Image Compression: JPEG and JPEG 2000

Maria Loginova
Jie (Kerry) Zhan

Abstract

How to save an image file efficiently without losing the quality? JPEG has been one of the answers for quite some years.

JPEG stands for Joint Photographic Experts Group, and these are the people who developed the standards of JPEG image compression. Over the past few years, JPEG has been a quite popular format for images, especially when working with the booming Internet. Using DCT (Discrete Cosine Transform) algorithms and related quantization method, JPEG can compress the original image file to a relative reasonable size without losing much detail. In this report, we will be exploring, with simulations and coding, the technical details of JPEG.

As time passed by, JPEG is gradually becoming insufficient in some applications due to its "block effects." Therefore, there is a need to develop new standards for better performance in image compression. That's original goal of JPEG 2000, that is, to represent an image using even less bytes while not losing the details and also not creating "artifacts" like JPEG did. So, followed by discussion of JPEG, there will be a description part for this new image compression standard.

Project Plan

Week 1 (Sun, Feb 2 - Sun, Feb 9)	Primary Resp.	Status
Initial Look on Image Coding	Maria/Kerry	Done
Search for Relevant Literature	Maria/Kerry	Done
Overall Project Plan Decisions and Project Plan Report Writing	Maria/Kerry	Done
Weeks 2,3 (Sun, Feb 9 - Sun, Feb 23)	**Primary Resp.**	**Status**
Overview of DCT and Quantization and their use in JPEG coding	Maria	Done
Overview of wavelets and their use in JPEG2000 coding	Kerry	Done
Weeks 4,5 (Sun, Feb 23 - Sun, March 9)	**Primary Resp.**	**Status**
Finding JPEG2000 demo software	Kerry	Done
Starting simple C++ drawing program	Maria	Done
Week 6 (Sun, March 9 - Sun, March 16)	**Primary Resp.**	**Status**
Mid-Report Writing	Maria/Kerry	Done
Week 7 (Sun, March 16 - Sun, March 23)	**Primary Resp.**	**Status**
Spring Break!		
Weeks 8,9 (Sun, March 23 - Sun, April 6)	**Primary Resp.**	**Status**
JPEG2000-standard research	Kerry	Done
Continue with the drawing program	Maria	Done
Week 10-13 (Sun, March 6 - Sun, May 4)	**Primary Resp.**	**Status**
Write simple DCT algorithm	Maria	Done
Write simple Quantizer algorithm	Maria	Done
Write simple Run-Length coder algorithm	Maria	Done
Combine Algorithms with a drawing program	Maria	Done
JPEG vs. JPEG2000	Kerry	Done
Prepare material for presentation	Maria/Kerry	Done
Week 14 (Sun, May 4 - End of the Semester)	**Primary Resp.**	**Status**
Anything Past Deadline	Maria/Kerry	Done
Any Newly Found Relevant Material	Maria/Kerry	Done
Final Report Writing	Maria/Kerry	Done

Project Accomplishments

PART I: JPEG Standard

JPEG is a very widely used lossy data compression standard. The figure 1 below outlines the basic steps involved in the compression process. First image data is divided into 8 by 8 blocks. These blocks are then transformed using a discrete cosine transform

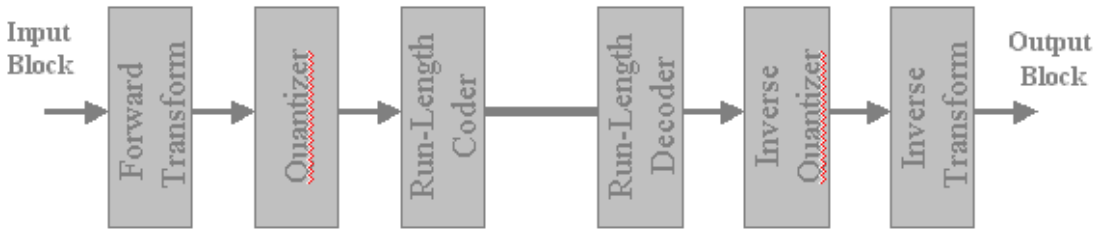
(DCT) that decomposes each 8 by 8 block into 64 DCT coefficients, each of which corresponds to a respective DCT basis vector. DCT transform is a lossless step.

Next this transformed image data is forwarded to a quantizer. The purpose of quantization is to achieve the maximum amount of compression by representing DCT coefficients with no greater precision than it is necessary to achieve the desired image quality. JPEG has defined matrices that can be used for quantization. This is the lossy step.

Finally the quantized coefficients are encoded achieving additional lossless compression.

In further sections of this report we will analyze each step and show how we implemented it in our simple C++ drawing program showing.

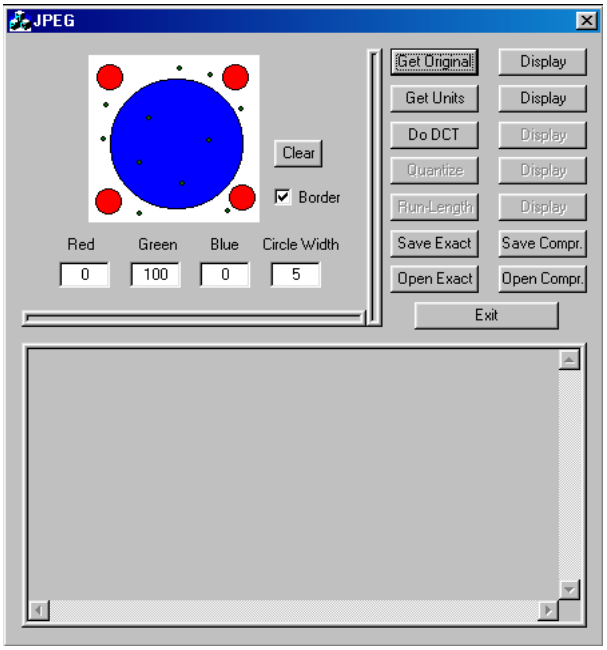
Figure 1. Compression and Decompression Steps



Obtaining Raw Data

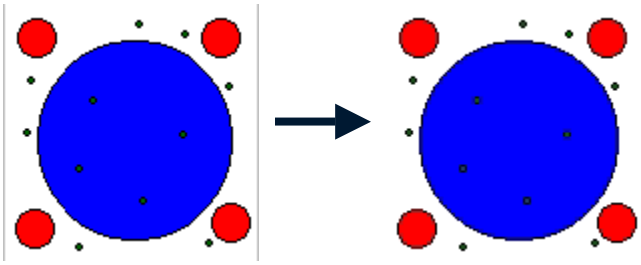
In our drawing program the raw data is obtained by taking the RGB values of each pixel from the 128x128 drawing window (Figure 2) and converting them into YCbCr.

Figure 2. Draw the Picture and Get its RGB Values



The next step is breaking up the 128x128 data matrix into minimal coding units (MCU's) where each MCU consists of 4 8x8 Y blocks, 1 8x8 Cb Block and 1 8x8 Cr block. This step is lossy because the chrominance values are obtained by taking the average of each four pixels. This affects the sharp edges if any are present in the image. Because of this step JPEG compression does not work well with cartoon-like images having sharp edges and color changes.

Figure 3. Breaking Up into MCU's and 8x8 blocks



DCT Transform

To perform DCT transform, first we had to find 1D basis:

$u(k; n) = a(k)\cos(\pi k(2n+1)/2N)$, where $a(0) = \text{sqrt}(1/N)$, $a(k) = \text{sqrt}(2/N)$ for $k = 1,\ldots,N-1$

and in our case N = 8 because of 8x8 matrices.

```
void CJPEGDlg::getBasis()
{
 int n,k;

 for(n = 0; n < 8; n++)
 {
  BasisArray[0].base[n] = sqrt(1.0/8.0) * 1.0;
 }


 for(k = 1; k < 8; k++)
 {
  for(n = 0; n < 8; n++)
  {
   BasisArray[k].base[n] = sqrt(2.0/8.0) * cos(k * 3.141592 * (2*n + 1) / 16);
  }
 }

}
```

Then find the full 8D basis using:

$U(k,l) = u(k)T(u(l))$

```
void CJPEGDlg::getFullBasis()
{
 int i, j;
 int k, l;

 for (k = 0; k < 8; k++)
 {
  for(l = 0; l < 8; l++)
  {
   for(i = 0; i < 8; i++)
   {
    for(j = 0; j < 8; j++)
    {
     fullBaseArray[k][l].fullBase[i][j] = BasisArray[k].base[j] * BasisArray[l].base[i];
    }
   }
  }
 }
}
```

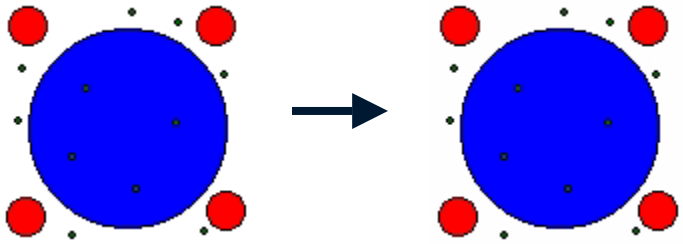
Finally to find DCT coefficients the following formula was used:

$T(k,l) = (U(k,l),S)$ where S is the 8x8 block that is being transformed.

```
for(count = 0; count < 64; count++)
{
 for(m = 0; m < 8; m++)
 {
  for(n = 0; n < 8; n++)
  {
   for(i = 0; i < 8; i++)
   {
    for(j = 0; j < 8; j++)
    {
     DCTcoefficients[count].YBlock1[m][n] += fullBaseArray[m][n].fullBase[i][j]
                                             * Units[count].YBlock1[i][j];
     DCTcoefficients[count].YBlock2[m][n] += fullBaseArray[m][n].fullBase[i][j]
                                             * Units[count].YBlock2[i][j];
     DCTcoefficients[count].YBlock3[m][n] += fullBaseArray[m][n].fullBase[i][j]
                                             * Units[count].YBlock3[i][j];
     DCTcoefficients[count].YBlock4[m][n] += fullBaseArray[m][n].fullBase[i][j]
                                             * Units[count].YBlock4[i][j];
     DCTcoefficients[count].CbBlock[m][n] += fullBaseArray[m][n].fullBase[i][j]
                                              * Units[count].CbBlock[i][j];
     DCTcoefficients[count].CrBlock[m][n] += fullBaseArray[m][n].fullBase[i][j]
                                             * Units[count].CrBlock[i][j];
    }
   }
  }
 }
}
```

Figure 4 demonstrates the image received by reconstructing the DCT coefficients. These operations produce no visible changes to the image.

Figure 4. Image Reconstructed from DCT coefficients



Quantization

Quantizing the DCT coefficients involves two steps. First finding the Qindex:

Qindex (f) = floor((f + Q/2))/Q)

```
void CJPEGDlg::getQindex()
{
 int count(0);
 int i, j;

 for(count = 0; count < 64; count ++)
 {
  for(i = 0; i < 8; i++)
  {
   for(j = 0; j < 8; j++)
   {
    Qindex[count].YBlock1[i][j] = floor((DCTcoefficients[count].YBlock1[i][j] +
                                        Qluminance[i][j]/2.0)/Qluminance[i][j]);
    Qindex[count].YBlock2[i][j] = floor((DCTcoefficients[count].YBlock2[i][j] +
                                        Qluminance[i][j]/2.0)/Qluminance[i][j]);
    Qindex[count].YBlock3[i][j] = floor((DCTcoefficients[count].YBlock3[i][j] +
                                        Qluminance[i][j]/2.0)/Qluminance[i][j]);
    Qindex[count].YBlock4[i][j] = floor((DCTcoefficients[count].YBlock4[i][j] +
                                        Qluminance[i][j]/2.0)/Qluminance[i][j]);
    Qindex[count].CbBlock[i][j] = floor((DCTcoefficients[count].CbBlock[i][j] +
                                        Qchrominance[i][j]/2.0)/Qchrominance[i][j]);
    Qindex[count].CrBlock[i][j] = floor((DCTcoefficients[count].CrBlock[i][j] +
                                       Qchrominance[i][j]/2.0)/Qchrominance[i][j]);
   }
  }
 }

}
```

And then finding quantized values using: Q(f) = Qindex(f) * Q

```
void CJPEGDlg::Quantize()
{
 int count;
 int i, j;

 for(count = 0; count < 64; count++)
 {
  for (i = 0; i < 8; i++)
  {
   for(j = 0; j < 8; j++)
   {
    QMatrix[count].YBlock1[i][j] = Qindex[count].YBlock1[i][j] * Qluminance[i][j];
    QMatrix[count].YBlock2[i][j] = Qindex[count].YBlock2[i][j] * Qluminance[i][j];
    QMatrix[count].YBlock3[i][j] = Qindex[count].YBlock3[i][j] * Qluminance[i][j];
    QMatrix[count].YBlock4[i][j] = Qindex[count].YBlock4[i][j] * Qluminance[i][j];
    QMatrix[count].CbBlock[i][j] = Qindex[count].CbBlock[i][j] * Qchrominance[i][j];
    QMatrix[count].CrBlock[i][j] = Qindex[count].CrBlock[i][j] * Qchrominance[i][j];
   }
  }
 }

}
```

In our program we used default JPEG Quantization matrices for our Qchrominance and Qluminance.

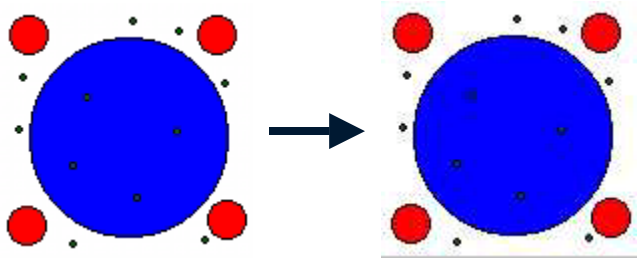
Qluminance Qchrominance

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

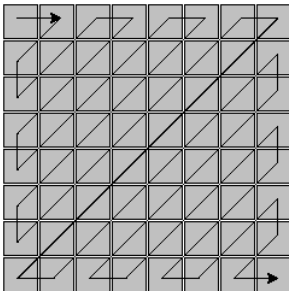
Figure 5 below shows the result of applying our quantization algorithm in the drawing program.

Figure 5. Image Reconstructed from Quantized DCT coefficients



Run-Length Coding

At this step the 8x8 blocks of quantized coefficients are written as strings using the Zig Zag ordering.



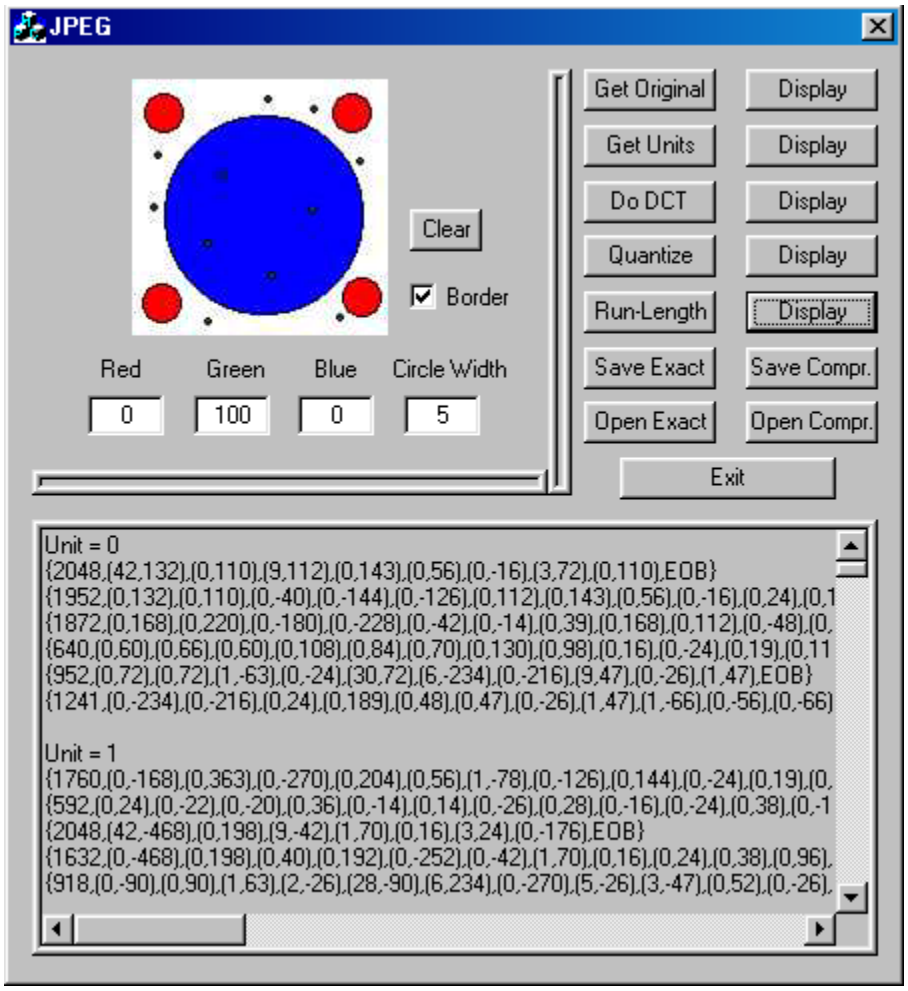
Each string starts with a DC coefficient which is the first coefficient in the matrix. It is followed by the pair of numbers, the first of which represents the amount of zero's before the non-zero coefficient and the second one is the non-zero coefficient itself. When there are no more non-zero coefficients in a matrix the string is ended with the symbol EOB – End Of Block. Figure 6 shows the image reconstructed from the run-length

representation. This is supposed to be a lossless step, but in our program there are a very few small differences probably due to various data conversions from variables of type double to integer. Overall the image is unchanged.

Finally, the run-length symbols are represented in binary using JPEG tables. Each symbol is encoded by representing the table category it belongs to and its position in that category.

Figure 6. Image Reconstructed from Quantized DCT

Coefficients that are extracted from Run-Length Representation



Part II: JPEG 2000

With the collaboration between Digital Image Group (DIG) and International Standard Organization (ISO) JPEG 2000 group, the JPEG2000 became an ISO standard in August, 2000, "ISO 15444: JPEG 2000 Image Coding System."

JPEG 2000, instead of DCT and quantization methods, uses the new technology called ***wavelet*** compression. The general concept is as following: the whole image is first scanned with a "mother wave", giving a very general and blurring whole image, then add more and more waves to the mother wave to make the image clearer and clearer.

The technical differences between JPEG and JPEG 2000 can be roughly represented by this semaphore. Imagine DCT compression as a long wall of blocks. When you compress the image, various square holes are knocked out at one end of the wall. When you push the end of the wall to compress it, many gaps remain that need to be spackeled. That's why a lot of the times we see those ugly square spackeled patches in JPEG files. In contrast, wavelet compression creates gaps that have rough, uneven edges. When you push one end of the wall, the edges mesh together but with fewer noticeable gaps being created, and the block effects in JPEG is eliminated very much in JPEG 2000.

JPEG 2000 has many advantages over JPEG, some obvious ones such as:

- Better image quality at the same file size (see following examples)
- Good image quality even at very high compression ratios
- Progressive transmission by pixel accuracy and resolution (Scalability)
- Error resilience.
- Default color space (for replacement of nonspecified RGB colors).
- And many more!

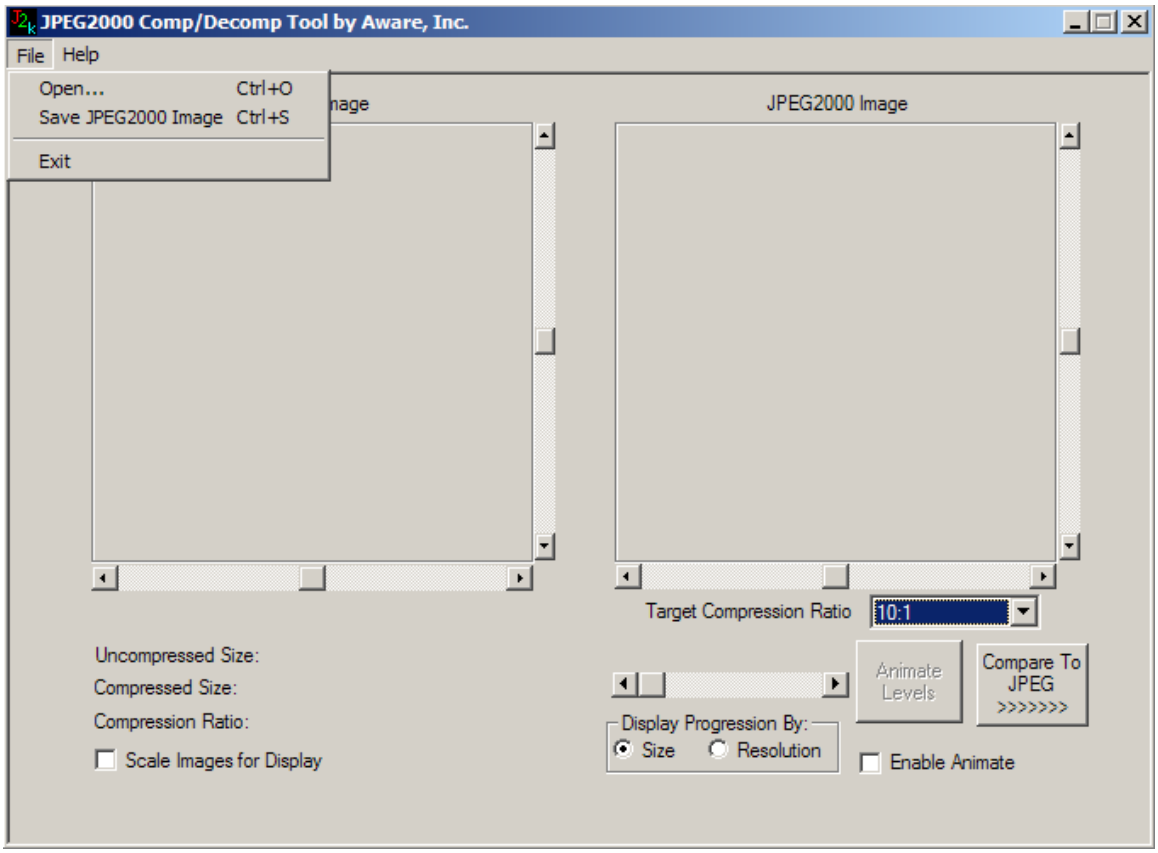
Let’s take a look at the image comparison between JPEG and JPEG 2000.

At compression ratio = 20:1 (above), both files are 32 KB, and the images are not very different from each other. At compression ratio = 100: 1 (below), if both files are kept at the same file size, it is obvious that there are “block effects” in the JPEG image because JPEG use DCT on blocks, not on the whole picture.

Let’s look another example. In the following images, the original image requires 3 Mega bytes to be represented. In the JPEG 2000 file, even at a compression ratio of 158:1, only 19 Kilo bytes are needed, while keeping almost the same image quality. However, with the same file size, JPEG file presents a very bad image quality, with a lot of block effects around the background part.

One of the special features of JPEG 2000 is called "Progressive Transmission by pixel accuracy." As described by LuraTech**, "**Progressive transmission allows images to be reconstructed with different resolutions and pixel accuracy, as needed or desired, for different target devices. The image architecture of JPEG2000 provides for efficient delivery of image data in Internet and client/server applications." The following picture is an illustration of this feature. At time 1, the image is not very clear because only certain waves are represented. As more and more waves are added, the image can become more and more clearer.

Since JPEG 2000 has not an ISO standard for long time, there are not a few companies providing the encoding software. The software I used is downloaded from Aware.com, a very simple tool to compare the compressed image using JPEG and JPEG 2000 standards. The interface is as following,



The software allows user to open an image, do the compression with the user's target compression ration (from 10:1 to 120:1), and finally save the compressed image. It also displays to the user, with the same compression ratio, the compressed JPEG 2000 image and JPEG image, along with the actual compression ratio and sizes of the compressed files.

Summary

JPEG 2000 has more complicated compression methods, so it will be very difficult to work out the simulation within this course's scope. However, since JPEG 2000 has more advantages over JPEG, it is expected that JPEG 2000 will become more and more popular in the coming years and should gradually replace the dominant weight of JPEG.

References

Books:

Compressed Image File Formats: JPEG, PNG, Gif, XBM, BMP(ACM Press), John Miano, Addison – Wesley Pub Co., August 19, 1999.

Websites:

http://www.jpeg.org - General Information about JPEG

http://www.aware.com - JPEG vs. JPEG2000 software demo

http://www.jpeg2000info.com - Information about JPEG2000 and many sample images