# A Hybrid Data Compression Scheme for Improved VNC

**Xiaozheng (Jane) Zhang and Hirofumi Takahashi**
**Department of Electrical Engineering**
**California Polytechnic State University**
**San Luis Obispo, CA 93401, USA**

## ABSTRACT

Virtual Network Computing (VNC) has emerged as a promising technology in distributed computing environment since its invention in the late nineties. Successful application of VNC requires rapid data transfer from one machine to another over a TCP/IP network connection. However transfer of screen data consumes much network bandwidth and current data encoding schemes for VNC are far from being ideal. This paper seeks to improve screen data compression techniques to enable VNC over slow connections and present a reasonable speed and image quality.

In this paper, a hybrid technique is proposed for improving coding efficiency. The algorithm first divides a screen image into pre-defined regions and applies encoding schemes to each area according to the region characteristics. Second, correlation of screen data in consecutive frames is exploited where multiple occurrences of similar image contents are detected. The improved results are demonstrated in a dynamic environment with various screen image types and desktop manipulation.

**Keywords**: Image manipulation and compression, and Virtual Network Computing.

## 1. INTRODUCTION

VNC [1] stands for Virtual Network Computing. It is a client/server software application that allows one to remotely access a desktop environment from anywhere on the internet. VNC has been gaining increased attention in recent years because it provides great flexibility and convenience in a mobile computing environment. For example, one can use VNC to access his personal Unix or PC desktop from any office on campus and from around the world. A system administrator can use VNC to help troubleshoot a remote computer, or to access and administer server machines without leaving his own computer. There exist many programs that provide one with the ability to view the screen on a remote PC. But VNC, besides being free, is small and simple and has the advantage of being fully cross-platform. Numerous groups [3-6] are currently actively developing VNC and related products.

Successful application of VNC requires rapid data transfer from one machine to another over a TCP/IP network connection. However transfer of screen data consumes much network bandwidth and current data encoding schemes for VNC are far from being ideal. When VNC is used over the internet, via a public network, security reasons require the use of encryption, which further exacerbates the bandwidth problem. Finally, mobile computing use of VNC with devices such as Palm Pilot would have to overcome the limitation of a slow network as well as the additional overhead of encryption over the internet. There is a high demand for a bandwidth optimized version of VNC.

The original VNC was developed by Olivetti Research Labs in Cambridge in 1998 [1-3]. The technology underlying VNC is a protocol called RFB that stands for Remote Frame Buffer. It supports six different encoding algorithms where all use lossless compression. The compression algorithm used in the original VNC software is very inefficient especially in compressing natural images and suffers from low performance in low-bandwidth network environments.

TightVNC [4, 7] developed by Kaplinsky is an improved version of the original VNC. Besides the various enhanced features it offers, it implemented a new encoding called "tight". TightVNC incorporates an additional data analyzer to determine statistical properties of pixel data before applying data filters. While TightVNC was shown to outperform standard VNC encoders in compression ratios, the design of the data analyzer is still primitive and leaves much room for improvement.

The objective of the proposed work is to develop improved screen data compression techniques to enable VNC over slow connections and present a reasonable speed and image quality. The paper is organized as follows. In Section 2, segmentation-based image coding is first described. We first review previous work in this area, we then present our algorithm that uses different encoding schemes to address varying region characteristics in a screen image, followed by experimental results. Section 3 presents our novel approach in coding similar image contents appeared in earlier frames using memory buffer. Finally, Section 4 offers our conclusions and future research direction.

## 2. SEGMENTATION-BASED IMAGE CODING

Computer screen image represents a class of compound image that consists of background, text, graphics, and natural image contents. The basic idea in improving coding efficiency of a compound image is based on the observation that screen image is composed of several distinct regions and no coding technique alone is able to efficiently handle all range of data. By allowing the compression algorithm being content adaptive, different image classes are better encoded using region specific characteristics.

### 2.1. Previous Work

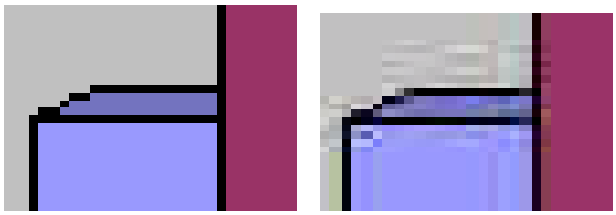
In general, there exist two main approaches for representing a compound image. One is based on MRC multilayer model [8] that decomposes an image into background, foreground, and mask layers. Each layer is then coded as an image independently from other layers. Among various applications are DjVu coder [9] and SPEC coder [10].

On the other hand, block-based segmentation classifies non-overlapping blocks of an image into different classes and

compresses each class differently according to its characteristics, such as in [11] where the method was applied to scanned documents.

### 2.2. Proposed Algorithm

We employ block-based segmentation since it provides low complexity which is crucial for VNC application. As opposed to conventional method where image blocks are classified into text, graphics, picture, etc., we tie the segmentation directly to the compression process. We consider JPEG as our baseline coding because it is a well-established digital image compression standard. While JPEG is very efficient in compressing color-rich images, it introduces noticeable artifacts when it is applied to text and graphic images. One such example is shown in Figure 1, where distortion around edges can be easily observed in a JPEG encoded image with graphic content. Various psychophysical studies [12] also suggest that human eyes are more sensitive to distortion around edges thus higher perceptual importance should be given to the text and edge regions, followed by the smooth and detailed regions. We therefore choose to use lossless compression to encode image blocks with text and graphic contents that are normally characterized by fewer color numbers and leave only color-rich images to JPEG.



(a) (b)

**Figure 1:** An example of artifacts introduced by JPEG on a graphic image. (a) Original image (24 bpp), (b) JPEG reconstructed image (4.946 bpp).

**2.2.1. Lossless Coding:** For lossless encoding of an image block with a color number of eight or less, only $N$ ($\leq 8$) color data (requiring $3*N$ bytes for an RGB color pixel) plus a bitmap are transmitted. The decoder is able to reconstruct the original image from these data. The bitmap requires only 1 bit for each entry when $N$ is 2, 2 bits if $N$ is 3 or 4, and 3 bits if $N$ is from 5 to 8. Note that one-color image does not require any bitmap.

**2.2.2. Lossy Coding:** If the color number in a 16x16 block is more than 30 (determined experimentally), the image will be encoded using JPEG. For images in between, we choose Block Truncation Coding for images with strong edges, while JPEG for the ones with weak edges. BTC (Block Truncation Coding) [13] is a lossy compression technique. The advantage of BTC is that it is relatively easy to implement but still preserves strong edges in an image. We adopted a modified version of BTC [14], where only one bitmap resulting from the sum of three color components is encoded. Figure 2 shows the performance of the improved BTC compared with JPEG and the original BTC in one test image.

**2.2.3. Complete Segmentation Scheme:** Figure 3 shows the flowchart of our segmentation process. Here we employ a simple color counting in combination with edge detection. Since each pixel has three components (R, G, B), we combine them using the following formula: pixel value = 1000R+G+0.001B. This is used for counting color numbers. In refinement 1, the algorithm checks all 4x4 sub-blocks inside the 16x16 block. If there is any sub-block having more than two colors, the image block is sent to JPEG; otherwise, the BTC is used. In refinement 2, two edge detection techniques are employed. In the first edge detection, magnitudes of the difference between adjacent pixels vertically and horizontally are computed for each pixel in the image block. If sum of the magnitudes is less than a threshold (T1=1500), the image is sent to BTC with a block size 4. Otherwise the block goes through the second edge detection, where the magnitude of the difference between adjacent pixels are computed for top, bottom, left and right sides of an image block. If the maximum value is less than a threshold (T2=10), the block will be compressed by BTC with a larger block size of 8, otherwise, the block size of 4 is chosen. This classification is helpful in better coding strong edges appearing on sides of the block. It should be mentioned here that even though we use PSNR as an objective measure in our final evaluation for reconstructed image quality, the segmentation method is developed by comparing compression results among various schemes for test images based on combining PSNR with visual inspection. For similar PSNR values, we choose the one with better visual results.

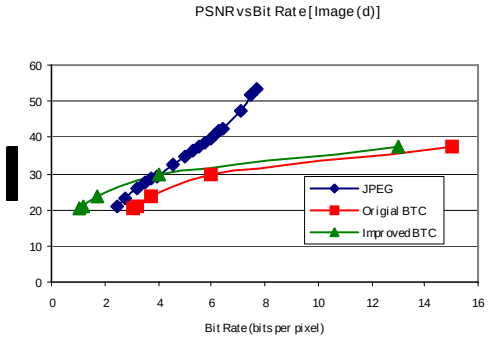



**Figure 2**: Comparison between JPEG, original BTC, and improved BTC using one test image.

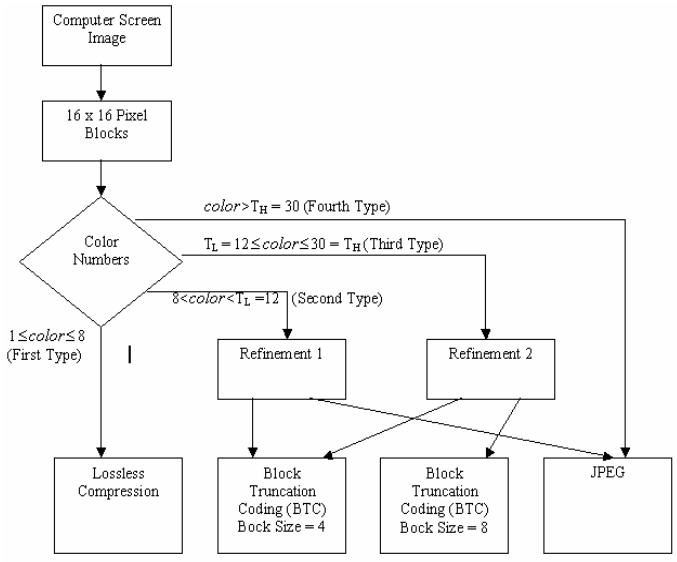



**Figure 3**: Flowchart of the Segmentation.

### 2.3. Experimental Results

The performance of the presented algorithm is evaluated and compared with JPEG on five test images with a size of 560*560. These images are chosen to cover a variety of screen image types. Summarized results in Table 1 show that in all cases except for image 2, the new algorithm produces much better results in terms of PSNR and bit rate. The new algorithm requires low bit rate while maintaining high PSNR. It should be mentioned that since we employ lossless coding and BTC for regions with strong edges, noticeable improvements are observed in those areas comparing to JPEG even though both might have similar PSNR values.

| | JPEG | | The New Algorithm | |
|---|---|---|---|---|
| | Bit Rate (bpp) | PSNR (dB) | Bit Rate (bpp) | PSNR (dB) |
| Image 1 | 4.25 | 31.37 | 3.01 | 39.25 |
| Image 2 | 4.81 | 31.04 | 4.73 | 31.85 |
| Image 3 | 3.45 | 34.03 | 1.87 | 33.59 |
| Image 4 | 4.89 | 31.32 | 3.57 | 33.34 |

**Table 1:** Performance Comparison between JPEG and the New Algorithm**.**

## 3. CODING OF SIMILAR IMAGE INSTANCES

In addition to the improvement on coding a single still image, the hybrid algorithm also exploits temporal correlation of screen data in consecutive frames. It is noted that the existing VNC detects updated region between two adjacent frames and transmits only those areas. This significantly reduces temporal redundancy in screen data. However VNC does not address the coding of similar image content in a series of image frames that occur frequently during a typical manipulation of desktop environments. For example, it is often observed that one might scroll up and down the contents in a window, or bring up and hide/cover a window from time to time. The background can also be covered and uncovered over and over. The same or similar image contents appear and re-appear in those situations. Figures 4-6 show a sequence of three partial screen images. As can be seen, during the process of bringing up a new window from the Windows Taskbar (located in lower portion of the screen, with the Auto-hide property enabled), multiple occurrence of the Taskbar content is detected in Figure 4 and 6. Hence, subsequent occurrence of the same Taskbar would be best encoded by reference to its first occurrence.

To implement the idea, we create a memory buffer covering up to 250 previously encoded image blocks at both encoder and decoder. It is critical to maintain the same copy at both sides. Each time a new image block of a certain minimum size is encoded, we first search through the memory buffer. If a good match is found, only the image id is transmitted. This idea is motivated by work [15] on long-term memory prediction, however with the difference that no motion compensation is performed here.

In our implementation, we first add a memory prediction encoding and its associated message to the RFB protocol. The message includes frame id, image id, and mode of action that could be either record or playback. The routine is then extended to first ask Memory Buffer for a match to the *rectangle* we are trying to encode. In doing so, we first build a candidate bag including all possible matching rectangles from the Memory Buffer. The most likely candidate is the one that has the same location and size as the *rectangle* to be encoded. This is characterized by a four-edge matching. Following that is three-edge, two-edge, and one-edge matching. The candidate bag is then searched for matches by comparing pixel values between the *rectangle* and the candidate in the bag. If no match is found, the image will be encoded using the original method, and the action is set to record. Both encoder and decoder will save this *rectangle* in their Memory Buffer. If a match is found, the mode of action is set to playback and the frame id and image id of the matched rectangle is transmitted. The decoder will use the id info to retrieve the image stored in its Memory Buffer. Most matches are not perfect, such as the Taskbar in Figures 4 and 6 – the clock in Figure 6 is updated. The algorithm can declare a match so long as the *rectangle* "almost" (80% at the moment) matches the image in Memory Buffer. In this case we recursively encode all the rectangles constituting the mismatched regions, however, at some point the overhead of recording and recalling such small rectangles is more than any realized savings. For this reason, the algorithm does not remember any image smaller than the `MINRECTSIZE` (currently 900, such as a 30x30 image block).


**Figure 4**: First Image in a Sequence with Taskbar

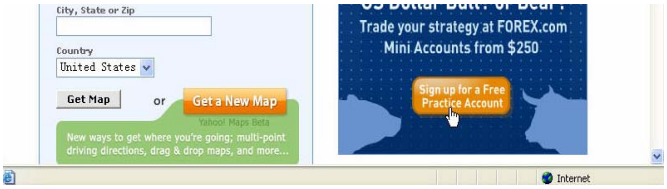



**Figure 5**: Second Image in a Sequence without Taskbar


**Figure 6**: Third Image in a Sequence with Taskbar

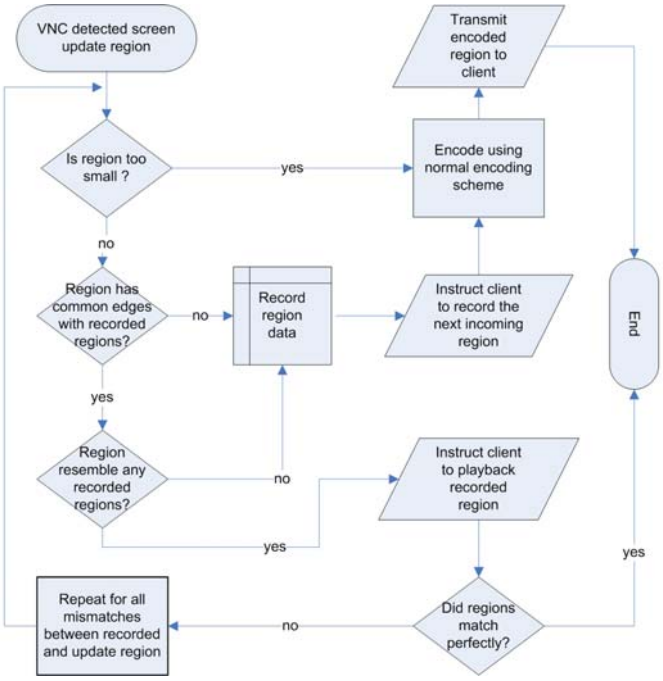



**Figure 7**: Flowchart of the Memory Coding.

## 4. CONCLUSIONS AND FUTURE WORK

In this paper we present a hybrid algorithm that improves the existing software package to enable VNC over slow connections and present a reasonable speed and image quality. The improvement was made in two areas. While the segmentation-based image coding has been exploited in previous research efforts, the idea and implementation of the coding of multiple occurrences of similar image contents is novel and has not been reported elsewhere. In this approach, the improvement was made in exploiting correlation of screen data in consecutive frames. It is often observed that during manipulation of a desktop environment, the same image content might appear from time to time, or at varying locations. Based on that, we encode multiple occurrences of the similar image block by referring to its first occurrence. This leads to significant savings in bandwidth.

The developed algorithm has been tested on several desktop images and during manipulation of desktop environments. Improved image quality and faster transmission speed has been observed visually. However, to objectively evaluate the new algorithm, further work is required. First incorporation of a record/playback mechanism in the original software is necessary so that each VNC session can be reproduced and tested on a variety of algorithms. Furthermore, performance measures need to be built into the original package to allow objective evaluation of our hybrid algorithm with existing techniques. It is our hope that this will justify the usage of our newly developed algorithm and lead to a wider deployment of VNC in the field of mobile computing environment.

## 5. ACKNOWLEDGEMENT

This work was sponsored by the *Department of the Navy, Office of Naval Research*, under Award # N00014-04-1-0436.

## 6. REFERENCES

[1] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual Network Computing," **IEEE Internet Computing**, Volume 2, Number 1, January/February, 1998.

[2] T. Richardson, **The RFB Protocol**, *RealVNC Ltd*, August 2003.

[3] http://www.realvnc.com

[4] http://www.tightvnc.com

[5] http://ultravnc.sourceforge.net

[6] http://www.tridiavnc.com

[7] K. V. Kaplinsky, "VNC tight encoder – data compression for VNC", Modern Techniques and Technology, 2001. **Proceedings of the 7th International Scientific and Practical Conference of Students, Post-graduates and Young Scientists**, February/March 2001.

[8] **Mixed Raster Content ITU-T Study Group 8**, Draft Recommendation T.44, March 1997.

[9] L. Bottou et al., "High Quality Document Image Compression Using DjVu", **Journal of Electronic Imaging**, Vol. 7, July 1998.

[10] T. Lin, and P. Hao, "Compound Image Compression for Real-time Computer Screen Image Transmission", **IEEE Transaction on Image Processing**, Vol. 14, Issue 8, August 2005.

[11]X. Li, and S. Lei: Block-based segmentation and adaptive coding for visually lossless compression of scanned documents. **In Proceeding of International Conference in Image Processing**, Volume 3, 2001.

[12] D. Marr, **Vision**, W. H. Freeman and Company, 1982

[13] O. Mitchell, E. Delp, and S. Calton, *Block Truncation Coding: A New Approach to Image Compression*, **IEEE International Conference on Communication**, IEEE Press, Piscataway, 1978.

[14] Y. Wu, and D. Coll, "Single Bit-Map Block Truncation Coding of Color Images", **IEEE J. Selected Areas Communication**, CA, 1995.

[15] T. Wiegand, X. Zhang, and B. Girod, Long-Term Memory Motion-Compensated Prediction, In **IEEE Transaction on Circuit and Systems for Video Technology**, February 1999.