LOSELESS IMAGE COMPRESSION FOR REMOTE DESKTOP SYSTEMS

V.V. Pronchev, A.V. Tuzikov

United Institute of Informatics Problems of the National Academy of Sciences of Belarus, 6 Surganava Street, Minsk 220012, Belarus, e-mails: *{pronchev,tuzikov}@mpen.bas-net.by*

Abstract. Modern cell phones are used in a wide variety of applications. This paper presents a lossless image compression algorithm for remote desktop systems. The algorithm is simple, efficient and suitable for using in cell phones. It uses two particular features of images generated by most applications running in windows environment: a large amount of rectangular areas with the same color and relatively a small amount of colors used at the same time.

1. Introduction

With the remote desktop software one can remotely view a computer desktop and interact with it over the Internet, using any kind of connection (modem, wireless, mobile phone). It is possible to run applications on the remote PC and do any other tasks like editing documents, managing emails, etc.

There are two main approaches in handling and transferring visual information in the remote desktop systems.

A good example of the first approach is RDP (remote desktop protocol) implemented by Microsoft in Terminal Services software. Only parameters of GDI (graphics device interface) function calls are passed to a terminal server client. The client displays primitives using these parameters. A number of techniques such as font cashing and bitmap cashing are used to increase performance. It is hard to implement RDP client on non-Windows based devices such as mobile phones.

A more straightforward approach is to generate the final screen image on a server and to pass it to the client using some sort of compression.

Modern cell phones are used in a wide variety of applications. Growing computational and communication capabilities of mobile devices made it possible to use them as MP3 and Video player, Web Browser, Infrared remote control, etc. The aim of this work is to make it possible to control remote PC over cell phone. Our development currently is based on Symbian Series 60 [5] platform (cell phones Nokia 7650/3650/3660/6600, Siemens SX1, Samsung SGH-D700). Devices based on Symbian operating system and Series 60 user interface are currently the most popular and best selling smartphones with several million units already on the market. There are a number of features which distinguish mobile devices from PC-based clients:

- Small spatial and color resolution of displays (176x208, 4096 colors).
- Very low network bandwidth (one can expect 1-4 Kbytes per second or less).
- High network latency.
- Relatively small amount of RAM on mobile devices (4MB for Nokia 7650/3650).
- Possible lack of a pointing device (only a few of currently available models have a touch-screen).

Taking into account these differences one can expect the following list of features to be implemented in a remote desktop system:

- Remove any kind of redundancy in transferring data; send color information in mobile client format (12 bit/pixel for Nokia 3650/7650).
- Efficient lossless image compression algorithm.
- Maximum simplicity of client software.
- Flexible server-side control over traffic bandwidth and frame rate.
- An ability to work in “zero traffic” mode if there is no changes on the computer screen. It will reduce both network traffic and power consumption.
- Combining mouse movement operation with scrolling of visible area. Using joystick for mouse control.
- Transferring and storing in the client memory a relatively small window rather than the entire frame. The window includes a visible area and small margins for scrolling without network latency. This will enable managing arbitrary large frame resolutions.
- PC keyboard emulation.

2. Image compression

Limited bandwidth and expensive internet traffic in modern GSM networks are one of the main problems. This paper presents a lossless image compression algorithm for remote desktop systems. The algorithm is simple, efficient and suitable for using in cell phones. It uses two particular features of images generated by most applications running in windows environment: a large amount of rectangular areas with the same color and a relatively small amount of colors used at the same time.

The algorithm consists of 2 passes. At the first pass it searches for rectangles of maximum area and encodes their positions, size and color. The next step is to encode colors of remaining pixels. The algorithm employs the adaptive arithmetic encoding [2, 3] for compression of position and size values. The "move to front transform" [4] is used in conjunction with the adaptive arithmetic encoding for color values processing.

Pixels are processed in raster order, left to right starting from the upper line.

Pass 1:

step 1:

- all pixels are considered to be "not encoded"
- D=0 (distance)

step 2:

- get next pixel (C – pixel color), if no pixels left go to step 3
- if the pixel is already "encoded" go to step 2
- search for a rectangle of maximum area filled with color C with upper left corner in the current position (S – area in pixels; X – width, Y – height)
- if $S < S_{min}$ we do not encode the rectangle, D = D + 1 and go to step 2
- set X = X - 1, Y = Y - 1
- evaluate limits X_{max} and Y_{max} for X and Y respectively based on frame boundaries
- encode D using an adaptive arithmetic encoder *(see Paragraph 4)*
- encode X, Y using a modified adaptive arithmetic encoder and limits X_{max}, Y_{max}
- encode C using the "move to front transform" in conjunction with the adaptive arithmetic encoding *(see Paragraph 3)*
- mark all pixels of the rectangle as "encoded"
- D = 0
- go to step 2

step 3:

- encode D

Pass 2:

For each "not encoded" pixel:

- encode its color C using the "move to front transform" in conjunction with the adaptive arithmetic encoding

In our experiments the value S_{min} = 9 provided the best compression ratio.

Encoding stages of the algorithm are shown in Figures 1 and 2. Figure 1 illustrates the ability of the decoder to produce a partially decompressed image based on rectangles encoded at Pass 1. The right picture on Figure 2 represents individual pixels encoded at Pass 2. For both of these pictures remaining pixels are filled with grey color. The left image on Figure 2 shows rectangles encoded at Pass 1 in different colors. A large number of pixels are encoded as a part of corresponding rectangles while only a small amount is encoded individually. Arithmetic coding is computationally expensive. Images with a high compression ratio require less operations of arithmetic coding. This feature of the algorithm allows the decoder in a mobile device to perform faster then the data transfer rate in mobile networks, regardless of the image complexity.

3. Move to front

The basic idea of the "move to front" (MTF) scheme [4] is to maintain a list, which represents the symbols of an alphabet. Frequently used symbols are located near the front of the list. A symbol is encoded by the index of this symbol in the list. After that the symbol is moved to the front of the list. The MTF scheme is a typical representative of a List Update Algorithm (LUA) [1].

Example: *MTF(2,1,1,1,1,0,2,0,2) = (2,2,0,0,0,2,2,1,1)*

MTF transform allows encoding pixel color values in context of previously encoded values. MTF output represents a number of different color values encoded after the last occurrence of the current color.

Fig. 1. Original image (left) and partially decompressed image based on rectangles encoded at Pass 1 (right).

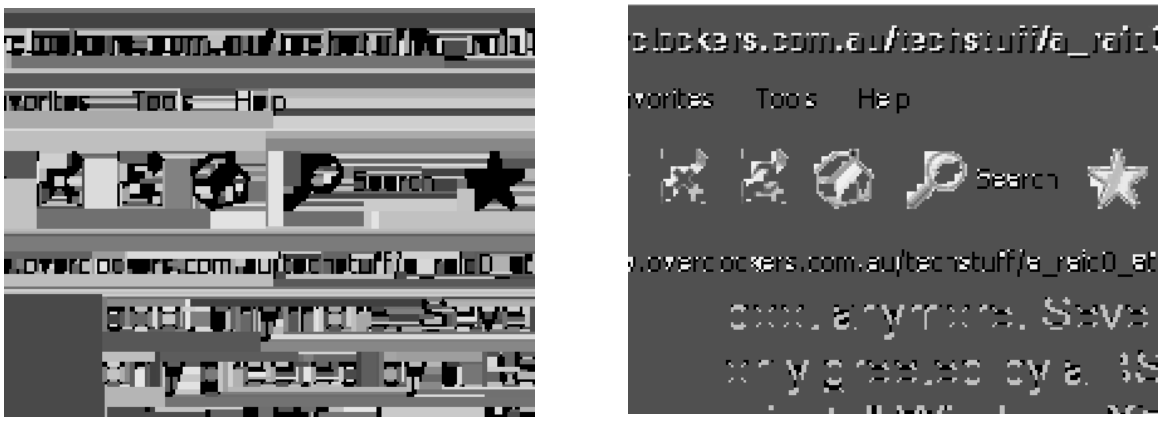

Fig. 2. Rectangles encoded at Pass 1 are colored randomly (left) and pixels encoded at Pass 2 (right).

Lower indexes in the MTF output get higher frequencies. Applying entropy coding to the MTF output produces better results than encoding color values themselves.

4. Adaptive Arithmetic coding

Arithmetic coding (AC) [2, 3] is a special kind of entropy coding. Unlike Huffman [2, 3] coding, arithmetic coding does not use a discrete number of bits for each symbol to compress. It reaches for every source almost the optimum compression in the sense of the Shannon theorem and is well suitable for adaptive models. The main idea behind arithmetic coding is to assign to each symbol an interval. Starting with the interval [0,1), each interval is divided in several subintervals, which sizes are proportional to the current probability of the corresponding symbols of the alphabet. The subinterval from the coded symbol is then taken as the interval for the next symbol. The result will be a binary representation of fractional value of any number in the specified interval.

In adaptive arithmetic coding, the probabilities assigned to symbols change as each symbol is coded. At the beginning frequency counters for all symbols are assigned to “1”. After encoding of each symbol the corresponding counter is increased. The increase amount affects the speed of adaptation. The decoder follows the same way.

Frequency table lookup is connected with large amount of memory access operations. Data to be encoded in this image compression algorithm have a nonuniform distribution. Lower absolute values will get higher frequencies. This is true for all 3 types of the data: position, size and color index. Both encoder and decoder significantly decrease lookup time by starting search from the lowest absolute value of the symbol.

A small modification was made to the generic algorithm to allow encoding a limited number of symbols starting from the first one. The limits used by the algorithm are known for both coder and decoder.

Frequencies of symbols beyond limit are assumed to be zero. This will eliminate redundancy in encoding parameters with variable range such as width and height.

5. Interframe compression

The algorithm effectively manages frame sequences. A special color value is used to represent colors to come from a previous frame. Only differences between frames are transmitted. No additional memory is required for storing the previous frame.

Displays of cell phones have a relatively small resolution and the most common operation is scrolling of visible area. Previous frame is shifted before using as a prediction for the next frame. The rectangle encoding technique perfectly suits for representing large empty rectangles generated as a result of scrolling operations.

Four different frequency tables are used in adaptive arithmetic encoder: for position, width, height and color index respectively. Probabilities accumulated at each frame are used in next frames. Frequency counters are divided by 2 after each frame to prevent counters from overflow and to increase the ability of the coder to adapt to different frequency distributions.

6. Performance

An image from Figure 1 was used to produce test results. The algorithm was run twice with the same image to show advantage of using symbol frequency distribution specific for this image over simple uniform distribution for arithmetic encoder. The algorithm achieved a compression ratio of 0.62 bit/pixel. The results were compared with a few popular compression programs and are summarized in Table 1.

Compression method	Compressed size, bytes	**Difference, %**
The algorithm, first run	2920	**0%**
The algorithm, second run with accumulated frequencies	**2676**	-8.4%
GIF	4792	**+65%**
RAR 3.0, default	3046	**+4.3%**
7-Zip 2.30, default	2964	**+1.5%**
PKZIP v4.00	3339	**+14.3%**
RAR 3.0 (-m5, maximum compression)	**2719**	**-6.9%**

Table 1. Compression ratio

We can point out the following advantages of the proposed algorithm:

- Simplicity of implementation (less than 300 lines of code in C++ for the decoder).
- Low memory requirements.
- An ability to display partially decompressed images. There are two levels of details: rectangles and individual pixels.
- The algorithm is well suited for handling image sequences produced as a result of scrolling operations.

The algorithm is implemented in a prototype remote desktop software system for cell phones Nokia 7650 / 3650. Performance of the decoder exceeds data bandwidth in cell phone networks. Server part works in Microsoft Windows environment.

References

[1]. S. Albers, M. Mitzenmacher, “Average Case Analyses of List Update Algorithms, with Applications to Data Compression”, *Algorithmica*, 21, 3, (1998), 312-329.

[2]. M. Ghanbari, "Video coding: an introduction to standard codecs”, *The Institution of Electrical Engineers, London, UK*, (1999).

[3]. D. Vatolin, A. Ratushniak, M. Smirnov, V. Yukin "Data compression methods" (in Russian), *Dialog MIFI*, (2002).

[4]. “Move To Front (MTF)”. http://www.data-compression.info/Algorithms/MTF/index.htm

[5]. “Technical specs of Symbian OS Series 60 devices”. http://my-symbian.com/7650/techdata.php