# Terms

Computer how-to documents

Computer help resources

Computer support resources

Computer knowledge base

Computer knowhow resources

Technical documents

Technical resources

Resources? Documents? Articles? Webpages?

Online technical resources??

# Needs for Research

Computer literacy is an important issue. Traditional methods for gaining literacy n computers are taking courses and reading books. This is good for acquiring general computer literacy. There are also many specific computer knowledge. Users can visit the official site of the software vendor or consult the official manual or read the built-in help. But the amount of information is limited. That’s why there’s a market for computer books. Reading the books is hard. It is restricted to the book on the user’s bookshelf. Books can be expensive.

Fortunately, there are tons of such knowhow knowledge on the web.

That’s why people look at the Internet. People no longer need go to bookstores. People gain knowledge from the Web. Some knowledge is general and can be applied over and over. The easy to learn and remember. Some is transient and disposable, only done it once. The goal is to follow the steps accurately.

However, search engine has the largest coverage. It is too general. It’s not suitable for computer knowledge. There is no specialized system for accessing and searching this knowledge base. People are aware of this problem. So they built specialized search engines built for special domains, such as news to show times and images, products to show prices and images, restaurants to show locations and ratings. These were mined from unstructured heterogeneous data.

There are specific needs for each stage of the knowledge life cycle.

**Need for an open platform for creating contextually accessible technical resources (creation).** Contextual help is useful. But its creation requires access to the API, which can be expensive. Thus, many references are created. But they are created independent of the applications, external the application. It is hard to link the page to the application. Thus, there is a need for a open platform for creators to easily link online technical resources to applications.

**Need for a centralized index of online technical resources (aggregation).** Articles containing computer knowhow are scattered all over the Web. But there has not been any systematic effort to collect and index them. In order to build a specialized search engine for these articles, we need to be able to discover and identify all these different types of documents. This can be challenging because these articles take on various forms. They can be about various operating systems, Windows, Mac OS, Linux. They can be called guides, manuals, walkthroughs, tutorials, or how-to. They can be written for novice and experts. Some have only text. Some have screenshot images. Some are videos. The algorithm needs to recognize all of them while at the same time being able to avoid including articles about software but not related to knowhow such as product review and catalogs.

**Need for an indexing scheme tailored to the properties of online technical resources (indexing).** Schemes traditionally adopted to index Web articles are based on keywords. Articles are broken down into keywords. An inverted index is constructed for efficient retrieval, where each word is a key and stored under each key is a list of articles containing the word. However, this scheme is not optimal for computer knowhow articles. It has been observed that users can follow articles faster when the articles contain screenshots to illustrate each step compared to text-only articles [cite]. It is conceivable users may prefer articles with screenshots. Since these articles are indexed only by the keywords they contain, there is no way to search them by visual contents. Moreover, for efficient retrieval, we need a way to break images down into *visual* keywords as well so that we can add them to the inverted index. Thus, there is a need for an optimized indexing scheme that not only considers both textual and visual content but also lends itself to the construction of an inverted index for efficient retrieval.

**Need for an intuitive input method for specifying query terms (querying).** Current input method for specifying queries is difficult. It is hard to come up with the right keywords. To identify the context, there are too many things need to be specified. Users need to indicate the operating system, the name of the application, the name of the window, and the topic, what the users want to know. The result is a long list of keywords, which can be ambiguous. The context and topic are indistinguishable from the search terms. From usability point of view, it is time consuming to type all the keywords.

**Need for a ranking scheme (ranking).** Current ranking schemes either based on text content or on image contents. However, simply ranking articles by images may result in articles at the top containing no useful text. Simply ranking articles by text may result in informative articles but on the wrong computer application.

**Need for a scheme to display excerpts to help users judge relevancy quicker (previewing).** Current presentation scheme tends to show except from the page by extracting excerpts and highlight the occurrences of the search term. This allows users to know how the search terms are in use. However, it is still difficult to quickly judge whether an article is relevant. The result may contain all the keywords, but without actually following the link to visit and read the page, the user may not really know whether the page is really relevant to the particular application window.

**Need for a more effective, convenient, interactive way to follow knowhow articles (consuming).** After the users found an article relevant to the task at hand, it is still challenging to actually follow the article. The users may already be at a step. It is hard to know where in the article the users can read about his current step. It is difficult to search within the reference for the part relevant to the current step. The users need to switch back and forth between the application and the reference. This is very inconvenient. It is even worse if the reference material is a screencast. The video player needs to be paused and played periodically. Seeking within the video is not possible.

# Goals and Objectives

The overarching goal of the proposed project is to address the needs identified in all stages of the knowledge life cycle as described above.

Develop an algorithm for detecting articles containing computer knowhow.

Users should be able to ….

# Proposed Methods

We propose a multi-modal approach. We use images and text.

## Creation

Allow content providers to attach arbitrary information to programs by images.

Establishing visual links. Submit an url to the system. And it downloads all the image and index the page. We have also explored using screenshots as hooks for annotation. Annotation systems are common on the web (e.g. WebNotes2 and Shiftspace3), where URLs and HTML page structure provide robust attachment points, but similar systems for the desktop have previously required application

support (e.g. Stencils [8]). Using screenshots as queries, we can provide general-purpose GUI element annotation for the desktop, which may be useful for both personal and community contexts. For example, consider a dialog box

for opening up a remote desktop connection. A user may want to attach a personal note listing the IP addresses of the remote machines accessible by the user, whereas a community expert may want to create a tutorial document and link the document to this dialog box. We plan to build an annotation interface allows a user to save

screenshots with custom annotations that can be looked up using screenshots. To save a screenshot of a GUI element, the user draws a rectangle around it to capture its screenshot to save in the visual index. The user then enters the annotation to be linked to the screenshot. Optionally, the user can mark a specific part of the GUI element (e.g., a button in a dialog box) to which the annotation is directed.

## Aggregation

There are many existing knowhow knowledge on the Web. We want to aggregate them into a searchable index. The brute-force method would be to systematically crawl major tutorial websites. For this research, we can use a short cut method. Bootstrap method. Use image search engine, take words from a corpus such as a book. Mix in words related to software name. Keep those with screenshots. Train a visual detector for screenshot images. What properties define tutorial knowledge rticles? It is necessary to filter useless pages that are not related to computer knowledge. There are text based methods. Also an image based method. Train a classifier for detecting pages that are computer knowledge.

We used three methods to collect screenshot images to populate our database. Currently, our prototype system contains more than 150k images in its index.

First, we submitted computer-related keywords to Bing Images to collect screenshot images of interactive programs. To increase the likelihood of obtaining the desired images, we sampled keywords from title bars of the dialog windows of various computer programs. Some examples of these keywords are properties, preferences, option, settings, wizard, custom, installation, network, sound, keyboard ... etc. We turned on the filter feature to keep only illustrations and graphics, rejecting obviously non-screenshot images such as images of faces and natural scenes. Using this method, we collected approximately 100k images.

Second, we used TinEye, a reverse image search engine that can take an image as the query and return a list of URLs to nearly identical copies of the image found on the Web; it is designed primarily for copyright infringement detection. We manually captured screenshot images of more than 300 interactive windows of popular programs across three of the most popular OS platforms (XP, Vista, and Mac OS). These images were submitted to TinEye to obtain about 5,000 images.

Third, we collected a library of 102 electronic books of popular software programs. We extracted all the image figures embedded in the electronic file (i.e., PDF documents). This method produced about 50k images.

Each method has its own pros and cons. While Bing Image Search provides the best variety of images, many of them are not visually relevant to any program at all. TinEye is able to provide visually relevant images, but these images are ranked only by visual relevancy; the page containing the highest ranked image may not necessarily contain any useful information. Computer books are professionally edited and thus contain the highest quality text; however, they cover a relatively limited range of applications and their content isn't as current as compared to the Web. By using all of these methods, we hope to create a rich repository of technical information that is both visually and textually relevant to, and accessible by, general computer users.

## Indexing

We propose to index knowledge by text and images. Specialized, multi-modal indexing scheme. Compared to keyword only and text only. Develop a scheme that uses images to create context and evaluate against a sizable dataset and show statistical significance more relevant results than start-of-the-art methods, when evaluated by human users. First page result. Above the fold result. Recall and precision. Use inverted index for fast retrieval. Offline processing.

Our prototype system indexes screenshots extracted from a wide variety of resources such as online tutorials, official documentation, and computer books. The system represents each screenshot using three different types of features (Figure 2). First, we use the text surrounding it in the source document, which is a typical approach taken by current keyword-based image search engines.

Second, we use visual features. Recent advances in computer vision have demonstrated the effectiveness of representing an image as a set of visual words [18]. A visual word is a vector of values computed to describe the visual properties of a small patch in an image. Patches are typically sampled from salient image locations such as corners that can be reliably detected in despite of variations in scale, translation, brightness, and rotation. We use the SIFT feature descriptor [11] to compute visual words from salient elliptical patches (Figure 2.3) detected by the MSER detector [12]. Hamming Embedding. Spatial Pyramid Matching.

Screenshot images represented as visual words can be indexed and searched efficiently using an inverted index that contains an entry for each distinct visual word. To index an image, we extract visual words and for each word add the image ID to the corresponding entry. To query with another image, we also extract visual words and for each word retrieve from the corresponding entry the IDs of the images previously indexed under this word. Then, we find the IDs retrieved the most number of times and return the corresponding images as the top matches.

Third, since GUI elements often contain text, we can index their screenshots based on embedded text extracted by optical character recognition (OCR). To improve robustness to OCR errors, instead of using raw strings extracted by OCR,

we compute 3-grams from the characters in these strings. For example, the word system might be incorrectly recognized as systen. But when represented as a set of 3-grams over characters, these two terms are {sys, yst, ste, tem} and {sys, yst, ste, ten} respectively, which results in a 75% match, rather than a complete mismatch. We consider only letters, numbers and common punctuation, which together

define a space of 50,000 unique 3-grams. We treat each unique 3-gram as a visual word and include it in the same index structure used for visual features

## Ranking

Query dependent. rank results by both visual and textual relevance. Identify many features. Learn ranking, using RankSVM. Improve visual search. Provide faceted search function. Consider more features.

Since not all features are equally important, it is necessary to set the weights appropriately in order to reflect these features' relative importance. In developing the prototype of our system, we set the weights in three stages. Table \ref{tbl:feature\_weights} lists all the features and indicates the stages when the weights are set. First (stage 1), we apply RankSVM to learn feature weights based on the training data collected using Amazon Mechanical Turk. RankSVM was

originally proposed by Joachims \cite{Joachims} to learn how to weight features from a set of subjective ordering constraints inferred from user click-through data to improved ranking. At the development stage, we did not have any click-through data to derive ordering constraints for RankSVM. Thus, we recruited workers from Amazon Mechanical Turk to provide us explicit article ratings from which to infer ordering constraints. We chose five query images known to retrieve a large of

number of articles, but they are ranked only by visual similarity. We shuffled the order of these articles and presented them in a list, Each worker was asked to rate each article in terms of usefulness on a 5-point scale. For quality control, we included two junk results and two duplicate results. To pass the quality test, workers must

successfully mark junk results as useless and give consistent ratings to duplicate results. Then, each pair of ratings provided us with an ordering constraint for training RankSVM. Note that since there was no easy way to ask the workers to provide meaningful keywords, this procedure only applied to the subset of features that do not depend on keywords.

## Querying

allow users to capture sceenshot as query. Allow users to type keywords as query. Use Java to provide cross-platform applicability. Allow users to take multiple screenshots as query. User Interface for Searching Screenshots Sikuli Search allows a user to select a region of interest on the screen, submit the image in the region as a query to the search engine, and browse the search results. To specify the region of interest, a user presses a hot-key to switch to Sikuli Search mode and begins to drag out a rubber-band rectangle around it (Figure 1). Users do not need to fit the rectangle perfectly around a GUI element since our screenshot representation scheme allows inexact match. After the rectangle is drawn, a search button appears next to it, which submits the image in the rectangle as a query to the search engine and opens a web browser to display the results.

Our proposed search engine supports mixed-modality queries in order to optimize the effectiveness in searching technical articles about interactive programs. A typical query consists of a screenshot of a program and, optionally, a set of keywords to specify what aspects of the program the retrieved articles are supposed to cover.

Screenshot queries can be specified in two ways. First, users can run a cross-platform, Java-based, client interface we developed to capture the screenshot of a selected window or an arbitrary screen region. The client interface will submit the screenshot as the image query to our search engine and display the search results in the default web browser. Alternatively, users can use any existing image capture utilities such as the snipping tool on Windows Vista or the Command-Shift+4 hotkey on Mac OS to capture screenshots. Users can use a Web interface to submit the screenshots to our search engine and view the results directly in a Web browser.

Keywords queries are optional and can be specified in three ways. They can be entered together with the screenshot queries on both the client and Web interfaces. Also, as the users are browsing the results, they can enter keywords to filter and/or refine the results.

## Previewing

show image excerpt. Show screenshot in context to let user know the context, what words are before and after that.

## Consuming

monitor the entire screen, matching the screen to the images in a tutorial article. Automatically scroll the page to that image. Allow users to search for content by image. Browser function called Find by image.