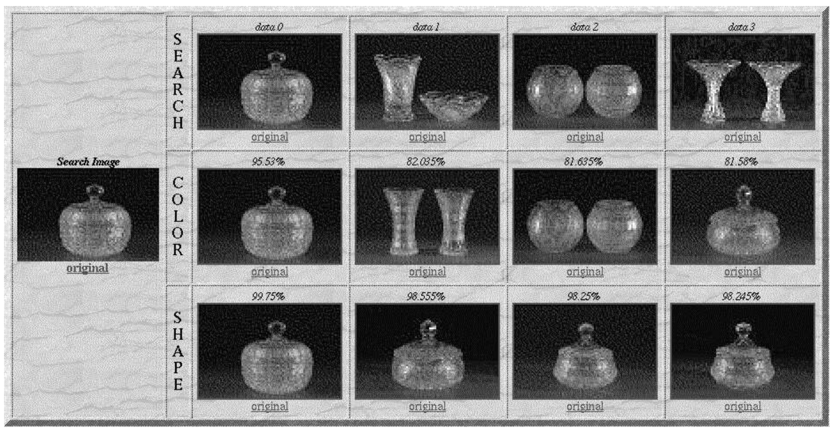
A Content-based Image Search Engine in Cyberspace

a Master's Thesis



Application-Oriented Computer Science
Leiden Imaging and Multimedia Group
Department of Computer Science

Leiden University

Yuri Lausberg

August 19, 1996

Preface

This thesis represents the final part of my computer science studies here at Leiden University. The reason for a thesis is to explore a certain aspect of a field within the computer science education. After a great number of courses I obtained a clear perspective of which field I would like to complete my education in. I ended up with the LIM (Leiden Imaging and Multimedia) group a subgroup of the HPC (High Performance Computing) group of application-oriented computer science with Dr. Nies Huijsmans and Dr. Michael Lew. They gave me the possibility for my thesis work. As I got the greatest interest in the Internet I started working on a project within the LIM (Leiden Imaging and Multimedia) group. This project lies within the Digital Image Databases and Internet.

Therefore I would like to thank Dr. D.P.(Nies) Huijsmans and Dr. Michael Lew for giving me the possibility to do my thesis under their supervision. I also would like to thank them for the care and patience they had for guiding me through the whole project.

As I am Dutch and I am studying here in Holland I chose to do all my thesis work in the English language. The reason for this was simply because one of my associate teachers (Dr. Michael Lew) is an US citizen. Another reason for me to do this thesis in english is because my wife is an US citizen too and after my graduation I am planning to immigrate to the USA.

I would also like to thank my wife Summer for always giving me the love and support I needed. Finally I would like to thank my parents for their patience to finally see me graduate.

Introduction

An important limitation of the information society of today is the dependency of textual information representation. This situation tends to continue despite the main influence of image information representation which is very common to the day to day routine in human society. We use our visual ability to see and understand visual information more than any other medium to communicate and collect information.

The World Wide Web which is actually a part of the Internet contains the four basic elements of hypermedia: text, images, audio and video. A big part of the World Wide Web contains visual information representation, simply because of this old saying: *"a picture says more than a 1000 words"*.

Because of this amount of information selective searching has become more and more important. Nowadays meta-information services are offered also called "search engines" which are based upon text pattern matching methods.

Image searching therefore is most interesting because it is almost not available on the World Wide Web. It would be very nice if you have your own photo-studio to see if your photos already exist elsewhere on the Web. This type of search engines are not (or almost not) available not because nobody is interested but simply because the problem of image understanding by computers has not yet been solved. However a growing number of researchers have been trying to find image query equivalents for the successful text-pattern matching and text-indexing techniques.

To obtain a clear view of the image understanding problem the reader should know that we will need to transform the digital image representation to other forms which have actual meaning or human oriented content. The main idea of achieving this is the concept of content-based retrieval and storage of image based information for digital libraries.

My project is divided into two parts. In the first part I've created a CGI program; a so called "search engine", usable for any WWW-browser on the World Wide Web. This search engine makes querying based on image content possible within the 19th Century Portrait Database. The search algorithms and feature extraction techniques for this content-based image retrieval have been implemented by other persons. The main idea was to see if we could make the results of the search methods available for anyone connected to the World Wide Web. Another motivation was to understand and overcome the problems involving Web programming.

In the second part of this project we tried to turn this concept of connecting this search engine to the World Wide Web around. Instead of connecting our private database to the World Wide Web I have created a database based on the images found on the World Wide Web. With this database a new search engine was born. This concept is rather interesting because public data is involved (not just some private collection of photos).

In order to collect these images we had to come up with a program, this is what we like to call a Web spider which is nothing more than an autonomous agent that searches the World Wide Web. Of course like any other database you would like to perform queries. This querying is based on image content, like *color* and *shape*, (but also text-pattern querying) of an example image. All by all I ended up with a search engine based on querying by example on image content and text-patterns.

In this report we would like to give the reader a detailed outline of the way we've been working on this project, the techniques involved and most of all the programming itself. It will also give a nice introduction to Web programming and all the basics involved. This report can be used for the experienced programmer as a reference manual for any future work for this project.

Contents

Chapter 1

The World Wide Web

1.1 Introduction

In the 1960s, researchers began experimenting with linking computers to each other and to people through telephone hook-ups, using funds from the U.S Defense Department's Advanced Research Projects Agency (ARPA).

ARPA wanted to see if computers in different locations could be linked using a new technology known as packet switching, which had the promise of letting several users share just one communications line. Previous computer networking efforts had required a line between each computer on the network, sort of like a train track on which only one train can travel at a time. The packet system allowed for creation of a data highway, in which large numbers of vehicles could essentially share the same lane. Each packet was given the computer equivalent of a map and a time stamp, so that it could be sent to the right destination, where it would then be reassembled into a message the computer or a human could use.

This system allowed computers to share data and the researchers to exchange electronic mail, or e-mail. In itself, e-mail was something of a revolution, offering the ability to send detailed letters at the speed of a phone call.

As this system, known as ARPANet, grew, some enterprising college students (and one in high school) developed a way to use it to conduct online conferences. These started as science-oriented discussions, but they soon branched out into virtually every other field, as people realized the power of being able to "talk" to hundreds, or even thousands, of people around the country.

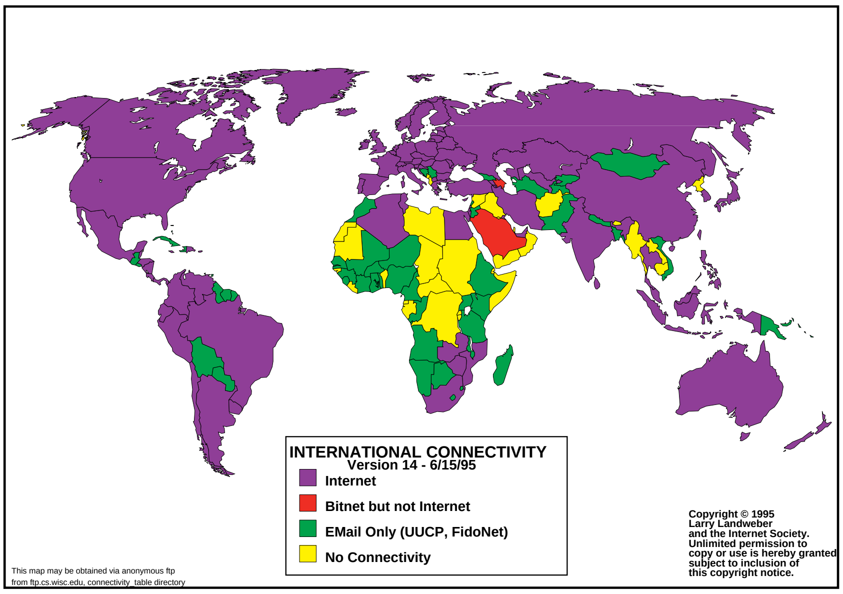

INTERNATIONAL CONNECTIVITY
Version 14 - 6/15/95
🟪 Internet
🟥 Bitnet but not Internet
🟩 EMail Only (UUCP, FidoNet)
🟨 No Connectivity
Copyright © 1995
Larry Landweber
and the Internet Society.
Unlimited permission to
copy or use is hereby granted
subject to inclusion of
this copyright notice.
This map may be obtained via anonymous ftp
from ftp.cs.wisc.edu, connectivity_table directory

Figure 1.1: Statistics available by FTP from nic.merit.edu.

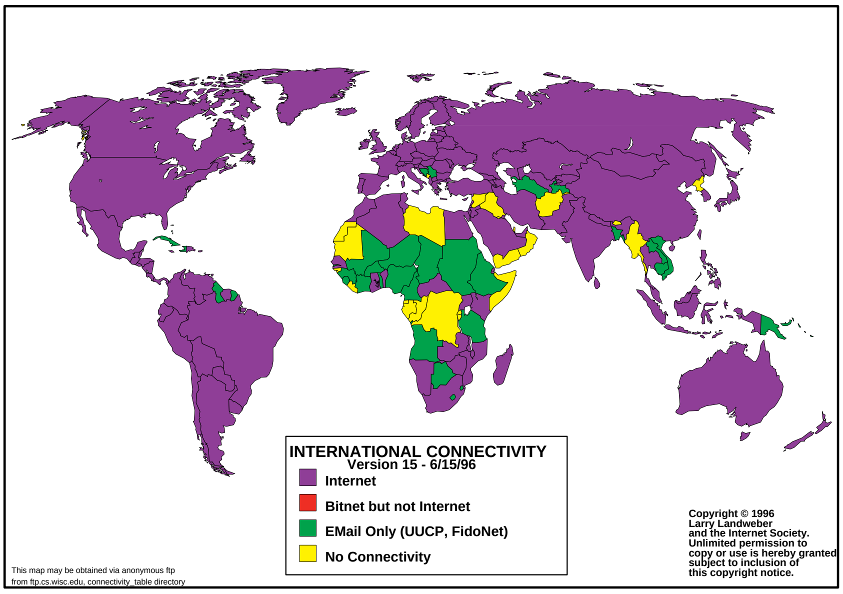

INTERNATIONAL CONNECTIVITY
Version 15 - 6/15/96
🟪 Internet
🟥 Bitnet but not Internet
🟩 EMail Only (UUCP, FidoNet)
🟨 No Connectivity
Copyright © 1996
Larry Landweber
and the Internet Society.
Unlimited permission to
copy or use is hereby granted
subject to inclusion of
this copyright notice.
This map may be obtained via anonymous ftp
from ftp.cs.wisc.edu, connectivity_table directory

Figure 1.2: Statistics available by FTP from nic.merit.edu.

In the 1970s, ARPA helped support the development of rules, or protocols, for transferring data between different types of computer networks. These "internet" (from "internetworking") protocols made it possible to develop the worldwide Net we have today. By the close of the 1970s, links developed between ARPANet and counterparts in other countries. The world was now tied together in a computer web.

In the 1980s, this network of networks, which became known collectively as the Internet, expanded at a phenomenal rate. Hundreds, then thousands, of colleges, research companies and government agencies began to connect their computers to this worldwide Net (see figure 1.1 and 1.2). Some enterprising hobbyists and companies unwilling to pay the high costs of Internet access (or unable to meet stringent government regulations for access) learned how to link their own systems to the Internet, even if only for e-mail and conferences. Some of these systems began offering access to the public. Now anybody with a computer and modem - and persistence - could tap into the world.

In the 1990s, the Net grows at exponential rates. Some estimates are that the volume of messages transferred through the Net grows 20 percent a month (see figure 1.1 and 1.2). In response, government and other users have tried in recent years to expand the Net itself.

1.2 How it works

The worldwide Net is actually a complex web of smaller regional networks. To understand it, picture a modern road network of trans-continental superhighways connecting large cities. From these large cities come smaller freeways and parkways to link together small towns, whose residents travel on slower, narrow residential ways.

The Net superhighway is the high-speed Internet. Connected to this are computers that uses a particular system of transferring data at high speeds. In the U.S., the major Internet "backbone" theoretically can move data at rates of 45 million bits per second (compare this to the average home modem, which has a top speed of roughly 28k8 bits per second). This internetworking "protocol" lets network users connect to computers around the world.

Connected to the backbone computers are smaller networks serving particular geographic regions, which generally move data at speeds around 1.5 million bits per second. Feeding off these in turn are even smaller networks or individual computers.

Nobody really knows how many computers and networks actually make up this Net. Some estimates say there are now as many as 5,000 networks connecting nearly 2 million computers and more than 15 million people around the world. Whatever the actual numbers, however, it is clear they are only increasing.

There is no one central computer or even group of computers running the Internet - its resources are to be found among thousands of individual computers. This is both its greatest strength and its greatest weakness. The approach means it is unlikely for the entire Net to crash at once - even if one computer shuts down, the rest of the network stays up. But thousands of connected computers can also make it difficult to navigate the Net and find what you want. It is only recently that Net users have begun to develop the sorts of navigational tools and "maps" that will let neophytes get around without getting lost.

The vast number of computers and links between them ensure that the network as a whole will likely never crash and means that network users have ready access to vast amounts of information. But because resources are split among so many different sites, finding that information can prove to be a difficult task, especially because each computer might have its own unique set of commands for bringing up that information.

If you choose to go forward, to use and contribute, you will become a "citizen of Cyberspace." If you're reading these words for the first time, this may seem like an amusing but unlikely notion – that one could "inhabit" a place without physical space. But put a mark beside these words. Join the Net and actively participate for a year. Then re-read this passage. It will no longer seem so strange to be a "citizen of Cyberspace". It will seem like the most natural thing in the world.

So how do we define the World Wide Web exactly? The World Wide Web is officially described as a "wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents". The WWW (World Wide Web) is basically a part of the Internet. When we speak of hypermedia, we're talking about the four basic document components:

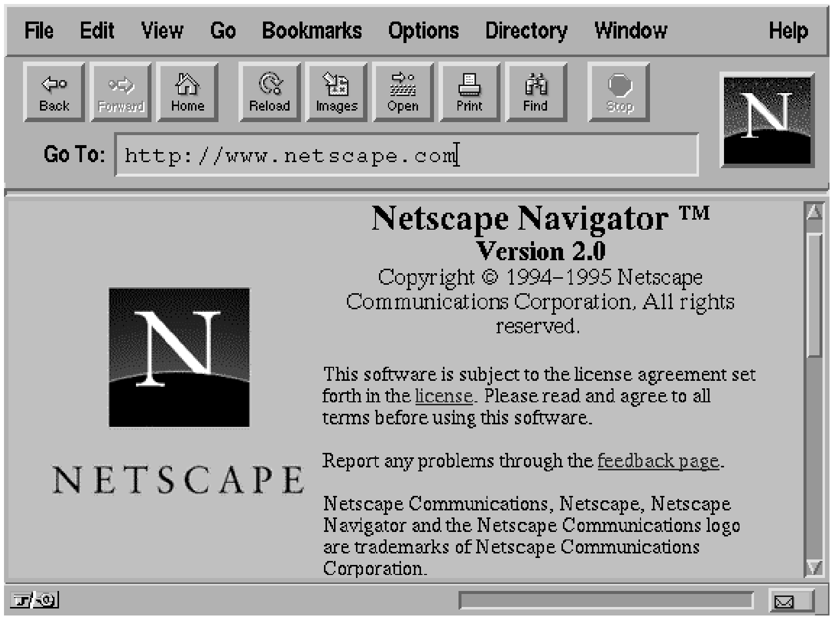
1. text
2. images
3. audio
4. video

The WWW supports all off these one way or another.

The World Wide Web exists virtually - there is no standard way of viewing it or navigating around it. However, many software interfaces to the Web have similar functions and generally work the same way no matter what computer or type of display is used.

1.3 Web client

Web software is designed around a distributed *client-server architecture*. A *Web client* (called a *Web browser* if it is intended for interactive use) is a program which can send requests for documents to any Web server. A *Web server* is a program that, upon receipt of a request, sends the document requested (or an error message if appropriate) back to the requesting client.



Netscape Navigator ™
Version 2.0
Copyright © 1994–1995 Netscape Communications Corporation, All rights reserved.

This software is subject to the license agreement set forth in the license. Please read and agree to all terms before using this software.

Report any problems through the feedback page.

Netscape Communications, Netscape, Netscape Navigator and the Netscape Communications logo are trademarks of Netscape Communications Corporation.

Figure 1.3: Netscape Web browser

The language that Web clients and servers use to communicate with each other is called the *HyperText Transfer Protocol (HTTP)*. All Web clients and servers must be able to speak HTTP in order to send and receive hypermedia documents. For this reason, Web servers are often called HTTP servers.

1.4 The Hypertext Transfer Protocol

The Hypertext Transfer Protocol [HTTP] has been in use by the World-Wide Web global information initiative since 1990. HTTP is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hyper-media information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

The Hypertext Transfer Protocol [HTTP] is a network protocol built for the WWW. Certain operations, called *methods*, are associated with the object oriented HTTP protocol. These methods define extensions to the HTTP commands and can be associated with particular types of network objects such as documents, files, or associated services.

The purpose of HTTP was to make available many sources of related, networked information via the Internet. It is possible to browse these information stores, to rapidly follow references from any source of information to other pertinent sources which may themselves be located at multiple remote locations. HTTP's functionality includes search and retrieval, front-end updates, and document annotation. HTTP allows an extensible set of methods to be designed and deployed. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI), a well-thought-out scheme that originated with the Internet Engineering Task Force (IETF). There are two types of URIs, the Uniform Resource Name (URN) and the Uniform Resource Locator [URL1996].

HTTP also features dynamic data representation through client/server negotiation. This allows WWW information systems to be built independent of the development of new information representations. The way information is represented can be worked out as part of the process of shipping data from servers to clients, and vice versa.

On the Internet, network communications occur via TCP/IP connections. Occasionally, certain situations may arise where this protocol suite is implemented atop another network protocol. In this case, HTTP objects may be mapped onto foreign transport mechanisms or networks. This is a simple, straightforward process, one that usually goes unnoticed by WWW users.

The HTTP protocol is stateless - that is, neither the client nor the server stores information about the state of the other side of an ongoing connection. Each minds its own business and manages state information for itself. This supports the simplest kinds of client and server applications, and helps to account for the broad reach and platform support found in the Web community. Most networking protocols depend on the notion of a transaction, which consists of the following elements:

- the establishment of a *connection* between a client and a server, to permit communications to occur
- a *request* from a client to a server, for specific services, resources, or other known topics of communication
- a *response* from the server to the client's request, supplying the requested resource or an explanation as to why it can't be delivered, and
- the termination of the connection, at the *close* of request/response communication

This rhythm *(connection, request, response, close)* describes the basic form of interchange between clients and servers the world over, both on and off the Web.

1.5 HTML

The HyperText Markup Language [HTML](see Appendix A at page 64) is a simple markup system used to create hypertext documents that are portable from one platform to another. HTML documents are SGML (Standard Generalized Markup Language) documents with generic semantics that are appropriate for representing information from a wide range of applications. HTML markup can represent hypertext news, mail, documentation, and hypermedia; menus of options; database query results; simple structured documents with in-lined graphics; and hypertext views of existing bodies of information.

HTML's evolution continues to be a fascinating story. Starting with a simple set of basic markup and text elements, HTML is evolving into a complex hypermedia markup language. Along with this evolution from simple to complex, there have been some digressions from the original basic markup model to today's complex collection of table definitions, mathematical notations, and complex markup requirements. For instance, HTML+ is one branch of the HTML tree that reached a dead end (but not before exerting a major influence on level 3 development). In fact, many artifacts from HTML+ have been integrated into the level 3 DTD (Data Type Definition), while numerous others met their demise.

1.6 Search Engines

To extract information of the Internet is quite a difficult task since the Internet became a huge hay stack of information.

Using the right tools makes researching the Web much simpler. There is a class of software tools called search engines that can examine huge amounts of information to help you locate Web sites of potential interest.

Here's how most of them work: Somewhere in the background, laboring in patient anonymity, you'll find automated Web-traversing programs, often called robots or spiders (see chapter 4 at page 33), that do nothing but follow link after link around the Web, ad infinitum. Each time they get to a new Web document, they peruse and catalog its contents, storing the information for transmission to a database elsewhere on the Web. At regular intervals these automated information gatherers transmit their recent acquisitions to a parent database, where the information is sifted, categorized, and stored.

When you run a search engine, you're actually searching the database that's been compiled and managed through the initial efforts of the robots and spiders, but which is handled by a fully functional database management system that communicates with the CGI (see chapter 2 at page 12) program for your search form. Using the keywords or search terms you provide to the form, the database locates "hits" (exact matches) and also "near-hits" (matches with less than the full set of terms supplied, or based on educated guesses about what you're really trying to locate).

The hits are returned to the CGI program by the database, where they are transformed into a Web document to return the results of the search for your perusal.

If you're lucky, all this activity will produce references to some materials that you can actually use! The term search-engine will be used throughout this thesis to represent a user-interface with which you can define queries.

Chapter 2

The Common Gateway Interface

2.1 Introduction

Since Web servers can generate custom built Web documents on-the-fly in response to interaction with end users, this makes online WWW publishing much more responsive and open-ended than traditional publishing. The need for this interaction with end users started a new development as an extension of HTTP called CGI, the Common Gateway Interface [CGI].

The Common Gateway Interface [CGI] is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon (see section 2.8 at page 22) retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

2.2 Dynamic behaviour

Quick response to change and dynamic behaviour are the underpinnings of the power behind the WWW. Dynamic behaviour is a vital aspect of the Web that is sometimes hard to identify or appreciate. This is particularly true when documents are created on-the-fly, usually in response to some event on the WWW. Although what appears to a user in response to a query may look like "just another Web page", it might actually be an evanescent document created for one-time use in direct response to that query. This technology is called the Common Gateway Interface [CGI].

For example, let's say that you wanted to "hook up" your Unix database to the World Wide Web, to allow people from all over the world to query it. Basically, you need to create a CGI program that the Web daemon (see section 2.8 at page 22) will execute to transmit information to the database engine, and receive the results back again and display them to the client. This is an example of a gateway, and this is where CGI, currently version 1.1, got its origins.

The database example is a simple idea, but most of the time rather difficult to implement. There really is no limit as to what you can hook up to the Web. The only thing you need to remember is that whatever your CGI program does, it should not take too long to process. Otherwise, the user will just be staring at their browser waiting for something to happen.

2.3 HTML-based interfaces

Today, we are living in a multi-platform world in which everyone wants to create applications that are platform independent. This can be done with expensive cross-compilers and expensive GUI generating software packages. But why not take advantage of other people's hard work and make things easy for yourself by using HTML. There is no reason why you and everyone else has to keep writing the"same" (more or less) code for generating windows, buttons, boxes, pictures, etc. You would like to be able to make an input field as easily as writing <input name="input"> , which can be done in HTML. You should not have to write code to handle selecting text in the window, or code to make the input "clickable". Your web browser will take care of all of that for you. Thus, ultimately, when you use HTML forms you only have to describe the way that you want the interface to look and you leave the work of generating it to the web browser.

2.3.1 What needs to be done?

Your interface will contain two main parts: a form which will obtain some type of useful information, and a program that will perform some type of computation on that information ultimately writing its results into yet another HTML document. Basically our main goal is to learn how to write these programs. They will be written to take CGI style input from our forms and generate output as HTML documents which can be accessed through your favorite web browser.

2.3.2 Why not just write a GUI?

The main reason for using HTML as a front end for your programming projects is that by doing so you are taking advantage of the fact that there exist Web browsers for virtually every platform that you might be interested in having your software running on. So as long as you have a compiler that can create executable code for the target machine which can "run from the command line" and there exists a web browser for that platform and presentation environment, the modifications to use HTML are minimal. In fact these modifications do not necessarily require the re-compiling of the executables, scripting languages like Perl (see section 2.6 at page 20) can be used as a "wrapper" for the existing code.

2.4 Getting started

What will I need to get started?

- First you will need a Web browser (see section 1.3 at page 8).
- You will need the facilities to generate the executable code: a compiler if you are writing in a compiled language(C, C++, PASCAL, etc.), or a script parser if you are writing in a language like Perl (see section 2.6 at page 20) or Tcl.
- Libraries to make extracting information from CGI style documents easier (see section 2.7 at page 21 for more details).
- Last but not least you will need a computer running a web server (see section 2.8 at page 22), and the server does have to allow the running of CGI programs. If you are not sure about this you should check with your local system administrator, or local computer expert.

2.5 Choice of programming language?

Programming languages come in many different flavors: procedural (e.g., C, Basic, and FORTRAN), object-oriented (e.g., C++, Smalltalk, and Java), logic languages like Prolog, and functional languages like Lisp. There are more programming languages out there than most of us could ever learn, even in several lifetimes!

Every programming language has its own solution domain and its own philosophy; that is, each one best solves specific types of problems, and an application's design is influenced by the language's philosophy. For example, Lisp isn't practical for number crunching, nor is FORTRAN the best choice for heavy-duty string manipulation. Languages taken from the same paradigm, like C and Pascal, have different syntaxes but you can translate between them with relative ease. Both are procedural languages with similar constructs and design principles.

Choosing the right programming language should be the result of a thorough and painstaking problem analysis. In some cases, a language might be selected simply because the programmer understands the language and can use its constructs and syntax to their best advantage. In other cases, languages are chosen because they have proved to be more understandable, more reliable, more efficient, or more extensible than others. When coming to writing code, there is seldom much sense in reinventing the wheel. Working from public domain, freeware, or shareware source code is usually a great way to learn a new language because of the variety of algorithms implemented and examples available. In any case, once a programming language is chosen, there is no going back without considerable work, pain, and suffering.

There are five primary considerations when choosing a language for CGI programming:

1. the amount of public source code in easily accessible repositories
2. the availability of support and infrastructure tools like debuggers, compilers, interpreters, tutorials, books, classes and *language aware* editors
3. your own level of knowledge of a particular language or class of languages
4. the desired throughput of data, compared to support for special operations
5. the *ilities* of the language: extensibility, modularity, usability, and reusability

All of these characteristics play an important role in your selection, but typically the first two or three outweigh the rest. One main distinction can be made involving the choice of a CGI programming language:

- Compiled CGI Programming Languages:
  - C
  - C++
- Interpreted CGI Programming Languages:
  - Perl
  - Tcl
  - Python
- Compiled/Interpreted CGI Programming Languages:
  - Java

2.5.1 Compiled CGI Programming Languages

Compiled languages create binary objects that are loaded and executed in the computer's main memory. These objects are the result of a compiler assembling ASCII source code into binary information (0s and 1s).

A compiler is typically native to a particular computer architecture and you can assume that binary objects from one architecture will not execute on another architecture. For example, don't expect Pascal code compiled into a binary object and run on a PC to execute successfully on a UNIX machine. Instead, the original Pascal source code must be compiled with a native UNIX Pascal compiler into an UNIX-compatible binary object.

C

The C language was first implemented in 1972 on an old dead and forgotten machine (a DEC PDP-9) that lay dormant in a hallway at Bell Labs. Brian Kernighan and Dennis Ritchie created C as a language for their own use and amusement. Over 20 years later, it's still one of the most popular languages in use.

C is a procedural program that describes the steps of an algorithm, like a procedure on how to install a water pump on a Chevy. You complete each successive step until you have a new and functional water pump installed, or your sorting algorithm assembles a list in ascending alphabetical order.

There are two primary advantages in choosing C as a CGI programming language. First, it can be compiled into a tight binary object that takes up minimal space compared to interpreted languages. Some C compilers include command-line options that instruct it to create an optimized binary object. This results in an even smaller binary object. Second, binary objects typically execute faster than interpreted languages. If speed of execution is a major concern, you should definitely pick a compiled programming language. The primary disadvantage to using C for CGI programming is that it is difficult to manipulate strings with C language constructs. Nearly 90% of all CGI applications involve heavy string manipulation. This means that character and string data must be massaged, transformed, converted, or translated from one to another.

By contrast, integer and floating-point math CGI applications are few and far between. Most CGI applications take string and character data such as `<FORM>` data or query data and return other string and character data based on some embedded heuristics. For example, a typical CGI application gathers `<FORM>` data from a "Comments and Suggestions Form" interface. The CGI application then assembles the `<FORM>` data into a MIME mail message and mails it to the designated address. It can also record <`FORM`> data to a log file. “Heavy string manipulation” describes a day in the life of a typical CGI applications engineer.

C++

C++, a successor to C, is a member of the object-oriented (OO) paradigm of programming languages. Object-oriented languages like C++ offer many advantages. They provide superior reusability of classes, which reduces the cost of development for similar applications. They also allow extensibility of core classes by allowing programmers to add functionality.

They also support modularity, which is the methodology of breaking a problem down into its smallest understandable units, where each module acts as a separate functional unit.

Finally, OO languages stress the reusability of classes across many different problem domains. Reusability of source code is a popular research topic because it reduces the cost of development. It can also increase the quality of source code but this depends heavily on the quality of the software engineering of the component classes in use. As yet, reusability is talked about far more than it’s practiced, so its benefits remain more theoretical than actual.

The main disadvantage of using C++ is that it belongs to the OO paradigm. OO development of source code is a completely different beast, requiring substantial training and street smarts. Designing classes for reusability, understanding the polymorphism of functions within a class, and providing effective management of classes within and across applications are new concepts for many application engineers.

Another disadvantage to using C++ is that there is only limited public domain CGI source code available. This is probably because CGI is a relatively new area, and few OO CGI applications have been released for public consumption. But this should begin to change once software engineers begin to develop OO Web solutions.

2.5.2 Interpreted CGI Programming Languages

In this section we look at some of the common interpreted languages you can use to create CGI applications – namely, Perl, Tcl and Python.

Perl

Perl (see also section 2.6) provides a lucid and succinct way to solve many programming problems typical in the CGI realm. Perl is not yet a standard part of UNIX, but is widespread.

The Perl language is intended to be easy to use, but also to be complete and efficient, rather than tiny, elegant, and minimal. Perl combines some of the best features of C, *sed*, *awk*, and *sh*. Programmers familiar with these languages should have little difficulty learning and applying Perl.

Today, most CGI applications use Perl because of its many positive characteristics. You will notice that in many of the public CGI source code repositories, more than half the code is written in Perl.

Tcl

The Tool Command Language (Tcl, pronounced "tickle") is a simple scripting language for extending and controlling applications. Tcl can be embedded into C applications because its interpreter is implemented as a C library of procedures. Each application can extend the basic Tcl functions by creating new Tcl commands that are specific to a particular programming task.

Accompanying Tcl is a very popular extension called Tk (pronounced "tee-kay"). It is a toolkit for the X Window System found on many UNIX machines. Tk extends the basic Tcl functionality with commands to rapidly build Motif or X Windows user interfaces. Tk is also implemented as a C library of procedures, allowing it to be used in many disparate applications. Like Tcl, Tk can be extended, typically by constructing new interface widgets and geometry managers in C.

Since Tcl is interpreted, Tcl applications typically will not execute as fast as their C counterparts. For a small class of applications this may become a disadvantage but with the blinding speed of today's computer systems, Tcl/Tk represents an adequate application system. If the speed of execution is critical in your application, don't fret. Tcl can be compiled or heavy-duty processing can be written in a compiled language such as C and C++, and the user interface programmed in Tcl/Tk. If this is unacceptable, create a throwaway prototype of the user interface using Tcl/Tk and get feedback from your target users to build a faster final implementation.

Because of its extensibility and depth, Tcl/Tk is adequate for all but the most processor-intensive applications. It is particularly well suited for those that require complex graphical displays or sophisticated user interfaces.

Python

Python is an interpreted, interactive, object-oriented programming language. It combines an understandable and readable syntax with note-worthy power compared to other interpreted languages. It has modules, classes, exceptions, and dynamic data types and typing. Python also provides interfaces to many system calls and libraries, and to various windowing systems like X11, Motif, Tk, and Mac. Python can even be used as an "extension language" for applications that require a programmable interface. Finally, new built-in Python modules can be implemented in either C or C++.

Python executes on many platforms, including UNIX, Mac, OS/2, MS-DOS, Windows 3.1, and NT. Python is copyrighted but freely usable and distributable to individuals as well as commercial institutions.

Python has been used to implement a few WWW modules. Currently these modules include a CGI module, a library of URL modules, and a few modules dealing with Electronic Commerce.

2.5.3 Compiled/Interpreted CGI Programming Languages

Java

Java is a new object-oriented programming language and environment from Sun Microsystems. Along with C and C++, Java is compiled into an architecture-neutral binary object and then interpreted like Perl or Tcl for a specific architecture. So, it's a dessert topping and a floor wax!

With Java, you can create either standalone "applications" or "applets" to be used within CGI applications.

Java is a strictly-typed object-oriented language, similar to C++ without many of that language's shortcomings. For instance, Java will not let you cast an integer type to a pointer.

Java applications can execute anywhere on a network, making it highly suitable for CGI applications. Another really interesting aspect of the language is that the Java compiler creates an "architecture-neutral" binary object. This object is executable on any platform that has a Java runtime system installed. You can write *one* Java program that can execute on all other supported platforms including Mac, UNIX, NT, and Windows.

Netscape Communications Corporation has licensed the Java language to implement within their Netscape Navigator browser. Their main motivation is to increase the extensibility of Navigator and to enable the creation of a new class of client/server networked applications.

Java and HotJava – the WWW browser from Sun written in Java – are freely available in binary form to individuals. Java can also be licensed to commercial institutions.

2.5.4 The selection of the CGI programming language

Programming Language	Available source-code	data-handling abilities	extensibility, modularity, usability, and reusability
C	+-	+	-
C++	+-	++	++
Perl	++	++	++
Tcl	-	+-	+-
Python	+-	+-	+
Java	+-	++	++

Table 2.1: advantages vs disadvantages

Considering all these choices (see table 2.1) we chose Perl as the CGI programming language for this project, not only because Perl is an interpreted language, but because there is a big load of Perl source-code available for CGI programming on the Net. Java is a good runner-up, but in this case unsuitable since we do want to stay in charge of the applications execution, since Java only executes when a Java runtime system is installed. Since Perl can be used in combination with the C++ language it meets all our requirements, like integer and floating-point math CGI applications (see also section 2.6 at page 20).

2.6 Perl

Perl, the Practical Extraction and Report Language [Perl], is an interpreted language optimized for easy manipulation of files, text, and processes. Perl is typically used when scanning text files, extracting text strings, and printing reports based on the information that's extracted. Perl was created by Larry Wall in the early 1980's. Perl provides the best of several worlds. For instance:

- Perl has the power and flexibility of a high-level programming language such as C. In fact many of these features of the language are borrowed from C.

- Like shell script languages, Perl does not require a special compiler and linker to turn the programs you write into working code. Instead, all you have to do is write the program and tell Perl to run it. This means that Perl is ideal for producing quick solutions to programming problems.
- Perl provides all of the best features of the script languages sed and awk, plus features not found in either of these two languages. Perl also supports a sed-to-Perl translator and a awk-to-Perl translator.

The Perl language is intended to be easy to use, but also to be complete and efficient, rather than tiny, elegant, and minimal.

Perl's syntax and structure is very similar to C. Many of the constructs in Perl like **if, for,** and **while** correspond to their counterparts in C. With Perl, you can manipulate and match regular expressions with great ease. Today, most CGI applications use Perl because of its many positive characteristics. You will notice that in many of the public CGI source code repositories, more than half the code is written in Perl (see also Appendix B at page 67).

2.7 CGI.pm - a Perl5 CGI Library

HTML documents can specify interactive fill-out forms – with input elements including text entry areas, toggle buttons, selection lists, popup menus, etc. (and your Web browser will instantiate such fill-out forms as sets of Motif widgets embedded inside the documents). This provides a way to provide arbitrarily sophisticated front-end interfaces to databases and search engines, as well as other network services (e.g., ordering pizzas).

As you now know, there are two methods which can be used to access your forms. These methods are GET and POST. Depending on which method you used, you will receive the encoded results of the form in a different way.

- The GET method

  If your form has METHOD="GET" in its FORM tag, your CGI program will receive the encoded form input in the environment variable QUERY_STRING.
- The POST method

  If your form has METHOD="POST" in its FORM tag, your CGI program will receive the encoded form input on stdin. The server will NOT send you an EOF on the end of the data, instead you should use the environment variable CONTENT_LENGTH to determine how much data you should read from stdin.

When you write a form, each of your input items has a NAME tag. When the user places data in these items in the form, that information is encoded into the form data. The value each of the input items is given by the user is called the value.

Form data is a stream of name=value pairs separated by the & character. Each name=value pair is URL encoded, i.e. spaces are changed into plusses and some characters are encoded into hexadecimal. The basic procedure is to split the data by the ampersands. Then, for each name=value pair you get for this, you should URL decode the name, and then the value, and then do what you like with them. Because others have been presented with this problem as well, there are already a number of libraries which will do this decoding for you.

One of these libraries is "CGI.pm" a Perl5 library for handling forms in CGI scripts. With just a handful of calls, you can parse CGI queries. However, it also offers a rich set of functions for creating fill-out forms. Instead of remembering the syntax for HTML form elements, you just make a series of Perl function calls. An important fringe benefit of this is that the value of the previous query is used to initialize the form, so that the state of the form is preserved from invocation to invocation.

Everything is done through a "CGI" object. When you create one of these objects it examines the environment for a query string, parses it, and stores the results. You can then ask the CGI object to return or modify the query values. CGI objects handle POST and GET methods correctly, and correctly distinguish between scripts called from <ISINDEX> documents and form-based documents. In fact you can debug your script from the command line without worrying about setting up environment variables. For more detailed information about "CGI.pm" see Appendix C at page 69.

2.8 HTTP Daemon

After doing almost everything on our little list (see section 2.4 at page 14) we now only need to take care of a computer running the web server. In our case we were able to set up a web server at one of the Indy machines. I chose to use the NCSA web daemon which is easy to install and most of all it's public domain software.

POWERED BY
NCSA HTTPd

Figure 2.1: NCSA HTTPd an HTTP/1.0 server

NCSA HTTPd is a program to serve information, much in the same way that Netscape is a program to browse information in the World Wide Web. From the Client-Server viewpoint, NCSA HTTPd is the Server to the Browser Client. Currently I'm using the NCSA HTTPd version 1.5.1 which is an HTTP/1.0 compatible server. For more detailed information about setting up your own HTTP server please see Appendix G at page 86.

Chapter 3

Leiden 19th Century Portrait Database

3.1 Content-based Image Retrieval

One of the tools that will be essential for future electronic publishing is a powerful image retrieval system. The author should be able to search an image database for images that convey the desired information or mood; a reader should be able to search a corpus of published work for images that are relevant to his or her needs. Most commercial image retrieval systems associate keywords or text with each image and require the user to enter a keyword or textual description of the desired image. This text-based approach has numerous drawbacks - associating keywords or text with each image is a tedious task; some image features may not be mentioned in the textual description; some features are "nearly impossible to describe with text"; and some features can by described in widely different ways. In an effort to overcome these problems and improve retrieval performance, researchers have focused more and more on *content-based* image retrieval in which retrieval is accomplished by comparing image features directly rather than textual descriptions of the image features. Features that are commonly used in content-based retrieval include color, shape, texture and edges.

It is hoped that content-based techniques can provide the basis for powerful *"query by example"* retrieval systems. For example the user might provide a sample picture and request similar pictures, a picture of an object and request pictures that contain the object, a set of colors and request images that contain those colors, and so on.

3.2 Introduction of LCPD

The problem area is locating copies and former copies in a large picture database. As objects [Huijsmans] choose pictures within a scanner window; the pictorial objects are 19th-century B/W studio portraits, whose front and back sides provide a testbed for graylevel and nearly binary images.

Copy location in this area is a realistic user query because:

- the playcard sized portraits mounted on carton were usually printed from a glass negative and sold to the customer a dozen copies at a time.
- the former set of obtained copies found their way into the private photo albums of family members and friends.
- at present more and more of the remaining portraits are found in private and institutional collections: both the Print Room of Leiden University and the Icongraphic Office in The Hague have about 10,000 so-called carte de visite portraits.
- for genealogists and art historians tracing copies of family photographs may give important clues to the name of the person(s) depicted, date of production and the relations of the person(s) depicted.
- known archives in the Netherlands contain over 50,000 of such portraits, distributed over more than 15 collections and either sorted on different keys or inaccessible in albums, making non-computerized copy retrieval close to impossible.
- an estimated 50 million of such portraits have been produced in the Netherlands between 1860 and 1914; about 5 million of these still reside in family archives. There is a growing interest within genealogical societies to date these portraits and to determine which persons are depicted.

The following similarity methods were implemented [Huijsmans]:

- pixel to pixel difference in intensity space
- pixel to pixel difference in gradient space
- pixel to pixel difference in thresholded gradient space
- row and column line integrals: the horizontal and vertical projections in intensity space
- row and column line integrals: the horizontal and vertical projections in gradient space
- row and column line integrals: the horizontal and vertical projections in thresholded gradient space

- 3x3 B/W spatial pattern statistics vector (on binarized gradient magnitudes only)

With this photo-database and precalculated results of the similarity methods we started the project with building a WWW user-interface that allows querying the database by looking up the corresponding precalculated results of the similarity methods (i.e. a search engine, see section 1.6 at page 11).

3.3 Implementation

Starting off with my own WWW-server and the right CGI/Perl libraries (see section 2.4 at page 14) we had to design the user-interface first. First off all we wanted to display multiple images on the screen, therefore we decided to use thumbnail sized images. One big advantage of using thumbnail sized images is that the user gets a better overview with more than just one image, also no information is lost because the user is still capable of understanding the image's contents. Another advantage is digital storage and retrieval which is essential to WWW programs.

The user-interface has to meet the following primary requirements:

- the interface must present a certain number of images (from the database) on the browser's screen, the number of rows and the number of columns must be variable to select a 2-D presentation space.
- the interface must be able to select images by clicking with the mouse on the according image.
- by selecting an image a new set of images (according to method, selected image, and selected 2-D representation space) must be presented on the screen.

So the first thing was to create some sort of a display which could show multiple thumbnail sized images, before a real user interface was born (see figure 3.1). The way of representing these thumbnail sized images to the user was to use a table-like layout. This way we could use rows and columns to display the thumbnail sized images. These thumbnail sized images are presented as so-called clickable image-maps, that means that the images are shown as buttons, so the user can select an image (or a location within that image) by a mouse click.

The images were also provided with an URL to offer the user the possibility to see the original image in its full size. This way the user is able to actually see if the chosen picture is really the correct one, because of it's thumbnail size.

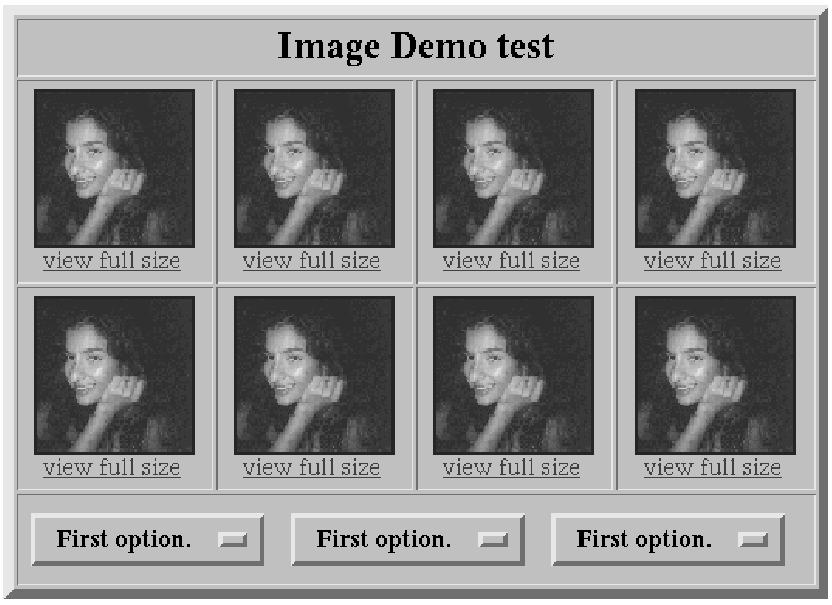


Figure 3.1: First layout http://ind156b.wi.leidenuniv.nl:8086/image.html

Next thing to try out was to see if it is easy to create buttons. In figure 3.1 you can see three so-called “popup-menu” buttons. With such a button the user can select an option (e.g. method) from a popup-menu.

The main problem to tackle was the problem of maintaining state information from invocation to invocation. To build a user interface as a HTML page you must preserve the parameters from the previous page in order to present a new one with respect to these parameters. But because multiple users can actually request this page at the same time a solution had to be found. With the “CGI.pm”(see section 2.7 at page 21) library we were able to tackle this problem.

After setting up a test-user-interface we could now start using the images from the *Leiden 19th Century Portrait Database*[Huijsmans]. The images had to be converted to an image format suitable for WWW-browsers, we chose the JPEG format, because we wanted to keep the size (in bytes) per image as small as possible, due to WWW-speed. Image quality of JPEG images are somewhat poor, but because we are dealing with just a demo the storage requirements were given the higher priority (see figure 3.2).

Next we made the user-interface operational; the user is able to select an image by a mouse click, the selected image is than swapped with the image in the left upper corner. So we chose the left upper corner to be the spot for the search image.

Note: underneath every shown thumbnail-sized image, the corresponding (*front & back*) URL is given, selecting this link will present the user the (zoomed-in) front- and back side of the corresponding photo.

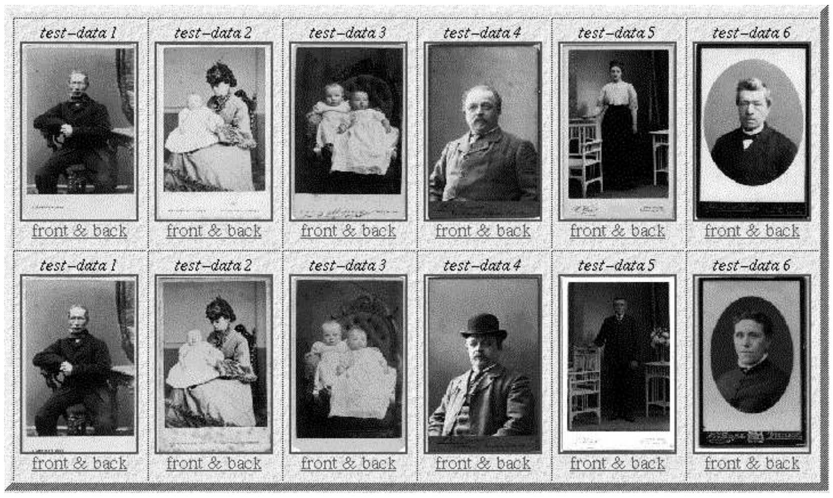


Figure 3.2: Layout of the thumbnail sized image area of the LCPD

3.4 Demo features

Now it was time to actually implement some extra options:

- Changing the number of images displayed
- The Similarity Methods
- Changing the photo view of images displayed
- New Random-set selection
- Test-set selection
- Help buttons

3.4.1 Changing the number of images displayed

The advantage of using a table view of the thumbnail sized images is that it is easy to change the size of it by adjusting the number of rows and the number of columns. By implementing two popup-menu buttons, one representing the number of rows displayed, and one representing the number of columns displayed, the user can select any size ranging from 1x1 to 25x25 images by selecting an appropriate display size from the corresponding popup menus (see figure 3.3).

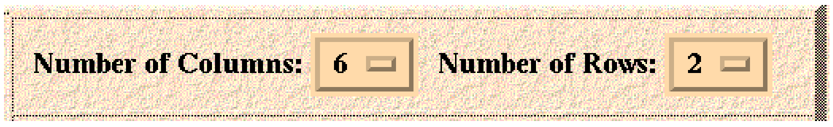


Figure 3.3: Changing the number of images displayed

3.4.2 The Similarity Methods

The result files of the similarity methods were used as lookup tables for the demo. The format of these result files were not all in the same format. To speed up searching within those result files we have made an index file for each result file which contains for every entry the number of bytes of that entry in the corresponding result file. With this index file we could make a fair estimation of where the desired results could be found, because some entries were skipped (no image available). The user interface just had to be added with another popup-menu button containing the different similarity methods (note: The similarity methods were already mentioned in section 3.2). See figure 3.4.

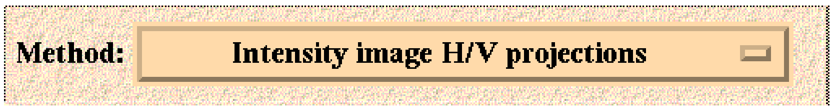


Figure 3.4: Selecting a similarity method

3.4.3 Changing the photo view of images displayed

As mentioned before [Huijsmans] both front and back sides of the LCPD were scanned and used for the similarity methods. This implies that for every image two result sets are present; one for the front-side of the corresponding portrait photo, and one for the back-side of that same portrait photo. To provide the user with this option we implemented two radio buttons representing the front- and backside of the photo view (note: radio buttons only allow one selection). See figure 3.5.

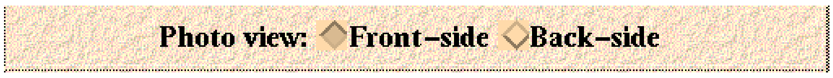


Figure 3.5: Changing the photo view of images displayed

3.4.4 New Random-set selection

To offer a greater functionality the user must be able to browse through the database to find his or her favorite image to start a new query with. To meet this requirement a randomizer option seemed to be the right solution. This way the user can select a new random set of thumbnail sized images to be displayed and hopefully find the right image to start his or her query. This option is implemented as a submit-button, after clicking this button a new random-set of images will be presented, see figure 3.6.

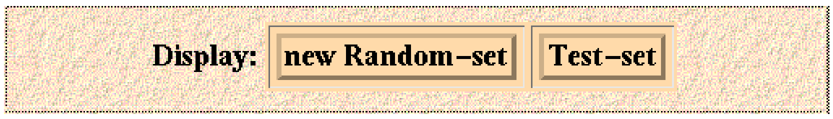


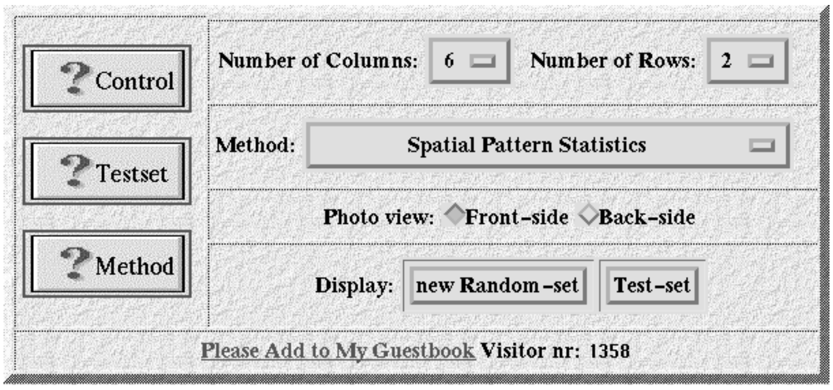
Figure 3.6: Selecting a new random set

3.4.5 Test-set selection

For first time users the functionality of this demo can be somewhat unclear. To explain how the results of certain similarity methods with certain photos should look like we added two testsets, one for the front-side thumbnail sized images, and one for the back-side thumbnail sized images. As a default setup this testset will be presented to the user. This option is implemented as a submit-button also, after clicking this button the corresponding test-set (see section 3.4.3) will be presented, see figure 3.6.

3.4.6 Help buttons

To create a more user-friendly atmosphere some help-pages were necessary. Within these pages the user can find information about usage and backgrounds of this demo. These helpbuttons are merely links to other static HTML-documents, and are implemented as clickable image-maps (see figure 3.7).


Control
Testset
Method
Photo view: Front-side Back-side
Display: new Random-set Test-set
Please Add to My Guestbook

Number of Columns: 6 Number of Rows: 2

Method: Spatial Pattern Statistics

Visitor nr: 1358

Figure 3.7: The control panel

Finally we grouped all these options to one control panel (see figure 3.7). Some extra features are also included (at the bottom of the control panel):

- Guestbook; extra user statistics, you can leave your email address and remarks about the demo here. These statistics are gathered and listed in a HTML document (*http://ind156b.wi.leidenuniv.nl/guestbook.html*).
- Visitor counter; keeps track of the total number of visitors that actually visited the demo, these statistics are also massaged, inspected and stored in a HTML document (*http://ind156b.wi.leidenuniv.nl/demostat.html*).

3.5 Conclusion

With the thumbnail-sized image table and the control panel, the user is now able to define his or her queries by adjusting the options and selecting an image or a submit-button (e.g. random-set or test-set). This search-engine is set up to be easily extended with extra features and new methods.

Some enhancements can be made:

- One thing that could be improved is using a uniform standardized format for storing the result files of the similarity methods. Currently we have to maintain several index files, which will only increase overhead and decrease delivery speed.
- The images shown as thumbnail sized images are actually zoomed-out images, however the full size presentation of the front- and back side are zoomed-in. To increase delivery speed of the thumbnail-sized images we

could store a separate thumbnail-size image copy of every image, to speed up retrieval, instead of scaling the original image.

Chapter 4

WWW Robots, Spiders and Webcrawlers

4.1 Introduction

In the previous chapter we discussed the concept of a search engine based upon a certain class of digital images from a local database. Now we would like to see if it is possible to create a search engine that can perform queries based on image content. This time we don't want to use a local database, instead we would like to create a digital image database. This database should contain a large variety of digital images which would be interesting enough to perform certain queries. The answer was clear, we wanted to use the World Wide Web as a resource for our database entries. To perform this task of collecting images from the Web we needed a robot.

4.1.1 What is a Robot?

The useful information returned by search engines on the Web doesn't just come out of nowhere. Rather, it is laboriously gathered by software automaton that cruise the Web, with either broadly or narrowly defined objectives and read through all the HTML documents they encounter, harvesting or calculating all information that meets their programming criteria.

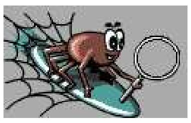


Figure 4.1: Typical Spider.

These programs are called *Web Robots*, or simply *robots*. Sometimes, you'll hear them referred to as *spiders* or *WebCrawlers* (see figure 4.1). No matter what they are called, these programs all perform similar tasks: they pick up selected pieces of information from the Web documents they find - at least the URL and the title, and if not more - and report their findings back to a logging program on the originator's server. This log is massaged, inspected and digested to create the database from which search engines pluck their responses to your queries.

In reality, a robot is nothing more than a browser-like program that uses ordinary HTTP protocols to request access to Web resources and documents. Robots typically understands links, URLs, and other selected HTML tags and information. They know how to catalog these tags or how to abstract statistics based on what they encounter. Since they don't actually display anything, you could think of them as a kind of "headless browser," that chew their way around the Net with an inexhaustible appetite for any new links that might come their way.

4.2 Robot reuse

Given that robots are good at gathering information, why might you hesitate before adding another one to the collection available on the Internet? For one thing, there are already 90-odd robots running somewhere on the Net as you're reading this (http://info.webcrawler.com/mak/projects/robots/active.html).

Unfortunately robots can put a considerable strain on network resources and tax the very servers whose contents they may be cataloging or measuring. Whereas a human might try to read and comprehend some of each document that he or she examines, robots can zip from one link to the next at dizzying rates.

Given the sizable number of robots already in existence, you might want to consider whether one of them could perform the kinds of tasks you're after, rather than creating a new one. Likewise, you might want to investigate one or more of the publicly available search engines.

To reuse an already available robot doesn't only mean that we don't have to *reinvent the wheel*, but we also don't have to worry about specific guidelines for robot writers and not to mention the robot exclusion standard [Koster94]. To avoid all these problems it was the best choice to seek for an existing robot we could reuse.

4.3 MOMspider

After a certain study of publicly available robots on the Web we have decided to use the Multi-Owner Maintenance spider [MOMspider] for my project. Actually no other public robots were available at the time of study, and fortunately MOMspider could perform our tasks. This robot is written in Perl and documentation for this robot is widely available. In this section a rough outline of the design and functionality of the [MOMspider] is described.

4.3.1 Design

Most documents made available on the World-Wide Web can be considered part of an infostructure – an information resource database with a specifically designed structure. Infostructures often contain a wide variety of information sources, in the form of interlinked documents at distributed sites, which are maintained by a number of different document owners (usually, but not necessarily, the original document authors). Individual documents may also be shared by multiple infostructures. Since it is rarely static, the content of an infostructure is likely to change over time and may deviate from the intended structure. Documents may be moved or deleted, referenced information may change, and hypertext links may be broken (dangling links).

As it grows, an infostructure becomes complex and difficult to maintain. Such maintenance currently relies upon the error logs of each server (often never relayed to the document owners), the complaints of users (often not seen by the actual document maintainers), and periodic manual traversals by each owner of all the webs for which they are responsible. Since thorough manual traversal of a web can be time-consuming and boring, maintenance is rarely or inconsistently performed and the infostructure eventually becomes corrupted. What is needed is an automated means for traversing a web of documents and checking for changes which may require the attention of the human maintainers (owners) of that web.

The Multi-Owner Maintenance spider [MOMspider] has been developed to at least partially solve this maintenance problem. MOMspider can periodically traverse a list of webs (by owner, site, or document tree), check each web for any changes which may require its owner's attention, and build a special index document that lists out the attributes and connections of the web in a form that can itself be traversed as a hypertext document.

4.3.2 Functionality

MOMspider gets its instructions by reading a text file that contains a list of options and tasks to be performed (an example instruction file is provided in

Appendix D at page 79). Each task is intended to describe a specific infostructure so that it can be encompassed by the traversal process. A task instruction includes the traversal type, an infostructure name (for later reference), the "Top" URL at which to start traversing, the location for placing the indexed output, an e-mail address that corresponds to the owner of that infostructure, and a set of options that determine what identified maintenance issues justify sending an e-mail message.

For each task, MOMspider traverses the Web, in breadth-first order, from the specified top document down to each leaf node. A leaf node is defined to be any information object which is not of document-type HTML (and thus cannot contain any further links) or which is outside the given infostructure. MOMspider determines the boundaries of an infostructure according to the task's traversal type: **Site, Tree,** or **Owner**. Site traversal specifies that any URL which points to a site (the pairing of hostname/IP address and port) other than that of the top document is considered to be a leaf node. Tree traversal specifies that any document not at or below the "level" of the top document is considered a leaf node, where level is determined by the pathname in the URL. Owner traversal specifies that any document beyond the top which does not contain an "Owner:" metainformation header equal to the infostructure name is considered a leaf node.

The Maintenance information produced by each task is formatted as an HTML index and output to the file specified in the task instructions. The index contains the following maintenance information:

- Information regarding how and when the index was generated (i.e. program options and execution time);
- A hypertext link to the one prior version of the index document;
- The following for each non-leaf document accessible via the "top";
  - An anchor which links to the actual document;
  - Document header info (Title, Modification Date, Expires Date, etc);
  - A list of all unique hypertext references made by the document, with each reference including:
    - The type of reference made (i.e. link, query, img, etc.);
    - An anchor which duplicates the reference;
    - Document header info if available (Title and Modification Date);
    - If the referenced object is within the current infostructure (i.e. not a leaf), then an additional anchor is provided to cross-reference jump to its own entry in the index document..

- A list of cross-reference anchors which point to interesting changes as reflected in the index entries.

MOMspider looks for four types of document change which may be of interest of the owner:

1. referenced objects which have redirected URLs (moved documents);
2. referenced objects which cannot be accessed (broken links);
3. referenced objects with recent modification dates; and,
4. owned objects with expiration dates close to the current date.

4.3.3 Efficient use of network resources

A key design constraint of MOMspider is that of efficiency – particularly in regards to network bandwidth usage. It would be irresponsible to develop a maintenance robot which overly taxed the limited resources of networks like the Internet. Therefore, MOMspider minimizes the load on network bandwidth by using the *HEAD* request (see Appendix E at page 81) for testing links, keeping track of nodes that have already been tested, grouping multiple tasks within a single execution, and allowing the user to restrict the traversal of certain URLs.

Aside from the restrictions described above regarding the task's traversal type, MOMspider also enables the user to specify any URL prefixes which must always be avoided or leafed. These URL prefixes are listed in the systemwide or user avoid files. Each entry in the file includes the action (Avoid or Leaf), the URL prefix on which to apply that action, and an optional expiration date for the entry. This allows the user to completely avoid documents for which maintenance is not a concern or which could trap an unsuspecting spider (some forms of computational hypertext can have that effect).

4.3.4 Being friendly to service providers

A second design constraint for MOMspider is that it minimize its impact on information providers (destination servers) while at the same time maximizing the indirect benefits they receive from the traversal process. All HTTP requests are similar to:

```
HEAD /path HTTP/1.0
User-Agent:  MOMspider/0.1
From:  user@machine.sub.dom.ain
Referer:  http://www.site.edu/current/document.html
```

This allows server maintainers to properly recognize the source of the request and, if necessary, place restrictions upon a particular spider. It also provides them useful information, including how to contact the person running the spider and what document contains the reference being tested.

As an additional precaution, MOMspider periodically looks for and obeys any restrictions found in a site's /robots.txt document as per the standard proposed by Martijn Koster [Koster94]. Before any link is tested, the destination site is looked-up in a table of recently accessed sites (the definition of "recently" can be set by the user). If it is not found, that site's /robot.txt document is requested and parsed for restrictions to be placed on MOMspider robots. Any such restrictions are added to the user's avoid list and the site is added to the site table, both with expiration dates indicating when the site must be checked again. Although this opens the possibility for a discrepancy to exist between the restrictions applied and the contents of a recently changed /robots.txt document, it is necessary to avoid a condition where the site checks cause a greater load on the server than would the maintenance requests alone.

Chapter 5

The Image Database

5.1 Introduction

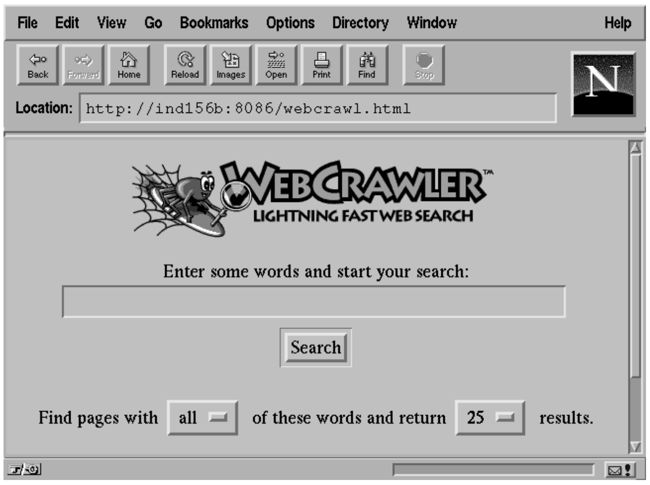
In the previous chapter we've discussed the concept of setting up a digital image database using a spider. This spider traverses selected pieces of the Web determined by its instruction file. These instruction files just don't come out of the blue.

5.2 Searching for images

First we assumed that we could just start the spider at one location and it would work its way through the whole Web. This seems to be an endless task as there are so many sites, of which you have no clue what information the spider will find. So we have decided to use multiple instruction files which only causes the spider to do a **Tree** traversal (see section 4.3.2 at page 35).

The first problem we encountered was how to determine which sites most likely contains digital images. For this answer we used an existing search engine "WebCrawler" (see figure 5.1). The reason for this particular search engine is that it was the only search engine that returned URLs of sites which most likely contains huge numbers of digital images.

With this information the creation of the instruction files for the spider could now start. This way we created 6,000 instruction files for the spider to traverse. However this spider retrieves more information than we actually need, therefore some alterations had to be made. It was also needed to write a program that can convert the data returned from the spider into a list containing the URLs of all the images found. But before we can start downloading these images we had to figure out how we want to store and index the images into the database.


File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://ind156b:8086/webcrawl.html
WEBCRAWLER
LIGHTNING FAST WEB SEARCH
Search
all
25

Enter some words and start your search:

Find pages with all of these words and return 25 results.

Figure 5.1: WebCrawler http://www.webcrawler.com

5.3 Thumbnails

Given the fact that digital images can be of any size and type we had to find a standard to store the images. So the amount of storage space per image had to be reduced in some way. The reduction of the storage space per image can be achieved by image compression and image scaling. The answer is the usage of thumbnail sized images as we've seen in Chapter 3. Of every image we would create a reduced 80x60 "thumbnail". This reduced image should fit (corresponding to its aspect ratio) within the 80x60 size (see figure 5.2). Small to medium databases, or even large ones can be manually viewed with a good fast browse of these "thumbnail" images (where 10 - 50 can be simultaneously displayed on a screen) and this is sufficient in many cases.

After determining the image size of a thumbnail image we also wanted to reduce the image storage space by using an image compression method. As we wanted to set up a large image database, this reduction should be significant, so we decided to use JPEG compression. JPEG compression also reduces image information, but this is slightly insignificant since the thumbnail images are just copies of the originals somewhere on the Web.

With the JPEG compression and the scaling techniques the average storage size in bytes per image is about 2Kbyte.

Figure 5.2: Example: left: the original image, middle: the 80x60 thumbnail image, right: the rescaled thumbnail image http://ind156b.wi.leidenuniv.nl:8086/summer.html

5.4 Database population

The database now created is just a projection of images found on the Web by our spider. The thumbnail images are copies of originally stored images somewhere on the Web. This brings us to another aspect concerning the database entries. Every database entry should contain at least the URL information of where the original image can be retrieved. Also information about the size of the image. Image size is needed to reproduce the image shown on your browser's screen. This is a sample of the database index:

```
image_tr_05#IMGINFO:76x78 #http://www.adventuresports.com/asap/norba/races/buffante.gif
image_tr_06#IMGINFO:167x252 #http://www.adventuresports.com/asap/norba/races/buff_002.gif
image_tr_07#IMGINFO:43x33 #http://www.adventuresports.com/asap/norba/races/norba_s.gif
image_tr_08#IMGINFO:32x32 #http://www.adventuresports.com/asap/norba/races/7home.gif
image_tr_09#BUILTIN:ERROR
```

For now we used three fields to store the image:

1. the entry field with the local filename of the thumbnail image; the format of the filename was chosen this way to be short and clear (e.g. `image_aa_00` is the first image and `image_ab_00` is the successor of `image_aa_99`). With this current format the storage of 100*26*26=67,600 images is possible.

2. the second field contains image information there are two possibilities:

   (a) the image information field denoted by "IMGINFO"; With this info an image can be displayed on the browser's screen at any scale, this was needed because showing an 80x60 thumbnail is simply too small.

   (b) the image information field denoted by "BUILTIN:ERROR"; This means an error has occurred during the downloading process, this is caused by several reasons, but we keep this entry in the database, because it corresponds with the same entry within the main URL list.

3. the URL; this is the location of the original image at that time (note: image URLs are subject to change over time).

A fourth field, the annotation field could easily be added to preserve descriptive keywords. At the moment the spider is unable to collect useful descriptional information about the localized image. The ultimate solution would be that the spider can collect significant textual descriptors localized near the image within the found HTML document.

The search speed within text files is rather low, because nothing can be said about the byte length per entry. Therefore we had to create an additional overlay index file. This file simply contains per entry the byte offset of that entry in the index file, by using a fixed size per entry in this overlay index file searching becomes unnecessary (by knowing the image identifier you can retrieve the byte offset immediately and thus the entry from the index file).

5.4.1 Adding images

To add an image to the database the following steps are performed:

- The spider must traverse a selected piece of the Web given by the instruction file
- The returned result file has to be examined and a temporary URL file is created
- The temporary URL file is then compared to the main URL file to see if there are multiple copies (e.g. identical URLs)
- The remaining unique URLs are then added to the main URL file from which the image retrieval (see Appendix E at page 81) will take place
- Finally when an image is successfully retrieved, scaled and compressed the entry will be added to the index file.

5.5 Future Work

To maintain a database of this size some aspects are easily overlooked. The main bottleneck is the expiration of URLs.

5.5.1 Removing and updating images

Sometimes downloading an image will fail, due to several reasons like, a time-out error occurred or no connection could be made to the server. This means that an error entry is made in the database indexfile. This downloading process for these error-like entries should be repeated in time.

Because URLs are subject to change – e.g. images are created, moved, or deleted by users, it would be handy if we could keep our local database up to date (e.g. thumbnail images should represent original images given its URL). This is quite an intensive task, since we have to rerun the spider to see if there are any changes made on the Web. Because this project is meant to be a demo, we didn't explore these problems. However future improvements can include making the database up to date every once in a while, making sure that images are present as claimed or new images are added.

Chapter 6

The Similarity Methods

6.1 Introduction

Now that we've created a large image database, some similarity methods had to be examined and implemented. Since we will use queries by example, some similarity methods based on color and shape are needed. We also used text matching to allow keyword searches on images. This section will specify the following three methods:

- Keyword matching
- Color histogram matching
- Edge oriented matching

6.2 Keyword matching

Fortunately searching text need not require understanding the text's meaning (Lycos simply extracts keywords using an algorithm that considers characteristics like word placement, word frequencies, etc). With this advantage it is easy to retrieve similar images based on descriptive keywords. In this project we've used the main URL list as a basis for our keyword database.

6.2.1 Keyword Database

From the URLs provided with every image keywords can easily be extracted. The only problem here is what is to be considered an useful keyword:

- the filename, this is the amount of text after the last "/" and before the last "`.`".

- several pathnames, this is the amount of text between two "/".

These pieces of text are not yet considered a keyword, because it can contain delimiters like "_" or numbers. So every piece of text is split up by these delimiters (any character which is not a letter). What is left can be considered a keyword if it has at least a length of 3 characters.

These keywords are then sorted and put in a keyword file which have the following layout:

```
base 9 46715
baseballtr 1 46783
basher 1 46791
basiccontrolbar 1 46799
basicinfo 1 46806
basin 2 46814
basket 83 46828
```

Every entry is divided by three fields:

1. The first field denotes the unique keyword, found within the URL as described before.
2. The second field contains the number of occurrences of this keyword as a whole.
3. The third field denotes an offset within the keyword index file.

The keyword index file contains offsets for every occurrence of the used keywords. The third field of the keyword file denotes the offset where the first occurrence of this particular keyword can be found in the keyword index file:

```
2224747
10406
117123
725493
738394
```

These offsets points to the corresponding positions in the main image index file where the image can be found containing that keyword. If this keyword has more occurrences then every successive offset points the the successive entry in the main image index file (see figure 6.1).

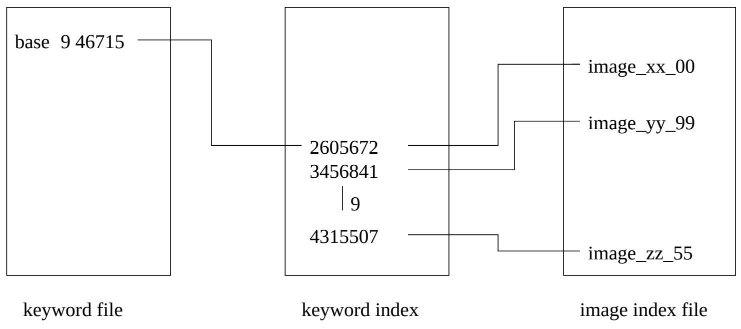


Figure 6.1: The keyword method

6.3 Color histogram matching

6.3.1 Introduction

A retrieval will fail if the query uses keywords which are not stored in the keyword database, even though the image is present. For example, an image with a woman strolling on a sunset beach will be referenced differently by different users. Some may use keywords such as “sunset”, others may use “beach”, “sand”, “lonely woman”, etc. In fact, it is rather difficult to accomplish flexible image retrieval using descriptive keyword approach. On the other hand, researchers on machine vision have yet to provide a solution on general object recognition from images. There is no way for us to automatically generate descriptive keywords from the images, thus text description of images all depends on human operators, which is an arduous task.

The labour involved with cataloging images by hand, and the difficulty of anticipating every user’s needs when assigning keywords to images, has led to the development of algorithms for retrieving images by their content. The goal of these algorithms is to quickly retrieve the images that are similar to a given image (example). The description of a color histogram matching method will be given in the next section.

6.3.2 The color space

Before we could start studying the algorithm we first had to determine which color space to use. The purpose of a color space is to facilitate the specification of colors in some standard, generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point. The color models most often used for image processing are the RGB (red, green, blue), and the HSI (hue, saturation, intensity) models. The RGB model was chosen in our case, since it can represent color and gray levels without problems and most image formats are based on the RGB model. One disadvantage of the RGB model is that we are dealing with a 3 dimensional color space.

6.3.3 Color Histograms

Given our discrete color space defined by the color axes (e.g. red, green and blue), the color histogram is obtained by discretizing the image colors and counting the number of times each discrete color occurs in the image array. One factor to be considered when creating the histogram is how finely the color space should be discretized. Various researchers have proven that human eyes are not as sensitive to colors as to brightness. Thus it might not be the case that the more finely we discretize the color space, the better histogram we have; while finer histograms require more memory space and more computation. Our experimental results have shown that color distribution of an image is well preserved when the image is discreted by 8x8x8 thus we choose to discretize the RGB color space into 512 equally sized cubes (see figure 6.2).

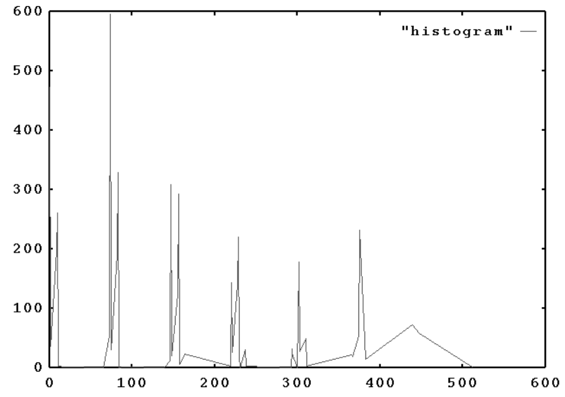


Figure 6.2: Sample of a RGB color histogram

Color histograms holds information on color distribution, but it lacks information on color locations. This may lead to the situation in which an image with a red balloon on top is matched to an image with a red car in the bottom. In addition, a histogram for the entire image tends to miss small image regions that will not produce strong peaks in the histogram.

6.3.4 The indexing algorithm

For this project we chose to define the color histogram to be an one dimensional vector. However the color space we use (RGB), is three dimensional. The problem that arises here is the correlation of color shades. In our model different shades of one color will not be correlated as in the 3-D model. By discretizing the color space from 256x256x256 colors to 8x8x8 colors we eliminate fine shades, and only the most significant colors will be preserved. A discrete color is defined as:

$$color_j = \sum_{i=1}^{3} x_i 8^i \qquad x \in [1,8], j \in [1,512] \tag{6.1}$$

where x represents the corresponding RGB color component. In this case the R-color component represents the least significant component, and the B-color component represents the most significant component.

The histogram matching algorithm we analyze in this section is essentially the same as those presented in [Swain and Ballard 1991].

The color histogram *H*(*M*) is defined as a vector $(h_1, h_2, ..., h_n)$ in a n-dimensional vector space, where each element h_j represents the percentage (of the total number of pixels in the image *M*) used of color *j* in the image *M*. These histograms are the feature vectors to be stored as the index of the database.

To measure the distance d between two histograms H and I one can use the metric included by the L_1-norm (6.2) as in [Swain and Ballard 1991] or a metric which is similar to the one induced by the L_2-norm (6.3) (see [Niblack *et al.*. 1993]).

$$d_{L_1}(I,H) = \|I - H\|_{L_1} = \sum_{l=1}^{n} |i_l - h_l| \tag{6.2}$$

$$d_{L_2}(I,H) = \|I - H\|_{L_2} = \sqrt{\sum_{l=1}^{n} (i_l - h_l)^2} \tag{6.3}$$

The L_1-distance between two histograms is always less than 2 and the L_2-distance is less than $\sqrt{2}$.

6.3.5 Indexing

Given these norms we chose to use the L_1-distance (6.2), due to performance reasons and the size of the database. To implement this method we used a testset of 30,000 images as part of the total image database. The difference measures are stored in a distance matrix (6.4).

$$\begin{pmatrix} dist_{1,1} & dist_{1,2} & dist_{1,3} & dist_{1,4} & \cdots & dist_{1,n} \\ dist_{2,1} & dist_{2,2} & dist_{2,3} & dist_{2,4} & & \\ dist_{3,1} & dist_{3,2} & dist_{3,3} & dist_{3,4} & & \\ dist_{4,1} & dist_{4,2} & dist_{4,3} & dist_{4,4} & & \vdots \\ \vdots & & & & \ddots & \\ dist_{n,1} & & \cdots & & & dist_{n,n} \end{pmatrix} \tag{6.4}$$

To speed up the calculations we made use of the fact that distances positioned on the diagonal of the matrix are not to be calculated since there is no use of measuring a distance with itself. We also used the fact that the distance matrix is symmetric (see 6.5).

$$\begin{pmatrix} xx & xx & xx & xx & xx & xx \\ dist_{2,1} & xx & xx & xx & xx & xx \\ dist_{3,1} & dist_{3,2} & xx & xx & xx & xx \\ dist_{4,1} & dist_{4,2} & dist_{4,3} & xx & xx & xx \\ \vdots & & & \ddots & xx & xx \\ dist_{n,1} & & \cdots & & dist_{n,n-1} & xx \end{pmatrix} \tag{6.5}$$

Instead of storing the whole matrix we sorted every column and stored 10 of the smallest distances per column. This way of storing the distances not only reduces space but also makes it easily extendable for new entries.

6.4 Edge oriented matching

In this project we tried to develop successful edge oriented matching techniques in two ways. In the first edge oriented matching techniques was based on intensity space and the second was based on gradient space. There is no way telling which method gives the best results, therefore those two techniques were implemented and evaluated.

6.4.1 Intensity space

For every pixel of an image in RGB format we calculated the Intensity. For any three R, G, and B color components, each in the range [0, 1], the intensity component in the HSI model is defined as:

$$I = \frac{1}{3}(R + G + B) \tag{6.6}$$

which yields values in the range [0, 1].

6.4.2 Gradient space

The most common method of differentiation in image processing in applications is the gradient. For a function $f(x, y)$, the gradient [Gonzalez and Woods] of f at coordinates (x, y) is defined as the vector:

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \\ \frac{\delta f}{\delta y} \end{bmatrix} \tag{6.7}$$

It is well known from vector analysis that the gradient vector points in the direction of maximum rate of change of f at (x, y). In edge direction an important quantity is the magnitude of this vector, generally referred to simply as the *gradient* and denoted ∇f. Common practice is to approximate the gradient with absolute values:

$$\nabla f \approx |G_x| + |G_y| \tag{6.8}$$

which is much simpler to implement, particularly with dedicated hardware.

Note from Eqs. (6.7) and (6.8) that computation of the gradient of an image is based on obtaining the partial derivatives $\delta f/\delta x$ and $\delta f/\delta y$ at every pixel location. Derivatives may be implemented in digital form in several ways. However, the Sobel operators have the advantage of providing both a differencing and a smoothing effect. Because derivatives enhance noise, the smoothing effect is a particularly attractive feature of the Sobel operators. From Figure 6.3, derivatives based on the Sobel operator masks are

$$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{6.9}$$

and

$$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{6.10}$$

where, as before, the z's are the gray levels of the pixels overlapped by the masks at any location in an image.

Computation of the gradient at the location of the center of the masks then utilizes Eq. 6.8, which gives one value of the gradient. To get the next value, the masks are moved to the next pixel location and the procedure is repeated. Thus, after the procedure has been completed for all possible locations, the result is a gradient image of the same size as the original image. As usual, mask operations on the border of an image are implemented by using the appropriate partial neighborhoods.

z 1	z 2	z 3
z 4	z 5	z 6
z 7	z 8	z 9

(a)

- 1	- 2	- 1
0	0	0
1	2	1

(b)

- 1	0	1
- 2	0	2
- 1	0	1

(c)

Figure 6.3: (a) 3x3 image region; (b) mask used to compute G$_x$ at center point of the 3x3 region; (c) mask used to compute G$_y$ at that point. These masks are often referred to as the Sobel operators.

6.4.3 Horizontal and Vertical Projections

Now as we have discussed the two edge detection algorithms we need to find a way to create a feature vector. First step is to scale all the images to contain the same number of pixels (e.g. 80x60 pixels), only for comparison purposes. This way we can easily use the L_1-norm (see (6.2)). Matching with projections is particularly handy in this case because it makes the size of the feature vector only 80+60 instead of 80*60 elements. We map the elements into discrete values in the range of [0,255]. The HV projection $P(M)$ is a vector $(p_{x_1},p_{x_2},...,p_{x_{60}},p_{y_1},p_{y_2},...,p_{y_{80}})$ where

$$p_{x_j} = \sum_{k=1}^{x} \mathbf{V}(k,j) \qquad j \in [1,y] \tag{6.11}$$

and

$$p_{y_i} = \sum_{l=1}^{y} \mathbf{V}(i,l) \qquad i \in [1,x] \tag{6.12}$$

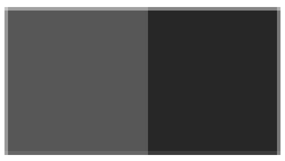
Here is **V** the appropriate value of one of the two edge detection algorithms per pixel. As discussed in section 6.3.5 the indexing algorithm stores its results the same way as discussed before (see Eq. 6.5).

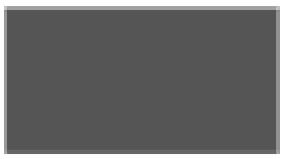
6.5 Evaluation

A preliminary evaluation suggests several problems with the retrieval techniques. The system was evaluated with a database testset of 30,000 images of any kind (as discussed in chapter 5).

6.5.1 Color histogram matching

The color histogram method, which is based on averaging of color distribution, returns satisfactory results for the database population. However, this means that this method lacks information on color locations as we have discussed before. We can overcome this problem by dividing the image into several subareas. Then for each subarea compute a color histogram. The similarity score will be determined by the weighted average of the subarea similarity scores.


(a) the original image


(b) the intensity image

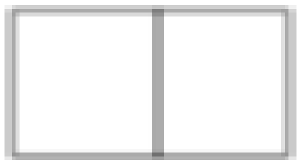

(c) the (inverted)gradient image

Figure 6.4: Intensity space vs Gradient space

6.5.2 Edge oriented matching

The main disadvantage of using the intensity method for detecting edges is shown in figure 6.4. The intensity method may miss color differences. In this example we used a simple testimage where two blocks are adjacent and different of color (see figure 6.4(a)). The intensity method just analyzes the color intensities and detects no difference in intensity when passing from one block to another (see figure 6.4(b)). However if we use the gradient method these kinds of edges will be detected (see figure 6.4(c)).

However the intensity method seems to work better than the gradient method in most cases.

6.5.3 Test-set testing

To obtain a more constructive evaluation of the methods a test-set approach should be considered. In this case a test-set can be constructed by using a keyword query. The results of this query could be used as a test-set (e.g. using the keyword "glass" or "lion" will return usable results). With this test-set the results of the applied methods can be compared and evaluated. (an example is given in figure 7.1 at page 55 with keyword "glass" and substring matching)

Chapter 7

Image Search Engine

7.1 Introduction

Web image search engines could be applied profitably in many areas; for instance, in searching on-line catalogs of consumer goods and services, or for enforcing image copyrights by sniffing out unauthorized copies on the Web. Such a Web crawler would also be useful to researchers studying image databases, serving as a very large testbed for image database indexing methods.

Picture yourself as a fashion designer needing images of fabrics with particular mixture of colors, a museum cataloger looking for artifacts of a particular shape or a photostudio artist needing a picture of a red car-like object. How do you find these images?

In this chapter the reader will get a description of an content-based image search engine that allows queries to be performed based on example images (as we've discussed earlier in chapter 3). The similarity methods involved are based on keyword matching, color histogram matching, and edge oriented matching (see chapter 6).

7.2 User Interface

Now that we have the database and the similarity results the user interface can be implemented. As querying is based on selecting example images, we chose the following approach. A query specification can include a keyword, and an example image selection. The user interface to support such queries has two main parts: The Query Result and Selection Window, and The Query Control Panel Window to display the results.

7.2.1 The Query Result and Selection Window

This is the area where all the query results are displayed. This area is also used for selecting a new image for the new query. This area is divided into four different areas:

- The search image; The left column is reserved to display the search image. This is the image selected by the user, a default image, or the first image from the returned result set of a keyword search.
- The search row; The top row displays the images returned by the returned result set of a keyword search, or just a sequence of images from the database. Note that the first (left) image is the same as the search image.
- The color row; The middle row displays the images as a result of color histogram matching. Note that the left image implies to be the best (or exact) match. All the images presented are results of color matching regarding the search image. The matching percentages are also given (100% means an exact match).
- The Shape row; The bottom row displays the images as a result of edge oriented matching (depends which shape method is chosen).

Every image displayed in the area (except for the search image) is a potential member for a new query. Also the original URL is given for every image displayed, so the user can retrieve the original image (normal size) (See figure 7.1).

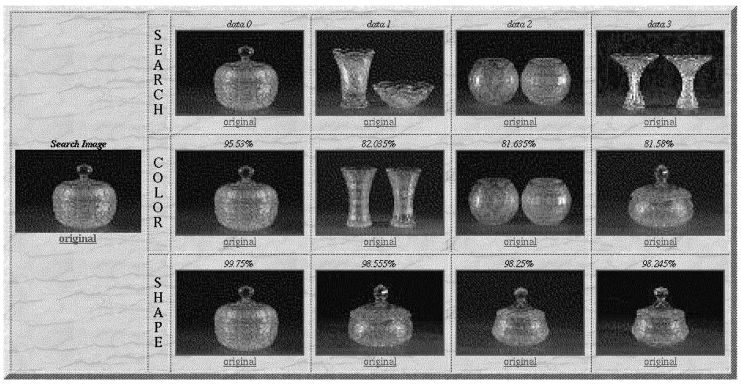


Figure 7.1: The image selection area

To start a new query by example the user simply can select one of the given images displayed, by clicking with the mouse. Also the number of matching hits, with respect to keyword matching is given on top of the result and selection area. If no keyword match was specified than the current total number of images in the database is given (See figure 7.2).

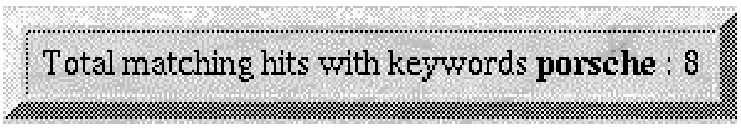

Total matching hits with keywords **porsche** : 8

Figure 7.2: The number of matching hits

7.2.2 The Query Control Panel Window

Within the control panel the user can modify a certain number of parameters regarding the image similarity methods and display settings.

Changing the number of columns

With this option the user can specify the number of columns being displayed by selecting one of the given values from the popup menu. The range of this selection lies within 0 and 10 (See figure 7.3).

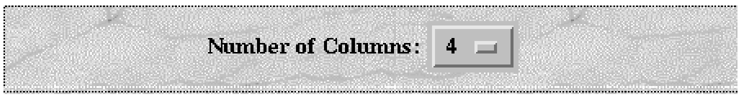

Number of Columns: 4

Figure 7.3: The number of columns field

The Keyword matching field

This field is divided into two parts:

1. The first part contains a small text field in which the user can type in the desired keyword.
2. The second part contains a popup-menu from which the user can select what type of matching should be performed, there are two kinds of matching:
    (a) Normal matching; the keyword matching algorithm will try to find the specified keyword as a whole word.
    (b) Substring matching; the keyword matching algorithm will try to find the specified keyword as part of a string.

(See figure 7.4)

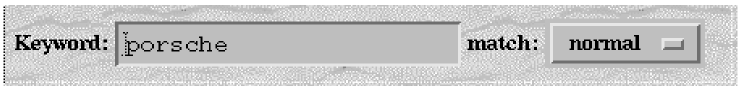

Keyword: porsche match: normal

Figure 7.4: The keyword matching field

The Shape method field

With this field the user can select the desired shape similarity method (see section 6.4 at page 49). This is also presented as a popup menu with the two methods as options (See figure 7.5).

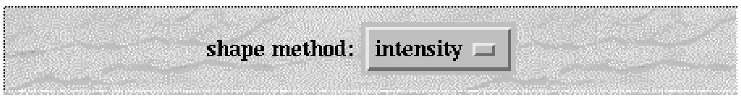

shape method: intensity

Figure 7.5: The Shape method field

The start search field

This is simply a submit button with which the user can start his or her query specified by the selected parameters. Note that starting a query can be done in three different ways:

1. By clicking the start search button.
2. By pressing enter after typing in a keyword.
3. By selecting an example image from the displayed images within the display area.

(See figure 7.6)

Figure 7.6: The start search field

Additional fields

Some additional fields are created for future purposes:

- Guestbook option; with this option the user can fill out an electronic form in which the user can leave its comments about the demo, this message is then added to the guestbook which is viewable for everyone.

- Visitor counter; this will display the number of different users that have visited this page (for statistical purposes).
- Add your own image option; with this option the user can fill out an electronic form with which the user can specify his or her own image which will be added to the database. The interface has already been build but images submitted are not yet processed and added to the main database. This option is still under construction.

(See figure 7.7)

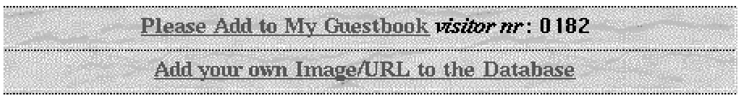

Please Add to My Guestbook
visitor nr: 0182
Add your own Image/URL to the Database

Figure 7.7: Additional fields

The control panel

Finally we grouped all these options to one control panel (See figure 7.8).

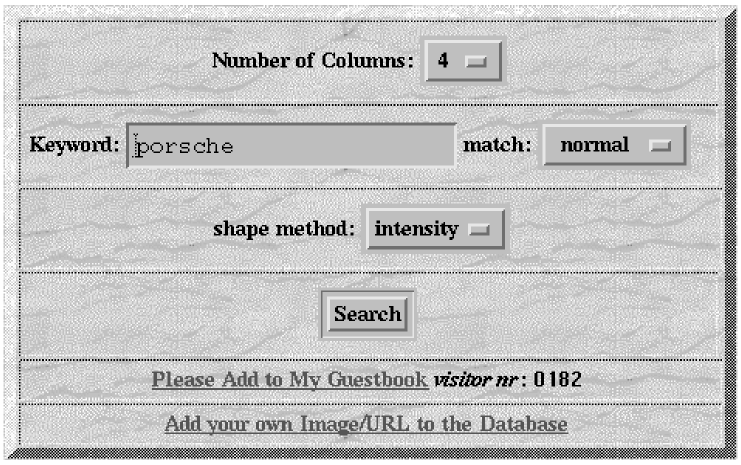
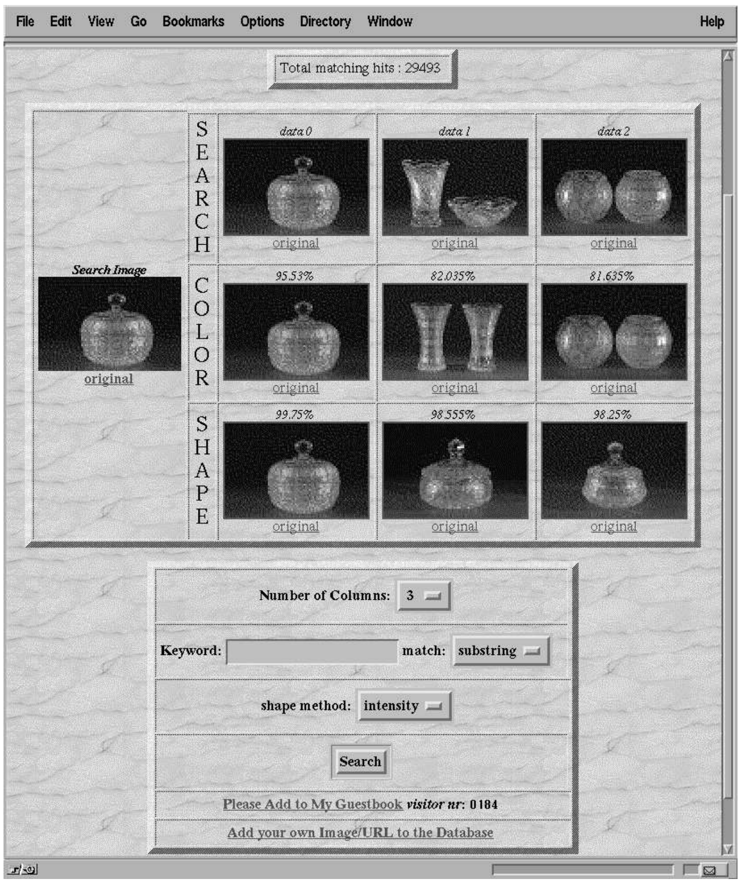


Figure 7.8: The control panel

The user interface layout

Combining the query result and selection window and the query control panel window the user interface was given the following layout (see figure 7.9):



Total matching hits : 29493

Number of Columns: 3

Keyword:

match: substring

shape method: intensity

visitor nr: 0184

Figure 7.9: The layout of the user interface

7.3 Conclusion

With the query result and selection window and the query control panel window, the user is now able to define his or her queries by adjusting the parameters and selecting an image or the search-button. This search-engine is also set up to be easily extendable with extra features and new future methods.

Chapter 8

Conclusion

As we have discussed before (see sections 3.5 and 7.3), these demos work properly. Although some improvements can be made. Extra features can easily be implemented and result files of new methods can be added.

8.1 Future Work

8.1.1 Color

Subarea division

The color retrieval technique has several weaknesses that were discussed in the evaluation (see section 6.5 at page 52. First the color method deals with the image as a whole rather than dividing the image into subareas. This way the color histogram is then determined for each subarea and the the color similarity score is just a weighted average of the subarea similarity scores.

Exact color match

The second problem with the color retrieval is that histogram intersection performs an exact color match. It must be modified to allow inexact color match so that one shade of green will match similar shades of green. This is relatively simple in the case of histogram intersection.

8.1.2 Edges

Edge detection

The edge detection algorithm does not construct good edge maps for every image as discussed in implementation and evaluation sections. There are two main problems with edge maps. First the edge maps for some images contain "extraneous" edges that carry no useful information for retrieval purposes even though they are perceptually prominent in the image. Some of these extraneous edges arise due to "color noise" (i.e. a small group of pixels whose color is sharply different than all their neighbors. More aggressive median filtering and a thinning procedure to strip out the shortest edges will eliminate some of these extraneous edges. In addition we should experiment with the gradient threshold to see if a threshold will give better results (e.g. keep only those edge pixels whose gradient strength is greater than the average plus two standard deviations). However preliminary experimentation suggests that increasing the threshold will eliminate not only some of the extraneous edges but portions of good edges as well.

8.1.3 Keywords

Keyword matching

The keyword matching method can be improved and extended:

- Usage of associative arrays, so searching will become unnecessary.
- Usage of the Data Base Management (DBM) library. Especially when the database significantly increases.
- Usage of a thesaurus, to locate alternative matches.
- Usage of user provided annotation fields to extend the keyword database.

8.1.4 User Interface

The user interface can be modified easily to the needs of new applications. Actually the user interface created for the first demo (LCPD) is designed to create general user interfaces within this application field.

8.1.5 Database

An incremental approach is used for database population growth. Also the actual comparison procedures made use of this incremental approach. Previous results are stored and used for newer calculations. Some aspects that can be improved:

- Usage of associative arrays, for the index files.
- Usage of the Data Base Management (DBM) library, for the index files.

Bibliography

[URL1996] Tim Berners-Lee, Uniform Resource Locators, Internet working draft, 1996/05/24, Published on the WWW at http://www.w3.org/pub/WWW/Addressing/URL/Overview.html

[SGML] ISO 8879, Information Processing - Text and Office systems - Standard Generalized Markup Language (SGML), 1986. http://www.iso.ch/cate/d16387.html

[HTML] Tim Berners-Lee and Daniel Connolly, Hypertext Markup Language, Internet working draft, 13 jul 1993. Published on the WWW at http://www.w3.org/pub/WWW/MarkUp/HTML.html

[HTTP] Tim Berners-Lee, Hypertext Transfer Protocol, Internet working draft, 5 nov 1993, published on the WWW at http://www.w3.org/pub/WWW/Protocols/HTTP/HTTP2.html

[Perl] David Till, *Teach Yourself PERL in 21 days*, 1995 by Sams Publishing, ISBN 0-672-30586-0

[CGI] Ed Tittel, Mark Gaither, Sebastian Hassinger, Mike Erwin, *World Wide Web Programming with HTML and CGI*, 1995 by IDG Books Worldwide, Inc., ISBN 1-56884-703-3

[Koster94] Martijn Koster, A Standard for Robot Exclusion, published on the WWW at http://info.webcrawler.com/mak/projects/robots/norobots.html

[MOMspider] Roy T.Fielding, Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web, June 17, 1994, published on the WWW at http://www.ics.uci.edu/WebSoft/MOMspider/WWW94/paper.html

[Huijsmans] D.P.Huijsmans, M.S.Lew, Efficient Content-based Image Retrieval in Digital Picture Collections using projections: (Near)-Copy location, Januari 11 1996, Technical Report 96-07

[Gonzalez and Woods] Rafael C. Gonzalez, Richard E. Woods, *Digital Image Processing*, 1992 Addison Wesley, ISBN 0-201-50803-6

[Swain and Ballard 1991] M.J. Swain and D.H. Ballard, Color indexing, Intern. Journal of Computer Vision, 7(1):11-32, 1991

[Niblack *et al.*. 1993] W. Niblack, R. Barber, et al. The QBIC project: Querying images by content using color, texture and shape, In Proc. of SPIE Electronic Imaging: Storage and Retrieval for Image and Video Databases, Feb. 1993

Appendix A HTML

As HTML is an application of SGML (Standard Generalized Markup Language), this specification assumes a working knowledge of [SGML].

Tags delimit elements such as headings, lists, character highlighting, and links. Most HTML elements are identified in a document as a start-tag, which gives the element name and attributes, followed by the content, followed by the end tag. Start-tags are delimited by < and >; end tags are delimited by </ and >. An example is:

```
<H1>This is a Heading</H1>
```

Some elements only have a start-tag without an end-tag. For example, to create a line break, use the `<BR>` tag. Additionally, the end tags of some other elements, such as Paragraph (`</P>`), List Item (`</LI>`), Definition Term (`</DT>`), and Definition Description (`</DD>`) elements, may be omitted.

The content of an element is a sequence of data character strings and nested elements. Some elements, such as anchors (i.e. one of two ends of a hyperlink), cannot be nested. Anchors and character highlighting may be put inside other constructs. The basic HTML document has the following structure:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML>
<HEAD>
<TITLE>Introduction to HTML</TITLE>
</HEAD>
<BODY>
<H1>Important Stuff</H1>
<P>Explanation about important stuff ....
</BODY>
</HTML>
```

How to deal with images? The IMG element allows an image file to be inserted within an HTML document along with the text. The ALT attribute defines parsed character data that will be displayed if the image is not or cannot be displayed by the browser. The SRC attribute identifies an URL for retrieving the image. If a single A element spans both an image and text, the CGI program will receive the HREF input values, if any, if the text is selected, or the x,y cursor pixel coordinates relative to 0,0 as the the upper-left corner of the image if the image is selected. Here is a simple example:

```
<IMG SRC="http://ind156b.wi.leidenuniv.nl:8086/images/picture.gif">
```

This will display an image on the Browser's screen! To make an image clickable you just have to anchor a link to it:

```
<A HREF="http://ind156b.wi.leidenuniv.nl:8086/sounds/sapper.au">
<IMG SRC="http://ind156b.wi.leidenuniv.nl:8086/images/picture.gif">
</A>
```

This will create a clickable image on the Browser's screen. After selecting the image by clicking on it the corresponding link will be followed, and in this case a soundsample will be played.

Using Tables. For making the result documents for the search engine the way they are I used new HTML 3.0 elements such as Tables. Here are some simple table guidelines, lifted from the HTML 3.0 specification:

- Table cells may include nested tables
- Missing cells are considered to be empty.
- Missing rows should be ignored; that is, if all cells spans a row and there are no further <TR> elements, then the implied row should be ignored.
- Cells cannot overlap.

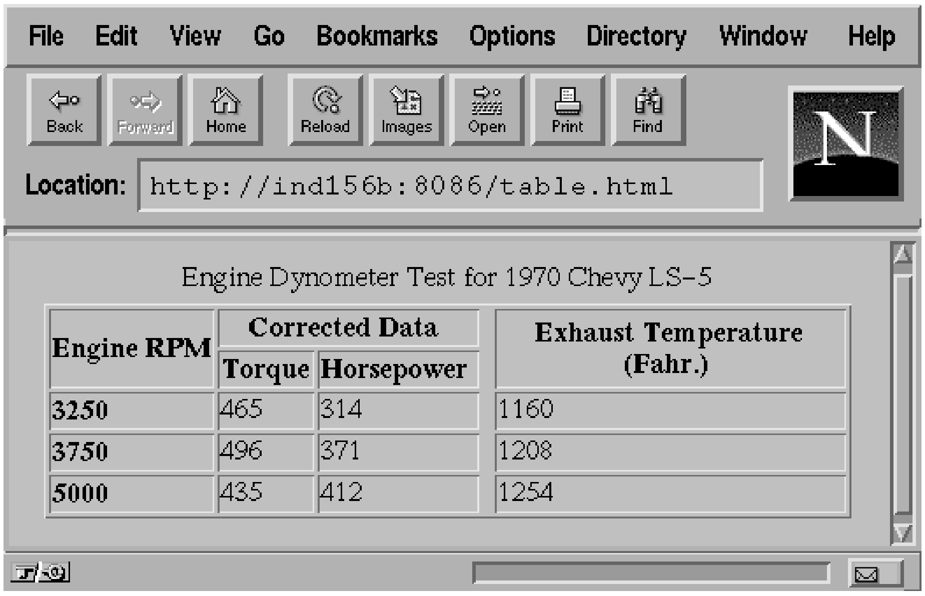
What kind of data can a table contain? Some examples:

- lists
- paragraphs
- forms
- figures

- headers
- other tables
- preformatted text

Let's take a look at an example of an HTML 3.0 table instance:

```
<TABLE BORDER>
<CAPTION>Engine Dynometer Test for 1970 Chevy 454 LS-5</CAPTION>
<TR><TH ROWSPAN=2>Engine RPM<TH COLSPAN=2>Corrected Data
<TH ROWSPAN=2>
<TH ROWSPAN=2>Exhaust Temperature (Fahr.)
<TR><TH>Torque<TH>Horsepower
<TR><TH ALIGN="LEFT">3250<TD>465<TD>314<TD><TD>1160
<TR><TH ALIGN="LEFT">3750<TD>496<TD>371<TD><TD>1208
<TR><TH ALIGN="LEFT">5000<TD>435<TD>412<TD><TD>1254
</TABLE>
```



http://ind156b:8086/table.html

Engine Dynometer Test for 1970 Chevy LS-5

Engine RPM	Corrected Data			Exhaust Temperature (Fahr.)
	Torque	Horsepower		
3250	465	314		1160
3750	496	371		1208
5000	435	412		1254

Figure 8.1: Netscape's rendering of a sample HTML 3.0 table

For **Forms** creation and details see Appendix C at page 69.

Appendix B Perl

I assume that Perl is installed on your system, if not see Appendix F at page 83 for URL information.

A Sample Perl Program:

Assuming that Perl is located in `/usr/local/bin` `#!/usr/local/bin/perl`

```
$inputline = <STDIN>;
print ("$inputline");
```

Line 1 is the header comment. Line 2 reads a line of input. Line 3 writes the line of input back to your screen.

Running a Perl program. To run the program shown in the previous listing, do the following:

1. Using your favorite editor, type the previous program and save it in a file called "foo.pl".
2. Tell the system that this file contains executable statements. To do this in the UNIX environment, enter the following command:
   `chmod +x foo.pl`
3. Run the program by entering the command:
   `foo.pl`

What is a Scalar Value?

Basically, a *scalar value* is one unit of data. This unit of data can be either a number or a chunk of text. There are several types of scalar values that Perl understands:

- Integers
- Floating-point numbers

- Character strings

These scalar values are interchangeable, that is, you can use a scalar variable that was assigned as a character string also as an integer and vice versa:

```
$string = "43";
$number = 28;
$result = $string + $number;
```

The value of $string is converted to an integer and added to the value of $number. The result of the addition, 71, is assigned to $result.

See [Perl] for more detailed information about Perl programming.

Appendix C CGI.pm

"CGI.pm" is a Perl5 library for handling forms in CGI scripts. With just a handful of calls, you can parse CGI queries. However, it also offers a rich set of functions for creating fill-out forms. Instead of remembering the syntax for HTML form elements, you just make a series of Perl function calls. An important fringe benefit of this is that the value of the previous query is used to initialize the form, so that the state of the form is preserved from invocation to invocation.

Everything is done through a "CGI" object. When you create one of these objects it examines the environment for a query string, parses it, and stores the results. You can then ask the CGI object to return or modify the query values. CGI objects handle POST and GET methods correctly, and distinguish between scripts called from <ISINDEX> documents and form-based documents. In fact you can debug your script from the command line without worrying about setting up environment variables.

If you want to install CGI.pm yourself please see Appendix F at page 83 for URL information.

Creating a new CGI object The most basic use of CGI.pm is to get at the query parameters submitted to your script. To create a new CGI object that contains the parameters passed to your script, put the following at the top of your perl CGI programs:

```
use CGI;
$query = new CGI;
```

In the object-oriented world of Perl 5, this code calls the new() method of the CGI class and stores a new CGI object into the variable named $query. The new() method does all the dirty work of parsing the script parameters and environment variables and stores its results in the new object. You'll now make method calls with this object to get at the parameters, generate form elements, and do other useful things.

An alternative form of the new() method allows you to read script parameters from a previously-opened file handle:

```
$query = new CGI(FILEHANDLE)
```

The filehandle can contain a URL-encoded query string, or can be a series of newline delimited TAG=VALUE pairs. This is compatible with the save() method. This lets you save the state of a CGI script to a file and reload it later. It's also possible to save the contents of several query objects to the same file, either within a single script or over a period of time. You can then reload the multiple records into an array of query objects with something like this:

```
open (IN,"test.out") || die;
while (!eof(IN))
{
   my $q = new CGI(IN);
   push(@queries,$q);
}
```

Fetching The Names Of All The Parameters Passed To Your Script

```
@names = $query->param
```

If the script was invoked with a parameter list (e.g. "name1=value1 & name2=value2 & name3=value3"), the param() method will return the parameter names as a list. For backwards compatibility, this method will work even if the script was invoked as an <ISINDEX> script: in this case there will be a single parameter name returned named "keywords".

Fetching The Value(s) Of A Named Parameter

```
@values = $query->param('foo');
    -or-
$value = $query->param('foo');
```

Pass the param() method a single argument to fetch the value of the named parameter. If the parameter is multivalued (e.g. from multiple selections in a scrolling list), you can ask to receive an array. Otherwise the method will return a single value.

As of version 1.50 of this library, the array of parameter names returned will be in the same order in which the browser sent them. Although this is not guaranteed to be identical to the order in which the parameters were defined in the fill-out form, this is usually the case.

Setting The Value(s) Of A Named Parameter

```
$query->param('foo','an','array','of','values');
   -or-
$query->param(-name=>'foo',-values=>['an','array','of','values']);
```

This sets the value for the named parameter 'foo' to one or more values. These values will be used to initialize form elements, if you so desire. Note that this is the one way to forcibly change the value of a form field after it has previously been set.

Appending a Parameter

```
$query->append(-name=>'foo',-values=>['yet','more','values']);
```

This adds a value or list of values to the named parameter. The values are appended to the end of the parameter if it already exists. Otherwise the parameter is created.

Deleting a Named Parameter Entirely

```
$query->delete('foo');
```

This deletes all the parameters and leaves you with an empty CGI object. This may be useful to restore all the defaults produced by the form element generating methods.

Importing parameters into a namespace

```
$query->import_names('R');
print "Your name is $R::name\n"
print "Your favorite colors are @R::colors\n";
```

This imports all parameters into the given name space. For example, if there were parameters named 'foo1', 'foo2' and 'foo3', after executing `$query->import_names('R')`, the variables `@R::foo1`, `$R::foo1`, `@R::foo2`, `$R::foo2`, etc. would conveniently spring into existence. Since CGI has no way of knowing whether you expect a multi- or single-valued parameter, it creates two variables for each parameter. One is an array, and contains all the values, and the other is a scalar containing the first member of the array. Use whichever one is appropriate. For keyword (a+b+c+d) lists, the variable @R::keywords will be created.

If you don't specify a name space, this method assumes namespace "Q".

Warning: do not import into namespace "main". This represents a major security risk, as evil people could then use this feature to redefine central variables such as @INC. CGI.pm will exit with an error if you try to do this.

Note: this method used to be called import(). As of version 2.20 import() has been changed to be consistent with other Perl modules. Please change all occurrences of import() to import_names().

Creating the HTTP Header

Creating the Standard Header for a Virtual Document

```
print $query->header('image/gif');
```

This prints out the required HTTP Content-type: header and the requisite blank line beneath it. If no parameter is specified, it will default to 'text/html'.

An extended form of this method allows you to specify a status code and a message to pass back to the browser:

```
print $query->header(-type=>'image/gif', -status=>'204 No Response');
```

This presents the browser with a status code of 204 (No response). Properly-behaved browsers will take no action, simply remaining on the current page. (This is appropriate for a script that does some processing but doesn't need to display any results, or for a script called when a user clicks on an empty part of a clickable image map.)

Several other named parameters are recognized. Here's a contrived example that uses them all:

```
print $query->header(-type=>'image/gif',
                     -status=>'402 Payment Required',
                     -expires=>'+3d',
                     -cookie=>$my_cookie,
                     -Cost=>'$0.02');
```

HTML Shortcuts

Creating an HTML Header

```
print $query->start_html(-title=>'Secrets of the Pyramids',
                -author=>'fred@capricorn.org',
                -base=>'true',
                -meta=>{'keywords'=>'pharoah secret mummy',
                        'copyright'=>'copyright 1996 King Tut'},
                -BGCOLOR=>'blue');
```

This will return a canned HTML header and the opening <BODY> tag. All parameters are optional:

- The title (**-title**)
- The author's e-mail address (will create a <LINK REV="MADE"> tag if present (**-author**)
- A true flag if you want to include a <BASE> tag in the header (**-base**). This helps resolve relative addresses to absolute ones when the document is moved, but makes the document hierarchy non-portable. Use with care!
- A **-xbase** parameter, if you want to include a <BASE> tag that points to some external location. Example:

  ```
  print $query->start_html(-title=>'Secrets of the Pyramids',
                  -xbase=>'http://www.nile.eg/pyramid.html');
  ```

- A **-meta** parameter to define one or more <META> tags. Pass this parameter a reference to an associative array containing key/value pairs. Each pair becomes a <META> tag in a format similar to this one.

  ```
  <META NAME="keywords" CONTENT="pharoah secret mummy">
  <META NAME="description" CONTENT="copyright 1996 King Tut">
  ```

  There is no support for the HTTP-EQUIV type of <META> tag. This is because you can modify the HTTP header directly with the header method.

- A **-script** parameter to define Netscape JavaScript functions to incorporate into the HTML page. This is the preferred way to define a library of JavaScript functions that will be called from elsewhere within the page. CGI.pm will attempt to format the JavaScript code in such a way that non-Netscape browsers won't try to display the JavaScript code. Unfortunately some browsers get confused nevertheless. Here's an example of how to create a JavaScript library and incorporating it into the HTML code header:

```
$query = new CGI;
print $query->header;

$JSCRIPT=<<END;
// Ask a silly question
function riddle_me_this() {
   var r = prompt("What walks on four legs in the morning, " +
                  "two legs in the afternoon, " +
                  "and three legs in the evening?");
   response(r);
}
// Get a silly answer
function response(answer) {
   if (answer == "man")
         alert("Right you are!");
   else
         alert("Wrong!  Guess again.");
}
END

print $query->start_html(-title=>'The Riddle of the Sphinx',
                          -script=>$JSCRIPT);
```

- **-onLoad** and **-onUnload** parameters to register JavaScript event handlers to be executed when the page generated by your script is opened and closed respectively. Example:

```
print $query->start_html(-title=>'The Riddle of the Sphinx',
                          -script=>$JSCRIPT,
                          -onLoad=>'riddle_me_this()');
```

  See JavaScripting for more details.

- Any additional attributes you want to incorporate into the <BODY> tag (as many as you like). This is a good way to incorporate other Netscape extensions, such as background color and wallpaper pattern. (The example above sets the page background to a vibrant blue.) You can use this feature to take advantage of new HTML features without waiting for a CGI.pm release.

Ending an HTML Document

```
print $query->end_html
```

This ends an HTML document by printing the </BODY></HTML> tags.

Creating Forms

General note 1. The various form-creating methods all return strings to the caller. These strings will contain the HTML code that will create the requested form element. You are responsible for actually printing out these strings. It's set up this way so that you can place formatting tags around the form elements.

General note 2. The default values that you specify for the forms are only used the first time the script is invoked. If there are already values present in the query string, they are used, even if blank.

If you want to change the value of a field from its previous value, you have two choices:

1. call the **param()** method to set it.
2. use the **-override** (alias **-force**) parameter. (This is a new feature in 2.15) This forces the default value to be used, regardless of the previous value of the field:

   ```
   print $query->textfield(-name=>'favorite_color',
                           -default=>'red',
                           -override=>1);
   ```

If you want to reset all fields to their defaults, you can:

1. Create a special defaults button using the defaults() method.
2. Create a hypertext link that calls your script without any parameters.

General note 3. You can put multiple forms on the same page if you wish. However, be warned that it isn't always easy to preserve state information for more than one form at a time. See advanced techniques for some hints.

General note 4. By popular demand, the text and labels that you provide for form elements are escaped according to HTML rules. This means that you can safely use "<CLICK ME>" as the label for a button. However, this behavior may interfere with your ability to incorporate special HTML character sequences, such as &Aacute; (Á) into your fields. If you wish to turn off automatic escaping, call the `autoEscape()` method with a false value immediately after creating the CGI object:

```
$query = new CGI;
$query->autoEscape(undef);
```

You can turn autoescaping back on at any time with `$query->autoEscape('yes')`

Form Elements

- Starting and Ending a form
  ```
  print $query->startform($method,$action,$encoding);
        ...various form stuff...
  print $query->endform;
  ```
- Text entry fields
  ```
  print $query->textfield(-name=>'field_name',
                          -default=>'starting value',
                          -size=>50,
                          -maxlength=>80);
  ```
- Big text entry fields
  ```
  print $query->textarea(-name=>'foo',
                         -default=>'starting value',
                         -rows=>10,
                         -columns=>50);
  ```
- Password fields
  ```
  print $query->password_field(-name=>'secret',
                               -value=>'starting value',
                               -size=>50,
                               -maxlength=>80);
  ```
- File upload fields
  ```
  print $query->filefield(-name=>'uploaded_file',
                          -default=>'starting value',
                          -size=>50,
                          -maxlength=>80);
  ```
- Popup menus
  ```
  print $query->popup_menu(-name=>'menu_name',
                          -values=>['eenie','meenie','minie'],
                          -default=>'meenie',
                          -labels=>{'eenie'=>'one','meenie'=>'two',
                                    'minie'=>'three'});
  ```

- Scrolling lists
```
print $query->scrolling_list(-name=>'list_name',
                             -values=>['eenie','meenie','minie','moe'],
                             -default=>['eenie','moe'],
                             -size=>5,
                             -multiple=>'true',
                             -labels=>%labels);
```
- Checkbox groups
```
print $query->checkbox_group(-name=>'group_name',
                             -values=>['eenie','meenie','minie','moe'],
                             -default=>['eenie','moe'],
                             -linebreak=>'true',
                             -labels=>%labels);
```
- Individual checkboxes
```
print $query->checkbox(-name=>'checkbox_name',
                             -checked=>'checked',
                             -value=>'TURNED ON',
                             -label=>'Turn me on');
```
- Radio button groups
```
print $query->radio_group(-name=>'group_name',
                             -values=>['eenie','meenie','minie'],
                             -default=>'meenie',
                             -linebreak=>'true',
                             -labels=>%labels);
```
- Submission buttons
```
print $query->submit(-name=>'button_name',
                          -value=>'value');
```
- Reset buttons
```
print $query->reset
```
- Reset to defaults button
```
print $query->defaults('button_label')
```
- Hidden fields
```
print $query->hidden(-name=>'hidden_name',
                     -default=>['value1','value2'...]);
```
- Clickable Images
```
print $query->image_button(-name=>'button_name',
                                -src=>'/images/NYNY.gif',
                                -align=>'MIDDLE');
```

When the image is clicked, the results are passed to your script in two parameters named "button_name.x" and "button_name.y", where "button_name" is the name of the image button.

```
$x = $query->param('button_name.x');
$y = $query->param('button_name.y');
```

- JavaScript Buttons
```
print $query->button(-name=>'button1',
                     -value=>'Click Me',
                     -onClick=>'doButton(this)');
```

- Autoescaping HTML
```
$query->autoEscape(undef);
```

Appendix D MOMspider

The Multi-Owner Maintenance spider. For URL information please see Appendix F at page 83.

The layout of a MOMspider instruction file as used for this project:

```
# This is a sample instruction file

SystemAvoid /home/ylausber/robot/MOMspider-1.00/avoid.mom
SystemSites /home/ylausber/robot/MOMspider-1.00/sites.mom
AvoidFile /home/ylausber/robot/MOMspider-1.00/.momspider-avoid
SitesFile /home/ylausber/robot/MOMspider-1.00/.momspider-sites
ReplyTo ylausber@cs.leidenuniv.nl

<Tree
      Name Animated GIFs Animation Images
      TopURL http://www.shore.net/%7Estraub/animated_gifs.htm
      IndexURL http://ind156b.wi.leidenuniv.nl:8086/spider/spider1.html
      IndexFile /home/ylausber/www/httpd/htdocs/spider/spider1.html
      EmailAddress ylausber
      EmailBroken
      EmailRedirected
      Exclude http://www.wi.leidenuniv.nl/
>
```

MOMspider's usage:

```
usage:  momspider [-h] [-e errorfile] [-o outfile] [-i instructfile]
                       [-d maxdepth] [-a avoidfile] [-s sitesfile]
                       [-A system_avoidfile] [-S system_sitesfile]
WWW Spider for multi-owner maintenance of distributed hypertext
infostructures.
Options:                                                     [DEFAULT]
```

- `-h Help -- just display this message and quit.`
- `-e Append error history to the following file.             [STDERR]`
- `-o Append output history to the following file.            [STDOUT]`
- `-i Get your instructions from the following file.`
  `[/home/ylausber/robot/MOMspider-1.00/instruct/.instruct0_0]`
- `-d Maximum traversal depth.                                [20]`
- `-a Read/write the user's URLs to avoid into the following file.`
  `[/home/ylausber/.momspider-avoid]`
- `-s Read/write the user's sites visited into the following file.`
  `[/home/ylausber/.momspider-sites]`
- `-A Read the systemwide URLs to avoid from the following file.`
  `[/home/ylausber/robot/MOMspider-1.00/system-avoid]`
- `-S Read the systemwide sites visited from the following file.`
  `[/home/ylausber/robot/MOMspider-1.00/system-sites]`

Appendix E Libwww-Perl

The libwww-perl distribution is a collection of Perl modules which provides a simple and consistent programming interface (API) to the World-Wide Web. The main focus of the library is to provide classes and functions that allow you to write WWW clients, thus libwww-perl said to be a WWW client library. The library also contain modules that are of more general use.

The latest version of the library is libwww-perl-5.00.tar.gz (163 KB) which was released May 26, 1996. You will need perl5.002 or better to use this version. (For perl5.001m you should continue to use the old libwww-perl-5b6.tar.gz (98 KB) which was released Nov 6, 1995.) For URL information please see Appendix F at page 83.

Libwww-perl for perl4 is used for the MOMspider (see Appendix D at page 79). Specific Libwww requests are: HEAD, GET, and POST.

I used libwww-perl-5b6 for the image retriever functions. The only function from libwww-perl-5b6 I used was the GET command:

Usage: GET [-options] <url>...

- -m <method> use method for the request (default is 'GET')
- -f make request even if GET believes method is illegal
- -b <base> Use the specified URL as base
- -t <timeout> Set timeout value
- -i <time> Set the If-Modified-Since header on the request
- -c <conttype> use this content-type for POST, PUT, CHECKIN
- -p <proxyurl> use this as a proxy
- -P don't load proxy settings from environment
- -u Display method and URL before any response

- -U Display request headers (implies -u)
- -s Display response status code
- -S Display response status chain
- -e Display response headers
- -d Do not display content
- -o <format> Process HTML content in various ways
- -v Show program version
- -h Print this message
- -x Extra debugging output

Appendix F
Important Links

HTML Links

- HyperText Markup Language (HTML)
  `http://union.ncsa.uiuc.edu/HyperNews/get/www/html.html`
- HTML Reference Manual
  `http://www.sandia.gov/sci_compute/html_ref.html`
- HTML Forms
  `http://www.hpl.hp.co.uk/people/dsr/html/forms.html`
- HyperText Markup Language (HTML): Working and Background Materials
  `http://www.w3.org/pub/WWW/MarkUp/`

CGI Links

- The Common Gateway Interface
  `http://hoohoo.ncsa.uiuc.edu/cgi/`

Perl Links

- UF/NA Perl Archive
  `http://www1.cis.ufl.edu/perl/`
- The Perl Programming Language
  `http://pubweb.nexor.co.uk/public/perl/perl.html`

CGI/Perl Libraries & Archives Links

- Perl WWW Documentation
  `http://www.perl.com/perl/wwwman/`

- cgic: an ANSI C library for CGI Programming
  `http://www.boutell.com/cgic/`
- Matt's Script Archive
  `http://www.worldwidemart.com/scripts/`
- CGI.pm - a Perl5 CGI Library
  `http://www-genome.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html`
- libwww-perl-5
  `http://www.sn.no/libwww-perl/`
- libwww-perl: Distribution Information
  `http://www.ics.uci.edu/pub/websoft/libwww-perl/`
- Web Developer's Virtual Library: Perl
  `http://www.charm.net/ web/Vlib/Providers/Perl.html`

Robots, Spiders, and Webcrawlers

- World Wide Web Robots, Wanderers, and Spiders
  `http://info.webcrawler.com/mak/projects/robots/robots.html`
- Guidelines for Robot Writers
  `http://info.webcrawler.com/mak/projects/robots/guidelines.html`
- Perl code to implement Robot Exclusion Standard
  `http://fuzine.mt.cs.cmu.edu/mlm/rnw.html`
- MOMspider – Distribution Information
  `http://www.ics.uci.edu/WebSoft/MOMspider/`
- Robot Mailing List:
  `http://info.webcrawler.com/mailing-lists/robots/info.html`

HTTP Daemon Links

- the NCSA HTTPd Home Page
  `http://hoohoo.ncsa.uiuc.edu/`

Project Links

- My bookmarks
  `http://ind156b.wi.leidenuniv.nl:8086/bookmarks.html`
- The Leiden 19th Century Portrait Database
  `http://ind156b.wi.leidenuniv.nl:8086/intro.html`

- Server statistics
  `http://ind156b.wi.leidenuniv.nl:8086/results.html`
- Demo statistics
  `http://ind156b.wi.leidenuniv.nl:8086/demostat.html`
- YurImage; a Content-based Image Search Engine
  `http://ind156b.wi.leidenuniv.nl:8086/cgi-bin/yurimage.cgi`

Appendix G
Setup of a HTTP Server

In this appendix we will explain the common procedure of how to setup your own HTTP server (The NCSA HTTPd, see Appendix F at page 83). NCSA HTTPd is an HTTP/1.0 compatible server for making hypertext and other documents available to Web browsers. The current version is 1.5.2.

Installation Instructions

Installing NCSA HTTPd can be broken down into these basic steps:

- Downloading the NCSA HTTPd Server
- Finding HTTPd a Good Home
- Configuring HTTPd
- Selecting Scripts
- Starting NCSA HTTPd

Downloading the NCSA HTTPd Server

If you already have a copy of the current release of NCSA HTTPd 1.5, you may skip to the next step. NCSA HTTPd is available as precompiled distributions and binaries for systems we have access to. There is also a new OneStep Download and Configure for easy access to NCSA HTTPd. If you already have NCSA HTTPd 1.4 running, you may update the HTTPd binary without the new support files.

Note: There are several new support files for the new authorization types, as well as examples of many of the other new features in the configuration files, so you might want to retrieve the support files anyways. A step-by-step guide on how to update the old server is available.

If your system is not on this list, or if you feel more comfortable doing so, you must compile a binary:

- AIX 4.1.4 - IBM RS/6000 Model 550
- BSD/OS 2.1 - Pentium 120
- IRIX 4.0.5 - SGI Indigo
- IRIX 5.3 - SGI Indy
- HP-UX 9.05 - HP 9000 model 715
- Linux 2.0.0 - Pentium 120
- OSF/1 3.0 - Dec Alpha
- SCO OpenServer 5.0 - Pentium 90
- SunOS 4.1.3 / Solaris 1.x - SPARCserver 690MP
- SunOS 5.4 / Solaris 2.4 x86 - Pentium 90 - Machine Unavailable
- SunOS 5.4 / Solaris 2.4 SPARC - SPARCstation 20
- SunOS 5.3 / Solaris 2.3 SPARC - SPARCstation 20
- Ultrix 4.3 - Dec Mips 3100

Finding HTTPd a Good Home

At this stage, you should move HTTPd's control files from the directory you have been working in to the directory you intend to use as ServerRoot in the server configuration file.

Unless you have redefined their locations in the server configuration file, you will have to move the following files and directories into ServerRoot:

- `httpd` : The server itself
- `conf` : Configuration files
- `logs` : Access log and error log
- `support` : Support programs
- `cgi-bin` : Server scripts

The logs directory should not be writable by the User your server is running as.

Configuring HTTPd

The NCSA HTTPd Server is a versatile piece of software, the result of hours and hours of blood, sweat, and tears by the NCSA HTTPd Development team and Beta testers the world over. Its versatility does come at a price, however, as you the webmaster must configure the server to fit your needs. There are 3 types of configuration that can be done.

Compile Time

The NCSA HTTPd server has several compile time flags. These are contained in the files `src/Makefile` and `src/config.h` of the distribution. These files are fairly well documented.

Startup Configuration Files

There are 3 files the server parses at start up (or after receiving a SIGHUP signal). There is some General information available on the configuration file format.

There are three configuration files which control Server Configuration, Resource Configuration, and Access Control. You should look at and modify the files `conf/httpd.conf-dist`, `conf/srm.conf-dist` and `conf/access.conf-dist` which come with the distribution so that they are correct for your server.

Run Time Configuration

The server also supports per directory configuration files, mostly for access control. The name of this file is set with the AccessFileName directive in the Resource Configuration file. See the Access Configuration documents for more information.

Selecting Scripts

NCSA HTTPd 1.5 comes with many CGI scripts, both useful and informative. NCSA HTTPd 1.5 does not support the former NCSA htbin scripts, and these scripts should be removed from the document tree. It is not necessary to install all of the scripts at your site (and we don't recommend it). Peruse the list of CGI scripts, and move those you want into the `cgi-bin/` directory you defined in your Server Resource Configuration File.

Starting NCSA HTTPd

Once you have completed the above steps, and you are using standalone you can start the server by typing httpd at the command line. *Note: You must be root* *in order to use a port less than 1024.* You may need to use one of the command line options to override the compile time paths to the configuration files. The flags are:

```
Usage:  httpd [-d directory] [-f file] [-v]
-d directory :  specify an alternate initial ServerRoot
-f file :  specify an alternate ServerConfigFile
-v :  version information (this screen)
```

If you are installing the server as root, you will probably want to automatically start HTTPd when the machine boots. This can be done through modifications to various `/etc/rc*` or `/etc/init.d/*` files, depending on your system.

If you are running the server from inetd, you need to edit the OS system files.