Application of Huffman Coding in Lossless Video Compression

Mahessa Ramadhana - 13511077
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
mahessaramadhana@itb.ac.id

Abstract—**Lossless video compression can reduce the file size of video while keeping the video exactly the same as the uncompressed video. This paper will explain the use of Huffman coding in lossless video compression using HuffYUV video codec.**

Index Terms—**compression, Huffman, video, lossless**

I. INTRODUCTION

Video is one media that people often use to record their important moments. Not only that, video is also widely used in the entertainment business, namely in television, theatres, and even in video games.

Decades ago, all videos are taken and stored in analog format. People use film reels to take and store movies. For home video, there are several different video tape formats available, such as Beta tape and VHS. Film reels work similar to single-shot camera films, only that they take sequence of images instead of single-shots, while video tapes work similar to cassettes, storing electrical information using magnetic tapes.

Nowadays, videos are also often taken and stored in digital format. Digital camera videos are now widely used, and videos can now be stored using digital media such as DVDs, Blu-rays, or even a computer hard disk.

The problem in digital videos, however, is the amount of storage they require. A minute of uncompressed standard definition video can take more than 1 gigabytes of storage. Imagine how much space it would take for a full-length movie, which usually take 90 minutes and more.

People then invented ways to reduce the file sizes of digital videos. These methods is calles video compression, and there are many kinds of them, each for differing uses.

II. THEORIES

A. Lossless and Lossy Compression

Compression can be roughly divided into two types: lossless and lossy compression. Each have their own strengths and weakness, one is better suited to certain uses than the other.

Lossless compression is a type of compression where there are no loss of data during the encoding process. This means that once you decode a compressed file, you will get exactly the same file as the original file. However, such compression method have relatively poor compression rate compared to lossy compression.

Lossy compression on the other hand, is a type of compression where some data are discarded during the encoding process to further reduce the file size. Better compression methods usually have priorities in which data to discard, with less important data having higher priority to be discarded. Such compression method can yield much smaller file size than lossless compression, but the decoded file will not be exactly the same as the source.

Which one is better depends on the situation. For compressing text files and programs, it is mandatory that there is absolutely no loss of data during the encoding process, otherwise, in the case of text files, some information may be rendered illegible, while for program files, the program may not work as it should be.

For music, pictures, and videos, lossy compression is the better option. Often, some information stored in music, pictures, and videos may be difficult, or even impossible, to be perceived by human senses. Such information may be discarded, which will make the decoded file different from the source, but to human senses, such difference may be hard to notice. Of course, there are cases where extreme compression can distort the music/picture/video badly that the difference starts to become clear.

B. Lossless Video Compression

While it's been mentioned that lossy compression is better suited for video, lossless compression also has its use for video compression. In video editing, a video file can go through many compression processes. Using lossy compression, every compression process will result in a loss of data. The more compression process the video underwent, the more data is lost. This is where lossless video comes in. Using lossless compression, one can keep the video file relatively small compared to uncompressed video while making sure nothing gets lost during the editing process. There are a few lossless video codecs available, one such codec is HuffYUV, which employs the Huffman coding method during the encoding process.

C. Basic Digital Video Theories

Digital videos are simply a sequence of digital images. A certain number of images are shown each second, which is called the frame rate. Each frame is constructed from a number of pixels. The number of horizontal pixel lines is called the vertical resolution, and the number of vertical pixel lines is called the horizontal resolution. Each pixel have their own color. The color of each pixel can be represented using several different colorspace, e.g. RGB, CMYK, and YUV. Video generally use YUV colorspace, which represents color using three 8-bit integers based on how human perceive color. These 8-bit integers (called channels, or color planes from now on) are the Y channel, which represents luminance (black and white color), U channel which represents blue chrominance (blue and yellow color), and V channel which represents red chrominance (red and green color).

Since videos are basically a sequence of images, the compression technique for images can be used to compress video. In HuffYUV's case, the compression technique is the same as Lossless JPEG, so in this paper we will use the information from how Lossless JPEG works.

D. Huffman Coding

Huffman coding is a technique that can be used to reduce the space required to store files. Huffman coding is a based on statistical coding, which means the more frequent a symbol occurs, the shorter its bit-representation will be. In other words, Huffman coding uses variable-length coding system as opposed to the standard fixed-length coding system. Fixed-length coding system uses the same length of bit-representation for all symbols, while variable-length coding system will use shorter representations for more frequent symbols, and longer representations for less frequent symbols, which will result in a reduction of the total length of bit-representation of data.

To assign bits to each symbol in, for example, a string, we must build a binary tree follow these steps:
1. Count the frequency of each characters.
2. Make a forest trees. All trees are one node, with the weight of a tree equal to the weight of the character in the node.
3. Choose two trees with lowest weight, call these trees T1 and T2. Make a new tree whose weight equals T1+T2 and whose left sub-tree is T1 and right sub-tree is T2.
4. Repeat step 3 until there is only one tree. This tree is the optimal encoding tree.
5. Assign bits to each symbol by traversing from the root until you find the symbol. Assign '0' to left sub-trees and '1' to right sub-trees.

For example, we will make a Huffman encoding tree from the string 'go go gophers'. First, we count the frequency of each characters.

character	frequency
'g'	3
'o'	3
' '	2
'p'	1
'h'	1
'e'	1
'r'	1
's'	1

Table 2.1 Frequency table

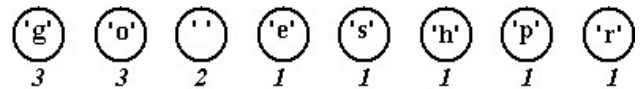Then, we make a forest of trees from the table.

Figure 2.1 Forest of one-node trees.

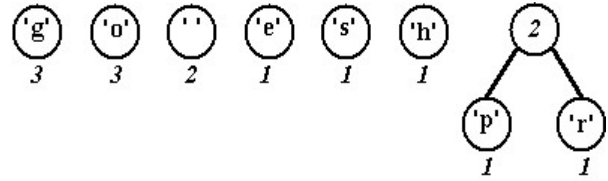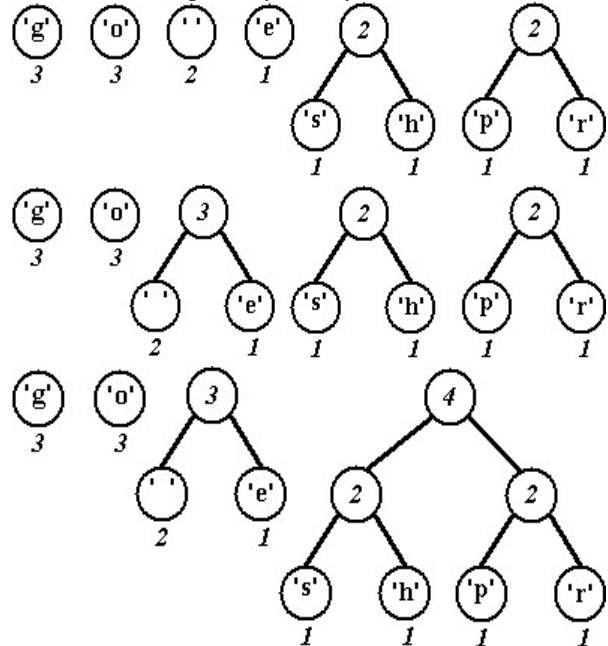Pick two nodes with the least weight, and make a new tree with weight equals the sum of the two trees.

Figure 2.2 First step.

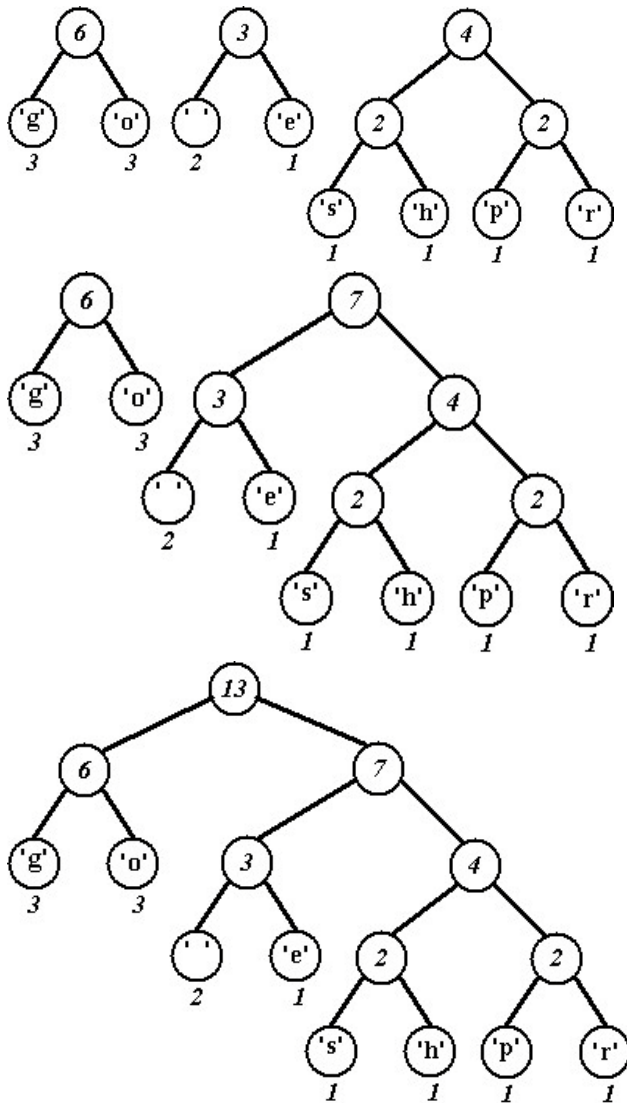Continue this step until you only have one tree.

Figure 2.3 The optimal Huffman tree.

After we obtain the optimal Huffman tree, we assign bit-representations to the characters, by assigning '0' to left sub-tree and '1' to right sub-tree. We now get:

character	representation
'g'	00
'o'	01
' '	100
'p'	1110
'h'	1101
'e'	101
'r'	1111
's'	1100

Table 2.2 Bit-representation for characters.

Now, using this table, if we convert the string 'go go gophers', we would have 000110000011000001 1110110110111111100. That is 37 bits of data. Using ASCII fixed-length coding, the string would end up as 104 bits of data. This shows how Huffman coding can significantly reduce the length of bit-representation, which

means a reduction in file size as well.

To decode a Huffman code, we need to traverse through the bit stream and the Huffman tree, going to the left sub-tree when we find '0' in the bit stream, going to the right sub-tree when we find '1' in the bit stream, and back to the root when we find a leaf and outputs said character. Using the example above, first we find '0', so we go left in the tree. Next, we find '0' again, so we go left again. We find 'g' in the tree, so we go back to the root and outputs 'g'. Next we find another '0', we go left again in the tree. Next is '1', so we go right, and find 'o'. Go back to the root and output 'o'. Repeat this step until the code is fully decoded. A code of 0001100000110000 01111011011011111100, using the tree from our previous example, will be decoded as 'go go gophers'. This decoding algorithm is very simple, and it runs very fast as well.

III. HUFFYUV

As mentioned above, lossless video compression is useful in video editing environments where people would like to make sure that no loss of data occurs during the countless editing processes. One popular lossless video compression to use is HuffYUV.

HuffYUV works in these steps:
1. Separate the color planes into Y, U, and V planes.
2. Use prediction function to predict each sample.
3. Use Huffman encoding to compress the predicted values.

"Prediction function" here refers to a method to further increase the efficiency of Huffman encoding. While Huffman coding can achieve great compression efficiency, its efficiency drops as the number of unique data rises. For example, if we only have two unique data, then we only need one bit representation for each data, '0' and '1'. But if we have three unique data, we need up to two bits representations for each data, '0', '10', and '11'. As the number of unique data rises, the number of maximum bit-representation rises as well. In video, each channel is represented by an 8-bit integer, which means each channel has the possibility of having up to 255 unique data. This will highly impact the efficiency of Huffman encoding. To mitigate this problem, HuffYUV uses the predictor function, to reduce the amount of possible unique data.

Predictor works by comparing current sample with its neighbors, then predicts the value for the current sample. HuffYUV has three choices for predictor methods:
1. Left predictor, which predicts from the sample left of the current sample.
2. Gradient predictor, which predicts from the sample left of current + sample above current – sample above-left of current.
3. Median predictor, which predicts from the median of the sample left of current, sample above current,

and the gradient predictor.

For example, using median predictor, if the current sample, say X, is 197, left sample, say A, is 193, above sample, say B, is 198, and above-left sample, say C, is 195. Then the gradient predictor is *A+B-C = 196*. The median of A, B, and the gradient predictor is 196, so the median predictor is 196. The predicted value for the current sample *is X – median predictor = 1*. For real-life footage, where most of the time, the difference of a neighboring pixels is usually very small, this effectively reduces the range of possible data to be Huffman-encoded, increasing compression efficiency.

The predicted values are then Huffman-encoded. While normal Huffman encoding creates a table for every data compressed, HuffYUV uses built-in tables for each channel in order to save time. These built-in tables are made from various experiments to find the table that works well with most common videos. While these tables may work quite well in most cases, they are not the optimal Huffman tables. However, some application allows the user to specify custom Huffman tabled to increase efficiency.

After the file has been encoded, the Huffman table is attached to the encoded file. This table is used to later decode the file. By attaching the Huffman table, in case that future HuffYUV encoder uses different table, the old files can still be decoded perfectly without having to explicitly support old tables.

It should be noted that HuffYUV decoding algorithm is really fast. This means that for video editing, HuffYUV encoded videos can be decoded quickly, thus speeding up the editing process. This makes HuffYUV a very suitable choice of video compression codec in video editing environments.

IV. CONCLUSION

Huffman coding is used in many compression algorithms. One of them is the HUffYUV video codec, which is a lossless video compression. Lossless video compression is useful in video editing environments where people need to avoid any data loss caused by lossy video compression. Huffman encoding efficiency for video and image compression can be further improved using predictor functions, with the assumption that in video and images, most of the time a sample's value is very close to its neighbors.

REFERENCES

[1]   http://www.animemusicvideos.org/guides/avtech/video3.htm
      Date of access: December 18th, 2012. 10:36 PM

[2]   http://www.cs.duke.edu/csed/poop/huff/info/
      Date of access: December 18th, 2012, 11:13 PM
[3]   http://neuron2.net/www.math.berkeley.edu/benrg/huffyuv.html
      Date of access: December 19th, 2012 00:23 AM
[4]   http://en.nerdaholyc.com/huffman-coding-on-a-string/
      Date of access: December 19th, 2012, 01:20 AM
[5]   *XIL Programmer's Guide August 1994*. California: Sun Microsystems, 1994 , ch. 17.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Desember 2012

Mahessa Ramadhana