Implementation of a Real-Time Software-only Image Smoothing Filter for a Block-transform Video Codec

Wesley F. Miaw
Lawrence A. Rowe

Computer Science Division-EECS
University of California
Berkeley, CA 94720

Jun 17, 2002

Abstract

The JPEG compression standard is a popular image format. However, at high compression ratios JPEG compression, which uses block-transform coding, can produce blocking artifacts, or artificially introduced edges within the image. Several post-processing algorithms have been developed to remove these artifacts. This paper describes an implementation of a post-processing algorithm developed by Ramchandran, Chou, and Crouse (RCC) which is fast enough for real-time software-only video applications.

The original implementation of the RCC algorithm involved calculating thresholds to identify artificial edges. These calculations proved too expensive for use in real-time software-only applications. We replaced these calculations with a linear scale approximating ideal threshold values based on a combination of peak signal-to-noise ratio calculations and subjective visual quality. The resulting filter implementation is available in the widely-deployed Open Mash streaming media toolkit.

1. Introduction

One of the most attractive new technologies enabled by the Internet is real-time video conferences. The ability to stream high quality video and audio for collaborative purposes has until recently been cost prohibitive because of the infrastructure and hardware required. The Internet and advancing technology have made it possible to use existing infrastructure with software and relatively cheap hardware to produce high quality real-time video and audio.

However, a single high quality video stream requires a large amount of bandwidth. For this reason, video streams are often compressed before transport. Block transform algorithms are widely used for streaming media (e.g. Motion-JPEG, MPEG, and H.26x). At high compression ratios (corresponding to low bit rates) the image quality of a decoded block transform stream noticeably degrades. One way the image degrades is through the artificial introduction of blocking artifacts. The solution is to apply a smoothing filter on block edges after the image is decoded.

The problem is to find a practical implementation of a smoothing filter. There are many algorithms which can remove blocking artifacts but they are too slow for real-time video which requires up to 30 frames per second. Ramchandran, Chou, and Crouse published an algorithm[1] that addressed this problem. Their algorithm to be fast enough for real-time applications.

We implemented the Ramchandran, Chou, and Crouse deblocking algorithm in the Open Mash streaming media toolkit[2]. The Mbone tools, including the video streaming tool *vic*, are widely used around the world for a variety of real-time streaming audio and video applications. Since users of Open Mash applications stream video at low bit rates and high compression ratios, improving video quality is important. However, any algorithm used to improve video quality must provide acceptable performance along with heuristics for application.

This paper describes an implementation of the Ramchandran, Chou, and Crouse (RCC) algorithm. This implementation provides a noticeable reduction in blocking artifacts at up to 30 frames per second on a consumer level CPU. The original RCC algorithm proved too slow for use in real-time applications. Consequently, the most time consuming calculations were replaced by heuristics. The remainder of the paper is organized as followed. Section 2 describes the implementation of the deblocking algorithm. Section 3 provides a performance evaluation of the filter, including both run time performance and image quality comparisons. Section 4 presents conclusions and directions for future work.

2. Implementation

This section describes the smoothing filter performed on block boundaries, the selection of the visual threshold value that determines how the smoothing filter behaves on pixels, and heu-

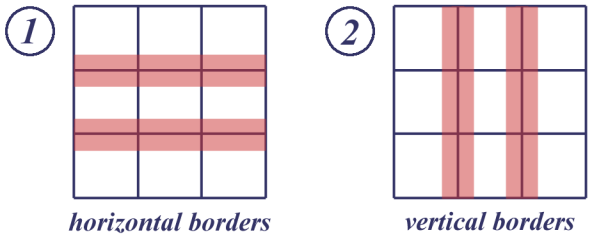

Figure 1. Smoothing pixels across block borders.

ristics to decide when to apply the filter. Both the visual threshold selection and the heuristics are different than the approach suggested by Ramchandran, Chou, and Crouse.

2.1 The Smoothing Algorithm

JPEG compression breaks an image into 8x8 blocks and compresses the blocks individually. For the purposes of this discussion, we will refer to the pixel in column i, row j of block n as $p_n[i,j]$.

The basic idea behind the RCC algorithm is to compare pixels across block borders for artifacts and reduce the difference between the pixel values. The implementation of this algorithm begins by comparing the two pixels above and the two pixels below the horizontal block borders and smoothing their values together. The same computation is done on the two pixels to the left and the two pixels to the right of the vertical block borders. Since there are no blocks adjacent to the edge of the image, those pixels are not smoothed. Figure 1 displays an image consisting of nine 8x8 blocks and indicates which pixels are smoothed across the block borders.

More specifically, the pixels adjacent to each horizontal border are processed as follows. First, the difference between adjacent pixels are calculated. Let the vertical difference between pixels be defined as follows (See Figure 2):

$$v\_diff\_above = p_1[1,7] - p_1[1,8]$$
$$v\_diff\_boundary = p_1[1,8] - p_2[1,1]$$
$$v\_diff\_below = p_2[1,1] - p_2[1,2]$$

The value of *v_diff_boundary* is compared to a threshold value *t*. If $|v\_diff\_boundary| \le t$, the two pixels are considered part of an artificial discontinuity introduced by compression. Otherwise, the two pixels are considered part of an actual discontinuity in the image. The two border pixels are altered by the following equations, where the *visual_threshold* value represents the maximum pixel difference value that cannot be detected by the human eye.

If $|v\_diff\_boundary| \le t$ then:

$$p1[1,8] = p1[1,8] - \alpha * v\_diff\_boundary$$
$$p2[1,1] = p2[1,1] + \alpha * v\_diff\_boundary$$

where

$$\alpha = (t - visual\_threshold) / (2 * t)$$

The RCC algorithm suggests the pixel values be modified even

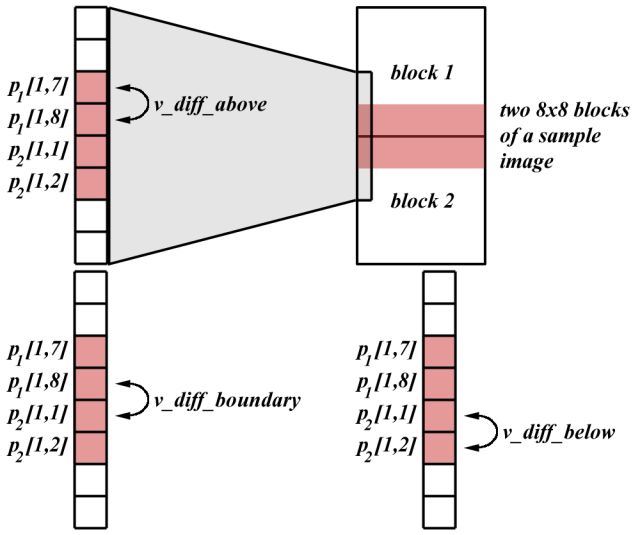

Figure 2. Calculating the vertical difference between pixels.

if they are considered part of an actual discontinuity in order to reduce the blockiness introduced by compression while maintaining the appearance of the discontinuity. The RCC algorithm uses the following equations to modify pixels that are part of an actual discontinuity.

If $|v\_diff\_boundary| > t$ then:

$$p_1[1,8] = p_1[1,8] - \alpha * t$$
$$p_2[1,1] = p_2[1,1] + \alpha * t$$

The general idea is to reduce the difference between pixels in artificial discontinuities to below the *visual_threshold* value and to slightly reduce the difference between pixels in actual discontinuities. However, these calculations created problems for video streams transmitted in sequential image blocks as illustrated in Figure 3a. For such a video stream, *vic* will receive individual blocks of the image over a relatively long period of time. As each block is received, it is added to an image memory buffer and the incomplete image is displayed to the viewer. However, when the renderer code receives these intermediate images, it has no way of recognizing that the image is incomplete. As a result, the boundary between the image blocks received and the area not yet received is identified as an actual discontinuity. Modifying the pixels at this boundary is incorrect behavior, and would blur pixels in the middle of the image with the surrounding gray color. Because subsequently received image blocks are inserted into this modified image, this error resulted in vertical and horizontal lines across the entire frame (see Figure 3b) and also in blurred text when character edges were located at block boundaries.

The renderer code modifies the original image because many block-transform video codecs send future frames as block differentials from the previous frame. These block differentials are inserted into the memory buffer of the previous frame. It is

(a) *(b)*

Figure 3.
(a) A partially complete sequentially transmitted image.
(b) The final image after smoothing actual discontinuities.

more efficient to reapply the algorithm to only the changed blocks and not the entire frame. Modifying the original data allows us to do this, but results in the incorrect behavior described above.

To correct this problem, we changed the algorithm so that it does not modify the pixel values at actual discontinuities. This kept the boundary between the area received and the area not yet received distinct, removed the problem of blurred text, and also resulted in an image of higher visual quality.

At this point, the values of *v_diff_above* and *v_diff_below* are recalculated as *new_diff_above* and *new_diff_below*. This recalculation is necessary since the pixel values of $p_1[1,8]$ and $p_2[1,1]$ may have changed. If the initial difference calculations were zero, but the new calculations are not, we should also smooth $p_1[1,7]$ and $p_2[1,2]$. In that case, we set $p_1[1,7]$ equal to the average of $p_1[1,7]$ and $p_1[1,8]$, and set $p_2[1,2]$ equal to the average of $p_2[1,1]$ and $p_2[1,2]$.

If *v_diff_above* = 0 and *new_diff_above* ≠ 0 then:
$p_1[1,7] = (p_1[1,7] + p_1[1,8]) / 2$

If *v_diff_below* = 0 and *new_diff_below* ≠ 0 then:
$p_2[1,2] = (p_2[1,1] + p_2[1,2]) / 2$

The above calculations are repeated for the rest of the pixels along the horizontal borders, and an equivalent calculation is done along the vertical borders. In the Open Mash implementation, these calculations are only done to the luminance component of images stored in YUV format. As a practical matter, the human eye detects edges based on luminance changes because chrominance changes have a negligible affect on edge detection.

2.2 Calculating the Threshold Value

This algorithm differentiates between artificial and actual edges in an image by comparing the magnitude of the luminance discontinuity to a threshold value. Discontinuities below a certain threshold are considered artificial edges and those above that threshold are considered natural edges.

The paper by Ramchandran, Chou, and Crouse calculated the threshold values by estimating the quantization error of an encoded image. However, this calculation proved too slow for reasonable-sized images. An NTSC video stream required approximately 90 milliseconds per frame to calculate an estimate of the quantization error on an AMD Athlon™ 700MHz processor[3]. That is three times longer than the time available to display the frame (33 milliseconds). This process could be improved by approximating the exponent, hyperbolic tangent, and hyperbolic sine functions for values close to zero, but this optimization only reduced the threshold computation time to 50 milliseconds per frame. The actual deblocking process took 20-30 milliseconds per frame on that same processor. Therefore, the total time required for both the quantization error estimation and deblocking was approximately 80 milliseconds per frame. To get the 30fps required for television, the entire process must take no longer than one frame time.

Additionally, the threshold values being produced by the quantization error estimation calculations were much higher than the expected threshold values. This discrepancy between the thresholds generated by Chou's code and our adaptation of this code to the Open Mash toolkit is probably due to a difference in when this calculation is done. Chou calculated the quantization error during the process of applying the DCT to the image. In contrast, an Open Mash application cannot access the image before receiving the encoded image. Therefore, any quantization error calculations can only be done from data in the encoded image and metadata provided by the encoder.

Because luminance values range from 0-255, a reasonable threshold value should be closer to 0 than 255. If the threshold value is close to 255, almost every luminance discontinuity in the image will be identified as artificial. Threshold values generated by Chou's code on the sample images were between 15-

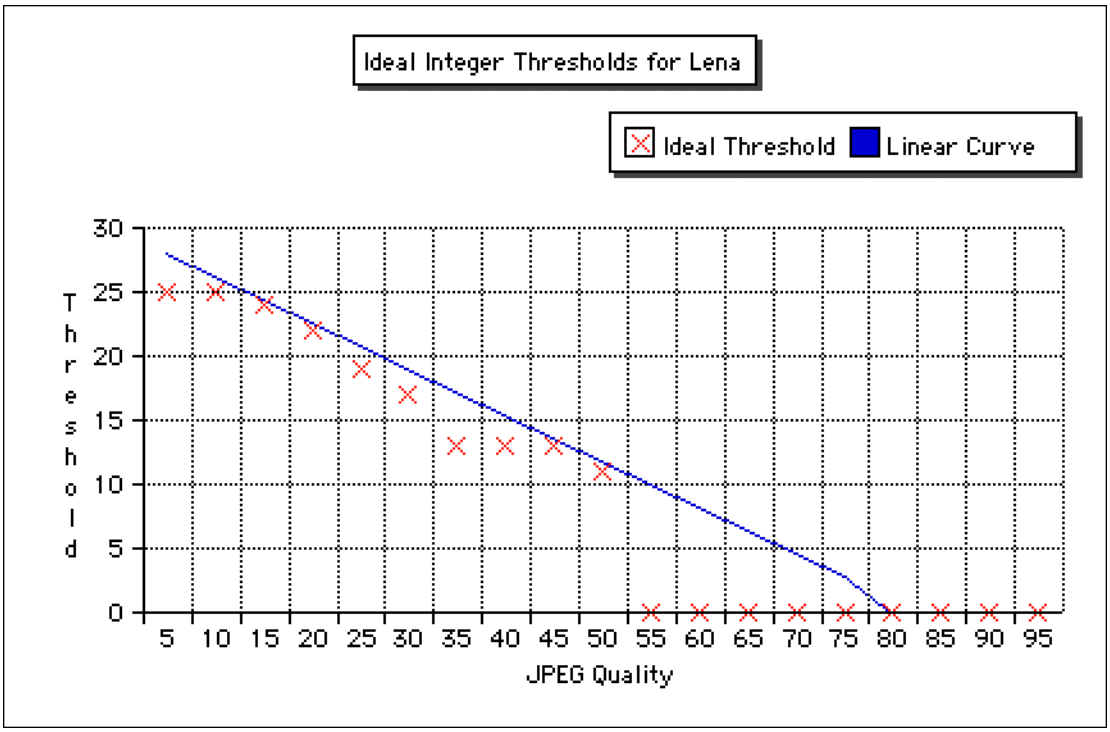

Figure 4. Ideal integer threshold values against JPEG quality levels.

20. However, our calculations on sample images and real-world video streams resulted in threshold values exceeding 150. Using thresholds of this magnitude tended to result in almost all pixel differences falling below the threshold value which meant no filtering was being performed.

To reduce the threshold calculation time and the relatively large threshold values being calculated, we decided to experiment with a threshold value of 1000 for all of the frames being transmitted. Because this value is much higher than the maximum 255, the difference across all block borders is reduced to zero and the α value is essentially 0.5. From a visual standpoint, blocking artifacts were noticeably reduced or eliminated at all JPEG quality levels. (JPEG quality level refers to the compression algorithm input parameter defined by the standard.) However new artifacts were created along diagonal edges of the image due to smoothing at block corners across actual image edges.

Because the threshold value of 1000 was creating additional image artifacts, we looked for the ideal threshold values for specific JPEG quality levels. Peak signal-to-noise ratio (PSNR) calculations on the Lena image produced a graph of the ideal integer threshold values (See Figure 4).

We chose a linear scale to approximate the ideal threshold values because a single threshold value did not apply to all JPEG quality levels and a linear threshold value could be quickly calculated. The linear curve used in Open Mash is a compromise between the calculated ideal threshold values for quality levels below 55 and actual perceived quality for levels above 55. The ideal threshold value derived from the PSNR calculations was zero for quality levels above 55, but the image requires filtering for images up to quality level 80. At quality levels above 80, the image did not exhibit blocking artifacts to the casual viewer. The curve is represented by the following equation:

$$t = \begin{cases} \frac{-18}{50} * quality + 29.8 & \text{if } quality < 80 \\ 0 & \text{otherwise} \end{cases}$$

where t is the threshold and *quality* is the JPEG quality the image was compressed at.

With this simple equation, the time required for the threshold calculation is negligible and the total time required to apply the smoothing filter is at most 30 milliseconds per frame on the processor. At that speed, it is possible to apply the filter to every frame of an NTSC signal transmitted at 30 frames per second not counting decoding time.

2.3 When to Apply the Smoothing Filter

After changing the RCC algorithm so that it does not smooth actual discontinuities, we have not encountered any images

Quality	JPEG	Ideal	Filtered	Delta PSNR
5	21.43	21.62	21.63	0.20
10	25.32	25.56	25.57	0.25
15	27.55	27.79	27.79	0.24
20	28.91	29.13	29.13	0.21
25	29.86	30.00	29.99	0.13
30	30.63	30.73	30.73	0.10
35	31.23	31.32	31.30	0.07
40	31.70	31.76	31.75	0.05
45	32.16	32.19	32.18	0.02
50	32.53	32.53	32.53	0.00
55	32.91	32.91	32.91	0.00
60	33.31	33.31	33.28	-0.03
65	33.84	33.84	33.79	-0.05
70	34.33	34.33	34.27	-0.06
75	34.92	34.92	34.86	-0.06

Figure 5. Comparing the PSNR (dB) of the filtered image to standard JPEG decoding.

where the smoothing filter resulted in a poorer final image. The original algorithm blurred light and dark pixels together resulting in blurred and distorted images. Since the algorithm used in Open Mash only blurs luminance values differing by a small amount, this blurring and distortion no longer takes place and all real-world and test images have benefited from the application of the filter.

However, there are still situations where one would not want to apply the smoothing filter for performance reasons. The application of the filter to an NTSC sized frame required at most 30 milliseconds. This leaves a small but adequate amount of time to display the image on the screen at 30 frames per second. But the same computer cannot apply the smoothing filter to multiple video streams simultaneously or while processing other tasks. Disabling the smoothing filter on an overburdened computer will increase frame rates in exchange for image quality.

2.4 The Source Code

The implementation of the algorithm described above may be viewed at http://www.openmash.org/lxr/source/codec/postdct.h and http://www.openmash.org/lxr/source/codec/postdct.cc.-

3. Evaluation

This section presents results of quality comparisons with and without the smoothing filters.

We applied the filter to the Lena image and compared the results to the unfiltered image. As seen from the table in Figure 5, the smoothing filter slightly improves the PSNR value of the image. It is also interesting to note that not smoothing actual image discontinuities resulted in a slightly higher PSNR value for the image. A visual comparison shows a noticeable improvement in the filtered image as was demonstrated by Ramchandran, Chou, and Crouse (See Figure 6).

4. Conclusion

In summary, the Open Mash implementation of the smoothing filter designed by Ramchandran, Chou, and Crouse provides a noticeable improvement to image quality with acceptable performance.

Further research with block transform codecs other than JPEG should be conducted to determine appropriate threshold heuristics for those streams. Experiments where the threshold calculation is executed by the encoder or where the correct threshold is included as part of the codec could result in improved visual results.

An image with higher quality may also be possible by individually applying the smoothing filter to separate areas of the encoded image. This segmentation would allow fine-tuning of the filter application to different areas of the image. It is also possible to use a threshold value for vertical block borders that

is different from the threshold value used for horizontal block borders in the same area of the image. More advanced heuristics would have to be derived to implement fine-tuning of the filter application.

Acknowledgments

Special thanks to Jim Chou for supplying sample source code of the deblocking algorithm and answering questions patiently and thoroughly.

Figure 6. A visual comparison of filtered images to unfiltered images.

References

[1] J. Chou, M. Crouse, K. Ramchandran. *A Simple Algorithm For Removing Blocking Artifacts In Block-Transform Coded Images*. IEEE Signal Processing Letters, Feb. 1998, Vol. 5, No. 2, pp. 33-35.

[2] Open Mash. http://www.openmash.org/, 2002.

[3] AMD Athlon™ 700MHz. http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_756,00.html, 2002.