UDC 621.397.3+681.3.06+006

Very High Speed JPEG Codec Library

Arito ASAI*, Ta thi Quynh Lien**, Shunichiro NONAKA*, and Norihisa HANEDA*

Abstract

This paper proposes a high-speed method of directly decoding and scaling down high-resolution JPEG images to low-resolution and small ones, which achieves a dramatic improvement in speed. The algorithm has an insignificant problem which causes noise in the rightmost and undermost pixels of output images in some minor cases. We have worked out some methods to lessen this problem, while hardly slowing down the processing speed.

Experiment results show that our approach, including the above related work, has successfully achieved a 5 to 10 times increase in speed, in comparison with methods using the de-facto standard JPEG library libjpeg.

1. Introduction

In recent years there has been an increase in the creation and usage of large, high-resolution images due to the increase in the resolution of images that digital cameras and the cameras built into mobile phones can handle. However, it is not easy to use these images because it takes processing time to decompress and display the high-resolution image data. In particular, because the resolution of current devices is lower than these images, devices must follow the procedure of decompressing, scaling down, and displaying every time an image needs to be displayed.

The time spent decompressing, scaling down, and displaying these images is one of the most profound causes of the decrease in the usability of user interfaces that display a large number of thumbnail images at the same time such as lists of common images or search screens.

The main subject of this paper is the implementation of a high-speed conversion method that scales down and decompresses high-resolution JPEG data to low-resolution data by decimating Discrete Cosine Transform (DCT) coefficients. We will also present solutions to various problems related to this method.

2. The Cost of Decompressing and Scaling Down JPEG Data

We based the work outlined in this paper on libjpeg of the Independent JPEG Group (IJG) to reach the desired high speed conversion. Fig. 1 shows the time cost for an orthodox JPEG decompression to RGB bitmaps using libjpeg. From this graph we can see that the time required to decompress JPEG data relies heavily on the number of pixels in the decompressed image.

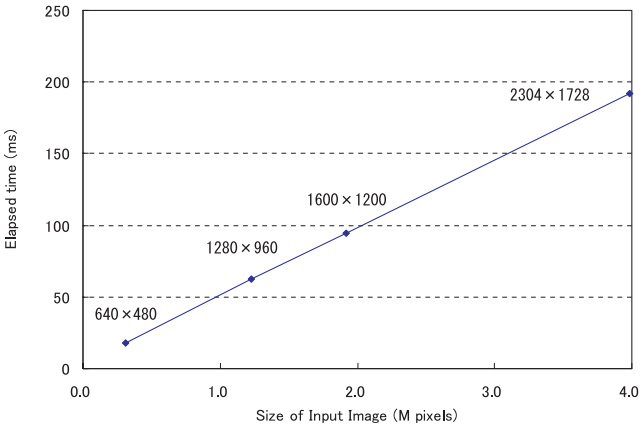

Fig. 1 Orthodox JPEG decompression to RGB bitmaps.

Original paper (Received December 20, 2007)

* Internet Business Development Division
New Business Development Division
FUJIFILM Corporation
Senzui, Asaka, Saitama 351-8585, Japan

** FUJIFILM Software Co., Ltd.
Manpukuji, Asou-ku, Kawasaki, Kanagawa 215-0004, Japan

Figs. 2 and 3 show the time cost of scaling down a high-resolution, decompressed image to a fixed resolution. We used an interpolation algorithm based on the BiCubic method for the scaling algorithm. Because this algorithm was built around the use of the inner product computation through convolution, SIMD instructions, representative of SSE2, were very effective, and we were able to gain a 3 to 5 times performance increase. From these graphs, we can see that the cost of scaling down the image relies heavily on both the number of pixels in the original image and the number of pixels in the image after scaling.

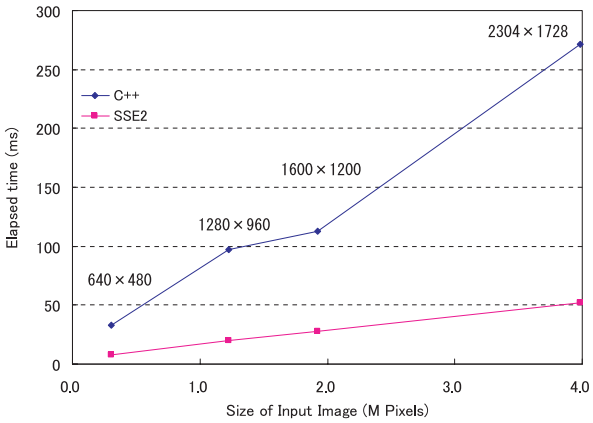

Fig. 2 RGB bitmap scaling (fixed output size QVGA).

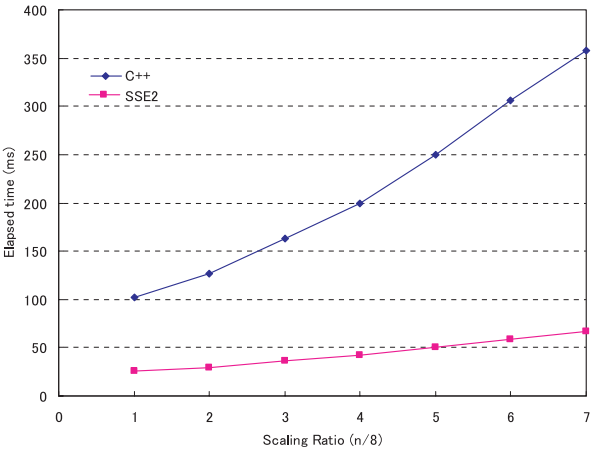

Fig. 3 n/8 RGB bitmap scaling (fixed input size 1600×1200).

3. Scaled Decompression through DCT Coefficient Decimation

JPEG data is obtained by applying a DCT to 8 × 8 pixel blocks and applying lossy compression in the frequency domain. For decompression, one would expect to execute the inverse DCT for the calculated DCT coefficients. However, to produce the final scaled down image, decompressing the DCT coefficients of the high frequency components that are not included in the scaled down image amounts to unnecessary work. By decimating the DCT coefficients to contain only the necessary frequencies and decompressing the JPEG, one can cut down on this unnecessary processing time and minimize the amount of memory used. Due to the nature of DCT, by retaining only the DCT coefficients for low frequency components and taking the inverse DCT of the orders that correspond to the remaining coefficients, one can obtain scaled decompressed data. Like the butterfly operation for the Fast Fourier Transform (FFT), a high-speed inverse DCT is known for performing $1/2^n$ scaling. By using this inverse DCT, 8 × 8 blocks can be directly scaled and decompressed to blocks of size 1 × 1, 2 × 2, or 4 × 4. The scaling ratio is 1/8, 2/8, or 4/8 respectively. This is called $1/2^n$ scaled decompression.

Let us move forward and consider the cases where the scaling ratio is 3/8, 5/8, 6/8, or 7/8. Because the high-speed inverse DCT was not known for these cases, even if scaled decompression were carried out, the inverse DCT computation would become a bottleneck, thus removing any merit from the procedure. We invented a high-speed, nth-order (n = 3, 5, 6, and 7) inverse DCT algorithm. This algorithm makes it possible to perform high-speed JPEG decoding while scaling down and decompressing with a ratio of n/8. Together with the $1/2^n$ scaled decompression discussed previously, we have realized a way of achieving n/8 scaled decompression. Fig. 4 shows a comparison of the time costs between this method and the method of decompressing the data to full scale and using the BiCubic method (via SSE) for scaling. As can be seen from this graph, we achieved a 2 to 6 times increase in speed.

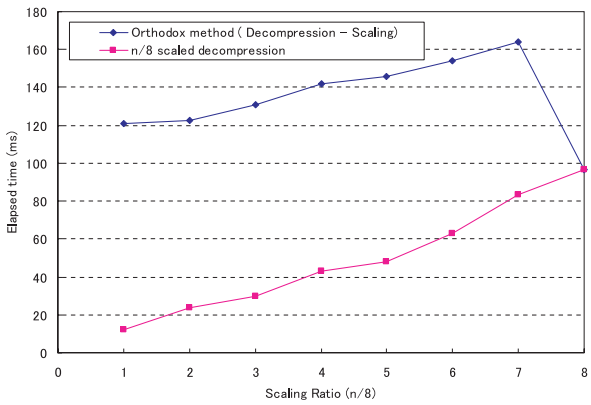

Fig. 4 n/8 Scaled JPEG decompression (fixed input size1600×1200).

4. Various Problems at the Edge of Images

4.1 Padding

As discussed earlier, JPEG data is processed in units of 8 × 8 pixel blocks. When the image width or height is not a multiple of 8, the image is padded out to the next multiple of 8. There is no clear standardized definition that states what kind of data should be used for padding.

In normal JPEG decompression, the entire image, including padding, is decompressed, and then the padding data is cut off based on the original number of pixels. However, because the padding cannot be cut off to an accuracy of one pixel in our n/8

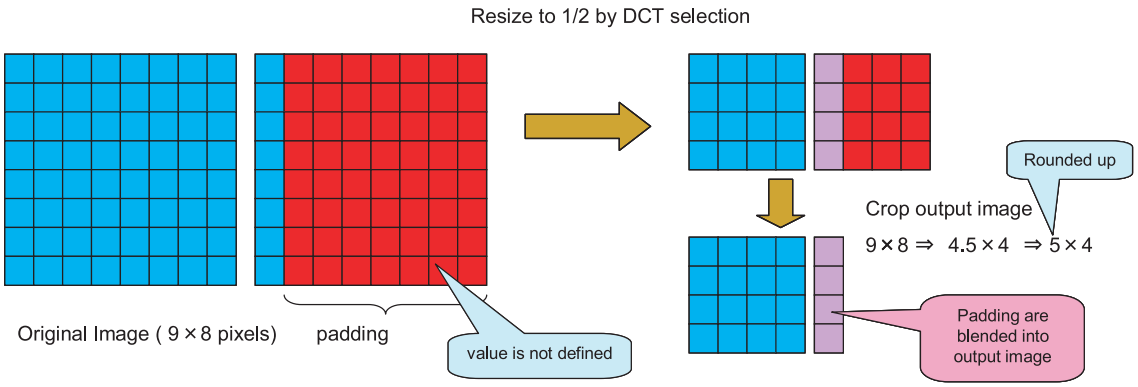

Fig. 6 Principle of padding-blended error.

scaled decompression, the effect of the padding can be seen on the right-hand and bottom edges of the image. See Fig. 5.

Fig. 5 Sample image with padding error.

Fig. 6 shows the theory behind this phenomenon. In the figure, the image width is 1 pixel greater than a multiple of 8. The right-most block contains 1 pixel column of correct image data and 7 pixel columns of padding data. If, for example, this image underwent 1/2 scaled decompression, the scaled down 4 × 4 pixel block produced by DCT decimation includes the result of rounding up the decimal values while the remainder are cut off as padding data. In the figure, 1 pixel column is included in the result and the remaining 3 pixel columns are cut off. However, the image data of this 1 pixel column included in the result contains 50% padding data, and this may become a problem.

The reason why the decimal values are rounded up is because it is not uncommon that practical JPEG data has a border that is 1 pixel wide, and it is undesirable to lose the border line due to truncation.

We used the following method to solve this problem. We abandoned the use of n/8 scaled decompression for blocks that contain padding and processed these edge blocks as follows. First, perform a full-scale decompression in the standard manner to produce 8 × 8 pixel blocks. Second, overwrite the padding pixels with edge pixels from the image. Third, perform an n/8 scaling using spatial averaging. Fourth, cut off undesired parts of the image. There is a reason why padding is not cut off after the full-scale decompression. If padding was cut off after the full-scale decompression, and the image was then scaled down to a fixed resolution, this block's scaling ratio would no longer exactly match n/8 due to rounding errors.

Additionally, we chose spatial averaging for the scale-down algorithm because the frequency response obtained by DCT coefficient decimation is closer to that obtained by this algorithm than those obtained by the BiCubic or BiLinear methods, and this algorithm provides the highest level of consistency with the original image, even for edge blocks.

Let us next consider this method's cost in terms of processing time. If the width or height of an image is not a multiple of 8 pixels, we cannot make use of the merits of the high-speed n/8 scaled decompression for blocks on the lower edge or right-hand edge of the image. However, for an image with a total number of pixels S, the number of edge blocks is of the order $\sqrt{S}$. In other words, the higher the resolution of the original image, the lower the percentage of edge blocks. That is to say, if the original image is large enough, the time cost of the method outlined above becomes insignificant. For example, all of the edge blocks in a 2-megapixel image make up less than 1.2 percent of the total image.

4.2 Resolution Problems with Edge Pixels

Due to the nature of the n/8 scaled decompression algorithm that it decompresses an 8 × 8 pixel block to an n × n pixel block, the scaling ratio exactly matches n/8. However, when the width or height of the original image is not a multiple of 8 pixels, you can not properly attain width and height that are scaled by n/8 due to rounding errors. The question then, is where do these rounding errors go?

In the discussion given in the previous section, padding data was re-created and then cut off after the n/8 scaling. In other words, the one-pixel border adjacent to the padding pixels along the right-hand and bottom edges of the image is scaled down and output together with the copied padding element. This means that the rounding error in n/8 scaling is completely contained in this one pixel border around the right-hand and bottom edges. To put this differently, the scaling ratio of these edge pixels is not consistent with the scaling ratio of all other areas.

This result causes a phenomenon in which the frequency components very close to the scaling ratio for edge pixels and those for other pixels are different. Fig. 7 shows an actual

example of this phenomenon. As you can see, the pattern in the line of pixels along the right-hand edge of the image is different. The theory behind how this phenomenon occurs is explained in Fig. 8.

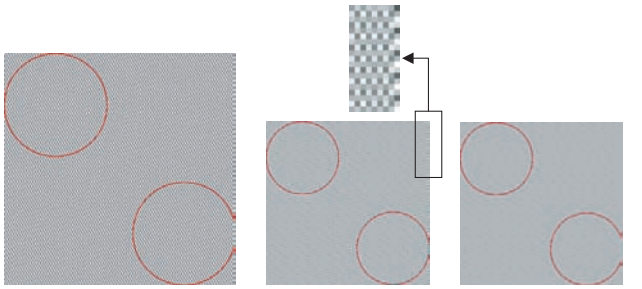

Fig. 7 Sample image with rounding error.

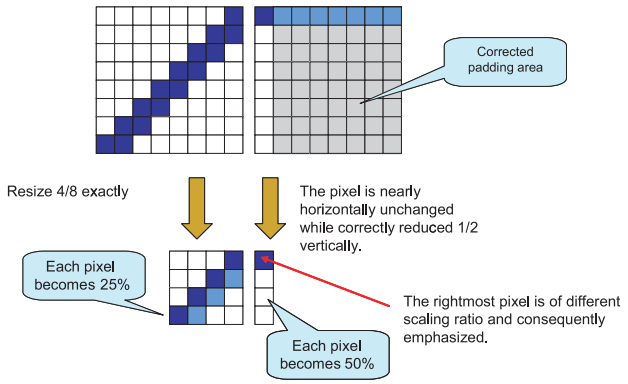

Fig. 8 Influence of rounding error margin in the right-hand edge of image.

By its nature, this problem cannot be resolved. However, we used the following algorithm to reduce its effects.

First, calculate image S via the n/8 scaled decompression method described in previous chapters (round up all decimal values).

Second, prepare image buffer K whose size is obtained by rounding to n/8th the original size. Copy S into K.

Third, calculate the actual weighting of the edge pixels in S (the decimal portions), carry out proportional division with the adjacent pixels, and calculate the modified pixel values.

Fourth, overwrite the edge pixels in K with the pixel values calculated above.

This algorithm corrects the over-contribution of the original image due to the scaling ratio of the edge pixels differing from the scaling ratio of all other pixels. It also controls the phenomenon of only the edge pixels being emphasized because they share their pixel data with the adjacent pixels.

The increase in processing cost for this procedure is relatively small when considered in the same way as the previous section.

4.3 Effects on Processing Time

The increase in time cost due to corrections for the edge blocks and edge pixels discussed in this chapter is shown in Table 1. We can see that the effect on processing time is very small as long as the goal is high-speed processing of high-resolution images.

Table 1 Elapsed Time for1/8 Scaled Decompression.

Input Image	before (ms)	after (ms)	rate
633 × 473	2.46	2.61	106%
1273 × 953	8.53	9.46	111%
1593 × 1193	12.31	12.69	103%
2297 × 1721	21.96	22.74	104%

5. JPEG Encoder

Up to the previous chapter, this paper has discussed a method to decompress high-resolution JPEG data to RGB bitmaps. However, there is a great need to convert high-resolution JPEG data into low-resolution JPEG data.

In this chapter, we will introduce YCC processing as a method that provides an even faster high-speed conversion to meet these needs. JPEG data is converted from the RGB color space to the YCC color space, and the DCT is performed in the YCC color space. Furthermore, it is common to subsample and save the Cr and Cb components to compress the JPEG data. For JPEG data with YUV420 subsampling applied, the Cr and Cb components have half the resolution of the original image, which is one fourth the original size. In other words, compared with decompression in the RGB color space, the output data size in YCC decompression (if the output data size from an RGB decompression is 3) is 1 + 1/4 + 1/4 = 3/2. This method reduces the size to one half. Because you can eliminate the process of changing from YCC to RGB during decoding and changing from RGB to YCC during re-encoding, this method makes even faster speeds possible. If your desired resolution is not n/8, you can obtain YCC data at the desired resolution by scaling down each plane in the YCC color space via the BiCubic method. You can input the result directly into a JPEG encoder.

The improvement in time cost to obtain a low-resolution JPEG image from a high-resolution JPEG image via this method is shown in Fig. 9.

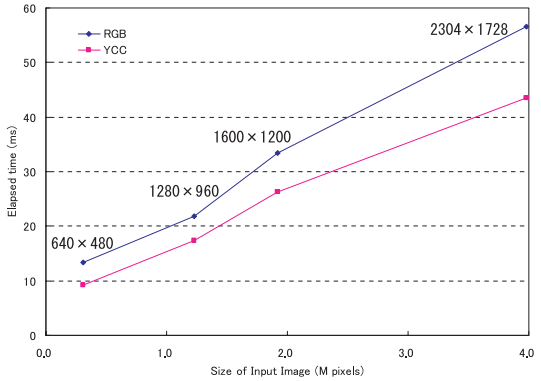

Fig. 9 JPEG decompression-QVGA scaling-JPEG compression sequence.

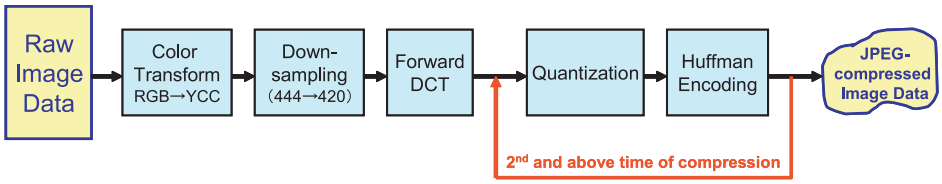

Fig. 10 Compression sequence and retrying pattern.

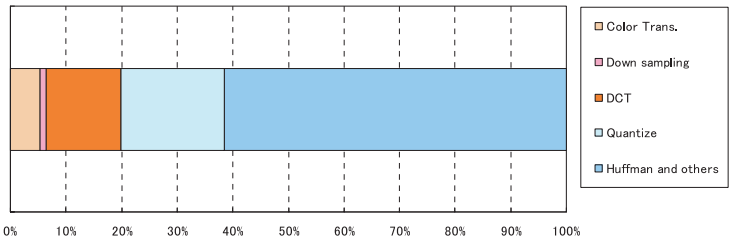

Fig. 11 Percentage of elapsed time for each JPEG compression sequence.

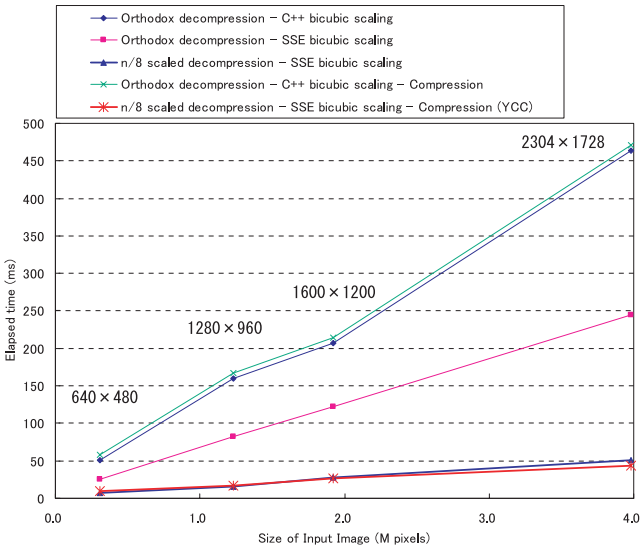

Fig. 12 Time required to convert JPEG into QVGA.

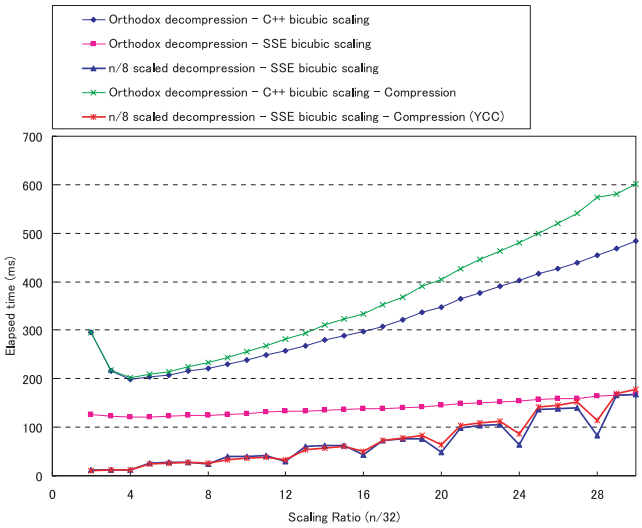

Fig. 13 Time required to compress 1600×1200-sized JPEG into each size.

6. Bit Rate Control

To provide JPEG images for devices such as mobile phones, it is necessary to define a limit for the size of JPEG data files. The size of JPEG data files changes based on the number of pixels and the quantization coefficient. The factor that can be externally manipulated is the quantization coefficient. The size of a JPEG data file and the image quality can be controlled by increasing or decreasing this coefficient. However, the relationship between the quantization coefficient and the bit rate is complicated, and because the number of pixels greatly affects this relationship, it is not possible to predict the quantization coefficient in advance.

Therefore, to control the bit rate of JPEG data, a binary search based on approximation metrics must be performed, and a quantization coefficient that falls within the desired range must be repeatedly searched for. The conversion of RGB to YCC and the DCT conversion need only to be performed once in the procedure in which the quantization coefficient is repeatedly changed through the logic processing sequence of the JPEG encoder as shown in Fig. 10. These initial steps can be excluded from subsequent repetitions. As shown in Fig. 11, the time spent processing the DCT coefficients is about 20 percent of the total time cost for the JPEG encoding.

7. Conclusion

This paper has shown that through the use of various methods, we have made it possible to decompress high-resolution images to high-quality, low-resolution images at high speeds, and then display the images or re-encode them in the JPEG format. In addition, our method avoids deterioration of the image quality visible in the edge pixels. Benchmarks from these conclusions are shown in Figs. 12 and 13. In cases where the scaling ratio is not n/8, we have employed a 2-step scaling process that makes use of the SSE implementation of the BiCubic method. As a result, there is a visible difference between the n/8 points and all other points in Fig. 13.

As discussed in this paper, we have achieved a 5 to 10 times increase in speed in terms of the processing time cost for the scaled decompression of high-resolution images and the creation of JPEG thumbnail images. This technology is currently employed in FUJIFILM Corporation's image conversion service designed for mobile phones, Keitai Picture. It is also applicable for, and is used in, various other fields such as high-speed scaling of images down to display size on devices such as mobile phones and high-speed production of images for thumbnail lists. We intend to expand the use of this technology to even more image applications in the future.

References

1) ISO/IEC 10918-1: 1994, Information technology –- Digital compression and coding of continuous-tone still images: Requirements and guidelines.
2) Thomas G. Lane, USING THE JIG JPEG LIBRARY, libjpeg. doc included in jpegsrc. v6b. tar. gz.