Non-Specialized File Format Extension

Blake W. Ford *blake.wford@gmail.com*
Department of Computer Science
Texas State University – San Marcos
San Marcos, 78666, USA

Khosrow Kaikhah *kk02@txstate.edu*
Department of Computer Science
Texas State University – San Marcos
San Marcos, 78666, USA

Abstract

The study expands upon previous work in format extension. The initial research purposed 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. The file format extension algorithm has been rewritten entirely in C++ and is now being distributed as an open source C/C++ static library, roughdraftlib. The files from our previous research are essentially binary compatible though a few extra fields have been added for developer convenience. The new data represents the current and oldest compatible versions of the binary and values representing the scaling ratio of the image. These new fields are statically included in every file and take only a few bytes to encode, so they have a trivial effect on the overall encoding density.

Keywords: Steganography, CAD, Metadata, Compatibility.

1. BACKGROUND

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. To resolve this issue we created a single sourcing steganography library called roughdraftlib.

This library integrates high-level design data into a standardized file format while maintaining backwards compatibility. Steganography is used to create an outlet for adding additional hidden information to the standardized file. Using our software, it is possible to build a single source format that is convenient for clients and workable for developers.

2. PORTABLE NETWORK GRAPHICS

Language unifying the code and developing a new internal architecture have made it much easier to expand the usefulness of the product. The software now has a pluggable target interface which allows for greater target diversity than the previous architecture. The most important improvement over the previous system to date is the addition of the Portable Network Graphics format as a possible target made possible by these changes. The PNG files exported by roughdraftlib are of the same quality as the old 24-bit bitmap targets, but with a much smaller disk footprint. This format uses the DEFLATE algorithm discussed previously to compress the image data. However, the effective compression of the DEFLATE algorithm on the raster data for these images is far greater than it was on the vector data according to our findings. In one test case, the PNG file produced was 1% the size of the already compressed 256-color bitmap produced by the older

RoughDraft application. DEFLATE's underlying LZ77 compression algorithm works well with repetitive datasets and the wire frames we have been testing are highly repetitive. In addition, roughdraftlib uses a feature of the PNG file format to hide data within an image without having to manipulate the visible data.

The information within a PNG file is broken up into small sections known as chunks. Chunks can either be critical or ancillary. As a rule, all PNG decoders are required to process critical chunks, but opt in to processing ancillary chunks. If a decoder cannot process an ancillary chunk, it typically ignores that piece of data. In roughdraftlib, all of the CAD data is stored in an ancillary chunk instead of in the least significant bits of the image data. Though either method is possible to implement using the PNG format, this method was chosen because it does not restrict the amount of data that can be embedded within the image.

For these reasons, PNG is likely to be the most popular target of roughdraftlib. Noting this, we had to revisit our previous choices regarding the layout and the future roadmap for the technology. Because the PNG format is more flexible and so different from the bitmap files studied before, some of our optimizations may no longer make sense when propositioning this format to outside vendors, while other considerations from our previous research still hold true in the PNG environment. The goal of this research is to verify the effectiveness of our format design and reevaluate the old bitmap format specific decisions that may hold back the adoption of vector bitmaps.

3. FEATURE EVALUATION

One universal feature that adds value to all of our potential targets is secondary format compression. While this may provide software developers with some additional overhead, the consumer facing experience is vastly improved in every domain. There is no tradeoff to be made here in terms of developer efficiency and disk footprint, because ample developer resources exist to attack and understand the problem of DEFLATE compression. Libraries and documentation on the algorithm are available for a variety of programming languages. Stylistically, few changes are foreseen in regards to the basic process by which roughdraftlib processes data, embeds its payload and targets individual file formats though some areas will require change.

In our initial research, we discounted the use of off the shelf file types due to their comparatively large size. However, now that a target exists that allows for boundless secondary format size we have called into question the importance of an extremely small but limited secondary data format. For instance, in the original research project, we found that embedding a standard DXF file in place of our custom format limits the maximum number of shapes possible in bitmap targets to about one tenth their previous value. The importance of this seemly negative statistic can now be reevaluated, considering that bitmaps will likely not be the most popular target of this library. Therefore, we are not optimizing this use case and using DXF as the backend format for all targets in an effort to increase adoption.

4. QCAD INSTRUMENTATION

We extended an open source version of the QCAD drafting application. With these additions to the code base, it is now possible to export CAD drawings in any of the formats available in roughdraftlib, PNGs with a compressed DXF chunk, or raw text based DXF files natively supported by QCAD. Through this exercise, we have been able to gather important size and integration data from which we will base the future roadmap of roughdraftlib.

QCAD was chosen as the host platform for a variety of reasons. Not only is the QCAD application industry capable, it is also open source and relatively easy to extend for our purposes. In addition, the creators of the application have also produced a suite of tools to convert DXF files into standard image formats. We assume since these tools exist, QCAD users are dealing with the duplicated source issues we are trying to solve. For this reason, QCAD users may be interested in converting to our approach. Lastly, QCAD ships with a standard DXF part library. The library contains CAD drawings of common blocks used in industry, like doors and appliances. We used these blocks as our baseline to ensure that our technology supports both the features and performance expected by CAD designers.

Using our instrumented QCAD program, we were able to produce some convincing size numbers for migration to either the highly compressed vector bitmap format or the easy to use DXF chunk derivative. We start each experiment with a standard part from the shipping library. This part is exported into each possible format discussed above. Then a qualitative assessment is made regarding the resulting image's quality followed by some file size comparisons.

For now, because all of the resulting images are lossless, there are only three static buckets of qualitative assessment. Our DXF chunk images have the highest quality, because they represent each CAD image verbatim. Vector bitmaps lag behind, because it does not yet support all of the shapes possible in a given DXF drawing, and pure DXF files are considered last as they cannot be viewed using standard imaging programs.

There are two primary size comparisons being made from each exported file. The first is raw file size to compare the disk footprint of the bitmaps, PNGs, and DXF files produced. This gives developers some indication of how standardizing on the end product would affect their systems. The second metric is a comparison of the two competing secondary data formats. The size of the CAD data in the final product determines how useful bitmaps and other steganographically limited file extensions will be in different applications.

File Type	File Size
24-bit bitmap	921.00 KB
256-color bitmap	307.00 KB
Raw DXF	21.00 KB
DXF Chunk PNG	5.25 KB
Roughdraftlib PNG	2.50 KB

TABLE 1: Raw File Size

The referenced tables were generated using an appliance drawing that ships with QCAD as a sample DXF file. The numbers represent the real world performance of the different encoding methods. Table 1 shows how each variation compares in terms of file size. Bitmaps produced under either the previous or DXF extension mechanisms have identical disk size and are generally larger than the source DXF file. The PNGs differ in size and the roughdraftlib variant is significantly smaller. It should be noted that both PNG exports are smaller than the original DXF source file.

Looking at the data size and capacity statistics helps demonstrate our conflict of interests. Table 2 depicts the size of each secondary format and Table 3 displays the embedding capacity of all possible targets. Table 4 illustrates the compatibility matrix derived from the previous two tables.

File Type	Format Size
Raw DXF	21.00 KB
DXF Chunk	3.00 KB
Roughdraftlib	300 B

TABLE 2: Data Format Size

File Type	Capacity
DXF, PNG types	Unlimited KB
24-bit bitmap	77 KB
DXF Chunk 256 bmp	1 KB
Roughdraft 256 bmp	960 B

TABLE 3: Data Capacity

File Type	Raw DXF	DXF Chunk	RoughDraft
DXF, PNG types	OK	OK	OK
24-bit bitmaps	OK	OK	OK
DXF Chunk 256 bmp	NP	NP	OK
Roughdraft256 bmp	NP	NP	OK

TABLE 4: Aggregate Data

This support matrix is representative of most files tested from the QCAD sample library. The roughdraftlib data format is the only option that allows us to support all of the target configurations currently available for this use case. When using the compressed DXF chunk as the secondary data backend, the resulting file typically fits into all but the 256-color bitmap target's available embedding space. For more complex designs, it should be noted that while the designs tested worked with our 24-bitmap targets the overall capacity for that format was greatly reduced when the DXF chunk method was used; approximately one-tenth the capacity of the roughdraftlib representation in this test.

5. NEW VALUE EQUATIONS

We established a relationship between this growth in data size and the number of shapes possible to embed in a file. This helps to clarify our arguments in regards to each of the secondary formats. In our original research, we used a static equation to determine the size of a secondary format in bitmap files.

Original Algorithm
File Size $< 54(\text{Header}) + (\text{Shapes})*72$ (1)

Using this equation, we determined that 24-bit bitmap with the dimensions 500x500 should be able to store around 10,000 shapes. Using compression, we increased this number conservatively by 30% for a final total of approximately 13,000 shapes when using the roughdraftlib format.

Compressed Algorithm
File Size $< 54(\text{Header})+(\text{Shapes})*50$ (2)

Because DXF files scale in a slightly different fashion we need to derive a second equation for competitive analysis. First, using QCAD we noticed that an empty DXF file defaults to roughly 11KB in size. With each additional shape, the file grows by an average of 130 bytes. When compressed, we are observing about an 80% decrease in the empty file size plus a 40 byte size increase per shape. Using this information, we derived the following corresponding equations.

DXF Algorithm

File Size < 54+88K+(Shapes)*1.2K (3)

Compressed DXF Algorithm
File Size < 54+16.8 K+(Shapes)*320 (4)

Applying our overhead tax and shape penalty rules for the DXF chunk format, we estimate that designs 2,400 or few shapes will be possible for an equal sized 24-bit bitmap encoded using our previous research.

For 256-color bitmaps a similar equations can be derived. In our earlier research, we chose not to noticeably change any of the 256 colors from the bitmaps palette. Since then, we have updated our approach. In our latest design, the color palette is artificially limited to 16 colors. The remaining dictionary space is filled with compressed secondary data. This increases the data size available for embedding shapes from 320 to 960 bytes; this is reflected in the tables [1]-[4]. For this format, we no longer hide data in the least significant bits of the color palette, so the overhead for each shape goes down as well. The following equations represent our current strategy for 256-color bitmaps.

Original Algorithm
960 Bytes < (Shapes)*9(Raw Size) (5)

Compressed Algorithm
960 Bytes < (Shapes)*6(Raw Size) (6)

DXF Algorithm
960 Bytes < 11KB+(Shapes)*130 (7)

Compressed DXF Algorithm
960 Bytes < 2.1KB+(Shapes)*40 (8)

Using these equations, linear placement would yield 105 possible shapes, compression would increase this number to around 135 and the DXF algorithms would be impossible.

With these results, we either have to consider either dropping the DXF chunk representation for 256-color bitmaps or improving our embedding technique. As there are two other file format choices, losing this target to gain the flexibility of the DXF chunk type does not seem like an unreasonable tradeoff, however, we also explored ways to more efficiently embed into 256-color targets. Our most recent approach involves encoding data into the bit field by duplicating the restricted color dictionary.

256-color bitmaps allow identical colors to be defined multiple times in their dictionary. Using this feature, we would expand our current dictionary usage from 16 colors to 32 by redefining the original set. This would limit free space in the dictionary from 960 bytes to 896, an initial loss of 64 bytes. However, this would allow us to assign logical 0s and 1s to the data in the image’s bit field. If a color reference in the bit field pointed to a value from the first set of 16 colors, it would indicate a 0 in the secondary file format. Likewise, a color pointing to a value in the second set would represent a 1. With this mechanism, we would recover the initial loss of 64 bytes in 512 pixels. If the area of the image is larger than 512 pixels, this method would allow for more encoding space than the previous version. The new algorithm would yield 32KB worth of encoding space from an image with our baseline dimensions of 500x500 pixels.

Original Algorithm
File Size < 54+32(Dict.)+(Shapes-100)*72 (9)

Compressed Algorithm
File Size < 54+32(Dict.)+(Shapes-100)*50 (10)

DXF Algorithm
File Size < 54+32+87K+(Shapes)*1.2K (11)

Compressed DXF Algorithm
File Size < 54+32+15.8K+(Shapes)* 320 (12)

6. SOFTWARE STACK COMPARISON

Taking into consideration all of this information, we intend to push our current software stack in a slightly new direction. Figure 1 shows the stack as it currently exists with the QCAD application.

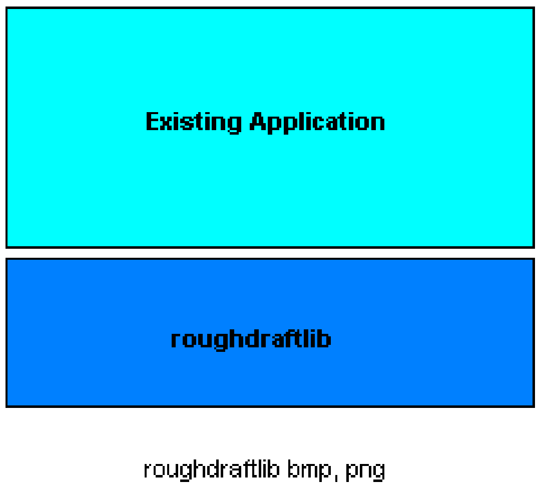

FIGURE 1: Stack for QCAD application

We copied and slightly modified the critical source from roughdraftlib in order to test the new standardized file approach to the single source problem. The resulting code is structured according to Figure 2. This code base has been name-spaced in order to allow two similar code paths to exist side by side for comparison. Currently the new DXF based single source utility is dependent upon the existing applications ability to produce one of the three carrier file types supported by our research.

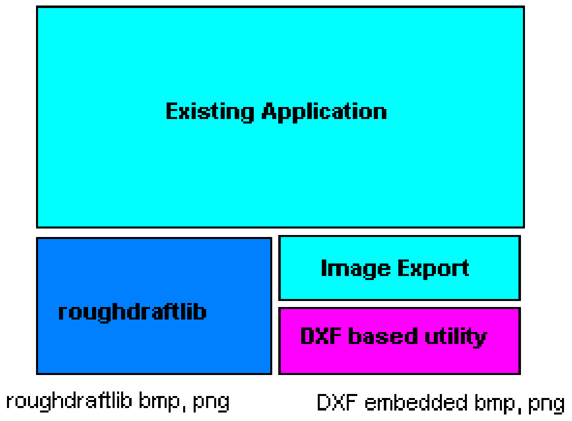

FIGURE 2: New Standardized File

We propose a new independent library derived from the embedding portion of the roughdraftlib code base. Instead of duplicating this code as we are now, roughdraftlib will become dependent upon the lower level-embedding library as depicted in Figure 3.

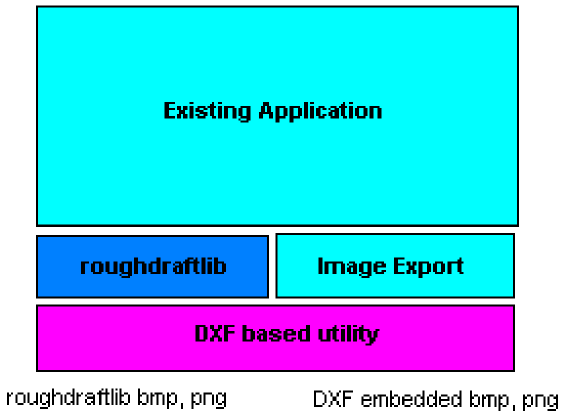

FIGURE 3: New Independent File

By moving the steganography code into a new independent layer, we give developers the option of using their own image file creation routines and standardized formats while preserving the integrity of our industry optimized approach. This will provide more flexibility when both adopting and continuing to add support for new features related to our research. With this software stack, existing applications can plug in with almost no effort and grow into our optimized format at their leisure. In addition, other industries can also take advantage of single sourcing immediately without the overhead of defining a fully optimized single source version.

7. CONCLUSIONS AND FUTURE GOALS

Following the previously mentioned code restructuring, we would like to target additional file types. The next likely candidate would be the JPEG file format. This format is a popular web format like PNG and can also be used in steganography. While the JPEG format can out perform PNG compression on certain images types, it is relatively ineffective when compressing diagrams, graphs, text, icons, and monochrome images. CAD images exhibit many of these qualities and we have observed that the size of the JPEG images we could produce through roughdraftlib would be around 500% larger than the comparable PNG version. Though this seems like a very large difference, keep in mind a JPEG image produced from roughdraftlib would still be drastically smaller than either of the bitmap files we currently export.

In this research iteration, we also leverage the work of an open source application to define and improve the usability of our product. This is a tread that will be repeated in the future of this product as well. One project of interest is Steghide source integration. Steghide is an open source steganography application that supports some of the formats targeted by roughdraftlib. Steghide has many interesting features that roughdraftlib does not like embedding capacity reporting and encryption and so long as we keep the appropriate licensing terms we can also take advantage of those features.

8. REFERENCES

[1] B. W. Ford and K, Kaikhah. “File Format Extension Through Steganography,” presented at the *International Conference on Software Engineering, Management & Application*, Kathmandu, Nepal, 2010.

[2] B. W. Ford and K, Kaikhah. “Honing File Format Extension Through Steganography,” presented at the I*nternational Conference on Infocomm Technologies in Competitive Strategies*, Singapore, 2010.

[3] G. Cantrell and D. D. Dampier. “Experiments in hiding data inside the file structure of common office documents: a steganography application.” *In Proceedings of the International Symposium on information and Communication Technologies*, 2004, pp. 146-151.

[4] G.A. Francia and T. S. Gomez. “Steganography obliterator: an attack on the least significant bits.” *In Proceedings of the 3rd Annual Conference on Information Security Curriculum Development*, 2006, pp. 85-91.

[5] J. Fridrich. “Minimizing the embedding impact in steganography.” In Proceedings of the 8th Workshop on Multimedia and Security, 2006, pp. 2-10.

[6] J. Fridrich, T. Pevný, and J. Kodovský. “Statistically undetectable jpeg steganography: dead ends challenges, and opportunities.” *In Proceedings of the 9th Workshop on Multimedia and Security*, 2007, pp. 3-14.

[7] C. M.C. Chen, S. S. Agaian, and C. L. P. Chen. “Generalized collage steganography on images." *In* Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 2008, pp. 1043-1047.

[8] Z. Oplatkova, J. Holoska, I. Zelinka, and R. Senkerik. “Detection of Steganography Inserted by OutGuess and Steghide by Means of Neural Networks.” *In Proceedings of the Third Asia International Conference on Modeling and Simulation*, 2009, pp. 25-29.