5-2007

YMAGE : a resource for real-time sharing of high resolution digital images.

Yetu A. Yachim
University of Louisville

YMAGE: A RESOURCE FOR REAL-TIME SHARING OF HIGH RESOLUTION DIGITAL IMAGES

By

Yetu A. Yachim
B.S., University of Louisville, 2006

A Thesis
Submitted to the Faculty of the
University of Louisville
J.B. Speed School of Engineering
in Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science

May 2007

YMAGE: A RESOURCE FOR REAL-TIME SHARING OF HIGH RESOLUTION DIGITAL IMAGES

Submitted by:______________________________________

Yetu A. Yachim

A Thesis Approved On

____

(Date)

by the Following Reading and Examination Committee:

____

Dr. Eric C. Rouchka, Thesis Director

____

Dr. Dar-jen Chang

____

Dr. Hichem Frigui

____

Dr. R. Eric Berson

ACKNOWLEDGEMENTS

I am indebted to those who have provided me with the technical knowledge and resources needed to complete this project: Dr. Eric Rouchka, Dr. Dar-jen Chang, and Dr. Hichem Frigui. To the friends, students and faculty alike, who have helped me through the Speed School undergraduate program I give many thanks. And, to my parents, without whose support I would not have realized any of my achievements, I offer my deepest gratitude.

ABSTRACT

Digital images have primarily been viewed using desktop applications. These types of programs attempt to load all of an image’s graphical data into memory in order to display the image in its entirety. The method is effective for images that are relatively small, but for large, high-resolution images, this tends to be slow and resource-exhaustive. Coupled with degraded system performance are the issues of storage and image access. Some newer programs make efforts to mitigate these problems, but few do so satisfactorily. The Ymage System was designed as an alternative to conventional approaches for managing and viewing images. The system provides high accessibility to data while minimizing demands on the machine from which the image is viewed. The design accomplishes these objectives and even the initial, low-performance implementation presented here can compete with existing systems. Further, it appears that the flexibility of the design lends itself to use for non-visual applications such as distributed image processing. It is possible that development of the Ymage System could make a significant impact on the approach to the development of image-handling software in the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The use of digital images has become critical in many areas. In medicine, for example, there is heavy reliance on digital imagery for diagnosing physical health problems. To be of any use, these images must be very detailed, but this presents issues. Detailed images require a great deal of storage space and a management system that can handle large files. Given the satisfaction of these requirements, there is also the problem of viewing the image. Traditional visualization and manipulation programs seem to target relatively small images, so using them on larger files results in a significant reduction of system performance. It is also rare for these programs to provide a means of accessing images that reside in the storage devices of systems on which they are not immediately running. The development of a system that effectively handles the storage, access, and viewing problems described would be of great value to the many disciplines that rely on digital imagery. The aim of this project was to design a system that facilitates the use of large images by efficiently managing image storage, retrieval, and display.

A. Digital Images

Digital images are numerical representations of visual information [1]. They can be formed by capturing and digitizing naturally existing visual data, or by generating digital data using computer programs. Digitizing an image involves mapping visual data points to discrete numerical values on cells of a digital grid. These grid cells are called picture elements, or pixels and the collection of adjacent pixels, each representing a different point, composes an entire digital image. How closely the image resembles the original visual data depends on pixel density and pixel depth. The former is a measure of how many pixels represent a fixed visual region; the latter describes the possible variability between pixels. As pixel depth and pixel density increase, so does the quality of the image, but the higher the image quality, the greater the amount of space required to store the image. Herein lies the image quality versus image size trade-off. The dilemma has produced man different file formats, each attempting to provide the right balance for particular use of images [2]. Since systems that deal with digital images must inevitably deal with files, it is important to look into the factors involved in the storage and retrieval of image files.

1. Image File Storage

Digital images contain several key components. The most important is the graphical data set which represents the true visual data; it can be in raster, geometric, or latent forms. Raster and geometric data have direct visual mappings and so they can be immediately rendered on a display device; latent data requires transformation into one of the other forms before rendering. In addition to the graphical data, image files usually contain self-describing information in the form of signatures, or magic numbers, and version numbers. Metadata and non-graphical data may also be present in the file, but these are dependent on the specific file format and the program used to generate it.

The data contained in an image file must be ordered in some fashion. The arrangement of data can be chosen to maximize retrieval efficiency. To achieve this, metadata should be at the beginning of a the file, followed by the file identification, then graphical data which can be preceded or succeeded by the non-graphical data [2]. The placement of metadata first provides the file reader with any important information upfront that may influence the way in which the file is used and prevent unnecessary traversal through the remaining data. Since an understanding of the file’s structuring of graphical and non-graphical data is vital to reading it effectively, the identification should follow the metadata. It would also be reasonable to keep the graphical and non-graphical data in independent, contiguous blocks. This would eliminate the need to skip about the file in search of these data. Though this structuring would provide the most benefit in terms of ease and flexibility of reading, not all image files are organized in this way [2].

Regardless of data order, the contents of image files need to be present in some form. Various encoding schemes are used for this purpose. Text encoding represents a data set in human-readable form that can be manipulated directly with simple text-editors. This encoding scheme usually yields very large files and requires a lot of processing before the display of the image is possible so it is not often used when storage space requires consideration. Character encoding presents data in a form that replaces equivalent text-encoding blocks with shorter data codes. This saves space, but reading this data requires software that understands these codes. Displaying images encoded in this way also requires processing. Binary encoding saves space and can be rendered directly, but the data is specific to a particular hardware platform and manipulating data of this type also requires special software. With all three encoding schemes, there are trade-offs usually between storage space and retrieval time. The application to which an image will be put dictates the selection.

Image data sets are often transformed to reduce the amount of space required for storage; this transformation is called compression. Several techniques are used to compress images [2]. Since the visual data that images represent are not random, analysis of an image can reveal features or patterns. One compression strategy involves exploiting these patterns by replacing them with succinct representations. This is particularly effective when features repeat throughout an image. Sometimes the data set that represents the image is too descriptive. In these cases, the precision of the image can be reduced by eliminating superfluous detail without losing any significant information; this is a reduction in image resolution. Yet another compression method reduces the number of bits needed for representing the values in the data set and is the same as minimizing pixel depth. It is possible to reduce image precision and the number of available pixel values (colors) for display purposes because the eye "adapts to a certain intensity level for a given field of view" [1] and does not differentiate values that are similar. Many applications require very high quality images which, as mentioned above, require large raw storage space. To make the storage of these images practical, compression is an imperative.

2. Image File Retrieval

The retrieval of image data from a file is, in theory, relatively easy. For data that is logically structured, the process involves opening a stream to the file, obtaining the metadata and format identification information, and then reading the desired data.

For unstructured files that cannot be randomly accessed, the reading process may require traversal of a significant portion of the file before reaching the required data. Once this data is reached, transformations may also be needed before the image can be rendered. These transformations may include decompression, decoding, and rasterization. After extracting and processing the image data, it can be sent to a display device, such as a monitor for visual presentation.

B. Applications of Digital Images

The significance of the digital image is clear from its widespread and varied uses. Images can be used to represent many different types of data and are heavily relied upon to help identify and solve different problems.

Digital images are commonly used like their analog counterparts for simple display purposes. They can be acquired through photography, generated with the aide of software, or scanned from film. Their visual appeal makes them useful for advertisements, art galleries, educational and instructive media, online consumer catalogs, and other non-analytical purposes.

A relatively recent use of digital images has been in the area of document storage. The representation of textual documents in digital image form is a useful way to preserve the contents of deteriorating analog documents. Not long ago, Google began an initiative to scan library books and these scans to a collection of searchable digital content [3]. Optical character recognition (OCR) technology provides the option of converting these images to texts for textual analysis purposes. The combination of visual detail without loss of textual information makes images useful for representing documents.

The true power of images lies in the analytical uses to which they can be put. The medical field makes extensive use of digital imagery in this way. Images produced by radiologists capture physical properties of the human anatomy through X ray, ultrasound, magnetic resonance, and positron emission [4]. These images can be analyzed visually or computationally to investigate anatomical features and physiological functions. The information gathered from the analysis of these images is relied upon for diagnosing health problems and possible treatments.

Other scientists also use digital image. Geologists, for example, employ digital imaging to study the Earth. Data that they collect from seismic surveys (a sound reflection technique) can be compiled to describe geological formations digitally [5]. The physical attributes of an area can be determined and used for various studies and commercial purposes, such as land changes over time that may determine safe building areas or profitable oil-drilling sites.

Remote sensing, or satellite imagery, is an area in which one of the earliest applications of digital imaging was made. Satellite images capture surface and atmospheric information from space with reflective wave methods. A great number of images are captured and the data available in these images can be coupled with other information, such as geological data, to reveal geospatial details.

Digital imagery is an integral part of many existing disciplines and advances will naturally prove them relevant to emerging fields. The significance of digital images provides sufficient motivation to investigate how they can be better managed and more effectively used.

II. EXISTING SOFTWARE

The uses described in the previous section show that digital images are very important. Their varied uses attest to their flexibility as a medium of data representation. Though the problems that they help to identify and solve may require different analytical approaches, applications that deal with digital images share the common problem of file retention and access. Of the many programs written to deal with images, few adequately address the fundamental issues of storage and retrieval. The following sections examine some programs that do attempt to handle these issues.

A. Standalone/Desktop Model

The standalone model is common for applications that act on local data. Programs in this category are either fed the location of data or conduct searches for image data on local media. These programs usually do not address storage; rather, the focus is on organizing and presenting data in an organized form which differs from that of the window manager. Picasa™, ImageQuery, and Photo Organizer Deluxe are three examples.

1. Picasa™

Picasa™ is a Google application for organizing images [6]. In keeping with Google's philosophy, Picasa™ searches local media for existing image files and presents the results in an organized fashion (see Figure 1). When images are found, thumbnails are created on-the-fly and cached in DB2 databases. Information about the files is saved to eliminate unnecessary future searches. Images that are imported into Picasa™ from external media such as CD's and DVD's are copied to the local hard drives.

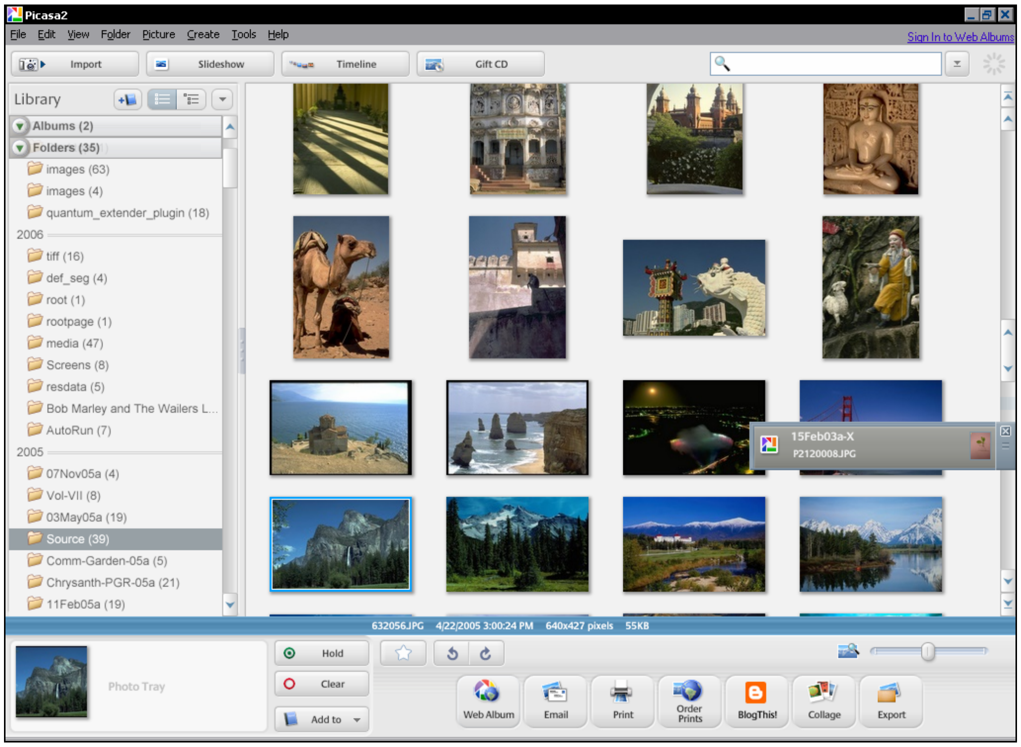
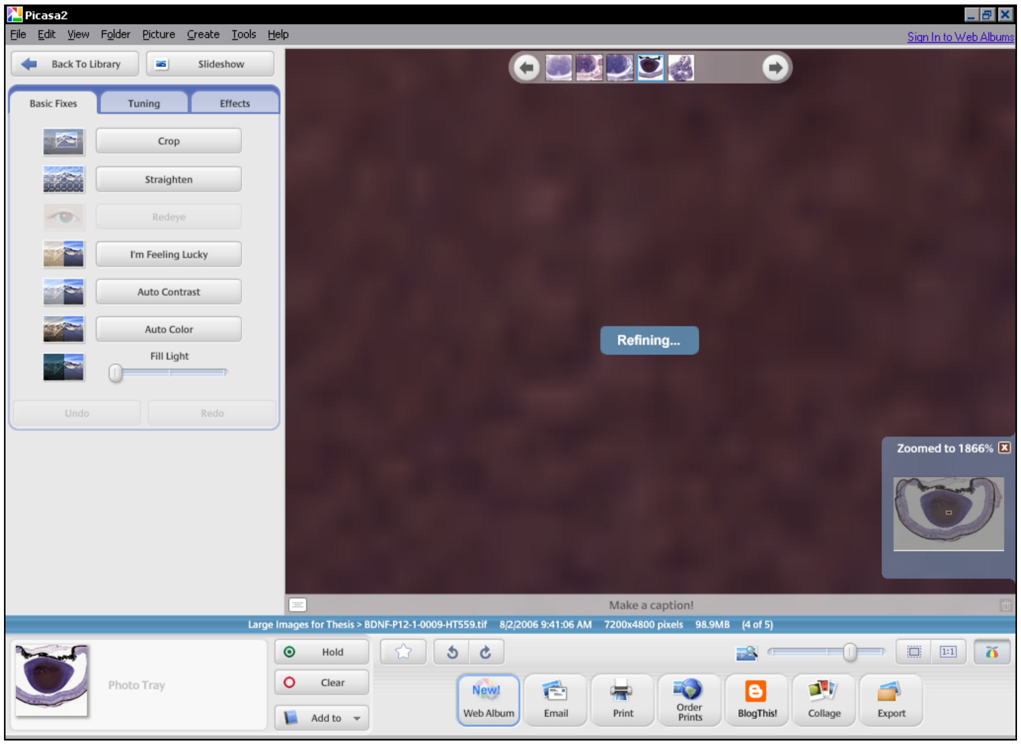


FIGURE 1 – Picasa™, Searching for Images

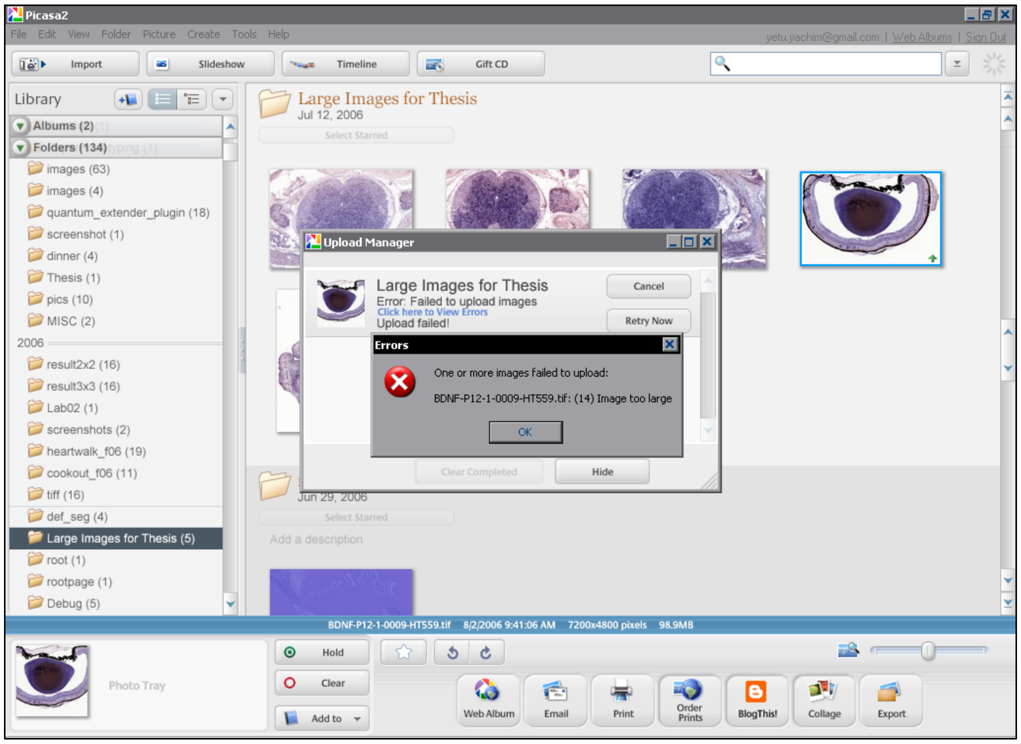
Along with file organizing, Picasa™ also provides visualization and image processing features. Clicking on a thumbnail triggers the display of a larger version of the image that can be cropped for export, more closely inspected by zooming, or otherwise manipulated. Documentation of its internals is not available, but it appears, from Figure 2, that Picasa™ loads only the part of an image is needed. This is likely to be a factor contributing to its high response times.



Large Images for Thesis > BDNF-P12-1-0009-HT559.tif 8/2/2006 9:41:06 AM 7200x4800 pixels 98.9MB (4 of 5)

FIGURE 2 – Viewing a Large Image in Picasa™

Picasa™ also has the ability to upload images to storage systems that Google maintains. Options are available to reduce upload time and storage size by compression and resizing. Without these modifications, the original images could be too large to store on the available space, as in the case demonstrated in Figure 3. It is unknown, here, whether Picasa™'s upload tool could not handle the size of that particular image or whether the problem was insufficient storage space. In either case, this exemplifies the size issue that must be considered when dealing with high resolution images.



Large Images for Thesis
Error: Failed to upload images
Click here to View Errors
Upload failed!

One or more images failed to upload:

BDNF-P12-1-0009-HT559.tif: (14) Image too large

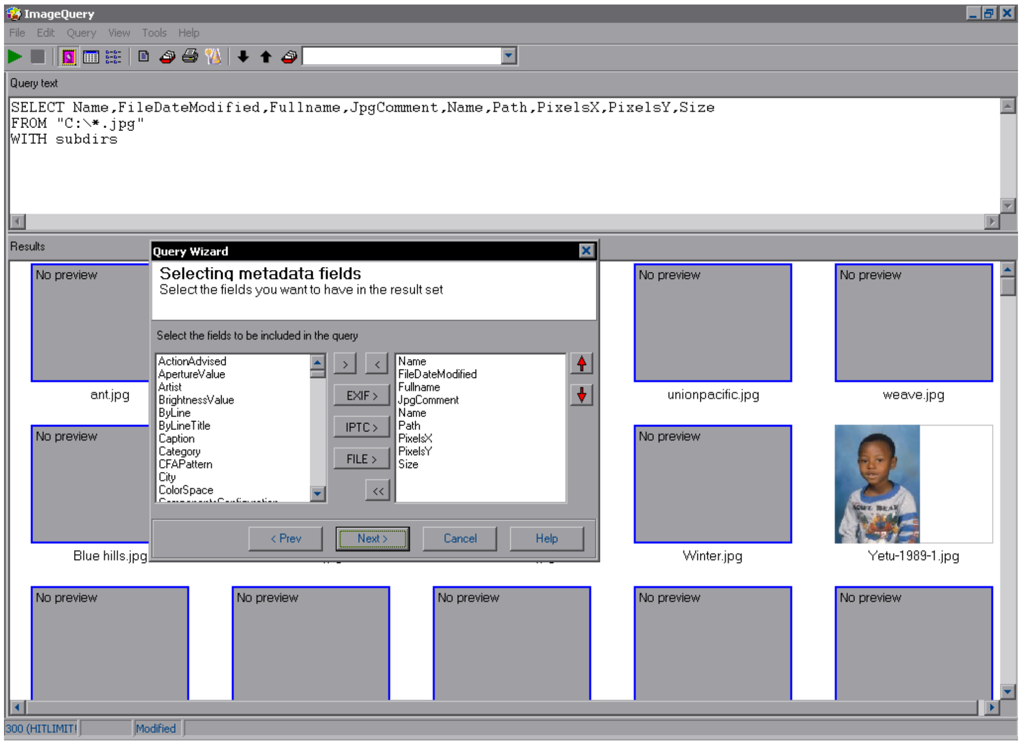
BDNF-P12-1-0009-HT559.tif 8/2/2006 9:41:06 AM 7200x4800 pixels 98.9MB

FIGURE 3 – Picasa™ Image Upload Failure

Picasa™ is a useful tool for locating and viewing images that reside locally, but it is not ideal for several reasons. First, the organizational features are only on the surface. The implied structure of the data is not maintained outside of Picasa™. This introduces a dependency of the data on the program and is problematic. Should the program malfunction or become obsolete, the apparent organization of the data is not necessarily guaranteed. Second, Picasa™ only truly maintains images that are locally stored. While it permits uploads to its storage facilities, and thereby provides greater access to those images, it does not appear that there is any way of viewing the remote version of uploaded images through the client Picasa™ software; these must be viewed through other services that Google provides. Upload to non-Google systems is also only possible through email -- a medium not well suited for transferring large images. Furthermore, storage space on Google servers requires yearly payment for capacity greater than 1024 megabytes. And, third, although performance is reasonable when viewing large (about 100-megabyte) images, there is still significant lag in responsiveness every time the image is viewed anew. At times, the lag is long enough to suggest that the application has frozen. These issues make Picasa™ inadequate for handling images of substantial size.

2. ImageQuery

ImageQuery was written by Armin Hanisch with an emphasis on retrieval [7]. ImageQuery takes SQL-like strings and uses them to search the local file system. These queries can be written to find images by metadata available in the file. ImageQuery neither handles image storage nor furthers the accessibility of image files, and only minimally addresses visualization. Searches return a set of thumbnails whose images can be viewed either with the internal viewer or an external program designated for the purpose. Figure 4 shows the results of a search and the specification of a query through the Query Wizard.



```
SELECT Name,FileDateModified,Fullname,JpgComment,Name,Path,PixelsX,PixelsY,Size
FROM "C:\*.jpg"
WITH subdirs
```

Selecting metadata fields
Select the fields you want to have in the result set

Select the fields to be included in the query

ActionAdvised	Name
ApertureValue	FileDateModified
Artist	Fullname
BrightnessValue	JpgComment
ByLine	Name
ByLineTitle	Path
Caption	PixelsX
Category	PixelsY
CFAPattern	Size
City	
ColorSpace	

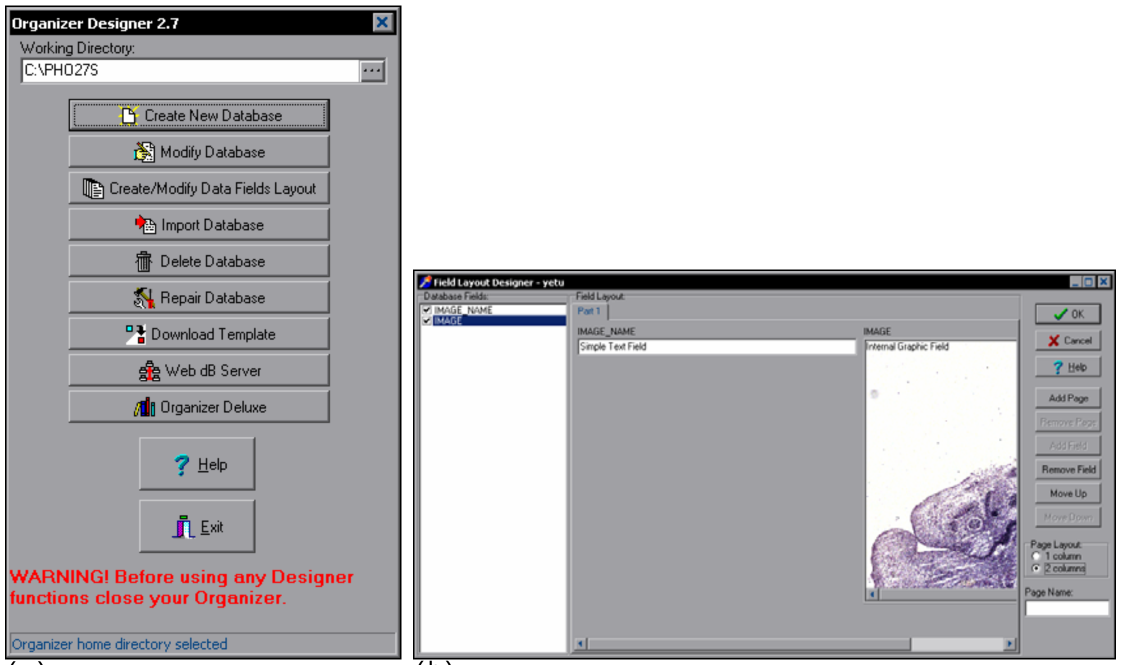
FIGURE 4 – ImageQuery, Searching for Images by Metadata

The program takes an interesting approach to file retrieval. Metadata in image files is underutilized and ImageQuery appears to be one of the few applications that try to fully capitalize on it. The fact that so little of the data is ever used, however, provides a counter-incentive to storing this data. A lack of the metadata’s availability in files then undermines the benefit that ImageQuery could provide. Consistent inclusion of metadata for files in storage would correct the problem. By adding a management component to enforce this inclusion, ImageQuery could be significantly improved.

3. Photo Organizer Deluxe

PrimaSoft's Photo Organizer Deluxe is a commercial database-centered, image storage and organizing system [8]. It is composed of two interdependent programs. The first is a database designer that allows the development of a data storage scheme consisting of various field types (see Figure 5). The second program is an organizer and viewer that facilitates data entry and visualization.

The designer is a flexible program that simplifies the process of describing how the data is to be organized. One notable feature is the option of storing image files in the database directly or inserting a reference entry that points to the file. Retaining image files directly in the database ensures that the data is present and organized as expected. The accompanying documentation warns, however, that the performance of the system may suffer if this option is used extensively [8]. This, and an inspection of system's files, indicates that the database is not meant to handle files that are very large. It is also unclear whether the data in the database is immediately accessible outside of the designer and organizer programs.



WARNING! Before using any Designer functions close your Organizer.

(a) (b)

FIGURE 5 – Photo Organizer, database designer
(a) Available options include database and server maintenance. (b) GUI-based database designing.

Although Photo Organizer makes an effort to address the storage and retrieval issues, it does not do so satisfactorily. The developers of the system noted that there may be problems with large image files, but only mitigate the problem by allowing storage outside of the system. This undermines the management control that the database should have over the data. File and web database export features are also provided to increase accessibility to data. However, the first is impractical for large images and the second is only available at a fee. The system is not ideal, but the idea of organizing images and accompanying data in a database may be the best approach to handling image storage and retrieval.

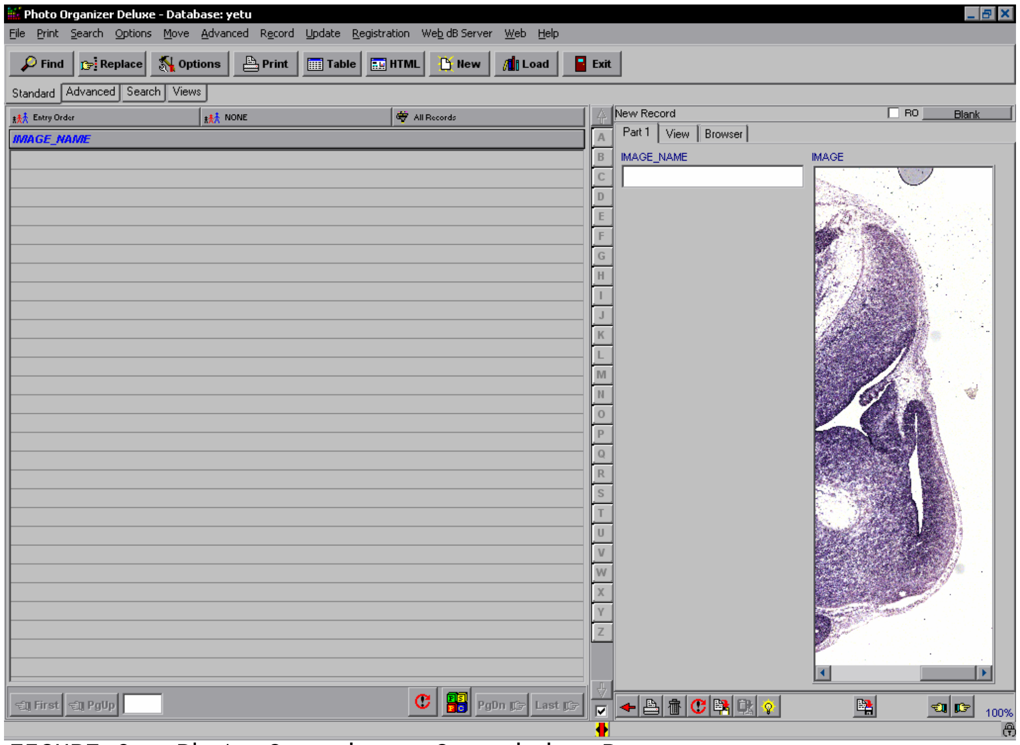


Photo Organizer Deluxe - Database: yetu
File Print Search Options Move Advanced Record Update Registration Web dB Server Web Help
Find Replace Options Print Table HTML New Load Exit
Standard Advanced Search Views
Entry Order
NONE
All Records
IMAGE_NAME
New Record
RO
Blank
Part 1 View Browser
IMAGE_NAME
IMAGE
First PgUp PgDn Last
100%
FIGURE 6 – Photo Organizer, Organizing Program

B. Client-Server Model

The client-server model is used for applications that require a high degree of access to data by users in different places. It answers the access problem by centralizing data and providing a mechanism to respond to external requests. This section describes two programs that approach image storage and retrieval by using the client-server model.

1. NASA World Wind

World Wind is an open-source application developed by NASA for viewing the entire Earth [9]. It downloads satellite data stored on Microsoft’s TerraServer database [10] and displays images for the areas being viewed. The client is designed for computers running Microsoft Windows. In Figure 7, World Wind is showing satellite images taken over Louisville, Kentucky and southern Indiana. Since the area is so large, the initial images are not detailed, but inward zooming reveals replaces these with images that represent the region in greater detail.

World Wind was written to handle geographical data, but its idea of loading parts of images on demand could be applied to other large images. The benefits are apparent. If only one image of high detail were to represent the entire Earth, an unreasonable amount of time and client storage space would be required to view it. Dividing the image into separate parts allows clients to request only the parts of the image that they need; this saves time and space.

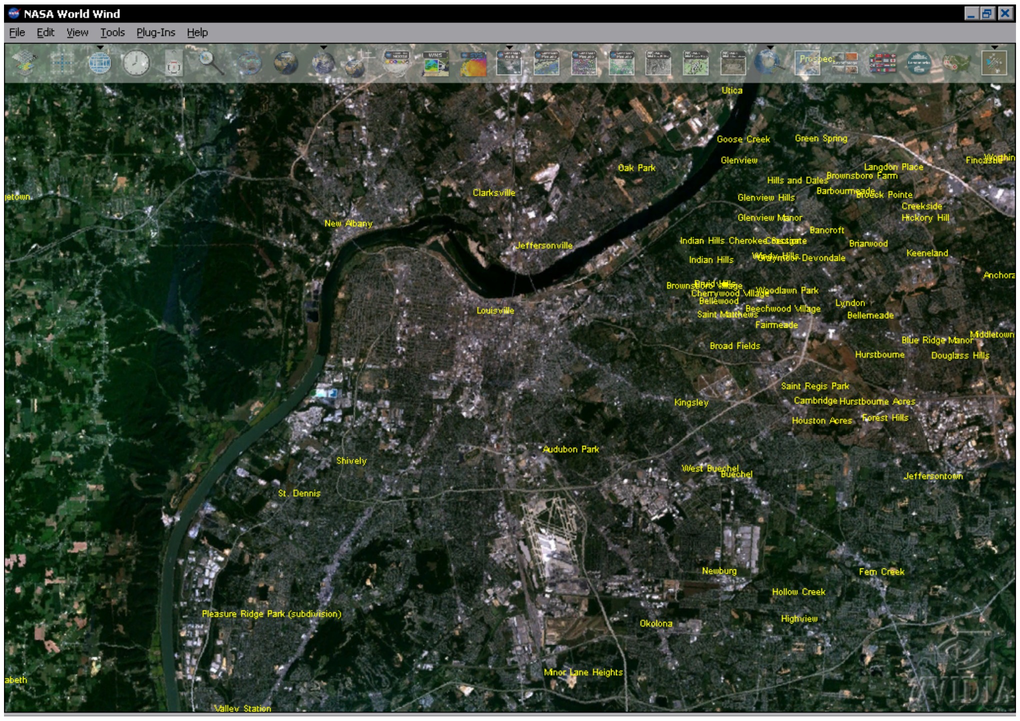
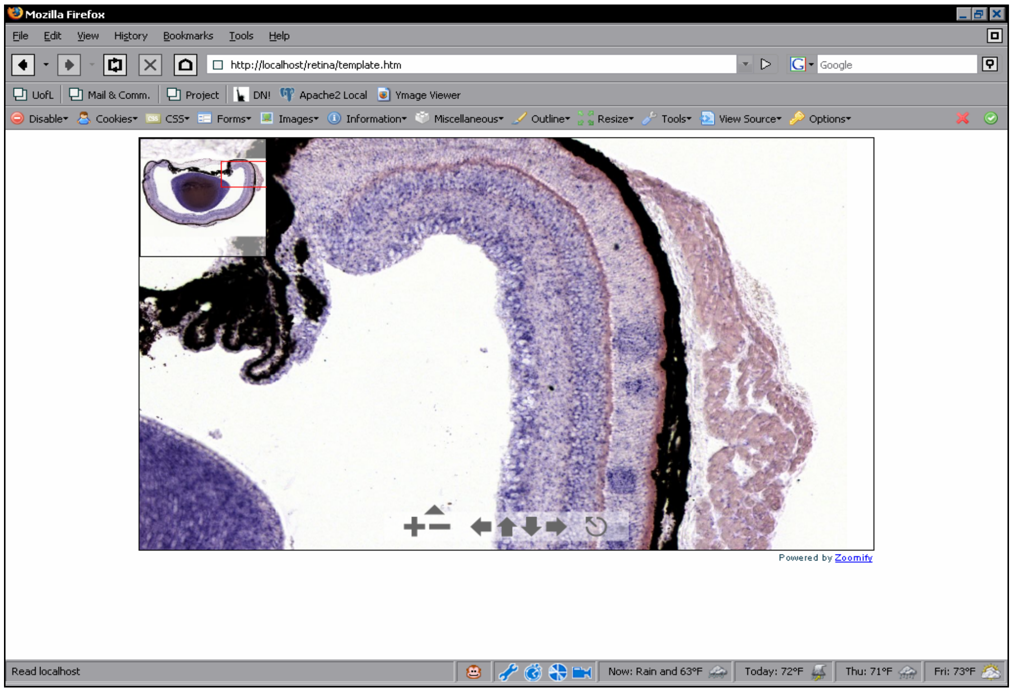


FIGURE 7 – NASA World Wind

2. Zoomify

Zoomify is a commercial system for viewing images [11] and is perhaps the system that best addresses the storage and retrieval needs of large images. Given an image, Zoomifyer EZ™ generates several thumbnails, divides each thumbnail into tiles, and compresses each tile. These tiles can then be viewed through an Adobe® Flash movie embedded in a web page. The details of this visualization element are unknown, but the behavior of the viewer suggests that the thumbnails are viewed as tile-groups and that zooming changes the tile-group in focus. Thumbnails that represent higher levels of detail are larger and consist of more tiles. During visualization, only the tiles in view are downloaded; this maximizes performance by reducing the amount of data that must be transferred for a given tile-group. Figure 8 shows a “Zoomify-ed” image being viewed in a browser-embedded Flash movie.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/retina/template.htm
Google
UofL Mail & Comm. Project DNI Apache2 Local Ymage Viewer
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
Powered by Zoomify
Read localhost
Now: Rain and 63°F Today: 72°F Thu: 71°F Fri: 73°F
FIGURE 8 – Zoomifyer EZ™, Embedded Adobe® Flash Viewer

The approach is successful and relatively well implemented, but some design issues are still apparent. Image and metadata are stored on a server as flat files and the viewer must reside on this server. The corruption of these files could prevent proper visualization. And, since these are just flat files, data maintenance (back-up, error correction, etc.) cannot easily handled. A more robust design with the ability to ensure the quality of the data over time would be an improvement.

III. THE YMAGE SYSTEM

It appears that there does not yet exist a system that fully answers the demands of large digital image storage and retrieval, but from a study of current software such as system is possible. The programs examined in the previous sections present effective solutions to different parts of the problem. It is from these approaches and an understanding of their shortcomings that the Ymage System was designed.

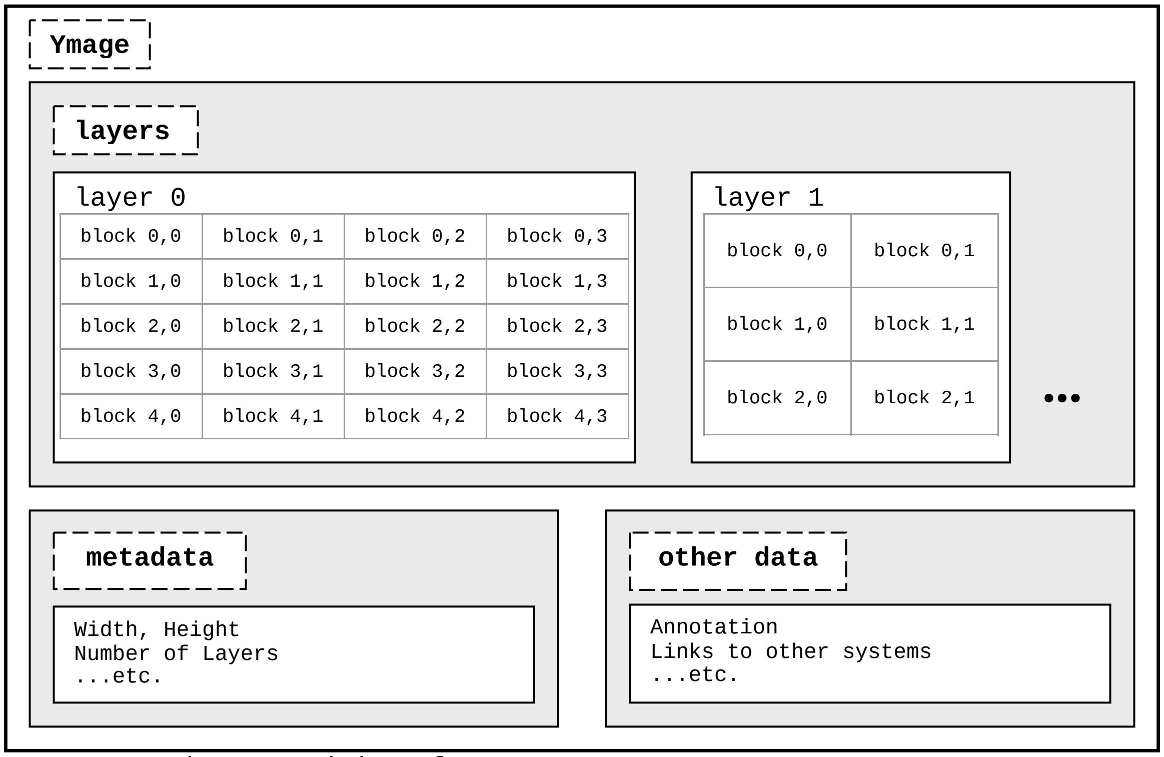


FIGURE 9 – The Composition of an Ymage

At the heart of the design is an abstraction called the “ymage.” An ymage is the collection of data that describes the original image and any additional information that may relate to it.

As illustrated in Figure 9, ymages contain layers, metadata, and “other data.” Metadata describe layers and their properties (such as width, height, the number of rows and columns, etc.) while “other data” is any supplementary information about an ymage (for example, annotation data). Layers are abstractions of the graphical data that represent scaled versions of the original image. Every ymage has at least one layer. The first (layer 0) represents the un-scaled original image; the second represents an image that is some factor smaller than the first; the third is that same factor smaller than the second, and so on. Every layer is divided into subsections called image blocks. For a given layer, all blocks are of equal dimensions but not necessarily the same dimensions as the blocks of other layers. Representing an image in this way ensures flexible access to information that describes it.

A. System Architecture

The overall system consists of three components, one to address each of the three perceived problems (i.e. storage, retrieval, and visualization). The need for structured storage is satisfied by the Data Storage Component (DSC). Transfer and manipulation of image data is handled by the Data Access Component (DAC). The Visualization Component (VC) provides the logistics for accurate image and data display. Each of the components is a relatively independent subsystem; this modularity eases system maintenance and makes the system flexible.

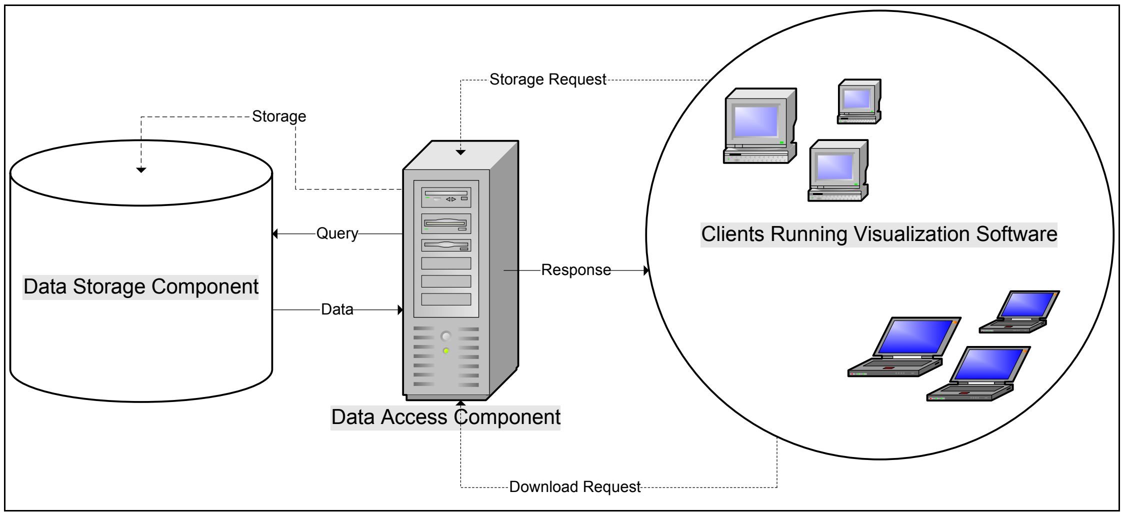


FIGURE 10 – Client-Server Design of the Ymage System

It is possible for modular systems to become inefficient if a clear protocol is not established. To reduce the likelihood of possible inefficiencies, communication paths between the different components are fixed and strictly enforced. As shown in Figure 10, client software and the Data Storage Component only communicate with the Data Access Component which is at the center of communication. The DAC validates data entering storage and polices data requests made by the client. Enforcement of this communication scheme maximizes access to data without jeopardizing its integrity or laying undue burden on the DSC.

1. Data Storage Component

The purpose of the data storage system is to retain information about an ymage. This includes the graphical data that would be stored in the files. Since this component will not restrict the types of image files that it retains, there is no guarantee that the graphical data will be accompanied by the needed metadata, so this information is stored in separate records. Other desirable data as it relates to a stored image can be retained in this system as well (e.g. annotation and image ownership information). In addition to keeping all of the image-related data in a structured manner, this subsystem provides flexible, direct access to its data.

The requirements of this component are consistent with the function of relational database management systems (DBMS). Since a DBMS stores data in customizable schemas, the structure of the data can be defined to the needs of the Data Storage Component. Most DBMS's also provide a powerful data retrieval mechanism (e.g. SQL querying) that can answer requests and return data in forms that differ from the schema. Data backup and maintenance are also incentives to use them, and additional features of a specific DBMS may introduce other benefits.

2. Data Access Component

The Data Access subsystem is the middleware component responsible for connecting the Data Storage Component to the client applications; all data storage and retrieval must be are through this component. The process of data storage begins first with the transfer of data to the DAC. After this component completes an analysis of the data, all of the necessary processing is undertaken (i.e. subdivision and compression), then the data is forwarded to storage (see Figure 11). This serves as a means of validating data integrity and ensures that only data which has been analyzed and properly processed is retained. This subsystem also handles requests from clients for data that is already in storage. By mediating access to the storage component, this subsystem can reduce the work load of storage system by capturing request statistics and making optimization decisions based on that information. This has the potential to speed up requests and overall system performance.

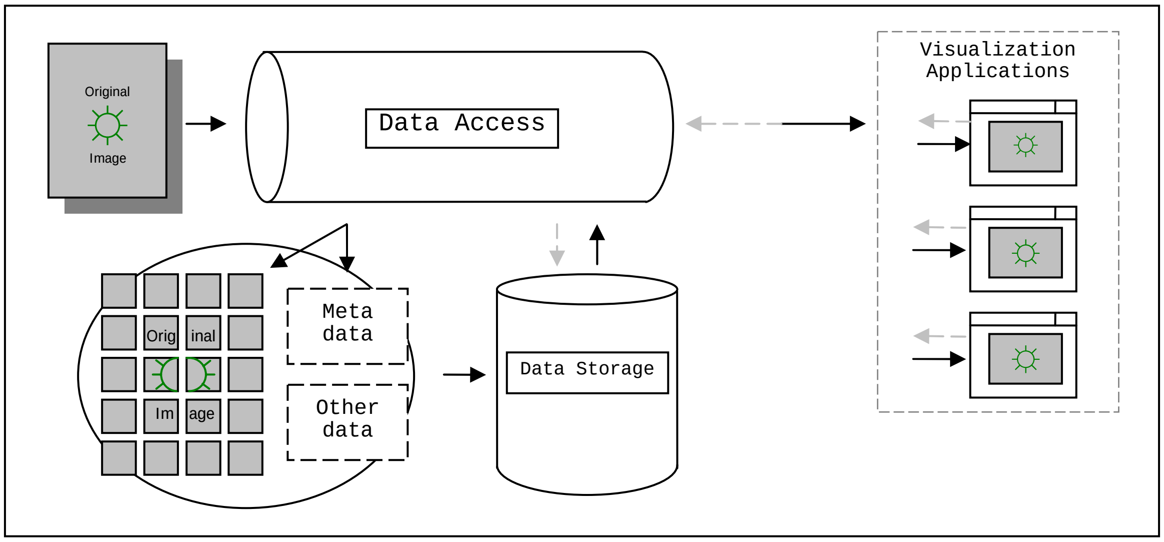


FIGURE 11 – The Role of the Data Access Component
An image is sent to Data Access where it is processed, and sent to storage. Applications can then make requests for the data through this component which finds and returns it.

Since the images targeted by the Ymage System are large both dimensionally and with regard to storage requirements, compressing and sub-dividing is necessary in order to make them usable. Integrated into the Data Access Component are the means of handling these processing needs.

In order to make requests of the storage system on behalf of the clients, this component must understand client requests. A simple mechanism consisting of a small vocabulary of commands was devised. When a request is handled, image files are returned without modification and non-graphical data is returned without unnecessary formatting.

3. Visualization Component

The Visualization Component is responsible for displaying the ymage data. To accomplish this, the system makes requests for stored data through the Data Access Component and combines the results obtained. Since the interface between the data and this component requires only knowledge of the simple vocabulary of Data Access commands, developing different types of this component is possible.

B. Implementation

This first implementation focuses on proving the feasibility of the Ymage System and developing code that can be easily understood for further development. With the future in mind, consideration was given to restrictions that may be imposed by the use of proprietary software, so external code and binaries were chosen from open-source projects. Because this version was implemented to ensure functionality, it is not a full-fledged high-performance system; section C describes some possible improvements. Nonetheless, this implementation demonstrates the idea behind the design and shows the feasibility of developing the Ymage System.

1. Development Environment and Tools

Cygwin is a Linux emulation environment for Microsoft Windows [12]. It provides access to much of the Linux API and allows the development and execution of Linux targeted software on machines running the Windows Operating System. It was chosen as the development environment for this implementation for two reasons. The first is that it increases the likelihood that code developed in this Windows platform would be portable to Linux. Cygwin was adopted, secondly, in order to leverage the wealth of available open-source development tools and libraries enjoyed by the Linux community, again, while guaranteeing that the code runs on the Windows platform.

Since various components of the system were written in different programming languages, an integrated development tool that could support this with minimal overhead was needed. The open-source Eclipse SDK 3.2 was selected. In addition to providing pluggable modules for many languages, Eclipse simplifies development with visual aids and code-refactory tools that are invaluable for efficient development.

Several independent libraries were used to perform tasks that varied from loading image files and handling database connectivity to managing memory allocation and processing HTTP requests. The Developer’s Image Library (DevIL, formerly OpenIL) [13] was used for image file loading, manipulation, and saving. PostgreSQL’s libpq library [14] was used to handle the database connectivity. The Apache Portable Runtime Libraries (libapr and libaprutil) [15] provided platform independent implementations of memory management functions. Libapreq [15], another Apache Project library, provided a reliable means of managing HTTP request data.

The availability of Cygwin packages containing some of these libraries made them very easy to use, but some needed to be built from source. All of the libraries listed above, were developed with portability in mind, so they supported the use of “make” and the GNU Autotools: autoconf, automake, and libtool [16]. The accessibility of these packages through Cygwin’s setup program made building and debugging the libraries relatively simple.

The environment and tools chosen were a great aide in the implementation process. Many of the programs that were needed, and most of the libraries, are available as Cygwin packages which are easy to install and have been thoroughly tested.

2. Data Storage using PostgreSQL

As previously mentioned, the Data Storage Component must provide a structured data retention mechanism that keeps data retrieval simple. Since relational database management systems are designed for this purpose, one was chosen to act as the storage component.

PostgreSQL is an open-source object-relational DBMS [14]. It is known for its platform independence, reliability, scalability, and compliance with existing standards. It was selected for these reasons and because of its ease of integration into the Cygwin environment.

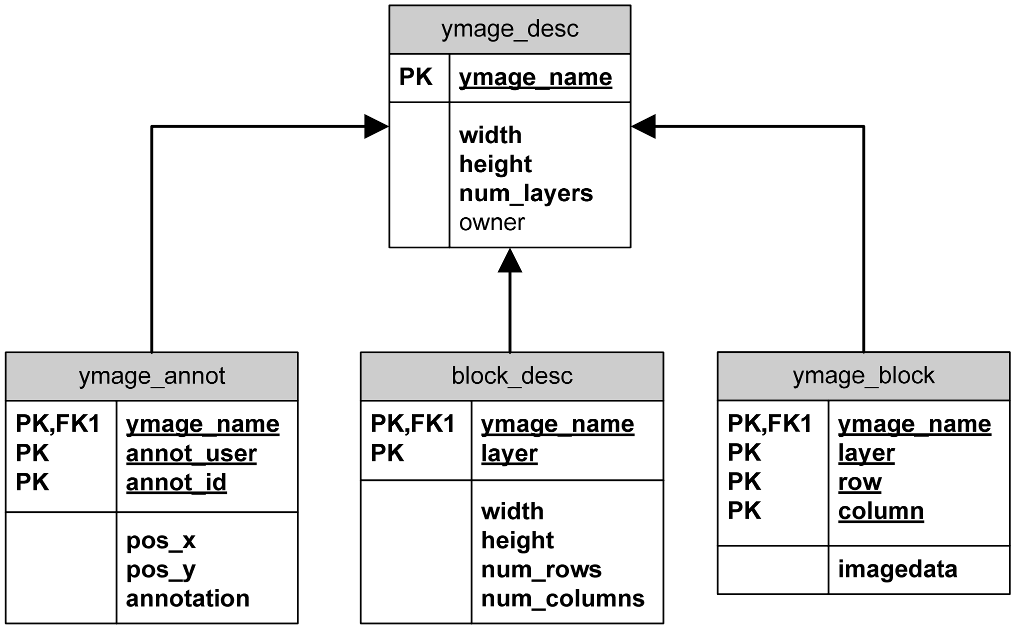


FIGURE 12 – Data Storage Schema

A simple schema was designed to describe the structure and types of data that would be held by the storage system. The schema consists of four tables and some stored procedures. The conceptual model behind the schema's design focuses on the idea of an "ymage." As mentioned before, this is an abstraction which consists of data that describes an image.

Ymages in the data storage schema consist of several parts. The first is a description found in the ymage_desc table. This table has one entry per ymage and contains the name of the ymage, its owner, the number of layers it contains, and the width and height of the original image. The name is a unique identifier and primary key, so the ymage_name field of every entry must be different.

Every ymage has one or more layers. A layer is an entity that corresponds to a scaled version of the original image. Layer-N is a copy of the original image with a size that is $(½)^N$ times that of the original. Layers are not explicitly described in the schema, but are nonetheless referenced for identification in the block_desc and ymage_block tables.

Blocks are sub-images that, when arranged in the proper order, fully represent the visual aspect of the original image. Each individual block is part of a layer and is uniquely identifiable by combining the ymage's name, layer, row, and column. The properties of any particular block are a function of the ymage and layer to which the block belongs. The width and height of blocks belonging to a layer are the same, but may be different from those of other layers. The bounds of a block’s position (relative to other blocks) are also dependent on the layer to which the block belongs. The block_desc table holds information about each layer’s blocks: the dimensions of the graphical block data and size of the layer grid. The ymage_block table has the graphical data for each block in the “imagedata” field, the other fields identify the block.

The remaining table, ymage_annot, holds annotation data. The location of the annotation (pos_x, pos_y) is given with respect to the dimensions in the ymage_desc table. The “user” field identifies the category or person who is associated with the annotation. The “annotation” field contains the information of interest. The remaining fields are referential and provide a means of linking the data to its respective ymage while ensuring the uniqueness of each record.

Functions are included to ensure that changes in the table schema have a minimal impact on the Data Access Component. These functions are primarily for inserting into, and removing records from, the tables. The one exception is a utility function for determining the next identification number for an insertion into the ymage_annot table.

3. Apache HTTP for Data Access

As with data storage, the Data Access Component requires a reliable, accessible, yet flexible implementation. Efficient file distribution capability is very important since this component will deal mostly with files. As possible implementation strategies were evaluated, it became apparent that the desired traits were already inherent in a class of software: web servers.

If a web server was to be used, however, it had to be customizable for the somewhat unorthodox purpose that it would serve. The Apache HTTP server appeared to be the suitable candidate. Its modularity and configurability meant that components which would be unnecessary or obstructive for the purposes of this component could be removed or replaced with those better suited to the needs of Data Access. Its open-source and platform independent nature suggested compliance with the development tools already selected. So the Apache web server (version 2.2) was chosen as the backbone on which the Data Access Component would be built.

None of the initially envisioned modifications to the server were required for this implementation. Instead, a module was developed to run as part of the default configuration. The architecture of the Apache server is rather complicated, but in short it consists of a minimalist core which loads functionality onto itself dynamically at runtime [15]. This functionality takes the form of modules and can perform any tasks related to request processing. Some of those loaded by default handle a wide range of web server needs, from enforcing security to determining how to respond to requests for certain types of files. To convert Apache from a web server to the Data Access Component of the Ymage System, a custom module was written named “mod_ymage.”

The module was developed with a command/parameter structure in mind. File requests sent to the server are intercepted and extensions are checked against the string “ymg”. If the extension of the requested file matches “ymg”, mod_ymage takes control of the request and makes an attempt to service it, otherwise it signals to Apache that it does not wish to handle it and passes the request along to another module. In order to service the request, mod_ymage first has to determine what the request is asking; it does this by extracting, parsing, and analyzing the query string. It looks for a value with the key named “cmd”. If cmd is found, its value signifies the course of action to take; if not found, mod_ymage responds to the request with a message explaining the problem. There are four “cmd” values (or simply commands) to which mod_ymage responds; these are UPLOAD, DOWNLOAD, GET, and ANNOTATE. They correspond to functions that can be applied to ymage data.

UPLOAD is interpreted as a desire to send an image file to the server for processing and submission to the Data Storage system. When this command is given, mod_ymage expects to find an accompanying image file that it can upload to a temporary location on the system’s local storage. If the file is not submitted as part of an HTTP POST request, mod_ymage rejects the command and returns a message indicating that the upload was not consistent with its expectations.

When a file is uploaded, it is processed with the goal of dividing the image into layers and blocks then ultimately constructing an ymage for storage. The process involves several steps.

TABLE I

IMAGE UPLOAD PROCESS

Step 1: Analysis	Image blocks must all be of the same dimension as dictated by the schema,

	so the image is evaluated and a simple algorithm is used to determine the best dimensions to use, given a target. The algorithm is described in Appendix A. This analysis is done on a per-layer basis since the dimensions of each layer are different.
Step 2: Block Extraction	With the block dimensions known, the original image is traversed and independent blocks are extracted and saved locally. This is done for each layer. Regardless of the original image file’s format, the extracted blocks are saved as JPEGs with parameters set for high compression. Metadata is collected during this step as well.
Step 3: Data Storage	With all of the required data retrieved, an ymage can be constructed. An entry is placed in the ymage_desc table as a reference to the new ymage; the ymage_name field is populated with the file request (i.e. the string preceding “.ymg” in the HTTP request). All image blocks and metadata are then sent to the database server.

A message indicating completion is returned upon success of all three steps. An appropriate failure message is returned otherwise.

Image blocks that reside in the Data Storage Component are retrieved with the DOWNLOAD command. Parameters are passed in order to identify the desired block; these parameters are “layer”, “row”, and “column”. Their values correspond to those in the ymage_block table and are submitted in the query string as key-value pairs. Mod_ymage queries the data in storage with these parameters and if the specified image block is found, a file representing the block is returned in response. A message indicating failure to find the data is returned otherwise.

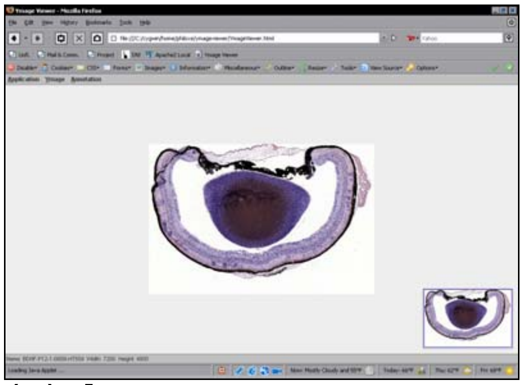
Non-graphical data is obtained using the GET command. The “item” parameter must be specified with this command; it takes one of three values: YMAGELIST, METADATA, and ANNOTATION. Specifying YMAGELIST causes a comma-delimited list of ymage names from the ymage_desc table to be returned. Use of METADATA indicates a request for information from either the ymage_desc, block_desc, or ymage_annot tables. To select which of these three, the “schema” parameter must be assigned a value. When “schema” is has the value “YMAGE”, the “width”, “height”, and “num_layers” values from the ymage_desc table are returned in semicolon-delimited list of key-value pairs. When “schema” is assigned the value “BLOCK”, “width”, “height”, “num_rows”, and “num_columns” are returned from the block_desc table in the same format. Assigning “schema” the value of “ANNOTATION_USER” returns a comma-delimited list of unique annot_user values. Assigning “schema” the value “ANNOTATION” retrieves the values from the “annotation” field of the ymage_annot table for a given user. This “schema” parameter must be accompanied by the “user” parameter to indicate which data set is desired. The response returned is a newline-delimited list of annotation records. Each record contains the location of the annotation as “x” and “y” key-value pairs separated by “||”. The content of the “annotation” field of the ymage_annot table is the last item in the returned string.

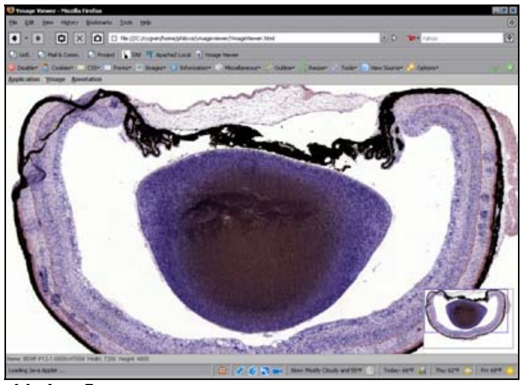
The last command, ANNOTATE, is used to request the placement of annotation data into the ymage_annot table. This command requires the inclusion of additional parameters, “x”, “y”, and “user”, which correspond to the pos_x, pos_y, and annot_user fields in the ymage_annot table, respectively. The command must be sent as an HTTP POST with the annotation data in the body of the request under the name “annotation”. Correct specification of parameters results in the placement of the data onto the database and a message indicating success. Failure to include any of the required parameters for this and all other commands causes mod_ymage to return a message indicating the missing parameter.

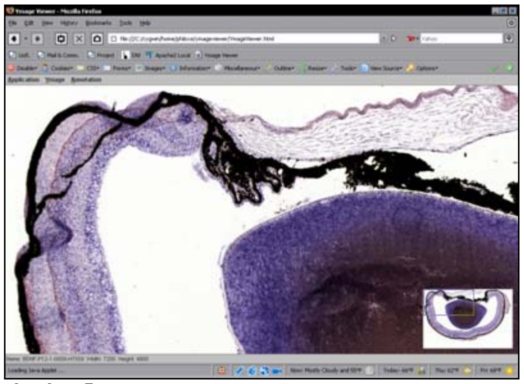
The Data Access Component's simplicity and use of the HTTP protocol makes it possible for different clients to be developed that use it. A web browser, for example, can be used to request data from the system and in fact, much of the system's debugging was done in this way before a viewing program was implemented. Future clients can be developed with new technologies and without the need to modify the Data Access Component; these clients only require the ability to make HTTP requests and interpret the results.

4. Java Applet Viewer

Many different approaches can be taken with respect to the Visualization Component. Initially, a Mozilla Firefox plug-in was developed for the purpose. It fulfilled the basic needs, but when experimentation with more advanced functionality was desired, it became more practical to use Java. The applet that was implemented is capable of using all of the features of the Data Access Component and behaves like a black box, hiding the image request details behind a graphical user interface.


(a) layer 4


(b) layer 3


(c) layer 2

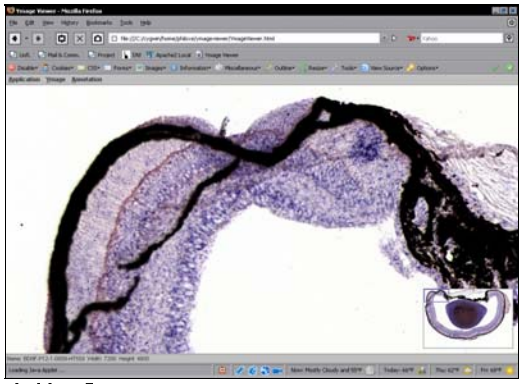

(d) layer 1

FIGURE 13 – Java Applet Visualization Component
Zooming in from the layer with the least detail (a) up through layers of greater detail.

The applet sends HTTP requests to the Apache Data Access Component and updates widgets with the data that it receives. To view an image, the URL of the a server running mod_ymage is entered and a list of available ymages is requested. The returned list is converted into a set of menu options from which an ymage name can be selected. Clicking on one of these menu items initiates the download and display of graphical data. To maximize the efficiency of viewing the ymage, display begins at the highest layer, i.e. the one which consists of a single image block. As the user zooms into the image, the applet determines whether to move to the next layer and initiate download requests for new image blocks or to resize the blocks that it already has. Figure 13 shows the applet displaying an image as the user starts from the layer with least detail (a) that the layer of most detail (d). When a layer is reached that is composed of more than one image block, the applet identifies which blocks are within the view port and downloads for display. Dragging the view port changes the visible region and image blocks which have not been downloaded are requested for subsequent display.

Upload features are also available in this applet. An image file can be located on the user’s local file system and previewed before upload. The preview image is a sub-sampled version of the original obtained using Java’s Image I/O API. While previewing, a rectangular region of the image can be selected for upload, should the whole image not be wanted.

The applet is independent of any particular Data Access server, so it can be used to view ymages from any server that runs an Ymage System Data Access Component. This flexibility can be attributed to the modularity of the overall system and maximizes data accessibility.

C. Evaluation and Possible Improvements

The implementation presented here is rather rudimentary. It is a model of how the system should generally work. As such, it does not fully exploit the strengths of the design. Improvements can be made to each subsystem in order for real performance gains to be realized.

Improvements can be made with respect to the Data Storage Component. Since PostgreSQL was available as a Cygwin binary package and other well known contenders such as MySQL were not, the former was selected partly out of convenience. Past studies [17] and current discussion have credited MySQL with being a faster relational database system and so an improvement in performance may be achieved in electing to use it. The schema used for this implementation would require only minor changes to ensure compliance with MySQL. As the performance degradation of this component becomes less of an issue, the schema can be expanded to more properly address security and data ownership concerns, system protocol enforcement, and inclusion of a greater assortment of data.

The Data Access Component is critical to the system’s overall performance since all data and requests are channeled through it. Improvements can be made to this component by exploiting the features and tweaking the operation of the Apache web server. Eliminating all modules that service demands inconsistent with the needs of the system and the hardware on which it is running would be likely to reduce overhead. Some of the modules that are loaded by default handle directory authorization, executing CGI scripts, and web authoring. These are not necessary for the Data Access to function. Along with removing such modules, a reduction in the number of database transactions by caching highly demanded data would also boost performance. The current implementation of this subsystem really consists of a web server with the ability to carry out functions consistent with the Ymage System. The ultimate goal is to develop this component enough that it becomes a subsystem wholly dedicated to the servicing of the overall systems demands.

Improvements of the Visualization Component could be made by adopting the modularity that is apparent on the overall system level. Graphical display should be the sole purpose of the Visualization Component; in this way, the focus can be maintained on image quality. Optimizations could come in the form of using technologies which are better suited to fast, high-quality distributed visualization. Image uploads should be implemented in a client branch of the Data Access Component and eliminate the need to integrate visual demands with file transfer logistics.

In addition to modifying the components, it is possible to improve performance by taking different approaches algorithmically; this is especially true for storage. The JPEG compression scheme used for saving image blocks reduces storage size considerably for any given layer, but still more space savings can be realized. The true difference between any two layers is the amount of detail present, all other information is redundant. Storage size can be reduced by saving a low-detail version of the original image and the differences (errors) between consecutive layers; this is the same as generating a Laplacian Pyramid [18]. The scaled images from which the layers are presently derived would serve as target images used to determine error values which could be saved in the place of the image blocks. Retrieving the image for a given layer would require the low-detail image to be super-sampled up to the size of the desired layer and adding error values for all layers between the super-sampled and the desired one. Since the exact error is known, the original image at the given layer can be obtained precisely. The errors at each layer can also be compressed to further reduce storage space. The drawback with this approach is the demand placed on the Visualization Component which must now process the image before displaying it. However, even this cost is likely to be low since less data will need to be requested for any given layer and, once obtained, can be reused for the display of subsequent layers.

For the practical purposes to which a system built from this architecture will be put, these changes would substantially improve usability. As a model, the present implementation demonstrates the possibilities of the design and is a respectable starting place for development of more rigorous software.

IV. RESULTS

Comparisons of the Ymage System (as implemented in this project) show that it is a more practical option most of the time than other commonly used digital image viewing software.

TABLE II

IMAGES TESTED

Ref #	Image Name	Size	Width	Height
1	P0-SC-QGFR146-Grina.tif	27.4 MB	4000 px	2400 px
2	P0-SC-QGFR142-Grm1.tif	30.1 MB	3900 px	2700 px
3	BDM-P14-1-0009-HT559.tif	76.9 MB	4800 px	5600 px
4	BDNF-P12-1-0009-HT559.tif	98.9 MB	7200 px	4800 px
5	wt-E13-5-Brain-QGF033-HT551.tif	251 MB	15400 px	17100 px

See Appendix B for images

Images from brain and retinal scans were used to test desktop applications, Zoomifyer EZ™, and the Ymage System. Table II lists the images and their properties. One important metric in determining how useful a program is for displaying an image is the time that it requires to do so. Load times are the time from when the program begins displaying a region to when all of the data for that region have been loaded and are visible. The load times of the desktop applications are compared in Table III. Though Picasa™ out-performed the others for images ranging from 27 to 99 megabytes, it failed to load the 251 megabyte image at its highest resolution within the practical time limit of 10 minutes. The other programs loaded the images within a reasonable time, but this was at the price of a significant performance reduction to all programs running on the computer[*].

TABLE III

DESKTOP APPLICATION LOAD TIMES

Image	Adobe Photoshop® CS2	GIMP2	Google Picasa2™
1	2 sec	3 sec	<1 sec
2	2 sec	3 sec	1 sec
3	6 sec	18 sec	2 sec
4	2 sec	33 sec	3 sec
5	21 sec	4 min 54 sec	>10 min

Load times of the Ymage System were comparable, if not better than those of the desktop programs. However, improvements in load time were at the cost of segmentation and upload overhead (see Table IV). Although the combined segmentation and upload times for the Ymage System are significantly longer than desktop application load times,

[*] Tests were conducted on a Dual AMD Athlon™ MP 2200+ (1.86 GHz processors) machine with 1GByte of PC2100 DDR RAM running Windows XP Professional with Service Pack 2.

this cost must only be endured once and compensation is made in storage space savings.

TABLE IV

YMAGE SYSTEM TEST RESULTS

Image	Segmentation Time	Seg + Upload	Post-Seg Size	Num of Layers	Num of Blocks	Load Time
1	5 sec	25 sec	2.17 MB	4	129	2 sec
2	6 sec	26 sec	2.78 MB	4	135	1 sec
3	16 sec	49 sec	4.39 MB	5	590	2 sec
4	31 sec	1 min 32 sec	7.20 MB	5	366	1 sec
5	6min 59sec	13 min 54 sec	64.6 MB	7	5190	1 sec

Zoomifyer EZ™ outperformed this implementation of the Ymage System with apparent load times of under one second for all images tested (see Table V). Greater storage savings were also realized with Zoomifyer EZ™ than with the Ymage System. Upload time to the web server for the former could not be accurately measured as there were several different, yet equally feasible, methods for completing the task; these times could be considered proportional to the size of the original image files.

TABLE V

ZOOMIFYER EZ™ TEST RESULTS

Image	Segmentation Time	Post-Seg Size	Load Time
1	12 sec	1.86 MB	<1 sec
2	10 sec	2.37 MB	<1 sec
3	23 sec	3.63 MB	<1 sec
4	32 sec	6.06 MB	<1 sec
5	4min 58sec	51.2 MB	<1 sec

V. CONCLUSION

The storage, retrieval, and visualization problems associated with the practical use of large, high resolution digital images can be resolved using a flexible, modular system design. Furthermore, there is sufficient open source-code to meet the requirements without the need of commercial software.

Conventional methods employed by the standalone applications proved to be inadequate for several reasons. First, the load times increased in proportion to image size. For images that are very large, the delay between the time an image was requested and the time it became visible was unreasonably long. And, second, the demand on system resources, in the form of storage space, system memory, and CPU use, rendered the viewing process and the computer unworkable. The time issue cannot be completely avoided; a great deal of data requires time to be processed. In the standalone applications this processing delay was observed every time the image was requested. With the Ymage System and Zoomifyer EZ™, there was a greater processing time realized, but it was only manifested once per image. During this time machine performance was affected to some degree; however, subsequent requests for the image were almost instantaneous regardless of image size. The reduction of required storage space also favored the Ymage System and Zoomifyer EZ™ over the client applications. It is apparent that the way in which these two approach image storage and retrieval are an improvement to the approaches taken by the standalone programs.

The Ymage System performed comparably to Zoomifyer EZ™. This was surprising since the implementation of the former was not completed with much focus on performance optimization. Making use of the operational exploits described in the “Evaluation and Possible Improvements” section could potentially improve the Ymage System enough to ultimately equal Zoomifyer EZ™ in performance. The difference between the two systems would then be in data storage where the organization and maintainability of data in the Ymage System is an improvement over the other system.

The modularity of the Ymage System affords itself well to other applications. Since data retrieval in the system is independent of the Visualization Component, the system is extensible for non-visual functions. One interesting use would be for distributed image processing.

Blocks could be obtained from different parts of an image by different machines and processed independently; the results could be combined later outside of the system or (with some modification to the Data Access Component) placed in Data Storage Component. The implications of organizing and providing access to image data as presented by the Ymage System has the potential to change the way in which images are viewed and manipulated. Novel approaches to image application design are sure to come from the options available through the architecture of this system.

VI. REFERENCES

[1] Jan Teuber, *Digital Image Processing*. New York, NY: Prentice Hall International (UK) Ltd., 1993.

[2] C.Wayne Brown and Barry J.Shepherd, *Graphics File Formats*. Greenwich, CT: Manning Publications Co., 1995.

[3] Michael Miller, "Using Google Book Search," in *Googlepedia: The Ultimate Google Resource* Que, 2006, pp. 475-479.

[4] Paul Suetens, *Fundaments of Medical Imaging*. Cambridge, UK: Cambridge University Press, 2002.

[5] *Image Databases: Search and Retrieval of Digital Imagery*. New York: John Wiley & Sons, Inc., 2002.

[6] Michael Miller, "Using Picasa," in *Googlepedia: The Ultimate Google Resource* Que, 2006, pp. 585-601.

[7] Armin Hanisch, "ImageQuery 1.4.4," Webpage. [Online]. Available: http://www.arminhanisch.de/software/imagequery_en.html

[8] PrimaSoft, "Photo, Picture Organizer Deluxe, v2.7," Webpage. [Online]. Available: http://www.primasoft.com/deluxeprg/photo_organizer_deluxe.htm

[9] Randolph Kim, "Learning Technologies, World Wind 1.4," Webpage. [Online]. Available: http://worldwind.arc.nasa.gov/

[10] Microsoft, "TerraServer-USA," Webpage. [Online]. Available: http://terraserver.microsoft.com/

[11] Zoomify Inc., "Zoomable web images!," Webpage. [Online]. Available: http://zoomify.com/

[12] Ed Bradford and Lou Mauget, *Linux and Windows Interoperability Guide* Prentice Hall, 2001, pp. 109-120.

[13] Meloni Dario, "DevIL: A full featured cross-platform Image Library," Webpage. [Online]. Available: http://openil.sourceforge.net/

[14] Jeff Perkins, *Postgresql* Muska & Lipman/Premier-Trade, 2007.

[15] "Apache HTTP Server Project," Webpage. [Online]. Available: http://httpd.apache.org/

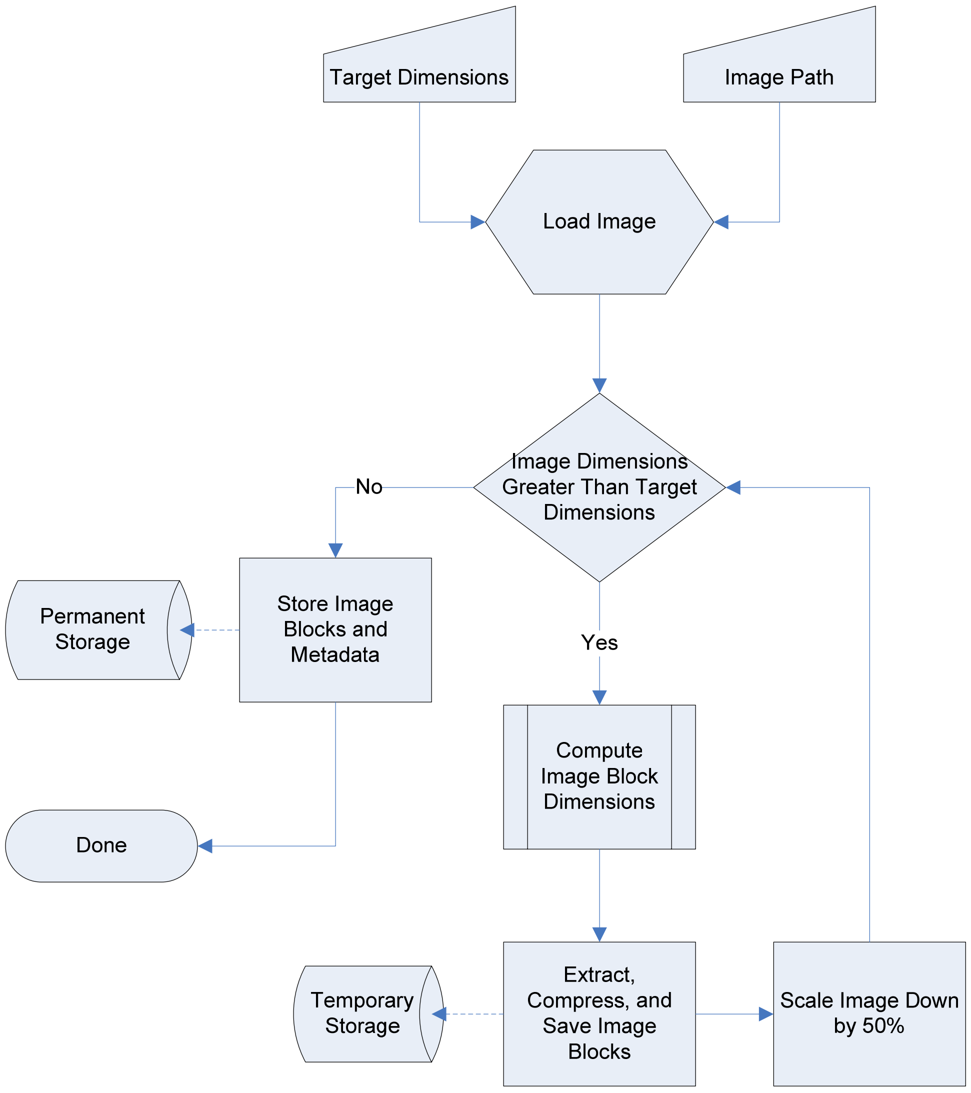
[16] Gary V.Vaughan, Ben Elliston, Tom Tromey, and Ian Lance Taylor, *GNU Autoconf, Automake, and Libtool*. Indianapolis, IN: New Riders Publishing, 2001.

[17] Fermi National Accelerator Laboratory, Computing Division, "PostgreSQL versus MySQL," Webpage. [Online]. Available: http://www-css.fnal.gov/dsg/external/freeware/pgsql-vs-mysql.html

[18] Peter J.Burt and Edward H.Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Transactions on Communications*, vol. 34, no. 4, pp. 532-540, Apr.1983.

APPENDIX A

A.1 Image Subdivision Process


Target Dimensions
Image Path
Load Image
Image Dimensions Greater Than Target Dimensions
No
Yes
Store Image Blocks and Metadata
Permanent Storage
Done
Compute Image Block Dimensions
Extract, Compress, and Save Image Blocks
Temporary Storage
Scale Image Down by 50%

A.2 Algorithm for Computing Image Block Dimensions

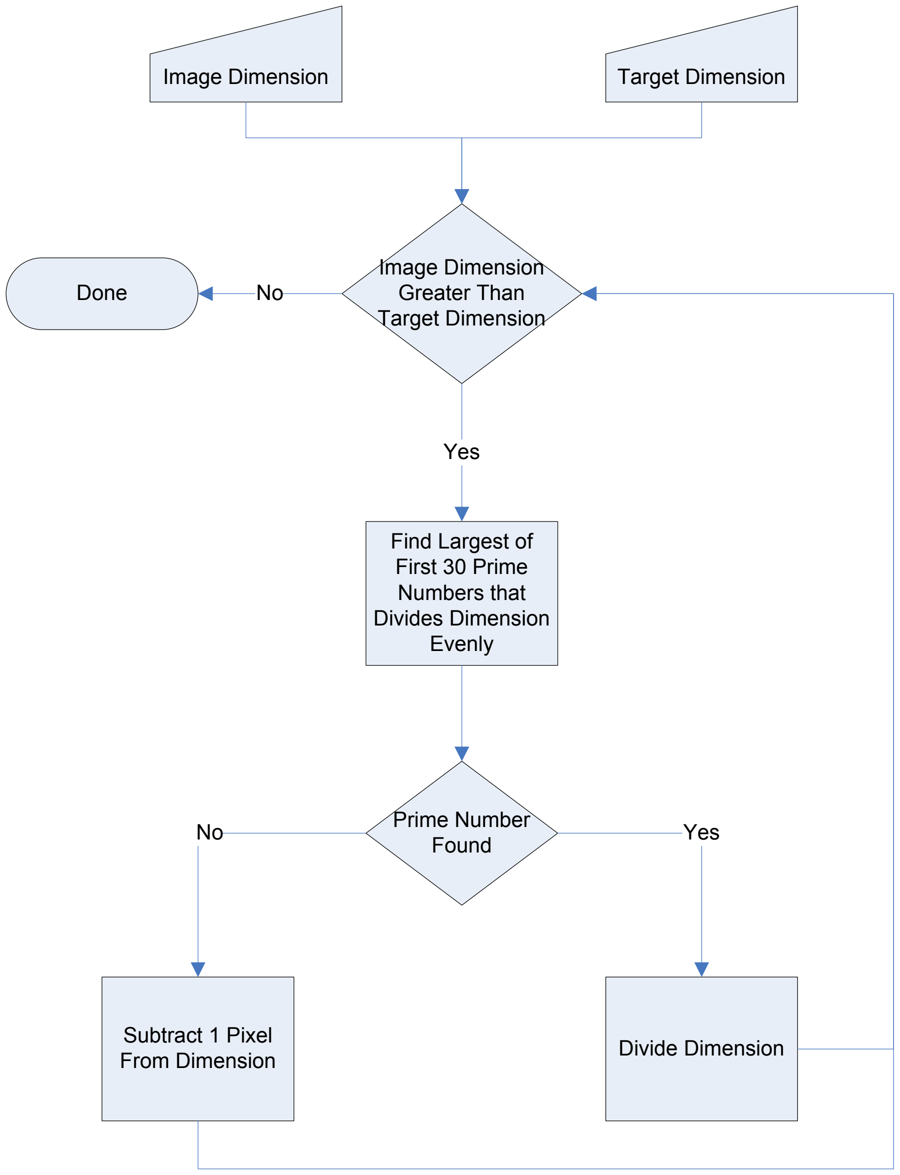
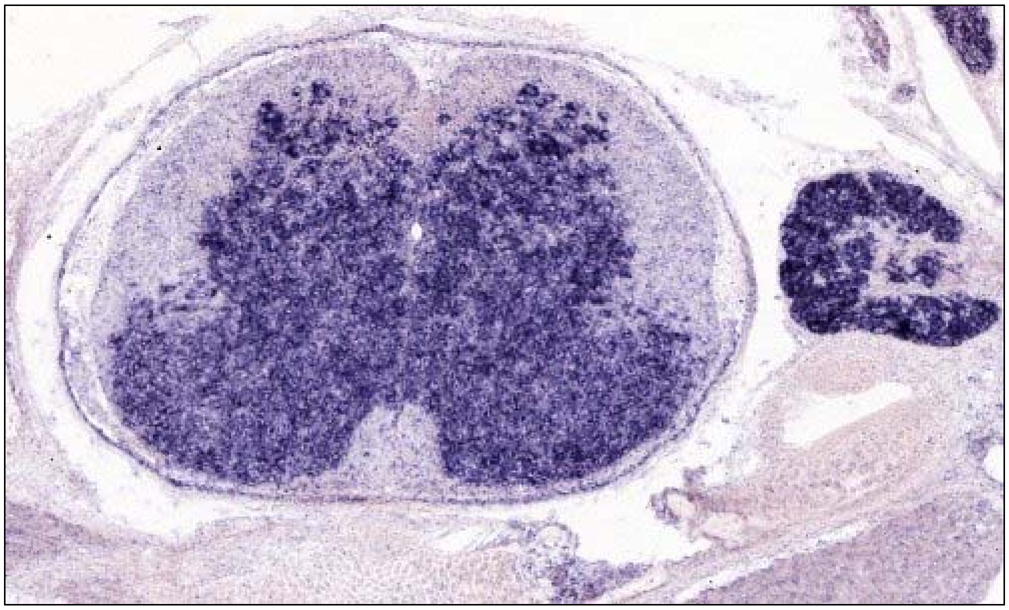


Image Dimension
Target Dimension
Image Dimension Greater Than Target Dimension
No
Done
Yes
Find Largest of First 30 Prime Numbers that Divides Dimension Evenly
Prime Number Found
No
Yes
Subtract 1 Pixel From Dimension
Divide Dimension

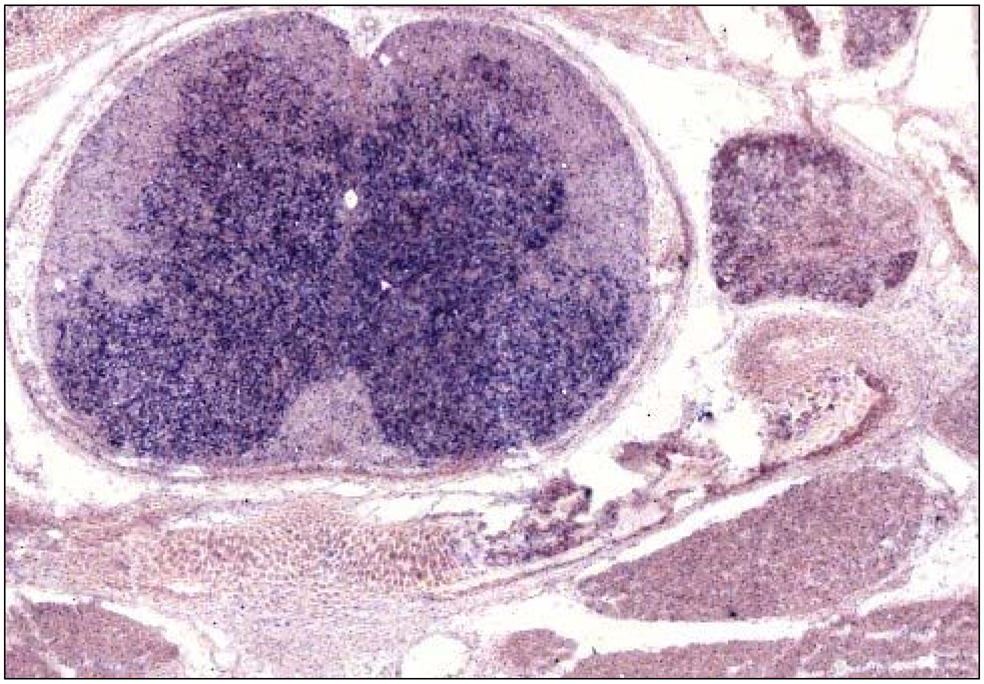
APPENDIX B

Images Tested



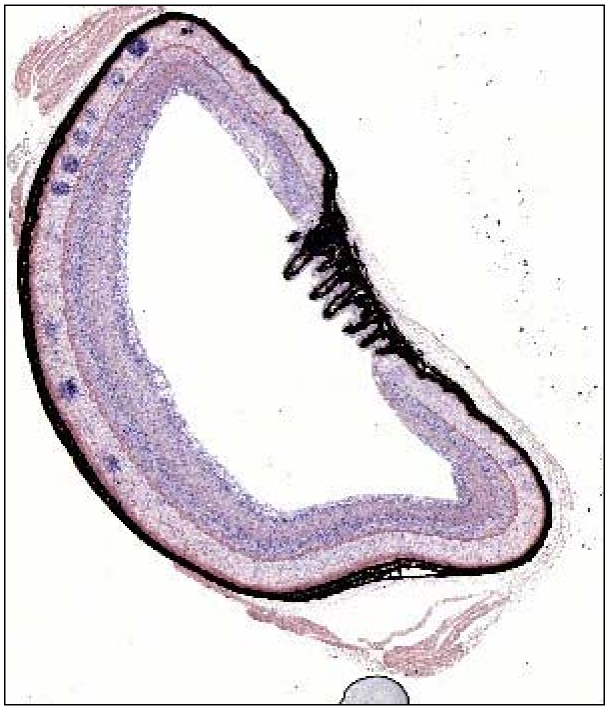
1. P0-SC-QGFR146-Grina.tif

[4000 by 2400 pixels, 27.4MB]



2. P0-SC-QGFR142-Grm1.tif

[3900 by 2700 pixels, 30.1MB]

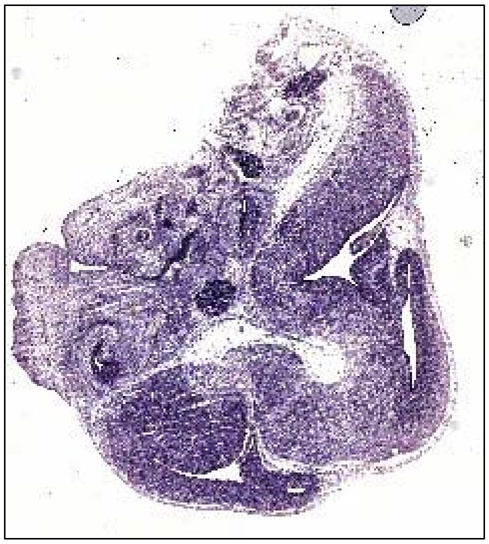


3. BDM-P14-1-0009-HT559.tif

[4800 by 5600 pixels, 76.9MB]

4. BDNF-P12-1-0009-HT559.tif

[7200 by 4800 pixels, 98.9MB]



5. wt-E13-5-Brain-QGF033-HT551.tif

[15400 by 17100 pixels, 251MB]

VITA

Yetu Yachim
4925 Charm Oak Dr.
Jefferson City, MO 65109

Phone: (573) 821-0742
Email: yetu.yachim@louisville.edu
yetu.yachim@gmail.com

EDUCATION

Master of Engineering, Expected: May 2007
Computer Science and Engineering
University of Louisville
GPA: 3.769 / 4.000

Bachelor of Science, December 2006
Computer Science and Engineering
University of Louisville
GPA: 3.783 / 4.000

SIGNIFICANT COURSEWORK

Microcomputer Design
Simulation of Discrete Systems
Design of Computer Algorithms
Computer Graphics

EXPERIENCE

University of Louisville CECS Department
Belknap Campus, Louisville, Kentucky
October 2006 – April 2007
Graduate Thesis Project

Marathon Oil Corporation
Office Building, Findlay, Ohio
May 2005 – August 2005
Web Developer and Mainframe Programmer

Marathon Ashland Petroleum, LLC.
Main Headquarters, Findlay, Ohio
August 2004 – December 2004
Applications Developer

Marathon Ashland Petroleum, LLC.
Texas Refining Division, Texas City, Texas
January 2004 – May 2004
Web Developer and Tier II Support

ORGANIZATIONS

Tau Beta Pi
National Society of Black Engineers (2006 VP)
Society of Porter Scholars
Association of Black Students (Web Developer)

HONORS

Woodford R. Porter, Sr. Scholarship
National Dean’s List (2002 – 2006)
CECS Outstanding Undergraduate Award (Spring 2006)
NSBE Outstanding Member Award (2006)