Image Transformations

Paul Vu
301169550
January 27th 2017

SFU

Project Overview

This project is broken into 2 parts. In the first part of the project I performed 10 transformations using variations of the main.m file provided. In there I perform rotations, translations, scaling, affine, shearing and non linear transformations on a grid.

In second part of the project I investigate the problem of rotating and translating images.

1st Transformation: Pure rotation of 180 degrees

For this transformation, I rotate the grid by 180 degrees only. This is confirmed by my point (0,0) rotating from the bottom left corner, to the top right corner, while my furthest point (4,4) rotates to (-4. -4). Here we can see the shape and spacing of the grid is maintained since no scaling or translations were performed.

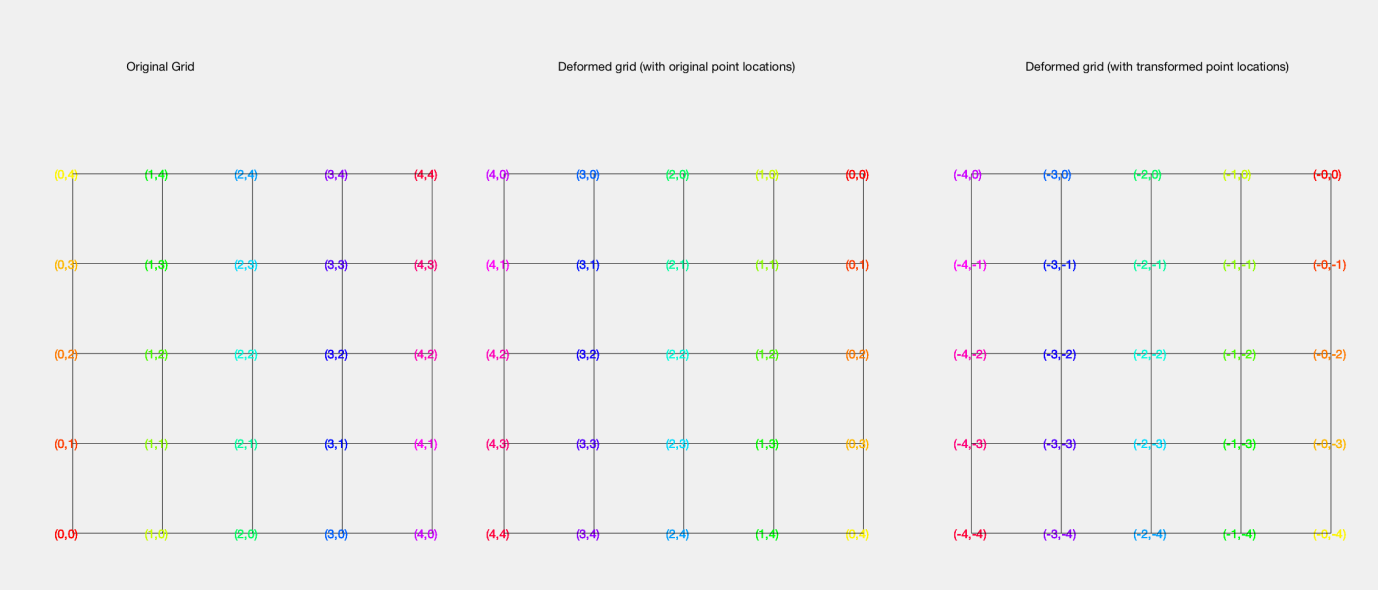


Figure1: Pure Rotation of 180 degrees

```
%-- 1st Translation by 180 degree pure rotation ----------------
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 180;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
```

Figure 2: Code for 1st transformation

2nd Transformation: Pure Translation by 10 in x direction

For this transformation, I rotate the grid by 0 degrees since it is a pure translation. I then do a pure translation in the X direction by adding 10 to all the x values and 0 to all the y values. This is confirmed when (0,0) is translated to (10,0). Again, our grid remains the same shape and spacing as there is no scaling or rotation.

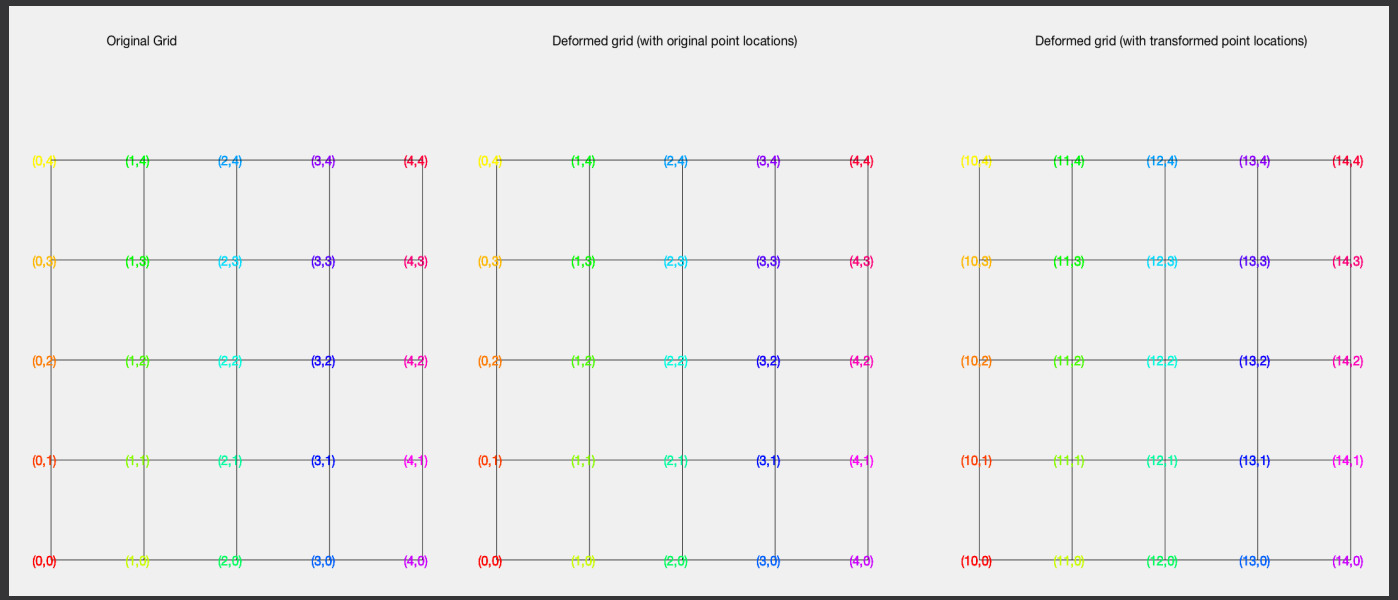


Figure 3: Pure Translation by 10 in x direction

```
%-- 2nd Transformation Pure Translation by 10 units in the x direction ----------
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 0;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX = FX + 10;
    FY = FY + 0;
    %FX = real(F); FY =imag(F); % extract x and y coordinate from the complex result
```

Figure 4: Code for 2nd transformation

3rd Transformation: Pure Scaling by 10 in x & y

For this transformation, I rotate the grid by 0 degrees since it is a pure scaling. I then do a pure scaling in the X direction & Y direction by multiplying each point by 10. This is confirmed when (0,0) is translated to (0,0), however (3,4) is translated to (30, 40). Our grid remains the same shape since we scaled in both directions by 10.

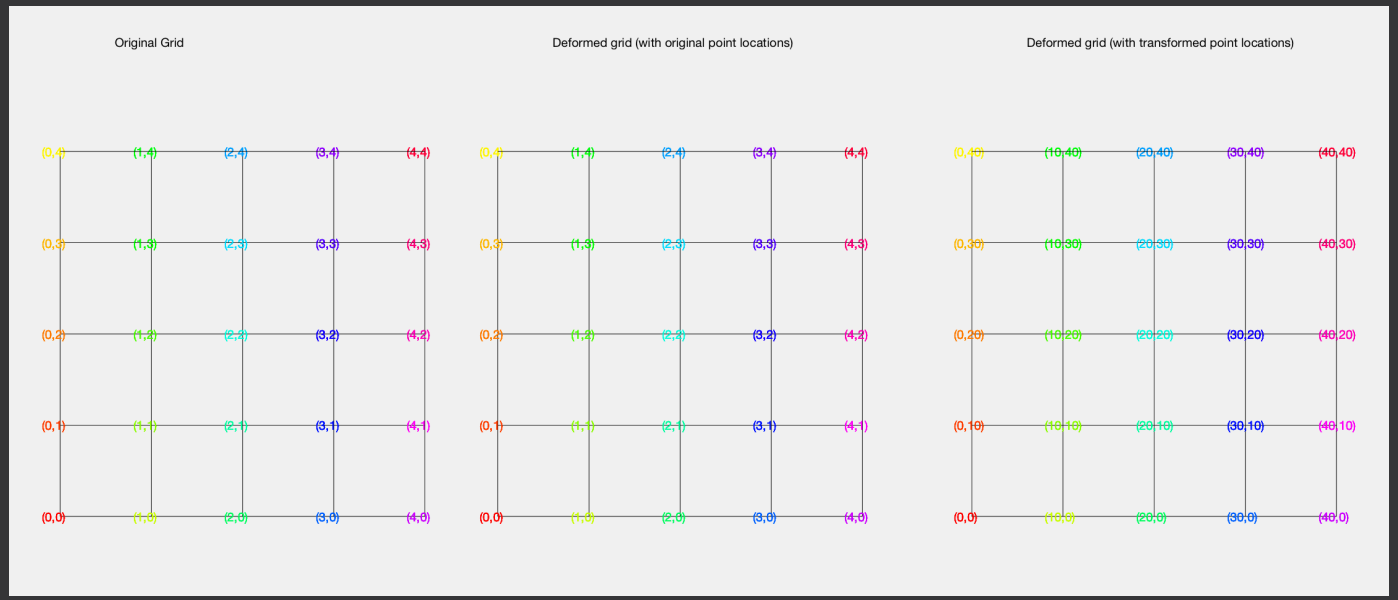


Figure 5: Pure Scaling by 10

```
%-- 3rd Transformation  Scaling by 10 units in the x & y direction ----------------
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 0;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX = FX.*10;
    FY = FY.*10;
    %FX = real(F); FY =imag(F); % extract x and y coordinate from the complex result
```

Figure 6: Code for 3rd transformation

4th Transformation: Vertical shear with coefficient m = 2

For this transformation, I rotate the grid by 0 degrees since I am evaluating pure Vertical shear. In a vertical shear, we will translate our (x,y) components to (x', y'). In a vertical shear x' = x, and y = mx + y, where m is the shear coefficient. Our results are confirmed when we see that the point (4,3) is transformed to (4, 4*2 + 3) = (4, 11).

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ mx + y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ m & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Figure 7: Vertical Shear Formula

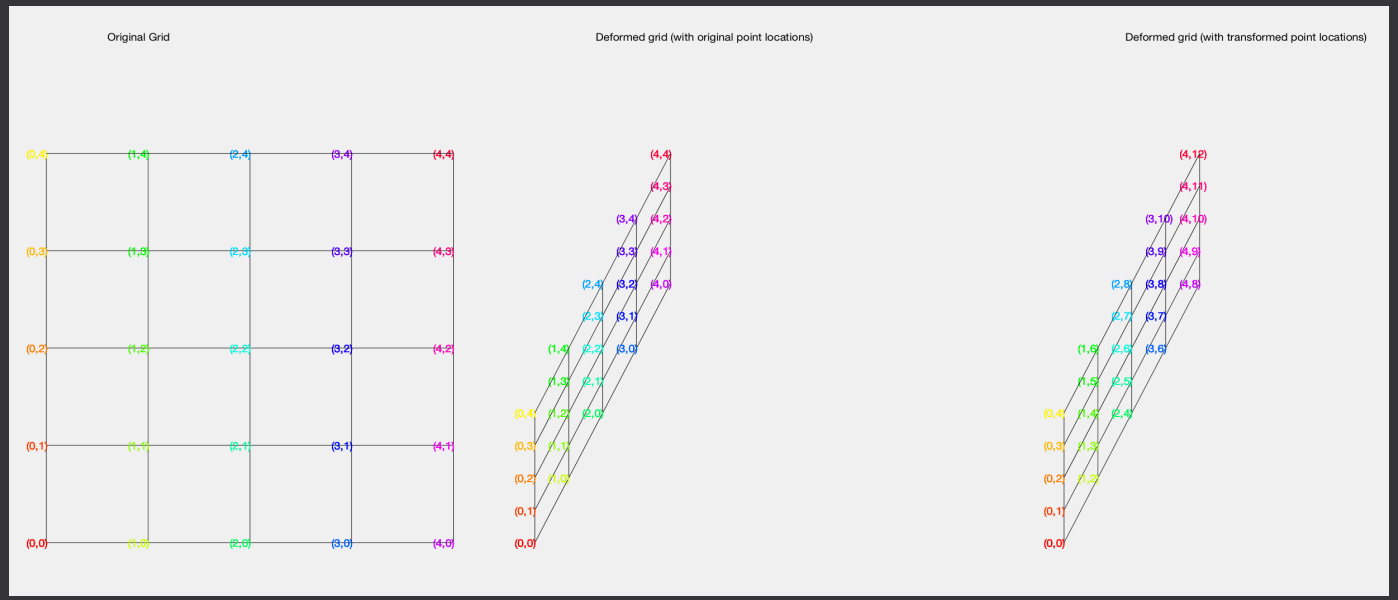


Figure 8: Vertical Shearing with m =2

```
%-- 4th Transformation: Vertical shear with coefficient m = 2 ---------------
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 0;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX = FX;
    FY = FY + FX.*2;
    %FX = real(F); FY =imag(F); % extract x and y coordinate from the complex result
```

Figure 9: Code for 4th transformation

5th Transformation: Affine

For affine transformations, I will use a combination of a shear, scale and translation transformation. An example equation of an affine transformation is given from wikipedia in figure 10. Here the x' = y and y' = x scaled by 2 + y. Both points are then shifted by -100. For my affine transformation I will utilize the same parameters and can confirm that the point (3,4) translates to (4, 10) I will then shift x and y by + 10 and thus (3,4) will be mapped to (14, 20).

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -100 \\ -100 \end{bmatrix}$$

Figure 10: Example Affine Formula from wikipedia

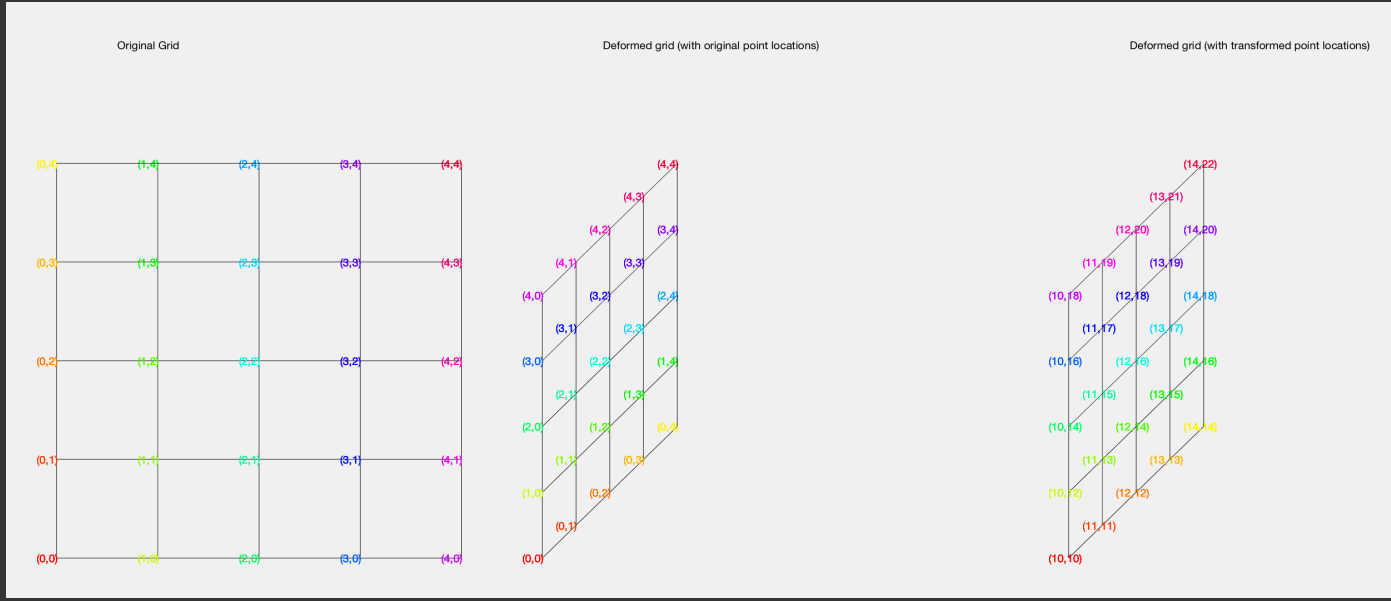


Figure 11: Affine Transformation

```
%-- 5th Transformation: Affine ---------------
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 0;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX_temp = FX;
    FX = FY + 10;
    FY = (FX_temp.*2 + FY) + 10;
    %FX = real(F); FY =imag(F); % extract x and y coordinate from the complex result
```

Figure 12: Code for 5th transformation

6th Transformation: Non Linear

For my non-linear transformation I chose x' and y' such that they were non linear values. As shown in figure 13 my x' and y' are clearly non linear equations. Thus plotting the transformation as shown in figure 14, the points are transformed non linearly.

$$x' = x^2 + xy + y$$
$$y' = y^2 + xy - x$$

Figure 13: Equation used to non linearize my x' and y' points

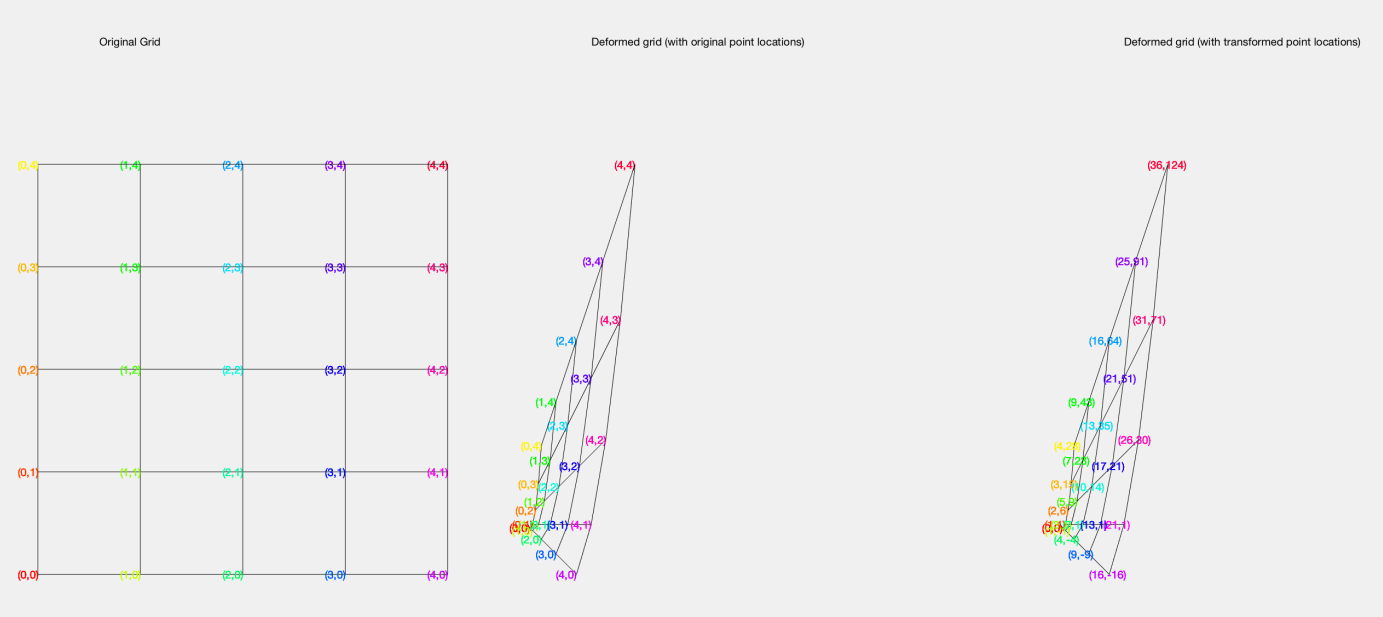


Figure 14: Non Linear transformation

```
%-- 6th Transformation: Non linear ---------------
% x' = X^2 + xy + y
% y' = y^2 + xy + x
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 0;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX = (FX.^2) + FX.*FY + FY;
    FY = (FY.^2) + FX.*FY - FX;
```

Figure 15: Code for 6th transformation

7th Transformation: Rotation & Translation

Below is a rotational and translational transformation. I first rotate the grid by 45 degrees and then shift it in the y direction by 5 units in the y direction. It is clear my origin is now (0,5) however it's interesting how (4,4) is shifted to (0,11) as a 45 degree rotation will map it back to the y axis.

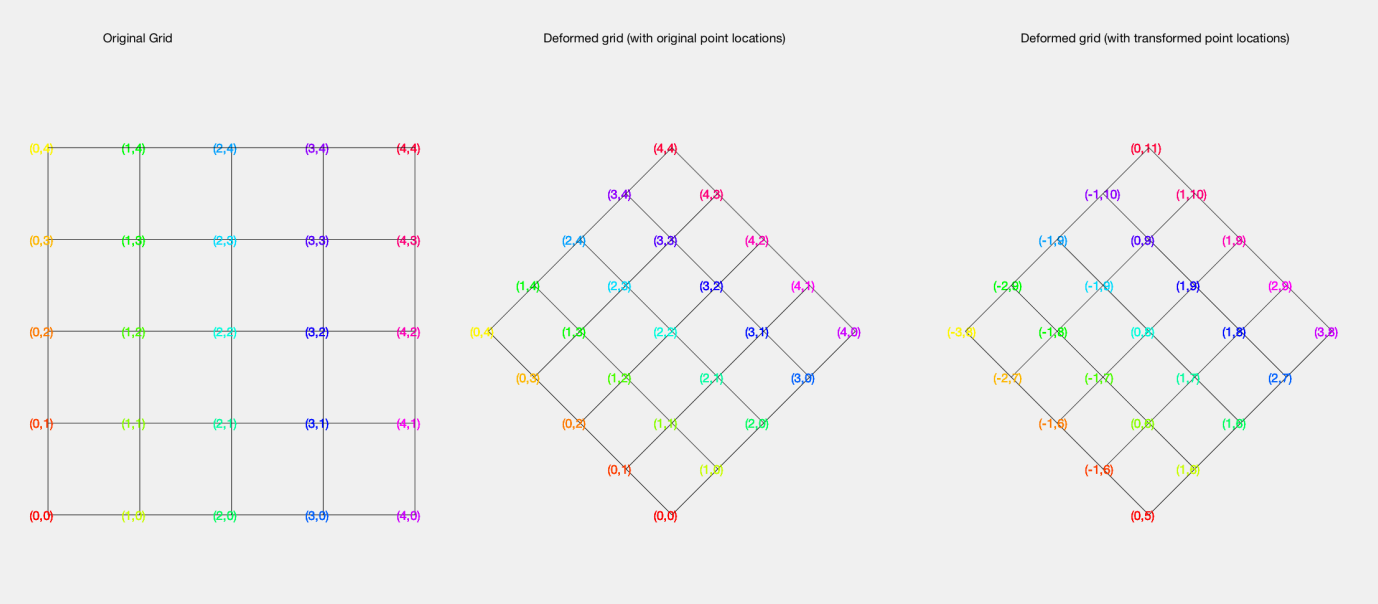


Figure 16: Rotational & Translational Transform

```
%-- 7th Transformation Translational Rotational ---------------
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 45;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX = FX;
    FY = FY + 5;
    %FX = real(F); FY =imag(F); % extract x and y coordinate from the complex result
```

Figure 17: Code for 7th transformation

8th Transformation: Scaling & Rotate & Shear

Below are the results from a grid that was rotated by 45 degrees, then sheared by a factor of m = 2. The grid was then scaled in the y direction by 2.

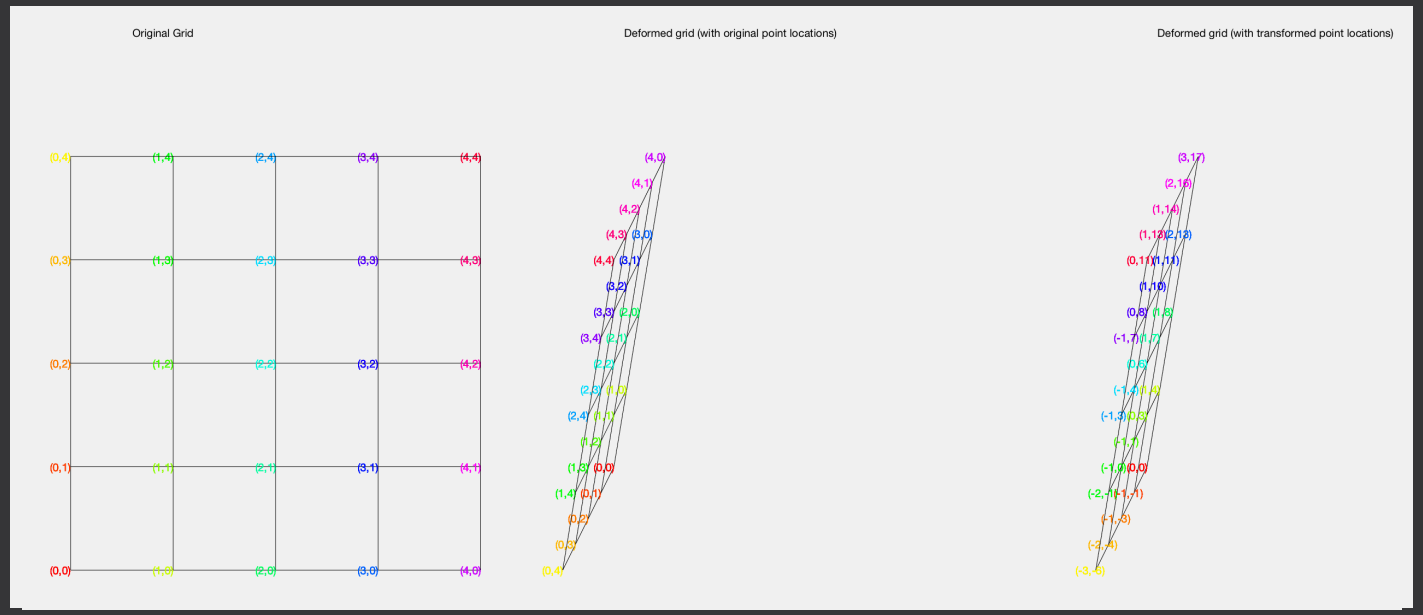


Figure 18: Rotational & Shear Transform

```
%-- 8th Transformation Translational Rotational Scaling and Shearing---------------
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 45;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX = FX;
    FY = 2.*(FY + FX.*2);
    %FX = real(F); FY =imag(F); % extract x and y coordinate from the complex result
```

Figure 19: Code for 8th transformation

9th Transformation: Non-linear Shear

In this transformation I take my previous non linear transformation, rotate it by 45 degrees and then shear it by a factor of m = 2.

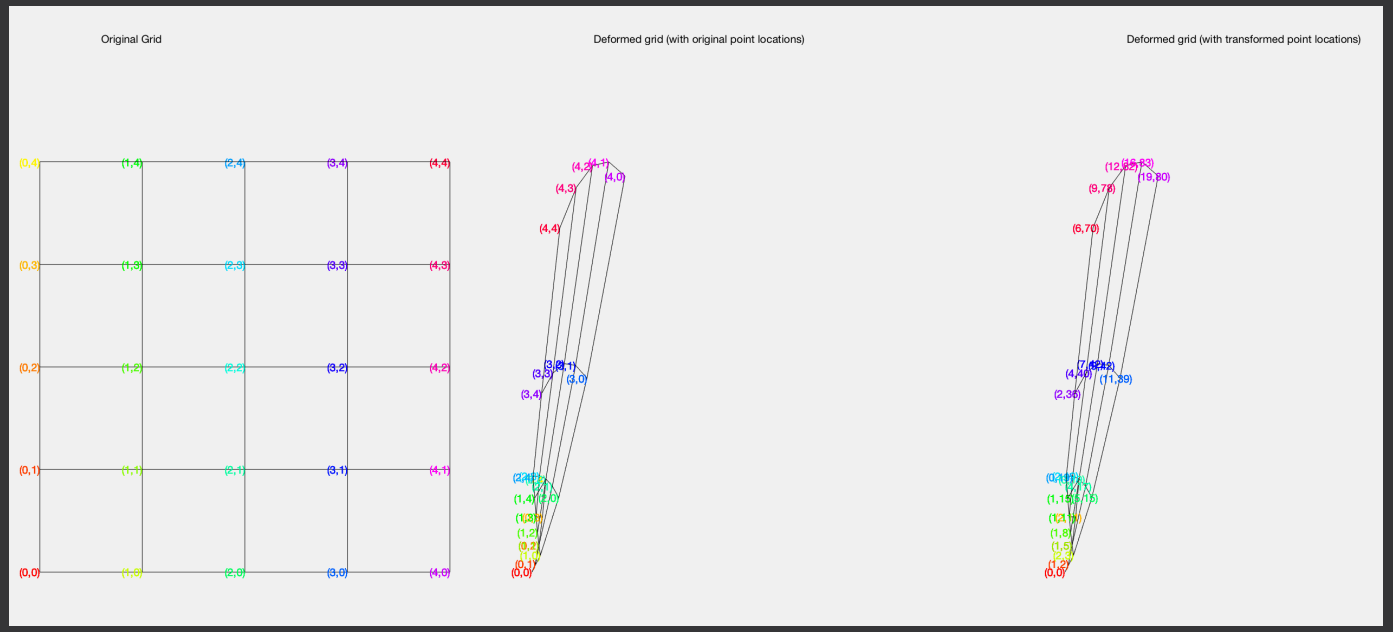


Figure 20: Non-linear Shear Transformation

```
%-- 9th Transformation Rotate, Non linear and shear---------------
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 45;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX = (FX.^2) + FX.*FY + FY;
    FY = (FY.^2) + FX.*FY - FX;
    FX = FX;
    FY = FY + FX.*2;
```

Figure 21: Code for 9th transformation

10th Transformation: Non Linear + Rotation

For my non-linear transformation I chose x' and y' such that they were non linear values different from my 6th transformation. I also rotate the image by 45 degrees. As shown in figure 22 my x' and y' are clearly non linear equations. Thus plotting the transformation as shown in figure 23, the points are transformed non linearly.

$$x' = x^2 + y$$
$$y' = y^2 - x$$

Figure 22: Equation used to non linearize my x' and y' points

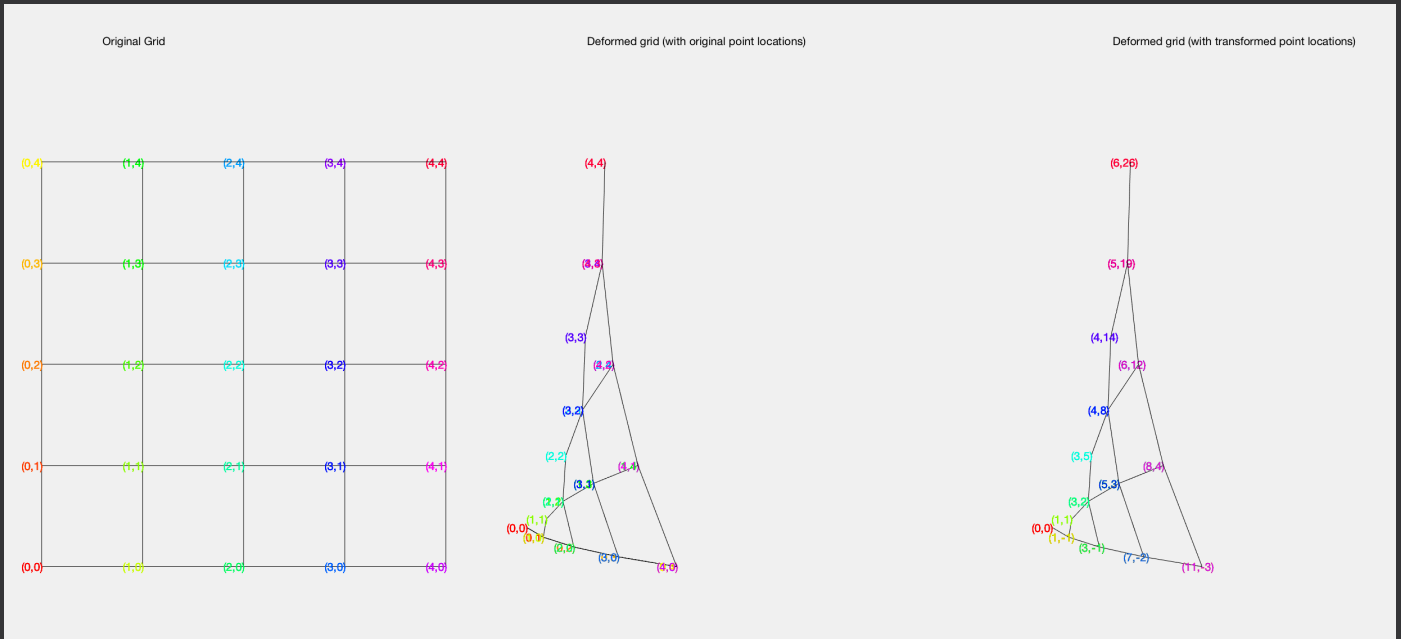


Figure 23: Non Linear transformation

```
%-- 10th Transformation: Non linear ---------------
% x' = X^2 + xy + y
% y' = y^2 + xy + x
   %define f(x,y) = ( fx(x,y), fy(x,y) ) where each x is mapped to fx(x,y)
   %and y is mapped to fy(x,y)
   % simple example fx = x cos(t) - y sin(t), and fy = x sin(t) + y cos(t)
   % is a rotation by angle t

    theta = 45;
    ct = cos(theta*pi/180); st = sin(theta*pi/180);
    FX = X.*ct - Y.* st;
    FY = X.*st + Y.*ct;
    FX = (FX.^2)  + FY;
    FY = (FY.^2) - FX;


    %FX = real(F); FY =imag(F); % extract x and y coordinate from the complex result
```

Figure 24: Code for 10th transformation

Part 2: Philosophy

In my design choice, I had to handle the problem of sampling when the image is out of view during a rotation. Since matlab rotates around (0,0) in the counter clockwise direction, If I were to follow that convention I would lose my FOV of the image. To get around that I'd have to rotate and shift to keep the FOV in picture. In my algorithm, I chose to work on a large canvas, for example I'd make a canvas of black pixels which was larger than my image. I would then rotate the image around the center to ensure that the image's FOV was preserved.

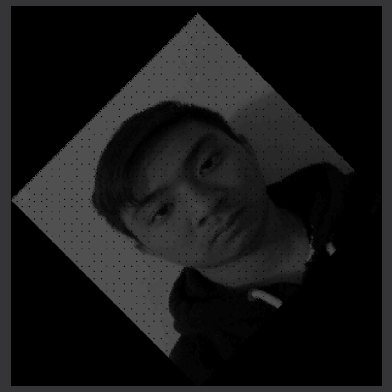


Figure 25: Example Rotations

Part 2: Philosophy cont…

In my design choice, I had to handle the problem of translating an image. If a user wanted to shift the image significantly, in theory the image could be off the map. There would be no way to see where the image was with respect to the origin! I chose to stretch my canvas such that my origin was the center of the black matrix, and any translation performed would be done with respect to the center. In the case the user translates the image off the map, my matrix will actually elongate and keep the image's FOV to illustrate the translation.

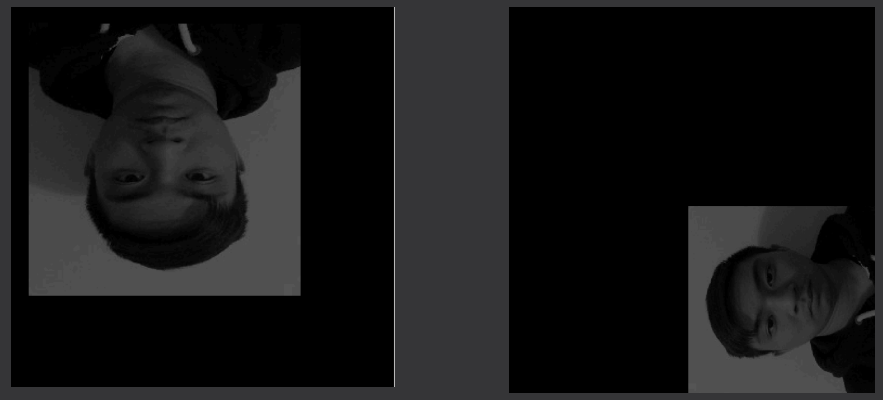


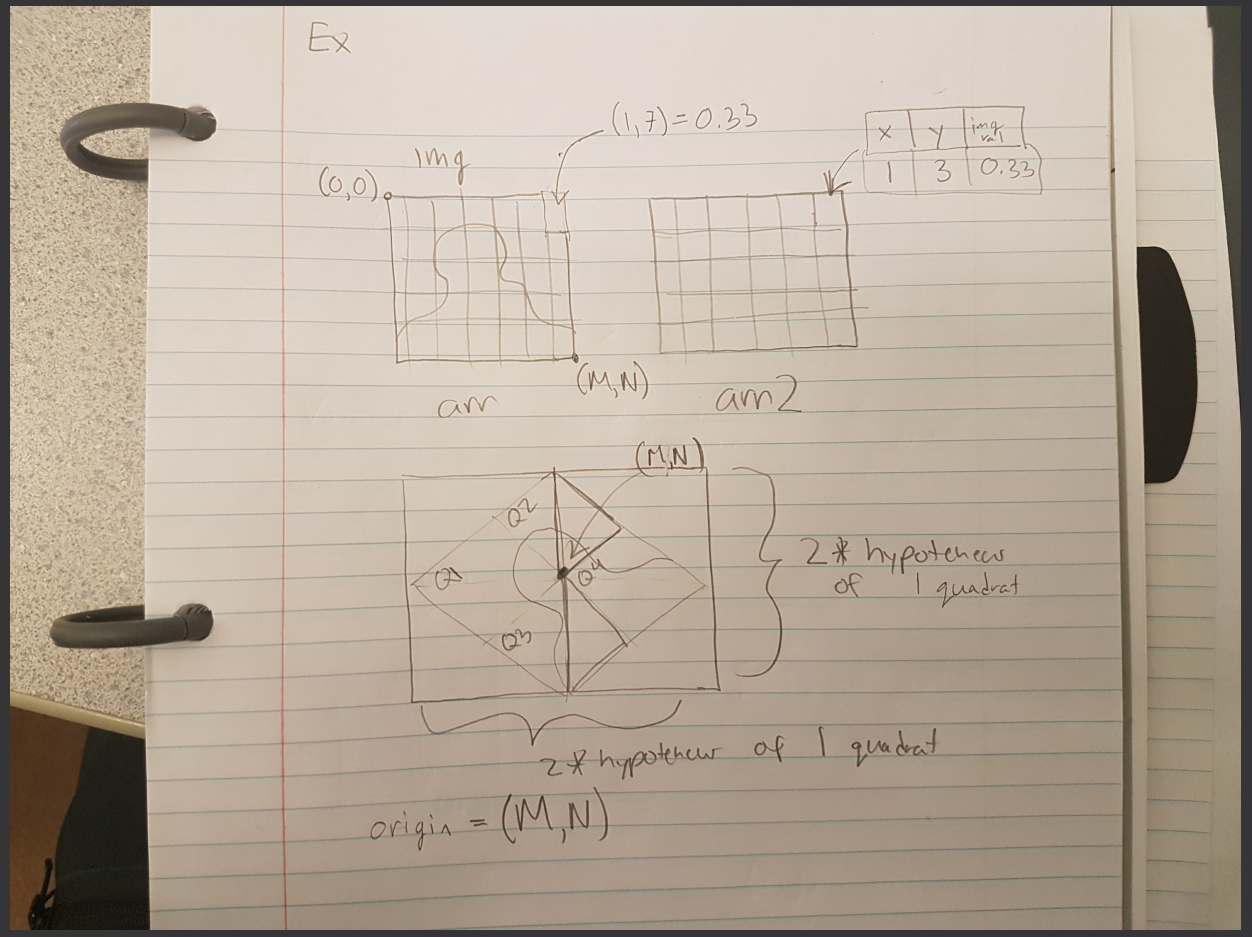
Figure 26: Example Translations

Part 2: Philosophy cont…

In my design choice, I had to handle the problem of translating an image. If a user wanted to shift the image significantly, in theory the image could be off the map. There would be no way to see where the image was with respect to the origin! I chose to stretch my canvas such that my origin was the center of the black matrix, and any translation performed would be done with respect to the center. In the case the user translates the image off the map, my matrix will actually elongate and keep the image's FOV to illustrate the translation.

Part 2: Philosophy cont…

In Transforming the image, I chose to follow the forward transformation as shown in the below equation. My rotation matrix is derived from the main.m code provided and in lecture. A example of my logic is provided below:



Ex.1: Illustration of my algorithm

$$\varphi(x) = Rx + T$$

Eq.1: Transformation Eqaution

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 - x_c \\ x_2 - y_c \end{pmatrix}$$

Eq.2: Rotation matrix * image coordinate matrix

Part 2: Transformation #1

For this transformation, I simply rotated the image by 45 degrees without shifting. There is a error in my code as I don't take in all the samples from my image. As you can see, my rotation didn't sample correctly and I determined that the error is in my sampling since I am taking the floor and ceil values from my rotation function. Therefore there are certain samples that get missed . On the positive side, my image rotates correctly, and is not cropped out since I'm rotating around the center and not at (0,0).

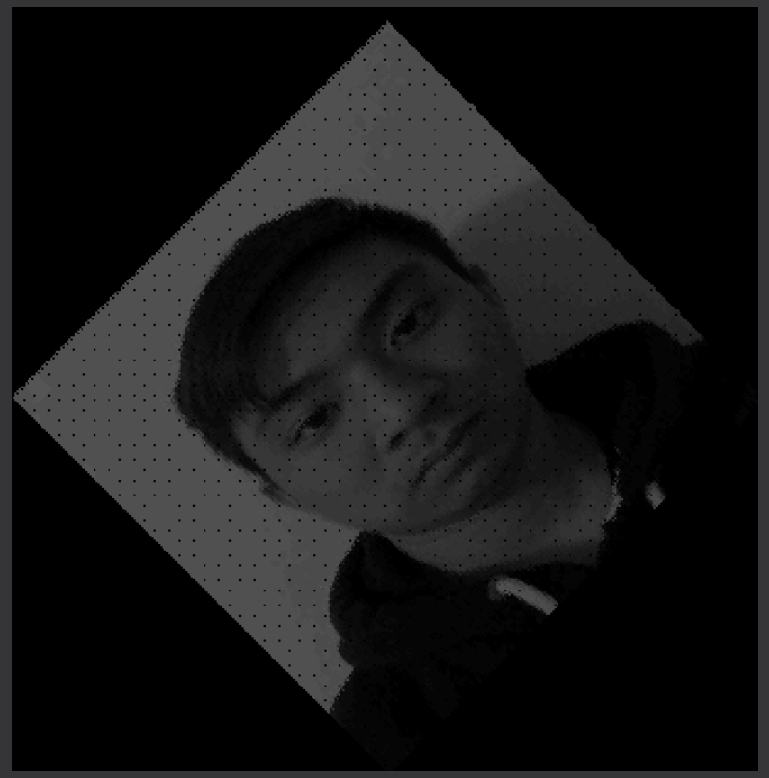


Figure 27: 45 degree rotation with no translation

Part 2: Transformation #2

In this rotation my algorithm works perfectly since I am not missing any samples from the ceil and floor rounding. It also illustrates that I can shift the image with respect to the center in the –x and -y direction.

Figure 28: 180 degree rotation with [-30,-30] translation

Part 2: Transformation #3

In this rotation my algorithm works perfectly since I am not missing any samples from the ceil and floor rounding. It also illustrates that I can shift the image with respect to the center in the +x and +y direction.

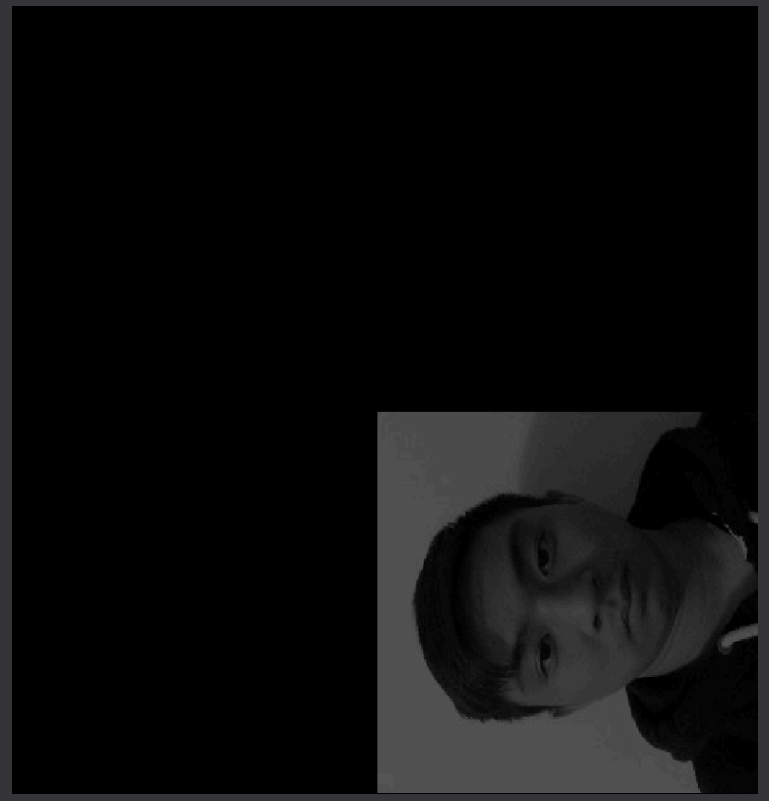


Figure 29: 180 degree rotation with [100,100] translation