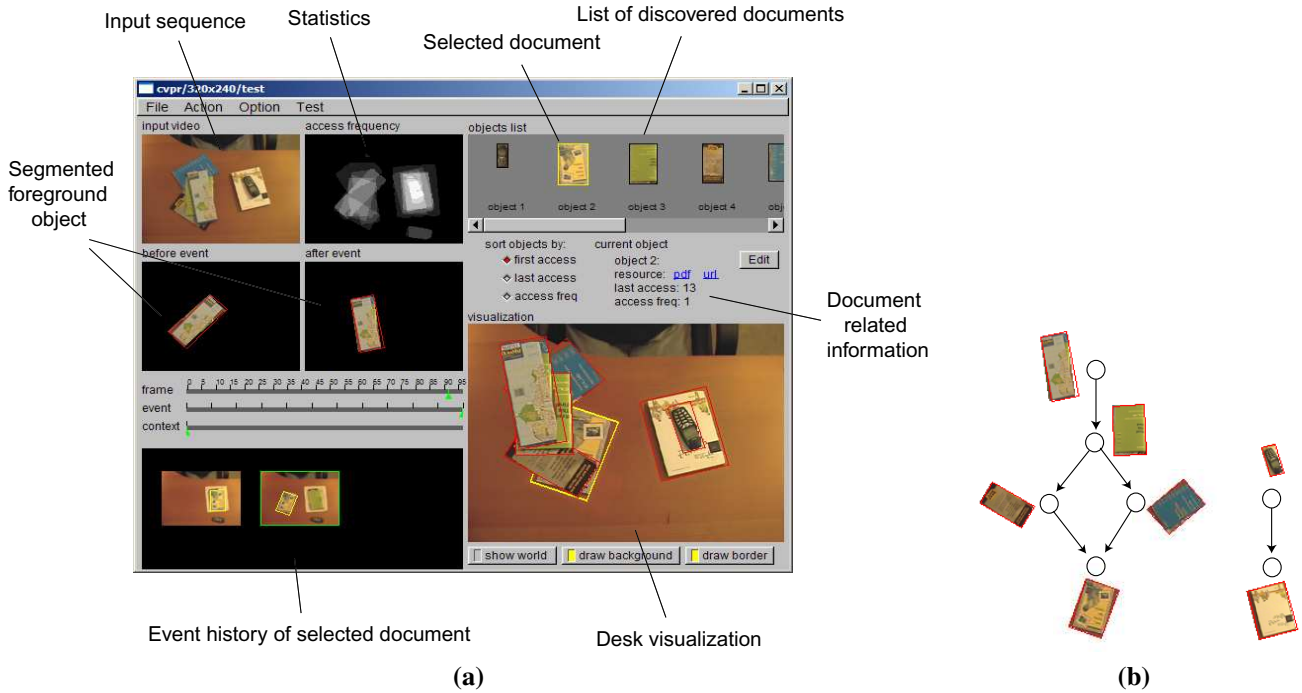

Figure 9: (a) Screenshot of a prototype query interface. The user selects the document of interest from the list of discovered documents on top right. The desk visualization expands the stack containing the selected document and highlights the document in yellow. The history of the selected document's movements is shown on bottom left. The statistics window shows the frequency of events in each image location, encoded as pixel intensity. The user may access document related information by clicking on the appropriate links. See a demo video at http://grail.cs.washington.edu/office/rtv4hci. (b) The scene graph corresponding to the most recent state of the desk.

4. Results

In this section, we demonstrate two example applications. The first one is a prototype of a generic interface to our system supporting a set of basic queries to locate a physical document and look up document related information. The second is a "virtual desktop" application that allows the user to virtually browse the document stack with mouse interaction. A demo video of both applications can be viewed at http://grail.cs.washington.edu/office/rtv4hci.

Both applications are demonstrated using the input sequence shown in figure 8. The original video was captured at 320x240 resolution and one frame per second. Each event took 2-3 minutes to process on an AMD Athlon 1.33 GHz PC. The sequence contains 11 events and 8 objects, including 7 documents and a cellular phone. Note that multiple documents overlap with each other in a complex manner, giving rise to the graphical structure shown in figure 9 (b), rather than clean stacks of documents.

Figure 9 shows a screenshot of the query interface, with accompanying descriptions of each interface component. It allows the user to visualize the current desk state, browse the history of the desktop, and view some simple statistics. The user can query a document of interest by selecting it from the list of objects shown on top right. When a document is selected, the visualization expands the stack containing the document and highlights it in yellow. If any document related information is stored in the system (e.g., a PDF version of the document), the user may access it by clicking on the relevant links. The user can browse the history of the desktop in three different modes: by individual frames, by events, and by events of the selected document. The statistics window shows the frequency of events in each image location, encoded as pixel intensity.

The second application enables the user to interactively search the virtual desktop with mouse interaction. The user is provided with an image of the current desktop. The objects in the image can be interactively moved around with the mouse, allowing the user to virtually browse the content of the stack on the desk. This may be particularly useful when the user is accessing a desk in a remote location. Figure 10 shows a screenshot of the interface.

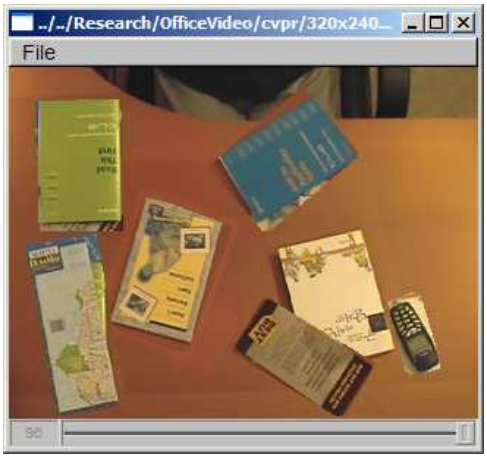

Figure 10: Screenshot of the virtual desktop application. Note that the objects have been click-and-dragged out of their places by the user to reveal the object under them. The original state of the desk is shown in the last image of figure 8. See a demo video at http://grail.cs.washington.edu/office/rtv4hci.

5. Discussion and future work

There are a number of future directions for improving and extending the current work. First of all, although we have shown the possibility of a desktop document tracking system by implementing prototype applications, we need to extend the capability of the vision system to handle more realistic situations. For instance, it limits the usefulness of the system to assume that only one document can move at a time, and only the topmost document can move. Relaxing these assumptions will require the ability to track multiple hypotheses, to cope with the increased uncertainties in tracking and recognition. At the same time, we can allow the system to solicit assistance from the user to resolve uncertainties and recover from mistakes. Another limitation of the current system is the assumption that the documents have distinct appearances. We think that using a feature-based matching technique such as [18] will enable us to differentiate documents with similar appearance, and make the system perform in real-time.

Secondly, we would like to extend the user interface to provide easier ways of interacting with the system, as well as support a wider set of queries. For instance, the user may use the system to find all documents that he did not use for the past 30 days, to help him clean his desk. The system needs to support ways to specify implicit search criteria, rather than forcing the user to manually traverse the list of objects and select the document of interest. To support an easier way to look up information related to a document in hand, we can also provide a "query by example" interface, where the user can simply "show" the document to the camera. A related interesting query is dealing with written annotations on documents. By detecting changes on the document surface, the system may be able to automatically "lift" these annotations and store them for the user.

Finally, this work could be extended to domains outside of the office environment that may also benefit from a similar tracking and recognition system with a video camera. Examples include bookshelves (e.g., library, bookstore), CD/DVD racks, bulletin boards, laboratories, warehouses, and kitchen counters.

Acknowledgments

This work was supported in part by National Science Foundation grant IIS-0049095.

References

[1] A. J. Sellen and R. H. R. Harper, *The Myth of the Paperless Office*. Cambridge, Massachusettes: The MIT Press, 2002.

[2] A. Kidd, "The marks are on the knowledge worker," in *Proc. of CHI*, 1994, pp. 186–191.

[3] 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 *Proc. of SIGGRAPH*, 1998, pp. 179–188.

[4] P. Wellner, "Interacting with paper on the DigitalDesk," *Comm. of the ACM*, pp. 86–97, 1993.

[5] N. Takao, J. Shi, and S. Baker, "Tele-graffiti: A camera-projector based remote sketching system with hand-based user interface and automatic session summarization," *Int. Jour. of Computer Vision*, pp. 115–133, 2003.

[6] D. Moore, I. Essa, and M. Hayes, "Exploiting human actions and object context for recognition tasks," in *Proc. of Int. Conf. on Computer Vision*, 1999, pp. 80–86.

[7] K. Fujii, J. Shimamura, K. Arakawa, and T. Arikawa, "Tangible search for stacked objects," in *Proc. of CHI*, 2003, pp. 848–849.

[8] D. Rus and P. deSantis, "The self-organizing desk," in *Proc. of Int. Joint Conf. on Artificial Intelligence*, 1997, pp. 758–763.

[9] H. Koike, Y. Sato, and Y. Kobayashi, "Integrating paper and digital information on enhanceddesk: a method for realtime finger tracking on an augmented desk system," in *ACM Trans. on Computer-Human Interaction*, 2001, pp. 307–322.

[10] T. Arai, K. Machii, S. Kuzunuki, and H. Shojima, "Interactivedesk: A computer augmented desk which responds to operations on real objects," in *Proc. of CHI*, 1995, pp. 141–142.

[11] W. E. Mackay and D. Pagani, "Video mosaic: Laying out time in a physical space," in *ACM Multimedia*, 1994, pp. 165–172.

[12] J. Rekimoto and Y. Ayatsuka, "Cybercode: Designing augmented reality environments with visual tags," in *Proc. of Designing Augmented Reality Environments (DARE)*, 2000, pp. 1–10.

[13] R. Want, K. P. Fishkin, A. Gujar, and B. L. Harrison, "Bridging physical and virtual worlds with electronic tags," in *Proc. of CHI*, 1999, pp. 370–377.

[14] J. Y. A. Wang and E. H. Adelson, "Representing Moving Images with Layers," *IEEE Trans. on Image Processing Special Issue: Image Sequence Compression*, pp. 625–638, September 1994.

[15] B. C. Sanders, R. C. Nelson, and R. Sukthankar, "A theory of the quasi-static world," in *Proc. of Int. Conf. on Pattern Recognition*, 2002, pp. 1–6.

[16] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," in *Proc. of Int. Conf. on Computer Vision*, 1999, pp. 377–384.

[17] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," in *Proc. of SIGGRAPH*, 1996, pp. 171–180.

[18] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. of Int. Conf. on Computer Vision*, 1999, pp. 1150–1157.