Digital Image Manipulation II:
Modern Applications of Linear Algebra

Oscar Alvarez and Hannah Grant
Texas Woman's University
June 2016

Applying Matrices to Computer Image Blending

1 Convex Combinations: Computer Rendering

A convex combination is a linear combination where both parts of the equation add up to one sole value, in this case, with a combined value of 1. A lowercase alpha (α) will the represent the controlling variable that determines the ratio of each side of the sum. This can be expressed by the following equation. We will be dealing with matrices in our project so our variables (J and B) are both matrices.

$$I = \alpha J + (1 - \alpha)B \tag{1}$$

In a convex combination, both α and $(1 - \alpha)$ are positive and the sum of the two must equal 1.

In terms of this type of image blending, α is the amount of image J showing in the final blended image, meanwhile, (1- α) is the amount of image B showing in the final blended image. In the end, both parts of the equation sum up the values of each image matrix to create 1 image.

We will be using this technique to create our own blended image using MATLAB. We will take one image, S, and a different image, B, and blend them together using equation (1).

2 Matrix Basics

We will need to know some basic linear algebra techniques to complete this project. In this section, we will learn how to multiply a matrix by a scalar (a one-dimensional number such as the number 3 or the number 8), and adding two matrices together to make one new matrix. First we can begin with multiplying a matrix by a scalar. Let us say

2.1 Scalar Multiplication with a Matrix

Let us say we have a scalar, k, and a matrix, A.

$$k = \begin{bmatrix}2\end{bmatrix} \qquad A = \begin{bmatrix}2 & 6 & 12\\ 4 & 7 & 4\end{bmatrix}$$

In order to multiply the scalar to the matrix, you would simply multiply the scalar to each element inside the matrix. This does not change the order of the values or shape of the matrix in any way. This is what it should look like.

$$\begin{aligned} k \cdot A &= \begin{bmatrix}2\end{bmatrix} \cdot \begin{bmatrix}2 & 6 & 12\\ 4 & 7 & 4\end{bmatrix} \\ &= \begin{bmatrix}(2\cdot 2) & (2\cdot 6) & (2\cdot 12)\\ (2\cdot 4) & (2\cdot 7) & (2\cdot 4)\end{bmatrix} \end{aligned}$$

Thus,

$$kA = \begin{bmatrix}4 & 12 & 24\\ 8 & 14 & 8\end{bmatrix}.$$

2.2 Matrix Addition

Let us say that we want to calculate the sum of matrices B and C, where

$$B_{(3x3)} = \begin{bmatrix}2 & 3 & 1\\ 1 & 5 & 3\\ 0 & 1 & 1\end{bmatrix} \quad \text{and} \quad C_{(3x3)} = \begin{bmatrix}1 & 2 & 7\\ 0 & 3 & 0\\ 2 & 2 & 1\end{bmatrix}$$

When performing matrix addition, both matrices must have equal dimensions because each element will be adding to its corresponding element. As shown above matrix B and matrix C both have dimensions of $3x3$, meaning that these

matrices meet the requirement for matrix addition. We now add each element to the element lying in the corresponding position to form one combined matrix. Our sum matrix should come out to be the same dimensions as our original two matrices.

$$B + C = \begin{bmatrix} (2+1) & (3+2) & (1+7) \\ (1+0) & (5+3) & (3+0) \\ (0+2) & (1+2) & (1+1) \end{bmatrix}$$
$$= \begin{bmatrix} 3 & 5 & 8 \\ 1 & 8 & 3 \\ 2 & 3 & 2 \end{bmatrix}$$

Therefore, $B+C$ has been combined into a new matrix. Let's call this matrix D

$$D = \begin{bmatrix} 3 & 5 & 8 \\ 1 & 8 & 3 \\ 2 & 3 & 2 \end{bmatrix}$$

3 MATLAB

MATLAB software is a powerful tool mathematicians can use when working with large matrices. We will be using MATLAB to alter an image and go over some basic coding structures.

3.1 Basic Coding Structures

These are some basic coding structures that we will need to manipulate an image using MATLAB.

3.1.1 The "For" Loop

A "for" loop is a command in MATLAB used to repeat a block of code for certain values or a certain amount of times. In a sense, we are constructing a row vector to determine how many times and what values we want to loop the block of code.

Let us produce some values, when we type

```
k=1:6
```

the result is:

```
k =

     1     2     3     4     5     6
```

We are actually creating a row vector that has the values of 1 through 6. If we wanted to create a row vector with specific values, we can create one by inserting the numbers in brackets only separated by a comma.

```
k_row_vector = [1,2,3,4,5,6]
```

Result:

```
k_row_vector =

     1     2     3     4     5     6
```

Let us say that we want to solve the function $y = x + 2$ for the numbers 16, 3, and 4. We can use a for loop to create a row vector with these values. In the block of code inside of the "for" loop, we would insert our equation that we would want to solve for. This will run the equation for the specified amount of times and for the specific values. Here is what a "for" loop would look like in this situation.

```
for x = [16,3,4];
    y = x+2
end
```

Our results will end up to be the values as requested.

```
y =

    18


y =

     5


y =

     6
```

Now we will do an example of a for loop using a matrix. First we will construct our matrix A. Then we will use a "for" loop to change some of our values.

```
1 -     A = [ [1 3 5]' [7 9 11]' [13 15 17]']
2
```

```
A =

     1     7    13
     3     9    15
     5    11    17
```

Let us say that we want to change the values of columns 2 through 3. We create a "for" loop for our matrix that only changes columns 2 through 3 using an equation.

```
4 -   for j=2:3,
5 -       A(j,:) = j^2+3
6 -   end
```

These are our results,

```
A =

     1     7    13
     7     7     7
     5    11    17


A =

     1     7    13
     7     7     7
    12    12    12
```

We can also place loops inside of loops which will make more sense once we begin applying the "for" loop in our SVD project code.

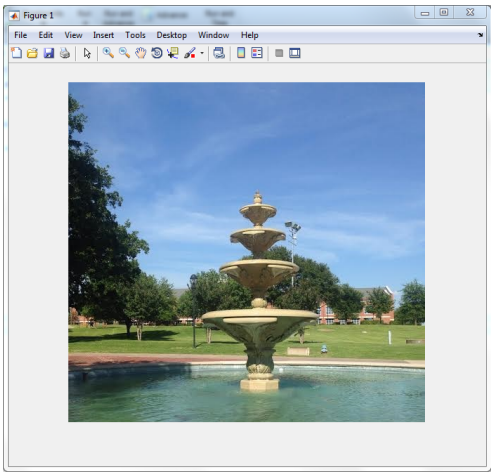
3.1.2 Importing/Altering an Image

First, we begin by saving the image in a location where MATLAB can find it. Usually, it would be a good practice to create a folder where you will be keeping all of your MATLAB files for the particular project. To import an image into MATLAB, we will need to run the following command:

```
1 -    VariableName = imread('fountain.jpg');
```

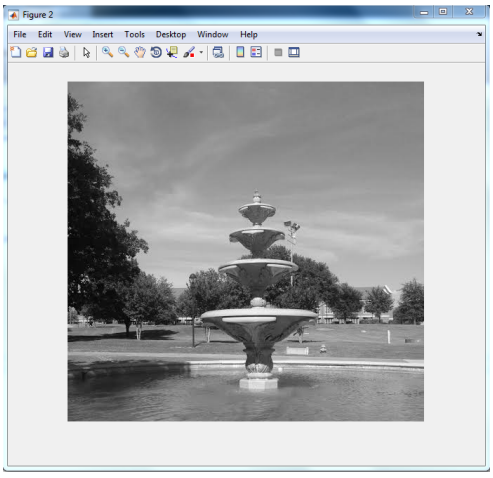
This will import our image and save it as a new variable that we can later use. If we want to view our image, you can display it in a new window where it will be represented by a figure (a figure is the new window that will pop up containing the image).

```
3 -    figure;
4 -    imshow(VariableName)
5
```



We can now alter the image with code directly. Here is an example; in order to create a black and white version of the image, we can input the command

```
6 -    NewVariableName = rgb2gray(VariableName);
7
8 -    figure;
9 -    imshow(NewVariableName)
```



4 Blending Two Images with MATLAB

Overlapping two images can be possible with these techniques derived from linear algebra and applying some simple coding commands into MATLAB. We will begin.

First off, we will need the two images that we will be wanting to overlap. (*NOTE THAT BOTH IMAGES MUST BE THE SAME SIZE for this project*). Dealing with images of different sizes is a longer, more difficult process. Here, we have chosen a picture of a local town book shop and a picture of a squirrel.

Using an image editing software, we cut the squirrel out and rearranged it in a way to appear bigger for the blending (GIMP was used for this example and Photoshop is an acceptable program, we can cut the squirrel out of the original picture). Once the squirrel was separated, we placed it on top of a white background; this makes it easier to see the two images separately when blending together. Black can also be used when dealing with two images that have a darker background. Make sure to keep the image dimensions the same between the two images.

Now that we have chosen our two images and edited them, we will begin the image blending process in MATLAB.
First, we will clear out all variables in MATLAB by placing the command "clear" at the beginning of our script. This will ensure that the code runs smoothly. We begin by importing the images into MATLAB and converting them into black and white (See 2.1.2). Choose an image to import. the order in which you import the images does not matter. For this example, we chose to import the image of the squirrel first, and the town shop second. This set of code imports the images and converts them to black and white. Afterwards, the image is converted into a matrix with values of double precision.

```
1 -    clear
2
3 -    sqrl = imread('SquirrelWithWhite.png');
4 -    BWsqrl = rgb2gray(sqrl);
5 -    S = double(BWsqrl);
6

8 -    bgbooks = imread('OperaHouse.JPG');
9 -    BWbgbooks = rgb2gray(bgbooks);
10 -   B = double(BWbgbooks);
11
```

Now that we have imported and converted the images into matrices labeled as variables, we can begin to apply the matrices created into our equation.

We will use a "for" loop in order to produce several α values at once.

As we use matrices in MATLAB, we need to be sure that they are identified as matrices in MATLAB. Within the equation, be sure to identify the matrices with the Variable name that you have chosen then parenthesis containing the dimensions that you want to be affected in the process of the equation. The colons tell MATLAB that you want to include all of the values in the rows and/or columns.

```
13 -   for alpha = [.1,.3,.5,.7,.9];
14
15 -       Blended = alpha*S(:,:) +(1-alpha)*B(:,:);
16
```

These are the values we have chosen for "alpha", .1, .3, .5, .7, and .9. This "for" loop will repeat the process of creating a new image for each of the "alpha" values. Inside the "for" loop and below the "alpha" values, is the equation that will blend both images together. We saved and named our new image matrix

as "Blended" to use later.

Now we need to tell MATLAB to show or display the images that we want to see. Placing this command inside the "for" loop means that it will produce an image in a new window for each "alpha" value.

```
    figure;
    imshow(uint8(Blended))

end
```

The "uint8" is necessary because MATLAB needs to know that you want to display the entire image as a whole - "uint8" contains all of the 8-bit data that is stored in variable "Blended."

This is the entire program code together that we have written for this image blending tutorial.

```
clear

sqrl = imread('SquirrelWithWhite.png');
BWsqrl = rgb2gray(sqrl);
S = double(BWsqrl);


bgbooks = imread('OperaHouse.JPG');
BWbgbooks = rgb2gray(bgbooks);
B = double(BWbgbooks);


for alpha = [.1,.3,.5,.7,.9];

    Blended = alpha*S(:,:) +(1-alpha)*B(:,:);

    figure;
    imshow(uint8(Blended))

end

```

Now we will run the code.

(a) α value .1 (b) α value .3 (c) α value .5 (d) α value .7 (e) α value .9

These are the images that were produced in MATLAB shown in respective order of their "alpha" values.

As a result, we can observe that as the α value increases, the image matrix that it is associated with (the squirrel) becomes more visible. The increasing α value also causes the values of the other image matrix (the town shop) to decrease, causing it to fade. The lack of values from one image is made up by the other image. This difference creates somewhat of a blending effect with our two images, where they do not necessarily overlap, but instead, blend in with one another.