Structure and Texture Filling-In of Missing Image Blocks in Wireless Transmission and Compression Applications

Shantanu D. Rane, Guillermo Sapiro, and Marcelo Bertalmio*

Department of Electrical and Computer Engineering

University of Minnesota, Minneapolis, MN 55455, USA

guille@ece.umn.edu.

Abstract

An approach for filling-in blocks of missing data in wireless image transmission is presented in this paper. When compression algorithms such as JPEG are used as part of the wireless transmission process, images are first tiled into blocks of 8×8 pixels. When such images are transmitted over fading channels, the effects of noise can kill entire blocks of the image. Instead of using common retransmission query protocols, we aim to reconstruct the lost data using correlation between the lost block and its neighbors. If the lost block contained structure, it is reconstructed using an image inpainting algorithm, while texture synthesis is used for the textured blocks. The switch between the two schemes is done in a fully automatic fashion based on the surrounding available blocks. The performance of this method is tested for various images and combinations of lost blocks. The viability of this method for image compression, in association with lossy JPEG, is also discussed.

Keywords: Restoration, interpolation, inpainting, filling-in, texture synthesis, JPEG, wireless transmission, compression.

EDICS: 1.4, 1.10

*Currently at Dept. de Tecnologia, University of Pompeu-Fabra, Pg. de Circumvallacio, 8 (Edifici Francia), 08003 Barcelona, Spain.

1 Introduction

General purpose images are most commonly compressed by lossy JPEG. JPEG divides the image into blocks of 8×8 pixels and calculates a 2-D DCT, followed by quantization and Huffman encoding, see [1]. In common wireless scenarios, the image is transmitted over the wireless channel block by block. Due to severe fading, we may lose an entire block, even several consecutive blocks of an image. In [2] the authors report that average packet loss rate in a wireless environment is 3.6% and occurs in a bursty fashion. In the worst case, a whole line of image blocks might be lost. Note that JPEG uses differential encoding for storing the average (dc) value of successive pixels. Hence, even if a single block is lost, the remaining blocks in that line (or reset interval) might be received without their correct average (dc) value. Two common techniques to make the transmission robust are Forward Error Correction (FEC) and Automatic Retransmission Query Protocols (ARQ). Of these, FEC needs extra error correction packets to be transmitted. As noted in [3], ARQ lowers data transmission rates and can further increase the network congestion which initially induced the packet loss. Instead, we show that it is possible to satisfactorily reconstruct the lost blocks by using the available information surrounding them.[1] This will result in an increase in bandwidth efficiency of the transmission. The basic idea is to first automatically classify the block as *textured* or *structured*, and then fill-in the missing block with information propagated from the surrounding pixels. In the case of structured blocks, the inpainting algorithm in [4] is used, while for textured regions we follow [5].[2] We test the proposed scheme with a variety of images and simulated block losses. We also combine this approach with JPEG compression itself, where the encoder voluntarily skips blocks, and these are reconstructed at the decoder in the same fashion as in the wireless scenario. This process improves the compression ratio, at little or no

[1] The location of lost data, that is, lost image blocks, is known in common wireless scenarios.

[2] Other algorithms could be used as well, e.g., [6, 7, 8, 9, 10], but the ones we use were shown in the literature to produce state-of-the-art results at an acceptable low computational cost.

quality degradation.

2 Previous related work

Most schemes reported in the literature deal with image transmission in error-prone environments using a combination of source and channel coding. The authors in [2] describe a packetization scheme in which the DCT coefficients array generated by JPEG is grouped such that bursty (consecutive) packet loss during transmission is scattered into a pseudo-random loss in the image domain (i.e., consecutive blocks are rarely lost in the image domain) . The ensuing reconstruction scheme benefits because, most frequency components can be recovered from adjacent blocks. However, large bursts may cause the errors to cluster in the image, and reconstruction suffers. It should be noted that the packetization scheme proposed in [2], when used with the reconstruction scheme described in our paper, is expected to further improve on the results reported here, and provide satisfactory reconstruction results even for very large bursts.

The authors in [3] also note that interleaving the image data before packetization avoids loss of contiguous areas in an image, facilitating reconstruction. This paper demonstrates reconstruction in the transform domain by expressing the lost data as a linear combination of blocks in the 4-neighborhood of the lost block. Four optimal weights (coefficients) need to be calculated per block based on combinations of available adjacent blocks. These weights, which result in a 10% space overhead, are used later in reconstruction. Strong diagonal edges are not well reconstructed by this method, as explained in [3].

Additional work on the reconstruction of missing data in block-based compression schemes is reported in [11], where the DCT coefficients of a missing block are interpolated from those with the same position in the neighboring blocks.

The novelty of our proposed scheme is in the separation of the lost blocks into different classes,

followed by the use of state-of-the-art image filling-in algorithms for textured and structured regions. This is done in a complete automatic fashion and without any side information.

3 The proposed algorithm

The reconstruction of lost blocks follows three steps:[3]

a. Classify lost blocks into texture and structure.

b. Synthesize blocks which were classified as texture (use Texture Synthesis)

c. Fill in blocks which were classified as structure (use Image Inpainting)

We now proceed to describe each one of these components.

3.1 Block classification

The first step in the reconstruction is to classify the lost blocks into texture or structure. This decision is taken at the receiver by querying the region surrounding the lost block. Lost blocks are, of course, excluded from the querying process. (Alternatively, we may perform this procedure at the transmitter, and then transmit one bit per block, indicating the presence of texture or structure. This entails the overhead of one extra bit per block.) At the core of this classification, is the method proposed in [12].[4] To determine whether or not a block has texture, we use a simple coarseness measure given by the number of local extrema in the neighborhood of the lost block. The number of local extrema are simply the pixels which are local row extrema as well as local column extrema.

[3]Recall that the missing blocks locations are given.

[4]A similar method was used by the research group at the Navy facilities in China Lake (C. Schwartz, personal communication).

Using the method of [12], the number of local extrema in a window of side $s \times s$ is given by

$$n = s^2 \frac{(UB + LB)}{2} \tag{1}$$

where UB and LB are respectively the upper and lower bounds for texture coarseness and are selected by the user. These coarseness values vary from 0 (no extrema) to 1 (all pixels in the selected window are extrema). We have used $UB = 0.16$ and $LB = 0.04$ as suggested in [12]. In this implementation, $s = 8$, giving $n = 6.4$. Thus, if a 8×8 block has fewer than n extrema, it will be classified to have structure, else it will be considered to contain texture.

The above technique is applied for each available block of 8×8 pixels in the immediate neighborhood of the lost block. Even if a single block from this neighborhood contains structure, we have first considered a decision in favor of structure. However, being reconstruction our primary goal, this criteria alone might be insufficient, as we illustrate now. Consider for example that we have lost a block containing an edge between two textured regions. The edge between two regions is certainly an expression of structure, and needs to be given precedence over texture even if the block in question has more than the necessary coarseness. The logic behind this will be understood in the next section, wherein, we require the textured region surrounding the block to fill it up. If we were to classify a block containing an edge as texture, we would not be able to reconstruct the edge later, as will become clear after examining the texture synthesis algorithm.

To overcome this limitation, we impose an additional constraint as follows. We consider the 8-neighborhood of a 8×8 block and calculate difference between the average values of the blocks on opposite sides of the center block (considering only available blocks). If the 4 resulting differences are below a threshold, we decide that an edge does indeed pass through the textured block. We then designate the block as structure, notwithstanding its high coarseness. This simple additional constraint has provided a correct classification in all tested images.

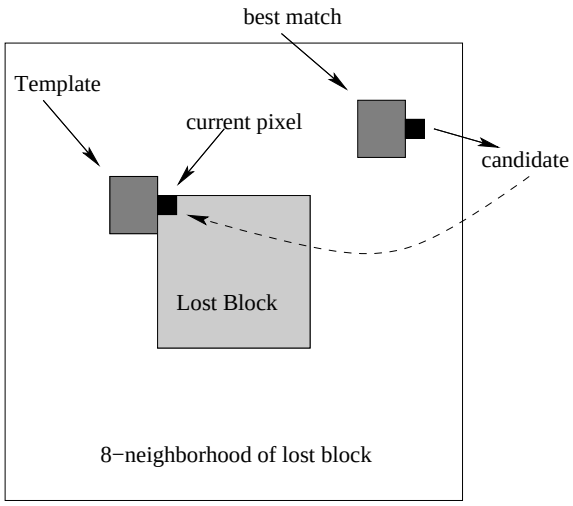


Figure 1: Texture Synthesis procedure

3.2 Texture synthesis

From the earlier classification, we conclude that when a block is classified as having texture, the entire 8-neighborhood of that block has texture. The missing block is then filled-in with the texture from its surrounding, following [5].

Let the region to be filled be denoted by Ω. The lost block will now be filled, pixel by pixel, in a raster fashion. Let I_t be a representative template touching the left of a pixel $p(i,j) \in \Omega$. We proceed to find a $\hat{I}_t$ from the available neighborhood, such that a given distance $d(I_t, \hat{I}_t)$ is minimized. As per [5], d is a normalized sum of squared differences (SSD) metric. Once such a $\hat{I}_t$ is found, we choose the pixel to the immediate right of $\hat{I}_t$, as our candidate for $p(i,j) \in \Omega$. For stochastic textures, the algorithm selects at random one of the pixels neighboring $\hat{I}_t$.

The template I_t can be a simple seed-block of 3×3 pixels as shown in Fig 1. Then, of all possible 3×3 blocks in the 8-neighborhood, the one with the minimum normalized SSD is found and a pixel to its right is copied into the current pixel in the lost block, as shown. This algorithm is considerably fast when using the improvements in [13, 14].

3.3 Image inpainting

Structure in an image can be an edge between two regions or a deterministic change in color or gray value. When the block classification algorithm detected a structured block, this is restored using the digital inpainting procedure introduced in [4].

Once again let Ω be the region to be filled in (inpainted) and $\delta\Omega$ be its boundary. The basic idea in inpainting is to smoothly propagate the information surrounding Ω in the direction of the isophotes entering $\partial\Omega$. Both gray values and isophote directions are propagated inside the region. Denoting by I the image, this propagation is achieved numerically solving the partial differential equation (t is an artificial time marching parameter)

$$\frac{\partial I}{\partial t} = \nabla(\Delta I) \cdot \nabla^{\perp} I,$$

where ∇, Δ, and $\nabla^{\perp}$ stand for the gradient, Laplacian, and orthogonal-gradient (isophote direction) respectively. This equation is solved only inside Ω, with proper boundary conditions in $\partial\Omega$ for the gray values and isophote directions.

Note that at steady state, $\frac{\partial I}{\partial t} = 0$, and $\nabla(\Delta I) \cdot \nabla^{\perp} I = 0$. This means that ΔI is constant in the direction $\nabla^{\perp} I$ of the isophotes, thereby achieving a smooth continuation of the Laplacian inside the region to be inpainted.

For details on the numerical implementation of this inpainting technique, which follows the techniques introduced in [15, 16], as well as numerous examples and applications, see [4].

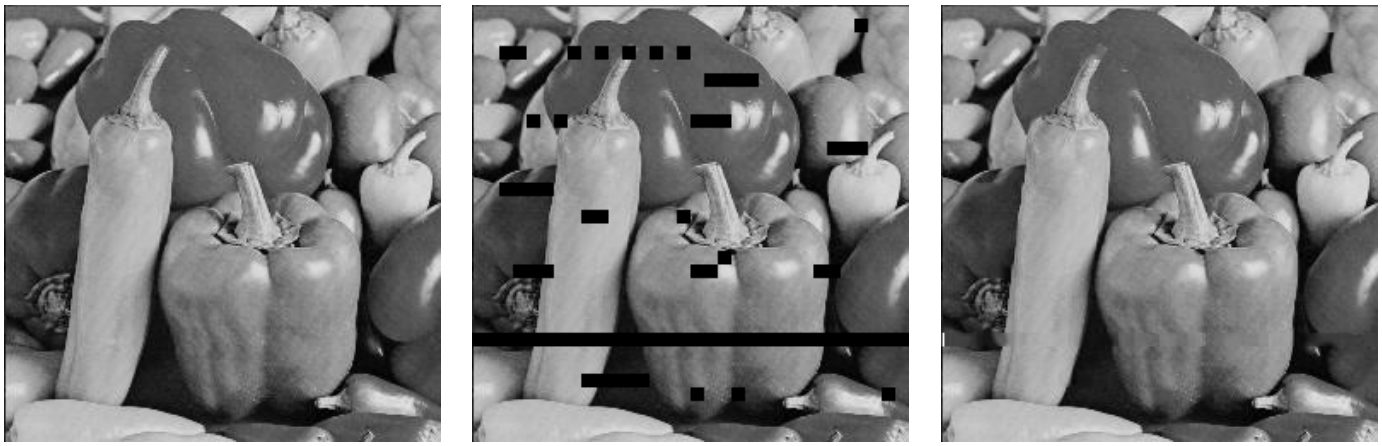
4 Experimental results

4.1 Application to wireless transmission of JPEG compressed images

Throughout our experiments, we have assumed, though it is not completely necessary, that the average (dc) value of a 8×8 block is known at the receiver. This is not needed by the block

Transmitted Image Received Image Reconstructed Image

Figure 2: Reconstruction results for 1,2 or 3 contiguous lost blocks

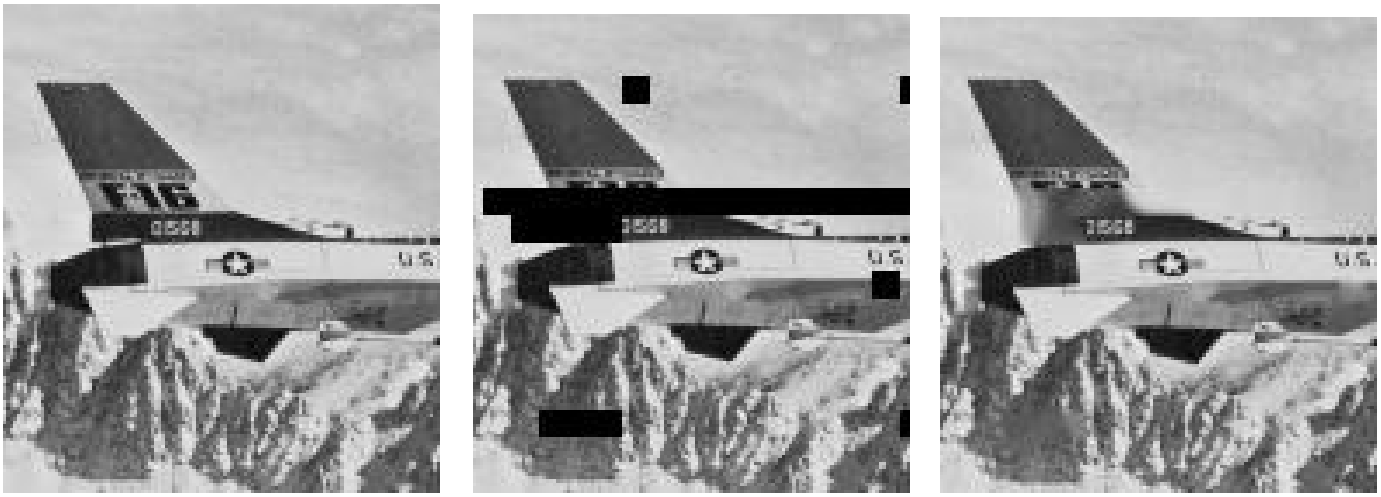


Transmitted Image Received Image Reconstructed Image

Figure 3: Reconstruction results for more drastic losses

reconstruction algorithm, but it is needed for the available surrounding blocks due to the differential encoding of dc values in JPEG. In case the dc value is not known, the method proposed in [17] gives a technique to estimate and correct the dc value of the lost block and the following blocks in the same line. Retransmission of the dc value of a lost block would not be a significant overhead either, since we just need 1 byte to be retransmitted per block.

Since we have no control over the fading channel, there is no prior information about the relative locations and number of blocks that can be lost in the process. We present various examples ranging from low to drastic losses of image information, and demonstrate our proposed technique to restore



Transmitted Image | Received Image | Reconstructed Image

Figure 4: Reconstruction misses details smaller than the mask

Image	% data lost	PSNR
Lena	2.7 %	43.3851
Peppers	7.32 %	33.8953
Jet	10.44 %	3.2778

[Image size = 256×256. See reconstructed Jet Image Fig 4]

Table 1: % data lost and PSNR values after reconstruction

the lost information . Figures 2, 3, and 4 show the results of reconstruction (from left to right, original, transmitted, and reconstructed image). Table 1 shows the amount of missing data along with the PSNR values after reconstruction. We can make the following observations:

a. When single blocks are missing from the image, they are satisfactorily reconstructed from the surrounding context. Note how the reconstructed image is almost identical to the original one, as expected from the algorithms used for filling-in.

b. When a few contiguous blocks are missing, the algorithm still reconstructs the blocks so as to be visually unrecognizable from the original.

c. As a drastic condition, if an entire line is missing, then, a good reconstruction is not always possible. See Fig 3. This is mainly due to the fact that most probably such a significant loss will cover entire objects. In general, when feature sizes are smaller than 8×8 pixels or are

totally covered by the missing line, it will be impossible to reconstruct the image correctly. See Fig 4. In such cases, we will be forced to request retransmission of the lost block (or an error block). Such instances will become increasingly rare when the image resolution is increased. Further, as explained in Section 2, if the packetization in [2] is used, then it is extremely rare to lose an entire line in the image domain. In that case, only independent lost blocks would need to be reconstructed from their neighborhood, and as indicated by Figs. 2 and 3, the above algorithm restores isolated blocks reliably.

4.2 Application to image compression

Since inpainting faithfully reproduces lost edges, and texture synthesis faithfully grows lost textures, we can afford to voluntarily remove some blocks containing structure and texture, even prior to compression. After *intentionally* removing the blocks and replacing them with their dc values, we compress the image using lossy JPEG with default settings. Finally, when the image is decompressed, the above algorithm is used to reconstruct the "lost" blocks.[5] This improves the compression ratio provided by lossy JPEG. In this situation, we have the freedom to choose the blocks which we want to remove, and later reconstruct. A common fixed mask to remove blocks from all images is not advisable of course, since for example, it may totally obliterate important details in images where feature sizes are less than 8×8 pixels, and hence cannot be reconstructed later.

The objective is to retain only the information which *cannot* be correctly reconstructed (minute but important details) and to remove as much as possible from the remainder of the image. The problem thus reduces to finding the best possible mask for a given image. Following the earlier distinction between texture and structure, our algorithm uses the following general rules to construct the mask

[5] As in the wireless scenario, we assume that the positions of the lost blocks are known, since many simple strategies can be devised to automatically detect this position (e.g., blocks with only dc as non-zero coefficients). We also automatically detect the block type, as before, so no overhead is needed at all.

Image	pixels	lossy JPEG		JPEG with fill-in		PSNR
		BPP	CR	BPP	CR	
Lena	256 × 256	1.31	6.13	1.16	**6.90**	36.23
Peppers	256 × 256	1.50	5.33	1.31	**6.11**	33.72
Jet	256 × 256	1.46	5.48	1.31	**6.11**	32.59
Lena	512 × 512	1.11	7.21	0.89	**8.99**	34.25
Peppers	512 × 512	1.04	7.69	0.81	**9.88**	32.10
Jet	512 × 512	1.04	7.69	0.93	**8.60**	36.08

Table 2: Improvement in Compression Ratio

automatically (see also concluding remarks):

a. For areas of texture, remove as many blocks as possible, since they can be satisfactorily reconstructed from a single seed block.

b. Remove a few, but not all blocks along an edge, so that the direction of the edge is properly preserved. Ideally, alternate blocks along an edge should be removed. Presently, a block containing an edge is masked only if the regions on either side of the edge are flat, i.e., the gradient is steep on both sides of the block. This is done, because, while experimenting with various cases in Section 4.1, it became clear that such edges are best reconstructed by inpainting.

c. For blocks with smooth variations (i.e., structure without edges), remove alternate blocks. Inpainting always restores these smooth variations.

Fig 5 shows how the algorithm is used in conjunction with lossy JPEG. Note that the image quality is very similar to the case in which JPEG alone is used. Other examples are shown in Figs 6 and 7. The mask is found automatically by the algorithm from the image and the output of a Canny edge detector. The bits per pixel (BPP) and compression ratio (CR) obtained by lossy JPEG alone, and the compression ratio obtained using JPEG with the above algorithm, are shown in Table 2. Note once again that no overhead is needed by the proposed compression strategy, and the bitstream it is fully JPEG compliant.

5 Conclusions and future directions

In this paper we have proposed a new technique for the filling-in of missing blocks in wireless transmission of JPEG (or block based) compressed images. We have shown that as long as the features in the image are not completely lost, they can be satisfactorily reconstructed using a combination of computationally efficient image inpainting and texture synthesis algorithms. This eliminates the need for re-transmission of lost blocks. When the image resolution is increased, the quality of reconstruction improves and a retransmission request is rarely required, resulting in a better effective data transmission rate.

Further, by intentionally (and automatically) dropping image blocks, and using this filling-in approach, we can improve the compression ratio provided by lossy JPEG, without altering the existing JPEG algorithm. As seen in Table 2, the improvement in compression ratio becomes more significant as the image resolution is increased.

A number of research directions should be taken following the results reported here. We have tried to use image-dependent information, i.e., texture and structure, to enhance the performance of JPEG. The compression ratio can be further increased by finding *better* masks by providing more image information. In a more general setting, the extension of the approach presented here, to color data needs to be investigated. Since the missing blocks in the different channels need not be in the same image position, information from different channels can be used in the block classification and reconstruction. Adding this to the current neighboring information used is expected to improve even further the quality of the results. A preliminary example is presented in Fig 8. Further results on this will be reported elsewhere.

Acknowledgments

This work was partially supported by a grant from the Office of Naval Research ONR-N00014-97-1-0509, the Office of Naval Research Young Investigator Award, the Presidential Early Career Awards for Scientists and Engineers (PECASE), a National Science Foundation CAREER Award, and by the National Science Foundation Learning and Intelligent Systems Program (LIS).

References

[1] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, 1991.

[2] E. Chang, "An image coding and reconstruction scheme for mobile computing," in *Proceedings of the 5th IDMS (Springer-Verlag LNCS 1483), p.137-148, Oslo, Norway, September 1998.*, 1998.

[3] S. S. Hemami, "Digital image coding for robust multimedia transmission," in *Symposium on Multimedia Communications and Video Coding*, (New York, NY), 1995.

[4] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," *Computer Graphics (SIGGRAPH 2000)*, pp. 417–424, July 2000.

[5] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," *IEEE International Conference on Computer Vision, Corfu, Greece*, pp. 1033–1038, Sept. 1999.

[6] C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera, "Filling-in by joint interpolation of vector fields and gray levels," *IEEE Trans. Image Processing*, to appear.

[7] T. Chan and J. Shen, "Mathematical models for local deterministic inpaintings," *UCLA CAM Report*, March 2000.

[8] D. Heeger and J. Bergen, "Pyramid based texture analysis/synthesis," *Computer Graphics (SIGGRAPH 1995)*, pp. 229–238, July 1995.

[9] S. Masnou and J. Morel, "Level-lines based disocclusion," *IEEE Int. Conf. Image Processing*, October 1998.

[10] E. Simoncelli and J. Portilla, "Texture characterization via joint statistics of wavelet coefficient magnitudes," *Proc. 5th IEEE Int'l Conf. on Image Processing*, 1998.

[11] Z. Alkachouch and M. Bellanger, "Fast dct-based spatial domain interpolation of blocks in images," *IEEE Transactions on Image Processing*, vol. 9, pp. 729–732, Apr. 2000.

[12] K. Karu, A. K. Jain, and R. M. Bolle, "Is there any texture in the image?," *Pattern Recognition*, vol. 29, no. 9, pp. 1437–1446, 1996.

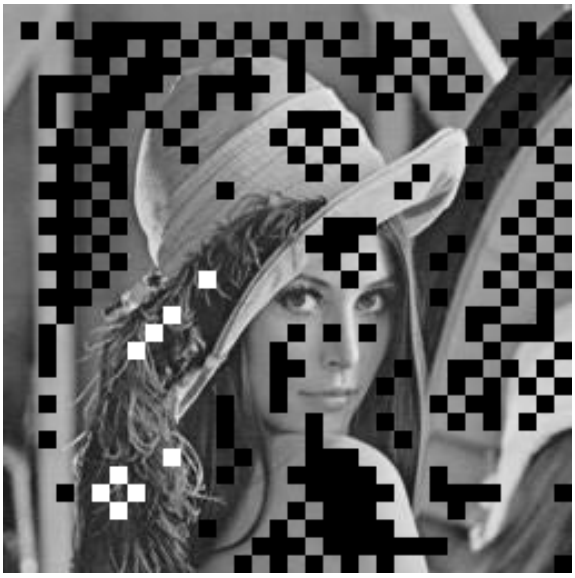
[13] G. Gorla, V. Interrante, and G. Sapiro, "Growing fitted textures," *IMA Technical Report (www.ima.umn.edu)*, February 2001.

[14] L. Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," *Computer Graphics (SIGGRAPH 2000)*, July 2000.

[15] A. Marquina and S. Osher, "Explicit algorithms for a new time dependent model based on level set motion for nonlinear debluring and noise removal," *UCLA CAM Report*, Jan. 1999.

[16] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, no. 60, pp. 259–268, 1992.

[17] S. Shirani, F. Kossentini, and R. Ward, "Reconstruction of baseline JPEG coded images in error prone environments," *IEEE Transactions on Image Processing*, vol. 9, pp. 1292–1299, July 2000.



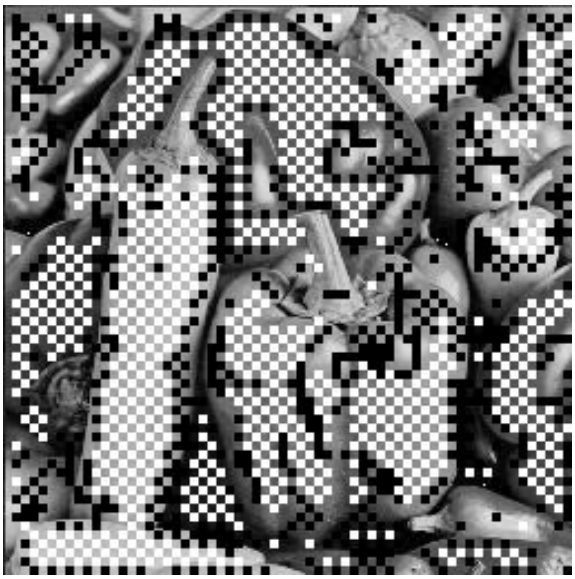
Masked Image
(Black = Structure, White - Texture)

dc values filled-in.
(Compressed with lossy JPEG)

Reconstruction using inpainting
and texture synthesis with JPEG

Image when JPEG alone is used
(Provided for comparison)

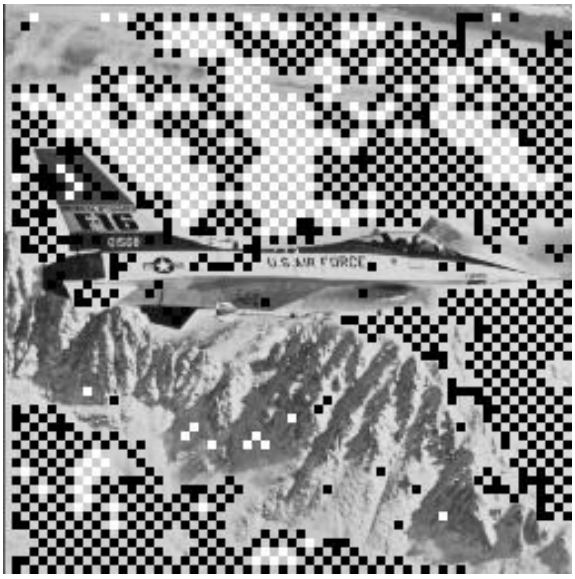
Figure 5: Stages in Compression of 256 $\times$ 256 Lena image



Masked Image

Reconstructed Image

Figure 6: Other examples(1) - Result after Decompression of 512 × 512 Peppers image



Masked Image

Reconstructed Image

Figure 7: Other examples(2) - Result after Decompression of 512 × 512 Jet image

Received Image (Some red blocks missing)

Red Component

Blue component inserted into missing
red blocks as initial condition
(note how the blue edges hint the red ones)

Reconstructed Image

Figure 8: Using information from blue component to reconstruct lost blocks in red component