Mathematics Behind Image Compression

Stefany Franco[1], Dr. Tanvir Prince[1], Ildefonso Salva[2], and Charlie Windolf[3]

Image compression is fundamental to NASA and the world's daily operations. Images are transmitted to NASA from satellites and even Mars, making it very important to send data as efficiently as possible through the low-bandwidth links to these locations. This project focuses its studies in three areas. First, a hands-on mathematical analysis of the singular value decomposition (SVD) compression. Second, on the area of two field experiments that explore the effect of light conditions, shot composition and content, as well as the time of day and other variables on the file sizes of images generated in a digital camera that implements JPEG compression. Third, is about an in-depth study of the JPEG algorithm. In the SVD study, the team analyzed mathematically how matrices are manipulated to return to its equivalent original matrix and the theory about SVD is reinforced by using the software Wolfram Mathematica to compress images from NASA satellites and Mars rover. Mathematica analyzed the file size and timing data for the compression process. In the field experiment, a camera with fixed focus, aperture, and other shooting parameters was used to take pictures at various times of day of the same scene to see how the amount and quality of daylight influenced JPEG's ability to compress images. The same camera with the parameters still fixed was used to shoot various locations, indoors and outdoors, at the same time of day to see how the content of the photo influenced JPEG file sizes. Finally, the team looked at JPEG's compression algorithm using Wolfram Mathematica to better understand its efficiency and power, since NASA's radiation-hardened computer processors are generally not powerful enough to compress images with JPEG. Loosely, the team found that JPEG is best able to compress images with little variation pixel to pixel in color or brightness, and that it provides better looking images at the same file size than SVD compression.

Keywords: Image Compression; compression ratio; Singular Value Decomposition (SVD); JPEG; Decompression; Huffman Coding; Discrete Cosine Transform; quantization.

Introduction

This research project is in connection to computer science and information theory. When something is compressed, it simply means information is stored in fewer bits than the original data. Compression can be classified in two categories: lossless or lossy. Lossless compression is when the data is compressed but the original information remains intact. Lossy compression however, is when the information is compressed but some data is lost along the process. Compression is useful because it can save storage space and also increases transmission capacity.

The team focuses on three areas of study namely: the exploration and manipulation of the mathematical algorithm of SVD and hands-on experiment of image compression using SVD with the help of Wolfram Mathematica program; the actual two field experiments using digital camera with fixed settings to determine if there are changes in the files size of the same image from 9 AM to 2:00 PM with 30 minutes interval and to determine the files sizes of different images done at fixed time of the day; and the analysis of the mathematical algorithm and hands-on experiment of JPEG compression using Wolfram Mathematica program.

SVD is applicable to image compression using the principles and operation of matrices. The image can be represented by a matrix of m by n size and can be decomposed into three matrices. The result of the multiplication of these matrices will reconstruct the original image when all singular values in decreasing order are utilized. In order to compress the image, the first few singular values are sufficient to produce a reduced file size of an image but still preserving the important elements of the said image. The more detail procedure can be seen on (Cooper & Lorenc, 2006). This principle of SVD compression using Wolfram Mathematica program is significant for storing the digital files and transmission of information to NASA and to the unman robot from Mars.

Two field experiments have to be conducted to infer whether the JPEG file size of an image has relation with time and various locations both indoors and outdoors provided that the settings of the digital camera are constant. The results of these two field experiments will be analyzed and will be illustrated through scatter plots and bar graphs.

The JPEG field experiments' outcomes will serve as a motivation to further the study of JPEG compression. Moreover, the team will examine the algorithm of JPEG compression by transforming mathematically the given image and apply Wolfram Mathematica to perform JPEG compression. To learn more about the mathematica software, please see (Purdue University) and (Wolfram Mathematica, 2013).

Importance of compression

Data is something very useful in the present stage. Basically everybody has access to it, some however need it more than others. Some can be threatened by the amount of space needed and their available budget for it. Others are threatened not only by the budget, but by available technology and human patience. With this in mind many will rely on data compression as a way to meet certain requirements and affordable data handling.

Image/Video data compression is a very critical technology for many operations in NASA (White). The goal of NASA is not only to lower the data amount by compressing it, because this will save money and save space. They also want to improve the time it takes to access that data as some people might need the data to be available right away, or their real time science may demand it.

1. Department of Mathematics, Hostos Community College, City University of New York. Bronx, NY 10451
2. Mott Haven Village Preparatory High School. Bronx NY 10466
3. Collegiate School. New York, NY 10024

In NASA, image compression is used for three main reasons. First, compression saves space. NASA receives millions of bits of data each day that require a huge storage facility. NASA also has two or more backups for all the information they have. By using compression, NASA saves an enormous amount of hard drive space. Second, image/video compression saves transmission time. For example, the NASA Mars Rovers sends back pictures and data which can take up to years to reach Earth if uncompressed due to the massive distance between the two. Distance also has a direct relationship with transmission rate, for example Mars is 3.74×10^8 km away from Earth, as opposed to the moon that is only 4.05×10^3 km. Lastly, compression saves money by saving hard drive space and time (Rahman Z.).

NASA identified various lunar/Mars mission requirements that involve transmission of image/video, these can be categorized into several types; high rate video, edited high rate video, low rate video, science imaging data, and telerobotics video.

Some of the image/video data mentioned before can benefit greatly from compression because it would take up less space and therefore can be transmitted faster. Other image/video data such as scientific data and telerobotics videos are very valuable and irreplaceable, so NASA is reluctant to consider any type of compression on these (Group).

Mathematics behind Singular Value Decomposition (SVD): Here we will only summarize the known result. Readers are encouraged to consult any introductory books on linear algebra, for example, (Strang, 2009). A nice chronological history of "Singular Value Decomposition" can be found on (Stewart, 1993). SVD is based on a theorem from linear algebra that says that a rectangular matrix, "A", can be decomposed into the product of three matrices.

$$A_{m,n} = \left(U_{m,m}\right)\left(D_{m,n}\right)\left(V^T_{n,n}\right) = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

Where:

- $A_{m,n}$ is a given matrix that represents an image
- $U_{m,m}$ is an orthogonal matrix wherein the columns of matrix U are the orthonormal eigenvectors of AA^T
- $V_{n,n}{}^T$ is the transpose of an orthogonal matrix V wherein the columns of matrix V are the orthonormal eigenvectors of A^TA.
- $D_{m,n}$ is a diagonal matrix wherein the diagonal elements are singular value, σ_i, equal to the square root of the eigenvalue associated with the eigenvectors u_i *and* v_i in descending order. $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$

Mathematica Lab experiment for SVD compression

In this experiment, the team used Wolfram Mathematica to study the effects of SVD compression on an image. SVD compression uses singular value decomposition of matrices to reduce the amount of information stored in an image so that it can be stored in much less space than the original image was. In singular value decomposition, a matrix – call it A – is decomposed into a series of coefficients multiplied by two other matrices derived from A. These coefficients are that matrix A's singular values. The series of singular values is arranged in decreasing order, so that the first singular values contain more information about the original matrix A than the later values.

This decomposition can be used to compress images. Images are represented as matrices inside computers. Since the first singular values contain more information than the later ones, by taking only the first few singular values it is possible to reduce the amount of information contained in an image and compress that image (Image Compression, 2011).
In Mathematica, we took an image of Hyperion, a small moon of Saturn, displayed below (Spacetelescope, 2013) and (Space):

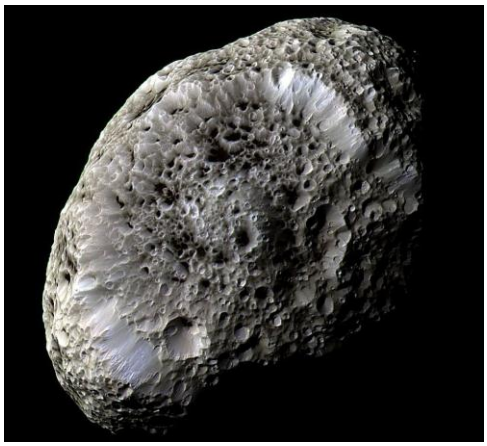

Figure 1: Hyperion image - a small moon of Saturn

We then used Mathematica to compress the image with SVD, varying the number of coefficients. A series of compressed picture using increasing number of singular values are given below, where we specify the number of singular value coefficients.

Readers are encouraged to study these pictures carefully and see the increasing quality of the picture as we increased the number of singular values.

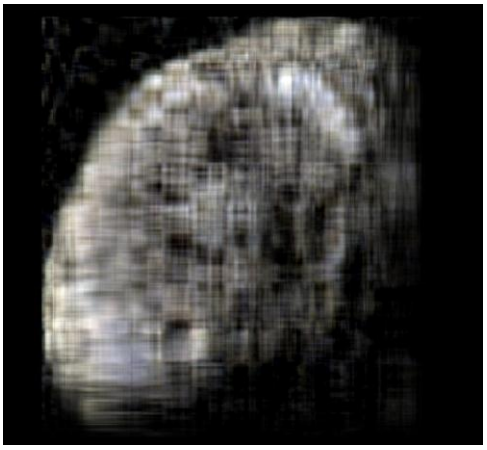

Figure 2: SVD with 10 coefficients

Figure 5: SVD with 40 coefficients

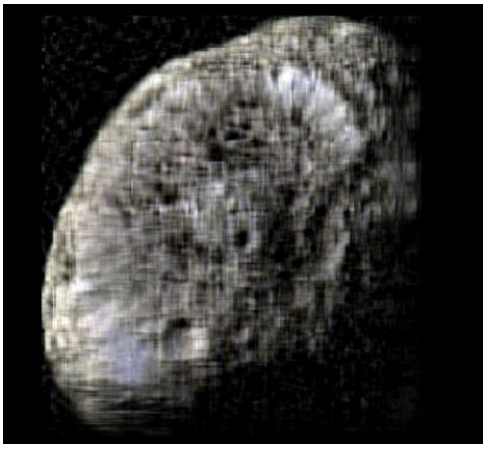

Figure 3: SVD with 20 coefficients

Figure 6: SVD with 50 coefficients

Figure 4: SVD with 30 coefficients

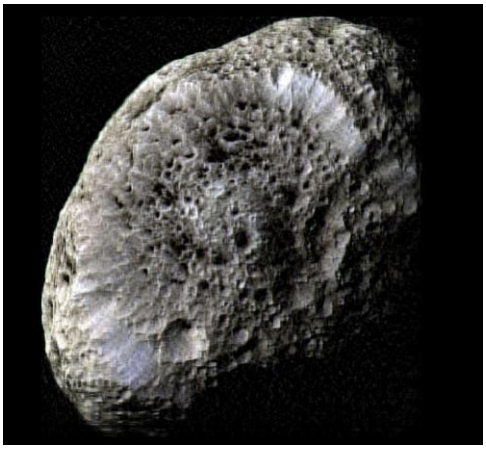

Figure 7: SVD with 60 coefficients

Figure 8: SVD with 70 coefficients

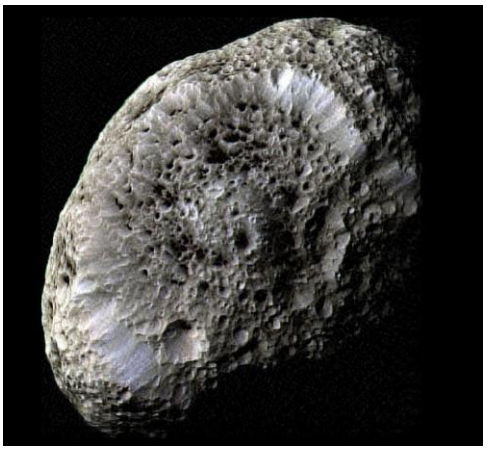

Figure 9: SVD with 80 coefficients

Figure 10: SVD with 90 coefficients

Figure 11: SVD with 100 coefficients

SVD Compression Data and Conclusions

The team found a decreasing exponential trend in compression ratio (original file size divided by the compressed file size) as the number of singular values increased. That trend makes sense because when all of the singular values are used, the resulting matrix is identical to the original matrix, and the resulting image is the same as the original image. They should thus have the same file size. In other word, as we take more singular values, we are incorporating more information about the image and thus the file size should increase. When we look at figure 14, we see this very clearly. Although it is not very clear that why this decreasing trend is exponential and not linear. A partial explanation might be that the computation complexity of the singular value decomposition is not in polynomial time and thus we see an exponential decay rather than a linear decay.

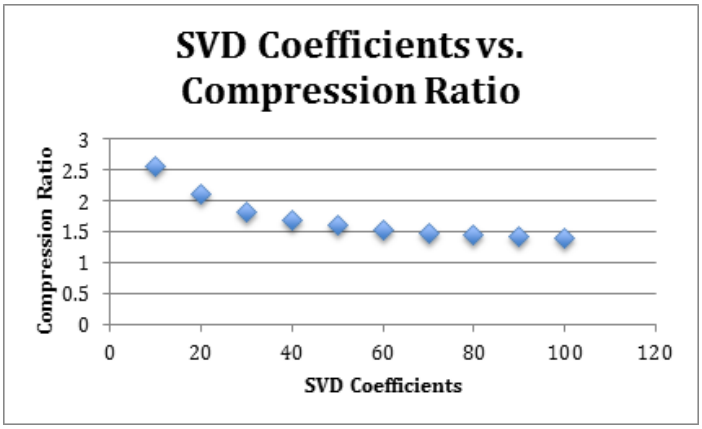

Figure 12: SVD coefficients vs. compression ratio

There was no trend found in how long it took to compress an image versus the number of coefficients used. The team speculated that this could be caused by Mathematica's SVD algorithm, which might calculate every singular value and then pick the ones requested. There might be some other reason behind it including the CPU performance of a specific computer, other task going on

behind the scene (for example, virus scan), the strength of internet connection on the specific place etc. Thus it is difficult for the team to exactly pin point the reason of the abnormality of the figure 15. The team plans to do a follow up on this topic in the next summer.

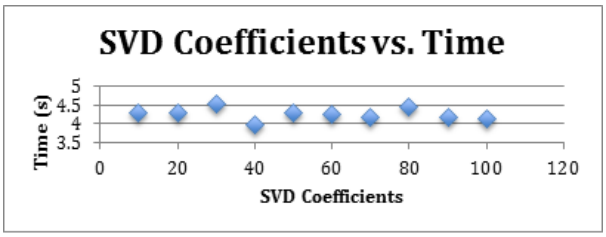

Figure 13: SVD coefficients vs. time

The team concluded that SVD is very good at reducing file sizes, but for very low file sizes (the compression ratio is high), it often looks worse than JPEG compressed images. As a compression algorithm, it is not often used for this reason. Another reason for this algorithm not being implemented in practice is the long time of computation.

Field Experiment

Objective: The objective for the two field experiments was to discover what makes JPEG compression efficient. Specifically, by shooting many pictures and isolating variables like time of day and location, the team attempted to see what kinds of lighting and shot composition yielded photographs that JPEG was able to compress to a higher degree.
Materials: digital camera with manual settings
Experiment 1: Time of Day vs. File Size
In Experiment 1, pictures were taken of one object from the same angle and position and with the camera's shooting parameters (focal length, shutter speed, aperture, ISO, etc.) all fixed. Pictures were taken every half hour and later imported to a computer where their file sizes were plotted against time. Through this procedure, a general trend was obtained for how the quality of light produced by the time of day affected the file size of the image after JPEG compression.

Figure 14: 9 AM image

Figure 16 was the first one taken, and the camera's shooting parameters were optimized for this level of light. Therefore, the picture came out very clearly and with visible detail. The file size was 4.4 megabytes.

Figure 15: noon time image

Since the shooting parameters were optimized during lower light conditions, Figure 17 was washed out by the bright noon sunlight. That element made the image less clear and less detailed. The file size was 3.6 megabytes.

Observations for Experiment 1:

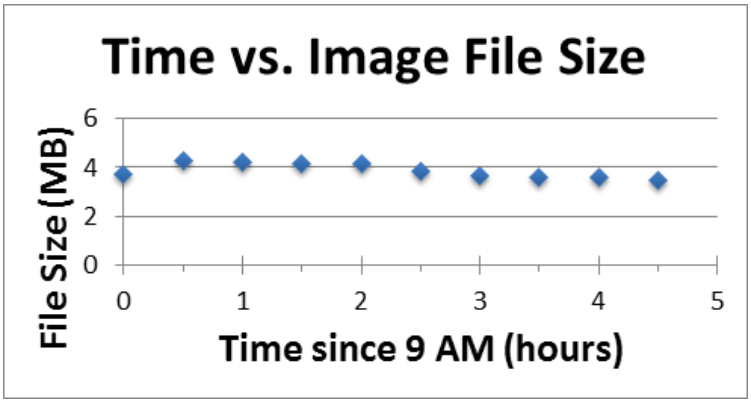

Figure 16: Time vs. image size

Figure 18 represent a graph of time versus file sizes. There is somewhat of a general trend, with the highest file sizes around 10 AM and the lowest file sizes around 1 PM.
All of the images were the same resolution (3000 by 4000 pixels for a total of 12 megapixels). Therefore, their raw bitmap file sizes should have been the same, 12 megapixels multiplied by three bytes per pixel (one byte for each of the

red, green, and blue channels) or 36 megabytes. JPEG stayed true to its advertised compression ration of about 10 to 1, producing a range of file sizes all between 3.5 and 4.5 megabytes.

Looking at the data, the team concluded that the lower file sizes around noon and 1 PM were produced by the "washing out" effect caused by bright light conditions and fixed camera parameters optimized for darker scenes. That effect yielded less dynamic range and less detail in the resulting images, which aligned neighboring pixel values more closely and allowed JPEG to compress the image more efficiently. JPEG was unable, however, to do much with the shots around 9 and 10 AM, since the dynamic range was large and neighboring pixel values often varied largely due to the shadows in the leaves of the tree. Therefore, JPEG's algorithm was not very efficient as it relies on the similarity of neighboring pixels.

Experiment 2: Location vs. File Size

Experiment 2 was similar in goal and procedure to Experiment 1 except for a change in the independent variable. Instead of fixing the location and angle and varying the time of day, we varied the location and fixed the time of day. At around 10:30 the team shot 23 photos in various conditions: indoors and outdoors, dark and light, green and gray. The photographs were imported to a computer where their file sizes were analyzed qualitatively against the shot's content. Since the data was collected differently than that of Experiment 1, there was no quantitative trend available for analysis, but the team was able to make some observations by comparing the pictures with the highest file sizes to the pictures with lower file sizes.

Figure 17: Outdoor park image (same time of the day)

Figure 19 had the largest file size, 4.4 megabytes. Note the dynamic range and varied bright and shadowy regions and the overall clarity of the shot.

Figure 18: Outdoor street image (same time of the day)

Figure 20 comes in the middle of the pack, with a 2.6 megabyte file size. Note how washed out the sky and sidewalk are and the presence of detail in the trees, buildings, and cars.

Figure 19: Indoor image (same time of the day)

Figure 21 has a 2 megabyte file size, putting it near the bottom of the group in file size. The obscurity of the shot cut out a lot of the detail and made many of the pixels, particularly in the lower right of the photograph, quite close to each other in color and brightness.

Figure 20: Outdoor sky image (same time of the day)

Figure 22 had the lowest file size, 1 megabyte. Since it is a shot of the sky, all of the pixels have values that are quite close to each other.

Observations for Experiment 2:

Figure 21: Image file sizes

Looking at the different images and making observations similar to those above, a conclusion was drawn that corroborated the conclusion from Experiment 1. Since the images with the smallest file sizes had the least detail and dynamic range, and since the images with the highest file size tended to have more detail and range, it seemed again that more detail made it more difficult to compress the image with JPEG. More specifically, if the values of neighboring pixels were not similar to each other, the algorithm was less effective at compressing that pixel neighborhood.

So in conclusion, we can say that fewer variations in color throughout an image will results higher compression ratio and still retain the important features of the image. On the other hand, the more details the picture is (with a lot of color changes from pixels to pixels), JPEG takes more file size to attain the compression ratio.

Experiment Using JPEG

Brief History of JPEG

In 1992, JPEG became an international standard for compressing digital still images. The acronym JPEG comes from the *Joint Photographic Experts Group*. JPEG was formed in the 1980's by members from the International Organization for Standardization (ISO) and the International Telecommunications Union (ITU). Over 80% of all images that are transmitted via the internet are stored using the JPEG standard. Despite the popularity of the standard, JPEG members quickly identified some issues with the format and also compiled a list of enhancements that should be included in the next generation of the format (Math is Fun).

In computing, JPEG is a commonly used method of lossy compression for digital photography (image). The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality.

JPEG compression is used in a number of image format. JPEG/Exif is the most common image format used by digital cameras and other photographic image capture devices; along with JPEG/JFIF, it is the most common format for storing and transmitting photographic images on the world wide web (www).

JPEG can be used to compress a digital still image. There are four basic steps in JPEG compression algorithm. (Mathematics, 2011)

1. Preprocessing
2. Transformation
3. Quantization
4. Encoding

How JPEG works?

- JPEG Compression breaks an image into a series of square patches of pixels or in other words 8x8 blocks.Then you substract 127 from each pixel intensity in each block. (Preprocessing)
- A two dimensional Discrete Fourier Cosine Transform is applied to each patch. (transformation)
- Transform coefficients that are very small in magnitude will have very little affect on the image and are, therefore, set to zero. (Quantization)
- The coefficients are then reduced in size by applying a compression algorithm. This compression algorithm is Hoffman Coding. (Encoding)
- When restoring the image, the image file is decompressed and each patch is sent through an inverse Discrete Fourier Cosine Transform. (Inverse Process)

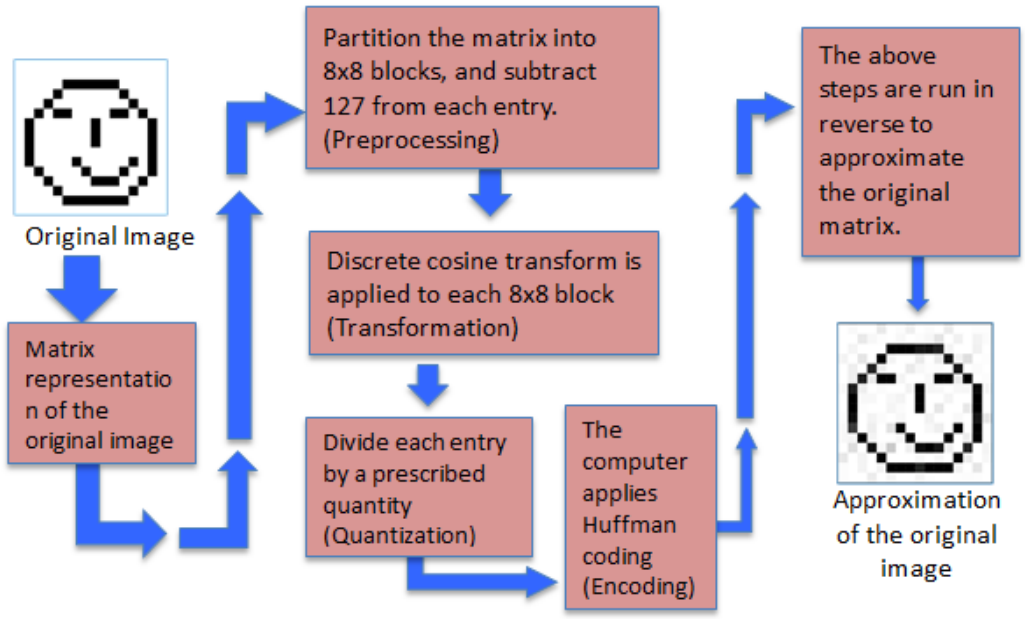

Figure 22: Block diagram of JPEG compression algorithm

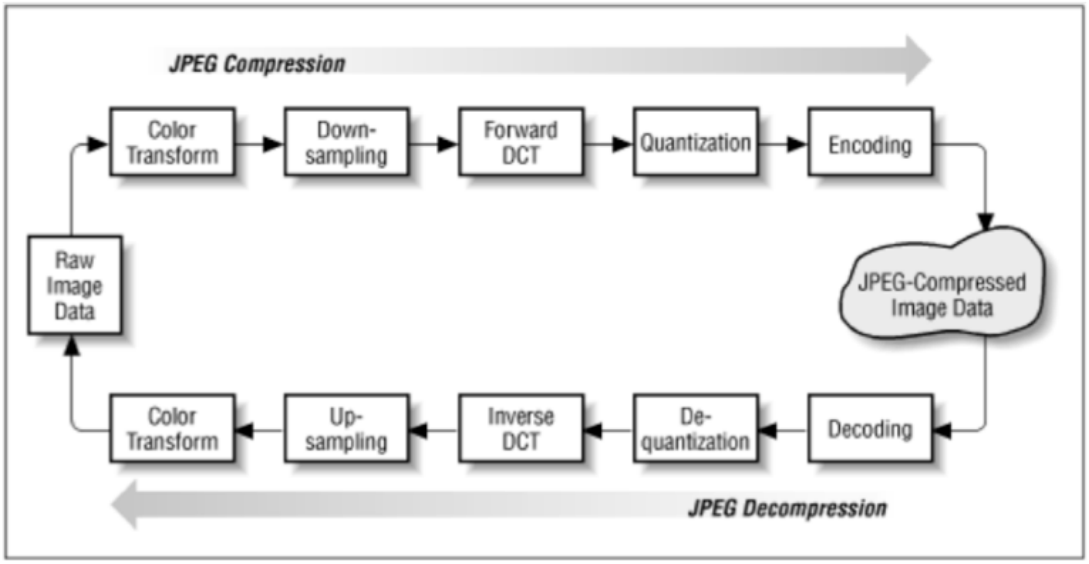

Figure 23: JPEG Compression-Decompression Algorithm

The following block diagrams will be explain in detail using Wolfram Mathematica. Example 1 (The detail explanation on the JPEG compression on the happy face) JPEG compression Algorithm of an artificial image “smiley face” using Wolfram Mathematica Program. As we proceed with the description, all the necessary commands of mathematica are also given in the appropriate places. The following image (black and white only using 0 and 1) is created using "Image" command. The size is 16 by 16.

```
a = Image[{{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
{1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1},
{1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1},
{1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1},
{1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1},
{1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1},
{1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1},
{1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1},
{1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1},
{1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1},
{1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1},
{1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1},
{1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1},
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}]
```

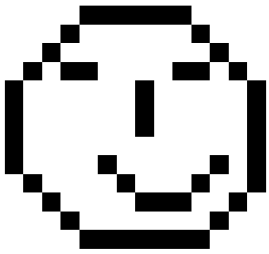

Figure 24: Artificial "smiley" image

Let us convert this image to a matrix - this will be a 16 by 16 matrix whose entries are 0 and 1

```
b = ImageData[a];
ImageData[a] // MatrixForm
```

1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.
1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.
1.	1.	1.	1.	1.	0.	0.	0.	0.	0.	0.	1.	1.	1.	1.	1.
1.	1.	1.	1.	0.	1.	1.	1.	1.	1.	1.	0.	1.	1.	1.	1.
1.	1.	1.	0.	1.	1.	1.	1.	1.	1.	1.	1.	0.	1.	1.	1.
1.	1.	0.	1.	0.	0.	1.	1.	1.	1.	0.	0.	1.	0.	1.	1.
1.	0.	1.	1.	1.	1.	1.	1.	0.	1.	1.	1.	1.	1.	0.	1.
1.	0.	1.	1.	1.	1.	1.	1.	0.	1.	1.	1.	1.	1.	0.	1.
1.	0.	1.	1.	1.	1.	1.	1.	0.	1.	1.	1.	1.	1.	0.	1.
1.	0.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	0.	1.
1.	0.	1.	1.	1.	1.	0.	1.	1.	1.	1.	1.	0.	1.	0.	1.
1.	1.	0.	1.	1.	1.	1.	0.	1.	1.	1.	0.	1.	1.	0.	1.
1.	1.	1.	0.	1.	1.	1.	1.	0.	0.	0.	1.	1.	0.	1.	1.
1.	1.	1.	1.	0.	1.	1.	1.	1.	1.	1.	1.	0.	1.	1.	1.
1.	1.	1.	1.	1.	0.	0.	0.	0.	0.	0.	0.	1.	1.	1.	1.
1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.

Preprocessing:

First we need to convert all the entry between 0 and 255. This is accomplished by using the "Byte" command. Recall that there are two scale of color - one is given by value between 0 and 1 and the other is given by value between 0 and 255 (this one only use integer).

```
c = ImageData[a, "Byte"];
ImageData[a, "Byte"] // MatrixForm
```

255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	0	0	0	0	0	0	255	255	255	255	255
255	255	255	255	0	255	255	255	255	255	255	0	255	255	255	255
255	255	255	0	255	255	255	255	255	255	255	255	0	255	255	255
255	255	0	255	0	0	255	255	255	255	0	0	255	0	255	255
255	0	255	255	255	255	255	255	0	255	255	255	255	255	0	255
255	0	255	255	255	255	255	255	0	255	255	255	255	255	0	255
255	0	255	255	255	255	255	255	0	255	255	255	255	255	0	255
255	0	255	255	255	255	255	255	255	255	255	255	255	255	0	255
255	0	255	255	255	255	0	255	255	255	255	255	0	255	0	255
255	255	0	255	255	255	255	0	255	255	255	0	255	255	0	255
255	255	255	0	255	255	255	255	0	0	0	255	255	0	255	255
255	255	255	255	0	255	255	255	255	255	255	255	0	255	255	255
255	255	255	255	255	0	0	0	0	0	0	0	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Now we need to make the data symmetric around origin - this is done by subtracting 127 from each entry of the matrix. After this all entry of the matrix will be between -128 and 128.

```
u[x_] := x - 127;
d = Map[u, c];
Map[u, c] // MatrixForm
```

$$\begin{pmatrix}
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & -127 & -127 & -127 & -127 & -127 & -127 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\
128 & 128 & -127 & 128 & -127 & -127 & 128 & 128 & 128 & 128 & -127 & -127 & 128 & -127 & 128 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 & -127 & 128 \\
128 & 128 & -127 & 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 & -127 & 128 & 128 & -127 & 128 \\
128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 & -127 & -127 & -127 & 128 & 128 & -127 & 128 & 128 \\
128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & -127 & -127 & -127 & -127 & -127 & -127 & -127 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128
\end{pmatrix}$$

Finally we need to partition the matrix into 8 by 8 submatrix. In this case, since the matrix is 16 by 16, we will get 4 pieces of 8 by 8 submatrix. If the dimension of the matrix is not divisible by 8, then we "pat" the matrix at the end - this means add minimum number of 0 so that the dimension will be divisible by 8. To partition a matrix, we use "Partition" command.

```
{e, f} = Partition[d, {8, 8}];
Partition[d, {8, 8}] // MatrixForm
```

$$\begin{pmatrix}
\begin{pmatrix}
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & -127 & -127 & -127 \\
128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\
128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 \\
128 & 128 & -127 & 128 & -127 & -127 & 128 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128
\end{pmatrix} &
\begin{pmatrix}
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
-127 & -127 & -127 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\
128 & 128 & -127 & -127 & 128 & -127 & 128 & 128 \\
-127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\
-127 & 128 & 128 & 128 & 128 & 128 & -127 & 128
\end{pmatrix} \\
\begin{pmatrix}
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & -127 & 128 \\
128 & 128 & -127 & 128 & 128 & 128 & 128 & -127 \\
128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & -127 & -127 & -127 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128
\end{pmatrix} &
\begin{pmatrix}
-127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\
128 & 128 & 128 & 128 & -127 & 128 & -127 & 128 \\
128 & 128 & 128 & -127 & 128 & 128 & -127 & 128 \\
-127 & -127 & -127 & 128 & 128 & -127 & 128 & 128 \\
128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\
-127 & -127 & -127 & -127 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128
\end{pmatrix}
\end{pmatrix}$$

To separate these four submatrix, we use four different name for four submatrix.

```
e1 = e[[1]];
e2 = e[[2]];
f1 = f[[1]];
f2 = f[[2]];
e[[1]] // MatrixForm
e[[2]] // MatrixForm
f[[1]] // MatrixForm
f[[2]] // MatrixForm
```

$$\begin{pmatrix}
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & -127 & -127 & -127 \\
128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\
128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 \\
128 & 128 & -127 & 128 & -127 & -127 & 128 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & -127 & 128 & 128 & 128 & 128 & 128 & 128
\end{pmatrix}$$

$$\begin{pmatrix} 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\ -127 & -127 & -127 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\ 128 & 128 & -127 & -127 & 128 & -127 & 128 & 128 \\ -127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\ -127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \end{pmatrix}$$

$$\begin{pmatrix} 128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & -127 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & -127 & 128 & 128 & 128 & 128 & -127 & 128 \\ 128 & 128 & -127 & 128 & 128 & 128 & 128 & -127 \\ 128 & 128 & 128 & -127 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & -127 & -127 & -127 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \end{pmatrix}$$

$$\begin{pmatrix} -127 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & -127 & 128 \\ 128 & 128 & 128 & 128 & -127 & 128 & -127 & 128 \\ 128 & 128 & 128 & -127 & 128 & 128 & -127 & 128 \\ -127 & -127 & -127 & 128 & 128 & -127 & 128 & 128 \\ 128 & 128 & 128 & 128 & -127 & 128 & 128 & 128 \\ -127 & -127 & -127 & -127 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \end{pmatrix}$$

Transformation:

First consider the submatrix e1 (the upper left hand corner). The process is exactly same for all four so we will just explain one in detail. This e1 is an 8 by 8 matrix. The transformation is to multiply the e1 as follows: U times e1 times U transpose where U is the 8 by 8 DCT matrix. This U will be an orthogonal matrix. This is a fixed matrix. There is only one 8 by 8 DCT matrix (although a few variation exist the one below is the most common one). Instead of U, we name it U8:

```
g[k_] := Piecewise[{{1, k < 0}, {1, k > 0}}, 1/Sqrt[2]];
m = 8;
U8 = Table[(4/m)*g[k] Cos[((2*n + 1)*k*Pi)/(2*m)], {k, 0, m - 1}, {n, 0, m - 1}];
V8 = Table[(4/m)*g[k] Cos[((2*n + 1)*k*Pi)/(2*m)], {k, 0, m - 1}, {n, 0, m - 1}] // MatrixForm
```

$$\begin{pmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2}\cos\left[\frac{\pi}{16}\right] & \frac{1}{2}\cos\left[\frac{3\pi}{16}\right] & \frac{1}{2}\sin\left[\frac{3\pi}{16}\right] & \frac{1}{2}\sin\left[\frac{\pi}{16}\right] & -\frac{1}{2}\sin\left[\frac{\pi}{16}\right] & -\frac{1}{2}\sin\left[\frac{3\pi}{16}\right] & -\frac{1}{2}\cos\left[\frac{3\pi}{16}\right] & -\frac{1}{2}\cos\left[\frac{\pi}{16}\right] \\ \frac{1}{2}\cos\left[\frac{\pi}{8}\right] & \frac{1}{2}\sin\left[\frac{\pi}{8}\right] & -\frac{1}{2}\sin\left[\frac{\pi}{8}\right] & -\frac{1}{2}\cos\left[\frac{\pi}{8}\right] & -\frac{1}{2}\cos\left[\frac{\pi}{8}\right] & -\frac{1}{2}\sin\left[\frac{\pi}{8}\right] & \frac{1}{2}\sin\left[\frac{\pi}{8}\right] & \frac{1}{2}\cos\left[\frac{\pi}{8}\right] \\ \frac{1}{2}\cos\left[\frac{3\pi}{16}\right] & -\frac{1}{2}\sin\left[\frac{\pi}{16}\right] & -\frac{1}{2}\cos\left[\frac{\pi}{16}\right] & -\frac{1}{2}\sin\left[\frac{3\pi}{16}\right] & \frac{1}{2}\sin\left[\frac{3\pi}{16}\right] & \frac{1}{2}\cos\left[\frac{\pi}{16}\right] & \frac{1}{2}\sin\left[\frac{\pi}{16}\right] & -\frac{1}{2}\cos\left[\frac{3\pi}{16}\right] \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2}\sin\left[\frac{3\pi}{16}\right] & -\frac{1}{2}\cos\left[\frac{\pi}{16}\right] & \frac{1}{2}\sin\left[\frac{\pi}{16}\right] & \frac{1}{2}\cos\left[\frac{3\pi}{16}\right] & -\frac{1}{2}\cos\left[\frac{3\pi}{16}\right] & -\frac{1}{2}\sin\left[\frac{\pi}{16}\right] & \frac{1}{2}\cos\left[\frac{\pi}{16}\right] & -\frac{1}{2}\sin\left[\frac{3\pi}{16}\right] \\ \frac{1}{2}\sin\left[\frac{\pi}{8}\right] & -\frac{1}{2}\cos\left[\frac{\pi}{8}\right] & \frac{1}{2}\cos\left[\frac{\pi}{8}\right] & -\frac{1}{2}\sin\left[\frac{\pi}{8}\right] & -\frac{1}{2}\sin\left[\frac{\pi}{8}\right] & \frac{1}{2}\cos\left[\frac{\pi}{8}\right] & -\frac{1}{2}\cos\left[\frac{\pi}{8}\right] & \frac{1}{2}\sin\left[\frac{\pi}{8}\right] \\ \frac{1}{2}\sin\left[\frac{\pi}{16}\right] & -\frac{1}{2}\sin\left[\frac{3\pi}{16}\right] & \frac{1}{2}\cos\left[\frac{3\pi}{16}\right] & -\frac{1}{2}\cos\left[\frac{\pi}{16}\right] & \frac{1}{2}\cos\left[\frac{\pi}{16}\right] & -\frac{1}{2}\cos\left[\frac{3\pi}{16}\right] & \frac{1}{2}\sin\left[\frac{3\pi}{16}\right] & -\frac{1}{2}\sin\left[\frac{\pi}{16}\right] \end{pmatrix}$$

We multiply e1 with the above DCT matrix as described before:

```
T1 = N[U8.e1.Transpose[U8]];
N[U8.e1.Transpose[U8] // MatrixForm]
```

705.25	40.5692	83.2934	-22.9806	63.75	115.531	34.5013	27.1075
81.6929	177.866	-48.3388	-28.8794	-81.6929	-130.228	-68.4012	-37.1634
127.9	-131.781	-159.375	38.2056	107.689	67.0225	76.9531	58.7123
28.6867	-115.938	178.911	18.2362	-28.6867	-32.3107	-105.158	-43.8372
127.5	-115.531	48.7921	40.5692	-127.5	-27.1075	117.795	22.9806
-19.1679	25.1653	-42.873	-50.9827	19.1679	101.53	38.5057	-70.8595
-190.983	179.636	13.2031	-59.737	93.3986	1.57928	-159.375	-49.1935
-16.2497	71.6647	-147.306	85.1504	16.2497	-137.707	78.6048	212.368

This is the end of the transformation of e1. We do the same for e2, f1 and f2. We do not display all the output. The transformation of e1, e2, f1, f2 are denoted by T1, T2, T3 and T4.

```
T2 = N[U8.e2.Transpose[U8]];
T3 = N[U8.f1.Transpose[U8]];
T4 = N[U8.f2.Transpose[U8]];
```

Quantization:

First consider T1 (the other are similar). The main idea is to ignore the lowest value. But how can we decide which one is lower and which one is not? This is why there is a fixed 8 by 8 luminance matrix (we call this Z in here) - you may call it a quantization matrix. This matrix is given below:

```
Z = {{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}};
Z // MatrixForm
```

The quantization matrix is designed to provide more resolution to more perceivable frequency components over less perceivable components (usually lower frequencies over high frequencies) in addition to transforming as many components to 0, which can be encoded with greatest efficiency.

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

Each element of T1 is divided by the corresponding element of Z and round to the nearest integer. This is the quantization process. The quantization of T1 is denoted by qT1, the quantization of T2 is denoted by qT2 etc.

```
qT1 = Table[Round[T1[[s]][[t]]/Z[[s]][[t]]], {s, 1, 8}, {t, 1, 8}];
qT1 // MatrixForm
```

44	4	8	-1	3	3	1	0
7	15	-3	-2	-3	-2	-1	-1
9	-10	-10	2	3	1	1	1
2	-7	8	1	-1	0	-1	-1
7	-5	1	1	-2	0	1	0
-1	1	-1	-1	0	1	0	-1
-4	3	0	-1	1	0	-1	0
0	1	-2	1	0	-1	1	2

qT2 = Table[Round[T2[[s]][[t]]/Z[[s]][[t]]], {s, 1, 8}, {t, 1, 8}];
qT2 // MatrixForm

$$\begin{pmatrix} 40 & -12 & 0 & -3 & 0 & -4 & 0 & -1 \\ 14 & -5 & 4 & 7 & 0 & 3 & 0 & 1 \\ 5 & 4 & -15 & -4 & 1 & -2 & 1 & -1 \\ 4 & 9 & 10 & 1 & 0 & 1 & -1 & 1 \\ 7 & 5 & 1 & -1 & -2 & 0 & 1 & 0 \\ -2 & -1 & -1 & 0 & 0 & -1 & 0 & 1 \\ -3 & -2 & 1 & 1 & 1 & 0 & -1 & 1 \\ 0 & -1 & -2 & -1 & 0 & 1 & 1 & -2 \end{pmatrix}$$

qT3 = Table[Round[T3[[s]][[t]]/Z[[s]][[t]]], {s, 1, 8}, {t, 1, 8}];
qT3 // MatrixForm

$$\begin{pmatrix} 42 & 5 & -5 & 5 & 4 & 2 & 2 & 1 \\ 0 & -18 & -5 & 4 & 5 & 2 & 3 & 1 \\ 5 & -2 & -5 & -5 & 3 & 1 & 1 & 1 \\ 1 & -1 & 6 & -3 & -2 & 1 & 0 & 0 \\ 5 & -5 & 1 & 3 & -1 & 0 & -1 & 1 \\ -7 & 6 & -3 & 1 & 1 & 0 & -1 & -1 \\ 2 & -2 & 1 & -1 & 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & -1 & 1 & -1 & 1 & 2 \end{pmatrix}$$

qT4 = Table[Round[T4[[s]][[t]]/Z[[s]][[t]]], {s, 1, 8}, {t, 1, 8}];
qT4 // MatrixForm

$$\begin{pmatrix} 32 & -8 & -1 & -5 & 3 & -4 & 3 & -2 \\ 1 & 19 & -7 & -6 & 2 & -3 & 2 & -2 \\ 9 & -1 & -8 & 0 & -1 & -1 & 0 & -1 \\ -8 & -11 & -4 & -2 & 0 & 1 & -2 & 0 \\ 0 & -2 & -3 & -1 & 1 & -1 & -1 & 2 \\ -5 & -2 & 1 & 1 & -1 & -2 & 1 & 0 \\ 3 & 3 & 1 & -2 & -1 & 1 & 0 & -2 \\ -3 & -3 & -1 & 0 & 1 & 1 & -1 & 0 \end{pmatrix}$$

Coding:
Now Huffman Coding is applied. Huffman coding is a lossless compression algorithm that eliminates redundant data.

Inverting the process:

This is NOT a true inversion because some steps for example the quantization are not invertible process. But we will get an approximation of the original image. The process is simple. Consider qT1 (quantization of T1): We will multiply each entry of qT1 by the corresponding entry of Z - we will call this matrix reverseqT1: This will be an approximation of T1.
*reverseqT1 = Table[qT1[[s]][[t]]*Z[[s]][[t]], {s, 1, 8}, {t, 1, 8}];*
reverseqT1 // MatrixForm

$$\begin{pmatrix} 704 & 44 & 80 & -16 & 72 & 120 & 51 & 0 \\ 84 & 180 & -42 & -38 & -78 & -116 & -60 & -55 \\ 126 & -130 & -160 & 48 & 120 & 57 & 69 & 56 \\ 28 & -119 & 176 & 29 & -51 & 0 & -80 & -62 \\ 126 & -110 & 37 & 56 & -136 & 0 & 103 & 0 \\ -24 & 35 & -55 & -64 & 0 & 104 & 0 & -92 \\ -196 & 192 & 0 & -87 & 103 & 0 & -120 & 0 \\ 0 & 92 & -190 & 98 & 0 & -100 & 103 & 198 \end{pmatrix}$$

Now multiply this above matrix on the left by U8 transpose and on the right by U8 - this will reverse the process of transformation. Of course, we round everything to nearest integer. This will be an approximation of e1. So we call it reverseT1.

```
reverseT1 = Map[Round, N[Transpose[U8].reverseqT1.U8]];
reverseT1 // MatrixForm
```

$$\begin{pmatrix} 133 & 120 & 127 & 158 & 100 & 148 & 124 & 108 \\ 140 & 130 & 100 & 118 & 125 & 134 & 118 & 161 \\ 131 & 115 & 164 & 135 & 140 & -135 & -150 & -125 \\ 114 & 155 & 96 & 135 & -140 & 114 & 136 & 134 \\ 145 & 106 & 137 & -101 & 106 & 173 & 132 & 102 \\ 129 & 106 & -104 & 100 & -127 & -133 & 113 & 148 \\ 118 & -104 & 107 & 147 & 121 & 146 & 132 & 111 \\ 134 & -131 & 118 & 119 & 153 & 119 & 114 & 132 \end{pmatrix}$$

Finally we add 127 to each entry to make the value 0 and 255 (if the entry is below 0 or above 255, it is assumed the value 0 and 255 respectively). We call it j1.

```
p[x_] := x + 127;
j1 = Map[p, reverseT1];
j1 // MatrixForm
```

$$\begin{pmatrix} 260 & 247 & 254 & 285 & 227 & 275 & 251 & 235 \\ 267 & 257 & 227 & 245 & 252 & 261 & 245 & 288 \\ 258 & 242 & 291 & 262 & 267 & -8 & -23 & 2 \\ 241 & 282 & 223 & 262 & -13 & 241 & 263 & 261 \\ 272 & 233 & 264 & 26 & 233 & 300 & 259 & 229 \\ 256 & 233 & 23 & 227 & 0 & -6 & 240 & 275 \\ 245 & 23 & 234 & 274 & 248 & 273 & 259 & 238 \\ 261 & -4 & 245 & 246 & 280 & 246 & 241 & 259 \end{pmatrix}$$

Do the same for other but not show the output:

```
reverseqT2 = Table[qT2[[s]][[t]]*Z[[s]][[t]], {s, 1, 8}, {t, 1, 8}];
reverseT2 = Map[Round, N[Transpose[U8].reverseqT2.U8]];
j2 = Map[p, reverseT2];
j2 // MatrixForm
```

$$\begin{pmatrix} 265 & 248 & 283 & 230 & 266 & 258 & 253 & 253 \\ 251 & 243 & 220 & 279 & 237 & 292 & 238 & 249 \\ 17 & 16 & 24 & 240 & 266 & 269 & 237 & 272 \\ 229 & 275 & 227 & 16 & 238 & 214 & 263 & 253 \\ 244 & 253 & 251 & 269 & 17 & 281 & 249 & 259 \\ 281 & 244 & -7 & -4 & 226 & -7 & 276 & 251 \\ -2 & 243 & 245 & 282 & 244 & 243 & 12 & 242 \\ -10 & 262 & 260 & 242 & 263 & 258 & 0 & 257 \end{pmatrix}$$

```
reverseqT3 = Table[qT3[[s]][[t]]*Z[[s]][[t]], {s, 1, 8}, {t, 1, 8}];
reverseT3 = Map[Round, N[Transpose[U8].reverseqT3.U8]];
j3 = Map[p, reverseT3];
j3 // MatrixForm
```

$$\begin{pmatrix} 271 & -19 & 252 & 270 & 263 & 242 & 248 & 259 \\ 250 & 13 & 265 & 241 & 249 & 270 & 245 & 262 \\ 246 & 41 & 256 & 233 & 274 & 216 & 8 & 265 \\ 246 & 242 & 4 & 240 & 262 & 307 & 246 & -22 \\ 284 & 214 & 267 & 16 & 245 & 188 & 294 & 234 \\ 225 & 290 & 240 & 255 & 6 & 278 & 261 & 284 \\ 251 & 276 & 241 & 269 & 238 & -17 & -13 & -22 \\ 251 & 259 & 254 & 232 & 296 & 242 & 259 & 265 \end{pmatrix}$$

```
reverseqT4 = Table[qT4[[s]][[t]]*Z[[s]][[t]], {s, 1, 8}, {t, 1, 8}];
```

reverseT4 = Map[Round, N[Transpose[U8].reverseqT4.U8]];
j4 = Map[p, reverseT4];
j4 // MatrixForm

$$\begin{pmatrix} -12 & 261 & 224 & 276 & 259 & 226 & 5 & 265 \\ 272 & 238 & 268 & 254 & 247 & 270 & 17 & 221 \\ 256 & 245 & 233 & 257 & -1 & 267 & 0 & 274 \\ 255 & 282 & 274 & -17 & 259 & 232 & -27 & 276 \\ 12 & -53 & 17 & 235 & 274 & 26 & 284 & 239 \\ 245 & 281 & 243 & 287 & -18 & 231 & 261 & 243 \\ 25 & -20 & -11 & -2 & 238 & 304 & 223 & 266 \\ 241 & 253 & 268 & 261 & 253 & 251 & 252 & 261 \end{pmatrix}$$

Put all j1, j2, j3 and j4 into a big matrix and create the image:
j = Join[Join[j1, j2, 2], Join[j3, j4, 2]];
Join[Join[j1, j2, 2], Join[j3, j4, 2]] // MatrixForm

$$\begin{pmatrix} 260 & 247 & 254 & 285 & 227 & 275 & 251 & 235 & 265 & 248 & 283 & 230 & 266 & 258 & 253 & 253 \\ 267 & 257 & 227 & 245 & 252 & 261 & 245 & 288 & 251 & 243 & 220 & 279 & 237 & 292 & 238 & 249 \\ 258 & 242 & 291 & 262 & 267 & -8 & -23 & 2 & 17 & 16 & 24 & 240 & 266 & 269 & 237 & 272 \\ 241 & 282 & 223 & 262 & -13 & 241 & 263 & 261 & 229 & 275 & 227 & 16 & 238 & 214 & 263 & 253 \\ 272 & 233 & 264 & 26 & 233 & 300 & 259 & 229 & 244 & 253 & 251 & 269 & 17 & 281 & 249 & 259 \\ 256 & 233 & 23 & 227 & 0 & -6 & 240 & 275 & 281 & 244 & -7 & -4 & 226 & -7 & 276 & 251 \\ 245 & 23 & 234 & 274 & 248 & 273 & 259 & 238 & -2 & 243 & 245 & 282 & 244 & 243 & 12 & 242 \\ 261 & -4 & 245 & 246 & 280 & 246 & 241 & 259 & -10 & 262 & 260 & 242 & 263 & 258 & 0 & 257 \\ 271 & -19 & 252 & 270 & 263 & 242 & 248 & 259 & -12 & 261 & 224 & 276 & 259 & 226 & 5 & 265 \\ 250 & 13 & 265 & 241 & 249 & 270 & 245 & 262 & 272 & 238 & 268 & 254 & 247 & 270 & 17 & 221 \\ 246 & 41 & 256 & 233 & 274 & 216 & 8 & 265 & 256 & 245 & 233 & 257 & -1 & 267 & 0 & 274 \\ 246 & 242 & 4 & 240 & 262 & 307 & 246 & -22 & 255 & 282 & 274 & -17 & 259 & 232 & -27 & 276 \\ 284 & 214 & 267 & 16 & 245 & 188 & 294 & 234 & 12 & -53 & 17 & 235 & 274 & 26 & 284 & 239 \\ 225 & 290 & 240 & 255 & 6 & 278 & 261 & 284 & 245 & 281 & 243 & 287 & -18 & 231 & 261 & 243 \\ 251 & 276 & 241 & 269 & 238 & -17 & -13 & -22 & 25 & -20 & -11 & -2 & 238 & 304 & 223 & 266 \\ 251 & 259 & 254 & 232 & 296 & 242 & 259 & 265 & 241 & 253 & 268 & 261 & 253 & 251 & 252 & 261 \end{pmatrix}$$

Image[j, "Byte"] [Finally This will create the JPEG compressed Image of the smiley face. Since the image is artificially created with only the value of 0 and 1, we see that the compressed image has a lot of noise around the 8 by 8 blocks. But in the everyday image, the nearby pixels have approximately equal value and thus in the compressed image the noise is minimum and very hard to recognized it through naked eye.]

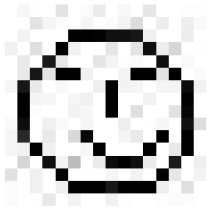

Figure 25: JPEG Compressed "smiley" image

Advantages of JPEG compression:

- Most common file used across the Web (80%).
- It can make image files smaller.
- Editing is not required to print a file in JPEG
- JPEG files can be processed in your camera.

Disadvantages of JPEG compression:

- Compression discards some data.
- Compression compromises image quality.
- JPEG does not handle line drawings well and it does not support animation.
- Compression is applied every time you save an image using JPEG.

Conclusion

This research project, which focused on three parts mentioned above, concluded that JPEG was more functional to compress images with little variation pixel to pixel in color or brightness. JPEG further generated better images at the same file size than SVD compression.

In doing SVD compression using Wolfram Mathematica program as shown in Figures 3 to 13, it was concluded that the relation between compression ratio and the singular values (SVD coefficients) was a decreasing exponential function as shown in figure 14. Also, there was no relation found in how long it took to compress an image versus the number of coefficients used as shown in figure 15.

In the SVD compression experiment, the team concluded based on Mathematica's timing data that there was no trend in compression time versus the number of SVD coefficients taken. The team speculated that this could be due to the fact that Mathematica calculates every singular value and then selects the required values, as well as due to random fluctuations in computer activity. The team also found that the fewer singular values were used, the smaller the resulting file size was. Moreover, incrementing the number of coefficients caused a greater increase in file size when the number of coefficients was small versus when it was large. As the

number of coefficients increased, the compression ratio approached one.

That Singular Value Decomposition (SVD) is an important topic in linear Algebra where students will put strong emphasis on exploring the application of image compression. This research project will encourage high school students to appreciate math by looking at the algorithm of SVD compression with the use of the properties and operation of matrices. SVD compression can be an elective math for senior students and will serve as one of the culminating activity. Students can compare and contrast how SVD can reconstruct the original image and how the Mathematica program can perform image compression. Students can select different kinds of images and apply Mathematica program in order to compress the image and measure the compression ratio.

Field experiment 1 (a graph of time versus file sizes) as shown in figure 18, revealed that there was apparently a general trend, with the highest file sizes around 10 AM and the lowest file sizes around 1 PM. Furthermore, the more detail was the image, the bigger was the file size.

Field experiment 2 (a graph of different images taken at the same time of the day) as shown in figure 23, brought a point that the images with the smallest file sizes had the least detail and the images with the highest file size seemed to have more detail.

The JPEG field experiments showed that JPEG is much better at compressing scenes that have little variation in light and tone. The image that was best compressed was an image of a uniformly blue sky, whereas JPEG had great difficulty compressing an image with dappled shadows and lots of colors. The experiment also showed that exposing an image with too much light, leading to a 'washed out' quality, could lead to smaller file sizes since some detail is eliminated.

The theoretical JPEG hands on experiment showed the problems with basic JPEG compression. There are lots of noise around the borders of the 8 x 8 blocks that the algorithm splits the image into, and there are obvious compression artifacts around edges, particularly in images of text or line drawings.

The team discovered that JPEG is more functional to compress images with little variation pixel to pixel in color or brightness. JPEG further generated better images at the same file size than SVD compression.

For future work, it may consider studying other compression programs and their algorithm used by NASA such as wavelet-based image compression like JPEG2000 and ICER.

Acknowledgement and Sponsors:

A special thank goes to Prof. Angulo Nieves of Hostos Community College for her grant to support the research. We also want to thank all of our other sponsors which include the following: New York City Research Initiative, The City University of New York, The City College of New York, Hostos Community College, United States Department of Education, Goddard Institute for Space Studies, Goddard Space Flight Center, NASA, NSF and NOAA.

This project is supported by the grant from U.S. DoE - CILES #P031C110158. This paper is presented in the research summit on Aug 1st, 2013 in the City College of New York.

References

Center, N. A. (n.d.). *Perceptual Optimization of Wavelet Image Compression. (n.d.).* Department of Electrical Engineering, UCLA.

College, H. M. (n.d.). *Change of Basis*. Retrieved from Math Totorial Website: http://www.math.hmc.edu/calculus/tutorials/changebasis/changebasis.pdf

Cooper, I., & Lorenc, C. (2006). *Image Compression Using Singular Value Decomposition*. Retrieved from Retrieved from College of the Redwoods: http://msemac.redwoods.edu/~darnold/math45/laproj/fall2006/iancraig/SVD_paper.pdf

Group, S. (n.d.). *JPEG Compression: What it is - when to use it - and when not to*. Retrieved from Retrieved from the university of Oslo website: http://folk.uio.no/inf9540/SVD.pdf

Image Compression. (2011). *Image Compression: How Math Led to the JPEG2000 Standard.* Retrieved from Image Compression: How Math Led to the JPEG2000 Standard. (2011). Retriwww.whydomath.org/node/wavlets/basicjpg.html

Math is Fun. (n.d.). Retrieved from http://www.mathsisfun.com/divisibility-rules.html

Mathematics, S. f. (2011). *Why Do Math?* Retrieved from Image Compression: How Math Led to the JPEG2000 Standard: http://www.whydomath.org/node/wavlets/basicjpg.html

Purdue University. (n.d.). *A Brief Introduction to Mathematica*. Retrieved from http://www.cs.purdue.edu/homes/ayg/CS590C/www/mathematica/math.html

Rahman Z., J. D. (n.d.). Image enhancement, image quality, and noise, Photonic Devices and Algorithms for Computing., (pp. VII, Proc. SPIE 5907).

Space, N. A. (n.d.). *National Aeronautics and Space Administration*. Retrieved from National Aeronautics and Space Administration: www.nasa.gov

Spacetelescope. (2013). *Spacetelescope.org*. Retrieved from Spacetelescope.org: https://www.spacetelescope.org/projects/fits_liberator/improc/

Stewart, G. W. (1993). *On The Early History of the Singular Value Decomposition.*

Strang, G. (2009). *Introduction to Linear Algebra* (4th ed.). Wellesley-Cambridge Press.

White, W. A. (n.d.). Data Compression for Full Motion Video Transmission. *Conference on Advanced Space Exploration Initiative Technologies*, (pp. 1-11).

Wolfram Mathematica. (2013). *Hands-on Start to Mathematica*. Retrieved from Wolfram: http://www.wolfram.com/broadcast/screencasts/handsonstart/