# SECRET FRAGMENT VISIBLE MOSAIC IMAGE

A graduate project submitted in partial fulfilment of requirements
For the degree of Master of Science in
Electrical Engineering

By

Rupa Shivani Kolluru

December – 2017

This graduate project of Rupa Shivani Kolluru is approved:

| | |
|---|---|
| _____ | _____ |
| Prof. Dr. Ramin Roosta | Date |
| _____ | _____ |
| Prof. Dr. Xiaojun (Ashley) Geng | Date |
| _____ | _____ |
| Prof. Dr. Shahnam Mirzaei, Chair | Date |

California State University, Northridge

ACKNOWLEDGEMENT

I wish to thank project chair and mentor Prof. Dr. Shahnam Mirzaei for his immense support for this project. I am also grateful to Prof. Dr. Ramin Roosta and Prof. Dr. Xiaojun (Ashley) Geng for their suggestions, insights and review on this project. I am also thankful to California State University, Northridge for providing all facilities needed as a part to complete this project. This accomplishment would not have been possible, had I not received help for different complexities involved in this project from all esteemed faculties mentioned above.

# DEDICATION

This graduate project is dedicated to my friends who have always supported me on every step.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Secret-Fragment-Visible Mosaic Image–A New Computer Art and Its Application to Information Hiding

By

Rupa Shivani Kolluru

Master of Science Electrical Engineering

The aim of this project is to build a mosaic image which can hide the information of a given image. Even though small blocks of the given image are visible in the created mosaic picture, but the placement of these blocks is so random in the mosaic picture that it is hard to visualize how would the original image look like. Thus, the name Secret fragment visible Mosaic picture. The given original image divides into smaller tiles of images and these tiles are rearranged to become the final carrier image looking like a mosaic picture, giving an effect of encrypting the image secretly. The mosaic picture requires taking the RGB characteristics of the given original image and they are into a new one-dimensional color scale which is used for further processing the given original image.

A carrier image from a database is selected which normally is most alike to the given original image i.e., original image using the one-dimensional color scale defined earlier. To find a tile image in the given original image which is the most similar to a block of the carrier image, a search algorithm is used. The similarity measure is calculated using the one-dimensional color scale in a mathematical formula. Then the target blocks are replaced by the most alike tile images in the original image, thus creating a preliminary mosaic picture. The tile fitting information is watermarked in the entire mosaic picture created using LSB replacement technique. Then this mosaic picture is taken as the final mosaic picture. The original given picture is retrieved with regard to the final mosaic picture using a secret key.

# 1. HISTORY OF MOSAICS

Mosaic is a piece of art made by gathering together small fractions of different types of materials like glass, tile. Mosaics are the ancient form of images. In image processing art provides valuable guidance for technical inventions. Digital mosaic picture creation has been a booming topic nowadays. In digital field, mosaics are images that are formed by smaller images called 'tiles'. Depending upon different aspects such as rotation, deformations, positioning constraints and dataset of the tiles different mosaics can be formed from the same source image. For example, creating a digital mosaic picture of an ancient man-made mosaic would be very difficult as it has to shape and size of the tiles of the image, the high importance of packing the tiles closely and tile orientation that highly influences the way the mosaic is seen. The tiles of the source image must follow the orientations specified by the artist to portray the true vision of the artist.

## 1.1 Classifications

The four different major classifications of mosaic pictures are:

1. Photo
2. Puzzle image
3. Crystallization
4. Ancient

Classifications type 1 and 2 of mosaics the source image is fragmented in smaller tile images of different rotation, color and size. Therefore, they are "Tile Mosaics". In the next two classifications, the mosaics form by overlapping a source image with a carrier image which is most alike to the source image so as to hide the source image as much as possible. This classification of mosaics is not a solid one, there can be examples which fit into more than one classifications. Other classifications can also be taken into account for digital mosaic pictures depending on which factors are taken into consideration. The figure below gives an idea of how mosaics can overlap.

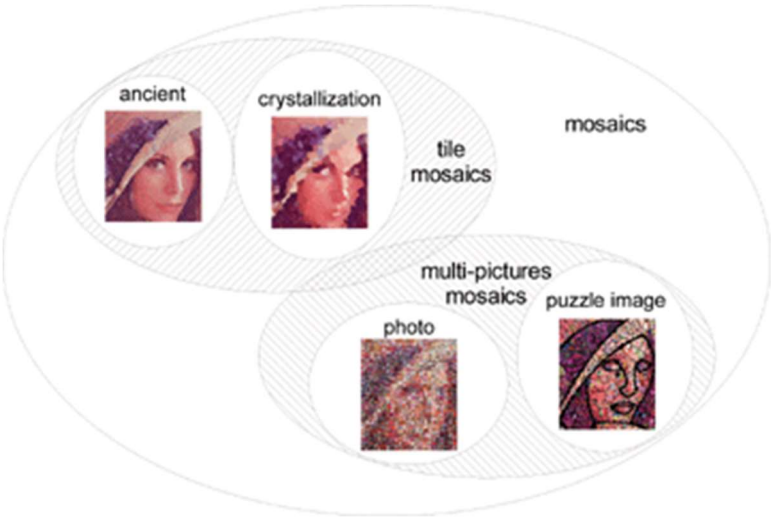


Figure 1 Classification of mosaics

## 1.2 Crystallization Mosaics

Computational geometry like voronoi diagrams combined with image processing can be smart approach to create sophisticated mosaic pictures. These crystallization mosaics portray similar effect as that of glass windows in buildings.

A Voronoi diagram gives an understanding about the distance of a certain set of points on a plane. It is named after Georgy Voronoi. Given a set of points, each point is allocated to a nearest site or plane. These specific points, whose closest sites are not distinct, form the Voronoi diagram. These equidistant sites are called Voronoi cells. Many different algorithms can be used to implement voronoi diagrams, but the most less time-consuming algorithm is plane-sweep algorithm developed by Fortune. The algorithm's worst-case complexity is

O(n log (n)).

For generating a digital mosaic picture Haeberli used Voronoi diagrams. The mosaic picture was created by paling the sites in a random manner and filling each region with the information of the source image. As this algorithm does not follow edge features, is creates a mosaic picture with different shapes and sites giving it a cellular look as shown in the figure below. Haeberli's idea although simple was a starting point for many other techniques to emerge.

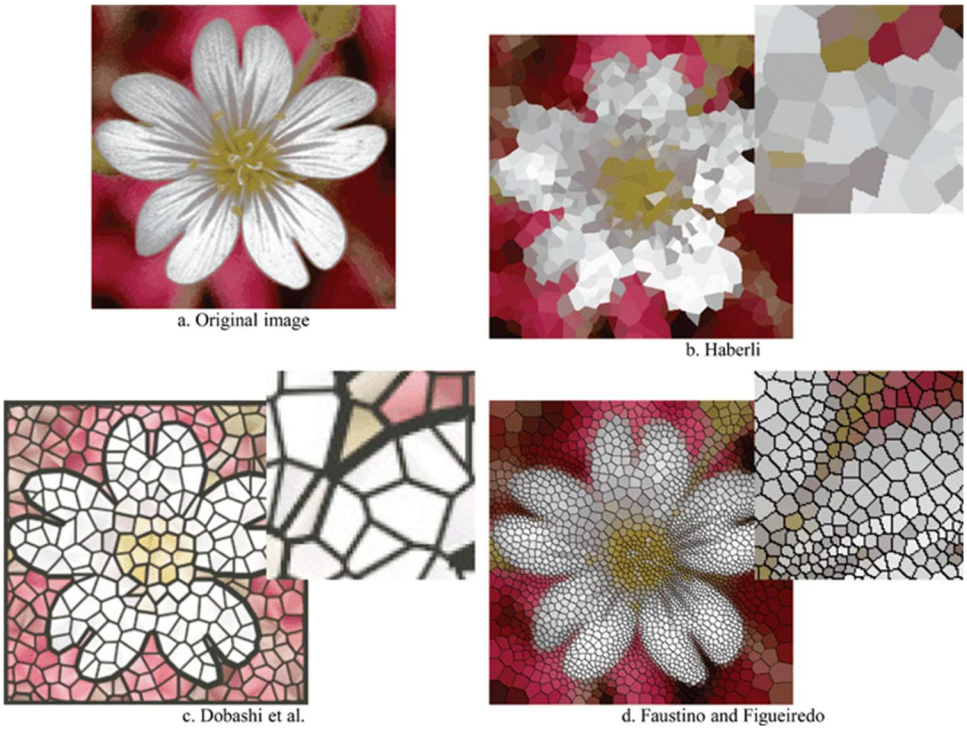


Figure 2 Comparison of different techniques (a) Original image (b) haberli (c) Dobashi et al (d) Faustino and Figueiredo

Dobashi et al had a better take at the original idea of Haberli. The final mosaic picture appeared to look like an image made of different colors of broken glass as the method used comprised both of edge information and Voronoi diagrams. The less flexibility in variation of tile shapes was the major disadvantage of this technique.

Faustino and Figueiredo put forth a method close to Dobashi's. The difference between these different techniques was the flexibility in variation of shapes and sizes of tiles.

The figure above i.e., figure 2 gives a comparison between different techniques and their visual appearances. Considering the features of an image such as edge features and their orientation, the original image does not go through a lot of changes. Only Faustino and Figueredo's approach variates the tile size depending on edge magnitude but without using relative orientation.

### 1.3 Ancient Mosaics

The artwork of ancient mosaics majorly constitutes of combining all the small colored mosaics. Rather than uniform or random distribution of tiles a well-organized use of images features such as orientation, shape and size would be more informative. So, the tiles were arranged in rectangular as it would be only in vertical or horizontal orientation. Such pattern would not allow the observer to see the whole image. To avoid such a setback, they placed tiles in such a manner that it made more emphasis on the edges of the important information to be displayed in the image. In creating a digital mosaic picture, we are not concerned about the materials used to lay the mosaic design but in the way the tiles of the image are laid down. Arranging the tiles in different ways would give different mosaic pictures. To describe the how the mosaic, look the term used was ‘opus’. And different techniques implemented are ‘opus musivum’ and ‘opus vermiculatum’.

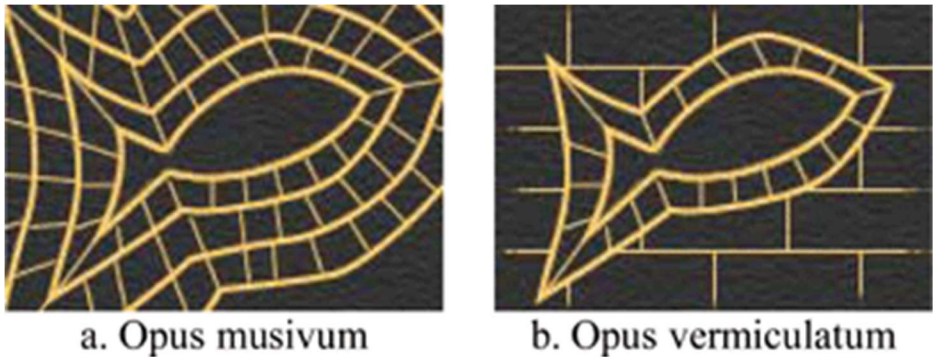


Figure 3 Different mosaic techniques (a) Opus musivum (b) Opus vermiculatum

Hausner was the first person to make an effort to create a realistic mosaic picture. He made this possible by using a mix of different techniques such as centroidal Voronoi diagrams, and other techniques such as Edge features, Manhattan distance.

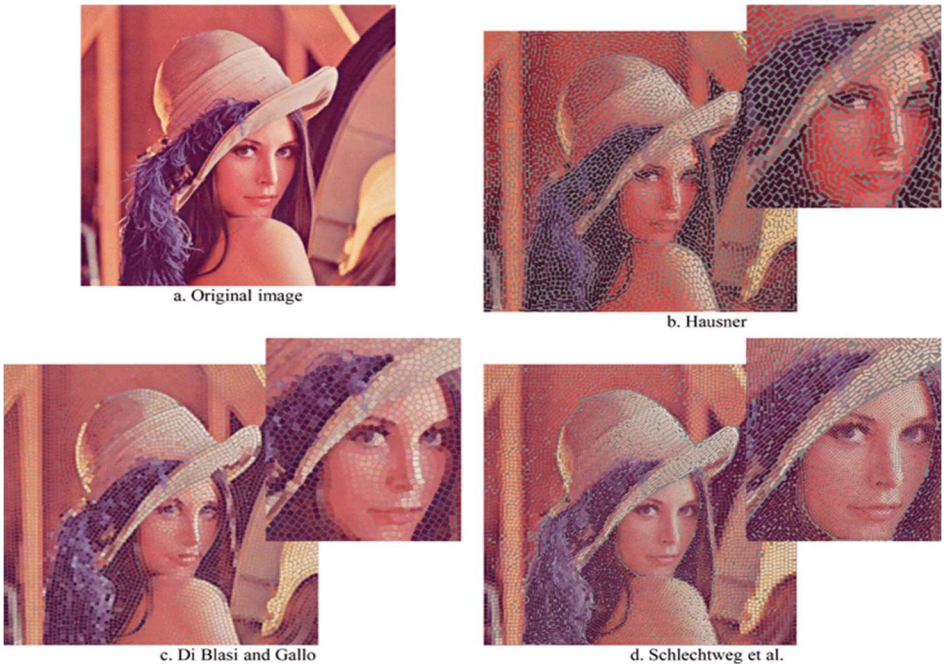


Figure 4 Mosaic techniques (a) Original image (b) Hausner (c) Di Blasi and Gallo (d) Schlechtweg et al

The algorithm implemented by Hausner is as follows,

Step 1: Begin by creating series of random points on the source picture.

Step 2: While the points are not converged:

1. For every point in source S, a pyramid shaped square is placed at the apex of the point.
2. In the Z axis, Change the orientation of each pyramid to line it up to the path of field.
3. Provide the pyramid with an orthogonal projection onto the XY plane, producing a Voronoi diagram.
4. Now for each Voronoi region, the centroid is to be located.
5. The point is to be moved to the centroid of its area.

If graphic hardware acceleration is not used, this algorithm takes a lot of time to get to the optimal solution.

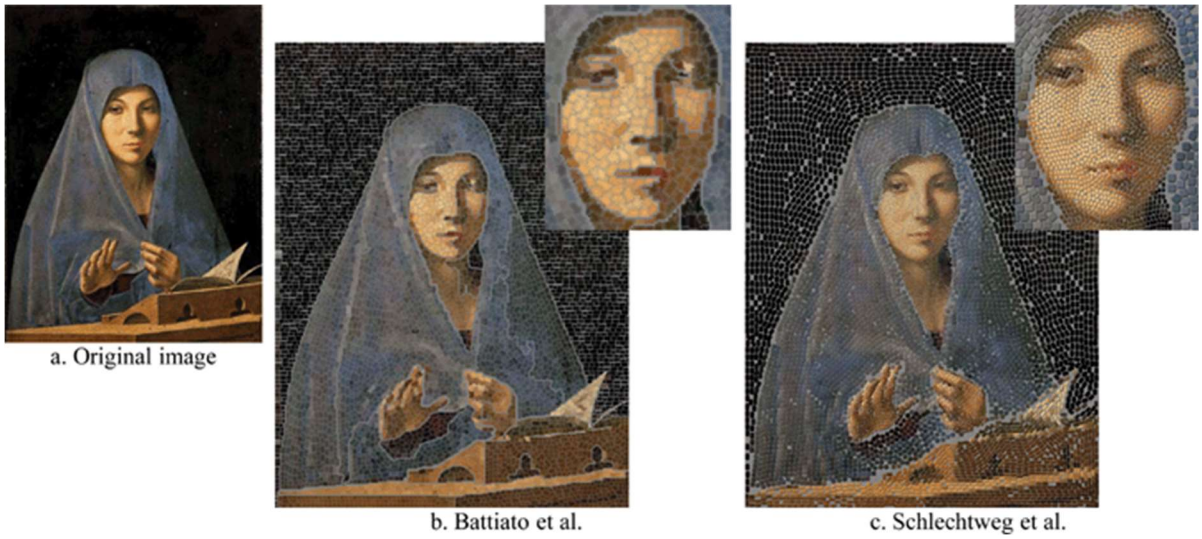


Figure 5 Mosaic techniques (a) Original Image (b) Battiato et al. (c) Schlechtweg et al.

Some more examples of creating ancient mosaics by using another algorithm is shown in figure 4c and 5b. This algorithm leads to more realistic results as it is based on directional guidelines. The image processing techniques used in the placement of the tiles accurately are given below:

1. Divide the source image into smaller tiles using statistical Region Merging algorithm.
2. Dividing the source image into two different regions, background and foreground.
3. For each pixel a distance transformed from the fragmented region is evaluated.
4. Calculate gradient matrix. After that, we begin to calculate the level line matrix.
5. a. Select a line of pixels that are not processed.

   b. The new tiles are places at constant distances through the path utilizing the information given in the gradient matrix.

A technique to implement ancient mosaics has been developed recently which is a stroke based look that uses agents called RenderBots. Each stroke is represented by individual RenderBots. When each RenderBot has completed its painting function then simulation is terminated, and the final image is created. The steps involved in the procedure are given below:

1. Creating several RenderBots of specific classes and spreading them in the environment.

2. a. Allocating functions to each bot like estimating velocity values and changing the local state of the bot and calculating new direction.
   b. Performing movement of the bot to a new point position after calculating the new position of the bot.
   c. Painting each render bot.

Figures 4d and 5c show two images which have been implemented using RenderBots. To orient the mosaic bots to the nearest edges the source images are segmented manually before generating the mosaics. The time complexity for implementation of this technique depends on quality desired by the artist as creating these mosaic pictures using RenderBots is an iterative process. The number of Mosaic Bots involved in the examples shown in the above figures are approximately 9000 and 8000 for figures 4d and 5c respectively and the time take was around an hour.

To conclude the section of ancient mosaics, the main key common in all the techniques is image features such as, the orientation and the position of the tiles. The different approaches used in different techniques are,

1. Based on centroidal Voronoi diagrams.
2. Iteratively changing the orientation and position of tiles.

### 1.4 Photo-mosaics

Photo-mosaics uses tile images to transform the source image into an image consisting of array of horizontal and vertical tiles. In this technique, the algorithm searches for images similar to the pixels in a tile in the source image from a large database and replaces the block with the tiles of that image and thus giving the final mosaic picture. In this approach edge features are not considered.

This art of photo mosaics has been there even before the era of computers. As seen in the figure 6a, it is a very low-resolution grayscale image of Abraham Lincoln, in which each pixel is a tile. But, when it is seen from a distance it can be seen as the image of Lincoln.

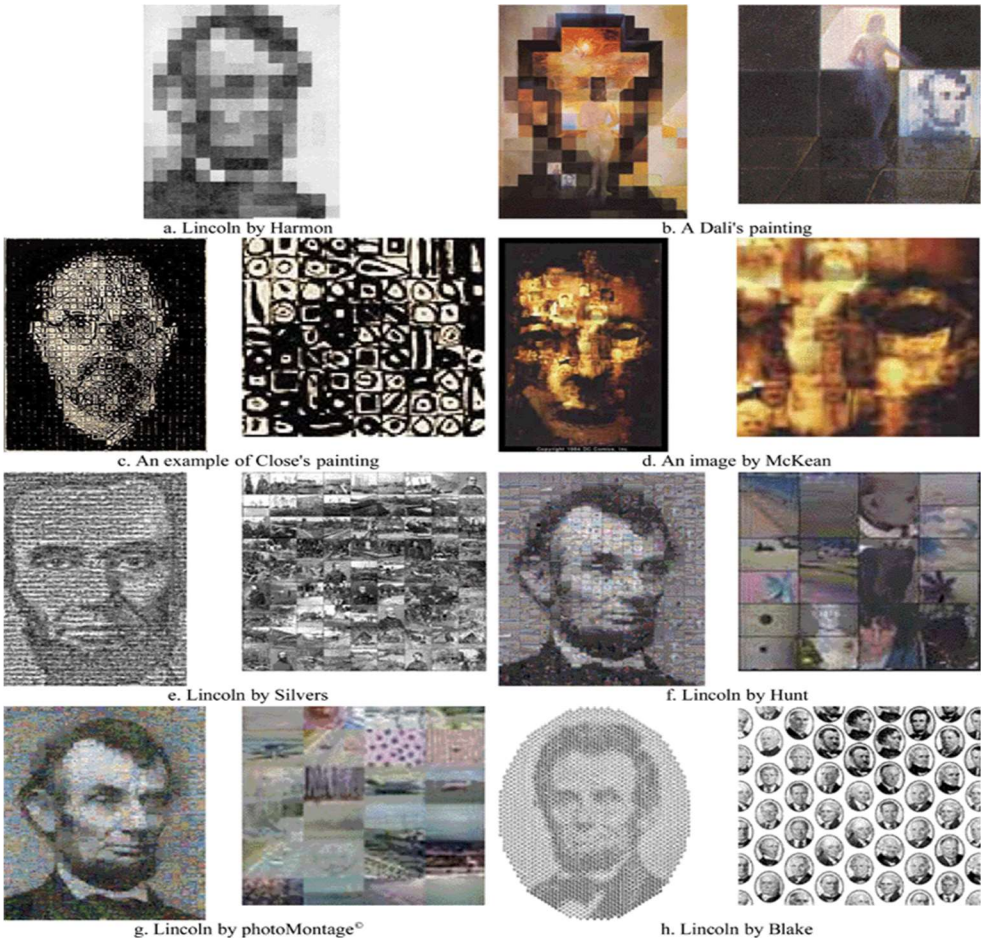


Figure 6 Different Photo mosaic techniques (a) Lincoln by Harmon (b) A Dali's painting (c) A Close's painting (d) An image by McKean (e) Lincoln by Silvers (f) Lincoln by Hunt (g) Lincoln by photoMontage (h) Lincoln by Blake

If each image is observed closely, each image is made up smaller images. When the image observed from a farther distance one can see it as a one whole image. These images shown above are the most complex images created at that time which had other images in one image.

As shown in figure 6c the artist Close started precisely gridded paintings. A few of these paintings give an impression of being generated by the computer. The earliest example of a photomosaic created in a computer is created by M C Kean for a popular comic as seen figure 6d. He got the thought of a tile being larger than a pixel when he was playing around with Adobe photoshop.

Computerized mosaic pictures created from tiled digital photographic pictures is recently developed approach as it requires a lot of computer resources. Silvers started working on photo mosaics as a student. In his images each tile was a block of images, it was a small image in itself. These smaller tile images match the overall tone of the image. Figures 6f and 6g show different implementations of Silvers approach.

According to the grayscale intensity of each image he fitted the tiles in the image to create the image of Abraham Lincoln. He set the tile images in a beehive grid manner and to utilize the space more he used oval images.

## 1.5 Puzzle Image Mosaics

Kim and Pellacini created another category puzzle image mosaic referred to as jigsaw Image Mosaic(JIM), wherein the tiles of the image are random arbitrary chosen shapes used from a preallocated database of shapes to obtain he final image. The concept of puzzle image mosaics is like photo mosaics, but the output is completely different. This problem is approached by defining a tile arrangement that decreases the energy function equation of the mosaic. And a generalized energy-based platform is created for different mosaic problems. The figure 7b shown below is an example of the approach discussed above,

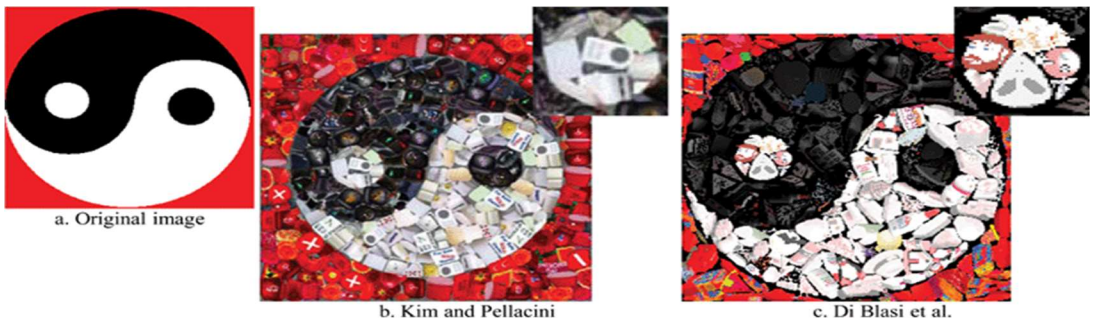


Figure 7 Different types of Puzzle mosaics (a) Original image (b) Kim and Pellacini (c) Di Blasi et al.

Another method for generation of puzzle mosaics is presented by Di Blasi et al shown in Figure 7c. This method is built on antipole strategy and generates excellent ouput

in a reasonable computational time. The method redefines the problem again as a search problem which must search a database to find the similar small tile image. To develop the data structure of antipole the tile's shape and color had to be mapped into the space X. Following steps have to be followed to implement this method:

1. Calculate the tile image's center of mass.
2. Subdivide the tile into 90 fragments and each fragment is considered as a vertex.
3. Calculate the Euclidean distance of each apex from the middle of the mass to each vertex and normalize it.

This creates a vector X of 90 component, where in X is a vector which has information of image features. The distance between two vectors is calculated by Euclidean distance. The calculation considers all possible mutual orientations of the two tile images. The algorithm is described as follows:

1. Begin with a source image.
2. Using the same method, a directional guideline detection is performed.
3. Perform tile rotations to attain an image Im.
4. Evaluate a Voronoi diagram Vr of same size as the source image and choose a random set of points.
5. Combine the images Im and Vr to get a final image F.
6. For every image Fi:
    a. the three steps mentioned above is used to acquire the feature vector x of $F_i$
    b. the similarity test is performed
    c. a color modification is done so as to make sure the median color Fi is same as the selected tile
    d. the tile is resized and rotated so as to fit and is painted all over the region.

# 2. INTRODUCTION TO IMAGE STENOGRAPHY

The art of putting together little blocks of different materials like stone, glass or other materials is defined as Mosaic Art. Image stenography also uses this principle to create an effect of hiding one image in another. It is more preferred than encryption as encryption hides the meaning of the information, but stenography hides the whole existence of the information. As the risk of cyber is increasing day by day it is one important issue that has to be addressed as data is one of the most crucial things to be taken care of during transmitting it over the internet. A secured data transmission is protecting the data from the hackers and unauthorized users. Data encryption is a technique to hide information by encrypting the data to cipher texts and transmitting the data with a unknown secret key. But, whereas Image stenography increases security to the next level by hiding the cipher text into the image, text or other formats. Watermarking is another technique that is used alongside with image stenography in the area of data security.

Until now a number image transmission techniques have been proposed but, each of them has its own flaws which leads to insecure transmission of images over the internet. There are many approaches to overcome this issue like data hiding technique, encryption methods, JPEG compression, etc. Hiding a huge amount of information in a single image is one of the main issues. So, if one wants to hide the information of a given image in another image the given image has to be compressed in advance. But, the technique use d in this project does not require the compression of the image.

The basic structure of image stenography constitutes of three components: the original image, the carrier image and the key. The carrier image is the image in which all the details of the given original image is embedded so that it can be securely transmitted. The key is a very essential part of this structure. Without the key the given original image cannot be retrieved from the carrier image.

Another kind of mosaic picture is created with this technique called as secret fragment-visible mosaic picture which has minor blocks of the given original image. In this kind of mosaic picture, all the pieces of the given original image can be seen but, the actual original image cannot be figured out by looking at it as the pieces are jumbled in a

very random manner. Hence, it can be said that the original image is embedded into another image forming a mosaic picture although the small blocks of it can still be seen by the observer. Therefore, comes the name of this mosaic picture, “secret fragment visible mosaic picture”.

The algorithm used in the project for image stenography modifies a given original image into a mosaic picture which has the same image features resembling the final carrier image which is selected by a method which uses a new scale of similarity. A secret key used in this process plays a prominent role in losslessly retrieving the given original image from the final mosaic picture.

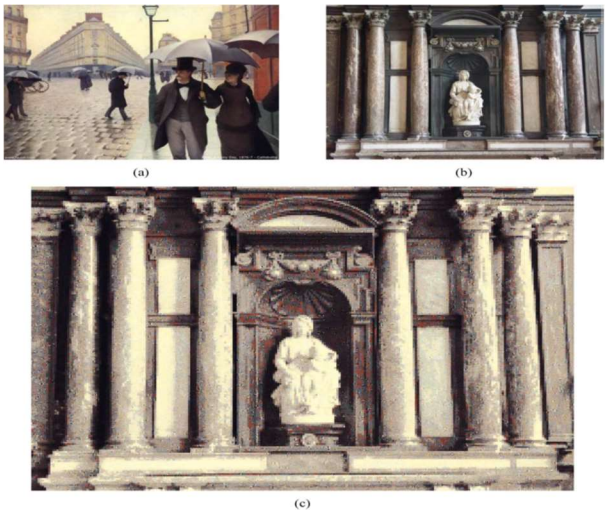


Figure 8 Example of secret fragment visible mosaic picture (a) Original Image (b) Carrier image (c) The Mosaic picture

## 2.1 Working

This project mainly consists of three phases:

Phase 1: Creating a large database, which can be then used to select a carrier image which is the most alike to the original picture by using a new similarity measure.

Phase 2: By using a search algorithm find a block of original image which is the most similar to the tile in the carrier image and replacing the tile with the block. Also embedding the information of the original image in this image and thus, forming the final mosaic picture.

Phase 3: Using the key retrieve the original picture from the final mosaic picture.

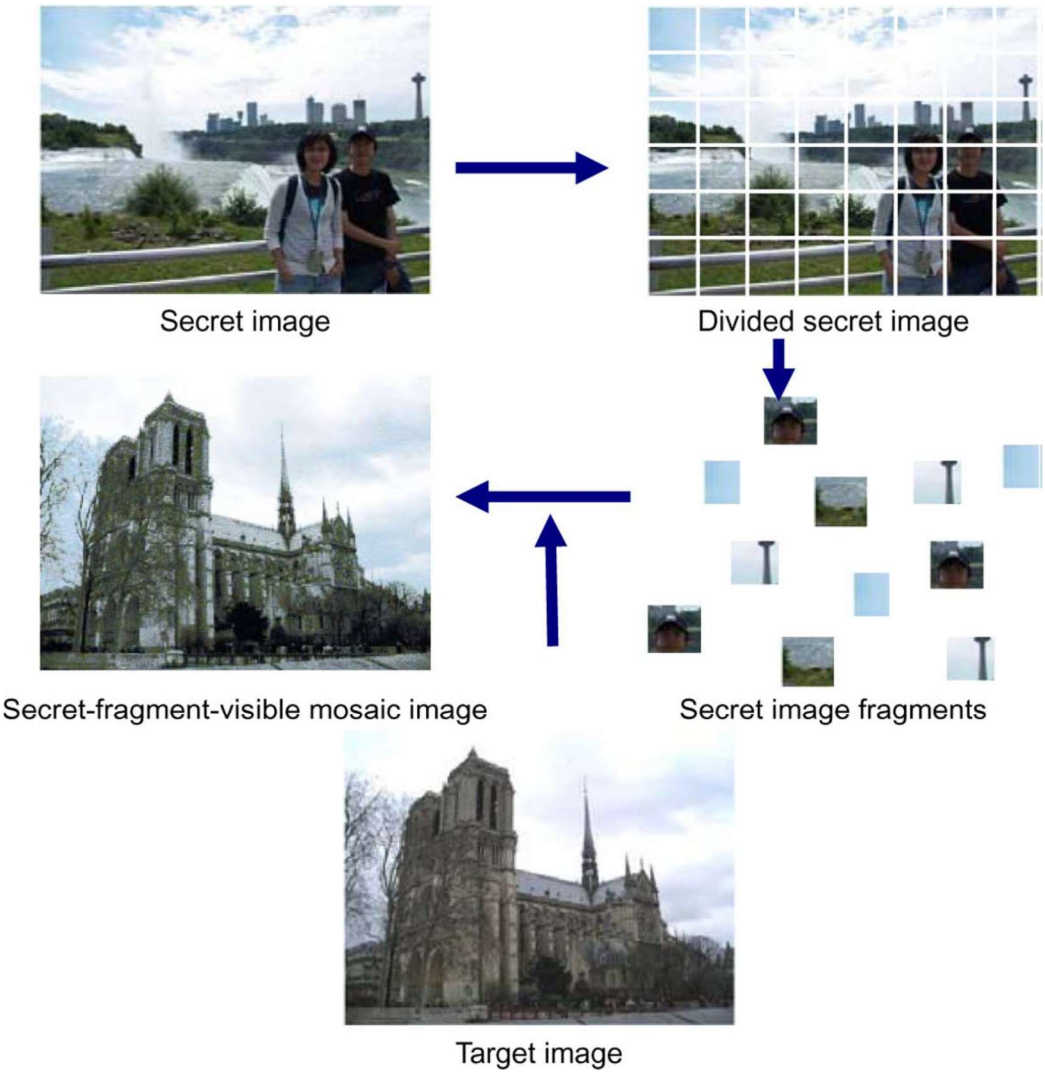


Figure 9 An illustration of creation of the Secret Fragment Visible Mosaic picture.

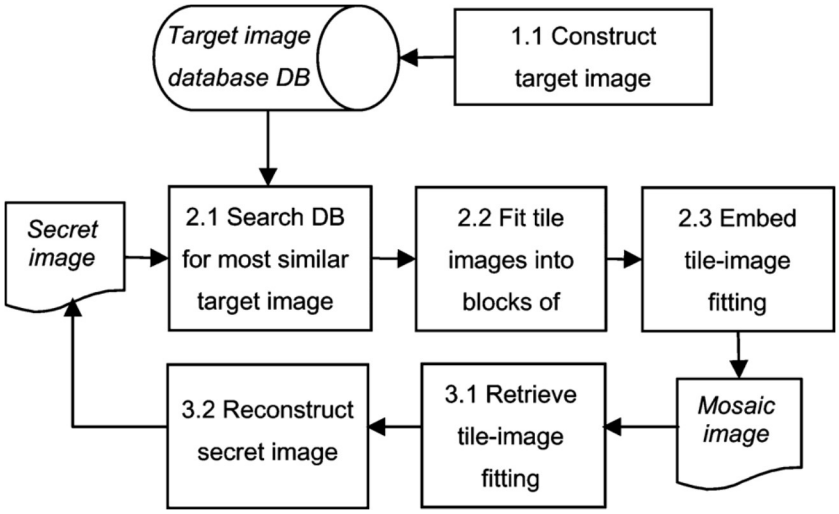


Figure 10 Flowchart depicting the creation of the secret fragment visible mosaic picture and original image recovery

The phase number one composes of steps involving to create a database of eligible carrier images.

The second phase has three different steps of operations:

Step 2.1: Skimming through a user selected database to find a carrier image that can be said to be more like to the given original image.

Step 2.2: Replacing each block of the carrier image with the most similar tile in the original image.

Step 2.3: Watermarking the mosaic picture with the information of the location of the tiles in the carrier image using lossless LSB replacement scheme.

The third phase is retrieval of the original image from the final mosaic picture using the above derived information. This phase has two steps of operations:

Step 3.1: Retrieving the information of location of the tiles of original image from the mosaic picture created in the former phase.

Step 3.2: With the information recovered from the above step recreating the given original image from the final mosaic picture.

## 2.2 Constructing the Database

The construction of database plays a very important role in finding a carrier image that is most alike to the original image. Therefore, it effects the creation of the final mosaic picture. It depends on the carrier image found and to what extent that image is indistinguishable to the original image which decides how accurately the selected image can embed the original image. If it is not similar enough to the original image the output mosaic picture may turn out to be distorted.

So, to generate a better looking final mosaic picture and for better hiding of the original image a large database is necessary. Searching a large database for an image that is most alike to original image comes under the concept of “Content Specific Image Retrieval”.

“Content Specific Image Retrieval” is different from the usual approaches. The phrase "Content Specific" refers to the concept of searching the contents of the image instead of data such as tags, descriptions or keywords related to the image. The term "Content" in this context refers to the image features such as, colors of the image, shapes of the image, textures of the image, or any other information associated with the image.

### 2.2.1 Different Techniques to tackle the problem of image retrieval:

**Query techniques:**

Query technique involves giving the CBIR system a sample image depending on which it continues its search of database for the similar image. Depending upon the user’s application the search algorithms may vary, but the output images should all share a common element with the sample image. Different ways of providing sample images to the CBIR system include:

1. The user may supply an existing image, or an image selected from a random set.
2. With a vague approximation of the final image attributes the user is looking for, a sample image is selected.

The main advantage of this technique is that it eliminates the difficulties that come up when the images are described with words.

**Semantic retrieval:**

Semantic retrieval technique begins with the request of the user like, “find pictures of marina Del Rey”. This is a very difficult request to process for the computers as this could be a place or a person or something else. So, the CBIR systems make use of less complicated image features such as texture, color or shape of the image. The given features are the used either followed by more input from the user about the criteria or with databases which can do a search with higher level image features. In general, image retrieval using this technique demands more user interface to search in an efficient manner.

**Relevance feedback:**

Using the available CBIR searching techniques to fulfill all the intentions of a wide range of users and their applications is a difficult task. CBIR can be successful done only when the application of the user is completely understood. So CBIR systems can utilize a new technique of relevance feedback, in which the user constantly gives a feedback and modifies the search results by marking the images if they are “relevant” or “not relevant” or “neutral” to the search.

**Iterative/machine learning:**

This approach can be used only if one has required information of the content of the image. This problem is much simplified if binary images are used. The content extraction becomes easy if the set of black pixels are assumed as objects and the rest pixels as background. After doing a lot of experiments using different methods it can be concluded that distance-based measures can compare binary images better than set based measures.

Considering working on gray scale or color images, basically there are two means of comparison:

1. Extracting some relevant objects by detecting shape, threshold, segmenting, shape and edge detection and then comparing these objects.
2. Comparing the entire image.

Method 1 results in high level image recognition whereas, method 2 results in low level image analysis. In both methods choosing the favorable feature parameters plays a very important role.

### 2.2.2 Comparing content using image distance values:

Measuring the image distance is one the most used methods in CBIR system for comparing two images. Comparing the similarity of two images using image distance values constitutes comparing various attributes such as color, shape, texture, etc., For instance, a distance value of 0 would mean a precise match with the query, in terms of the attributes considered. If the value is more than 0 then it would mean that there are a lot of alikeness between the images. Then, based on distance values, the search results are sorted.

**Color:**

Color similarity based distance measures implement a color histogram for each picture that measures the number of pixels in a picture having specific values. Comparing pictures with respect to colors is one of the commonly use techniques, as it ideally can be done by completely ignoring the image size or orientation. Further, block color measurement by segments and a three-dimensional relationship between different color segments was also experimented.

**Shape:**

Shape in this context refers to a small region of the image. These are obtained by implementing fragmentation or edge detection on the whole image. Shape descriptors have to be unaffected by rotation, translating or scale of the image. Some of the examples of shape descriptors include, Fourier transforms and Moment invariant.

As size of the tile images in the original image are smaller than the actual image and the factor accounted for is just the visual appearance of the image, so the color factor plays a major role. Therefore, the accurate image similarity measure to select an image from a large database in this context would be through extracting the color distributions of the images which introduces the concept of Indexed Color.

**Indexed Color:**

The process of Indexed Color involves recovering the pictures or videos from the database based on their color content. The major attributes to be achieved by the process of Indexed Color are automated extraction of color, efficient retrieval and indexing. There are two major classes of methods for Indexed Color: local or region color and global color distribution. The major difference between these two methods is that in the global color distribution method, the color distribution is done on the entire image whereas in local color distribution, the matching is done between the local regions of the image.

When the user gives just a sample picture Indexed Color by global distribution gives optimum results. Global color distribution method works for this instance as the user is not concerned about the smaller regions of the image. Although, if the user wanted to search for things within the image this method would not help much as it does not help in determining the localized color regions.

In contrast, Indexed Color by local or regional color helps in matching partial images. For instance, if the end user wanted to find all the pictures in which the moon is in the right portion of the image, then regional indexing would help in solving the problem. Regional or local Indexed Color is more complicated as it needs representation and extraction of all the local regions. It becomes a very elaborate task if the image is large or the database is huge.

In any case, both the methods need a system which would implement automated extraction and effective color representation for an effective recovery of the image or the video. The technique most useful for the application of this project is global color distribution.

## 2.3 Localized or Region Color image extraction feature:

This Indexed Color system consists of two stages:

1. Segmentation.
2. Region property extraction as shown in Figure 11.

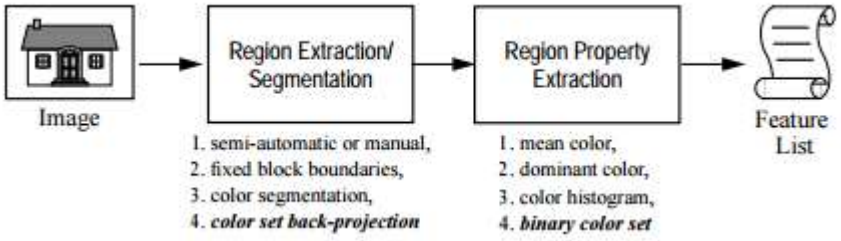


Figure 11 An outline of color image extraction feature

### 2.3.1 Region extraction

Acquiring the three-dimensional boundaries of the regions that would be helpful for user's application is the main aim of Region extraction system. The method of region extraction and image segmentation are very different from each other. In region extraction, one image pixel could be allocated to more than one regions but in Image segmentation one image pixel is allocated as only one fragment.

This method is the most used method when compared to segmentation as it gives an objectified picture of the image. There are a lot of methods that are used for extracting the region. The lowest complicated extraction technique is either non-computerized or partial-computerized.

This method, is a very monotonous task for the users to assess the pictures and visually identify the relevant information. Another region extraction method is extracting a constant segment of block of the image. There is more chance of obtaining a match between the blocks if the blocks are represented by color content.

### 2.3.2 Region feature extraction

As, the regions are recognized, each region is categorized as a set of features. The main aim using color representation to represent blocks is to precisely note the color features of each region. There are easier ways to represent the region's color characteristics. One way is to represent the it by the average of all the colors of the region. Another way is to represent it by the most dominant color of that region which utilizes a three-dimensional vector. And another way to determine the color of that region is to use color histograms.

As multi-dimensional feature vector represents a color histogram and assessing them is mathematically very complex they are meant to give optimal results if they are used to represent global color instead of the local color regions.

The combined probabilities of all the three-color channels i.e., R, G, B. characterizes a color Histogram.

$$H_{r,g,b}[r, b, g] = N.\ Prob\ \{R = r; G = g; B = b\}$$

where N = total number of pixels in the given image. The histogram of colors obtained by plotting the each color of the image separately and by including each color's pixels. As there are a lot of levels in each color, one way is changing the histogram of three-color channels into a new single variable histogram.

The mathematical equation to change a three-channel image to a single transform is :

$$m = r + Nr \times g + Nr \times Ng \times b$$

where "Nr", "Ng" and "Nb" represent the bin values for colors red, blue and green, respectively. Once the three characteristics of the image are reduced to a single variable, the histogram of that variable is:

$$h[m] = N.\ Prob\ \{M = m\}$$

1-D histogram is generated according to the method described above the color values (r, g, b) have to be dequantized to fewer levels, resulting in new colors (r', g', b'). Now, the new transformations, i.e., (r', g', b') are changed into a single-color characteristic by the mathematical equation,

$$F(r', g', b') = r' + Nr \times g' + Nr \times Ng \times b'$$

Since the results obtained by implementing the above method are very noisy as shown in figure 12c and 12g were created respectively with figures 12a and 12e as input original images and figures 12b and 12f as chosen carrier images, another approach is implemented. In this method the color transformation function is changed to

$$h(r',g',b') = b' + Nb \times r' + Nb \times Nr \times g'$$

Where Nr, Ng, Nb are assigned a value 8 and the biggest value is given to the green channel color and the smallest value is given to the blue channel i.e., 1. These values are assigned to the channels in this manner as the human eye is most sensitive to green color and least sensitive to blue color.

In addition to all Nr, Ng, Nb values set 8, it speeds up the process of creating the mosaic image. The h function defined above redefines the image with a one dimension h-colorscale. The mosaic image created using above h-function method is shown in figure 12d and 12h. As seen in the figures comparatively the method implemented with h-function has less noise compared to the images created using the other method.

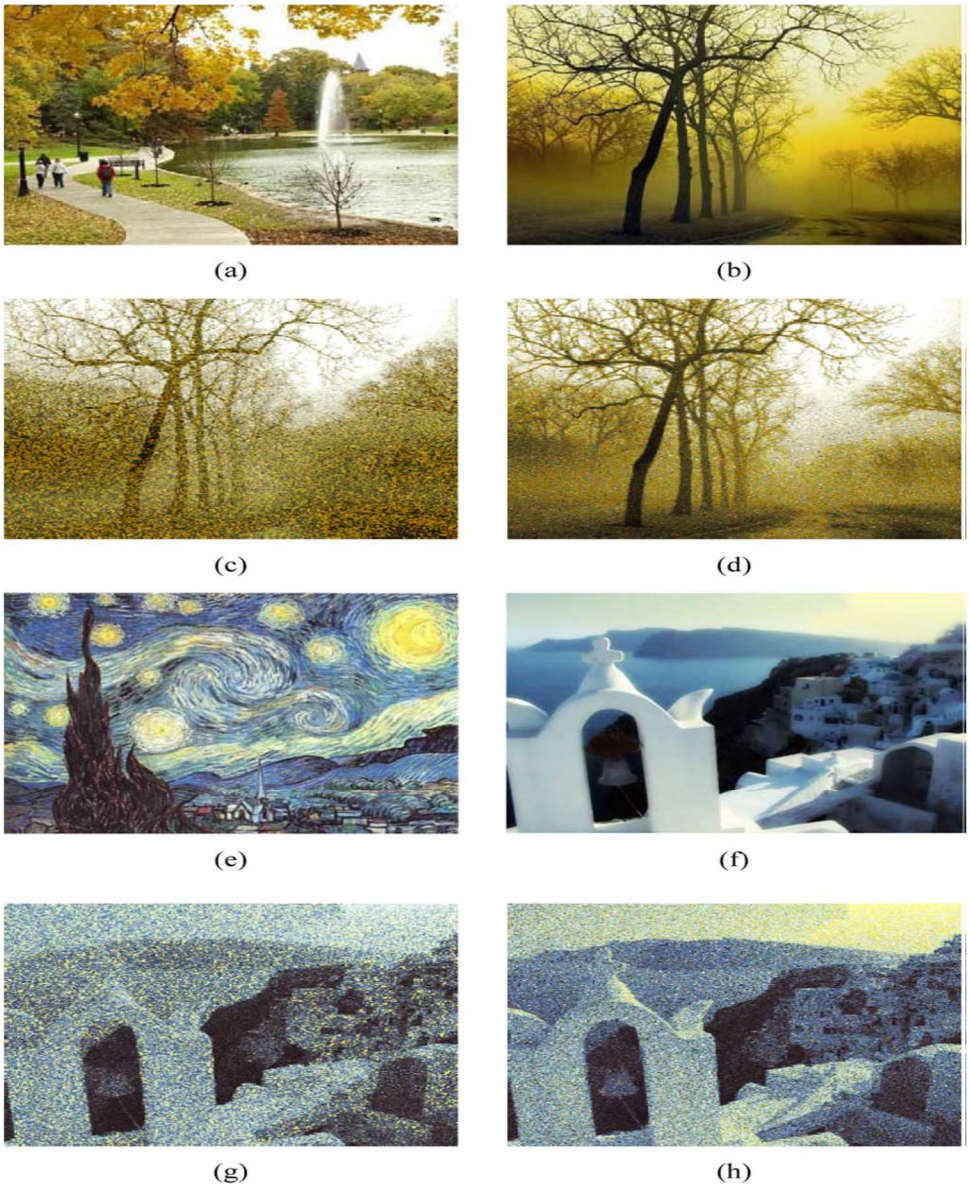


Figure 12 Different examples of creating mosaic pictures depending on the different image similarity measures 12 (a) and 12 (e) Original Images 12 (b) and 12 (f) Carrier Images. 12 (c) and 12 (g), 12 (d) and 12 (h) Mosaic pictures developed using different similarity measures

Now, after concluding on a technique to develop less noisy final mosaic pictures, which helps in finding the most similar carrier image, tile images in original image that are similar to the block sin the carrier image have to be determined. A new method is devised to measure the color characteristic of a tile or a block called as h-feature written as hc , and is calculated as below:

1. The average of all the RGB pixel values of each tile or block c is evaluated (rc, gc, bc).
2. These values are re-quantized to Nr, Ng, Nb using the equation below,and the thre color channel (rc, gc, bc) are transformed into (rc',gc',bc') which now represent the new color channels red, green, blue respectively.

$$h(rc',gc',bc') = bc' + Nb \times rc' + Nb \times Nr \times gc' \quad (1)$$

Now, the procedure to construct a database "DB" of potential candidates to be selected as carrier images from a set of M randomly selected images with size Zc which are used in the process of mosaic image creation is as follows,

For each input image D in M:

1. Divide the whole image D into blocks of carrier image of size Zt.
2. Calculate the h-feature value using the equation (1) for every block in the carrier image and create a h-feature histogram H for every image D.
3. The h-feature values of the of all the blocks of every carrier image and the histogram values of every image D are stored in the database "DB".

## 3. CARRIER IMAGE SELECTION AND IMAGE SIMILARITY MEASURE

Since, now a database is created, a carrier image has to be selected from the database that is most similar in terms of the similarity measure to the given original image S. To accomplish this, the original image S is divided into blocks of a preselected size Zt, then the h-feature components of every block in the original image are calculated by equation (1), and the h-feature histogram Hs of S is developed. The image similarity value m (S, D) between original image S and each potential carrier image D is given by:

$$m(S, D) = \frac{1}{\sum_{h=0}^{511} |H_S(h) - H_D(h)|} \tag{1}$$

Where, Hd is the histogram of h-features of every carrier image in DB. And Hx(h) with X = S or D represents the number of blocks of the image in the "bin" of the feature value of (h). As the value of m(S,D) increases the similarity of D and S also increases. If the contents of h-features in HS and Hd are matching, then S and D are said to be completely similar. After calculating the similarity values for every potential carrier image in DB compared to the original image S, an image D0 which has the largest similarity value is selected from DB as the potential carrier image for S which is used in the creation of the mosaic image.

### 3.1 Incorporating the tile images into carrier blocks:

Now that the desired carrier image D0 is selected, the issue of fitting the tiles of the original image S into each carrier block in D0 arises. The problem of incorporating these tile images into o the selected carrier image in a optimal way may be considered to be a problem of single source closest path. This method deals with recognizing a path in a graph in which the sum between the vertex edge weights is the least. In this scenario, vertex of the graph is represented as the state of fitting a tile and the act of fitting the tile image in each carrier block is denoted by the edge of the graph, whose weight is depicted by the similarity value between the carrier block and the tile of the original image.

Consequently, if the number of carrier blocks or the tile images are N, the graph for this is a N-level tree with three properties:

1. The single source for this graph is the root at the first level.
2. All the N tile images that are to be selected for replacing the carrier block are the N vertices at the second level.
3. And the solution to fit all the N tile images into all the N carrier blocks is considered to be leaf vertex at the Nth depth.

There are many algorithms that can be used to solve this problem, some of them are:

### 3.1.1 Dijkstra's algorithm:

This algorithm helps in solving the problem of finding a path between the vertices that has the least sum. There are many versions to this algorithm, one of them is its original variant, finding the closest path between two vertices, but a more commonly used one is the one in which a single vertex is considered as the source and the algorithm determines the closest path from that source to every other vertex in the graph.

This algorithm mainly works on determining the path that is closest from the origin vertex to every other vertex. It can also be used in determining the path that is closest from the origin vertex to a destination vertex and as it reaches to the closest path the algorithm exits.

For instance, if cities depict the vertices of the graph and the cost of driving between these cities depict the edge, this algorithm can be used to determine the closest route possible to get from point A to point B or closest route from one city to every other. This algorithm is by far the most fastest and the most used single source closest path algorithm for directed graph with positive weights.

The algorithm is as follows:

1. Initially, let the first vertex be the primary vertex and the next vertex considered be Y. This algorithm assumes an initial distance value from the primary vertex to the vertex Y and changes it later as the algorithm proceeds further.

2. Set the distance for the primary vertex to be 0 and the distance from the primary vertex to every other vertex as infinity.
3. Mark primary vertex to be current and all the other vertices as unvisited. Build unvisited set which has all the unvisited vertices.
4. Consider the current vertex and all its neighbors, determine their distances. Compare this distance with the distance already given and allocate the distance which is lesser of the two.
5. After allocating distances to all the neighbors of the current vertex mark that vertex to be visited vertex and delete that vertex from the unvisited set so that one vertex not revisited again.
6. The algorithm stops when it reaches a vertex that has been marked as visited already or a distance between two vertices is marked as infinity, i.e., when it reaches to a vertex which is not connected to the primary vertex.
7. If not, consider the unvisited vertex which the least distance and set it as the new current vertex and repeat steps from 3 to 7.

### 3.1.2 A*(A star) search algorithm:

This algorithm uses heuristics to solve the problem of single pair closest path and using heuristics speeds up the search. A* is a most commonly used algorithm to find path and traversing a graph. And is most widely used for its high performance and precision.

A* is a greedy search algorithm in which it tries to solve the problem by finding a path among all the paths leading to the solution that has a lowest value and also among these paths selects the path that would get to the solution most quickly. The algorithm starts from the primary vertex and builds a tree of paths considering that vertex as the starting point, allotting paths one step at a time. This goes on until it finds the primary vertex.

During every iteration, the algorithm decides which path must be expanded into one or more longer paths. It implements this by determining an approximation of the total weight that would sum up to go to the destination vertex using a function,

$$f(n) = g(n) + h(n),$$

where in “n” represents the destination vertex, the value of the path from the primary vertex to all the vertices is given by “g(n)”, and the heuristic that determines a path that is the shortest from the primary vertex to the destination vertex is represented by “h(n)”.

To select the closest path repetitively this algorithm uses a priority queue called as open set. In the algorithm at every step, the vertex with the least value of f(x) is deleted from the open set, and the values of adjacent vertices are also changed, and the vertices are placed in the open set. This process is repeated till the destination vertex has the lowest value of f(x) compared to other vertices in the open set or till it is null. The destination vertex with the value f(x) is finally the path that is the closest, as heuristic function value h is zero at the destination vertex.

With the help of the algorithm so far, the length of the closest path is determined. But, to track the flow of the algorithm, it is modified so that it saves the previous visited vertex by the current vertex. So, the last vertex will point to its previous vertex and this goes on until at some point the vertex’s previous vertex is the start vertex itself.

### 3.1.4 Floyd–Warshall algorithm:

This algorithm gives a solution to the problem of all pairs closest path. This algorithm can find the closest path between all pairs of vertices in one loop. But, it does not keep track of the paths themselves. This can be determined by making simple changes to the algorithm. The algorithm can be described as follows:

The Floyd–Warshall algorithm takes each pair of vertices and compares all the paths possible between them for all pairs in the graph. It does this with $\Theta(|V|^3)$ comparisons in a graph and with $\Omega(|V|^2)$ number of edges, and with all possible combinations are verified. This done through constantly improvising the approximation on the path that is the closest from the origin vertex to the destination vertex, till the assumed approximation is the lowest. Let’s say, there is a graph G with vertices V

numbered from 1 to N. Moreover, a function named shortest_ path (m, n, z) is defined that gives the possible closest path from point m to n by using the vertices that range from {1, 2,…..z} as intermediate points.

The closest path for each of these pair of vertices can be either the path which makes use of vertices only in the range {1, 2,…..z} or the path which makes use of the vertices that can go from m to z+1 and then from z+1 to n. As it clear that the best path from m to n only makes use of the vertices 1 through z and if there is a closest path from m to z+1 to n, then the final length of the path would be the combination of the closest path from m to z+1 and from z+1 to n.

If v (m, n) is the value of the edge connecting the vertices m and n, the function closest_path (m, n, z+1) with the best case can be defined as:

$$\text{closest\_path}(m, n, 0) = v(m, n)$$

and the recursive case is

$$\text{closest\_path}(m, n, z+1) = \min(\text{closest\_path}(m, n, z), \text{closest\_path}(m, z+1, z) + \text{closest\_path}(z+1, n, z))$$

The function closest_path (m, n, z) is run, for all the pairs (m, n) wherein z = 1 to N until the closest path for all the pairs is found.

### 3.1.5 Johnson's algorithm:

This algorithm is another way to determine the closest path between weighted edge graphs. It also works for negative edge weights. It uses the Bellman-Ford algorithm to eliminate all the negative weights. And then, uses Dijkstra's algorithm on the new transformed graph. This algorithm is described as:

1. An initial vertex k is included that is connected by zero weight edges to every other vertex.

2. Start with a new vertex k, for every edge e the least vale h(e) of a path from k to e is determined. The algorithm ends if a negative cycle is detected.
3. Then, every edge from m to n with length v (m, n) is assigned a new length,

   v (m, n) + h(m) + h(n) and this is calculated by using Bellman-Ford algorithm.
4. At last, the vertex k is removed, and the closest path from every s to every other vertex is found by using Dijkstra's algorithm.

The complexity of time for this algorithm is given by,

$$O(|V|^2 \log |V| + |V||E|)$$

### 3.1.6 Search algorithm implemented in this project:

According to the requirements of this project, Dijkstra's algorithm may be used to find the solution for the tile fitting information in the carrier image. This algorithm has a complexity of time of "O ($V^2$) "where, "V" in this context depicts the tile image. Calculation of the value of "V" is given by,

$$V = 1 + N + N \times (N-1) + \cdots + N \times (N-1) \times \cdots \times 1$$
$$= \sum_{n=1}^{N} \left( \frac{N!}{n!} \right)$$

where N is the number of tiles

As N increases, the value V also increases thus increasing the time complexity of the algorithm. In image stenography, the images used are usually very large images and so the number of tile images N in each of the original images is also a very large number. Hence, the time required to reach to an optimal solution by using Dijkstra's algorithm very high, which is not possible practically.

So, to solve the problem, a new greedy search algorithm has to be used which can conclude to an optimal solution in a practical amount of time. The search algorithm implemented in this project is a greedy algorithm that is not so optimal, but it is okay for the requirements in this project.

The concept of both Dijkstra's algorithm and A* algorithm is used in this project. The tile image for which the most similar block has to be found in the carrier image is considered as the origin vertex of the tree at level one and the blocks in the carrier image as the neighboring vertices. And the edge between the origin vertex and any other vertex is the similarity measure between the tile image and the block image. The action fitting the tile into the block image is considered as a leaf vertex. Dijkstra's algorithm is used to greedily find the similar block in carrier image and A* algorithm is used solve the tree problem.

A similarity measure has to be determined to find a tile image similar to the block image. a selection function for the greedy search algorithm to select a tile images the most similar to each carrier block. To accomplish this, the average of Euclidean distance between the tile image and block images is considered.

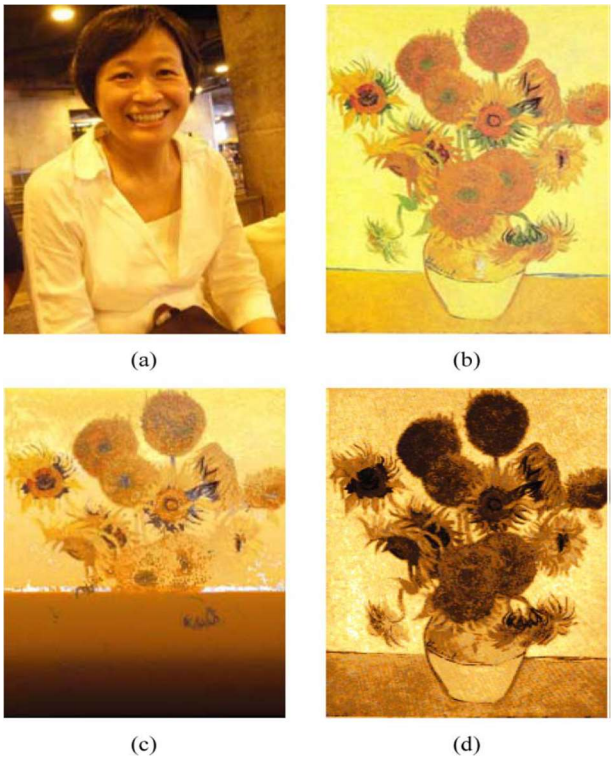


Figure 13 Examples of different Mosaic pictures using different similarity measures. (a) Original image (b) Carrier image (c) Mosaic picture produced using Euclidean distance (d) Mosaic picture created using h-features.

The result obtained using Euclidean distance is Figure 13c with Figure 13a as the original image and figure 13b as the carrier image. It is seen clearly that the image 13c is distorted in the lower parts. Hence, the similarity measure used in this technique does not yield in satisfactory results.

The image obtained by using Euclidean distance is distorted i.e., the lower part of the image is filled with irrelevant colors as the number of tile images are constrained to original image's size. When the algorithm starts filling the carrier image with blocks of original image as it reaches towards the end of the image the tile images to fit into the carrier image keep on decreasing resulting in a poorly constructed mosaic image.

So, a viable solution to this problem is to use a similarity measure calculated on the basis of a concept called h-features. A block similarity value m(s, d) for a tile image s in original image having a h-feature measure of hs and a carrier block d having a h- feature measure of hd is given by,

$$m(s, d) = \frac{1}{|hs - hd|} \qquad (2)$$

The similarity value calculated on the basis of h-features takes into account the difference in the intensities between the tile images and the carrier blocks and produces a mosaic picture, which visually looks like the carrier image but has the contents of the original image. As seen from the figure the Figure 13d is a very drastic improvement of the figure 13c.

## 4. EMBEDDING INFORMATION OF ORIGINAL IMAGE IN THE CARRIER IMAGE

As the carrier image is now selected from the preselected database, the information of the original image must be embedded into the carrier image to form the preliminary mosaic image. Using the similarity measure defined above based on the concept of h-features is used to find the most similar carrier block for a tile image. In this process, the placement of every tile image into the carrier block is saved in a sequence called LR, original recovery sequence. To obtain the mapping information, for the top most left carrier block d1 in the carrier image D0, determine the most similar tile image si in the original image S using equation (2), and replace the carrier block d1 with tile si and save the placement mapping si →d1 in LR.

For the next placement mapping, select the carrier block to the right of d1 i.e., d2 and, determine its most similar tile sj and save the placement mapping sj →d2. This greedy technique goes on until all the carrier blocks in the carrier image are exhausted and every block has a most similar match in the original image. Finally, LR has two sequence of indexes LR1 = i,j,k……..and LR2 = 1,2,3……...with mappings $i \rightarrow 1, j \rightarrow 2, k \rightarrow 3,$ and so on. The resulting LR may regarded to include two block- index sequences, L1 = i,j,k…… and

L2 = 1,2,3,…… with mappings $i \rightarrow 1, j \rightarrow 2, k \rightarrow 3,$ and so on. As LR2 is already in sequence, it does not have to be considered which saves up data of LR that has to be embedded into carrier image. And now, LR consists of only the sequence LR1 i.e., the sequence of the tile images si, sj, sk,……..

To recover the original image as is, the information of the dimensions of the original image should be embedded in the carrier image. For accomplishing this, the information of height, width of the original and the predefined size of each tile in the original image is converted into a binary string and is concatenated together to obtain a sequence of bits. This sequence of bits is embedded into the first two blocks of the preliminary mosaic image by using a simple watermarking technique called “Reversible Contrast Mapping(RCM)”.

The LR sequence, which consists the placement mappings of all the tile images in the original image has to be watermarked in the preliminary mosaic image too. For this, all the blocks of the preliminary mosaic are randomized using a function called random_blks, in which the function randperm( ) is used to generate a random sequence of permutations of the indices of the blocks. And this sequence can be retrieved back by using a function called "rng", which uses a seed as the secret key to get back the same sequence. This secret key plays a vital role in retrieving the original image.

A new image is created which consists of the random blocks of the preliminary mosaic. The sequence of bits of LR are embedded into this new mosaic image using the same watermarking technique that was used in embedding the information of the dimensions of the original image. The mosaic image thus produced that has all the information about the original image is the final mosaic image desired.

### 4.1 Watermarking scheme (LSB replacement technique):

The lossless least significant bit replacement method used in this project is a technique called "Reversible Contrast Mapping(RCM)", which is one of the methods for watermarking the image. This technique uses a mathematical integer transform on pairs of pixels. For some specific pairs of pixels even though the LSBs of the modified pixels are lost, the RCM can still be reversed. Altering just the LSBs of the pair of pixels will not change the visual appearance of the image, which is perfect for hiding the data.

Most of the reversible watermarking techniques have a data compressing stage, which amplify the computational complexity of watermarking. Although, there are some watermarking techniques that do not compress the data. For instance, circular histogram scheme does not compress the data but, its embedding capacity is very low.

#### 4.1.1 Reversible Contrast Mapping watermarking scheme:

Let [0, R] be range of colors of each pixel, where R = 255 and let (a,b) be a pair of pixels. The mathematical transformation applied to the pair of pixels is,

$$a' = 2a - b,\ b' = 2b - a \qquad (3)$$

To avoid underflowing or overflowing of the data, the transform is constrained to domain D Є [0, R] x [0, R] . Then the transforms would be,

$$0 \leq 2a - b \leq R, \quad 0 \leq 2b - a \leq R \tag{4}$$

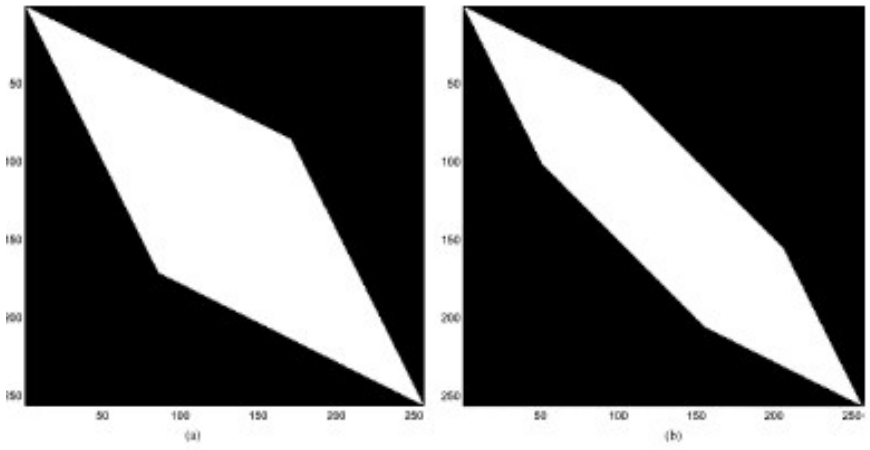


Figure 14 Represents the transform domain D: (a) without and (b) with controlled distribution.

The inverse transform of the above transform is defined as follows,

$$a = \text{ceil}\left[\frac{2}{3}a' + \frac{1}{3}b'\right] \quad , \quad b = \text{ceil}\left[\frac{1}{3}a' + \frac{2}{3}b'\right] \tag{5}$$

Even though the LSBs of the pixels that are transformed are lost, the forward and the reverse transform together yields to exact results. If a' and b' are not altered the reverse transform reverses the forward transform accurately, without the need of using the ceil function. In the technique of watermarking, the LSBs of a', b' are set to "0". So, if the LSB of a' was "1", the original pixel values a and d reduce by 2/3 and 1/3 respectively. Likewise, if the LSB of b' was "1", the associated original pixel values reduce by 1/3 and 2/3.

But, if the LSBs of both a' and b' are "1", then the ceil function helps in retrieving the right answers. From equation (3), it can be derived that, if (a', b') are both odd numbers then, automatically (a, b) are also odd numbers. Hence, we can say that in the domain D without the pair of odd numbers, although the LSBs of the transformed pairs are lost, the inverse transform reverses the pairs perfectly.

It has to be made sure that it does not change how the picture looks originally when the transform is applied on the pixels. So, if we consider the sum and the difference of the transforms, it would be “a’ + b’ = a + b” and “a’ – b’ = 3(a – b)”, respectively. Hence, it can be concluded that RCM conserves the color level averages, and increments the differences between the pixels after transformation, increasing the contrast of the image.

This process replaces the LSBs of the pairs of pixels that are transformed. In the process to retrieve the watermark and to get the original pairs of pixels, every pair that has been transformed has to be recognized accurately. For every pair of pixels, if the LSB of the first pixel is “1”, then the pair of pixels are transformed and if it is “0” the pixels are not transformed.

If (a, b) € D has odd values, the inverse RCM technique cannot recover the original pair of pixels. These pairs can be still utilized for embedding data if they are accurately recognized during the process of detection. This is resolved by changing the LSB of the first pixel to “0”. During the recovery process, LSBs of the pair of pixel are changed to “1” and the condition (4) is checked. If the pair of pixels satisfy all the conditions, it can be said that the pair had odd values. To uncertainties during recovery process, the odd pairs on the boundaries have to be deleted. The pairs that come under uncertainty domain can be identified by solving the equations: “2a – b = 1, 2b – a = 1, 2a – b = R, 2b – a = R”. This domain consists of 170 pairs. Additionally, let the domain which have all the uncertainty pairs be Dc.

### A. Watermarking

The watermarking procedure is as follows:

1. Sort the image into pairs of pixels.
2. For every pair (a, b):

   If (a, b) € Dc:

   a. And if the pixels have odd values, transform the pair according to the mathematical equation (3) and change the LSB of a’ to “1” and data can be encoded in the LSB of b’.
   b. And if the pixels do not have odd values, change the LSB of “a” to “0,” and data can be encoded in the LSB of “b”.

If (a, b) !€ Dc :

c. Change the LSB of "a" to "0," and save the original value of b.

3. Finally, watermark the entire image with the bits obtained in the previous steps.

Moreover, the saved LSB of the pair in the step 2c is embedded into the LSB of the nearest accessible transformed pair. Therefore, all the data required to retrieve the original pair of pixel is either embedded into the corresponding pair or a pair near it. Even if the image is cropped, apart from the pixels in the border, all the other pixels can be retrieved along with the information embedded in them.

**B. Detection and original recovery**

Retrieving the original pixels and information embedded in it is as follows:

1. Sort the whole image into pairs of pixels.
2. For every pair (a', b'):

   If the LSB of the transformed pixel a' is "0":

   a. And if it not in the range of domain Dc, change the LSBs of the transformed pair (a', b') to "1" and the original pair (a, b) is retrieved by modifying the LSB of a' with the respective value obtained from the watermark sequence.
   b. And if it is in the range of domain Dc, modify the LSBs of the transformed pair (a', b') to "1" and also save the LSB of b' in the retrieved watermark sequence and get back the original pixels by changing the LSBs of ( a', b') to "1".

   If the LSB of the transformed pixel a' is "1":

   c. Save the LSB of b' in the retrieved watermark sequence and change the LSBs of (a', b') to "0"and restore the original pair (a, b) by applying the inverse transform (5) to them.

## 5. ALGORITHM OF SECRET FRAGMENT VISIBLE MOSAIC IMAGE CREATION

Mathematically, calculate the height and width of the original image S, Hs and Ws respectively, with Zt being the size of each tile in the original image S, the total number of bits needed to indicate the index of a tile image is Nx , and the total number of bits needed to specify the recovery sequence of the original image LR is:

$$N = \frac{Ws \; x \; Hs}{Zt} \tag{6}$$

$$Nx = [\log_2 N] + 1 \tag{7}$$

$$N_R = N \; x \; Nx \tag{8}$$

Also, as the image used in this project is a RGB image, it has three color channel which can be used for embedding bits. Since the watermarking scheme works with only one color channel at a time the total number of bits that can possibly be embedded into one tile is given by:

$$NT = \frac{(3 \; x \; Zt)}{2} \tag{9}$$

### 5.1 Mosaic image creation algorithm

Phase 1 : To choose the most similar carrier image

Step 1: Split the entire image into tile images each of a predefined size of Zt, and compute the values of height Hs and width Ws of the tile image and calculate the total number of tile images in the original image S by using the equation (6).

Step 2: Choose a carrier image D0 from a preselected database, which is the most alike to the original image S, with help of the similarity measure defined in equation (1).

Phase 2 : Incorporating the most similar tiles into the corresponding carrier blocks

Step 3: Extract the h-feature values of all the carrier images stored in the database and compute the " h-feature" values of all the tile images in the original image.

Step 4: Implementing a greedy search algorithm and determine the most similar tile image si,sj,sk….. in the original image corresponding to the carrier blocks

d1,d2,d3…….in the carrier image using h-feature values calculated in the earlier steps and also save the mapping sequence as secret recovery sequence LR.

Step 5: Finally, incorporate the tile images into the respective carrier blocks to produce a preliminary mosaic image.

Stage 3—Watermarking the tile fitting data into the carrier image

Step 6: Generate a binary sequence with information of the width Ws and height Hs of each tile in the original image and also its size Zt and watermark it into the first two blocks of the preliminary mosaic image produced in the earlier

Step 7: Generate a sequence of binary bits LR.

Step 8: Generate a temporary image with blocks the mosaic image randomly selected using a random number generator and a seed and watermark this temporary image with the bits of LR.

Step 9: Finally, a mosaic image is generated which has all the information about the original image.

# 6. ORIGINAL IMAGE RETRIEVAL

Retrieval of the original image is backwards of the process of creating the mosaic image.. This algorithm is implemented as follows,

Step 1: Retrieve the width Ws and height Hs of S as well as the size Zt of the tile images from the first two blocks of image U in a raster-scan order using a reverse version of the lossless LSB replacement scheme.

Step 2: Calculate NR, the total length of the recovery sequence LR.

Step 3: Create a temp image which includes all other pixels which were not used for embedding the info of S. Then, repetitively select randomly a block from this temp image, utilizing the seed and the same random generator used in the creation process and implement the reverse watermarking scheme and obtain all the bits of LR.

Step 4: To get the index of each tile image convert every Nx bits of LR into a integer and this the sequence od indices LR = i,j,k…….

Step 5: Obtain the original placements of the tile images of original image in the respective carrier blocks.

Step 6: Generate the original image by considering the block image to be the corresponding tile image by utilizing the mappings obtained in the step above.

# 7. CONCLUSION

The mosaic image formed in this project is by composing the small parts of an input original image in the carrier image. Even though the small parts of the original image are visible to the observer, they are placed so randomly in the carrier image that people seeing the image cannot figure out what would the original image look like.

A improvised color scale is used in this project helped in developing a technique to measure image similarity called as h-features which not only made the process of measuring the similarity easier but also more effective.

The greedy search algorithm used in this project for scanning the tile images in the original image which are the most alike to fit the blocks of the carrier image, is efficient for the specific task for which it is used and is easy to implement. The information about the placement information of tiles into the carrier image blocks is embedded into the carrier image by randomly scrambling the blocks of the carrier image by using a secret key.

The LSB replacement technique is used to embed information of the original image into the scrambled carrier image blocks. Low Math complexity and lack of additional data compression makes this an apt technique. The retrieval of the original image is only possible with the help of the secret key which makes the whole process secure.

# 8. EXPERIMENTAL RESULTS

Figure 15 Original image

Figure 16 Carrier image 1

Figure 17 Carrier image 2

Figure 18 Carrier image 3

Figure 19 Carrier image 4

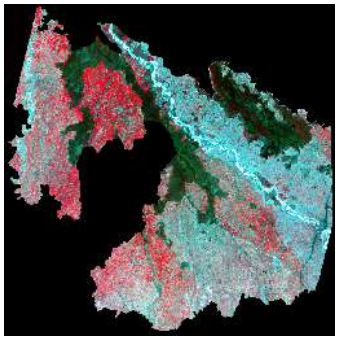

Figure 20 Carrier Image 5

Figure 21 Carrier Image 6

Figure 22 Carrier Image 7

Figure 23 Selected Carrier image

Figure 24  Final Mosaic Image

# REFERENCES


1. J. Lai and W. H. Tsai, "Secret-fragment-visible mosaic image—A new computer art and its application to information hiding," IEEE Trans. Inf. Forens. Secur., vol. 6, no. 3, pp. 936–945, Sep. 2011.
2. S. Battiato, G. Di Blasi, G. M. Farinella, and G. Gallo, "Digital mosaic framework: An overview," Eurograph.—Comput. Graph. Forum, vol. 26, no. 4, pp. 794–812, Dec. 2007.
3. P. Haeberli, "Paint by numbers: Abstract image representations," in Proc. SIGGRAPH, Dallas, TX, 1990, pp. 207–214.
4. Hausner, "Simulating decorative mosaics," in Proc. SIGGRAPH, Los Angeles, CA, Aug. 2001, pp. 573–580.
5. Y. Dobashi, T.Haga, H. Johan, and T. Nishita, "A method for creating mosaic image using voronoi diagrams," in Proc. Eurographics, Saarbrucken, Germany, Sep. 2002, pp. 341–348.
6. M. S. Lew, N. Sebe, C. Djeraba, and R. Jain, "Content-based multimedia information retrieval: State of the art and challenges," ACM Trans. Multimedia Comput., Commun., Appl., pp. 1–19, Feb. 2006..
7. D. Coltuc and J. M. Chassery, "Very fast watermarking by reversible contrast mapping," IEEE Signal Process. Lett., vol. 14, no. 4, pp. 255–258, Apr. 2007.
8. M. U. Celik, G. Sharma, A. M. Tekalp, and E. Saber, "Lossless generalized LSB data embedding," IEEE Trans. Image Process., vol. 14, no. 2, pp. 253–266, Feb. 2005.
9. D. Coltuc and J.-M. Chassery, E. J. Delp and P. W. Wong, Eds., "Simple reversible watermarking schemes: Further results," in Proc. SPIE: Security, Steganography, Watermarking Multimedia Contents VIII, 2006, vol. 6072, pp. 739–746

# APPENDIX A

Matlab code:

```
database = dir('C:\Users\rk717306\Desktop\database\*.tiff');
for i = 1 : length(database)
      image = strcat('C:\Users\rk717306\Desktop\database\',database(i).name);
       I = imread(image);
       func_red = @(block_struct) mean2(block_struct.data(:,:,1));
       r2 = blockproc(image,[4 4],func_red);
       func_green = @(block_struct) mean2(block_struct.data(:,:,2));
       g2 = blockproc(image,[4 4],func_green);
       func_blue = @(block_struct) mean2(block_struct.data(:,:,3));
       b2= blockproc(image,[4 4],func_blue);
        Im = uint8(zeros(64,64,3));
          Im(:,:,1) = r2;
       Im(:,:,2) = g2;
       Im(:,:,3) = b2;
       figure;
       subplot(2,2,1);
       imshow(I,[]);
       title('original image');
       subplot(2,2,2);
       imshow(Im,[]);
       title('after averaging');
       Nr =8;
       Ng = 8;
       Nb = 8;
       h_features = b2+ (Nb*r2) + (Nb*Nr*g2);
      DB{i} = hist(h_features);
      Hf{i} = h_features;
   end
 I = imread('lena.tiff');
```

```
rs2 = blockproc('lena.tiff',[4 4],func_red);
gs2 = blockproc('lena.tiff',[4 4],func_green);
bs2= blockproc('lena.tiff',[4 4],func_blue);
 h_features1 = bs2+ (Nb*rs2) + (Nb*Nr*gs2);
Hs = hist(h_features1);
Im_S = uint8(zeros(64,64,3));
 Im_S(:,:,1) = rs2;
        Im_S(:,:,2) = gs2;
        Im_S(:,:,3) = bs2;
        figure;
        subplot(2,1,1);
        imshow(I,[]);
        title('original image');
        subplot(2,1,2);
        imshow(Im_S,[]);
        title('after averaging');

for j = 1 : length(database)
     y = 0;
      Hd = DB{j};
    for k = 1 : size(Hd)
       x = abs(Hs(k) - Hd(k));
        y = y + x;
   end
 m(j) = 1/y;
 end
>> [maxi,loc] = max(m);
 tar_img = strcat('C:\Users\rk717306\Desktop\database\',database(loc).name);
D = imread(tar_img);
figure;
imshow(D)
```

%greedy search process to find most similar tile images in S corresponding to carrier blks in D, to construct secret recovery sequence LR

```
tar_hfeatures = Hf{loc};
for x = 1 : 64
      t_hf(:,x) = tar_hfeatures(x,:);
   end
for x = 1 : 64
      i_hf(:,x) = h_features1(x,:);
  end
 sim1 = cell(4096,1);
>> for i = 1 : 4096
       for x = 1 : 4096
          sim(x,i) = 1/(i_hf(x) - t_hf(i));
       end
       sim1{i} = vec2mat(sim(:,i),64);
        [max_sim(i),LR(i)] = max(sim1{i}( : ));
       [LR_row(i),LR_col(i)] = ind2sub(size(sim1{i}),LR(i)) ;
   end
>>  LR_row1 = vec2mat(LR_row,64);
LR_col1 = vec2mat(LR_col,64);
```

% fitting the tiles in S into the corresponding blks in carrier image D to generate preliminary mosaic image U_im

```
r_u = cell(64,64);
g_u  = cell(64,64);
b_u  = cell(64,64);
for i = 1 : 64
       for j = 1 : 64
                  r_u{i,j}  =  I((LR_row1(i,j)*4)-3:(LR_row1(i,j)*4),(LR_col1(i,j)*4)-3  :
(LR_col1(i,j)*4),1);
                  g_u{i,j}  =  I((LR_row1(i,j)*4)-3:(LR_row1(i,j)*4),(LR_col1(i,j)*4)-3  :
(LR_col1(i,j)*4),2);
```

```
                b_u{i,j}  =  I((LR_row1(i,j)*4)-3:(LR_row1(i,j)*4),(LR_col1(i,j)*4)-3  :
(LR_col1(i,j)*4),3);
      end
  end

r_u1 = cell2mat(r_u);
g_u1 = cell2mat(g_u);
b_u1 = cell2mat(b_u);

U_im = uint8(zeros(256,256,3));
U_im(:,:,1) = r_u1;
U_im(:,:,2) = g_u1;
U_im(:,:,3) = b_u1;
figure;
imshow(U_im);

%transforming LR into binary string with length NR

bin = de2bi(LR);
LR_seq = reshape(bin.',1,numel(bin));
[ws,hs,d] = size(I);
zt = [4 4];
S_info = [ws hs zt];
S_info_bin = de2bi(S_info,'left-msb');
S_info_binseq = mat2cell(S_info_bin,[4],[4,4,1]);
S_info_binseq{3}(:,2:4) = 0;
j = [1 3 5 7];
m = [2 4 6 8];

%checking all conditions and embedding the info into the pixels

for n = 1:3
```

```
      for k = 1:4
        for i = 1:4
            x = uint16(U_im(k,j(i),n)) ; y =uint16( U_im(k,m(i),n));
             x1 = de2bi(x,12); y1 = de2bi(y,12);
    if(~(2*x - y == 1) | ~(2*y - x == 1) | ~(2*x - y == 255) | ~(2*y - x == 255))
        if((x1(1) == 0) & (y1(1) == 0))  %because in maltab lsb comes first
          xt = (2*x - y) ; yt = (2*y - x);
          xt_bin = de2bi(xt,12); xt_bin(1) = 1; xtd = bi2de(xt_bin);U_im(k,j(i),n)=xtd;
          yt_bin = de2bi(yt,12) ; yt_bin(1) = S_info_binseq{n}(k,i);ytd = bi2de(yt_bin) ;
U_im(k,m(i),n) = ytd;
        elseif((x1(1) == 1) | (y1(1) == 1))
           x1(1) = 0 ; xtd = bi2de(x1) ; U_im(k,j(i),n) = xtd;
           y1(1) = S_info_binseq{n}(k,i); ytd = bi2de(y1) ; U_im(k,m(i),n) = ytd;
         end
         else
            x1(1) = 0; xtd = bi2de(x1) ; U_im(k,j(i),n) = xtd;
            ytd = bi2de(y1); U_im(k,m(i),n) = ytd;
          end
      end
  end
end
[indi,M,K] = random_blks(U_im1,[4,4,size(U_im1,3)]);
%creating a array of blocks of M of each size [4 4]
[rows cols colors] = size(M);
blkr = 4;
```

```
blkc = 4;
numrowblk = (rows/blkr);
numcolblk = (cols/blkc);
blkvecr = [blkr * ones(1,numrowblk)];
blkvecc = [blkc * ones(1,numcolblk)];
blks = mat2cell(M,blkvecr,blkvecc,colors);

LR_seqr = reshape(LR_seq,8,6144)';
LR_seqrf = zeros(11718,8);
LR_seqrf(1:6144,:) = LR_seqr;

U_im_new = uint8(zeros(size(M)));

new_blks = mat2cell(U_im_new,blkvecr,blkvecc,colors);


% watermarking one colorband at a time
for m1 = 1 : 3906
      new_blks{m1}(:,:,1) = watermarking(blks{m1}(:,:,1) ,LR_seqrf(m1,:));
  end

for m2 = 1 : 3906
       new_blks{m2}(:,:,2) = watermarking(blks{m2}(:,:,2),LR_seqrf((3906 + m2),:));
  end

 for m3 = 1 : 3906
       new_blks{m3}(:,:,3) = watermarking(blks{m3}(:,:,3),LR_seqrf((7812 + m2),:));
  end

%creating an image with the accumulated blocks

U_im_new = cell2mat(new_blks);

%creating the final scrambled image which has info of dim of original image and original
image itself

M1 = uint8(zeros(size(U_im)));
M1(1:4,1:8,:) = U_im(1:4,1:8,:);
M1 = U_im_new; %scrambled image with the info

recovered_img = reshape(M1(indi),size(U_im1)); %final mosaic image with info in it
```

Functions defined:

Random_blks.m:

```
function [indi,M1,K] = random_blks(M,S);

% check if the size of the blocks is specified for each dimension of M
NdimM = ndims(M) ;
sizM = size(M) ;  % size of the matrix
nS = numel(S) ;

if nS==1,
   S = repmat(S,1,NdimM) ; % scalar expansion
elseif nS ~= NdimM,
   error('Number of elements of S should equal the number of dimensions of M.') ;
end

% S should contain positive integers only
if ~isnumeric(S) || any(S ~= fix(S)) || any(S<1),
   error('S should be a numeric array with positive integer values.') ;
end

% vectors are 2D and S can be a scalar in this case:
% the singleton dimension of M should be taken care of.
if NdimM==2 && any(sizM==1) && nS==1 && all(S~=1),
   S(sizM==1) = 1 ;
end

if all(S==1),
   % Each element of M is a single block. We can just randomize the linear
   % indices.
   indj = randperm(numel(M)) ;
else

   nb = sizM ./ S ;  % how many blocks in each dimension, this should be an integer
   B = cell(NdimM,1) ;
    % for each dimension of M:
   for i=1:NdimM,
      % check if the number of blocks is an integer
      if nb(i) ~= fix(nb(i)),
         error(['Size mismatch: the size of the matrix M (= %d) in dimension %d\n' ...
               'is not an integer number of times the specified blocksize (= %d).'],...
            sizM(i),i,S(i)) ;
      else
         % expand the size in that dimension. B is used for mat2cell
         B{i} = repmat(S(i),1,nb(i)) ;
```

```
    end
  end

  % M can be a numerical or cell array. By randomizing the (linear)
  % indices, we do not have to worry about the actual contents of M.
  indj = reshape(1:numel(M),sizM) ;
  % convert to a cell array. Each cell contains a block.
  C = mat2cell(indj,B{:}) ;
  K = rng(92,'twister');
  C(randperm(numel(C))) = C ;
   % convert back from cell array
  indj = cell2mat(C) ;

end

% use these randomized block indices to shuffle M
M1 = reshape(M(indj),sizM) ;

if nargout>1,
  indi(indj) = 1:numel(M) ;
end
```

watermarking.m :

```
function [out_blk] = watermarking(blk_data,LR_seqr)

 LR_seq1 = vec2mat(LR_seqr,2);%changing it into a 2,4 mat to make it fit into the for
loop
 LR_mat = zeros(4,4);%adding zero columns in between to convert it into 4,4 mat to fit
into the for loop
 LR_mat(:,1:2:end) = LR_seq1;

 datamat = blk_data;

for r = 1 : 4
  for c = [1  3]
        x = uint16(datamat(r,c)) ; y = uint16(datamat(r,c+1));
          x1 = de2bi(x,12); y1 = de2bi(y,12);
      if(~(2*x - y == 1) | ~(2*y - x == 1) | ~(2*x - y == 255) | ~(2*y - x == 255))
        if((x1(1) == 0) & (y1(1) == 0))  %because in maltab lsb comes first
          xt = (2*x - y) ; yt = (2*y - x);
          xt_bin = de2bi(xt,12); xt_bin(1) = 1; xtd = bi2de(xt_bin);datamat(r,c) =xtd;
          yt_bin = de2bi(yt,12) ; yt_bin(1) = LR_mat(r,c); ytd = bi2de(yt_bin);
datamat(r,c+1) = ytd;
         elseif((x1(1) == 1) | (y1(1) == 1))
          x1(1) = 0 ; xtd = bi2de(x1) ; datamat(r,c) = xtd;
          y1(1) = LR_mat(r,c); ytd = bi2de(y1) ; datamat(r,c+1) = ytd;
```

```
            end

          else
             x1(1) = 0; xtd = bi2de(x1) ; datamat(r,c) = xtd;
             ytd = bi2de(y1); datamat(r,c+1) = ytd;
         end
       end
end

 if nargout == 1
   out_blk = datamat;
 end

end
```

rev_watermarking.m

```
function [seqrecov,recov_blk] = rev_watermarking(blk_data)

datamat = blk_data;
seqrecov1 = vec2mat(seqrecov,2);%changing it into a 2,4 mat to make it fit into the for
loop
seqrecov_mat = zeros(4,4);%adding zero columns in between to convert it into 4,4 mat to
fit into the for loop
seqrecov_mat(:,1:2:end) = seqrecov1;

for r = 1 : 4
   for c = [1  3]
          x = uint16(datamat(r,c)) ; y = uint16(datamat(r,c+1));
          x1 = de2bi(x,12); y1 = de2bi(y,12);
          if(~(2*x - y == 1) | ~(2*y - x == 1) | ~(2*x - y == 255) | ~(2*y - x == 255))
            if(x1(1) == 1)
              seqrecov_mat(r,c) = y1(1) ; x1(1) = 0; y1(1) = 0; x1dec = bi2de(x1); y1dec =
bi2de(y1);
              xt = ceil((2/3)*(x1dec)) + ceil((1/3)*(y1dec)) ; yt = ceil((1/3)*(x1dec)) +
ceil((2/3)*(y1dec));
              datamat(r,c) = xt; datamat(r,c+1) = yt;
            elseif(x1(1) == 0)
              seqrecov_mat = y1(1); x1(1) = 1; y1(1) = ~y1(1); x1dec = bi2de(x1); y1dec =
bi2de(y1);
              datamat(r,c) = x1dec; datamat(r,c+1) = y1dec;
            end
          elseif(x1(1) == 0)
              datamat(r,c) = x1; datamat(r,c+1) = y1;
          end
```

```
    end
end

if nargout > 1
    recov_blk = datamat;

  end

end
```