## FILE FORMAT EXTENSION THROUGH STEGANOGRAPHY

Presented to the Graduate Council of Texas State University-San Marcos in Partial Fulfillment of the Requirements

for the Degree

Master of SCIENCE

by

Blake W. Ford, B.S.

San Marcos, Texas May 2010

# FILE FORMAT EXTENSION THROUGH STEGANOGRAPHY

	Committee Members Approved:
	Khosrow Kaikhah, Chair
	Mina Guirguis
	Mark McKenney
Approved:	
J. Michael Willoughby Dean of the Graduate College	

#### **ACKNOWLEDGEMENTS**

First and foremost I would like to thank all of the family and friends, of whom there are too many to name individually, that have supported my studies over the past few years. A special thanks is in order to my wife, Kasey. Without her love and support, none of this would have been possible.

Thanks are also in order to the Texas State University-San Marcos faculty members who took the time and effort to review my work. Dr. Khosrow Kaikhah contributed significantly to this document as the committee chair and deserves extra attention.

This manuscript was submitted on April, 13<sup>th</sup> 2010.

## **TABLE OF CONTENTS**

Page
ACKNOWLEDGEMENTSiii
LIST OF TABLESvii
LIST OF FIGURESviii
ABSTRACTix
CHAPTER
1. INTRODUCTION1
2. AN INTRODUCTION TO DIGITAL STEGANOGRAPHY5
2.1 Digital Steganography Jargon
3. RELATED RESEARCH15
3.1 How Human Perception Affects Steganography Design15 3.1.1 Steganographic Obliterator15

3.2 Considering File Statistics	17
3.2.1 Profile Restoration	
3.2.2 Strategic Placement	19
3.3 Change Magnitude and Embedding Impact	20
3.3.1 Change Magnitude and Human Perception	
3.4 Non-Conventional Methods	
3.4.1 Collage Steganography	22
3.4.2 Taking Advantage of Metadata	
3.5 Basic Steganography Detection	26
3.5.1 Using Neural Networks	
4. FILE FORMAT EXTENSION THROUGH STEGANOGRAPHY	28
4.1 File Format Extension	28
4.1.1 Extensibility	30
4.1.2 Considered Alternatives to Steganography	
4.2 Pilot Extension Design Decisions	
4.2.1 LSB Packaging	
4.2.2 24-bit Bitmap File	34
4.3 Industry for Comparative Analysis	35
4.3.1 Computer Aided Design	
4.3.2 Alternatives to CAD	38
4.4 RoughDraft	39
4.4.1 Vector Bitmaps	
4.4.2 TurboCAD	40
4.4.3 Usability Features	
4.4.4 Secondary Format Features	41
4.5 Perception Research	
4.5.1 Background Information	
4.5.2 The Experiment	43
4.5.3 Results	
4.6 Industry Niche	48
4.6.1 Contributions	48
5. APPLICATION	50
5.1 Initial Design Requirements	50
5.1.1 User Interface Requirements	
5.1.2 Drafting Feature Requirements	
5.1.3 Steganography Feature Requirements	
5.1.4 Secondary Format Requirements	
5.1.5 Application Feature Requirements	
5.1.6 Future Use Feature Requirements	

5.2 Software Layers	60
5.2.1 bmpmanip	60
5.2.2 bmpmanipProxy	61
5.2.3 RoughDraft (UI Layer)	62
5.2.4 RoughDraft (Steganography Layer)	63
5.3 Technical Details	65
5.3.1 Shape Data Format (SDF)	65
5.3.2 Math in the Shape Calculator	66
6. RESULTS	69
6.1 Usability	69
6.1.1 Format Compatibility	70
6.1.2 Dual Mode Editing	73
6.1.3 Layering	74
6.1.4 Workarounds	75
6.2 Practicality	
6.2.1 General	79
6.2.2 Mechanical	82
6.2.3 Architecture	
6.2.4 Secondary Format	85
7. CONCLUSION	91
SOURCE CODE APPENDIX	94
REFERENCES	163

## LIST OF TABLES

Table	Page
6.1 File Format to Size (KB)	80
6.2 File Format to Size (KB) for Alternative Formats	

## **LIST OF FIGURES**

Figure	Page
2.1 Microdot Illustration	9
3.1 Collage Steganography Example	23
4.1 Conversion Illustration	37
4.2 Shade Test Example	45
4.3 Maze Test Example	46
4.4 Modified Maze Test Example	47
5.1 TurboCAD Screenshot	53
5.2 RoughDraft Screenshot	59
5.3 RoughDraft's File Menu	59
6.1 Layering Illustration	75
6.2 Control Pixel Workaround	76
6.3 Secondary Image	78
6.4 File Format to Size (KB)	
6.5 File Format to Size (KB) for Alternative Bitmap File Formats	82
6.6 Complex Shape Creation	84
6.7 Electrical Symbol Examples	85
6.8 File Size to Encoding Density	
6.9 Steganography Algorithm Comparison	
6.10 File Format to Size (KB) with Vector JPEG Format	

#### **ABSTRACT**

## FILE FORMAT EXTENSION THROUGH STEGANOGRAPHY

by

Blake W. Ford, B.S.

Texas State University-San Marcos

May 2010

SUPERVISING PROFESSOR: KHOSROW KAIKHAH

Several file formats in common use can be significantly compressed or refined without noticeable losses in integrity. While various techniques have successfully been used to reduce the footprint of these formats, uncompressed files continue to be readily available in modern systems. The present study repurposes the extra space provided by an unrefined format to store metadata about the file in question. This process does not negatively impact the original intent of the format and allows for the creation of new derivative file types with both backwards compatibility and new features.

#### **CHAPTER 1. INTRODUCTION**

Simple interactions between engineers and their clients can at times be surprisingly difficult, because most contemporary drafting programs use proprietary technologies to archive data in application specific formats. When sharing data with clients, source files either need to be converted to a format in common usage or the client will need to install an application specific viewing program. In either case, the maintenance associated with keeping the client up to date can be tedious. This research resolves the sharing dilemma.

Developing protected formats provides IP shareholders with a number of marketable advantages, so custom formats remain prevalent in the industry. Not disclosing some aspects of a software product helps organizations gain exclusive control of their IP, so specifications for proprietary file formats are not made publicly available. Hiding information creates a barrier to completely understanding a piece of software's operation. This allows software organizations to uniquely tailor their custom format to benefit the software appliances they choose.

These benefits are particularly important when a file is licensed to an outside vendor. Under the parent institution, the cost of the rights to use a

proprietary file format for a subsidiary may be heavily discounted or free.

However, when outside organizations license a proprietary file format for their own use, fees may be used to balance the competitive marketplace. License holders may also be subject to additional restrictions imposed by the format's owners. These restrictions can artificially make the proprietor's applications either faster or more functional than those of their licensed competitors.

Proprietary file formats have limited supportability due to their restrictive nature. Computer systems instead rely on a series of standardized data interpretations. Standardization promotes interoperability and quality in a system by providing a uniform engineering norm to developers in an open forum. These formats are normally designed by committee and represent what has been decided by several organizations as the lowest common denominator. When software providers agree on a compatible standard, they suspend their open discussions on how to use the new format. The files produced by these committees often focus on ease of interpretation and do not normally contain requirements for creation. Bitmaps, for example, store image data in a linear string of pixels. This model is easy to interpret and imposes no restrictions on how the pixel information is generated. The individual business units determine the process of creating and adding value to the standardized file. Each organization will design a business model and product that they feel best fits the marketplace. For bitmaps, most applications allow users to draw an image using a variety of shapes, but the experience varies subtly from editor to editor. The

authors of each application define its specific behavior, and this creativity invites deviation. By allowing organizations to develop heterogeneous editing experiences a format can be adapted to work well in a variety of environments.

If designers roll their value-adding IP into a proprietary format, it usually serves as the native editing environment for a developer. When the editing process is finished, the host application exports the data to a standardized format. This allows an organization to get the benefits of both privatization and standardization.

Alternatively, a software group may also choose to openly publish their intermediary file format. Open file formats provide different benefits to end users, like flexibility from specific software vendors and increased interoperability between programs. A feature rich open file format motivates improvements outside of proprietary file formats from individual software vendors. Smaller organizations may use open formats to break down competitive barriers.

Software groups standardizing on an open file format need strengths, like improved autocompletion, superior UI design, or lower cost, in areas not related to the way the application specific data is saved.

To solve the file-sharing problem, this project integrates high-level design data into a standardized file format while maintaining backwards compatibility. A digital steganography technique was used to create an outlet for adding additional hidden information to the standardized file. Using this process, it is possible to build a single source format that is convenient for clients and

workable for developers. The implementation defined in this research is an open standard; though, the process could work equally well under privatization.

#### **CHAPTER 2. AN INTRODUCTION TO DIGITAL STEGANOGRAPHY**

#### Overview

Chapter 2 gives the reader a brief history of steganography. It provides background information on terms and concepts used in the field of steganography.

### 2.1 Digital Steganography Jargon

In order to better describe some of the specific scenarios within digital steganography, it is helpful to understand some common terms in advance.

### 2.1.1 Stylistic Steganography Terms

*channel* - refers to the type of cover file (ex: image).

*carrier* - the area of a file in which the data is to be hidden.

payload - the data to be embedded into a carrier.

package - the modified carrier data within a file. Also known as a message.

<u>encoding density</u> - the percentage of file elements(ex: pixels) used to embed a payload.

#### 2.1.2 Detection Terms

suspects - a group of files suspected to contain a payload.

<u>candidate</u> – a file identified from a group of suspects by some form of steganalysis.

#### 2.2 The Origin of Digital Steganography

Steganography is the science of concealing a hidden message in plain sight in order to avoid detection. In contrast to cryptography, messages sent using steganography techniques are designed to be inconspicuous. Traditional encrypted messages provoke decryption attempts by drawing attention to their irregular form. Strategically, this is like sending a tank onto a battlefield. The vehicle is clearly identifiable as a tank, yet it is still difficult to neutralize. In contrast, steganography is a stealth technology. The phrase most often used to describe this approach is, "Security through obscurity".

Invisible ink, ink that is invisible during or sometime shortly after application, is among the most relatable forms of steganography. Invisible ink is designed to dry invisible but become opaque under some certain process. Given the proper information an informed recipient can recover a previously invisible message written with the substance. This basic concept can be found in all forms of steganography.

#### 2.2.1 Physical Forms of Steganography

Physical steganography can be found in applications ranging from espionage and anti-counterfeiting to magic shows. Early forms have been practiced since the days of ancient Greece. This is revealed in *The Histories of Herodotus*, which contains some of the first recorded accounts of physical

steganography.

The physical manifestations of steganography are innumerable. These include secret inks, messages written on the adhesive side of stickers, collaged art, and small dots or tears on a fake cover letter. While some of these techniques are considered novelties, physical steganography has been used consistently in real world applications since historical times.

Physical steganography often becomes vital to organizations in war torn European countries working through insecure postal channels. Microdots in particular have been effective in avoiding mailroom screening. Reducing full size images to a very small size creates a single microdot. To the naked eye, the shrunken image looks like a solid colored speck on a sheet of paper. When the message reaches its intended recipient, the original image can easily be recovered using a microscope. The standard magnification required is 40x.

Drafting a cover letter and placing several microdots within punctuation marks has proven to be an effective mask for secret messaging since World War I.

Today, various corporations have modernized the use microdots in order to security tag their goods. The following image depicts an example of a microdot.



Figure 2.1: Microdot Illustration. [7].

#### 2.2.2 Emergence of Digital Steganography

In the mid-1980s, the power available in personal computers became sufficient to solve classic steganography problems, which has greatly contributed to the development of many different niche programs and techniques.

Additionally, new domain specific process and properties have allowed the science to move in previously unimagined directions. The resulting field of digital steganography is now a diverse area of science that can take on many different forms.

Digital steganography can be applied to a diverse range of file formats and communication standards used today. The key to using any digital steganography technique is introducing the right amount of skew into the file or service. Skew can be defined in many ways, but generically skew can be considered as changes to the source. Gating the skew of the data as well as the

vital statistics of the information being transferred are important when using digital steganography.

High Definition Compatible Digital (HDCD) is an example of how steganography can be used in digital audio. HDCD is a patented digital steganography process that allows for compact disc playback on both standard players and more advanced players that allow for a greater level of audio quality. This dual behavior is achieved by packaging the equivalent of 20 bits worth of data in a 16-bit space. Normal players only interpret the 16 bit data while the advanced players can derive an additional 4 bits of data from the same space by using specialized audio filters.

#### 2.3 Techniques in Digital Steganography

Digital steganography, like physical steganography, can be employed using a variety of techniques. Each has advantages and disadvantages and it is important to be aware of different packaging styles when considering how to deploy a custom application. For example, some applications may require greater encoding density, while others require greater security. Choosing the wrong technique for an application can easily render the whole process ineffective.

#### 2.3.1 Least Significant Bit

The Least Significant Bit packaging method, or LSB, is a popular form of digital steganography. LSB is an excellent example of how steganography can be

used with digital audio or image files. This technique maintains the size and profile of the source file adding to the robustness of the hidden message. Most media formats are designed with the outer limits of human perception in mind, which makes LSB the pattern of choice for packaging messages in these channels. LSB has been shown to be quite versatile and the implementation is straightforward. All of these factors contribute to the continued adoption of LSB in steganographic applications. The primary objective when using this method is to barter a marginal amount of image quality in order to create undefined space within the carrier space.

#### 2.3.2 Information Exchange

LSB tends to distort global file statistics and preserving these values is crucial to some steganography applications. In images color consistency, the number of times a pixel is present within a file, is one of many global file statistics. Information exchange saves global file statistics by preserving a channel's original data set and packaging hidden data using reorganization. The file is first analyzed for carrier data that represents chunks of information within the payload. Once appropriate blocks are found they are swapped with other data blocks of relative likeness within the image. Carrier data are always reorganized in identical logical blocks. For example, in a raster image pixels can be used as the logical block. When executed correctly, a reasonable cover file remains present and many of the overall file statistics are maintained. Unlike

LSB, if a user attempts to verify a host image using color consistency after it has been packaged using information exchange, the modified file will pass verification.

#### 2.3.3 Chaffing and Winnowing

Corn farming was the inspiration for steganography by chaffing and winnowing. In farming, chaff is the name given to the husk and other inedible parts of the stalk, and winnowing is the process used to separate the chaff from the more desirable parts of the plant after harvesting. In digital steganography chaff is used to describe useless cover data intended to obfuscate the payload data. To encode a transmission, the payload is interleaved into chaff to create a believable cover. The normalized information is then sent to the recipient. When the message arrives to the receiver it is their responsibility to winnow the data to extract the payload message. To determine which data are valuable and which data are not the sender and receiver should agree on a dictionary of words or phrases that are valid for secret messages before the transmission. Any words or phases not included in this dictionary are regarded as cover data.

#### 2.4 Steganography Detection and Protection

Steganography can be a tool for espionage and unlawful deceit. For this reason, corporations, online communities, and even governments have started to defend themselves with anti-steganography processes and software.

Understanding the security environment is essential when creating an operational package to avoid programmatic screening.

#### 2.4.1 Steganalysis

Programs that filter suspect files into candidates are said to use steganalysis. These tools detect faulty statistics within viable steganography channels. This is accomplishable in a variety of ways including random carrier data sampling. Some are even designed to reverse engineer a packaging process, revealing the payload to unintended parties.

Suspect files should exhibit some form of statistic consistency within themselves and in some cases within a group. Random carrier data sampling is a process used to test a file's relative consistency. The sampling test is administered by randomly comparing potential carrier data to other similar sites within the same file or set of files. Large contiguous areas of inconsistent data will result in the failure of the test. Failing a sampling test is one indicator that a suspect file is a candidate.

LSB for example, though typically imperceptible to humans, is often discoverable by steganalysis. Research in finding and eradicating LSB payloads is aggressive, because as a packaging method LSB has favorable carrier data and encoding density characteristics. Even with advanced iterations of the LSB method, outlying statistical artifacts are typically evident.

#### 2.4.2 Countermeasures

Detecting a package with steganalysis tools is limited and success is not guaranteed. Using simple countermeasures that discourage packaging payloads may be more reliable and preferable if detection proves ineffective.

Prohibiting or closely scrutinizing the transfer of files known to contain vulnerable carrier data is one high level countermeasure. If all of the approved files in a work environment have very limited encoding capacity it may be impossible to covertly transmit a proper message.

A countermeasure called grouping can also be used to increase the effectiveness of a steganalysis tool. If a large set of suspect files exists, logically grouping the files and batch processing smaller sets can improve the odds of detection. One grouping strategy is to group files that derive from the same source into one set. If a group of files are from the same video camera for example, their carrier data should exhibit similar statistics. Any files deviating from the composite pattern would be marked as candidates.

#### **CHAPTER 3. RELATED RESEARCH**

#### Overview

This chapter is devoted to illustrating how outside research in the field of digital steganography has affected this project.

#### 3.1 How Human Perception Affects Steganography Design

Standardized media file formats are often designed to represent data in ranges near or slightly beyond human perception. This characteristic makes these files desirable for encoding data, because small changes to specific areas of the file are unobservable under normal circumstances. In general the area of choice for placing a hidden message is the least significant bits of a given data representation.

#### 3.1.1 Steganographic Obliterator

Steganographic Obliterator (Francia and Gomez, 2006) is a proof-of-concept steganalysis program designed to remove all possible hidden data from various common hosts like image and audio files. The technique shown in the application can be applied to various media formats and is designed to prevent the transfer of malicious, or illegal steganographic data. The program uses an

extremely brute force approach by completely destroying the user specified number of least significant bits within a file. The application allows the user to destroy a subset of the least significant bits for supported file types. The areas cleared are the most likely carriers for a payload given the file type selected. Selectively erasing data in this way requires intimate knowledge of the host file type in order to prevent file corruption. Removing bits from the host file will result in a perceivable amount of data skew from the original, but the tradeoff from a security prospective is quite small. This tactic does not attempt to detect or recover steganographic data from a file and performs the same algorithm on the potential host regardless of whether or not suspect data is present. The research indicates that since it is common for embedded data to be placed in random order and encrypted, detection may be near impossible. Obliterator's designers suggest that given the amount of confidence a user can have that any unintended data has been removed from a file, these consequences are worth the results. So, while the Steganographic Obliterator's technique is aggressive, it does provide a blanket solution with high success rates for removing illicit data with minimal impact to a host file.

This method is simple, generic, and effective. Clients filtering only supported file types and able to handle the side effects of this process will be successful in stopping most if not all information trafficking, especially when using an automated deployment system.

How this research benefits our project:

- Qualifies ideal data encoding range within a given byte
- Shows the limits of LSB security countermeasures
- Contends that slightly perceivable differences are acceptable
- Demonstrates security automation
- Discusses the difficultly in securing multiple formats

#### 3.2 Considering File Statistics

Linear placement is a term used to describe the process of embedding hidden data in a continuous fashion from the earliest point possible. Linear placement has the tendency to give a host file awkward characteristics when audited for atypical patterns. An example metric used to reveal hidden information in a host file is the number of times a particular color occurs in a picture. For this reason, security-minded programs typically have no trouble detecting linear placement, and as such more advanced responses to the datahiding problem have evolved to overcome the detection problems associated with this practice. The final data manipulation techniques are similar, but the actual placement is determined by using calculations that are precisely tailored to the medium at hand and the security protocols the files are likely to face.

#### 3.2.1 Profile Restoration

Profile restoration involves using only part of the allotted steganographic space for hidden information. The space that remains is then used to perform corrections to the overall statistical profile of the file in order to avoid detection. The authors of the research project, "Statically Undetectable JPEG Steganography", (Fridrich, Pevný and Kodovský, 2007) observed one profile restoration method that partitions the prime encoding space within a host file in half. The first half is used to encode embedded data in a linear fashion, and the second for statistical repairs. When reviewing the file in its entirety, no blemishes will be apparent and it is likely the hidden data will pass through a high-level profile screening procedure undetected. Irregularities can be found when batch processing the file in sections however, so profile restoration is not the best solution for some security environments. Additionally, not all files are candidates for profile restoration. A new host should be chosen if the profile of the current one is too unique. A solid color photo is a good example of a poor profile restoration host, as any changes to a solid color file will be unrestorable in terms of color consistency.

Profile restoration is an effective countermeasure when used properly and the security environment is well understood. Not accounting for the right statistics affects programs using restoration, and in some cases restoration may not even be possible. Restricting a prototype application to encoding data within the boundaries set by previous restoration researchers makes testing the practicality

of an embedded data format while not limiting the use of this mechanism possible.

#### 3.2.2 Strategic Placement

Another sub-category of note is strategic placement. This concept provides a different form of statistical soundness and also aims to impact the user visible data less than profile restoration, which potentially touches the entirety of a given file.

When embedding hidden data using strategic placement, a texture finding routine is run using the host file's data to determine the placement sites for the encoded message. Textured areas highlight irregular churn within a media file and are the building blocks of strategic placement. By placing the implanted data within the textured swatches of the file, the message is likely to get mixed in with the noisy bits that surround it and go unnoticed.

The effectiveness of this method is highly dependent upon the number, size, and detail of the textures available. Fewer, less complicated textures will lead to lower embedding capacity. For this reason, strategic placement does not work well as a generic solution. Screening for an acceptable texture to message ratio, or arbitrarily inserting texture into a potential host file eliminates this issue.

How this research benefits our project:

- Highlights deeper embedded format restrictions
- Shows techniques which make detection difficult
- Calls out problems with some host files
- Shows the need for adaptation to security environment

#### 3.3 Change Magnitude And Embedding Impact

Research in change magnitude has one fundamental goal: to determine whether or not it is better to make fewer, large changes or many, small changes to a host file when embedding secondary data.

#### 3.3.1 Change Magnitude And Human Perception

The research paper "Minimizing the Embedding Impact in Steganography," (Fridrich, 2006) looks into the topic of change magnitude using linear placement. Under this constraint, the paper states that in order to minimize embedding impact on a host file two primary suggestions should be employed. First, the embedding application should purposely use up to 25% more space than what is required by the message itself. This is an easy way to interleave and break up the embedded data within a host file thus creating a more realistic looking result. Secondly, Fridrich, the researcher on the project, suggests that using more small changes is advantageous to making less, more drastic ones. The individual units of information that make up the hidden

message should be as granular and as minimally invasive as possible despite the fact that this may result in more changes to the host file.

The project was able to show some practical advantages to using smaller changes and extra space. However, the research does not acknowledge useful results for adaptive placement variation at the time of publishing. Also, the change size rules discussed in Fridrich's research are only meant to decrease human perception of possible changes. Depending largely upon what kind of steganalysis tools are deployed at a current location, these data modes will have varying levels of impact on the likelihood of programmatic detection.

How this research benefits our project:

- Shows how to greater obscure human detection
- Place further restrictions on amount of data embedded
- Helps qualify the types of host file changes to make

#### 3.4 Non-Conventional Methods

The competition between steganography and steganalysis application developers can be equated to small-scale arms race. Each side encourages the other to aim higher by constantly improving their underlying technology. The result is that many applications from both sides become ineffective and need revision after a relatively short period of time. This has lead to a tertiary area of study focused on hiding data from those technically proficient in the world of contemporary digital steganography.

#### 3.4.1 Collage Steganography

Conventional steganography research focuses on changing a host file while maintaining most of its original characteristics. This includes outward, human perceivable traits and statistical, programmatically detectable profiles.

Collage steganography challenges the conventional approach by building a new file from the host image with different outward characteristics, which are statistically sound during analysis.

"Generalized Collage Steganography on Images," (Mei-Ching Chen, Agaian and Chen, 2008) purposes a simple collage application that requires two primary image repositories. One repository holds what is considered as the host files for this method, and the other holds the data files. A dictionary is constructed using relative terms to connect the host image files with a subset of the data

image files. Finally, another map is created to hold the messages associated with each data image.

When the user is ready to create a hidden message a special algorithm is run. This procedure determines the single best host file and appropriate accompanying data files to create a seamless, layered image (based upon data to encode, size, relative keywords, color palette, blank space, etc.). Once a proper host is chosen, the data files are placed at specific coordinates over the base image in order to represent the encoded data. The end result looks something like a child's sticker book. To decode the message, a receiver would need the data images map, and access the data image repository. The key to effective collage data images is to include transparent backgrounds to prevent unsightly edges in the final product.



Figure 3.1:
Follow from left to right-->
The chosen cover image is analyzed for blank space. Depending upon the message to be encoded, different related images are placed over the cover image.

Collage steganography is extremely covert and capable of fooling very advanced stegoanalysis tools. If a client's application needs to preserve the

outward characteristics of the host file however, then this method cannot be used. For most applications, the largest inhibitor to using the collage proposal over convention steganography is the relatively low embedding capacity.

Generalized collage steganography can be used to decrease the effects of this limitation. The generalized method improves on the traditional collage approach by using a hybrid of both collage and conventional steganography.

#### 3.4.2 Taking Advantage of Metadata

Metadata is space within a typical file format reserved for future use or otherwise not directly visible to the user. These areas can also be used to input steganographic data.

Cantrell and Dampier studied the practicality of applying the metadata tactic to common office documents from Microsoft and zip files (Cantrell and Dampier, 2004). The effort shows that this method maintains file size, and is more robust than least significant bit encoding overall given that common editors do not allow for these data to be edited. Administrators and developers may even be unaware that this safety loophole exists, as there is not much training in the area of file metadata. As such, an additional benefit from an anti-security standpoint is that the metadata space is often not a target for steganalysis programs. The team also showed that data could be successfully added to a file past the end of file marker in zip files, but the process was found to be impractical. After the process, file size is not maintained and hidden data tend to

be more susceptible to corruption, as external editors will not consider data past the file marker as part of their normal operation.

Metadata space is incorporated into many file formats, and appears to be an acceptable placement site for hidden messages. A major flaw with this system is that the unique file spaces needed as part of this process are small, and rare within a normal file format. Cantrell and Dampier recommend encoding a single message across multiple files in order to work around this issue.

How this research benefits our project:

- Gives a different perspective
- Points out untapped data resources
- · Gives our approach greater data capacity
- Shows how to keep crucial data detection safe
- Reinforces intimate file knowledge principle
- Discourages adding steganographic data at the end of a file

#### 3.5 Basic Steganography Detection

Conventional steganography is detectable in a variety of ways. Some clues to the payload's existence and the specific packaging method are almost always visible under close analysis of the host file.

#### 3.5.1 Using Neural Networks

"Detection of Steganography Inserted by OutGuess and StegHide by means of Neural Networks" (Oplatkova, Holoska, Zelinka and Senkerik, 2009) discusses the effectiveness of using neural networks to detect whether or not hidden data had been implanted into a host file. The project is intriguing, because it uses only readily available software tools to perform both the embedding and the detection.

Steghide and OutGuess are two popular, open source steganography programs capable of hiding data in various media file formats. These two programs take different approaches to storing hidden data and are staples in the study of both digital steganography and its detection. These applications were used to develop files with hidden content for the neural network related research paper. The messages encoded ranged in size from 0.04% to 6% of the host file size, well below the normal safe threshold defined as 20%.

In order to detect the marked images, a multilevel approach was used.

The first step in this process was to develop useful training sets for the neural network. Experimentally, it was determined that using Huffman's coding extracted

from the clear and tainted images could serve as proper input into the system. In order to obtain this information, the program JPEGSnoop was used.

To perform the actual analysis a feedforward neural net with one hidden layer was shown to be the most effective. The tool chain Neural Networks for the Mathematica environment was configured with those settings, and the input data were processed.

While unable to decode the hidden messages, the study showed that the method is promising for detection. By utilizing readily available off-the-shelf tools, this project could make an impression on any organization looking to avoid some sort of steganographic attack.

How this research benefits our project:

- Illustrates steganography's digital footprint
- Validates research based on Steghide and OutGuess
- Gives ranges for proper embedding thresholds

### **CHAPTER 4. FILE FORMAT EXTENSION THROUGH STEGANOGRAPHY**

### Overview

File format extension is an attempt to solve deployment problems associated with current file formats without creating a new revised standard. This chapter provides an overview of the challenges and benefits associated with extending a file standard.

#### 4.1 File Format Extension

Maintaining software compatibility while expanding the feature set on a legacy code base is a persistent challenge in the field of computer science.

Certain assumptions or restrictions placed on the original project are likely not to apply years after a product's release. These limitations need to be addressed in order to extend the life span of the product.

If compatibility is not an issue, developers can introduce a new standard with an updated API and features that reflect current programming trends. In the case of file formats, typically this approach is favored only if the original standard has seen poor adoption rates. Once a format gains notoriety in the programming community, developers are hesitant to move to new standards despite the

addition of an improved feature set. For IP holders, supporting a new file format requires non-trivial design work, implementation, and testing responsibility. There needs to exist some assurance that the format will make a client more productive to justify the effort. This concern may even extend beyond an IP holder's product line. The proposed benefit to other related software suites could be of equal concern when making the decision to support a new format due to the large demand for interoperability. Digital products are rarely developed using a single application. They come together more like a machine; there are many moving parts.

Compatibility is highly prized, so file extension is exponentially more desirable than starting a new format from scratch. File format extension is the process of creating a hybrid file format that addresses gaps in the parent standard while retaining backwards compatibility. Format extension preserves the conventional usability of the legacy standard and integrates previously unsupported features. Extension is preferable to file format revision, because it increases the likelihood of adoption by leveraging existing technologies.

Backwards compatibility allows users to opt in to new features at their own pace while still giving them access to the information they are used to receiving. In contrast, if a developer starts creating content using a new standard, their clients are forced to upgrade their support tools immediately.

## 4.1.1 Extensibility

When adapting an old file format for new purposes, developers have to work within the constraints of the existing standard. In some cases, clear methods for expanding the format may have been laid out during the design phase avoiding this issue. This design principle is called extensibility, and it defines some metric to determine the amount of effort required to implement extensions.

It is nearly impossible to seamlessly merge two divergent technologies into one cohesive unit. The flying car serves as a good example of the problems with this idea. These vehicles are capable of traveling on conventional roads as well as in the air. Independently, each of the parent products (automobiles and airplanes) is well suited for the tasks they need to perform. However, no elegant solution that combines the two has been found. The resulting product is either too plane-like or too car-like and typically performs in a poor fashion in either domain. Similarly, backwards compatibility is one of the most appealing aspects of extension, but often the parent format is designed without proper extensibility making it a hindrance to the new format.

Contemporary digital steganography techniques can be used to solve integration challenges in files with low extensibility. By committing to this solution, the restrictions proposed by previous steganography research must be compatible with the requirements for the new extension. Using a standard steganography technique forces the end user application to contend with the

space, security and other tradeoffs associated with the chosen packaging method. Assuming these requirements can be met, backwards compatibility is assured and the developer can focus solely on creating the new extension. If the intention of this study is to present a practical form of extensibility using steganography, then it is paramount to ensure that the end results take all of these requirements into account.

## 4.1.2 Considered Alternatives to Steganography

Our solution to the CAD file compatibility problem stated earlier is a program called RoughDraft. RoughDraft uses Least Significant Bit packaging to extend standard bitmap (.bmp) image files. The bitmap file standard does not contain any extensibility features that could be utilized for this task, but the format is readable by almost any piece of computer equipment available today.

Normally, LSB is used to create space to package data unrelated to the image file, but in this project it is used to store CAD metadata. This extension expresses the raster data as a vector graphic. More on this topic will be discussed later.

Outside of revision and steganography, other proposals for creating this extension were also considered.

The simplest extension concept reviewed was to append the necessary data to the end of the image file. Adding extra data to bitmaps is possible and could be implemented in other projects, however it was rejected while designing RoughDraft. Adding data to the end of file is not a creative or elegant solution.

Because doing so risks future file corruption and bloats the file size, this technique was considered inadequate.

Advanced image processing was the other serious alternative considered for use in RoughDraft. Using this approach, RoughDraft would have produced a normal bmp file from the initial set of CAD data generated in the application. The vector data would be reconstructed each time the file was opened in RoughDraft by some image processing routine that traced the image into solid lines and shapes. Image processing avoids the problems with file size, and while appropriate, it is not without issue. The largest problem stemmed from the amount of implementation effort and concerns about supportability. The transformation from a normal file to vector graphic objects using only image manipulation seemed inherently broken and inaccurate. Some programs, like Potrace, exist to do this statically, but none were found that performed this operation continuously. The end product could have lacked robustness without hard data. Additionally, the implementation would have been much more complex and therefore more error prone.

### 4.2 Pilot Extension Design Decisions

Some research has been done on the development of new file formats that would include steganography as part of their original design, but to our knowledge no studies are looking into the prospect of using this technology to extend files in a backward compatible way. Ultimately this project is an attempt to

depart from the traditional and deceptive uses of steganography, and to solve new problems with the process. To prove the value of file extension through steganography, one distinct use case was chosen. The decisions involved in limiting the research to this application are documented in the next section.

## 4.2.1 LSB Packaging

As stated previously, RoughDraft utilizes a common form of digital steganography called Least Significant Bit (LSB) stegangraphy. In LSB data are hidden within the lowest bits of noisy sound or image files. The key difference between this approach and other software variants is the packaging of metadata into the original data set. Most research in LSB is focused upon embedding orthogonal data, which is intended for use in an application outside of the one used to interpret the host data. In this study, we use the hidden data in conjunction with original content to change the editing experience of a standard image to mimic that of a conventional CAD application. By using LSB, the image should maintain backwards compatibility with existing bitmap viewing applications.

The LSB packaging problem has been studied extensively, and the advantages and disadvantages are understood well. For this reason, the decision was made to use LSB as the mode of steganography for this project. Some advantages to using LSB include compatibility with a wide range of popular file formats and the preservation of the original file size.

## 4.2.2 24-bit Bitmap File

Bitmap files are simple and support for the format is bundled into most operating systems. These two factors are the primary motivations behind the adoption of this particular image standard.

24-bit bitmap files consist of a fifty-four-byte header followed by uniform data blocks for each pixel in the image. In a standard bitmap, each pixel will be composed of three single color elements, red green and blue. The 24-bit designation signifies that each of these elements will be represented by a full byte of data. Other bitmapped files, like PNG, were considered, but their complexity distracted from the basic scientific demonstration in this research.

Bitmap files are viewable on almost every visual computer platform. This includes web browsers and cellular phones as well as common desktop operating systems. Windows, Mac OS, and many Linux distributions natively support bmp viewing and editing. Given the age and diverse operating system compatibility of the format, bitmaps are also familiar to most computer users. By adopting a simple, common standard in RoughDraft, the explanation of the underlying process is presented with more clarity.

The format chosen for RoughDraft also needed to work well with LSB steganography. LSB Packaged 24-bit bitmaps frequently store data with a density of two to three bits per pixel. This allows RoughDraft to store relatively large amounts of secondary data while keeping the channel's data skew low and undetectable.

Admittedly using bitmaps may not be the optimal solution for deploying this technology. One problem with consistently using bitmap files is their size. Compared to other compressed image formats, a bmp image is typically much larger than the same image rendered in another format. In the final application, the PNG or JPEG file format may be preferred, despite lower encoding density. The transfer of bitmap files over networks or the Internet significantly slower when compared to either of these standards. When deploying a large-scale system using the process described in this research, hard disk space may also become an issue. Despite the drawbacks, some of the symptoms can be treated. The slowdown in Internet transfer rates may be unperceivable given a fast enough connection and a reasonable file size, and file size can be reduced using normal compression methods like zip or tar. The problems and solution will be discussed in greater detail in the following chapters.

### 4.3 Industry for Comparative Analysis

The concept behind the CAD bitmap extension theoretically could work for any file format supported by contemporary steganography tools. Most of these formats are standards for transferring media, like images and video. For this research, the CAD industry's standard file types were challenged by the custom format developed using alongside RoughDraft. This is primarily in response to the problems with industry standardization within this field.

## 4.3.1 Computer Aided Design

Computer Aided Design is employed in most aspects of engineering today. CAD technology is often associated with drafting, but contemporary programs provide designers with the tools to specify much more than shapes and angles. The addition of design properties like materials or manufacturing processes along side dimensional data within a single file has greatly increased the productivity of the design process. In fact, CAD tools have become so integral in society that they have frequently been the motivation behind advances in computer hardware.

CAD file formats traditionally have had low open standard adoption rates, however, and this fragmentation has become a problem plaguing the industry. For this reason, each CAD software suite typically stores data in a proprietary application specific format. Transferring proprietary data between different applications can be frustrating or impossible. In response, a variety of programmatic libraries and software tools to convert different formats have become available, but the conversion process is less than ideal. At times, a waterfall process has to be used in order for data from one application to be readable by another. The problem is so persistent that some consulting firms even specialize in format conversion.

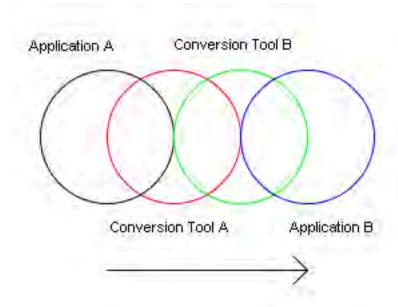


Figure 4.1:
Follow from left to right-->
Application A has the ability to save to a format supported by Conversion Tool A.
Conversion Tool A can convert the input format into a format supported by
Conversion Tool B. Finally, an additional conversion is made to support
Application B.

During the design phase of any product using CAD, the plans will most likely need to pass through several groups or even companies before the final product is declared complete. If all of these entities are not using the same software suite they are likely to run into either conversion problems or error. The problem is more apparent when customers are involved in the design process. Clients of engineering firms are not guaranteed to have any CAD software available on their systems, much less the specific brand used by the developers of their goods. In response, many CAD companies offer viewing applications for their proprietary file formats. Working with these programs however is typically frustrating and unproductive. The solution for many firms is to make exports of

their CAD data into common image file formats, like JPEG or bitmap, for customer consumption. With design iteration, these static representations are likely to become stale quickly and each time the design changes clients may demand a new rendering.

RoughDraft's goal is to solve the interactivity dilemma between all the participants in the CAD design process. To create a file editable and viewable by both clients and designers without conversion or client side application upgrades. By simplifying the experience, we can provide evidence of the practically of file format extension through steganography.

### 4.3.2 Alternatives to CAD

Many non-CAD programs also save data in a non-directly consumable template format, and as needed export this data for general use. If a file meets these two conditions, it is considered a candidate for file format extension through steganography. A good example of a non-CAD program with this behavior is Adobe Photoshop. The native file format for Photoshop is the Photoshop Document or (.psd). This format is not recognized by any standard OS, but can be converted into almost any OS supported image format.

Two additional candidate formats were reviewed to determine if either could provide a better example of the single source methodology presented in RoughDraft. Waveform Audio File Format (.wav) and Musical Instrument Digital Interface (MIDI) tracks successfully satisfy the requirements for candidacy and could make an interesting demo in a future application. The formats are regular

channels for steganography payloads and can be edited in a variety of template-based programs. If a musical application were developed in place of RoughDraft it likely would have been based on TuxGuitar. This program accepts data in a visual guitar tablature format, Guitar Pro (.gp3, .gp4, .gp5) and is capable of exporting that data to an audible MIDI file. It seems possible given the current criteria, it would be possible to package Guitar Pro data file into the MIDI itself.

There are two primary reasons a musical approach was not chosen for this research. The first is simple familiarity. Building on previous CAD experience helped eliminate some of the time and effort associated with learning new standards. The second is based on the perceived presentation value of an image over a sound file. Given the ways in which we intend to present this research, images seemed like a stronger choice.

### 4.4 RoughDraft

RoughDraft was developed using all of the research accumulated while studying file format extension through steganography. Comparing contemporary CAD software applications and grouping together common capabilities determined the features for RoughDraft.

### 4.4.1 Vector Bitmaps

The target file type for RoughDraft has the extension (.bmp) like any normal bitmap file. The steganography extension makes the format more

complex than a standard bitmap file, so the name bitmap does not seem fully appropriate. The resulting product will instead be referred to as a vector bitmap. At first, the name seems to be at cross-purposes because bitmaps are normally associated with raster graphics not vector. However, since the metadata for this standard will be in a vector style while maintaining the raster façade, vector bitmap accurately reflects both aspects of the file format's heritage.

### 4.4.2 TurboCAD

TurboCAD is marketed as a low-cost alternative to AutoCAD, the industry leader in computer design software. It is a mid-range product suitable for use in some professional industries, but not all. TurboCAD was chosen as the competitor for RoughDraft precisely because of its position in the middle of the CAD marketplace. TurboCAD offers most of the features a designer could want in a modern CAD tool, so working towards emulating its functionality allows our alternative to appeal to most of the CAD industry.

### 4.4.3 Usability Features

Most of the features for RoughDraft derive directly from TurboCAD and other commercial CAD software. When using RoughDraft the user should feel as if there is some commonality between its environment and that of other CAD software. RoughDraft is a prototype, so only a 2D version will be produced. This

restriction implies that the general feature set will be composed of basic shapes, layers, and some shape modification functions.

# 4.4.4 Secondary Format Features

One important internal feature for RoughDraft was the secondary data model, which is not specified or proven by any process yet known. The goal was to create a simple model that could create complex interactions between each CAD element. It was paramount that this model work well when designing architectural or mechanical plans. Additionally, any information that could be assumed or derived from the existing data needed to be stricken from the secondary data layer, as carrier data is at a premium with a steganography channel.

Drawing Exchange Format (DXF) is a CAD file format used to export data from one proprietary drawing application to another. The standard was developed by Autodesk and has been used with limited success to create a consistent conduit between programs. The lack of adoption is attributed to the fact that Autodesk did not publish standards for the files until years after the initial release. DXF was the first option considered for RoughDraft's secondary data layer. Creating a DXF from the vector data and then packaging it into the bitmap file using standard steganography tools was the simplest design solution to develop a full scale CAD extension application. The drawback is the space inefficiency when packaging a DXF file. The smallest pure text DXF file successfully created

during testing was nearly 4KB in size. A binary DXF representation is also available, but only provides 25% more space efficiency on average.

A custom binary format was found to be the optimal approach given the restrictive space resources available. Technically, it became far superior to store the limited 2D data set in a non-DXF format. However, DXF greatly influenced the end product, and it was determined that any custom shape should be DXF realizable. Simply put, there should be a straightforward method to convert a custom shape into a DXF representation. RoughDraft was designed, as a viable competitor to standard CAD programs, so it made sense to have a method available to retrofit old designs. Some common companion file types like (.stl) were also considered during the development of the custom format. These files are processed by rapid prototyping machines and produce actual 3D models of a CAD design. While 3D designs are not possible in the current revision of RoughDraft, interoperability is an important consideration for future development.

# 4.5 Perception Research

For this project some additional color perception research was carried out in order to justify the number of least significant bits used in RoughDraft. There has been extensive research on human perceivable changes to bitmaps and other image files. Independent research in this area was performed to discern whether or not the specific type of image produced by RoughDraft required finer restrictions than conventional cover images.

## 4.5.1 Background Information

24-bit bitmap files use 24-bits to describe each pixel in an image. This is equal to three bytes, and each byte represents one of the three base colors that make up each pixel. The base colors for bitmap files are red, green, and blue.

Unsigned bytes can represent a number from 0 to 255; likewise each base color is assigned a number in this range that corresponds to its shade. When varying shades are combined the resulting pixel appears on screen as one of approximately 16 million different colors. The variations between similar colors at this level of depth are so small that they are practically unnoticeable. Most steganography applications use up to three least significant bits per byte on 24-bit bitmaps. This establishes a normal human perceivable range for standard applications in segments of 8 shades per base color per pixel.

### 4.5.2 The Experiment

CAD images typically have less variation than standard cover images like photographs. Designs are normally rendered using a series of geometric shapes with occasional shading. For this reason, there was concern that the images produced by RoughDraft would need to be more conservative about the number of bits used for packaging than a conventional cover image. An experiment was setup up to establish a finer range for human perception when suboptimal blending conditions were present.

The experiment was designed with three different stages. A dozen subjects participated in the survey. They were first administered a test to determine how well they perceived the difference between two shades of the same color. The testing material consisted of a 1024x768 24-bit bitmap split into two equal halves by one of four planes: horizontal, vertical, upper left corner to lower right corner, or upper right corner to lower left corner. Up front each participant knew that two similar colors were present on the image, and that each of these colors covered one half of the visible space. Three sequential tests were run; one for each of the base pixel colors, red, green, and blue. Each test started with one half of image at the highest numeric shade of the color being tested and the other at this shade minus one. Since 24-bit bitmap files were used, the highest numeric shade and its companion were 255 and 254 respectively. If the user could not identify the dividing plane, the less valued half of the image was reduced by another shade. This process was repeated until the dividing plane could be identified. The average difference perceivable using this test was around three shades.

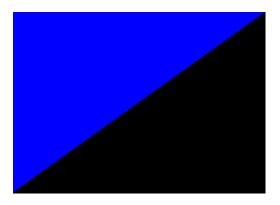




Figure 4.2: Follow from left to right-->

A template image for the shade test (left). By the time a subject could perceivable the dividing plane, the image would look something like the image on the left. Here the image is split into two halves one at blue 255 and the other at blue 240. The shade differences were noticeable long before this point in the full size tests. Here the effect is exaggerated for demonstration purposes.

Using the results from the first stage of the survey, a second round of testing was administered to each subject. The goal was to see if the user could identify an arbitrary shape using the shade difference they perceived in the first stage. Three mazes were created for this stage, one for each base pixel color. The background of each image was the highest numeric color code for the base color being tested and within each image an arbitrary maze like shape was drafted using the lowest perceived shade difference from the first test. The participants were asked to travel the maze from start to finish under observation. Each participant eventually ran all of the mazes successfully, though none did so without some difficulty. This reinforced the results from the first test proving that the difference was difficult for each subject to perceive.

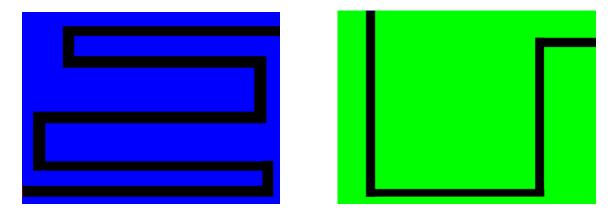


Figure 4.3: Blue and Green templates used during Stage 2 testing.

The red maze used during this stage had a special modification to further verify the difficultly in perceiving the minor shade difference. A dead end was placed in the maze and surprisingly none of the participants discovered it. The modification seemed necessary in order to add a truly arbitrary element to the test. The test required so much of each subject's concentration that they missed something that seems obvious under less difficult circumstances.

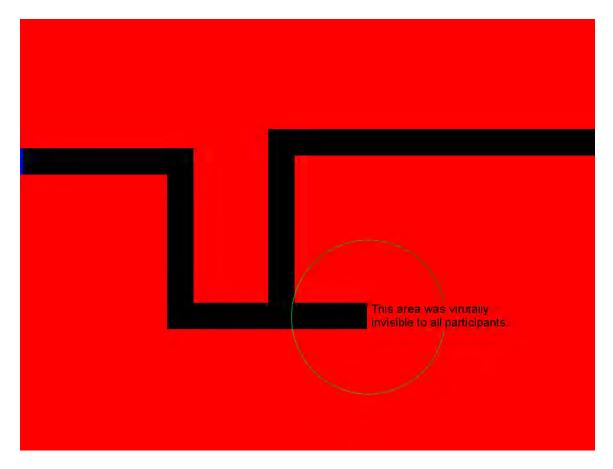


Figure 4.4: The dead end area circled in this example is difficult to detect when the pattern is colored near the perception gap.

A third stage of testing was administered after the completion of RoughDraft. Each study participant was asked if they could identify the packaged bitmap produced by RoughDraft when placed beside a normal unmodified bitmap of the same image. Using the examples from *Results* chapter, no subject was able to successfully identify the image modified by RoughDraft.

#### 4.5.3 Results

Based on the testing results, RoughDraft should be able to use two bits per byte as carrier data in each image. Three shades looks to be the definitive perception boundary for this file format. This is slightly less than some contemporary steganography programs, but the results from our survey seem clear. A discerning user could still identify some irregularities within a file packaged with this density, but given a natural inclination to finish a perceived image it is unlikely. Further restrictions and final technical details of secondary file format for RoughDraft will be discussed in the next chapter.

### 4.6 Industry Niche

RoughDraft and vector bitmaps are meant to appeal to a specific audience. This niche has been defined as best possible and a summary of the state of the current technology is given.

### 4.6.1 Contributions

We have given developers the tools needed to start solving the file-sharing problem. This research provides a clear path to integrating high-level design data into a standardized file format while maintaining backwards compatibility.

Leveraging contemporary digital steganography research, we created an outlet for adding additional hidden information into a standard image file.

This feature is highlighted in the demonstration program, RoughDraft. The vector bitmap files produced by the application provide clients and developers with a single point of contact when referencing a design. Clients normally view CAD data by way of either a secondary viewer or file type. Single sourcing prevents errors and additional maintenance by keeping both parties up to date on design changes. Vector bitmaps should also appeal to anyone interested in making their CAD content more accessible. The files are completely backwards compatible with normal bitmaps so they can be embedded in web pages and viewed on systems where no CAD viewing applications are available.

Most organizations try to standardize on one file type for consistency and repeatability though many CAD applications support multiple formats. Vector bitmaps will likely appeal to groups without strict format requirements and those new to the CAD industry looking to standardize, because they offer the single source feature not available in any other format.

RoughDraft is currently the only application capable of producing vector bitmaps, though it should be possible to integrate the format in other applications. The prototype is not available for direct consumption and only supports 2D CAD drawings. Full source is available in the appendix. File format extension through steganography has shown potential usefulness in other use cases and industries though only a few have been explored.

### **CHAPTER 5. APPLICATION**

### Overview

RoughDraft is a basic CAD application that uses packaged bitmap files to store vector graphic data. This chapter describes how our contribution improves the implementation of RoughDraft.

## 5.1 Initial Design Requirements

In order to prove the practicality of file format extension through steganography, it was necessary to build an effective demonstration application. After performing extensive industry research a CAD application was determined to be the most suitable model to demonstrate our breakthrough. From this initial concept a stricter list of requirements for RoughDraft was created.

# **5.1.1 User Interface Requirements**

The user interface for RoughDraft needed to emulate other CAD applications as closely as possible. The objective for the application as a whole was to show the usefulness of the backend data storage mechanism, so any UI element that strayed from the layout or general functionality of a conventional

CAD program was deemed unacceptable. These elements would be distracting and possibly provoke a negative attitude about the industry adoption of the vector bitmap standard. For RoughDraft, TurboCAD was used as the analogue for the industry's standard mouse-based user interface. The 2D drafting interface for a contemporary drafting application is composed of two primary components, the drawing window and the toolbar.

The drawing window is a large undefined space used by the user to build their design. Background color, zoom, and optional grid lines are just a few of the many settings associated with the drawing window. These settings constrain the drafting environment in order to increase productivity. Some of these properties are saved with the application as global preferences for the user, while others are stored with the drawing. For RoughDraft, only features saved independently with each drawing were considered for replication. All runtime or general preference based features were not implemented due to the lack of impact on the vector bitmap format. As a result, background color was the only configurable drawing window setting considered during development. Other settings, like window size, would be determined by using attributes available in the standard bitmap format. The rest of the drawing window features were statically set to values that were deemed appropriate for the application.

The toolbar element of the user interface is a simple column of buttons floating to the left of the drawing window by default. Its purpose is to provide easy access to the basic shapes and editing tools a user would like to use in order to

build their design. In TurboCAD, there are three primary shapes selectable via the toolbar: a line, an arc, and a rectangle. Replicating the behavior of these buttons as closely as possible was an immediate UI design requirement. The cursor option is another useful feature from the toolbar that was adopted into RoughDraft. Other tools like text and trim were left out of the prototype's UI design. While there are workarounds available for some of the more advanced editing use cases, it was determined that macros for these would not be available through the UI. More on this topic will be discussed later. If the product goes through a second revision, these features should be reconsidered.

A separate layer toolbar is also featured in TurboCAD. Layers were designed into RoughDraft, but instead of making a separate specific toolbar, layer selection was incorporated into the existing toolbar. Most desktop software, including contemporary CAD applications, gives a user the ability to float palettes independently. This feature does not add anything to the proof of practicality for vector bitmaps and was dismissed.

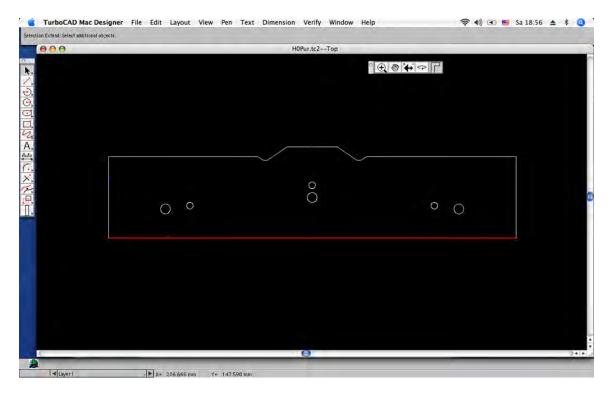


Figure 5.1: A screen shot of the TurboCAD 2D user interface for Mac OSX. The drawing window occupies most of the screen space while the toolbar docks itself to the left by default.

One feature not available in TurboCAD, but available in other popular drafting suites is a command line interface. Given the restrictive graphical user interface being implemented for RoughDraft, a command line option was thought to be a good alternative. When using a GUI, it is often difficult to create exacting design without marked grid lines and grid snapping, which auto corrects the cursor to point to the nearest grid point. These features were designed out of RoughDraft due to their difficulty to develop, lack of importance to the background mechanism, and time constraints. That being the case, a pure command based interface was not desirable either. For example, selecting created objects in the drawing space is a task that is much easier for the user to

perform with a GUI. Additionally, GUIs are helpful for beginners who are not familiar with a product and its tools. RoughDraft needed to include a useable GUI to address these concerns, so it was designed with the most important graphical features. In RoughDraft, expert features are accessible through a command line interface and high-level tasks can be performed through GUI.

## **5.1.2 Drafting Feature Requirements**

The focus for the prototype application was to include everything needed for a reasonable 2D design tool. In the design, two areas were highlighted as the must have drafting features for the application: a set of primary shapes and layering.

Four primary shapes were integrated for use in RoughDraft. The shape tools allow users to create lines, rectangles, circles, and swatches. Three of the shapes are common, but the swatch is slightly unique. Swatches were a concept developed during the development of RoughDraft to handle specific use cases and workarounds. The swatch shape is simply a rectangle that is filled with a solid color instead of taking on the attributes of the background behind it. Using a combination of these basic shapes it is possible to create complex 2D designs.

Layering is a feature available in all modern CAD applications. The feature gives a user the ability to associate different shapes into organized groups based on color. Most contemporary programs have a static color palette as well as an outlet to define custom layer colors. Adding layers to RoughDraft had some

impact on the vector bitmap format, so it needed to be implemented in the prototype. The custom color feature did not seem to have impact on the data storage, so it was not selected as a priority for this development cycle. Five colors are supported as layers in the prototype; black, red, green, blue, and white.

## 5.1.3 Steganography Feature Requirements

RoughDraft does not incorporate any advanced or complicated steganography techniques. An alternate approach was preferred given that the core requirement for the program was to prove the practicality of packaging CAD data into bitmap file. CAD data from RoughDraft is encoded using simple, linear LSB, and the files produced would likely be flagged as candidates if screened by a steganalysis application. The final packaging has no impact on the practicality of the vector bitmap format, so no time was spent developing a more complicated implementation. By creating a few sample designs it was possible to determine how much carrier data were required from an average CAD design. An alternative method was devised to determine whether or not different techniques could be realized without actually using the specific technique in question. An encoding density calculator was added to the project to accomplish this. If RoughDraft is deployed, the program can be adapted at that time to best fit the needs of the user.

### **5.1.4 Secondary Format Requirements**

The secondary format was fashioned to be small in order to minimize the use of the scarce carrier data resource. The application was designed to programmatically generate as much data as possible from this basic format in order to conserve carrier data resources. Every shape used in Rough Draft is derived from uniformly sized blocks of data. The data block consists of six unique fields. The first field in this block identifies which base shape the data block represents. The next field is used to identify the shape's containing layer.

Positioning data fills the remainder of the block; it includes starting X position, starting Y position, ending X position, and ending Y position.

The formatted data blocks lay end to end in an uncompressed, linear fashion. As a contingency, should the need arise, run length encoding or some other compression method can be used to save additional space when packaging the secondary data.

# **5.1.5 Application Feature Requirements**

Windowed file-handing applications often feature a File menu, which provides easy access to the most common file handling functions. Five file handling functions were designed into RoughDraft. The File menu contains shortcuts for creating new images, opening previously saved images, packaging the shape data, saving to the current file, and saving the current file under a different name. In some applications, the New operation allows a user to specify

settings and constraints like image size to the new file. This feature is not included in the design, because it does not affect the secondary format. An Exit button was also placed along side the file operations for ease of use. Even though Exit does not perform any file operations, it is widely acceptable to place a shortcut to terminate a program within the File menu. RoughDraft's File menu options are also accessible through shortcut keys; this is customary with windowed applications.

Most of these functions are standard on windowed programs, but the packaging shortcut is unique to RoughDraft. This button is used to modify the image data, transforming the formatted data from a normal bitmap to a vector bitmap. This function was not designed into the save features so that RoughDraft could operate as a conventional bitmap editor as well as a vector bitmap editor. The dual nature of the program allows for annotations or other non-technical drafting to be placed alongside the data critical to a CAD drawing.

# **5.1.6 Future Use Feature Requirements**

The design for the prototype includes only the basic drafting features required for CAD development. Since the design has not yet been deployed or tested by individuals outside the project, internal extensibility features were included in RoughDraft in order to ease the development of features missing in the initial design.

In future versions of the application, new shapes may be required in order to help realize some designs. The secondary format was created with enough space to accommodate several new base shapes, should the need arise.

Similarly, the format is prepared to accept new layers if and when they are desired.

If the space reserved for future use in the secondary format becomes inadequate, it may limit the creation of new features within future revisions of the program. At this point in time, an entirely new secondary format may be required. Versioning information integrated into the secondary format allows applications to easily identify which format they are working with and eases the development burden associated with maintaining backwards compatibility should the data layout need to change radically.

The evolution of RoughDraft could lead to its usage in 3D design. While no features were designed into the prototype explicitly to support 3D shapes, certain aspects of this expansion were considered. The 3D contingency plan for RoughDraft is based on the programmatic extrusion of the basic shapes. Each primary 2D shape will contain an additional distance parameter in order to express its presence in the third dimension. This common technique is used in conventional drafting programs in order to allow drafters to easily expand their designs from 2D to 3D.

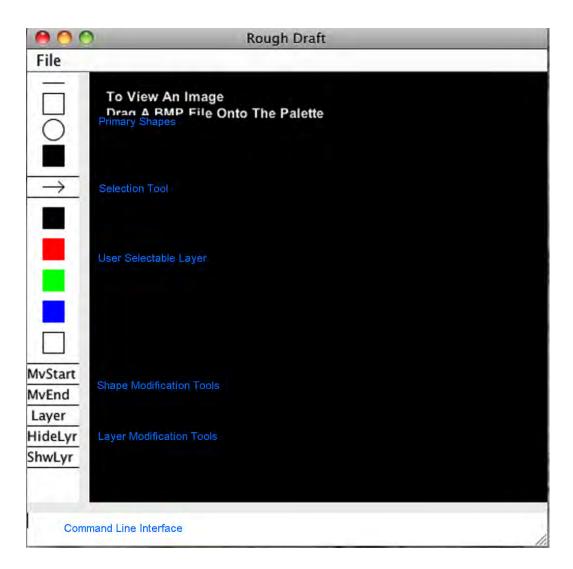


Figure 5.2: A screenshot of the RoughDraft user interface.

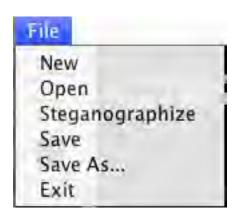


Figure 5.3: A screenshot of RoughDraft's file menu.

## 5.2 Software Layers

RoughDraft is implemented using three different software products. The first piece of software created for the project is a bitmap editor known as bmpmanip. On top of this layer a thin proxy library, bmpmanipProxy, allows communication between the image editing library and the top level editing application, RoughDraft. This layout proved advantageous during initial development and also provided a path to extend the number of file formats supported by RoughDraft in the future.

# 5.2.1 bmpmanip

In order to decouple the image editor from the bitmap format, all of the image manipulation logic is contained in a standalone library bmpmanip. This portion of the code is implemented in C++. This aspect of the project helped determine what generic image information is vital in the creation of vector bitmaps and saved development effort by clearly splitting some of the functionalities.

By creating an interface, fixing bugs in the image manipulation code becomes less iterative on the entire code base. Changes to this area of the code would not require rebuilds of the higher software layers, so long as the same interface remained exposed. The library works as a standalone tool and has been successfully complied into a BSD dynamic library, a Linux shared object,

and a Windows dynamic link library. Developing the library from scratch helped establish a good working knowledge of the 24-bit bitmap format.

Using an independent image library highlights the functions future image drivers would need to expose in order for RoughDraft to support new formats. Currently the implementation is bitmap specific, but in forth coming iterations of the library the interface could be made generic to support a variety of image types.

# **5.2.2** bmpmanipProxy

The GUI for RoughDraft is implemented in Java. The caveat to writing this section of code in Java was that it created a language barrier between RoughDraft's UI and the low-level bmpmanip library. The image library was developed well in advance of any of the other pieces of RoughDraft using C++, and reimplementing this code in Java would have wasted a considerable amount of time. The goal was to use bmpmanip and its API as is, because it was performing well and rewriting the bit-banging sections of the code in Java was undesirable. In order to achieve this, the Java Native Interface was used to glue the two software layers together. JNI is a special feature of the Java programming language that allows code running in a Java Virtual machine to interact with code written in other languages. The resulting layer was called bmpmanipProxy.

In OS X, the native development environment for RoughDraft, bmpmanipProxy is an actual file installed on the user's system. For other OSes,

the functions can be exported from the bmpmanip shared library. In total, bmpmanipProxy only exposed about eight of the twenty-one functions from bmpmanip. In future iterations of RoughDraft, these functions will be the first targets for generalization in order to expand the supported file formats for RoughDraft.

# 5.2.3 RoughDraft (UI Layer)

The feature requirements for RoughDraft required the implementation of a significant GUI feature. GUI tools and APIs like Swing and AWT, available in the Java Foundation Classes, met the application's design requirements and provided the best developer experience for the task. One factor contributing to the positive developer experience is that Java is supported as a native API on OS X. Code created in Java is also easily ported to other operating systems.

The graphics for RoughDraft were created primarily using JFrames, because they provide an easy way to manipulate a window. Borders, titles, and buttons were added to RoughDraft's JFrames in order to facilitate user interaction. A BufferedImage object was used to represent the drawing window for RoughDraft. This class provides an excellent soft copy of the raster pixel data contained within the bitmap files. Pixel data in this object are accessible by the setRGB() and getRGB() functions and the dimensions of the image and can easily be initialized by calls into bmpmanip. The Graphics2D class provided all of the code necessary to draw RoughDraft's basic shapes: line, rectangle, circle,

and swatch. Graphics2D works with the BufferedImage object in order to derive the geometric equivalent of a vector shape in raster data. Using these image tools saves time by avoiding the unnecessary development of tedious geometry code.

In order to handle user input, components within RoughDraft implement some common UI interfaces. The KeyListener and MouseListener classes do the backend observation of the user input by default and allow the developer to fill in the reaction code. RoughDraft also extends the TransferHandler class within the Swing toolkit. This allows the application to support drag and drop as a method of opening existing image files.

Finding a comparable, portable library in C++ for the GUI may be an option for future versions of RoughDraft, but it is not likely that the application will ever be purely developed in one language. High-level language features like garbage collection and sandboxed code are valuable when developing this type of desktop application and they are readily available in Java.

# 5.2.4 RoughDraft (Steganography Layer)

The RoughDraft prototype was developed using the iterative software development model. The application was built in four primary iterations: bmpmanip, the prototype Java bitmap viewer RoughCut, a bitmap editor based on RoughCut, and finally RoughDraft. RoughDraft contains elements from all of the previous works and added the vector bitmap feature.

The user initiates the steganography process for each vector bitmap through the "Steganographize" button in the UI. This calls an internal function called computeElementCodes(void). This initiates a chain reaction that first encodes the global control data for the secondary format. Once these data are in place, a data structure containing all of the shapes for the design is iterated over, and the data corresponding to each shape is then encoded linearly into the next available set of pixels. Finally, the in-memory representation of the modified image data is reloaded into the drawing window.

The prototype currently handles all of the steganography logic within the Java layer of the program alongside the UI. This is by no means a long-term solution. In future revisions of the program, an interface with the functions necessary to perform this task will be created and will reside in area completely separate from the UI. This code would ideally go into the existing low-level C++ layer or an additional low-level component where the many bitwise operations could be better encapsulated. The current placement of the code is largely related to the undefined elements of the application at the time of development. A steady steganography algorithm is ready to be ported, but that was outside of the scope of the prototype. While this code remains tied up with the UI layer for now, all of the logic for the process has been encapsulated into a subclass called stegoUtils. This should help to pinpoint the areas of code that need to be moved and lessen the burden of porting the steganography code to another layer in the application.

### 5.3 Technical Details

All of the intricate details of RoughDraft's internal structure are not necessary for the assertion of the practicality of the vector bitmap format presented here. Therefore, this section describes the small critical sections of code that demonstrate the usefulness of the underlying technology.

## 5.3.1 Shape Data Format (SDF)

The secondary format produced for the prototype is based on an extensive review of information that meets the current needs of the application. This format, known as the Shape Data Format (SDF), is a string of data representing the different shapes in the design.

There are two primary components to the SDF. The first is a short header describing the SDF version the data are formatted in and the number of shapes following it. The secondary format used in RoughDraft is designated with the letter "S". In the future, as the program changes, this field can be updated to denote a new SDF format. The number of shape elements immediately follows this version byte. This parameter is also a byte long and can represent any number from 0-255. This is artificial inhibitor to the size of the data format that can be resolved if need be. Both of these numbers are encoded into the first pixel of the image. The third byte of this pixel is reserved for future use in order to increase the maximum value of the number of shapes parameter.

The second component is an indeterminate number of shape objects. Each object is placed in a linear fashion next to one another in a contiguous line. Individual shape elements are seventy-two bits long. The initial four bits contain the base shape code number used in RoughDraft. Currently only four of the sixteen codes are used: line (0x0), square (0x1), circle (0x2), and swatch (0x3). The next four bits describe the layer in which the shape resides. Like the shape codes only a fourth of the allotted layers are currently valid: Black (0x0), Red (0x1), Green (0x2), and Blue (0x3). The last 64 bits of the preliminary SDF is divided into four equal parts; each represents a component of a 2D coordinate on the drawing. The first two 16 bit numbers represent the starting X-axis position and the starting Y-axis position of the shape respectively. Similarly, the last two numbers represent the ending position in corresponding order.

This simple data format provides enough data to form all of the shapes desired for RoughDraft in the near future. The prototype version of SDF is not overly conservative in its footprint and remains uncompressed. If problems do arise in the usage of the format, trimming the format size or compressing the data are immediate options to consider optimizing the procedure.

## 5.3.2 Math In the Shape Calculator

This project needs to provide expectations for other steganography processes outside of Least Significant Bit. An external C++ application called shapeUtils was built to generate the calculated encoding density of various

steganography processes. The program uses the bmpmanip library to extract the file size and other vital information from a carrier file. Using this information, shapeUtils derives the number of shapes possible given the current file and the steganography process desired.

The simplest steganography calculations shapeUtils generates are based on the size of the channel. The most basic static calculation performed in shapeUtilis represents the maximum number of shapes possible when packaging one bit of SDF data per pixel in a program like RoughDraft. In RoughDraft the least significant bit of each byte is used to embed data. Raw data size (Total file size minus header information) is available to shapeUtils through bmpmanip. This number is then divided by seventy-two (SDF format size) to produce the final result. Other static calculations are also included in shapeUtils. For instance, the maximum number of shapes possible using a safe rendition of RoughDraft's linear LSB can be produced from the area of the image. RoughDraft blindly writes to both image data and padding bytes. As an improvement to the design, RoughDraft could be restricted to write SDF data only to image data bytes. The proposed change would prevent stegoanalysis tools testing for exacting padding data from detecting vector bitmap files. To generate this value, RoughDraft first calculates the area of the image in pixels from the width and height variables provided by bmpmanip. Multiplying the calculated area by three produces the total number of pixel bytes within the image. Dividing the total number of pixel bytes by the SDF format size produces the number of shapes possible using the

safer form of RoughDraft's linear LSB. Rough statistical profile restoration numbers can also be produced statically by shapeUtils. To do this, the program simply divides any previous static method's results by two. Further details on statistical profile restoration can be found in Chapter 3.

The shapeUtils program can also perform more complex calculations on a file. Using the pixel data, also available through bmpmanip, it is possible to generate useful numbers for estimating the maximum number of shapes possible using logistical profile restoration. The tool assumes that the image provided conforms to the color restrictions used in RoughDraft. The shapeUtils application first searches the image for areas matching the layers used in RoughDraft. For each pixel that corresponds to a supported layer, it is assumed that one bit of information may be stored.

### **CHAPTER 6. RESULTS**

### Overview

The initial testing on RoughDraft has generated a wealth of data on the usability and practically of the application. This chapter documents how this information was aggregated, interpreted, and translated into suggestions and improvements in future designs.

### 6.1 Usability

As with any software endeavor, a number of surprises revealed themselves as RoughDraft matured into a full-fledged application. Each of these challenges was met with a solution, but due to certain constraints not all of the solutions were ideal. The tradeoffs made to accommodate these previously unforeseen adjustments to the core code were generally minor and should not inhibit further research. Due to these limitations however, the RoughDraft prototype is obviously not an ideal application for developing a production level CAD design. Most of its limitations are related to UI bugs or small technical glitches that could be resolved given a longer development period. The usability

problems discussed below are related to specific practicality limitations or technological concerns that could compromise the core goals of the project.

## 6.1.1 Format Compatibility

The control data for SDF are currently stored in the bottom left pixel of every vector bitmap. This goes against the original design of the application and changes to the visible image content. A late breaking design change forced the developers to occupy this pixel with raw data. Not implementing this change would have resulted in the loss of backwards compatibility for some operating systems.

The 24-bit bitmap standard defines an unused four-byte space within the header. In some documentation this area is described as "Application Specific Space." This space was chosen in the design phase for RoughDraft as the home for all of SDF's control data, because this space is not editable in a standard bitmap editor. The control information is critical when reproducing an SDF design, so it was determined that this data should be placed out of harm's way. Further reasoning behind this decision is discussed in Chapter 3 Section 3.4.2. It became immediately clear after implementing this solution that this area would not provide a satisfactory location for the control data due to OS compatibility issues. Placing arbitrary values into the Application Specific Space works on both Windows and Ubuntu (Linux test system OS), but not on OS X. OS X enforces a rule that requires all four bytes of this space to be written as zeroes. If the image

data do not comply with this rule the file is unrecognizable as an image by the operating system's default viewing application. The simplest solution was to move the control data to the first mutable space and continue building the rest of the application. The visible impact was small enough to justify the time saved by not completely redesigning for the prototype, so we continued development in this manner. The ultimate fix for the problem is to use steganography to incorporate the control data into the image in the same way as the rest of the SDF data. This will place further restrictions on the encoding density of the file, but causes no visible impact to the image. It should be noted that no subject noticed the obvious bottom left pixel difference between the vector and standard bitmap versions of a design in testing. The issue did however raise immediate concerns about the possibility of a similar problem involving the padding bytes used in bitmap files. These bytes are used to keep the data representing each horizontal line in the image in multiples of four bytes. Normally these bytes are written as "00", but to date there have been no compatibility problems when writing arbitrary values to the padding space of the bitmap file across platforms. RoughDraft adds SDF data to these bytes as if they represent actual pixels, so padding zero enforcement could become an issue if the application is ported to new platforms.

The file header for a bitmap contains all of the size and depth information for the image including width and height. Unsigned numbers are normally used to represent dimensions, but uncompressed bitmap files are designed to support

signed height values. This was unknown during initial development and therefore led to some code duplication and complication in the prototype. In a normal bitmap file, the first three bytes of pixel data represent the bottom left pixel of the image. The data are translated from this point as each pixel from left to right until a byte offset of three times the width has been read. This creates the bottom row of the image. Each row of the image is built from the bottom up, skipping any padding bytes, in the same fashion until the top row is complete. This behavior is associated with an image that uses a positive header value for height. If a negative value is assigned for the image's height, the starting pixel is in the upper left and the rows are built from left to right and top to bottom in a similar manner. Consistency for the default sign of a bitmap's height when creating a new image is not guaranteed across different OSes. On Windows and Linux new images are created using the positive height scheme, while on OS X negative image heights are used as the default. All systems tested support viewing either format equally well. RoughDraft images use the positive height data representation by default, though negative height images are also supported. The SDF data for a negative height image will be moved from the bottom of the image to the upper portion after importing a negative height image into RoughDraft, but there are no other functional differences. In order to streamline the code, in the future, all images will most likely be converted to one style as both are supported equally well. The decision as to which height format would be chosen greatly depends upon whether or not there is a clear victor in terms of capability or functionality.

## 6.1.2 Dual Mode Editing

RoughDraft allows certain parts of an image to be editable in a conventional editor without disturbing the SDF data. By making the image not just viewable, but also editable, clients without RoughDraft installed on their machine are able to easily markup and comment on a design in the developer's source file using standard OS image editing tools. Additionally, this feature allows developers to use existing bitmaps as backgrounds or accents in their designs.

While commenting and accenting is possible in RoughDraft, advanced knowledge of the vector bitmap format is required in order to prevent data corruption. In order to fully support this, a zoning feature needs to be added to RoughDraft. This feature would help a designer designate safe areas for the client to add their content. One possible solution to the problem involves the creation of a new toolbar option called a raster block. Unlike other options in the standard toolbar, no information related to raster blocks would be stored in the SDF. Raster blocks would look similar to the rectangle from the standard toolbar when drawn on screen. They would provide a visual indication that the area contained within them is safe to edit. An underlying zoning tool would find SDF data in the image and prevent the designer from placing raster blocks over those areas. The clients would then work under a social contract with the designers to comment or accent only within raster blocks in order to prevent data corruption.

The use of background images is also possible, though this feature is severely limited. A design that incorporates a background image is only fully

editable in a single session of RoughDraft. After the file is saved, the original image data that existed beneath any new shapes is lost forever. The symptom of this behavior is that deleting shapes in subsequent editing sessions results in the SDF data for that shape being removed, but not the raster data. The raster data could be removed as well, but white space or some other default would need to be substituted in its place.

## 6.1.3 Layering

Layering is a hard requirement when drafting a design of any reasonable scale. RoughDraft provides designers with basic tools for layering and the unique ability to share only part of their design in a bitmap compatible format. By turning off a layer the developer can remove all of the visible raster data from the image while maintaining the underlying SDF representation. It is possible to recover the visual representation of the hidden data at any time using this feature.

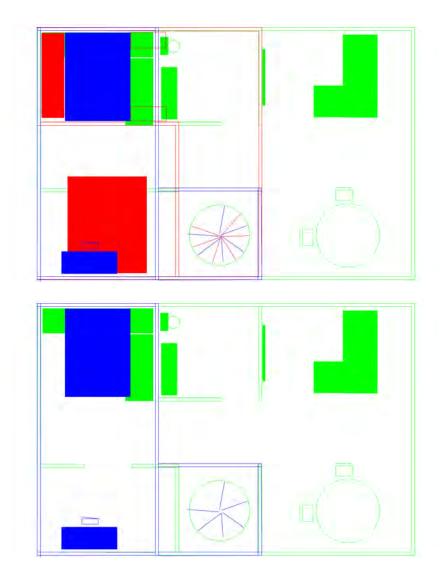


Figure 6.1:
Follow from top to bottom-->
The image on the top shows a complete CAD illustration. The image below contains the same shape data, but hides the visible aspects of the red layer.

## 6.1.4 Workarounds

Subtle compatibility and functional issues affecting the design could be resolved with a more comprehensive study of the bitmap format and its common usage. These issues did not affect the underlying abilities of the prototype

substantially. In order to satisfy the primary use cases for the project, a few workarounds were used instead of tackling these problems directly.

In CAD files, title blocks are often used to display important information related to the design. This can include, but is not limited to, the drawing name, creation date, design revision, or a company logo. Title blocks are typically placed on the bottom or sides of a drawing. The default location of the control data for vector bitmap files was strategically placed in the most likely path of a title block in order to better mask the dark control pixel created during the steganography process. Even though the control pixel was not detectable during testing, the addition of a title block makes this implementation detail practically seamless.

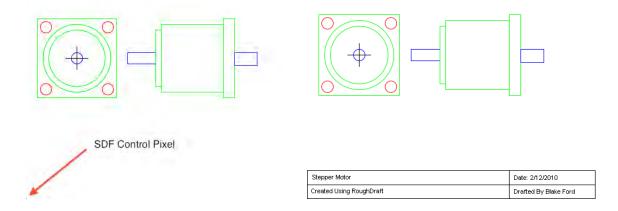


Figure 6.2: Follow from left to right-->

The image on the left shows a part drafted in RoughDraft without a title block. Without a title block, the SDF control pixel is visible in the lower left hand corner of the image, On the right, a title block is used to mask the control pixel.

Comments, accents, and background images can be added if a developer is very careful. For most of the small designs tested, the SDF data fit completely within a very small portion of the bottom of the image. In the figure above, the text was carefully placed within the title block after the drawing was completed. Because the lettering was strategically placed, the CAD data in the image were not corrupted. As a general rule, if a title block is already being used, the space above the block should be safe to conventionally edit. In Figure 6.2, the title block is fifty pixels high and five hundred pixels across. Given the current RoughDraft encoding scheme, over one thousand shapes would have to be created before the title block is overrun by SDF data. If the title block standard is not a detailed enough metric, a calculation could be added to the shapeUtils in order to return a guaranteed safe line for raster editing given the encoding detail. When adding raster data to a vector bitmap the user should also avoid placing any object directly under or over any shapes in the image. Developers that need this functionality should try to reserve some areas of the image specifically for this data when using the prototype. The area close to the top of the image is the least likely to be used by RoughDraft during the encoding process, so it is the best place to consider adding secondary graphics.

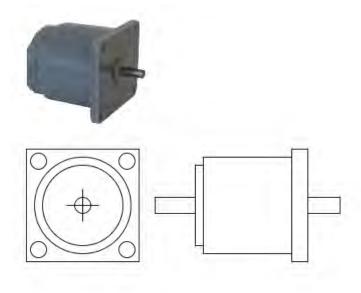


Figure 6.3:
The CAD data in the image above was created using RoughDraft. The secondary 3D image was added later in by a piece of conventional bitmap editing software. Placing the image at the top of the file avoided a collision with the embedded SDF data.

Using only native RoughDraft shapes, while developing designs, provides the optimal editing experience. This gives the program the ability to redraw the entire design from scratch disregarding any explicit pixel data. Users are encouraged to use as many native shapes as possible in their designs. A feature was also added to RoughDraft to reinforce this principle. Whenever a user creates a new drawing, a white background swatch is automatically added to the SDF. This gives the user the cleanest experience possible when creating a new design. This swatch can be resized or deleted just like any standard shape,

should the user wish to add raster images to sections of their file. Matching the auto swatch to the full image size guarantees that any accessory raster data will be deleted during the next editing session.

### 6.2 Practicality

The proof of concept vector bitmap file extension was designed to show that file format extension through steganography has a legitimate place in the CAD industry. Leveraging our background research, some hard numbers on the performance of the vector bitmap backend were produced.

### 6.2.1 General

In general there are relatively few practical problems with deploying vector bitmaps. One minor issue is the file extension. Reusing bmp when vector bitmaps imply something different seemed like unnecessarily confusing nomenclature. The extension vbmp was used in some experimentation with high levels of success so long as the user associated the extension with the viewer/editor of their choice. Association is a manual process, so there is a slight practicality problem from a user prospective.

The second general issue is much larger and in certain circles may doom the format. Uncompressed bitmaps are large in comparison to almost any other image format and definitely when compared to any standard CAD file type. The

stepper motor example above (Figure 6.3) was rendered in nine distinct file formats. The following table shows a comparison of format to size:

Table 6.1.
File Format to Size (KB):

File Format to Size (KB):	
Vector Bitmap (vbmp)	754
Portable Document Format (pdf)	213
AutoCAD 2007 (dxf)	33
AutoCAD 2007 (dwg)	25
TurboCAD file format (tc2)	20
Portable Network Graphics (png)	16
AutoCAD R12 (dxf)	12
AutoCAD R12 (dwg)	8
AutoCAD R12	4
(dxf, manually reduced interpretation)	

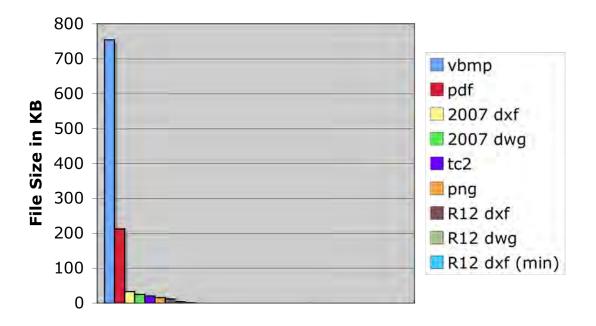


Figure 6.4 Graph representation of Table 6.1.

The R12 variants of the dxf and dwg file formats are from an older version of AutoCAD, Release 12. The format originated around 1992, but can still be used in modern CAD systems. The R12 files are smaller, but they also support fewer features. Because the dxf file is formatted in ASCII text, the generated file is capable of being trimmed by hand to remove boilerplate information. A dxf file modified in this manner is represented on the graph as R12 dxf (min).

If file size does present a problem for users, there are a few solutions. The easiest would be to compress the file whenever possible. This may include archiving or emailing the image. When compressed with the Archive Utility in Mac OSX, the file is only around 12KB in size. All of the OSes tested were packaged with compression utilities, so adding these tools as a common dependency

should not introduce any further problems. Users could also reduce the dimensions of their images. Each pixel of white space in a 24-bit bitmap occupies three bytes of space. Reducing the size of an image to contain only important elements of the design could dramatically reduce its disk footprint. Alternatively, porting the technology to lower quality bitmaps would also provide space savings. Using 256 or 16 color bitmaps would greatly reduce the file size, though it would also reduce the encoding potential.

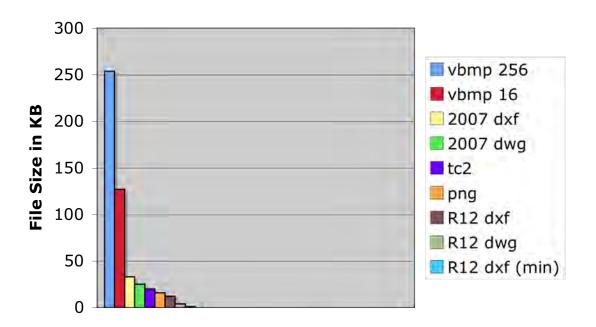


Figure 6.5
Graph in Figure 6.4 is updated to include alternative bitmap formats.

### 6.2.2 Mechanical

Machines, tools, parts, and other mechanical design tasks require specific types of functionality other forms of drafting do not. While RoughDraft does

support the tools necessary to create a large subset of all 2D mechanical designs, using the program may present additional limitations to mechanical drafters.

When drafting a machine subtle features and tolerances can be critical to the design. For other industries, only the basic layout of an object may be required. In finished mechanical goods, for example, almost nothing has hard straight edges and the design for these products needs to include this detail. Most of these edges are beveled or rounded (chamfered or filleted, in technical terms). Creating complex edges is possible in RoughDraft, but the procedure is more difficult than in a conventional CAD application. Other CAD applications provide tools that automatically create these objects when a hard edge is selected. In order to create fillets using RoughDraft, several basic shapes need to be drawn together to produce the desired effect. An additional line must be drawn for each soft edge for chamfers. RoughDraft could be adapted to include macros that place the shapes needed to build chamfers and fillets for a given edge automatically thus adding this particular feature. The caveat here is that using a macro to place shapes would rapidly increase the number of shapes needed to represent the design. Complex mechanical designs approach the shape limit faster than other designs, and therefore, the use of these macros may compromise the usefulness of the format.



Figure 6.6:

The oval shape in the figure above can be created inside of RoughDraft. In order to build this shape, five basic shapes are required: two circles, two lines, and a single background colored swatch. This could present a storage problem is the design contains large numbers of complex shapes.

### 6.2.3 Architecture

In contrast, architectural design utilizes simpler shapes and hard edges.

For this reason, RoughDraft may be more beneficial when developing buildings and structures rather than mechanical parts.

A typical building is defined using primarily rectangles because builders use standard construction materials and techniques that conform to this shape. Less detail work helps avoid the CAD complexities associated with mechanical design. Rectangles are a native shape in SDF and therefore, RoughDraft lends itself well to architectural design. Using a macro to draw four SDF lines was an early design consideration because this polygonal structure is a normal behavior in a standard CAD editor. By making the rectangle a native shape, each rectangle only occupies 25% of the SDF space that a non-native rectangle would.

Construction standardization also reduces the amount the work needed to transition RoughDraft into 3D for architectural designs. Building codes regulate

most of the design characteristics of a given building, so the amount of custom design work used in most architectural projects is small. Given the layout of the building and the code to be enforced, it is possible to generate most of the detailed design features of a structure and render the building in 3D. Automated building tools like these are available in conventional architectural drafting programs like Chief Architect.

There are still a few challenges associated with using RoughDraft in architectural drawing. The largest problem currently is the fixed 1700x800 pixel total size for images. Buildings represent some of the largest CAD designs, and in order to accommodate certain designs scaling tools and a zoom feature need to be introduced into the program. Drafting the electrical components of a building may also be a problem. These components are typically represented by symbols composed of several small shapes with varying areas of shading. This level of detail will result in the same problems associated with detailed mechanical drawings.



Figure 6.7: Examples of common electrical symbols used in architectural drawings.

## **6.2.4 Secondary Format**

The secondary format was tested using a variety of designs and plans pulled from source materials in industry. The largest of the test plans was for a

residential home shown in *Figure 6. 1*. The design models a three story, 24'x16' residence. Some furniture blocks were also included in the drawing. Ninety-two shapes were used to create the image, which implied that any pixel data offset 6678 bytes past the header information remains conventionally editable.

The drawing is dimensioned at the largest size available for the prototype, 1700x800 pixels. The file size for this image is around 4.1 megabytes. Given that 4.1 megabytes is the maximum file size for files created in RoughDraft, 250 or more designs fit into a 1 gigabyte space. Using the specifications from the machine used to develop RoughDraft, this means that one workstation could hold up to 20,000 large designs. Logistical profile restoration is the strictest steganography technique studied during the course of this project, and conservative estimates show that around 3200 shapes should be possible to embed into the file. Using linear placement, the least restrictive form of steganography tested, this estimate jumps to around 60,000 shapes. In images with the default dimensions of 500x500, these numbers drop to 550 and 10,000 respectively.

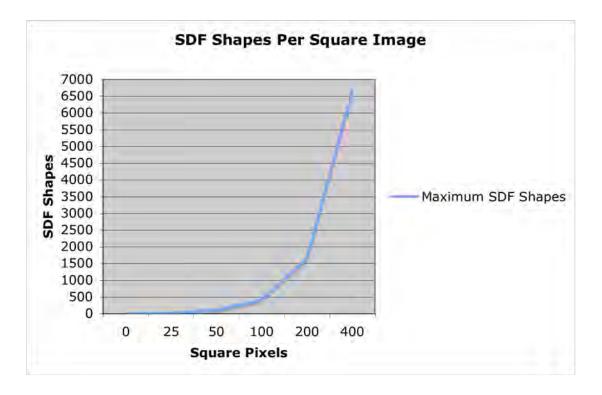


Figure 6.8: File size to encoding density.

For perspective, consider the encoding density when using Steghide's graph theoretic approach to steganography. Steghide has a feature that statically returns the potential embedding capacity of a file in bytes. When using Steghide's algorithm, the large architectural drawing has a capacity of 166 kilobytes. This number when translated into SDF is just below 19,000 shapes. The smaller 500x500 drawing is only capable of around 3500 shapes or an encoding capacity of 30.5 kilobytes when using this process. The range we established with our estimates, 3200 to 60,000 shapes using the large image, encapsulates this empirical data. This further proves that we have designed RoughDraft to allow for the creation of large-scale designs using a creditable steganography process.

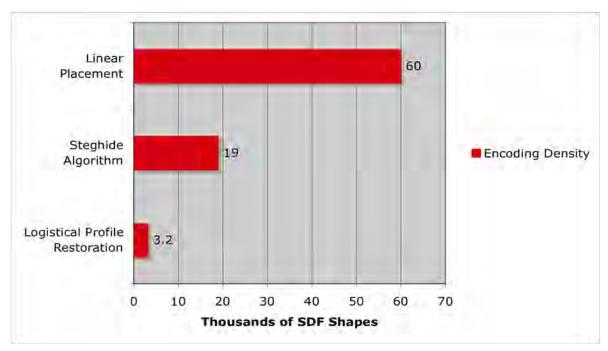


Figure 6.9: Comparison of the encoding density of different steganography algorithms using the large architectural drawing.

Steghide's embedding capacity was also used to produce some numbers related to JPEG images for future use. The JPEG version of the architectural plan has an encoding density of around 3.6 kilobytes or 410 SDF shapes. In contrast, the stepper motor drawing from *Figure 6. 2* is only capable of containing 475 bytes or 7 shapes when the JPEG file format is used. These values provide some insight into the tradeoffs of using compressed images. The process seems completely unacceptable for the stepper motor design. For architectural drawing however, the payoff in terms of file size is spectacular. The entire image could be recreated in a JPEG format with over 300 shapes to spare, and the footprint of the image would shrink from 4.1 megabytes to 119 kilobytes. To combat the low

encoding density issue, applications adopting JPEG as their native format would need to pick apart the JPEG compression algorithm to find the optimal approach, but it would be well worth the effort for the space savings.

Table 6.2.
Table 6.1 is updated to include alternative bitmap formats and the new JPEG representation

Vector Bitmap 256 (vbmp)	254
Vector Bitmap 16 (vbmp)	137
AutoCAD 2007 (dxf)	33
AutoCAD 2007 (dwg)	25
TurboCAD file format (tc2)	20
Portable Network Graphics (png)	16
Vector JPEG	13
AutoCAD R12 (dxf)	12
AutoCAD R12 (dwg)	8
AutoCAD R12	4
(dxf, manually reduced interpretation)	

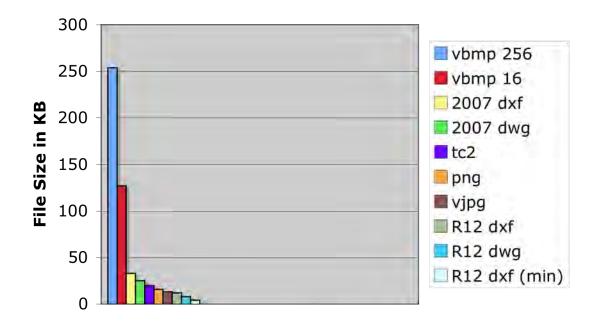


Figure 6.10
Graph in Figure 6.5 is updated to include the new JPEG representation

### **CHAPTER 7. CONCLUSION**

We have accomplished our goal of creating a single source representation of a CAD image using steganography. The encoding density and low impact nature of the process leads us to believe that it is plausible to use this form of extension in a production application. Due to the potential for integration into an existing CAD application, there will be no further development on the RoughDraft prototype. RoughDraft is a close ended prototype. Also known as throwaway prototypes, these programs are more adept at displaying interesting new features, but lack the underlying architecture to support long-term development. Instead of attempting to build RoughDraft into a final system, the concepts should instead be transferred to other outside applications.

Combining this research into a stable software product is the next step to creating a viable file format alternative. To present the extension feature, a subset of the features available in contemporary CAD programs needed to be implemented from scratch in RoughDraft. This is a distinct disadvantage to throwaway prototyping, but feature creep was kept to a minimum to lessen the impact. Creating the infrastructure needed to build a robust and extensible CAD application is an exhaustive process. Rather than adding more of these features

into RoughDraft, time should instead be applied to researching the data storage backend.

Seeding different industries with RoughDraft-like applications would be helpful in assessing the opportunity cost of the tradeoffs found in this research. Polished single source formats could be a disruptive innovation in industries dominated by proprietary formats if designed properly. Beta testers and market research will be crucial to developing the final product, because design choices that work for developers are not guaranteed to work for their clients. For example, bitmaps may have unforeseen, inherent limitations that make them an unlikely candidate for large-scale consumption. Analyzing these type of data will indicate what features are needed to encourage a wider user base. Clients that use a piece of software are the best source of design feedback, and the more organizations that get involved with this project, the greater access developers will have to this information.

Motivating improvements in non-proprietary areas of individual software products is the paramount goal of RoughDraft's developer. There are a variety of neutral open CAD formats that share this goal, like IGES and STEP, but Autodesk DWG files are still the most commonly used file format in the industry. The high adoption rate of DWG helps Autodesk grab about 85% of the market share in the CAD industry. Almost all of their competitors license the format, and many government contracts even require designs to be submitted in DWG. The industry is currently so drawn to their model that innovation is at risk.

The sharing problem is ultimately the result of keeping the status quo and does not necessarily represent the way CAD design has to be done. From an outside perspective, it is difficult to understand why the industry exists in its current state. Fortunately, ignorance can at times lead to insightful observations. The caveats to using tools and techniques geared towards the Autodesk style of development may not satisfy all customers, and users should not have to subscribe to them if it limits their productivity. Our solution gives a client the option of standardizing transactions with their designer on a single source representation of a project. This process avoids the maintenance associated with managing two file formats and should lead to a more user-friendly experience for both parties.

# **SOURCE CODE APPENDIX**

```
#ifndef magicnumbers h
#define magicnumbers_h
//BMP Offsets as established by standard
#define BSpace 0
#define MSpace 1
#define BEGINBMPSZ 2
#define ENDBMPSZ 5
#define BEGINAPPSPACE 6
#define ENDAPPSPACE 9
#define BEGINPXLOFFSET 10
#define ENDPXLOFFSET 13
#define BEGINHDRREM 14
#define ENDHDRREM 17
#define BEGINWIDTH 18
#define ENDWIDTH 21
#define BEGINHEIGHT 22
#define ENDHEIGHT 25
#define BEGINPLANES 26
#define ENDPLANES 27
#define BEGINDEPTH 28
#define ENDDEPTH 29
#define BEGINBI RGB 30
#define ENDBI RGB 33
#define BEGINRAWDATASZ 34
#define ENDRAWDATASZ 37
#define BEGINHORRES 38
#define ENDHORRES 41
#define BEGINVERRES 42
#define ENDVERRES 45
#define BEGINPALCOLORS 46
#define ENDPALCOLORS 49
#define BEGINIMPCOLORS 50
#define ENDIMPCOLORS 53
#define RAWDATASTART 54
//Other Data
#define PRIMARYCOLORS 3
#define BYTESIZE 8
#define HEXDIGITSIZE 4
#define PADDING 0x00
#endif
```

```
#ifndef bmpmanip_h
#define bmpmanip h
enum axisEnum {
   xAxis,
   yAxis
};
class bmpmanip{
    public:
    virtual ~bmpmanip();
    static bmpmanip* getBMPmanip(const char* inputFile, int
version):
    static bmpmanip* qetBMPmanip(const char* inputFile,
unsigned long width, unsigned long height, int version);
    //public utilities
    virtual void printHeaderInfo() = 0;
    virtual void colorWipe(int r, int g, int b) = 0;
    virtual void write(const char* outputFile) = 0;
    virtual void updatePixel(int r, int g, int b, int x, int
    virtual void updateLine(int r, int g, int b, int
coordinate, axisEnum axis) = 0;
    virtual int getVersion() = 0;
    virtual void serialize(const unsigned char memblock[],
int size) = 0:
    virtual void deserialize(unsigned char* memblock, int
size) = 0;
    //Get BMP Parameters
    virtual unsigned long getSizeInBytes() = 0;
    virtual unsigned long getPixelOffset() = 0;
    virtual unsigned long getHeaderRemain() = 0;
    virtual unsigned long getWidth() = 0;
    virtual unsigned long getHeight() = 0;
    virtual unsigned long getPlanes() = 0;
    virtual unsigned long getPixelDepth() = 0;
    virtual unsigned long getBI RGB() = 0;
    virtual unsigned long getRawDataSize() = 0;
    virtual unsigned long getHorizonalres() = 0;
    virtual unsigned long getVerticalres() = 0;
    virtual unsigned long getPaletteColors() = 0;
    virtual unsigned long getImpColors() = 0;
    protected:
```

```
bmpmanip();

private:
   bmpmanip(const bmpmanip&);
};
#endif
```

```
#include "bmpmanip.h"
#include "v1bmpmanip.h"
#include "magicnumbers.h"
#include <stdio.h>
bmpmanip::bmpmanip(){}
bmpmanip::~bmpmanip(){}
bmpmanip* bmpmanip::getBMPmanip(const char* inputFile, int
version)
{
    bmpmanip* versionSpecific = new v1bmpmanip(inputFile);
    if(versionSpecific)
    {
     return versionSpecific;
    }
    else
     return NULL;
}
bmpmanip* bmpmanip::getBMPmanip(const char* inputFile,
unsigned long width, unsigned long height, int version)
{
    bmpmanip* versionSpecific = new v1bmpmanip(inputFile,
width, height);
    if(versionSpecific)
    {
     return versionSpecific;
    else
     return NULL;
}
```

```
#ifndef v1bmpmanip h
#define v1bmpmanip_h
#include "bmpmanip.h"
class v1bmpmanip: public bmpmanip{
    public:
    v1bmpmanip(const char* inputFile);
    v1bmpmanip(const char* inputFile, unsigned long width,
unsigned long height);
   ~v1bmpmanip():
    //public utilities
    void printHeaderInfo();
    void colorWipe(int r, int g, int b);
    void write(const char* outputFile);
    void updatePixel(int r, int g, int b, int x, int y);
    void updateLine(int r, int g, int b, int coordinate,
axisEnum axis);
    int getVersion();
    void serialize(const unsigned char memblock[], int size);
    void deserialize(unsigned char* memblock, int size);
    //Get BMP Parameters
    unsigned long getSizeInBytes();
    unsigned long getPixelOffset();
    unsigned long getHeaderRemain();
    unsigned long getWidth();
    unsigned long getHeight();
    unsigned long getPlanes();
    unsigned long getPixelDepth();
    unsigned long getBI_RGB();
    unsigned long getRawDataSize();
    unsigned long getHorizonalres();
    unsigned long getVerticalres();
    unsigned long getPaletteColors();
    unsigned long getImpColors();
    //deny generation
    private:
    v1bmpmanip():
    v1bmpmanip(const v1bmpmanip&);
    //methods
    private:
    bool _initialize(const char* inputFile);
    //Utilities
```

```
bool checkIfValid():
    void parseHeaderInfo();
    void adjustNumbersForPadding();
    void reorder2byteLSB(unsigned long& data, int
farOffset);
    void reorder4byteLSB(unsigned long& data, int
farOffset):
    void _bmpOrder2byteValue(unsigned long data, int
valueOffset);
    void _bmpOrder4byteValue(unsigned long data, int
valueOffset);
    //data
    private:
    unsigned char* _memblock;
    int _size;
    int _pixelBytesPerLine;
    int _paddingBytesPerLine;
    int _bytesPerLine;
    //BMP Parameters
    unsigned long _sizeInBytes;
   unsigned long _pixelOffset;
    unsigned long _headerRemain;
    unsigned long _width;
    unsigned long _height;
   unsigned long _planes;
    unsigned long pixelDepth;
   unsigned long _BI_RGB;
    unsigned long _rawDataSize;
   unsigned long _horizonalres;
    unsigned long _verticalres;
   unsigned long _paletteColors;
   unsigned long _impColors;
};
#endif
```

```
#include "v1bmpmanip.h"
#include "magicnumbers.h"
#include <stdio.h>
#include <stdlib.h>
v1bmpmanip::v1bmpmanip(const char* inputFile)
{
    _sizeInBytes = 0;
    _pixelOffset = 0;
    _headerRemain = 0;
    _width = 0;
    _{\text{height}} = 0;
    _{planes} = 0;
    _pixelDepth = 0;
    _BI_RGB = 0;
    _rawDataSize = 0;
    _horizonalres = 0;
    _verticalres = 0;
    _paletteColors = 0;
    _impColors = 0;
    _pixelBytesPerLine = 0;
    _paddingBytesPerLine = 0;
    _bytesPerLine = 0;
    if(_initialize(inputFile))
     if(checkIfValid())
          printf("Validated\n");
     }
     else
     {
          printf("Failed Validation\n");
          return;
     }
     parseHeaderInfo();
        adjustNumbersForPadding();
    }
}
v1bmpmanip::v1bmpmanip(const char* inputFile, unsigned long
width, unsigned long height)
{
    _pixelOffset = 54;
    _headerRemain = 40;
    _width = width;
```

```
_height = height;
   _{planes} = 1;
   _pixelDepth = 24;
    _BI_RGB = 0;
   _horizonalres = 2835;
   verticalres = 2835;
   _paletteColors = 0;
   _impColors = 0;
    adjustNumbersForPadding();
    _sizeInBytes = _pixelOffset+(_bytesPerLine *
labs(_height));
   _size = _sizeInBytes;
   _rawDataSize = _sizeInBytes - _pixelOffset;
    _memblock = new unsigned char [_sizeInBytes];
    _memblock[BSpace] = 'B';
   memblock[MSpace] = 'M';
    _bmpOrder4byteValue(_sizeInBytes, BEGINBMPSZ);
   //Skip Unused, app specific Space
    //_memblock[6] - _memblock[9]
    _bmpOrder4byteValue(_pixelOffset, BEGINPXLOFFSET);
   bmpOrder4byteValue( headerRemain, BEGINHDRREM);
    _bmpOrder4byteValue(_width, BEGINWIDTH);
   bmpOrder4byteValue( height, BEGINHEIGHT);
    _bmpOrder2byteValue(_planes, BEGINPLANES);
    _bmpOrder2byteValue(_pixelDepth, BEGINDEPTH);
    _bmpOrder4byteValue(_BI_RGB, BEGINBI_RGB);
    _bmpOrder4byteValue(_rawDataSize, BEGINRAWDATASZ);
   _bmpOrder4byteValue(_horizonalres, BEGINHORRES);
    bmpOrder4byteValue( verticalres, BEGINVERRES);
    _bmpOrder4byteValue(_paletteColors, BEGINPALCOLORS);
    _bmpOrder4byteValue(_impColors, BEGINIMPCOLORS);
    for (int i = _pixelOffset; i < _sizeInBytes; i++) {</pre>
     memblock[i] = 0xFF;
    }
}
bool v1bmpmanip::_initialize(const char* inputFile)
```

```
FILE* filePtr = NULL;
  filePtr = fopen(inputFile, "rb");
  if (filePtr != NULL)
  {
    fseek(filePtr, 0, SEEK_END);
    _size = ftell(filePtr);
    rewind(filePtr);
    _memblock = new unsigned char [_size];
    fread(_memblock, 1, _size, filePtr);
    fclose(filePtr);
    printf("the complete file content is in memory \n");
  }
 else
    printf("Unable to open file\n");
    return false;
  return true;
}
v1bmpmanip::~v1bmpmanip()
{
    if(_memblock != NULL)
     delete[] _memblock;
}
bool v1bmpmanip::checkIfValid()
{
    bool valid = false;
    if(_memblock[BSpace] == 'B')
      if(_memblock[MSpace] == 'M')
     valid = true;
    return valid;
}
```

```
void v1bmpmanip::parseHeaderInfo()
{
    //Get Size of BMP file
   reorder4byteLSB( sizeInBytes, ENDBMPSZ);
    //Skip Unused, app specific Space
    //_memblock[6] - _memblock[9]
    //Get Pixel Offset
    _reorder4byteLSB(_pixelOffset, ENDPXLOFFSET);
   //Bytes of Header left
   _reorder4byteLSB(_headerRemain, ENDHDRREM);
   //Width in Pixels
   reorder4byteLSB( width, ENDWIDTH);
    //Height in Pixels
   _reorder4byteLSB(_height, ENDHEIGHT);
    //Planes Used
   _reorder2byteLSB(_planes, ENDPLANES);
   //Bits per pixel
   _reorder2byteLSB(_pixelDepth, ENDDEPTH);
    //Compression
   _reorder4byteLSB(_BI_RGB, ENDBI_RGB);
    //BMP Raw Data (After Header)
   reorder4byteLSB( rawDataSize, ENDRAWDATASZ);
   //Horizontal Resolution
   _reorder4byteLSB(_horizonalres, ENDHORRES);
    //Vertical Resolution
   _reorder4byteLSB(_verticalres, ENDVERRES);
    //Colors in Palette
    _reorder4byteLSB(_paletteColors, ENDPALCOLORS);
    //All Colors Important?
   _reorder4byteLSB(_impColors, ENDIMPCOLORS);
}
void v1bmpmanip::adjustNumbersForPadding()
```

```
_pixelBytesPerLine = (_pixelDepth*_width)/BYTESIZE;
    _paddingBytesPerLine = (_pixelBytesPerLine%HEXDIGITSIZE);
    _bytesPerLine = _pixelBytesPerLine+_paddingBytesPerLine;
}
void v1bmpmanip::printHeaderInfo()
    printf("Size %li bytes\n", _sizeInBytes);
    printf("Pixel Offset %li bytes\n", _pixelOffset);
    printf("Header Remaining %li bytes\n", _headerRemain);
    printf("Width %li pixels\n", _width);
    printf("Height %li pixels\n", _height);
    printf("Planes used %li\n", _planes);
    printf("Pixel Depth %li bits\n", _pixelDepth);
    printf("Compression Data %li\n", _BI_RGB);
    printf("Raw Data Size %li bytes\n", _rawDataSize);
    printf("Horizontal Resolution %li pixels/meter\n",
horizonalres);
    printf("Vertical Resolution %li pixels/meter\n",
_verticalres);
    printf("Colors in Palette %li\n", _paletteColors);
    printf("All Colors Important? %li\n", _impColors);
}
void v1bmpmanip::_reorder4byteLSB(unsigned long& data, int
farOffset)
{
    unsigned long temp = 0;
    data = memblock[far0ffset] << 24;</pre>
    temp = _memblock[far0ffset-1] << 16;</pre>
    data = data | temp;
    temp = memblock[far0ffset-2] << 8;</pre>
    data = data | temp;
    data = data | _memblock[far0ffset-3];
}
void v1bmpmanip:: reorder2byteLSB(unsigned long& data, int
farOffset)
{
    unsigned long temp = 0;
    temp = _memblock[far0ffset] << 8;</pre>
    data = data | temp;
```

```
data = data | _memblock[far0ffset-1];
}
void v1bmpmanip:: bmpOrder4byteValue(unsigned long data, int
valueOffset)
{
   _memblock[valueOffset] = (data);
   _memblock[valueOffset+1] = (data >> 8);
   _memblock[valueOffset+2] = (data >> 16);
   _memblock[valueOffset+3] = (data >> 24);
}
void v1bmpmanip::_bmpOrder2byteValue(unsigned long data, int
valueOffset)
{
   _memblock[valueOffset] = (data);
   _memblock[valueOffset+1] = (data >> 8);
}
void v1bmpmanip::write(const char* outputFile)
  FILE* filePtr = NULL;
  filePtr = fopen(outputFile, "w");
  if (filePtr != NULL)
  {
    fwrite(_memblock, 1, _size, filePtr);
   fclose(filePtr);
  }
 else
  {
    return;
  printf("New BMP file created \n");
}
void v1bmpmanip::colorWipe(int r, int g, int b)
{
    for(unsigned int i = 0; i < _height; i++)</pre>
     updateLine(r, g, b, i, xAxis);
}
```

```
unsigned long v1bmpmanip::getSizeInBytes()
    return _sizeInBytes;
}
unsigned long v1bmpmanip::getPixelOffset()
    return _pixelOffset;
}
unsigned long v1bmpmanip::getHeaderRemain()
    return _headerRemain;
}
unsigned long v1bmpmanip::getWidth()
    return _width;
}
unsigned long v1bmpmanip::getHeight()
    return _height;
}
unsigned long v1bmpmanip::getPlanes()
    return _planes;
}
unsigned long v1bmpmanip::getPixelDepth()
    return _pixelDepth;
}
unsigned long v1bmpmanip::getBI_RGB()
    return _BI_RGB;
}
unsigned long v1bmpmanip::getRawDataSize()
    return _rawDataSize;
}
unsigned long v1bmpmanip::getHorizonalres()
    return _horizonalres;
```

```
}
unsigned long v1bmpmanip::getVerticalres()
    return _verticalres;
}
unsigned long v1bmpmanip::getPaletteColors()
    return _paletteColors;
}
unsigned long v1bmpmanip::getImpColors()
    return _impColors;
}
void v1bmpmanip::updatePixel(int r, int g, int b, int x, int
y)
{
    int y0ffset = _pixel0ffset + (y*(_bytesPerLine)); //Get
Vertical component
    int updatePixelOffset = yOffset + ((_bytesPerLine /
_width)*x); //Shift by Horizontal component
    memblock[updatePixelOffset] = b;
    _memblock[updatePixelOffset+1] = g;
    _memblock[updatePixelOffset+2] = r;
}
void v1bmpmanip::updateLine(int r, int g, int b, int
coordinate, axisEnum axis)
{
    int offset = 0;
    int i = 0;
    //Colors y = value
    if (axis == yAxis) {
     offset = _pixelOffset + (coordinate*(_bytesPerLine));
     for(i = offset; i < offset + _pixelBytesPerLine;</pre>
i=i+PRIMARYCOLORS)
     {
          _{memblock[i]} = b;
```

```
_{memblock[i+1]} = g;
          _{memblock[i+2]} = r;
     }
     for(int j = 0; j < _paddingBytesPerLine; j++)</pre>
          _memblock[i+j] = PADDING;
     }
     return;
  }
  //Colors x = value
  if (axis == xAxis) {
      offset = _pixelOffset + ((_bytesPerLine /
_width)*coordinate);
      for(i = offset; i < _pixelBytesPerLine*_height;</pre>
i=i+_pixelBytesPerLine)
       _{memblock[i]} = b;
       _{memblock[i+1]} = g;
       _{memblock[i+2]} = r;
      return;
  }
  //Something is wrong
  printf("OH NO!");
}
int v1bmpmanip::getVersion()
{
    return 1;
}
void v1bmpmanip::serialize(const unsigned char memblock[],
int size)
{
     delete _memblock;
        _memblock = new unsigned char[size];
        for (int i = 0; i < size; i++) {
        _memblock[i] = memblock[i];
}
```

```
void v1bmpmanip::deserialize(unsigned char* memblock, int
size)
{
    if (_memblock != NULL) {
        if(size >= getSizeInBytes())
        {
            for (int i = 0; i < getSizeInBytes(); i++) {
                memblock[i] = _memblock[i];
            }
        }
        else {
        printf("Internal memblock is NULL");
      }
}
</pre>
```

111 RoughDraftJar: RoughDraft.mf CONSTANTClass RoughDraftPaletteViewClass RoughDraftMouseListenerClass RoughDraftKeyListenerClass RoughDraftDnDAppHandlerClass RoughDraftFrontEndClass RoughDraftComponentClass RoughDraftToolsPaletteClass RoughDraftClass RoughDraftHeader ToolPaletteMouseListenerClass CommandWindowKeyListenerClass libbmpmanipProxy.jnilib jar cmf RoughDraft.mf RoughDraft.jar ./Utilities/\*.class \*.class libbmpmanipProxy.jnilib: RoughDraftHeader ./build/Release/libbmpmanipProxy.jnilib @cp ./build/Release/libbmpmanipProxy.jnilib libbmpmanipProxy.jnilib @printf "Found Proxy JNI lib\n" RoughDraftHeader: RoughDraftClass javah RoughDraft

CONSTANTClass: ./Utilities/CONSTANT.java javac ./Utilities/CONSTANT.java

RoughDraftPaletteViewClass:

./Utilities/RoughDraftPaletteView.java javac ./Utilities/RoughDraftPaletteView.java

RoughDraftShape: ./Utilities/RoughDraftShape.java ./Utilities/RoughDraftShape.java

RoughDraftKeyListenerClass:

./Utilities/RoughDraftKeyListener.java javac ./Utilities/RoughDraftKeyListener.java

CommandWindowKeyListenerClass:

./Utilities/CommandWindowKeyListener.java javac ./Utilities/CommandWindowKeyListener.java

RoughDraftMouseListenerClass:

./Utilities/RoughDraftMouseListener.java javac ./Utilities/RoughDraftMouseListener.java

ToolPaletteMouseListenerClass:

./Utilities/ToolPaletteMouseListener.java javac ./Utilities/ToolPaletteMouseListener.java

RoughDraftDnDAppHandlerClass:

./Utilities/RoughDraftDnDAppHandler.java javac ./Utilities/RoughDraftDnDAppHandler.java

```
RoughDraftFrontEndClass: ./Utilities/RoughDraftFrontEnd.java
    javac ./Utilities/RoughDraftFrontEnd.java

RoughDraftComponentClass:
./Utilities/RoughDraftComponent.java
    javac ./Utilities/RoughDraftComponent.java

RoughDraftToolsPaletteClass:
./Utilities/RoughDraftToolsPalette.java
    javac ./Utilities/RoughDraftToolsPalette.java

RoughDraftClass: RoughDraft.java
    javac RoughDraft.java
    clean:
```

rm \*.class ./Utilities/\*.class \*.jar

```
#include "jni.h"
#include "RoughDraft.h"
#include "bmpmanip.h"
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string>
bmpmanip* proxy = NULL;
JNIEXPORT void JNICALL Java_RoughDraft_proxyInit(JNIEnv *env,
jobject, jstring inputFile)
    const jbyte* argvv = (jbyte*)env-
>GetStringUTFChars(inputFile, NULL);
    const char* argv =(char *) argvv;
    proxy = bmpmanip::getBMPmanip(argv, 1);
}
JNIEXPORT void JNICALL Java_RoughDraft_proxyCreateNew(JNIEnv
*env, jobject, jstring inputFile, jint width, jint height)
    const jbyte* argvv = (jbyte*)env-
>GetStringUTFChars(inputFile, NULL);
    const char* argv =(char *) argvv;
    proxy = bmpmanip::getBMPmanip(argv, width, height, 1);
    const char* saveFile = strdup(env-
>GetStringUTFChars(inputFile, 0));
    proxy->write(saveFile);
}
JNIEXPORT ibyteArray JNICALL
Java_RoughDraft_proxyLoadFile(JNIEnv *env, jobject)
{
    jbyteArray jb;
    if (proxy != NULL) {
    unsigned char* memblock = new unsigned char[proxy-
>getSizeInBytes()];
    jb = env->NewByteArray(proxy->getSizeInBytes());
    proxy->deserialize(memblock, proxy->getSizeInBytes());
    env->SetByteArrayRegion(jb, 0, proxy->getSizeInBytes(),
(ibyte *)memblock);
```

```
}
    return (jb);
}
JNIEXPORT jint JNICALL
Java_RoughDraft_proxyGetPixelOffset(JNIEnv *, jobject)
    return proxy->getPixelOffset();
}
JNIEXPORT jint JNICALL
Java_RoughDraft_proxyGetSizeInBytes(JNIEnv *, jobject)
    return proxy->getSizeInBytes();
}
JNIEXPORT jint JNICALL Java_RoughDraft_proxyGetWidth(JNIEnv
*, jobject)
    return proxy->getWidth();
}
JNIEXPORT jint JNICALL Java RoughDraft proxyGetHeight(JNIEnv
*, jobject)
    return proxy->getHeight();
}
JNIEXPORT void JNICALL Java_RoughDraft_proxyWrite(JNIEnv
*env, jobject, jstring imgPath, jbyteArray imgBuf)
₹
    unsigned char* memblock = new unsigned char[proxy-
>getSizeInBytes()];
    memblock = (unsigned char*)env-
>GetByteArrayElements(imgBuf, NULL);
    proxy->serialize(memblock, proxy->getSizeInBytes());
    const char* saveFile = strdup(env-
>GetStringUTFChars(imgPath, 0));
    proxy->write(saveFile);
}
```

```
JNIEXPORT void JNICALL Java_RoughDraft_freeProxy(JNIEnv *,
jobject)
{
    delete proxy;
}
```

```
import Utilities.*;
import java awt *;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
class RoughDraft extends RoughDraftComponent
    public static void main(String args[])
     //Create class instance
     JTextArea paletteText = new JTextArea("\n
                                                   To View An
           Drag A BMP File Onto The Palette");
Image\n
     RoughDraft imgViewer = new RoughDraft(paletteText);
     RoughDraftFrontEnd frontEnd = new
RoughDraftFrontEnd(imgViewer);
     RoughDraftKeyListener keyboardHandler = new
RoughDraftKeyListener(imgViewer);
     RoughDraftMouseListener mouseHandler = new
RoughDraftMouseListener(imgViewer);
     RoughDraftToolsPalette tools = new
RoughDraftToolsPalette();
     ToolPaletteMouseListener toolsMouseHandler = new
ToolPaletteMouseListener(imgViewer);
     tools.setPreferredSize(new
Dimension(CONSTANT.DRAWPALETTEWIDTH,
CONSTANT.DEFAULTDISPLAYHEIGHT));
     tools.setBackground(Color.white);
     JTextArea commandWindow = new JTextArea("\n");
     CommandWindowKeyListener commandKeyHandler = new
CommandWindowKeyListener(imgViewer, commandWindow);
     //Drag and Drop Code
     RoughDraftDnDAppHandler ah = new
RoughDraftDnDAppHandler(imgViewer);
     ah.setOutput(paletteText);
     paletteText.setTransferHandler(ah);
     //Attach Keyboard Handler
     imgViewer.addKeyListener(keyboardHandler);
     paletteText.addKeyListener(keyboardHandler);
     commandWindow.addKeyListener(commandKeyHandler);
     frontEnd.addKeyListener(keyboardHandler);
```

```
//Finish up
     imgViewer.setPaletteFrame(frontEnd);
     imgViewer.addMouseListener(mouseHandler);
     tools.addMouseListener(toolsMouseHandler);
     frontEnd.setLayout(new BorderLayout(10,10));
     frontEnd.add(tools, BorderLayout.WEST);
     frontEnd.add(paletteText, BorderLayout.CENTER);
     frontEnd.add(commandWindow, BorderLayout.SOUTH);
     frontEnd.setSize(new
Dimension (CONSTANT. DEFAULTDISPLAYWIDTH,
CONSTANT.DEFAULTDISPLAYHEIGHT));
     frontEnd.setVisible(true):
    }
    public RoughDraft(JTextArea jta)
     paletteText = jta;
     prepareText(paletteText);
    }
// RoughDraftComponent Interface
    public void createImage(String imgPath, int newWidth, int
newHeight)
     initialize();
     proxyCreateNew(imgPath, newWidth, newHeight);
     freeProxy();
     initializeImage(imgPath);
        shapeList.add(new shapeData(RoughDraftShape.SWATCH,
Color.white, 0, 0, newWidth, newHeight));
     contextSwitch(RoughDraftPaletteView.IMAGE);
     refreshImage();
    }
    public boolean initializeImage(String imgPath)
     if (imgPath != "")
          savedPath = imgPath;
          //Start External Code
          proxyInit(imgPath);
            imageByteArray = proxyLoadFile();
          pixelOffset = Math.abs(proxyGetPixelOffset());
          embeddingOffset =
```

```
pixelOffset+CONSTANT.BYTESPERPIXEL;
          sizeInBytes = Math.abs(proxyGetSizeInBytes());
          width = Math.abs(proxyGetWidth());
          height = proxyGetHeight();
          if (height < 0)</pre>
          {
               heightNegative = true;
               height = Math.abs(height);
          }
          freeProxy();
          //End External Code
          padding =
(((CONSTANT.PIXELDEPTH*width)/CONSTANT.BYTESIZE)%CONSTANT.HEX
DIGITSIZE);
          if(width > CONSTANT.MAXDISPLAYWIDTH || height >
CONSTANT.MAXDISPLAYHEIGHT)
          {
               initialize();
               setVisible(false);
               Image!");
               paletteText.setVisible(true);
               paletteFrame.add(paletteText,
BorderLayout.CENTER);
               paletteFrame.setSize(new
Dimension (CONSTANT. DEFAULTDISPLAYWIDTH,
CONSTANT.DEFAULTDISPLAYHEIGHT));
               return false;
          }
          else
               int smallImagePadding = 0;
               if (height < CONSTANT.DRAWPALETTEHEIGHT)</pre>
                    smallImagePadding =
CONSTANT.DRAWPALETTEHEIGHT - height;
               paletteFrame.setSize(new
Dimension(width+CONSTANT.JFRAMESTANDARDWIDTH+CONSTANT.DRAWPAL
ETTEWIDTH,
height+CONSTANT.JFRAMESTANDARDHEIGHT+smallImagePadding));
```

```
if(imageByteArray[CONSTANT.CODELETTEROFFSET]
== (byte)'S')
                {
    reclaimData(imageByteArray[CONSTANT.NUMBEROFSHAPESOFFSET]
);
                }
                return true;
          }
     }
        else
     {
          initialize();
          return false;
     }
    }
    public void contextSwitch(RoughDraftPaletteView view)
     if (view == RoughDraftPaletteView.IMAGE)
          paletteText.setVisible(false);
          setVisible(true);
          paletteFrame.add(this, BorderLayout.CENTER);
     }
     else
     {
          setVisible(false);
          initializeImage("");
          paletteText.setText("\n
                                      To View An Image\n
Drag A BMP File Onto The Palette");
          paletteText.setVisible(true);
          paletteFrame.add(paletteText, BorderLayout.CENTER);
          paletteFrame.setSize(new
Dimension (CONSTANT. DEFAULTDISPLAYWIDTH,
CONSTANT.DEFAULTDISPLAYHEIGHT));
     }
    }
```

```
public void drawShape(Color shapeColor, RoughDraftShape
shape, int startingX, int startingY, int endingX, int
endingY, boolean addToShapeList)
    {
     //Use shapes, because BufferedImage.getRGB() will help
us deserialize!
     if (initialized)
     {
          display.setColor(shapeColor);
          switch (shape) {
                case LINE:
                     Line2D line = new Line2D.Float(startingX,
startingY, endingX, endingY);
                     display.draw(line);
                     break:
                case SOUARE:
                     Polygon sq = createSquare(startingX,
startingY, endingX, endingY);
                     display.draw(sq);
                     break:
                case CIRCLE:
                     double radius =
Math.sqrt(Math.pow(startingX - endingX, 2) +
Math.pow(startingY - endingY, 2));
                     display.drawOval(startingX - (int)radius,
startingY - (int)radius, (int)radius*2, (int)radius*2);
                     break:
                case SWATCH:
                     Polygon sw = createSquare(startingX,
startingY, endingX, endingY);
                     display.fill(sw);
                     display.draw(sw);
                     break:
                default:
                     break:
          }
          if (addToShapeList)
                shapeList.add(new shapeData(shape, shapeColor,
startingX, startingY, endingX, endingY));
                redrawAllShapes();
```

```
}
          repaint();
     }
    }
    public void highlightShape(int x, int y)
     Iterator <shapeData> listIter = shapeList.iterator();
     boolean found = false;
     while (listIter.hasNext () && !found)
     {
          shapeData s = listIter.next();
          if (x \ge s.startX-5 \&\& x \le s.startX+5)
                if (y >= s.startY-5 && y <= s.startY+5)</pre>
                {
                     redrawAllShapes();
                     selectedShape = s;
                     drawShape(Color.yellow, s.shape,
s.startX, s.startY, s.endX, s.endY, false);
                     found = true;
                }
          }
          else
          {
                selectedShape = null;
                redrawAllShapes();
          }
    }
    public void highlightNextShape()
     Iterator <shapeData> listIter = shapeList.iterator();
     redrawAllShapes();
     if (selectedShape != null) {
          while (listIter.hasNext ())
          {
                shapeData s = listIter.next();
                if (s == selectedShape)
```

```
{
                     if (listIter.hasNext()) {
                          selectedShape = listIter.next();
                     }
                     else {
                          selectedShape =
shapeList.iterator().next();
                }
                highlightShape(selectedShape);
          }
     }
    }
    public void highlightLayer(Color layer)
     Iterator <shapeData> listIter = shapeList.iterator();
     selectedLayer = layer;
     redrawAllShapes();
     while (listIter.hasNext ())
     {
          shapeData s = listIter.next();
          if (s.color == layer)
                drawShape(Color.yellow, s.shape, s.startX,
s.startY, s.endX, s.endY, false);
     }
    }
    public void hideSelectedLayer()
     System.out.println("Layer Hidden");
     hiddenLayerList.add(selectedLayer);
     selectedLayer = null;
     refreshImage();
    }
    public void showSelectedLayer()
     System.out.println("Layer Visible");
     hiddenLayerList.remove(selectedLayer);
     selectedLayer = null;
```

```
refreshImage();
    }
    public void deleteShape()
     shapeList.remove(selectedShape);
     refreshImage();
    }
    public void reColorShape(Color recolor)
     int order = shapeList.indexOf(selectedShape);
     shapeList.remove(selectedShape);
     drawShape(recolor, selectedShape.shape,
selectedShape.startX, selectedShape.startY,
selectedShape.endX, selectedShape.endY, false);
     selectedShape = new shapeData(selectedShape.shape,
recolor, selectedShape.startX, selectedShape.startY,
selectedShape.endX, selectedShape.endY);
     shapeList.add(order, selectedShape);
     refreshImage();
    }
    public void moveStartpoint(int x, int y)
     if(selectedShape != null)
          int order = shapeList.indexOf(selectedShape);
          shapeList.remove(selectedShape);
          drawShape(selectedShape.color, selectedShape.shape,
x, y, selectedShape.endX, selectedShape.endY, false);
          selectedShape = new shapeData(selectedShape.shape,
selectedShape.color, x, y, selectedShape.endX,
selectedShape.endY);
          shapeList.add(order, selectedShape);
            refreshImage();
     }
    }
    public void moveEndpoint(int x, int y)
     if(selectedShape != null)
          int order = shapeList.indexOf(selectedShape);
```

```
shapeList.remove(selectedShape);
          drawShape(selectedShape.color, selectedShape.shape,
selectedShape.startX, selectedShape.startY, x, y, false);
          selectedShape = new shapeData(selectedShape.shape,
selectedShape.color, selectedShape.startX,
selectedShape.startY, x, y);
          shapeList.add(order, selectedShape);
            refreshImage();
     }
    }
    public void setShape(RoughDraftShape shape)
     currentShape = shape;
    }
    public RoughDraftShape getShape()
     return currentShape;
    }
    public void computeElementCodes()
     int offset = embeddingOffset;
     imageByteArray[CONSTANT.CODELETTEROFFSET] = (byte)'S';
     //Casting limits the current app to 256 shapes!
     imageByteArray[CONSTANT.NUMBEROFSHAPESOFFSET] =
(byte)shapeList.size();
     if(imageByteArray != null)
          for (int i = 0; i < shapeList.size(); i++)</pre>
                //Shape
    codeShapeIntoPixelComponents(shapeList.get(i).shape.shape
Index, CONSTANT.SMALLELEMENTBITS, offset);
               offset+=CONSTANT.SMALLELEMENTBITS;
               //Color
    codeShapeIntoPixelComponents(stegoUtils.convertSupportedC
olorIntoColorCode(shapeList.get(i).color),
```

```
CONSTANT.SMALLELEMENTBITS, offset);
               offset+=CONSTANT.SMALLELEMENTBITS;
               //Start
    codeShapeIntoPixelComponents(shapeList.get(i).startX,
CONSTANT.LARGEELEMENTBITS, offset);
               offset+=CONSTANT.LARGEELEMENTBITS;
    codeShapeIntoPixelComponents(shapeList.get(i).startY,
CONSTANT.LARGEELEMENTBITS, offset);
               offset+=CONSTANT.LARGEELEMENTBITS:
               //End
    codeShapeIntoPixelComponents(shapeList.get(i).endX,
CONSTANT.LARGEELEMENTBITS, offset);
               offset+=CONSTANT.LARGEELEMENTBITS:
    codeShapeIntoPixelComponents(shapeList.get(i).endY,
CONSTANT.LARGEELEMENTBITS, offset);
               offset+=CONSTANT.LARGEELEMENTBITS;
          }
     }
     //Reload the working set with the new embedded color
data
     loadImageFromImageByteArray();
    }
   public void saveImage()
     proxyInit(savedPath);
     loadImageByteArrayFromImage();
     proxyWrite(savedPath, imageByteArray);
     freeProxy();
   public void saveImage(String path)
     proxyInit(savedPath);
     loadImageByteArrayFromImage();
     proxyWrite(path, imageByteArray);
     savedPath = path;
     freeProxy();
```

```
// End RoughDraftComponent Interface
    //TODO: This is still needed. Try to remove it
    public void setPaletteFrame(JFrame frame)
    {
     paletteFrame = frame;
    }
    public void paint(Graphics g)
     if(imageByteArray != null)
          if (!initialized)
                drawBMP();
                initialized = true;
          }
          g.drawImage(image, 0, 0, this);
   }
    //Private Methods
    private void loadImageByteArrayFromImage()
     int offset = pixelOffset;
     int y = 0;
     if(!heightNegative)
          for (y = proxyGetHeight()-1; y >= 0; y--,
offset+=padding)
          {
                for (int x = 0; x < proxyGetWidth(); x++,
offset+=CONSTANT.BYTESPERPIXEL)
                {
                     int pixel = image.getRGB(x, y) &
0x00FFFFFF;
                     writePixel(offset, pixel);
                }
          }
```

```
}
     else
     {
          for (y = 0; y < Math.abs(proxyGetHeight()); y++,</pre>
offset+=padding)
                for (int x = 0; x < proxyGetWidth(); x++,
offset+=CONSTANT.BYTESPERPIXEL)
                  int pixel = image.getRGB(x, y) & 0x00FFFFFF;
                  writePixel(offset, pixel);
                }
       }
     }
    }
    private void loadImageFromImageByteArray()
     int offset = pixelOffset;
     int pixelBuf[] = {0, 0, 0};
     if(!heightNegative)
          for (int y = height-1; y >= 0 && offset <
CONSTANT.TOTALELEMENTBITS*shapeList.size()+pixelOffset; y--)
          {
                for (int x = 0; x < width && offset <
CONSTANT.TOTALELEMENTBITS*shapeList.size()+pixelOffset; x++,
offset+=CONSTANT.BYTESPERPIXEL)
                {
                     pixelBuf =
stegoUtils.unpackPixel(imageByteArray, offset);
                     image.setRGB(x, y, buildRGB(pixelBuf));
                }
          }
     }
     else
          for (int y = 0; y < height && offset <</pre>
CONSTANT.TOTALELEMENTBITS*shapeList.size()+pixelOffset; y++)
          {
```

```
for (int x = 0; x < width && offset <
CONSTANT.TOTALELEMENTBITS*shapeList.size()+pixelOffset; x++,
offset+=CONSTANT.BYTESPERPIXEL)
                {
                     pixelBuf =
stegoUtils.unpackPixel(imageByteArray, offset);
                     image.setRGB(x, y, buildRGB(pixelBuf));
                }
          }
     }
    }
    private void writePixel(int offset, int pixel)
     imageByteArray[offset] = (byte)( pixel & 0x000000FF);
//Blue
     imageByteArray[offset+1] = (byte)((pixel & 0x0000FF00))
>> 8); //Green
     imageByteArray[offset+2] = (byte)((pixel & 0x00FF0000))
>> 16); //Red
    }
    private int buildRGB(int pixelBuf[])
     //
               ARGB
     //\text{rgb} = 0 \times \text{FF00FF00}; // \text{green}
     int pixel = 0xFF000000 | (pixelBuf[2] << 16); //Blue
                         | (pixelBuf[1] << 8 ); //Green</pre>
     pixel
               = pixel
               = pixel
                            | (pixelBuf[0]);
     pixel
     return pixel;
    }
    private Polygon createSquare(int startingX, int
startingY, int endingX, int endingY)
     int xPoints[] = {startingX, endingX, endingX,
startingX};
     int yPoints[] = {startingY, startingY, endingY,
endingY};
     return new Polygon(xPoints, yPoints, 4);
    }
```

```
private void drawBMP()
     Graphics2D g2 = createGraphics2D(width, height);
     //Display BMP
     int offset = pixelOffset;
     int i = 0; //Total Pixels
     int x = 0;
     int y = 0;
     if(!heightNegative)
     {
          y = height-1; //Position
     }
     while (offset < sizeInBytes)</pre>
          int pixelBuf[] = {0, 0, 0};
          pixelBuf = extractPixel(offset);
          image.setRGB(x, y, buildRGB(pixelBuf));
          x++; i++; offset+=CONSTANT.BYTESPERPIXEL;
          if(i % width == 0)
                offset+=padding;
                i+=width;
                x = 0;
                if (!heightNegative)
                     y--;
                }
                else
                {
                     y++;
                }
          }
     }
     display = g2;
    }
    private Graphics2D createGraphics2D(int w, int h)
     Graphics2D g2 = null;
     if (image == null || image.getWidth() != w ||
image.getHeight() != h)
```

```
{
            image = (BufferedImage) createImage(w, h);
        }
        g2 = image.createGraphics();
        q2.setBackground(getBackground());
        g2.clearRect(0, 0, w, h);
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        return g2;
    }
    private int [] extractPixel(int offset)
     int pixelBuf[] = {0, 0, 0};
        for (int i = 0; i < pixelBuf.length; i++)</pre>
     {
          pixelBuf[i] = (short)imageByteArray[offset+i];
          if(pixelBuf[i] < 0)</pre>
                pixelBuf[i] = CONSTANT.COLORDEPTH +
pixelBuf[i];
          }
     }
     return pixelBuf;
    }
    private void initialize()
     imageByteArray = null;
     display = null;
     pixelOffset = 0;
     embeddingOffset = 0;
     sizeInBytes = 0;
     width = 0;
     height = 0;
     initialized = false;
     savedPath = null;
     padding = 0;
     shapeList.clear();
    }
```

```
private void prepareText(JTextArea jta)
     ita.getCaret().setBlinkRate(0);
     jta.setBackground(CONSTANT.DEFAULTBACKGROUND);
     jta.setFont(CONSTANT.DEFAULTFONT);
     jta.setForeground(CONSTANT.DEFAULTTEXT);
     jta.setDisabledTextColor(CONSTANT.DEFAULTTEXT);
     jta.setEnabled(false);
    }
   private void codeShapeIntoPixelComponents(int shapeCode,
int numberOfPixelComponents, int pixelComponentIndex)
     int pixelComponentColorCode = 0;
     int bitmask = (int)Math.pow(2, numberOfPixelComponents-
1);
     for (int i = numberOfPixelComponents-1; i >= 0; i--,
pixelComponentIndex++, bitmask/=2)
          //Grab color code, embed code information, reinsert
into the image buffer
          pixelComponentColorCode =
imageByteArray[pixelComponentIndex];
          imageByteArray[pixelComponentIndex] =
stegoUtils.steganographize(pixelComponentColorCode,
(shapeCode & bitmask) >> i);
     }
     System.out.println(shapeCode);
    }
   private void reclaimData(int numberOfShapes)
     int embeddedPixelBuf[] = extractPixel(embeddingOffset);
     int pixelBuf[] = {0, 0, 0};
     int code = 0;
     for (int i = 1; i < 24*number0fShapes; <math>i++)
          pixelBuf = extractPixel(embeddingOffset+(i*3));
          embeddedPixelBuf = concat(embeddedPixelBuf,
pixelBuf);
     }
```

```
for (int j = 0; j < numberOfShapes; j++)</pre>
          shapeData acquiredShape = new
shapeData(RoughDraftShape.UNDEF, Color.white, 0, 0, 0, 0);
          switch
(stegoUtils.pixelComponentsIntoCodeShape(embeddedPixelBuf,
0+(j*CONSTANT.TOTALELEMENTBITS), 4)) {
               case 0:
                  acquiredShape.shape = RoughDraftShape.LINE;
                case 1:
                  acquiredShape.shape =
RoughDraftShape.SQUARE;
                     break:
               case 2:
                  acquiredShape.shape =
RoughDraftShape.CIRCLE;
                     break:
               case 3:
                  acquiredShape.shape =
RoughDraftShape.SWATCH;
                     break:
               default:
                     break;
          }
          acquiredShape.color =
stegoUtils.convertCodeRangeIntoSupportedColor(stegoUtils.pixe
lComponentsIntoCodeShape(embeddedPixelBuf,
4+(i*CONSTANT.TOTALELEMENTBITS), 4));
          acquiredShape.startX =
stegoUtils.pixelComponentsIntoCodeShape(embeddedPixelBuf,
8+(j*CONSTANT.TOTALELEMENTBITS), 16);
          acquiredShape.startY =
stegoUtils.pixelComponentsIntoCodeShape(embeddedPixelBuf,
24+(j*CONSTANT.TOTALELEMENTBITS),16);
          acquiredShape.endX =
stegoUtils.pixelComponentsIntoCodeShape(embeddedPixelBuf,
40+(j*CONSTANT.TOTALELEMENTBITS),16);
          acquiredShape.endY =
stegoUtils.pixelComponentsIntoCodeShape(embeddedPixelBuf,
56+(j*CONSTANT.TOTALELEMENTBITS),16);
          shapeList.add(acquiredShape);
```

```
}
    }
    private int[] concat(int[] A, int[] B)
     int[] C = new int[A.length+B.length];
     System.arraycopy(A, 0, C, 0, A.length);
     System.arraycopy(B, 0, C, A.length, B.length);
     return C;
    }
    private void redrawAllShapes()
     Iterator <shapeData> listIter = shapeList.iterator();
     while (listIter.hasNext ())
          shapeData s = listIter.next();
          if (!hiddenLayerList.contains(s.color)) {
               drawShape(s.color, s.shape, s.startX,
s.startY, s.endX, s.endY, false);
    private void refreshImage()
     drawBMP();
     redrawAllShapes();
     repaint();
    }
    private void highlightShape(shapeData shape)
     Iterator <shapeData> listIter = shapeList.iterator();
     boolean found = false;
     while (listIter.hasNext () && !found)
          shapeData s = listIter.next();
          if (s == shape)
          {
               redrawAllShapes();
               selectedShape = s;
               drawShape(Color.yellow, s.shape, s.startX,
```

```
s.startY, s.endX, s.endY, false);
               found = true;
          }
     if(!found)
          selectedShape = null;
          redrawAllShapes();
    }
   }
   //Load native library
   static {System.loadLibrary("bmpmanipProxy");}
   //Native method declaration
   native void
                   proxyInit(String imgPath);
   native void
                   proxyCreateNew(String imgPath, int width,
int height);
   native byte[]
                   proxyLoadFile();
                   proxyGetPixelOffset();
   native int
   native int
                   proxyGetSizeInBytes();
   native int
                   proxyGetWidth();
   native int
                   proxyGetHeight();
                   proxyWrite(String imgPath, byte[]
   native void
imageByteArray);
                   freeProxy();
   native void
   //Initialized by external code//
   private byte imageByteArray[] = null;
   private int pixelOffset = 0;
   private int embeddingOffset = 0;
   private int sizeInBytes = 0;
   private int width = 0;
   private int height = 0;
   private BufferedImage image = null;
   private String savedPath = null;
   private JFrame paletteFrame = null;
   private JTextArea paletteText = null;
   private boolean initialized = false;
   private boolean heightNegative = false;
   private Graphics2D display = null;
   private RoughDraftShape currentShape =
RoughDraftShape.UNDEF;
   private int padding = 0;
```

```
private int tabbingIndex = 0;
    java.util.List<Color> hiddenLayerList = new
ArrayList<Color>();
    private Color selectedLayer = null;
    java.util.List<shapeData> shapeList = new
ArrayList<shapeData>();
    private shapeData selectedShape = null;
    private class shapeData
    {
     public RoughDraftShape shape;
     public Color color;
     public int startX;
     public int startY;
     public int endX;
     public int endY;
     shapeData(RoughDraftShape shape, Color shapeColor, int
startX, int startY, int endX, int endY)
          this.shape = shape;
          this.color = shapeColor;
          this.startX = startX;
          this.startY = startY;
          this.endX = endX;
          this.endY = endY;
     }
    }
}
class stegoUtils
    static public byte steganographize(int
pixelComponentColorCode, int codeBit)
    {
     if (codeBit == 0)
          pixelComponentColorCode = (pixelComponentColorCode
& 0x000000FE);
     }
     else
     {
          pixelComponentColorCode = (pixelComponentColorCode
& 0x000000FF);
         pixelComponentColorCode = (pixelComponentColorCode |
```

```
0x00000001);
     }
     pixelComponentColorCode =
packPixelComponentColorCode(pixelComponentColorCode);
     return (byte)pixelComponentColorCode;
    }
    static public int [] unpackPixel(byte imageByteArray[],
int offset)
    {
     int pixelBuf[] = {0, 0, 0};
        for (int i = 0; i < pixelBuf.length; i++)</pre>
          pixelBuf[i] = imageByteArray[offset+i];
          if(pixelBuf[i] < 0)</pre>
               pixelBuf[i] = CONSTANT.COLORDEPTH -
(pixelBuf[i] + CONSTANT.COLORDEPTH/2);
          }
     }
     return pixelBuf;
    static private int packPixelComponentColorCode(int
pixelComponentColorCode)
     if(pixelComponentColorCode > CONSTANT.COLORDEPTH/2)
          pixelComponentColorCode = -
(pixelComponentColorCode-= CONSTANT.COLORDEPTH/2);
     }
     return pixelComponentColorCode;
    }
    static public int extractEmbeddedBit(int
pixelComponentColorCode)
     int codeBit = pixelComponentColorCode & 0x00000001;
     if (codeBit == 0)
```

```
{
          return 0;
     }
     else
     {
          return 1;
     }
    }
    static public int pixelComponentsIntoCodeShape(int
embeddedByteArray[], int pixelComponentIndex, int
numberOfPixelComponents)
     int code = 0;
     int iter = 1;
     for (int i = pixelComponentIndex; i <</pre>
pixelComponentIndex+numberOfPixelComponents; i++, iter++)
          code = code |
(stegoUtils.extractEmbeddedBit(embeddedByteArray[i]) <<</pre>
(numberOfPixelComponents - iter));
     }
     System.out.println(code);
     return code;
    }
    static public Color
convertCodeRangeIntoSupportedColor(int colorCode)
     Color codedColor = Color.white;
     switch (colorCode)
          case CONSTANT.BLACK:
                codedColor = Color.black;
                break;
          case CONSTANT.RED:
                codedColor = Color.red;
                break;
          case CONSTANT.GREEN:
                codedColor = Color.green;
                break:
```

```
case CONSTANT.BLUE:
                codedColor = Color.blue;
                break;
          default:
                codedColor = Color.white;
                break;
     }
     return codedColor;
    }
    static public int
convertSupportedColorIntoColorCode(Color suppColor)
     int codedColor = 15;
     if (suppColor == Color.black)
          codedColor = CONSTANT.BLACK;
     if (suppColor == Color.red)
          codedColor = CONSTANT.RED;
     if (suppColor == Color.green)
          codedColor = CONSTANT.GREEN;
     if (suppColor == Color.blue)
          codedColor = CONSTANT.BLUE;
     return codedColor;
    }
}
```

```
package Utilities;
import java.awt.*;
public class CONSTANT{
    //BMP Constants
    public static final int COLORDEPTH
                                          = 256;
    public static final int PIXELDEPTH
                                              24:
    public static final int BYTESIZE
                                               8:
    public static final int HEXDIGITSIZE
                                               4;
    public static final int BYTESPERPIXEL =
                                               3;
    //JFrame Constants
    public static final int JFRAMESTANDARDWIDTH
                                                  = 22;
   public static final int JFRAMESTANDARDHEIGHT =
32+32/*Menu*/:
    public static final int DRAWPALETTEWIDTH
                                                  = 50:
    public static final int DRAWPALETTEHEIGHT
                                                  = 405;
    //Scaling Constants
    public static final int MAXDISPLAYWIDTH
                                                  = 1780;
    public static final int MAXDISPLAYHEIGHT
                                                     865:
    public static final int DEFAULTDISPLAYWIDTH
                                                     500:
    public static final int DEFAULTDISPLAYHEIGHT =
                                                     500:
    public static final int DEFAULTFOLD
                                                       2:
    public static final int DEFAULTSCALE
                                                       1:
    //Style Constants
    public static final Font DEFAULTFONT = new Font("Arial",
Font.BOLD, 13);
    public static final Color DEFAULTBACKGROUND =
Color.black;
   public static final Color DEFAULTTEXT = Color.white;
    //Shapes Constants
    public static final int STARTPOINT
                                        = 15:
    public static final int ENDPOINT
                                        = 35;
    public static final int OFFSET
                                        = 10;
   //Bit Banging Constants
    public static final int CODELETTEROFFSET
                                                     54:
    public static final int NUMBEROFSHAPESOFFSET =
                                                     55;
    public static final int SMALLELEMENTBITS
                                                  =
                                                     4;
    public static final int LARGEELEMENTBITS
                                                  = 16:
   public static final int TOTALELEMENTBITS
                                                  = 72:
    //Internal Color Range
    public static final int BLACK
                                       0;
```

```
public static final int RED = 1;
public static final int GREEN = 2;
public static final int BLUE = 3;
}
```

```
package Utilities;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class CommandWindowKeyListener extends JApplet
implements KeyListener
{
   private RoughDraftComponent imgViewer;
   private JTextArea window;
   private RoughDraftShape shape = null;
   private boolean shapeSet = false;
    private int coordinate[] = new int[4];
   private int ndx = 0;
    public CommandWindowKeyListener(RoughDraftComponent app,
JTextArea commandWindow)
     imgViewer = app;
     window = commandWindow;
    }
   public void keyPressed(KeyEvent e)
     if(e.getKeyCode() == KeyEvent.VK ENTER)
          String command = window.getText();
          processText(command);
     }
     if(e.getKeyCode() == KeyEvent.VK_BACK_SPACE)
          imgViewer.deleteShape();
     if(e.getKeyCode() == KeyEvent.VK_CONTROL)
          imgViewer.highlightNextShape();
   public void keyReleased(KeyEvent e)
   {
    }
```

```
public void keyTyped(KeyEvent e)
   }
   private void processText(String text)
     if(shapeSet)
          coordinate[ndx] =
Integer.parseInt(text.substring(19).trim());
          ndx++;
          window.setText(null);
          if (ndx == 4)
               imgViewer.drawShape(Color.black, shape,
coordinate[0], coordinate[1], coordinate[2], coordinate[3],
true):
               ndx = 0;
               shapeSet = false;
          }
          else
               window.setText("Coordinate
"+Integer.toString(ndx+1)+": ");
          }
     else if (text.trim().equals("ln") && !shapeSet)
          processShape(RoughDraftShape.LINE);
     else if (text.trim().equals("sq") && !shapeSet)
          processShape(RoughDraftShape.SQUARE);
     else if (text.trim().equals("cl") && !shapeSet)
          processShape(RoughDraftShape.CIRCLE);
     else if (text.trim().equals("sw") && !shapeSet)
```

```
processShape(RoughDraftShape.SWATCH);
}
else{
    window.setText(null);
}

private void processShape(RoughDraftShape parsedShape)
{
    window.setText(null);
    shape = parsedShape;
    shapeSet = true;
    window.setText("X Start Coordinate: ");
}
```

```
package Utilities;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.Component;
public abstract class RoughDraftComponent extends Component
    abstract public void createImage(String imgPath, int
newWidth, int newHeight);
    abstract public boolean initializeImage(String imgPath);
    abstract public void contextSwitch(RoughDraftPaletteView
view);
    abstract public void drawShape(Color shapeColor,
RoughDraftShape shape, int startingX, int startingY, int
endingX, int endingY, boolean addToShapeList);
    abstract public void highlightShape(int x, int y);
    abstract public void highlightNextShape();
    abstract public void highlightLayer(Color layer);
    abstract public void hideSelectedLayer();
    abstract public void showSelectedLayer();
    abstract public void deleteShape();
    abstract public void reColorShape(Color recolor);
    abstract public void moveStartpoint(int x, int y);
    abstract public void moveEndpoint(int x, int y);
    abstract public void setShape(RoughDraftShape shape);
    abstract public RoughDraftShape getShape();
    abstract public void computeElementCodes();
    abstract public void saveImage();
    abstract public void saveImage(String path);
}
```

```
package Utilities;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
//Drag and Drop
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.Transferable;
import java.io.BufferedReader;
import java.io.Reader;
public class RoughDraftDnDAppHandler extends TransferHandler
    private RoughDraftComponent imgViewer;
    private JTextArea output;
    public void TransferHandler() { }
    public RoughDraftDnDAppHandler(RoughDraftComponent app)
     imgViewer = app;
    }
    public boolean canImport(JComponent dest, DataFlavor[]
flavors) {
     // you bet we can!
     return true;
    public boolean importData(JComponent src, Transferable
transferable) {
     DataFlavor[] flavors =
transferable.getTransferDataFlavors();
     DataFlavor listFlavor = null;
     DataFlavor textFlavor = null;
     int lastFlavor = flavors.length - 1;
     String path = null;
     // Check the flavors and see if we find one we like.
     // If we do, save it.
     for (int f = 0; f <= lastFlavor; f++) {</pre>
          if (flavors[f].isFlavorJavaFileListType()) {
               listFlavor = flavors[f];
          }
          if (flavors[f].isFlavorTextType()) {
               textFlavor = flavors[f]:
```

```
}
     }
     // Ok, now try to display the content of the drop.
     try {
          if(listFlavor != null)
                java.util.List list =
(java.util.List)transferable.getTransferData(listFlavor);
                //List is Windows Compatible
                path = list.toString().replace("[", "");
                path = path.replace("]","");
          }
          else {
                //Fall to backup method. Linux needs this.
                if(textFlavor != null)
                {
                     BufferedReader br = null;
                     String line = null;
                     Reader r =
textFlavor.getReaderForText(transferable);
                     br = new BufferedReader(r);
                     line = br.readLine();
                     if (line != null) {
                          path = line.substring(7);
                     }
                     br.close();
                }
                //Utimately Fail out
                else
                {
                     // Don't know this flavor type yet...
                     //println("No text representation to
show.");
                }
          }
     catch (Exception e) {
          println("Caught exception decoding transfer:");
          println(e);
          return false;
     }
     if(path == null)
          return false;
     }
     //Context Switch
```

```
if(imgViewer.initializeImage(path))
     {
        imgViewer.contextSwitch(RoughDraftPaletteView.IMAGE);
     }
     return true;
    public void exportDone(JComponent source, Transferable
data, int action) {
     // Just let us know when it occurs...
     System.err.println("Export Done.");
    }
    public void setOutput(JTextArea jta) {
     output = jta;
    protected void print(Object o) {
     print(o.toString());
    protected void print(String s) {
     if (output != null) {
          output.append(s);
     }
     else {
          System.out.println(s);
     }
    }
    protected void println(Object o) {
     println(o.toString());
    protected void println(String s) {
     if (output != null) {
          output.append(s);
          output.append("\n");
     }
     else {
          System.out.println(s);
     }
    protected void println() {
     println("");
}
```

```
package Utilities;
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class RoughDraftFrontEnd extends JFrame{
    private RoughDraftComponent imgViewer;
   public RoughDraftFrontEnd(RoughDraftComponent app)
     super("Rough Draft");
     imgViewer = app;
     JMenu file = new JMenu("File");
     file.setMnemonic('F'):
     JMenuItem newItem = new JMenuItem("New");
     newItem.setMnemonic('N');
     file.add(newItem);
     JMenuItem openItem = new JMenuItem("Open");
     newItem.setMnemonic('0');
     file.add(openItem);
     JMenuItem stegoItem = new JMenuItem("Steganographize");
     stegoItem.setMnemonic('G');
     file.add(stegoItem);
     JMenuItem saveItem = new JMenuItem("Save");
     saveItem.setMnemonic('S');
     file.add(saveItem);
     JMenuItem saveAsItem = new JMenuItem("Save As...");
     saveItem.setMnemonic('A');
     file.add(saveAsItem);
     JMenuItem exitItem = new JMenuItem("Exit");
     exitItem.setMnemonic('X');
     file.add(exitItem);
     //adding action listener to menu items
     newItem.addActionListener(
          new ActionListener(){
          public void actionPerformed(ActionEvent e)
              SaveFileDialog sfd = new SaveFileDialog();
```

```
imgViewer.createImage(sfd.saveFile(), 500,
500):
          }
         }
        );
     openItem.addActionListener(
          new ActionListener(){
          public void actionPerformed(ActionEvent e)
    imgViewer.contextSwitch(RoughDraftPaletteView.TEXT);
          }
         }
     );
     stegoItem.addActionListener(
          new ActionListener(){
          public void actionPerformed(ActionEvent e)
          {
                imgViewer.computeElementCodes();
          }
     );
     saveItem.addActionListener(
          new ActionListener(){
          public void actionPerformed(ActionEvent e)
                imgViewer.saveImage();
         }
     );
     saveAsItem.addActionListener(
                                        new ActionListener(){
          public void actionPerformed(ActionEvent e)
              SaveFileDialog sfd = new SaveFileDialog();
                imgViewer.saveImage(sfd.saveFile());
          }
         }
     );
     exitItem.addActionListener(
          new ActionListener(){
          public void actionPerformed(ActionEvent e)
                dispose();
```

```
System.exit(0); //calling the method is a must
          }
         }
     );
     JMenuBar bar = new JMenuBar();
     setJMenuBar(bar);
     bar.add(file);
}
class SaveFileDialog {
    public String loadFile
    (Frame f, String title, String fileType) {
     FileDialog fd = new FileDialog(f, title,
FileDialog.LOAD);
     fd.setFile(fileType);
     fd.setLocation(50, 50);
     fd.setVisible(true);
     return fd.getDirectory()+fd.getFile();
    public String saveFile() {
     FileDialog fd = new FileDialog(new Frame(), "Save...",
FileDialog.SAVE);
     fd.setFile("*.bmp");
     fd.setLocation(50, 50);
     fd.setVisible(true);
     return fd.getDirectory()+fd.getFile();
}
```

```
package Utilities;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class RoughDraftKeyListener extends JApplet implements
KeyListener {
    private boolean ctrlDown = false;
   private RoughDraftComponent imgViewer;
   public RoughDraftKeyListener(RoughDraftComponent app)
     imgViewer = app;
    }
    public void keyPressed(KeyEvent e)
     if(e.getKeyCode() == KeyEvent.VK_CONTROL)
          ctrlDown = true;
     if(e.getKeyCode() == KeyEvent.VK_N && ctrlDown)
          //Context Switch
          if (imgViewer.isVisible()) {
    imgViewer.contextSwitch(RoughDraftPaletteView.TEXT);
          }
     }
     if(e.getKeyCode() == KeyEvent.VK X && ctrlDown)
          System.exit(0); //calling the method is a must
     }
    }
    public void keyReleased(KeyEvent e)
     if(e.getKeyCode() == KeyEvent.VK_CONTROL)
          ctrlDown = false;
    }
   public void keyTyped(KeyEvent e){}}
```

```
package Utilities;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class RoughDraftMouseListener extends JApplet
implements MouseListener
{
   private RoughDraftComponent palette = null;
   private int startX = 0;
   private int startY = 0;
   private int clickNumber = 0;
    private RoughDraftShape current = null;
   public RoughDraftMouseListener(RoughDraftComponent draft)
    palette = draft;
   public void mouseClicked(MouseEvent evt)
     if (palette.getShape() == RoughDraftShape.PENCIL)
          palette.highlightShape(evt.getX(), evt.getY());
          //Reset
          clickNumber = 0;
          startX = 0;
          startY = 0;
     else if (palette.getShape() ==
RoughDraftShape MOVESTART)
     {
          palette.moveStartpoint(evt.getX(), evt.getY());
          //Reset
          clickNumber = 0;
          startX = 0;
          startY = 0;
     else if (palette.getShape() == RoughDraftShape.MOVEEND)
          palette.moveEndpoint(evt.getX(), evt.getY());
          //Reset
          clickNumber = 0;
          startX = 0;
```

```
startY = 0;
     else if (clickNumber == 0 && (palette.getShape() ==
current))
     {
          clickNumber++;
          startX = evt.getX();
          startY = evt.getY();
     else if (palette.getShape() != current)
          clickNumber = 0;
          current = palette.getShape();
          startX = evt.getX();
          startY = evt.getY();
          clickNumber++;
     }
     else {
          palette.drawShape(Color.black, palette.getShape(),
startX, startY, evt.getX(), evt.getY(), true);
          //Reset
          clickNumber = 0;
          startX = 0;
          startY = 0;
     }
    }
   public void mousePressed(MouseEvent evt) {
        // do nothing
    public void mouseReleased(MouseEvent evt) {
        // do nothing
    }
   public void mouseEntered(MouseEvent evt) {
        // do nothing
    public void mouseExited(MouseEvent evt) {
       // do nothing
   }
}
```

```
package Utilities;
//Context Switch Constants
public enum RoughDraftPaletteView {IMAGE, TEXT}
```

```
package Utilities;
//Context Switch Constants
public enum RoughDraftShape {
LINE (0),
SQUARE (1),
CIRCLE (2),
SWATCH (3),
UNDEF (4),
PENCIL (5),
MOVESTART (6),
        (7)
MOVEEND
HIGHLAY (8);
public final int shapeIndex;
    RoughDraftShape(int shapeIndex) {
     this.shapeIndex = shapeIndex;
    }
}
```

```
package Utilities;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.geom.Line2D;
import java.awt.Polygon;
import java.awt.geom.Arc2D;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
public class RoughDraftToolsPalette extends JPanel
   private BufferedImage bimg;
   public void drawToolsPalette(int w, int h, Graphics2D g2)
     g2.setStroke(new BasicStroke(1.0f));
     q2.setColor(Color.black);
        // draw shapes
     g2.draw(new Line2D.Float(CONSTANT.STARTPOINT,
CONSTANT.OFFSET, CONSTANT.ENDPOINT, CONSTANT.OFFSET));
     int squareX[] = {CONSTANT.STARTPOINT, CONSTANT.ENDPOINT,
CONSTANT.ENDPOINT, CONSTANT.STARTPOINT);
     int squareY[] = {2*CONSTANT.OFFSET, 2*CONSTANT.OFFSET,
4*CONSTANT.OFFSET, 4*CONSTANT.OFFSET};
     Polygon sq = new Polygon(squareX, squareY, 4);
     q2.draw(sq);
     Arc2D circle = new Arc2D.Float(CONSTANT.STARTPOINT,
5*CONSTANT.OFFSET - (CONSTANT.OFFSET/2), 20, 20, 0, 360, 0);
     g2.draw(circle);
     q2.draw(drawSwatch(7, Color.black, q2));
        // draw select icon
     q2.draw(drawLine(10));
     // pencil shape goes here
     g2.draw(new Line2D.Double(CONSTANT.ENDPOINT-5,
10.5*CONSTANT.OFFSET, CONSTANT.ENDPOINT,
11*CONSTANT.OFFSET));
     g2.draw(new Line2D.Float(CONSTANT.STARTPOINT,
11*CONSTANT.OFFSET, CONSTANT.ENDPOINT, 11*CONSTANT.OFFSET));
     g2.draw(new Line2D.Double(CONSTANT.ENDPOINT-5,
11.5*CONSTANT.OFFSET, CONSTANT.ENDPOINT,
```

```
11*CONSTANT.OFFSET)):
     g2.draw(drawLine(12));
        // Colors Palette
     g2.draw(drawSwatch(13, Color.black, g2));
     g2.draw(drawSwatch(16, Color.red, g2));
     g2.draw(drawSwatch(19, Color.green, g2));
     g2.draw(drawSwatch(22, Color.blue, g2));
     g2.setColor(Color.black);
     int whiteX[] = {CONSTANT.STARTPOINT, CONSTANT.ENDPOINT,
CONSTANT.ENDPOINT, CONSTANT.STARTPOINT);
     int whiteY[] = {25*CONSTANT.OFFSET, 25*CONSTANT.OFFSET,
27*CONSTANT.OFFSET, 27*CONSTANT.OFFSET};
     Polygon whiteSwatch = new Polygon(whiteX, whiteY, 4);
     g2.draw(whiteSwatch);
        // Object Tools
     g2.draw(drawLine(28));
     // move start point
        g2.drawString("MvStart", 0, new
Float(29.5*CONSTANT.OFFSET)):
     g2.draw(drawLine(30));
     // move end point
        g2.drawString("MvEnd", 0, new
Float(31.5*CONSTANT.OFFSET));
     g2.draw(drawLine(32));
     // highlight layer
        g2.drawString(" Layer ", 0, new
Float(33.5*CONSTANT.OFFSET)):
     g2.draw(drawLine(34));
     // hide layer
        g2.drawString("HideLyr", 0, new
Float(35.5*CONSTANT.OFFSET)):
```

```
q2.draw(drawLine(36));
     // show layer
        g2.drawString("ShwLyr", 0, new
Float(37.5*CONSTANT.OFFSET));
     g2.draw(drawLine(38));
    }
    public Graphics2D createGraphics2D(int w, int h)
        Graphics2D g2 = null;
        if (bimg == null || bimg.getWidth() != w ||
bimg.getHeight() != h)
            bimg = (BufferedImage) createImage(w, h);
        }
        g2 = bimg.createGraphics();
        g2.setBackground(getBackground());
        g2.clearRect(0, 0, w, h);
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        return g2;
    }
    public void paint(Graphics g)
        Dimension d = new Dimension(getSize());
        Graphics2D g2 = createGraphics2D(d.width, d.height);
        drawToolsPalette(d.width, d.height, g2);
        g2.dispose();
        g.drawImage(bimg, 0, 0, this);
    }
    private Line2D drawLine(int multiplier)
     return new Line2D.Float(0, multiplier*CONSTANT.OFFSET,
CONSTANT.DRAWPALETTEWIDTH, multiplier*CONSTANT.OFFSET);
    }
    private Polygon drawSwatch(int multiplier, Color
swatchColor, Graphics2D g)
     int swatchX[] = {CONSTANT.STARTPOINT, CONSTANT.ENDPOINT,
```

```
CONSTANT.ENDPOINT, CONSTANT.STARTPOINT};
    int swatchY[] = {multiplier*CONSTANT.OFFSET,
    multiplier*CONSTANT.OFFSET, (multiplier+2)*CONSTANT.OFFSET,
    (multiplier+2)*CONSTANT.OFFSET};
    Polygon swatch = new Polygon(swatchX, swatchY, 4);
    g.setColor(swatchColor);
    g.fill(swatch);
    return swatch;
}
```

```
package Utilities;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ToolPaletteMouseListener extends JApplet
implements MouseListener
{
    private RoughDraftComponent palette = null;
    public ToolPaletteMouseListener(RoughDraftComponent
draft)
     palette = draft;
    public void mouseClicked(MouseEvent evt)
     int y = evt.getY();
     if (y < 2*CONSTANT.OFFSET)</pre>
          palette.setShape(RoughDraftShape.LINE);
     else if (y < 4*CONSTANT.OFFSET)</pre>
          palette.setShape(RoughDraftShape.SQUARE);
     else if (y < 6*CONSTANT.OFFSET)</pre>
          palette.setShape(RoughDraftShape.CIRCLE);
     else if (y < 10*CONSTANT.OFFSET)</pre>
          palette.setShape(RoughDraftShape.SWATCH);
     else if (y < 12*CONSTANT.OFFSET)</pre>
          palette.setShape(RoughDraftShape.PENCIL);
     else if (y < 16*CONSTANT.OFFSET)</pre>
```

```
doColorBasedAction(Color.black);
else if (y < 19*CONSTANT.OFFSET)</pre>
     doColorBasedAction(Color.red);
else if (y < 22*CONSTANT.OFFSET)</pre>
     doColorBasedAction(Color.green);
else if (y < 25*CONSTANT.OFFSET)</pre>
     doColorBasedAction(Color.blue);
else if (y < 28*CONSTANT.OFFSET)</pre>
     doColorBasedAction(Color.white);
else if (y < 31*CONSTANT.OFFSET)</pre>
     palette.setShape(RoughDraftShape.MOVESTART);
else if (y < 33*CONSTANT.OFFSET)</pre>
     palette.setShape(RoughDraftShape.MOVEEND);
else if (y < 35*CONSTANT.OFFSET)</pre>
     palette.setShape(RoughDraftShape.HIGHLAY);
else if (y < 37*CONSTANT.OFFSET)</pre>
     palette.hideSelectedLayer();
else if (y < 39*CONSTANT.OFFSET)</pre>
     palette.showSelectedLayer();
```

```
}
     else
     {
          palette.setShape(RoughDraftShape.UNDEF);
     }
   }
   public void mousePressed(MouseEvent evt) {
        // do nothing
   public void mouseReleased(MouseEvent evt) {
        // do nothing
    }
   public void mouseEntered(MouseEvent evt) {
        // do nothing
    public void mouseExited(MouseEvent evt) {
        // do nothing
    private void doColorBasedAction(Color input)
     if(palette.getShape() == RoughDraftShape.PENCIL)
          palette.reColorShape(input);
     else if(palette.getShape() == RoughDraftShape.HIGHLAY)
          palette.highlightLayer(input);
}
```

## REFERENCES

- [1] Cantrell, G. and Dampier, D. D. 2004. Experiments in hiding data inside the file structure of common office documents: a steganography application. In Proceedings of the 2004 international Symposium on information and Communication Technologies (Las Vegas, Nevada, June 16 - 18, 2004). ACM International Conference Proceeding Series, vol. 90. Trinity College Dublin, 146-151.
- [2] Francia, G. A. and Gomez, T. S. 2006. Steganography obliterator: an attack on the least significant bits. In *Proceedings of the 3rd Annual Conference on Information Security Curriculum Development* (Kennesaw, Georgia, September 22 - 23, 2006). InfoSecCD '06. ACM, New York, NY, 85-91. DOI= http://doi.acm.org/10.1145/1231047.1231066
- [3] Fridrich, J. 2006. Minimizing the embedding impact in steganography. In *Proceedings of the 8th Workshop on Multimedia and Security* (Geneva, Switzerland, September 26 27, 2006). MM&Sec '06. ACM, New York, NY, 2-10. DOI= http://doi.acm.org/10.1145/1161366.1161369
- [4] Fridrich, J., Pevný, T., and Kodovský, J. 2007. Statistically undetectable jpeg steganography: dead ends challenges, and opportunities. In *Proceedings of* the 9th Workshop on Multimedia & Security (Dallas, Texas, USA, September 20 -21, 2007). MM&Sec '07. ACM, New York, NY, 3-14. DOI= http://doi.acm.org/10.1145/1288869.1288872
- [5] Mei-Ching Chen; Agaian, S. S.; Chen, C. L. P., "Generalized collage steganography on images," *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, vol., no., pp.1043-1047, 12-15 Oct. 2008
- [6] Oplatkova, Z.; Holoska, J.; Zelinka, I.; Senkerik, R., "Detection of Steganography Inserted by OutGuess and Steghide by Means of Neural Networks," *Modelling & Simulation*, 2009. AMS '09. Third Asia International Conference on, vol., no., pp.7-12, 25-29 May 2009.
- [7] http://www.scienceinafrica.co.za/2005/november/microdot.htm

VITA

Blake Wayne Ford was born in Houston, Texas, on May 14, 1985, the son

of Melinda Ann Baros and David Wayne Ford. After completing his work at

Brazos High School, Wallis, Texas, in 2003, he received a Bachelor of Science

degree from Sam Houston State University in December 2006. During the

following years he was employed by National Instruments in Austin, Texas.

Concurrently, in January 2007 he entered the Graduate College of Texas State

University-San Marcos.

Permanent Address: 9226 Jollyville Rd. #140

Austin, Texas 78759

This thesis was typed by Blake W. Ford.