

EXPERIMENTAL TECHNIQUES IN THE RECORDING AND DISPLAY
OF ARCHAEOLOGICAL MATERIALS

A Thesis

by

SAMUEL ALLEN KOEPNICK

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of
MASTER OF ARTS

May 2011

Major Subject: Anthropology

Experimental Techniques in the Recording and Display of Archaeological Materials

Copyright 2011 Samuel Allen Koepnick

EXPERIMENTAL TECHNIQUES IN THE RECORDING AND DISPLAY
OF ARCHAEOLOGICAL MATERIALS

A Thesis

by

SAMUEL ALLEN KOEPNICK

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF ARTS

Approved by:

Chair of Committee,	C. Wayne Smith
Committee Members,	Donny L. Hamilton
	Donald H. House
Head of Department,	Donny L. Hamilton

May 2011

Major Subject: Anthropology

ABSTRACT

Experimental Techniques in the Recording and Display of Archaeological Materials.

(May 2011)

Samuel Allen Koepnick, B.A., University of Nevada, Reno

Chair of Advisory Committee: Dr. C. Wayne Smith

In the area of the display of data and images from archaeological sites there is very little uniformity. Universities, museums, and institutions use a variety of techniques and software. Because of the lack of a common framework for storing information gathered from the field a great deal of time is lost converting between disparate file formats and learning new program structures. The goal of this project is to create an open platform to accomplish the specialized tasks of recording and displaying data from the field, specifically dealing with the unique problems associated with sites in an underwater context. The final result should be freely available and adaptable.

Many challenges were overcome over the course of this project. Providing security, estimating the user's level of technical ability, creating a simple but effective interface, creating a three dimensional object viewer, and using only tools freely available for public use were the primary problems. The software chosen to author the platform as well as the hardware requirements were intentionally left to a minimum to ensure that users without access to the latest hardware would still be able to use these

tools. In addition to these requirements, the final product would have to be hardware agnostic, as well as operating system neutral.

As tempting as it would be to call this project complete, it is very much still an evolving work in progress. As new challenges arise the platform should be robust enough to be able to adapt. The modular design of the platform will ensure that future users will be able to adjust and even create completely new components to add functionality and customize the software to their needs.

DEDICATION

To my friends and family...

And to every computer geek who has ever had an interest in anthropology and to every anthropologist who has ever had an interest in computer science. It turns out that it is indeed possible to pursue both.

ACKNOWLEDGEMENTS

In a cross-disciplinary project such as this, a diverse group of advisors and friends becomes an indispensable resource.

First I would like to thank each member of my advisory committee; Dr. Smith for his technical knowledge, unique solutions to odd problems, and his consistently optimistic attitude, Dr. Hamilton for his willingness to let me try unorthodox ways of accomplishing tasks, and to Dr. House for letting me take classes far outside of my major field and for presenting the material in a very understandable way.

In addition thanks go out to the Immersive Visualization Center (IVC) and Ben Sutherland for letting me run some experiments on their hardware, the Visualization Laboratory for allowing an anthropologist to take their courses and use their equipment, and the Wilder 3-Dimensional Imaging Laboratory for the use of various imaging devices and graphics software.

Finally, thank you to my family, the Gatos and Segue, Danny and Patti Baird, Donovan and Mary Anne Rolaff, Michael and Lisa Crates, Mary Crates, and my parents, Frank and Susan Koepnick. All of whom had an unwavering faith.

NOMENCLATURE

OS	Operating System
RAM	Random Access Memory
CPU	Central Processing Unit
NURBS	NonUniform Rational B-Spline
API	Application Programming Interface
SQL	Structured Query Language
GUI	Graphical User Interface
PHP	Hypertext Preprocessor

TABLE OF CONTENTS

	Page
ABSTRACT	iii
DEDICATION	v
ACKNOWLEDGEMENTS	vi
NOMENCLATURE	vii
TABLE OF CONTENTS	viii
LIST OF FIGURES	x
LIST OF TABLES	xi
CHAPTER	
I INTRODUCTION - BACKGROUND AND PROBLEMS	1
II THE DATABASE	5
Choices	5
Artifact Structure	7
Scantling Structure	9
Discussion	10
III PROGRAMMING STRUCTURE	12
Basic Structure	12
Functions	17
User Interface	20
Security	27
Problems	29

CHAPTER		Page
IV	MODULES.....	30
	Artifacts and Scantling.....	30
	Audio.....	32
	Graphics.....	34
	Data.....	37
	Three Dimensional Models.....	38
	Video.....	44
	Map.....	45
V	CASE STUDY: KYRENIA.....	47
	Introduction.....	47
VI	SUMMARY.....	55
	The Good.....	55
	The Bad.....	56
	The Ugly.....	57
	The Future.....	60
	WORKS CITED.....	62
	APPENDIX A.....	63
	APPENDIX B.....	80
	VITA.....	84

LIST OF FIGURES

FIGURE	Page
3-1 Example of File Hierarchy.....	16
3-2 HSV Color Model.....	24
4-1 Audio Player	34
4-2 Web View of Gallery.....	35
4-3 Web View of Zoom Function	36
4-4 Web View of Opacity Function.....	37
4-5 Keyboard Controls.....	44
5-1 Text Module	49
5-2 Spreadsheet Module.....	50
5-3 Presentation Module	51
5-4 Static Three-Dimensional Model.....	53
5-5 Dynamic Three-Dimensional Model	54

LIST OF TABLES

TABLE	Page
1-1 Commercial Software Prices	2
2-1 Database Comparison	5
2-2 Artifact Table Structure	8
2-3 Scantling Table Structure	9
3-1 Primary Color Set	25
3-2 Complimentary Color Set	26
3-3 Comparison Between Primary and Complimentary	26
5-1 Trial Machines	48

CHAPTER I

INTRODUCTION - BACKGROUND AND PROBLEMS

We live in an era of unprecedented access to information at the touch of a button. Advances in technology have led to unique avenues of research combining formerly disparate aspects of very different fields. These advances have produced partnerships between previously unrelated fields of study. From an archaeological point-of-view this has been a major boon to research in general. Surveys can now be done using remote sensing equipment, excavations are aided by advanced location and imaging devices and the results of these operations can be exhibited to an audience using a seemingly infinite array of multimedia approaches.

With any new technological field there are those who would attempt to use it as a means of making money. Commercially available software has some attractive benefits. Most companies employ a staff of developers, customer service specialists, and salespeople devoted to their product. Their livelihoods are at stake and in a niche market such as academic archaeology a few bad experiences can permanently tarnish a reputation. Typically it is in their best interest to make sure their customers are happy. Of course this comes at a cost. Since the market is so narrow, these companies cannot rely on bulk orders and their prices often reflect this. While paying exorbitant prices for software may be considered normal by well-endowed departments and major institutes,

This thesis follows the style of *American Journal of Archaeology*.

it creates an extremely difficult scenario for students, smaller universities, and museums already burdened with budget concerns and cutbacks. Below is a listing of common software solutions and their retail prices as of April, 2009. There is little wonder why software piracy is so rampant.

One potential solution to this problem is the use of open source software. Many of the programs commonly used by researchers have an open source analog that is able to perform most, if not all of the same functions as its commercial counterpart.

TABLE 1-1 – COMMERCIAL SOFTWARE PRICES

Name	Purpose	Price	Source	Open Source
Adobe Photoshop	General image manipulation	\$699.00	Adobe.com (January 2009)	GIMP
Adobe Dreamweaver	Web design and syntax highlighter	\$399.00	Adobe.com (January 2009)	NVU ¹
Autodesk Maya	Polygonal modeling and rendering	\$1,995.00	Autodesk.com (January 2009)	Blender 3D ²
Rhinoceros	NURBS modeling and rendering	\$995.00	Rhino3d.com (January 2009)	Blender 3D ³
Site Recorder	Site mapping and artifact database	\$3,000.00	3hconsulting.com (January 2009)	MySQL ⁴

When this project was begun, the final goal was nebulous. The problems were well-known but a common solution everyone could agree upon was not. As the general framework was mapped out it became evident that creating one program to take over all of the tasks performed by the listed software was out of the question. A concept of

¹ NVU Download Page, 30 April 2009, <http://www.net2.com/nvu/download.html>

² Blender Download Page, 30 April 2009, <http://www.blender.org/download/get-blender/>

³ Ibid.

⁴ MySQL Download Page, 30 April 2009, <http://dev.mysql.com/downloads/>

integrating the source code from each of the open source projects was put forth but quickly rejected as being too ambitious for this project.

As it turned out the simplest solution turned out to be the correct one. The answer was not to consolidate a vast amount of functionality into one program; rather it was to create a framework capable of displaying data from a variety of nonproprietary file formats. By keeping it open-ended a user is able to quickly choose what sorts of information they want displayed. If one project calls for public access of stereo-lithography files or if another project decides to limit access to high-resolution images then it is a simple matter of changing a configuration file.

Many projects have approached the issue of storing and displaying data gathered from archaeological sites. Foremost was the Nautical Archaeology Digital Library. This was a joint venture between researchers at the departments of Anthropology, and Computer Science at Texas A&M University. The software that was created as a result of this partnership was both elegant and efficient. It was however geared towards the display of treatises and chronological data rather than more general data types.⁵

The goal for this project was an open source framework that allows researchers to collaborate, share data, and present findings with a minimal amount of effort. Once installed, the day-to-day operations should be as simple as dragging and dropping files into predetermined directories. The software would then mimic the directory structure when displaying that data. Since a great many data types would be involved and not every archaeological project will necessarily need every type, it was decided to design

⁵Nautical Archaeology Digital Library, 30 April 2009, <http://nadl.tamu.edu>

the framework in this project in a modular way. The exception to this modular philosophy was the database which forms the backbone of the relational data, namely the artifact information and the scantling information.

The information to be displayed was broken up into four broad categories with two branching into subcategories. These categories were (1) basic images, (2) high resolution images, (3) data, and (4) multimedia. The multimedia category includes three-dimensional objects, audio, and video while the data category consists of presentation-style slides, text documents, and spreadsheets. It should be noted that because of the modular design of this project, additional modules or file parsers can be added as the need arises. By using various built-in functions, users can create regular expression rules to parse text files for whatever purpose is needed. Parsers created for this project include a basic Rich Text Format (RTF), Comma Separated Value (CSV) spreadsheet, and Tab Delimited (TXT) spreadsheets.

Another important aspect is that of security. Users have to be able to secure their information against unwanted intrusion. This security would have to be simple enough for the average user to be able to grasp, while robust enough to stand against a determined attack. Data can be a very dangerous thing. Working with high profile cases requires a secure environment. This project is meant to be as universally applicable as possible and this means giving the user the option to be as open or as locked down as they choose.

CHAPTER II

THE DATABASE

CHOICES

The cornerstone of this project is the relational database where all of the data resides. As with the application software, there was a wide array of databases from which to choose. The biggest deciding factors were: price, scalability, performance, operating system neutrality, and interaction with other applications. Several experiments were undertaken with the major database offerings at the time of writing. The experiments were conducted using the software listed below in Table 2-1.

TABLE 2-1 – DATABASE COMPARISON

Firebird 2.11⁶	
Pros:	Open source and free, under continuous development.
Cons:	Relatively new entry to the database market, high network latency, low benchmark.
Microsoft SQL 2008⁷	
Pros:	Enterprise level reliability, high availability of learning resources.
Cons:	Will only work on Windows servers, high cost.
MySQL Community Server⁸	
Pros:	Open source and free, great deal of community development, large library of APIs
Cons:	Graphical interface is inefficient, requires a large amount time to install properly
Oracle Database 10g⁹	
Pros:	Free, proven reliability, high benchmark
Cons:	Not used as much as other offerings, closed source, difficult to install.

⁶ FirebirdSQL Technical Page, 12 July 2009, <http://www.firebirdsql.org/>

⁷ Microsoft SQL Editions, 12 July 2009, <http://www.microsoft.com/sqlserver/2008/en/us/editions.aspx>

Hinz 2009, MySQL Developer's Page, 12 July 2009, <http://dev.mysql.com/downloads/>

⁹ Oracle Technology Page, 12 July 2009, <http://www.oracle.com/technology/software/index.html>

TABLE 2-1 – CONTINUED

Microsoft Access 2007¹⁰	
Pros:	Low cost, easy graphical interface.
Cons:	Will only work on Windows and Macintosh operating systems, communications to web servers is difficult at best.

As with most types of software the final decision was not only what was most cost efficient and effective, but also which was already known well by the developer. In this case MySQL was chosen because it has no cost, and because SQL grammar is common to many types of database languages and is the most likely to be known by a user needing to make changes to suit a particular project.

Databases are the hallmark of professionally designed and maintained applications. When done correctly they can streamline work and perform amazing feats. When done incorrectly they can cause slowdowns, malfunctions, security issues, and general frustration. Because of the potential for catastrophic data loss for users not equipped with database backup software, the decision was made to design the database to be as simple and robust as possible. This includes automatic backup systems and clearly defined and structured functions.

The database is of the *InnoDB* type using a collation and character set of UTF-8. The alternative to using *InnoDB* was using *MyISAM*. While *MyISAM* is touted to be better at offsetting data handling functions to the operating system, *InnoDB* is more

¹⁰ Microsoft Access Home, 12 July 2009, <http://office.microsoft.com/en-us/access/default.aspx>

efficient at running queries as well as cleaning tables and rebuilding index files after a crash.¹¹ The 8-bit Unicode Transformation Format is quickly becoming a standard for transmitting data in a variety of characters. By using UTF-8 this project is able to recognize and store textual data from a staggering variety of character sets. Hindu, Mandarin, Japanese, Georgian, Russian, and many others are all represented. In addition, UTF-8 maintains backward compatibility with UTF-7 and ASCII.¹²

ARTIFACT STRUCTURE

Although this project could easily be adapted to be the primary means of storing archaeological data for a survey or excavation, it was not designed with this purpose in mind. Rather it was designed to provide a means to quickly and effectively store and present pertinent information such as rapidly changing field notes, initial observations, and other types of data which may not fit into any normal mold. The artifact table has a deceptively simple structure with only four fields. The first is the “ID” field which contains the internal identification established and used by the database. The datatype is an unsigned *mediumint* which can increment over 16 million times.¹³ The number field is a 20 byte *varchar* used to hold the identification of the artifact in the manner dictated by the project. The “Tags” field is used to hold a series of tags assigned to each artifact. Typically a relational database will accomplish this task with a set of keys. A primary key on the first table will relate to a foreign key on a second table. The database server

¹¹ Comparisons between *InnoDB* and *MyISAM*, 6 July 2009, <http://www.innodb.com> and <http://dev.mysql.com> respectively.
Pike and Thompson 1993, 43 - 50
Hinz 2009, Numeric Datatypes in MySQL, 22 June 2009,
<http://dev.mysql.com/doc/refman/5.0/en/numeric-types.html>

will then cross-reference the fields and output a resulting table. The reasons for attempting a new method of doing this are threefold. First, to the knowledge of the author it had never been tried this way. Second, early benchmarks showed that with a small enough set of tags (less than 100 per artifact) the server showed a performance increase. Lastly, this method of uniform field tagging is much simpler to grasp for one trying to make adjustments to the database. There is a far less likely chance of making a catastrophic alteration resulting in broken relationships and an unusable application. The field is of the *mediumtext* variety which will hold 16,777,215 characters per artifact.¹⁴ The final field is named “Description” and as its name implies holds a description of each artifact. The field is a standard *text* which will hold 65,535 characters per artifact.¹⁵ The “Tags” fields acts as a database within a database. This allows for new data types to be entered without changing the underlying structure of the main table. The importance and use of the “Tags” field will be discussed at length in a later chapter. The artifact table structure is summarized below in Table 2-2.

TABLE 2-2 – ARTIFACT TABLE STRUCTURE

Name	Type	Extras
ID	Mediumint (0 – 16,777,215)	Key, AutoIncrement
Number	Varchar(20 characters)	
Tags	Mediumtext (16,777,215 characters)	
Description	Text (65,535 characters)	

¹⁴ Datatype Limitations in SQL, 1 August 2009, <http://www.ispirer.com/doc/sqlways38/Output/SQLWays-1-196.html>

¹⁵ Ibid.

SCANTLING STRUCTURE

Like the artifact table, the scantling table set up specifically for shipwreck databases utilizes a double identification system. The “ID” field is a 10 byte unsigned integer which contains the automatically incremented identifier created by the database. The “friendlyID” is a 10 byte *varchar* and is what is displayed to the end-user as the identification. The “side” is a 10 byte *varchar* field and contains one of *Port*, *Starboard*, or *Center* to designate the side of the ship from which the timber came. The “type” is a 10 byte *varchar* field and contains one of *Frame*, *Keel*, *Post*, or *Planking* to designate the type of timber. Both of these fields will accommodate any sort of data used to describe either location or purpose of the timber. The “type” field is linked to a subdirectory of the “scantling” folder. For each type of timber input to the table, a corresponding subdirectory must be created. Each of these subdirectories will contain images corresponding to various entries in the scantling table. The application in turn will process these images and display them to the user. The scantling table structure is summarized below in Table 2-3.

TABLE 2-3 – SCANTLING TABLE STRUCTURE

Name	Type	Extras
ID	Mediumint	Key, Autoincrement
friendlyID	Varchar(10 characters)	
side	Varchar(10 characters)	
Type	Varchar(10 characters)	
Description	Text	

DISCUSSION

Both tables were intentionally kept as simple as possible. The reasoning behind this was twofold. Although even the most casual of computer users are at the very least aware of the existence of databases, their application in scenarios such as the one outlined by this project is not common knowledge. Since a major part of this endeavor is to create a collaborative suite as modular and adjustable as possible, having a large part of it dependent on a database which a user may not feel comfortable changing was not an option. Theoretically one could eschew the two modules which rely on the database and still have a fully functioning collaboration package. The other reason for the simplicity of the tables is one of sharing. The primary dataset should be in an open format to ensure future compatibility, and should be viewable by interested parties. MySQL tables in and of themselves fulfill neither of those criteria.

Despite the goal that the core of this project would not be an extensive database, the fact is that it is still a necessary part of the suite. As an important part it is necessary to backup and periodically ensure the integrity of the data held within the database. The folder “backup”, child to the root node is the repository of the backup system. The system is a simple query to the database dumping all of the data into a CSV(Comma Separated Value) spreadsheet. The spreadsheet is renamed with a timestamp and stored in the repository. A similar function is able to read a CSV file and store the contents in the database. In this way the data can be backed up incrementally allowing researchers to rebuild the database from various points in time before the current state. Another benefit to these functions is the ability to edit the spreadsheet in a more user-friendly

application offline and then copy the updated data back to the central server. Using this option a team could utilize some of the more sophisticated scripting features and macros available in commercial spreadsheet applications while still storing everything in a centralized location.

The backup operations could be automated by using scripting functions found on most modern operating systems. These languages differ based upon the operating system and can range from simple Windows Batch Scripting, to more general languages such as PHP and Perl. Adding functionality for an automated backup system was not intended for this version of the project, however examples in a variety of languages are included in the documentation and is a feature that will almost certainly be added in the next iteration. In the meantime sample scripts written in BASH and designed to be run as a Cron Script will have to suffice.

CHAPTER III

PROGRAMMING STRUCTURE

BASIC STRUCTURE

The most fundamental decision regarding the underlying framework of this project was that of which programming language would be used. The choice has to reflect the fundamental aspects of the project, namely it has to be an open language, and it has to be hardware agnostic, and OS neutral. The main languages reviewed were, the variants of C (C, C++, C#), the variants of BASIC (BASIC, Visual Basic, VB.net), Python, and PHP.

The C language and its various incarnations are the standard choice for creating solid applications. Because the programs are compiled from source into an executable binary, most errors are caught before they can cause problems. One major problem however is that binaries compiled for Windows based systems will not be natively executable in other operating systems. This would mean either that Linux/UNIX and Apple users would be required to have either software allowing Windows API's to run in userspace, for example Wine¹⁶, virtualization software such as VMware¹⁷, or the software would have to be ported to fit every operating system on which it could conceivably run. Another problem is the ability of other users to change the software. This project was intentionally left as generic as possible to allow users to customize it to

¹⁶ WINE Specifications, 1 August 2009, <http://www.winehq.org/>

¹⁷ VMWare Editions Page, 1 August 2009, <http://www.vmware.com/>

fit whatever role was needed. While C and its variants are the standard among computer scientists, the majority of users will have only a cursory knowledge at best. Expecting a non-specialist to be able to edit the code, ensure the correct libraries are in place, and recompile everything was simply not realistic.

BASIC and its offshoots were considered primarily for their ease of programming. Once again the primary problem was a lack of OS neutrality. The later variants of BASIC, Visual Basic and VB.net are tied to the Windows operating system¹⁸. While it is possible to get software developed in BASIC to work in other operating systems, it is substantially more difficult than the C language and the results are typically less stable. As of this writing Microsoft has made student editions of its VB.net editor available free of charge, but one cannot always assume that this will be the case. Because of the proprietary nature of the language it also fails to be open enough for the purposes of this project.

Python is a relatively new programming language. The modern version, Python 2.0 was released in 2000¹⁹. It is a very high-level language in that it leaves memory management and many other tasks to the operating system allowing the programmer to concentrate on the core of a project.²⁰ Python is OS neutral and runs basically the same on Windows, Linux/UNIX, and Apple machines. It has a large number of libraries enabling it to interface with various databases, and natively parse XML documents. GUI

¹⁸ Microsoft Visual Basic Specifications, 21 September 2009, <http://msdn.microsoft.com/en-us/vbasic/default.aspx>

¹⁹ Python Home Page, 21 September 2009, <http://www.python.org/>

²⁰ Python Software Foundation 2008, Executive Summary

development is handled by third-party libraries, typically TCL/TK. The code is relatively simple for a beginner and requires nothing more than a text editor to edit. There are also a number of libraries that allow OpenGL acceleration in applications.²¹ These reasons would make Python an ideal choice for a project such as this. The major problem with choosing Python was that the final goal of this project was to create an application that could be shared and viewed by anyone regardless of whether or not they had the authoring software. While programming in Python is a relatively easy experience, installing the framework to run it can be difficult. This difficulty is further compounded if the user interface requires third-party libraries which may not be part of the standard Python distribution.

In the end the decision was made to use PHP. PHP is a remarkably flexible language combining the object oriented strength of the C languages with the flexibility of markup languages. It has been in use since 1995 and has a tremendous number of libraries in active development to further increase its functionality.²² In essence PHP generates dynamic HTML (Hypertext Markup Language) code. The decision to use PHP took out a lot of the programming difficulty from the project. The GUI would still have to be created, but the actual creation and rendering of the windows would be taken care of by the server and internet browser respectively. PHP interfaces quite easily with MySQL making the creation of the basic functions for querying unnecessary.²³ The programming language is generic and is quite easy for a new programmer to become

Petrone 2002, 4

²² The PHP Group 2009, Instruction Manual, 4 May 2009, <http://php.net/manual/en/index.php>

²³ Ibid.

familiar. While the project may have the look of a website, in fact it is a complete application which happens to use HTML coupled with a server to generate and display the appropriate data to the end user.

With the programming language decided upon, the question then became how would one add new media to the project? The beginnings to the answer were based solely on experience. In the majority of archaeological projects, the data collected remains in the hands of the principal investigator. It can then be assumed that the vast majority of the material would come from a single source. Thus the primary means of transporting the media would be a direct file transfer rather than an uploading interface.

The next step was determining how the media would be displayed and shared. Ideally as little coding as possible would be required. The simplest solution turned out to be the correct one. Rather than keeping a database of cross-referenced descriptions and filenames, or requiring the page to be hardcoded to account for changes, the decision was made to have the pages build themselves dynamically by searching through a set of directories and subdirectories and linking to files that fit the types outlined in the configuration file. In the case of the *gallery*, *data*, and *multimedia* modules a menu is populated representing the file structure taking the appropriate directory in the *assets* folder as a root. In this way new galleries could be created simply by making a new directory, and adding the appropriate media to it. Each module has a corresponding entry in the configuration file where it loads information on the files it is set to retrieve. The entries are single-dimensional arrays consisting of the file extension that should be added to the page. A visual representation of the configuration file can be found below.

The first entry is the viewing entry, the particular file extension that will be displayed natively in the browser. In the case of HTML(.htm), JPEG(.jpg), and text(.txt) files the browser will simply display, in the case of the more advanced text and spreadsheet files, file parsing functions were created. Since the data module is more complicated than the others, primarily because of the fact that each subdirectory displays a different type of data, another entry in the configuration file was made to hold the appropriate information. These entries were arrays with the first element representing the name of the subdirectory and the second representing the file extension that would be displayed.

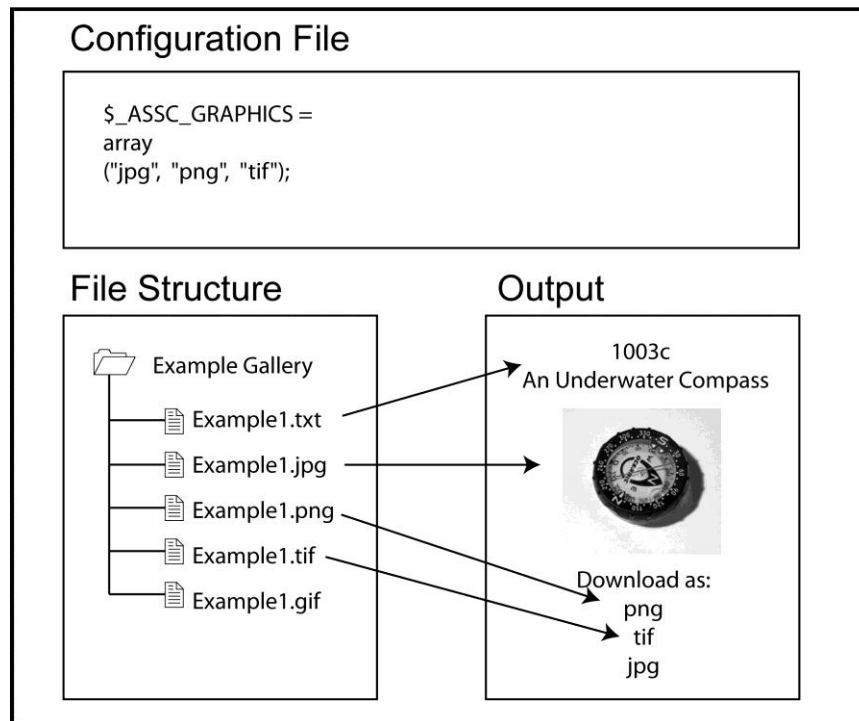


FIGURE 3-1 – EXAMPLE OF FILE HIERARCHY

FUNCTIONS

One of the most important features of this project is its modularity. Designing it in an object-oriented way makes it much easier for the developers and the users to edit the code to fit whichever needs are present. There are a number of functions utilized to keep the total filesize to a minimum and to optimize performance and flexibility. A complete listing of these functions can be found in the Appendix. The critical ones are listed here:

1) DatabaseConnect()

- Returns – A handle to the database connection.
- This function is called when the software needs to communicate to the database.

2) getDirectories(location:String)

- Parameters – Location relative to the “assets” folder.
- Returns – An array of subdirectories.
- This function is called when every subdirectory needs to be known. Typically this is used to populate tables of content.

2) getFiles (extensions:Array, location:String)

- Parameters – Array of file extensions stored as strings, location relative to the “assets” folder.
- Returns – An array of every file within the *location* parameter which has the same extension as that of the first element in the *extensions* parameter.

- This function is called when every instance of a particular filetype in a directory needs to be known. Typically this is used to form a list of the top-level objects to be displayed on a particular page.

3) getAllFiles (extensions:Array, location:String, filename:String)

- Parameters – Array of file extensions stored as strings, location relative to the “assets” folder, the filename.
- Returns – An array of every file within the *location* parameter which has the same name as the *filename* parameter and has an extension found in the *extensions* parameter. Typically this is used to form the hierarchal list from the top-level objects downward.

4) getCaption (location:String, filename:String)

- Parameters – Location relative to the “assets” folder, the filename.
- Returns – A string containing the first 64 bytes of data from a text file with the same name as the *filename* parameter and located in the *location* parameter.
- Typically this function is used for media which may display a caption to help explain the data. A modified version of this function is used to add descriptions to galleries.

5) getScantling (ID:Integer) / getArtifact(ID:Integer)

- Parameters – Unique ID for the object being referenced.
- Returns – An associative array containing pertinent information obtained from the database query.

- This function is used to reference various traits about artifacts and timbers for display on the page.

6) displayZoom (location:String, filename:String)

- Parameters – Location relative to the “assets” folder, the filename.
- This function uses both the phpThumb function as well as the MojoZoom function to display a scaled down version of a graphic as well as its full resolution counterpart in a series of *DIV* tags.

7) isPortrait (filename:string)

- Parameters – Location relative to the “assets” folder.
- Returns – A boolean indicating whether or not the orientation of the image being passed is portrait or landscape.
- This function is used by the displayZoom function in order to determine the location of the zoom window.

8) initializeDatabase ()

- This function contains and executes the SQL statements used to create and initialize the tables used by the project.

9) backupDatabase ()

- This function contains and execute the SQL statements used to backup the entire database into a single .sql file.

USER INTERFACE

When everything is said and done, a program can only be as good as its user interface. The GUI is the bottleneck between the user and the software. In most modern programs the software spends the vast majority of its time waiting for instructions from the user. The best user interfaces are those designed around the anticipated skill levels of its users and laid out in a logical manner. Another important aspect is consistency. If two controls look alike, they should have similar functions. The final characteristic should be accessibility. The program should be set up to correctly scale up or down based on a user's need, in addition HTML tags should be configured to work with screen readers.

The template designed for this project was a very simple two tiered interface popular in many modern applications. The banner is the center of attention and contains a link for the index page. Below the banner is the main navigation bar. The navigation bar is user-adjustable from the settings file. Ideally the number of items on this bar should not exceed six for the sake of visibility, although a limit was not explicitly set in the code. The font used was Arial primarily for its ubiquity among operating systems as well as its readability onscreen. A fallback generic sans-serif font was also incorporated for older operating systems and browsers.

This navigation system is known as a “Hierarchal” or “Sequential” system.²⁴ Galitz describes a series of general steps used to create the underlying structure of a

Galitz, 2008, 313

GUI. Step 1 is to know one's user or client.²⁵ Step 2 is to understand the business functions.²⁶ And Step 3 is to understand the principles of good interface and screen design.²⁷ The first two steps were explored earlier in this write-up. The final user or client was expected to be anthropologists, archaeologists, and any social scientist needing to share work securely with colleagues and associates. The business functions were examined in the preceding section. The final step however is extremely subjective and can vary greatly between people and cultures.

David Ruble postulates that four of the most important architectural trade-offs to be found in client/server systems are:

- Rapid Response Time at the Client
- Heterogeneity of Client Hardware
- Heterogeneity of Client Software
- Minimal Network Communication²⁸

To a great extent his first and fourth trade-offs are dependent on one another. Often rapid response time is directly linked to the amount of network traffic being pushed through the system. The textual and style protocol utilized by this project is HTML (HyperText Markup Language) and lends itself well for compactness. Unfortunately the multimedia modules of this framework are inherently difficult to trim to manageable levels. By their very nature video, audio, and three-dimensional data are

²⁵ Ibid. 71

²⁶ Ibid. 103

²⁷ Ibid. 127

²⁸ Ruble, 1997, 235 - 236

heavy and don't respond well to compression on-the-fly. It is important to note that Ruble was writing in the late 1990's when broadband speeds were relegated to the corporate world. The current state-of-the-art allows for the streaming of high quality video and audio without putting too much strain on a user's bandwidth.

The other two trade-offs that Ruble discusses involve heterogeneity of both hardware and software. The need for an application framework to span a multitude of hardware and software platforms was discussed earlier. The use of HTML as a means of encoding information ensures software neutrality. At the current stage of development the framework does not recognize differences between hardware configurations. The multimedia modules were coded in Flash which is not currently supported in any mobile hardware. As of this writing Adobe has not released a version of Flash compatible with standard mobile phone operating systems including offerings from Android and Apple. Unfortunately this means that even if the framework were to render differently to take advantage of specific screen resolutions available to mobile devices, the majority of the project would not be viewable. The user would need to develop a workaround for the end-user.

The idea of universal usability across heterogeneous systems leads to the idea of *Situational Awareness*. Basically situational awareness is the ability of an operator to predict future situations and events in a system.²⁹ This knowledge builds up over time and almost every computer user has it to some extent. By bundling disparate types of

Hoff, T, and Cato A Bjorkli, 2008, 137 978-82-519-2341-5

information into this framework, end-users should have a much easier time working with their data. Instead of having to develop situational awareness for a variety of systems, they only need to develop it for one which maintains similar controls throughout the application.

Each of the modules has its own particular navigation system optimized for the type of data it will be displaying. The optimization is based on the number of data elements, the means of displaying (video, audio, render, etc.), and the size of the viewing area. For most of the modules the navigation system consisted of a left oriented index of pertinent files or pages. Each of those links would in turn open up the item in the center content section of the page.

An important part of any design project is the color scheme. While this aspect may seem trivial in comparison to the other parts, in reality presenting a clean interface with colors that go together well is a cornerstone of any application. The first step is the easiest, which is to select a color. Using the system outlined below almost any color will work. The caveat to that is that due to the constraints of HTML rendering not all colors are “web safe”. In addition older monitors often may not have the same color gamut as that used by more modern display devices. That is the range of colors a particular device is capable of displaying may not necessarily be at the latest state-of-the-art. The vast majority of monitors manufactured today are capable of displaying a gamut of 16,777,216 colors (24 bit color and 8 bit alpha channel) at a resolution of 1280x1024 pixels, the decision was made to make the lowest color and resolution combination be 256 colors (8 bit) at a resolution of 800x600. This combination is commonly by web

developers and software engineers alike as a minimum requirement for systems. The initial color was arbitrarily decided to be a light shade of blue. For the purposes of choosing colors the HSV (Hue Saturation Value) color model was used rather than RGB (Red Green Blue) or CMYK (Cyan Magenta Yellow Black). The reasoning behind this is that the HSV system was designed around human senses while the other models were designed for display purposes. The HSV model can be visualized as a cylinder of colors (see Figure 3-2). As one moves around the perimeter the value changes. As one moves along the Y-axis the value changes, and as one moves toward the center the saturation changes.

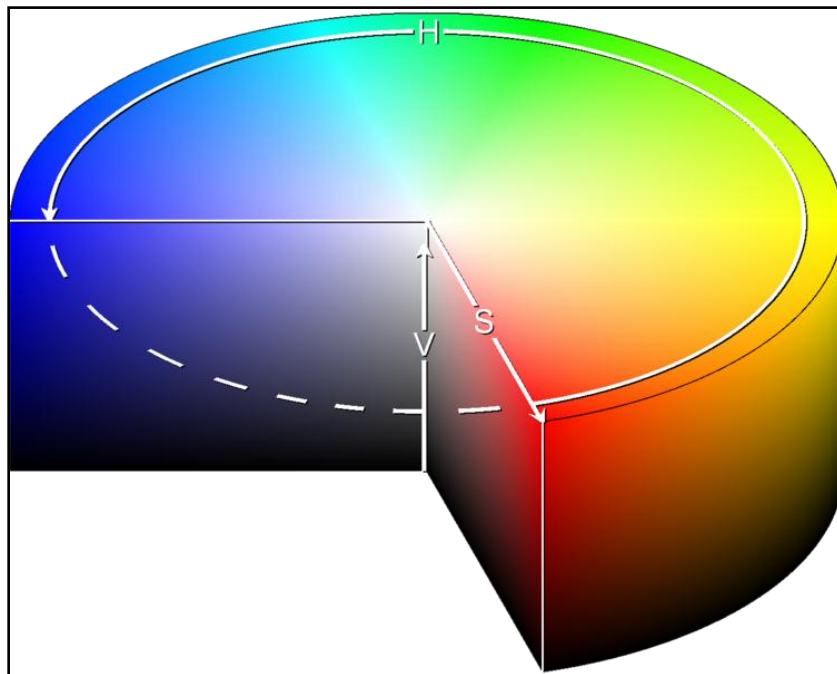









FIGURE 3-2 - HSV COLOR MODEL

As one can see from the model, limiting the Hue to a constant yields a rectangular cross-section of similar colors. From this point it's a simple matter of selecting colors by moving down the Saturation axis while subtly altering the Value. A total of seven colors were chosen in this manner. Their HSV and browser compatible hexadecimal RGB values are listed below in Tables 3-1, 3-2, and 3-3.








TABLE 3-1 – PRIMARY COLOR SET

H	220°	220°	220°	220°	220°	220°	220°
S	100%	50%	25%	67%	67%	50%	25%
V	73%	73%	73%	48%	24%	55%	63%
	#003EBA	#5D7CBA	#8B9BBA	#29447B	#14223D	#455D8B	#7986A1
							

The result is a monochromatic mix of colors which fit together quite well. There was however, a distinct lack of contrast. This was fixed by calculating the complimentary color for each of the samples shown above and slightly tuning the Value. This adjustment of the value was necessitated by the nature of this part of the color table. Despite their symmetrical nature the human brain will perceive some complimentary colors differently. After the initial calculations it was decided to increase the Value of the brown colors so as to seem as vibrant as the blues. The algorithm was quite simple and resulted in an increase in value between the disparate colors between 9% and 27% as

a function of their perceived vibrancy. The algorithm was applied and tested solely on the perception of the author.

TABLE 3-2 – COMPLIMENTARY COLOR SET

H	40°	40°	40°	40°	40°	40°	40°
S	100%	50%	25%	67%	67%	50%	25%
V	100%	100%	100%	66%	33%	75%	87%
	#FFAA00	#FFD580	#FFEABF	#A88338	#54411C	#BF9F60	#DECBA6
							

Putting the two color rows side by side shows their ability to contrast one another. In addition each color will adequately contrast text colored either pure white or pure black.

TABLE 3-3 – COMPARISON BETWEEN PRIMARY AND COMPLIMENTARY

#003EBA	#5D7CBA	#8B9BBA	#29447B	#14223D	#455D8B	#7986A1
#FFAA00	#FFD580	#FFEABF	#A88338	#54411C	#BF9F60	#DECBA6

The colors chosen for this project are not hardcoded into the system. The user is able to adjust or completely replace the initial color scheme as they see fit. In addition future iterations will include the ability to allow the end-user to choose themes. This

will give individuals with sight handicaps or trouble discerning colors the ability to choose a color scheme with a higher contrast.

SECURITY

When programming with the intention of mass dissemination, providing security is a fundamental concern. When sensitive information may be stored, the development of a secure environment becomes a task of the utmost importance. While there are countless ways of gaining access to critical parts of a website, only the most common were specifically guarded against while more general methods were used to secure against generic attacks.

This project makes use of both GET and POST procedures to transmit information between pages. The GET method is implemented by adding a variable and an assignment operator behind a url. An example of this would be “example.php?Name=Bruce&ID=342” In this instance two variables named ‘Name’ and ‘ID’ are passed to the PHP application named ‘example’. The POST method is essentially the same with the assignments made in the code of the referring application. The variables are handed off from application to application seamlessly without the user’s knowledge. While POST is generally considered to be more secure than GET, both are transmitted in plain text and even with encryption can be intercepted. The biggest threat with using these procedures is a vulnerability to Cross Site Scripting (XSS). Using XSS an attacker could theoretically inject malicious code into the website causing extreme damage. With a single line of code the entire database could be irretrievably erased, or users could be tricked into submitting personal information. To

combat this, a very simple script was used to sanitize external inputs. The sanitizing functions use regular expressions to remove or neutralize executable scripts. There are several functions, each designed to change the input in different ways. The first removes all non-alphanumeric characters, the second removes all non-numeric characters, the third removes all functions that could be used to gain access to a local computer, and the fourth returns HTML code formatted to display as text. The appropriate function was called at each and every input point to ensure that hackers utilizing Cross Site Scripting would be unable to gain access to the website.

The second and more general way used to secure the site can be found in the configuration file. Using PHP's built-in capabilities, a series of functions were created to limit access based on a user's IP address. There are two levels of distinction in this security feature. The first limits general access. The administrator may limit based upon a top-down system. For example, limiting to 192.168.3.X would allow any computer whose IP address began with 192.168.3 to access the site. Limiting to 128.X.X.X would allow any computer with an IP address beginning with 128 to access the site. This same system applies to the administrative areas of the site.

Although secure HTTP connections are not enabled by default in this project, that option is available for advanced users. The reason for this is that different web servers will configure HTTPS differently, most universities and institutions make use of some sort of firewall which could block ports needed for secure HTTP connections, and obtaining and setting up a signed certificate can be a difficult prospect.

PROBLEMS

One of the biggest problems encountered in academia is how does one control the level of sharing? Although the concept of transparency is openly embraced by this project, the stark truth is that researchers can often be very territorial about their projects and though they may want to publish all of their findings sometimes it is not convenient to do so. Because of this fact, this project by default will not publish links to full resolution graphics, documents, or any other type of raw data or binary file. However, these default settings are easily overridden in the configuration file. For each of the modules a number of file types may be associated. For example, if data were displayed from a Collada file, one could associate ASCII Object files (.obj) and Materials files (.mat) with that particular module. These associated files appear as links alongside the displayed data. The point of this is to give the end user a friendlier file format than the type used to display the information via a web browser.

CHAPTER IV

MODULES

The core of this project is the display of data in a meaningful way to both researchers and the general public. Because each archaeological excavation and survey is different, expecting a uniform solution that will fit every problem is simply unrealistic. However, the ways in which data can be displayed and distributed is finite. The solution to the problem of anticipating a user's need was to create broad-use modules. An excavation may need to display photos of artifacts *in situ*, workers excavating, historical background, reconstructions or any combination thereof. Rather than create a function for every possible contingency, it was much simpler to create a series of generic modules customizable by the user.

ARTIFACTS AND SCANTLING

The most common type of information brought from archaeological sites is in the form of artifacts. When the site in question is of a ship, that category should be expanded to include hull remains. This module is special in that it requires functions and procedures from the other modules. In addition the artifact and scantling modules are the only ones which rely explicitly on the project database to display information.

The multimedia nature of artifact data makes this module the most dynamic as well as the most difficult to code. Giving the user too many options in the configuration file could lead to confusion and the mishandling of the configuration, however making

the page too static could give investigators the sense that they could not adjust the program to fit their needs. In the end a middle road was taken. A field would hold a value to be used to reference each filetype, in the *scantling* table that field would be *friendlyID*, in the artifact table it is *number*. If more than one instance occurred, for example a series of photographs, then the file names would be referenced using a bracketed array system, *example[0].jpg*, *example[1].jpg*. By using this method the user maintains a great deal of control over which media to display on a per artifact basis, while the configuration file maintains a measure of simplicity.

Initially the interface designed called for the categories to be a hierarchal structure. The user would be allowed to assign an artifact to one of any number of categories. However after a series of field seasons categorizing and cataloguing artifacts, a system based on tags was deemed more appropriate. Each artifact could have any number of user-definable tags assigned to it. A sherd could be classified as “whiteware”, “pipe-clay”, and “Spanish” and a search through any of those terms would bring up the artifact and all similarly tagged. This system conforms to the reality that systems of classification seldom match perfectly between investigators. The database field used to hold the artifact tags is the *Tags* field and is of the text variety. This means that a maximum of 16,777,215 characters can be stored in each instance. Obviously this is far more data than any one artifact could possibly contain; the nature of the database engine allows this without any loss of functionality or performance. The tags are stored with semicolons as delimiters. The first subentry is the field name, the second is the

data. For instance, a cannon could have the following tags:

material=iron;nationality=Spanish;year=1689;excavationyear=1997

The true power and innovation behind this system of tagging is that each artifact can belong to any number of subgroups. Different queries could be saved to bring up information for a variety of tasks. This approach is similar to the standard database model the key difference is the approach outlined in this section allows for multiple entries into a single field. For instance a compound artifact could have the following tags: *material=ivory;material=antler;material=sinew;*

In this case a search for any of those particular materials would return this artifact.

Another aspect to the tagging system is that it allows for altering fields used to store information about the artifact without changing the fundamental structure of the database itself. In a normal database, if one wanted to add another field, the database table itself would have to be altered, the datatype specified, and other optional functional arguments stated. With this system the only thing required is to add a semicolon, and two arguments.

AUDIO

The audio module was one of the last objects to be added to the project. At first it did not seem as if it would be necessary to include this type of media in a project designed for the display of data. However, there could potentially be scenarios where it is advisable or even necessary to present to the end user an audio clip. Instances where a guest lecture or classroom discussion should be added, or adding commentary to help

disabled browsers more fully understand what is being explained are just a few of the possible scenarios.

There were two main options available for conveying audio to a user by means of a web browser. The first was to create a Small Web Format (.swf) application by using either Adobe Flash or Adobe Flex to create an audio player. The file to be played would be passed as an argument to the executing file. This is a typical method of accomplishing this task. The second option was to simply embed the audio file into a dynamically generated webpage and trust the operating system of the end user to have some method of playing audio already installed.

The decision was made to create a Flash application to present the data. Although some older browsers on esoteric operating systems may have difficulty playing media files embedded in this way, this approach reaches the largest audience. In addition, since the media content is presented as a downloadable file the option is available to save the file to the end user's computer and play offline rather than streaming from the collaboration suite itself.

Since the project simply passes the name of the audio file as an argument into the SWF file, the user is able to create their own streaming media player to replace the one programmed for the site. For the sake of compatibility the player was designed to be extremely simple with only a few controls. These controls are, a combination "Play" and "Pause" button, and volume controls.

The audio application (see Figure 4-1) was coded in ActionScript2.0 and parses the call arguments natively using the *FlashVars* routine. The title as displayed is simply

the filename stripped of its extension. The description is another, optional argument passed by the calling function.



FIGURE 4-1 – AUDIO PLAYER

The audio player is capable of playing MPEG Layer 3 (.mp3) and Waveform Audio (.wav) files. Like the rest of the project, the source code for the audio player is licensed under the GPLv2 and is available to be adjusted to fit the needs of the user.

GRAPHICS

In the context of this project, the term “graphics” applies to a static image regardless of its source. For the display of graphics, two modules were created. The first creates a standard album, useful for displaying thumbnails which when clicked opens the full-size image (see Figure 4-2). One potential problem was the scaling of the image. Originally an open-source program called “phpThumb” was used to dynamically resize images based upon existing entries in the configuration file. However the

performance of the server was extremely poor when images greater than 5 megabytes were resized, or when there were more than ten concurrent connections. Another option would have been to allow the user's browser to resize the image; however significant image degradation and aliasing artifacts necessitated a different approach. In the end the decision was made to leave this particular responsibility up to the user. In that way they could tailor each image to whichever dimensions they saw fit. One problem with this approach is the potential for users to simply drop in full resolution photos without resizing them. The hope is that extensive documentation and a warning system will limit this possibility.

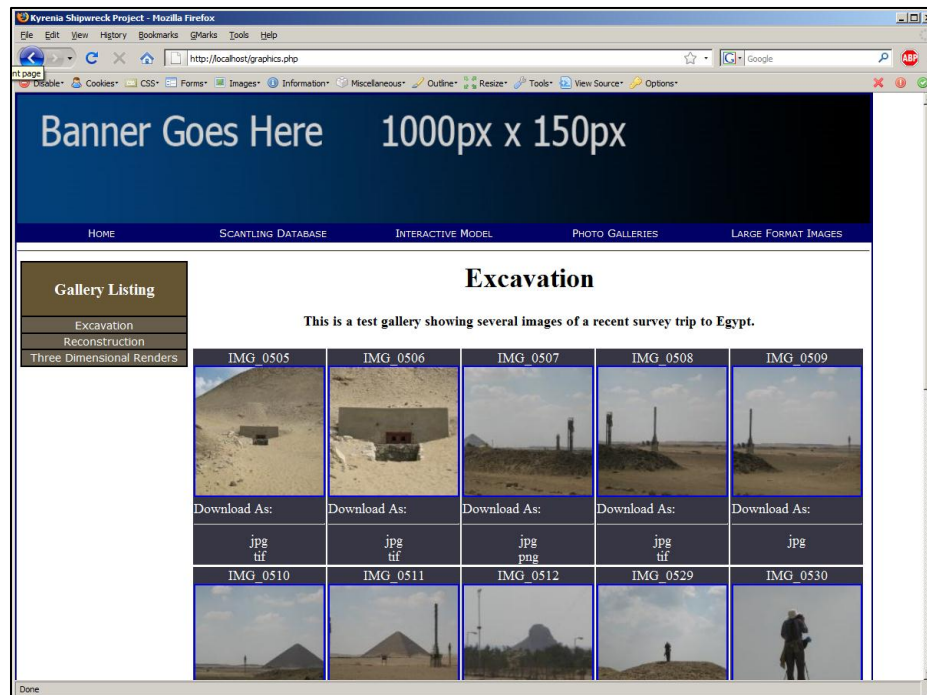


FIGURE 4-2 – WEB VIEW OF GALLERY

The other module created for the display of static graphics is built upon the idea a user may want to dynamically zoom into a specific area of an image. An open-source project named “MojoZoom” was used to fill this need. Using a combination of CSS and HTML DIV tags, a low resolution image is loaded onto the page with a high resolution version of the same image in the same directory. As the user hovers over the low resolution image a small box is shown (see Figure 4-3). That box corresponds to the corresponding area of the high resolution image which is displayed in another division. The result is an aesthetically pleasing effect which allows the user to easily navigate images in which minute details are important.

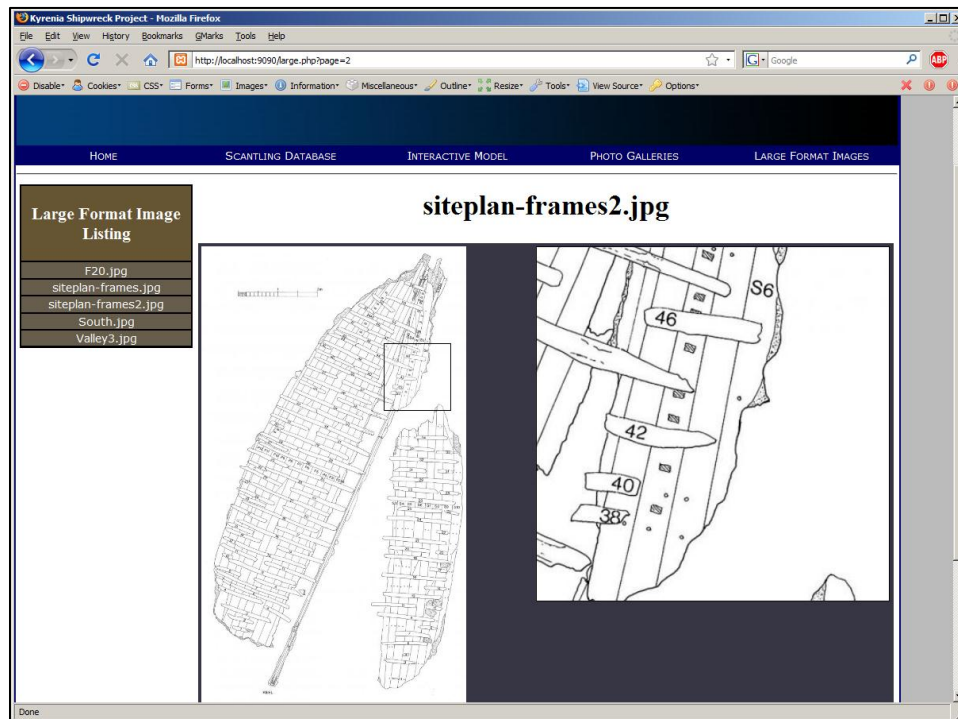


FIGURE 4-3 – WEB VIEW OF ZOOM FUNCTION

The final piece to the video module was another open-source project named “Videobox”. Through a Javascript and a stylesheet Videobox dynamically generates an HTML div with the same dimensions as the page set with a black background with an opacity of 80% (see Figure 4-4). Over this another div containing the image is generated. The overall effect is an animated dimming of the screen followed by the image loading. From a programming standpoint the effect was not necessary, however it lends the application a professional look which can be quite important.

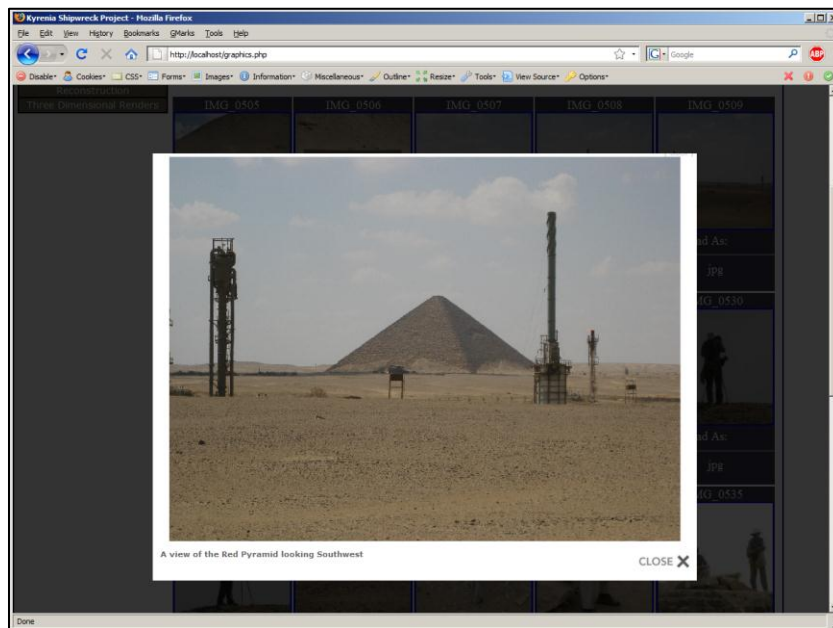


FIGURE 4-4 - WEB VIEW OF OPACITY FUNCTION

DATA

The data modules were designed for the display of text based information. The two primary types considered were standard text documents, and spreadsheets. In

keeping with the open philosophy of this project the file formats chosen to display the data were plain text (.txt), and comma separated values (.csv). Using a combination of arrays and regular expressions it was a simple matter to construct a parsing function to convert the data from the plain text formats into HTML code.

As was discussed in the Problems section of the previous chapter, the ability to share is an important part of this project, and doing so as neutrally as possible is important. To this end the decision was made to include a function to dynamically convert the displayed data files into Portable Document Format (.pdf). Over the last several years this format has proven to be popular when a text document must be displayed in a consistent manner between operating systems.

THREE DIMENSIONAL MODELS

At the time of writing there is no universally accepted way of rendering a three-dimensional object using HTML. In order to keep the project as dynamic as possible, the decision was made to leave the three-dimensional display up to the end-user. A placeholder was created so that a variety of programs could be created to fulfill the function. A basic imaging program was created in Adobe Flash for the purpose of this project, and as to act as a basic system for those users unable to spend the time and effort creating one. By keeping the system open, users now have the ability to customize either the included program or their own for whatever needs their particular project has. The built-in viewing program consists of a single object and a single camera operable by various keyboard inputs. Projects that require a more specialized viewer could easily

incorporate panoramic view, and a more dynamic user interface just to name a few examples.

The decision to use Adobe Flex as the authoring software and SWF as the application type was not made lightly. While Adobe Flex is free for students for non-commercial use, it is closed source and uses proprietary file formats to store information. However, Adobe Flex itself cannot display three-dimensional data. In order to do that a series of open source libraries had to be incorporated. Papervision3D is a series of libraries available for Adobe's ActionScript language. These libraries allow a developer to program applications combining three-dimensional rendering with the ubiquity of Adobe. Other authoring software was experimented with; however none could match Adobe Flex for ease of use, or the SWF format for standardization. To clarify, Flex is an SDK released by Adobe which combines the ease of use of an XML design philosophy with the flexibility of the object oriented language of ActionScript providing the backbone. The final file output is of the type Flash. Some of the other software tested included:

- Microsoft Silverlight – A new entry in the web based display of vector graphics. Microsoft has been trying to gain ground against its rival Adobe by making the development of Silverlight relatively inexpensive and by incorporating it into its familiar Visual Studio .net package. As of this writing the libraries in development for displaying three-dimensional images in Silverlight are in either alpha or pre-alpha phases and are not yet suitable for production.

- VRML / X3D – VRML was initially created in 1994 to create a standard way of displaying three-dimensional data on the internet. In 1997 the project upgraded its code and took on the name VRML97, after that the codeset was reengineered and is currently known as X3D. X3D is based upon XML files making it extremely flexible and open. The primary downside to this is that special viewers are required in order to render the data on a user's machine. Unlike the Adobe and Microsoft offerings (Flash and Silverlight respectively) which are commonly preinstalled on most major browsers, a user would have to know how to choose and install a reader.
- 3DMLW – This is also a relatively new addition to the world of three-dimensional modeling. Like X3D it uses an XML system making it extremely easy to edit. It also uses OpenGL to render its objects, an API found on the majority of modern operating systems. However a lack of a GUI for modeling could make learning a daunting task for new users. Another drawback is a lack of runtime model loading. Every model and texture that will be used must be loaded initially, this tends to make the model sizes rather large increasing the time between the initial selection and the display of the model.

The primary reasons Adobe Flex was chosen over 3DMLW were its small file size, and its primacy in the current market. While it is entirely possible that 3DMLW

will grow in acceptance over the next several years, its current status is that of a newcomer and its future is uncertain.

The Flash application designed for this project presented difficulties in designing an algorithm, although once that was done, the programming aspect proceeded rather smoothly. The working model files were to be loaded at runtime along with any texture files associated with them. The stage is a simple plane, itself an instance of the core primitives library of Papervision3D. The texture of the stage is loaded at runtime and is assigned based upon a variable passed to the application. Four possible options are available, and the user is free to replace any of them to create a new stage. A series of *for* loops create a tiling effect that essentially copies the stage sixteen times. For the best appearance the stage textures should be made seamless.

Ideally the model files should be of the ubiquitous object type (.obj). These files are encoded in plain text and are associated with a material file (.mtl) which contains information on how to texture the model. However, Papervision3D has limited support for object files and at the time of this writing is still in the very experimental phase. After some experimentation it was discovered that only very simple triangulated models were able to load successfully, the others caused the application to crash. In no instance was a material file successfully loaded. The next best option was to use the file type with the most documentation from Papervision3D. Collada is an open, XML based file used to store both geometries and texturing information for three-dimensional models. While it is not one of the better known file types, it is supported by many of the most widely used three-dimensional software suites.

The process of creating and altering the models for display within the project is as follows. It should be noted that the workflow and application choice were all the prerogative of the author. So long as the end file is a standards compliant Collada (.dae) file with an associated texture file the software should be able to display it properly.

The first step is to create the object file. This can be done in any number of commercial and open source software packages. For this project the NURBS models were created using Rhinoceros 3D v4.2, the polygonal models in Maya 2008, and the real-world objects scanned using a Minolta 710 imaging device and processed with RapidForm 2006. The object files were saved in the various native formats of their authoring software, and exported as Object (.obj) files for further processing.

The next step is to triangulate and texture the model. This was done using Maya 2008. The triangulation was done using the triangulate function and the texturing was done by mapping the UVs appropriately and exporting the map into an image editing program for use as a template. After the texturing image was created, it was applied to the model and rendered. The rendering was simply to act as a thumbnail image for later display by the software.

Although Maya 2008 does have an export function available for the Collada (.dae) format, the results were often unnecessarily larger and did not always display properly when compared to the same models generated by other three-dimensional packages. This necessitated saving the files into a neutral Maya (.fbx) format, loading them into MilkShape 3D and exporting them into the appropriate Collada (.dae) format.

The result was a small object file with an associated texture that could be compressed or altered to suit the needs of the user.

The final step was optional. The way in which Maya 2008 saves texture files is to put them into a subdirectory relative to the path of the model file. While this method may be appropriate for large-scale projects, this project required that the texture file be placed in the same directory as the model file so that both would be accessible by the end user. The solution was to simply move the texture file into the same directory as the model file. Next the model file was altered using a simple text editor to point the texture entry to the new location.

The movement controls needed to be as simple and intuitive as possible. If a too many controls were implemented, the software ran the risk of being overly complicated. Too few controls would leave out critical movements. The final decision was made to have six camera movement keys, two rotational keys, and six texture keys. The standard keyboard arrow keys were mapped so that the up and down arrows moved the camera forward and backward respectively while the left and right arrows rotated the object clockwise and counterclockwise respectively. The “W” and “S” keys were mapped to move the camera up or down relative to the X-Z plane while the “E” and “D” keys were mapped to change the angle of the camera relative to the X axis. In this way the end user’s right hand controls the basic commands while the left hand controls the altitude and pitch of the camera (see Figure 4-5).

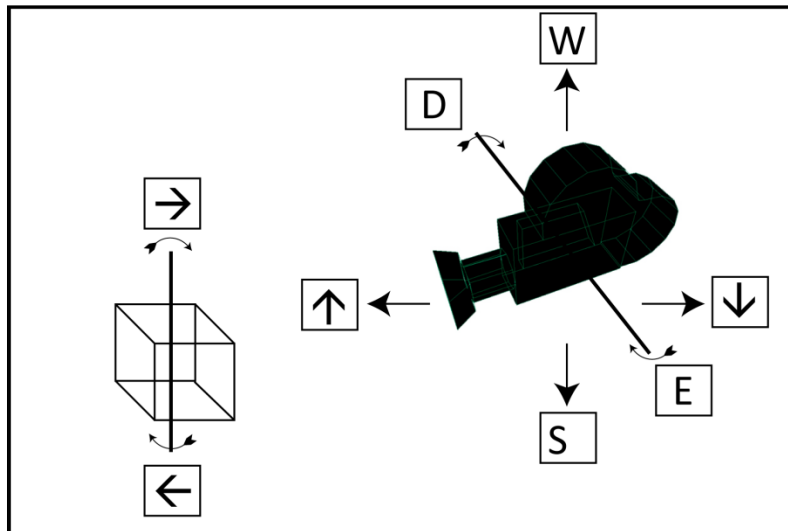


FIGURE 4-5 – KEYBOARD CONTROLS

VIDEO

Like the audio module, the video module was not part of the original design for the framework. However in order for this project to be considered truly multimedia, as many methods of conveying data must be used as possible. Although the traditional idea of video is that of a documentary or commentary, there are other reasons for implementing a video player. These reasons include displaying a slideshow, a presentation, or a scripted three-dimensional render.

The original video module was coded in Adobe Flash using ActionScript 3.0 as the primary language. Some notable problems, namely how to dynamically pass the location of a movie file at runtime using either POST or GET procedures proved to be

more difficult than originally anticipated. Several days of experimentation yielded a workable video player. However, a number of cryptic runtime errors and cross operating system inconsistencies showed that this was not an acceptable solution for a stable framework. Instead the Videobox application was used once again. The viewing window and control set was ideal for video function and included a scrub bar.

The problem of passing arguments to the player was bypassed by referencing the file directly. Originally the video player was called, and the name of the file it was to play was passed as an argument. In the new arrangement the file itself is called, and by using the “rel” method of the HTML Anchor tag it is told which application to use to display itself. This approach simplifies matters tremendously and allows direct linking to videos using short hyperlinks. From a security standpoint this approach is preferable because without arguments to be parsed, the chances of the site falling prey to cross site scripting are substantially lower.

MAP

The mapping functions were programmed near the end of the project and were not originally intended to be included. Not all archaeological projects will necessarily need to include a map and the map included in this project may not have the functionality that some projects require. There are three functions at the core of the mapping module. The first is a simple command to point to the location of the map file. The image file can be either a JPEG (.jpg) or a PNG (.png). Since the project already includes functionality for on-the-fly resizing of images, it would have been simple to include this in the map module. However requiring the user to manually resize their

map images to the proper 1,000 pixels wide is beneficial in that they will be responsible for maintaining the proper aspect ratio and cropping boundaries. The next function involves drawing the grid. The drawGrids function accepts a single variable specifying the distance in pixels between the gridlines. The function then creates a series of custom HTML divisions at the specified interval. The final function centers around adding artifacts to the proper location on the map. Each artifact in the database has an X and a Y coordinate which the user can use in whichever format they choose. The mapping function will only work if that format is entirely integer based. The map must use the same coordinate system as that used by the artifact database. For instance, if a map has a scale of 100:1 and uses meters as its system of measurement, then the artifacts must be similarly recorded.

CHAPTER V
CASE STUDY: KYRENIA

INTRODUCTION

In 1967 a Cypriot diver happened upon a mound of amphorae off of the northeastern coast of Cyprus near Kyrenia.³⁰ This find would mark a treasure-trove of data for the archaeological team from the University of Pennsylvania Museum. For two seasons they excavated and recorded the various timbers and artifacts from the site. What emerged was one of the most complete archaeological finds of the time. When Mr. Richard Steffy started teaching at Texas A&M University he brought all of his notes, images, and drawings with him. This treasure trove of data proved to be invaluable in proving the feasibility of projects such as this.

The data used for this case study encompassed nearly every module designed for the project. There were a wide variety of documents, spreadsheets, and presentations accumulated over decades of research and peer-reviewed publications. There were images taken of the site, timbers *in situ*, and various recreations. There was a plethora of data on the structure of the ship that could be used in the specially designed 'scantling' module. There was documentary video footage, which unfortunately could not be included in the final analysis because of copyright issues. There were even a number of three-dimensional models created by the author specifically to test the capabilities in that

Bass 1972, 52-60.

regard. What follows is an unbiased review of the strengths and weaknesses of the software as used in a real world application.

The trials were done in three different environments in order to benchmark speed, bandwidth, and to ensure a consistent user interface. See Table 5-1 for an overview of these environments.

TABLE 5-1 - TRIAL MACHINES

Operating System	Clock Speed	Cores	RAM	Network Type
Windows 7	3.6 GHz	4	8 GB	1.0 GB LAN
Windows XP	1.6 GHz	1	1 GB	801.11g WLAN
Ubuntu Linux	2.0 GHz	2	2 GB	100 MB LAN

The data modules were the most difficult to program for and still present the least polished aspect of this project. When this project was begun the *de facto* standard for productivity software was Microsoft's Office 2003. This version saved documents in an encoded binary file making it extremely difficult for outside editors to view and change content. The workaround at the time was to export the various documents into more neutral formats. This method worked, but in the conversion process important formatting and metadata would be irretrievably lost. Recently, Microsoft has embraced a more open approach to saving documents. In the latest iteration of its office suite Office 2007, the encoded binary has been replaced by an uncompressed archive file containing a series of XML files which in turn contain all of the user generated content,

formatting, and metadata. This new openness will allow the creation of modules which will be able to open these documents and display them on the end users monitor with the proper formatting intact.

This problem extends to the spreadsheet and presentation modules as well. In the instance of the spreadsheet modules each file must be exported as a CSV in order to be read properly, while the presentation must be exported as a series of JPGs. See Figures

The screenshot displays a web interface titled "Archaeological Data Demo". At the top, there is a navigation menu with links for HOME, ARTIFACTS, SCANTLING, GALLERIES, LARGE FORMAT MEDIA, DATA, MULTIMEDIA, and MAP. Below the menu is a sidebar with a "Data Objects" section containing links for Presentations, Reports, Site Report, Text, and Spreadsheets. The main content area is titled "Viewing in Category Reports" and includes "Download Options: .docx | .txt". The text describes the Kyrenia ship, a 4th-century BC Greek merchant ship discovered in 1967, and its preservation at the Ancient Shipwreck Museum in Kyrenia Castle. It also mentions the ship's sinking in the Mediterranean during the time of Alexander the Great and its discovery in 1974.

FIGURE 5-1 - TEXT MODULE

5-1, 5-2, and 5-3 for examples of the text, spreadsheet, and presentation modules respectively.

Archaeological Data Demo

HOME ARTIFACTS SCANTLING GALLERIES LARGE FORMAT MEDIA DATA MULTIMEDIA MAP

Data Objects

- Presentations
- Reports
- Spreadsheets
- test1
- KyreniaVTK

Viewing in Category Spreadsheets

Download Options: [.xls](#)

Frame	3D Completed
F03	Yes
F08	Yes
F14	Yes
F20	Yes
F25	No
F31	No
F35	Yes
F40	Yes
F47	Yes
F55	Yes

winscp425setup.exe

FIGURE 5-2 - SPREADSHEET MODULE

The screenshot displays the 'Archaeological Data Demo' website. The header features a dark blue background with a stylized tree logo on the left and the title 'Archaeological Data Demo' in white. Below the header is a navigation bar with links: HOME, ARTIFACTS, SCANTLING, GALLERIES, LARGE FORMAT MEDIA, DATA, MULTIMEDIA, and MAP. On the left side, there is a 'Data Objects' sidebar menu with options: Presentations, ABC, Kyrenia Slideshow, SlideShow, Test2, Reports, and Spreadsheets. The main content area is titled 'Viewing Kyrenia Slideshow in Category Presentations' and includes 'Download Options:'. Below this, there is a navigation sequence: 'Previous' followed by a series of numbered links (1-21) and a 'Next' link. The central focus is a photograph of a wooden boat with a white eye-shaped logo on its side, docked in a harbor with buildings in the background.

FIGURE 5-3 - PRESENTATION MODULE

The three-dimensional module was another very difficult several months in programming. A similar problem to that of the neutral file types in the previous modules reared its head here. An OBJ file is simply a series of coordinates outlining how a polygonal mesh exists as a series of vertices in Cartesian space. There are two ways in

which this surface can be colored. In the simplest manner each vertex is assigned a color value based on the RGB (Red, Green, Blue) color index. The other way is to assign a texture map to the object. This map has the same vertices as the actual model and simply describes how a bitmap is shaped around the complex geometric features. The former method allows for a very general coloring effect but is useful in that the amount of processing power and bandwidth needed to apply it is negligible. The latter method can produce surprising detail but has other drawbacks. First, it roughly doubles the size of the file which means more processing power, bandwidth, and time would be needed to display. Second, realistically it can only be applied toward the end of a model's production, as even a minute change in vertices will greatly alter the way the image is wrapped around the model. Finally, the end result is a series of files which may not be useful to an end user, rather than simply double clicking on a single file, the user is presented with a series of files which they may not have any familiarity with at all. Until the industry decides on an open standard which will allow for texture, bump, shadow, and occlusion mapping it will be up to the user to decide how best to present their three-dimensional work.

See Figures 5-4 and 5-5 for examples on the presentation of static and dynamic three-dimensional models respectively.



FIGURE 5-4 - STATIC THREE-DIMENSIONAL MODEL

As a whole the module worked surprisingly well on every system. The load times for models with an external texture map were rather long, but this could be alleviated by having the application hosted on an internal network. The camera controls seem to be intuitive and allow for a single-handed approach to navigating around the model.

One of the problems with this module is the application of a texture file over a complicated polygonal geometry. The time needed to complete this procedure was entirely dependent on the client computer. For moderately recent machine specifications this is not a problem. Older machines will have issues applying these textures. The

obvious solution is to implement a timer in the client-side software. This timer will run for a specified amount of time, and if the model has not loaded by its end a lower resolution fallback option will be implemented.

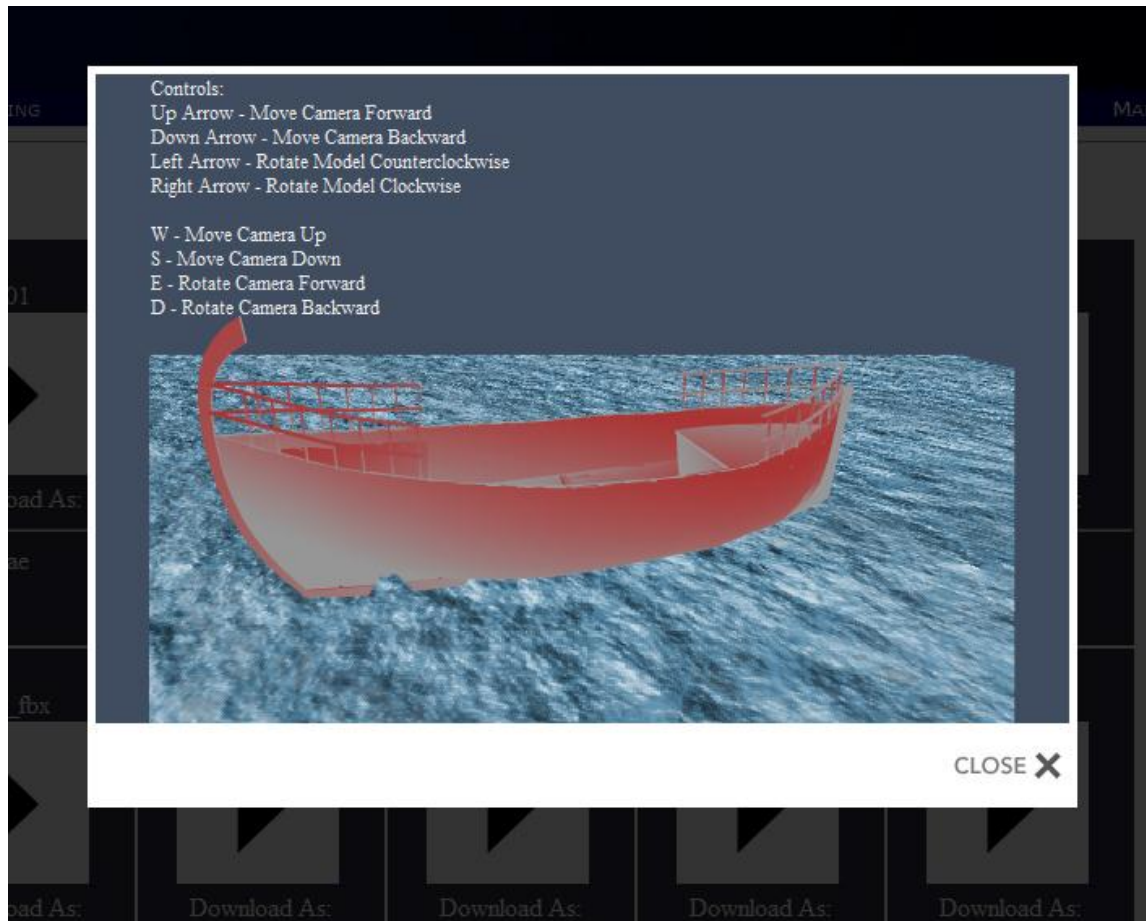


FIGURE 5-5 - DYNAMIC THREE-DIMENSIONAL MODEL

CHAPTER VI

SUMMARY

THE GOOD

The initial goal of this project was to create a framework for the display of archaeological data and the collaboration of different types of work. These goals, vague as they were, have been accomplished. All of the data types likely to be encountered or produced by an archaeologist in either the field or a laboratory environment have been examined and accounted for in the software.

Raw data in the form of plain text can be displayed through a number of built-in parsing functions depending on which format the user wants. Audio data is output as a stream, an included audio player programmed in ActionScript 2.0 in Adobe Flash acts as an intermediary between the end-user and the stream. A similar situation exists for the video player, with a barebones player streaming the video feed to the end-user. The three-dimensional object viewer was written in ActionScript 3.0 in Adobe Flex. The viewer incorporates the open source framework *Papervision 3D*. It loads objects and texture-maps specified by the end-user and displays them accordingly. The project was left open enough that any other file types can easily be added the configuration file with their corresponding parsing function. In addition any file can be output as a binary stream in and of itself.

At this stage of development the project works with a minimum of bugs. Each parsing function works precisely as needed and the data is displayed correctly. The

audio and video files stream correctly and the photo galleries are optimized based on a user's monitor resolution. The very large image format viewer works quite well, although there is room for improvement in a number of its aspects. The mapping functions currently work as a proof-of-concept but are not yet ready for field use yet.

THE BAD

The biggest concern while testing this project is one of performance. Several aspects of the codebase can be optimized to provide a better end-user experience. Improvements in functions and subroutines will help, but the fact of the matter is that a project relying on a programming language designed to output to another programming language which in turn is rendered using any number of 3rd party applications is going to have performance issues.

In hindsight this project would have been better served if the software itself had been contained in a single executable file. However the arguments made in the beginning of this work still pertain. Operating system neutrality should still be of primary concern, however the fact remains that Microsoft's Windows operating system commands ninety-two percent³¹ of the user base, thus software written for Windows would have the most potential for market penetration.

Future versions of this software will most likely be written in a combination of C++ and Java. A server will still be needed to coordinate, parse, and display the data.

NetMarketShare - Browser Share Trends, September 12 2009, <http://netmarketshare.com/browser-market-share.aspx?qprid=1>

Using the languages outlined above also give the ability to use hardware acceleration to aid in the display of three-dimensional images. Offloading the visual processing to an end-user's GPU will free the CPU for data processing and should provide a marked performance increase.

THE UGLY

As it turned out, some of the goals of this project were mutually incompatible. While designing a universally usable interface is impossible, creating one that is understandable by the majority of one's target audience certainly is. However the goal of creating a framework robust enough to tackle any potential problem or data type while at the same time programmed simply enough to be adjusted by a non-expert was not realistic.

The software itself does work as it was intended. The user interface is clear, concise, and devoid of bugs. However after some preliminary testing it has been shown to be remarkably easy to break if certain aspects of the code are edited incorrectly. The error checking functions that were built-in are designed to catch and log errors in database connectivity and malicious cross-site scripting. Clearly there needs to be a series of functions to review the software and display potential problems after changes are made to the code. These audit functions would need to be capable of running snippets of the code and catching errors on-the-fly.

The multimedia modules work as intended. If the project were to be redone, more time would have been spent on the three-dimensional module and less on the audio

and video. The three-dimensional module should also have been programmed in a more open language. Using ActionScript 3.0 in Adobe Flex was appropriate, however using this particular language sacrificed the open philosophy of the project. At the time it was thought that users could simply write their own players for three-dimensional objects. After much discussion with members of the intended user base it was discovered that one of the unwritten rules of computer science was making its presence known. Users will typically keep whichever applications are preinstalled rather than going through the trouble of searching for or creating their own.

At the time of this writing new advancements in coding and multimedia embedding in browsers are beginning to make some of the included players in the modules obsolete. Changes in HTML implementation, specifically HTML5 will allow audio and video to be embedded directly into HTML without the need for a 3rd party viewer.³² However there will still be a niche for 3rd party viewers, especially when an application requires a customizable interface. Experiments were made to test the feasibility of displaying video without a 3rd party viewer directly in the browser. Currently there is not a universally agreed upon codec for the display of video. The major players are extolling their various compression algorithms and different browsers support different types.³³

The video types are:

- Theora

HTML5 Technical Specifications, September 12 2009, <http://dev.w3.org/html5/spec/Overview.htm>
HTML5 Video Specifications, September 12 2009, <http://www.w3.org/TR/html5/video.html>

- MPEG4
- H.264

While the audio types are:

- AAC
- Vorbis
- MP3

It would have been in the realm of possibility to create a module that would recognize the end user's browser and display the appropriate file. The problem with this approach is that the server would need to have three versions of each file stored; tripling the necessary storage space for what is arguably the largest of the file types.

Eventually there will have to be a unified codec structure. When that happens all that will be required to bring that into this framework will be the uncommenting of a few lines of code. Until then 3rd party viewers will have to be used.

A similar situation exists for 3-dimensional models although a similar solution is almost definitely not forthcoming. Security concerns and technological barriers prevent programmers from tapping into the potential power of an end user's computer to present 3-dimensional shapes. While the libraries needed, namely OpenGL, are almost universally available, differing hardware specifications make any kind of standardization problematic. The most common approach, as well as the one used in this framework is to have a set of functions which will take a 3-dimensional object or scene and allow it to be displayed as a rapid succession of images controllable by the user. This gives the illusion of viewing a 3-dimensional object natively while still keeping the software

within the realm of feasibility for the majority of end users. Doubtless as HTML rendering technology matures there will be a more standardized approach to displaying 3-dimensional data. When this happens, much like in the video module an adjustment to a few lines of code will suffice to keep the framework updated. This further displays the power of a modular framework.

THE FUTURE

As frameworks such as this become more and more common there will be an expanding need for a unified set of API's. As discussed previously API's are created to allow various applications to communicate through a shared group of libraries. In this way applications can be tailor-made to fit almost any environment conceivable.

For projects needing advanced three-dimensional capabilities a client could be designed to take advantage of the full power of an end-user's computer. This software would act as a middle agent to transfer data from a central server and display it to the end-user. The application would connect to the server by using the aforementioned API's.

The power of an open source project is also its greatest weakness. The ability to change a project to better suit the needs of an organization, better known as "forking" ensures that a project will always be relevant as long as individuals are willing to put in the work to keep everything up-to-date. This does create a problem of maintaining an "official" distribution. As of this writing the official repository for this project is hosted on Google's Code site. Official updates can be downloaded as stable releases or as

nightly builds. In addition to this, Google acts as an SVN server allowing additional collaboration for anybody willing to devote the time and effort to adding functionality to this project.

If there is one thing that can be taken from the study of anthropology it is that there is a great deal of power in community. Open source projects tap into this potential and represent the future of software design and collaboration. The project as it stands right now is simply a work in progress. It will only be as useful as the community makes it.

WORKS CITED

Galitz, Wilbert. 2008. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. Hoboken:Wiley Publishing.

Hinz, Stefan. 2009, 19 July. "MySQL Reference Manual" *MySQL.com*, <http://dev.mysql.com/doc/refman/5.0/en/numeric-types.html>.

Ruble, David A. 1997. *Practical Analysis and Design for Client/Server and GUI Systems*. Upper Saddle River:Prentice Hall Publishing.

Hoff, Thomas and Cato A. Bjorkli. 2008. *Emodied Minds - Technical Environments: Conceptual Tools for Analysis, Design and Training*. Trondheim:Tapir Academic Press.

Pike, Rob and Ken Thompson. 1993. "Hello World." *Proceedings of the Winter 1993 USENIX Conference*. pp. 43-50.

Petrone, Jason. 2002. "3-D Programming with Python." *Linux Journal* 94:4830.

APPENDIX A

SECURITY ANALYSIS

The security audit for this project was conducted using Tenable Network Security's Nessus vulnerability scanner. The following plugins were used: Backdoors, CGI abuses, CGI abuses : XSS, Databases, and Web Servers.

Localhost

Scan time :

Start time :	Thu Feb 26 15:14:08 2009
End time :	Thu Feb 26 15:16:40 2009

Number of vulnerabilities :

Open ports :	16
Low :	7
Medium :	2
High :	0

Port mysql (3306/tcp)

MySQL Server detection

Synopsis :

A Database server is listening on the remote port.

Description :

The remote host is running MySQL, an open-source Database server. It is possible to extract the version number of the remote installation by receiving the server greeting.

Solution :

Restrict access to the database to allowed IPs only.

Risk factor :

None

Plugin output :

The remote MySQL version is 5.1.30-community

Nessus ID : 10719

Web mirroring

The following CGI have been discovered :

Syntax : cginame (arguments [default value])

/styles/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/index.php (page [-1])

/swf/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/swf/history/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

. (keep [1] page [0] sort [friendlyID] image [5])

/swf/ConeExample.swf (timber [complete])

/includes/phpThumb.php (src [../assets/scale//F20.jpg] w [400] zc [1])

Directory index found at /styles/

Directory index found at /swf/

Directory index found at /swf/history/

The following CGI have been discovered :

Syntax : cginame (arguments [default value])

/styles/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/index.php (page [-1])

/swf/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/swf/history/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

. (keep [1] page [0] sort [friendlyID] image [5])

/swf/ConeExample.swf (timber [complete])

/includes/phpThumb.php (src [../assets/scale//F20.jpg] w [400] zc [1])

Directory index found at /styles/

Directory index found at /swf/

Directory index found at /swf/history/

Nessus ID : 10662

HyperText Transfer Protocol Information

Synopsis :

Some information about the remote HTTP configuration can be extracted.

Description :

This test gives some information about the remote HTTP protocol - the version used, whether HTTP Keep-Alive and HTTP pipelining are enabled, etc...

This test is informational only and does not denote any security problem

Solution :

None.

Risk factor :

None

Plugin output :

Protocol version : HTTP/1.1

SSL : yes

Pipelining : yes

Keep-Alive : yes

Options allowed : (Not implemented)

Headers :

Date: Thu, 26 Feb 2009 21:15:03 GMT

Server: Apache/2.2.11 (Win32) DAV/2 mod_ssl/2.2.11 OpenSSL/0.9.8i

mod_autoindex_color PHP/5.2.8

X-Powered-By: PHP/5.2.8

Content-Length: 2197

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html

Nessus ID : 24260

HTTP TRACE / TRACK Methods

Synopsis :

Debugging functions are enabled on the remote web server.

Description :

The remote webserver supports the TRACE and/or TRACK methods. TRACE and TRACK are HTTP methods which are used to debug web server connections.

In addition, it has been shown that servers supporting the TRACE method are subject to cross-site scripting attacks, dubbed XST for "Cross-Site Tracing", when used in conjunction with various weaknesses in browsers. An attacker may use this flaw to trick your legitimate web users to give him their credentials.

See also :

http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf

<http://www.apacheweek.com/issues/03-01-24>

<http://www.kb.cert.org/vuls/id/867593>

Solution :

Disable these methods.

Risk factor :

Medium / CVSS Base Score : 5.0

(CVSS2#AV:N/AC:L/Au:N/C:P/I:N/A:N)

Solution :

Add the following lines for each virtual host in your configuration file :

```
RewriteEngine on
```

```
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
```

```
RewriteRule .* - [F]
```

Alternatively, note that Apache versions 1.3.34, 2.0.55, and 2.2 support disabling the TRACE method natively via the 'TraceEnable' directive.

Plugin output :

The server response from a TRACE request is :

TRACE /Nessus31286.html HTTP/1.1

Connection: Close

Host: localhost

Pragma: no-cache

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*

Accept-Language: en

Accept-Charset: iso-8859-1,*,utf-8

CVE : CVE-2004-2320

BID : 9506, 9561, 11604

Other references : OSVDB:877, OSVDB:3726

Nessus ID : 11213

Web mirroring

The following CGI have been discovered :

Syntax : cginame (arguments [default value])

/styles/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/images/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/assets/html/Stuff/images/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/backups/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/index.php (page [-1])

/swf/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/swf/history/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

. (keep [1] page [0] sort [friendlyID] image [5])

/swf/ConeExample.swf (timber [complete])

/includes/phpThumb.php (src [../assets/scale//F20.jpg] w [400] zc [1])

Directory index found at /styles/

Directory index found at /images/

Directory index found at /backups/

Directory index found at /assets/html/Stuff/images/

Directory index found at /swf/

Directory index found at /swf/history/

The following CGI have been discovered :

Syntax : cginame (arguments [default value])

/styles/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/images/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/assets/html/Stuff/images/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/backups/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/index.php (page [-1])

/swf/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

/swf/history/ (C=S;O [A] C=D;O [A] C=N;O [D] C=M;O [A])

. (keep [1] page [0] sort [friendlyID] image [5])

/swf/ConeExample.swf (timber [complete])

/includes/phpThumb.php (src [../assets/scale//F20.jpg] w [400] zc [1])

Directory index found at /styles/

Directory index found at /images/

Directory index found at /backups/

Directory index found at /assets/html/Stuff/images/

Directory index found at /swf/

Directory index found at /swf/history/

Nessus ID : 10662

HyperText Transfer Protocol Information

Synopsis :

Some information about the remote HTTP configuration can be extracted.

Description :

This test gives some information about the remote HTTP protocol - the version used, whether HTTP Keep-Alive and HTTP pipelining are enabled, etc...

This test is informational only and does not denote any security problem

Solution :

None.

Risk factor :

None

Plugin output :

Protocol version : HTTP/1.1

SSL : no

Pipelining : yes

Keep-Alive : yes

Options allowed : (Not implemented)

Headers :

Date: Thu, 26 Feb 2009 21:15:02 GMT

Server: Apache/2.2.11 (Win32) DAV/2 mod_ssl/2.2.11 OpenSSL/0.9.8i

mod_autoindex_color PHP/5.2.8

X-Powered-By: PHP/5.2.8

Content-Length: 2197

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html

Nessus ID : 24260

HTTP TRACE / TRACK Methods

Synopsis :

Debugging functions are enabled on the remote web server.

Description :

The remote webserver supports the TRACE and/or TRACK methods. TRACE and TRACK are HTTP methods which are used to debug web server connections.

In addition, it has been shown that servers supporting the TRACE method are subject to cross-site scripting attacks, dubbed XST for "Cross-Site Tracing", when used in conjunction with various weaknesses in browsers. An attacker may use this flaw to trick your legitimate web users to give him their credentials.

See also :

http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf

<http://www.apacheweek.com/issues/03-01-24>

<http://www.kb.cert.org/vuls/id/867593>

Solution :

Disable these methods.

Risk factor :

Medium / CVSS Base Score : 5.0

(CVSS2#AV:N/AC:L/Au:N/C:P/I:N/A:N)

Solution :

Add the following lines for each virtual host in your configuration file :

RewriteEngine on

RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)

RewriteRule .* - [F]

Alternatively, note that Apache versions 1.3.34, 2.0.55, and 2.2 support disabling the TRACE method natively via the 'TraceEnable' directive.

Plugin output :

The server response from a TRACE request is :

TRACE /Nessus17222.html HTTP/1.1

Connection: Keep-Alive

Host: localhost

Pragma: no-cache

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*

Accept-Language: en

Accept-Charset: iso-8859-1,*,utf-8

CVE : CVE-2004-2320

BID : 9506, 9561, 11604

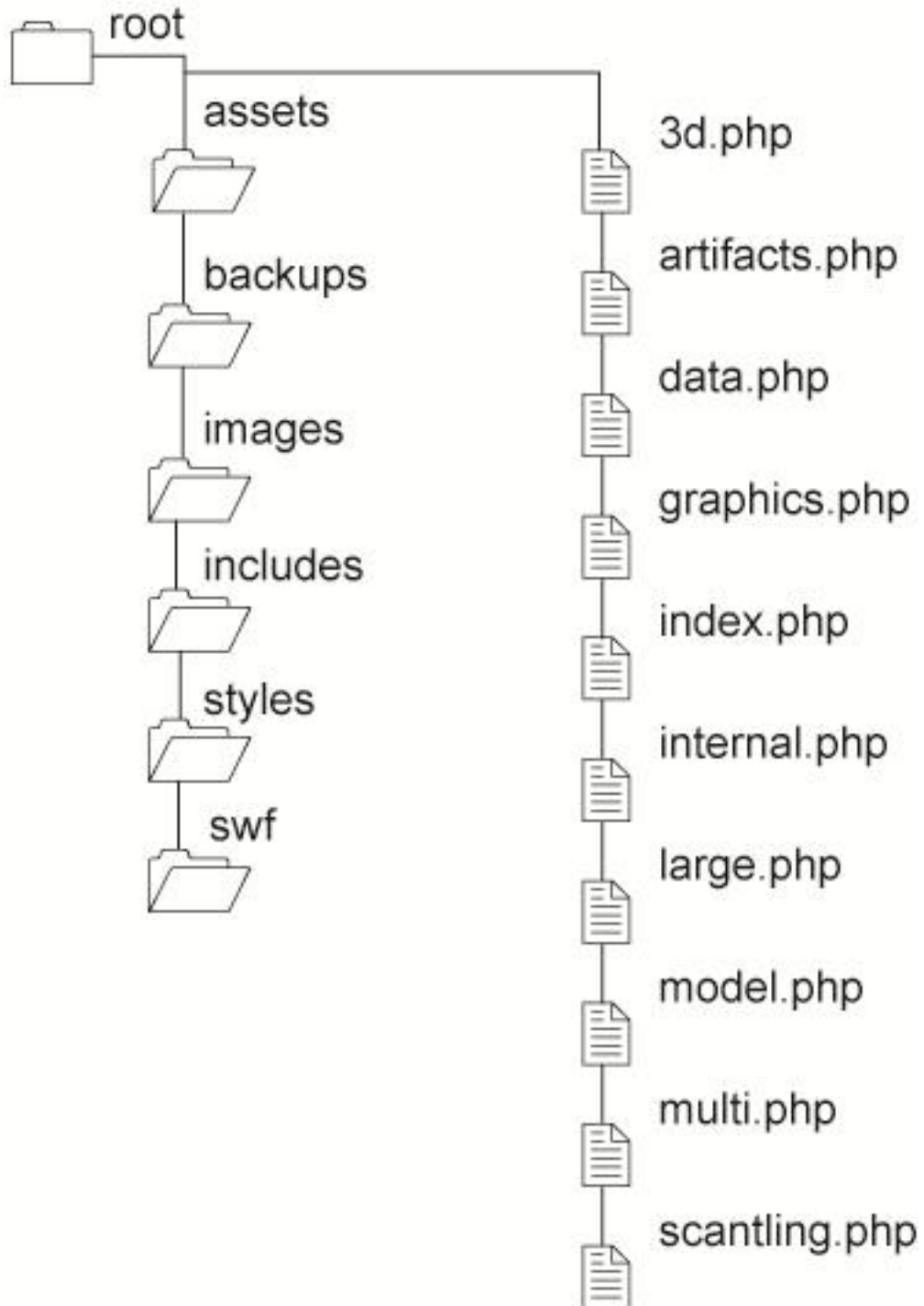
Other references : OSVDB:877, OSVDB:3726

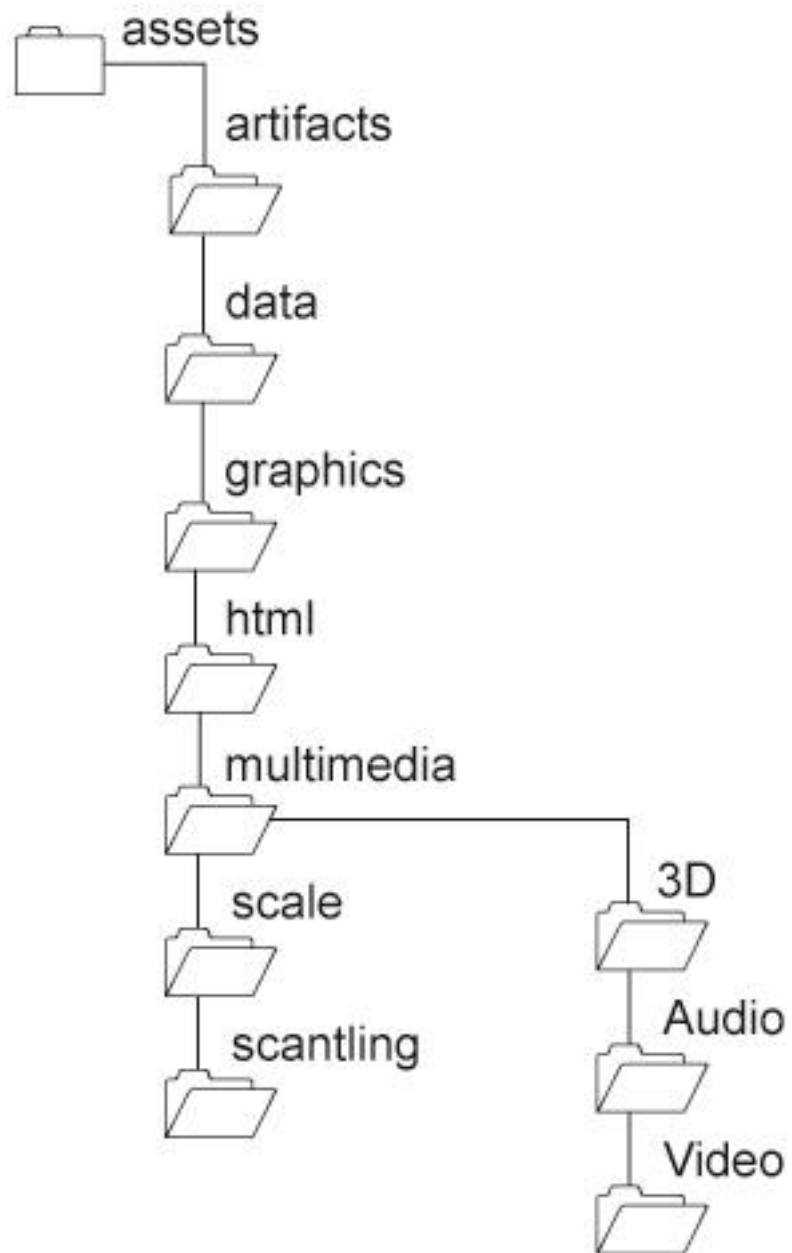
Nessus ID : 11213

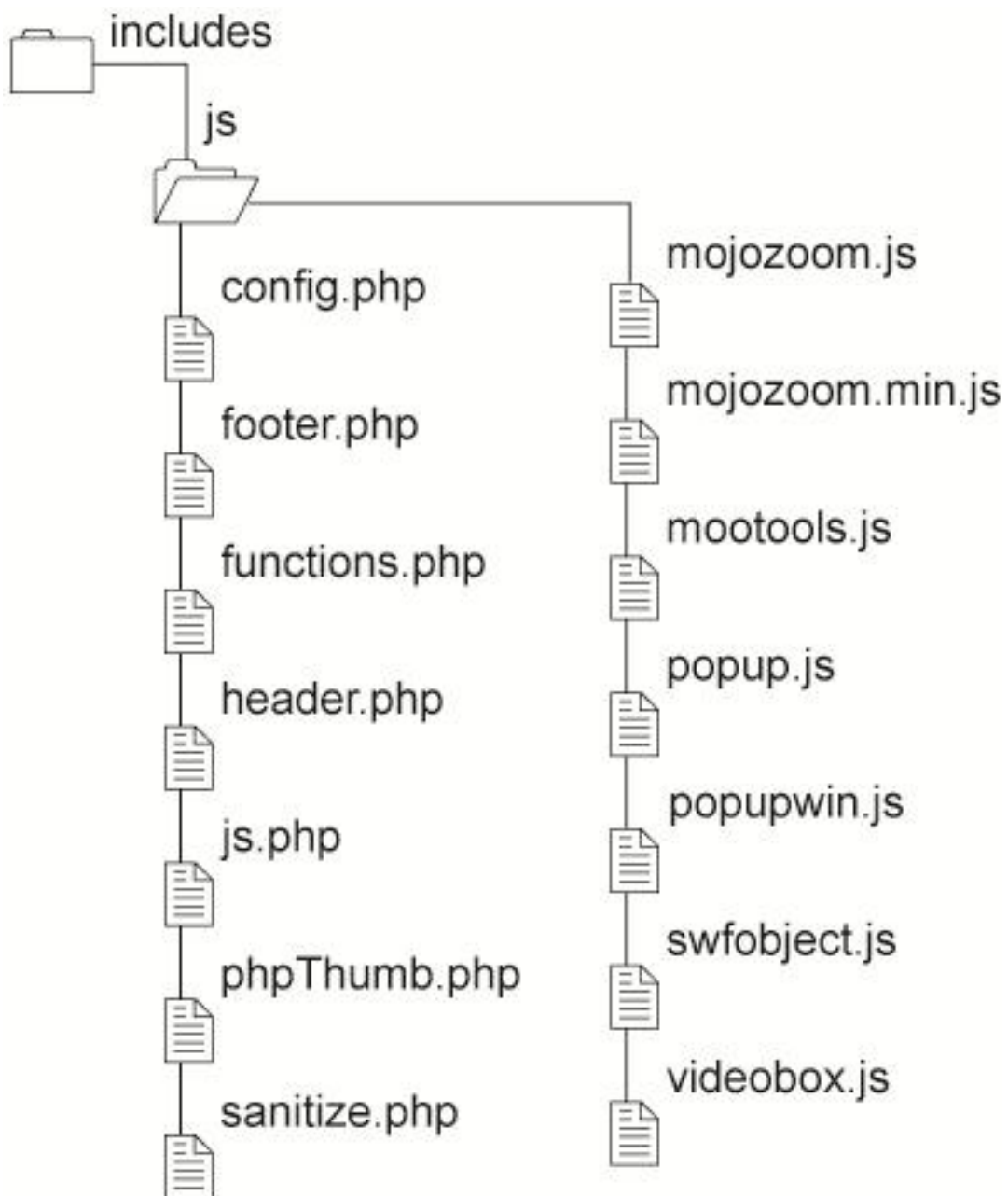
HyperText Transfer Protocol Information

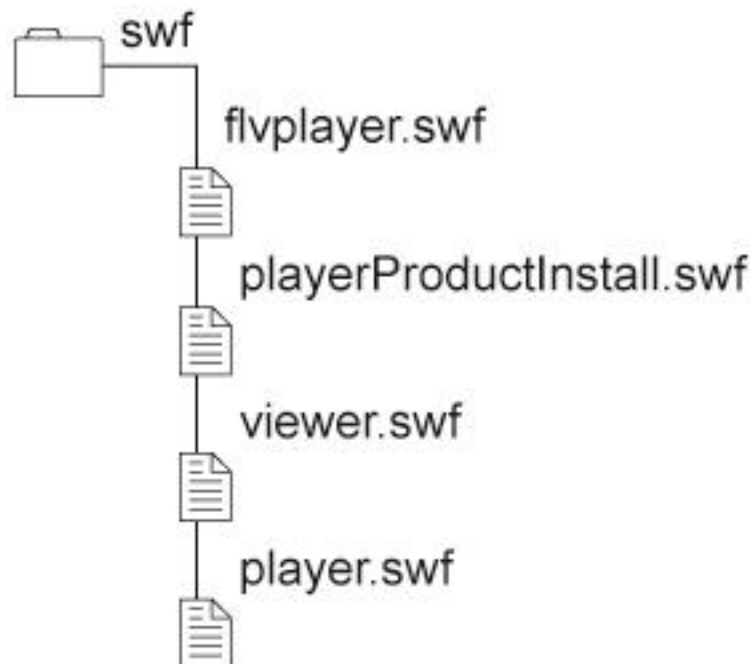
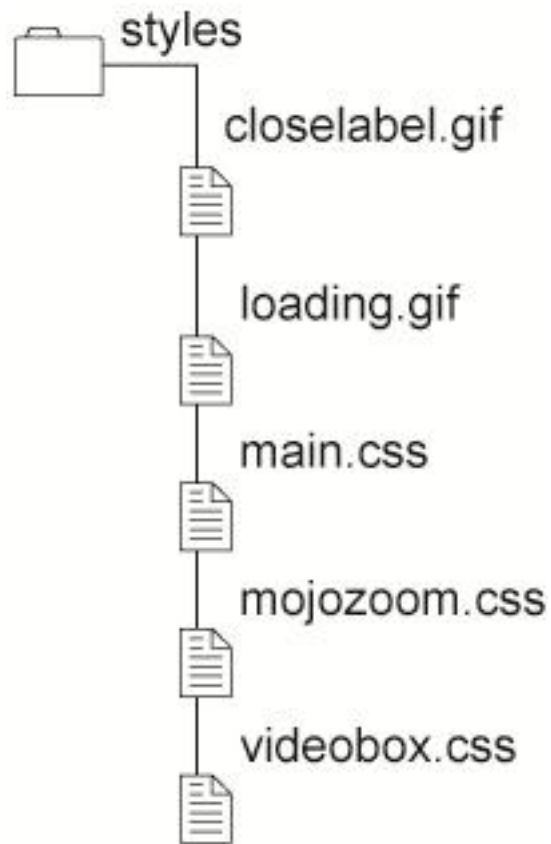
APPENDIX B

APPLICATION ARCHITECTURE









VITA

Samuel Allen Koepnick received his Bachelor of Arts in Anthropology at the University of Nevada Reno. During his time at Texas A&M he has received master level certifications in the fields of technical SCUBA diving, international affairs, and a variety of aspects in the field of computer science. His hobbies include travel, programming, writing, and long walks on the beach. Mr. Koepnick is a contributor to a number of open source projects involving multimedia applications and graphical design. His field experience includes surveys, excavations, and technical consultations in the Great Basin of the Western United States, Portugal, Spain, the Arctic, the Caribbean, and Egypt.

Mr. Koepnick may be reached at 4354 TAMU, College Station, TX 77844. His email is koepnick@gmail.com.

Name:	Samuel Allen Koepnick
Address:	Texas A&M Department of Anthropology 4354 TAMU College Station, TX 77844
Email Address:	koepnick@tamu.edu
Education:	B.A., Anthropology, University of Nevada at Reno, 2005 M.A., Anthropology, Texas A&M University, 2011