IMAGE MANİPULATİON WİTH AFFİNE MATRİX

Homework #1

Abdurrahman Ahmet Daştan
200104004127

Index

1. **What is the Affine Matrix and Image Manipulation Methods**
2. **Zoom Operation**
   - **2.1. Forward Method**
   - **2.2. Backward Method**
   - **2.3. Backward Method with Bilinear Interpolation**
3. **Scale Operation**
   - **3.1. Forward Method**
   - **3.2. Backward Method**
   - **3.3. Backward Method with Bilinear Interpolation**
4. **Horizontal Shear Operation**
   - **4.1. Forward Method**
   - **4.2. Backward Method and Backward Method with Bilinear Interpolation**
5. **Rotation Operation**
   - **5.1. Rotation Operations**
   - **5.2. Images of Rotation Operations**
6. **Conclusion**
7. **Append**
   - **7.1. Forward Mapping Function**
   - **7.2. Backward Mapping Function**
   - **7.3. Backward Mapping Bilinear Interpolation Function**
   - **7.4. Bilinear Function**
   - **7.5. Inverse Matrix Function**
8. **Sources**

Image Manipulation with Affine Matrix

1. What is the Affine Matrix and Image Manipulation Methods

This homework assignment includes 4 operations these scale, zoom, shear, and rotate. We will do this operations with using 3 approach these forward mapping, backward mapping, and backward mapping by using bilinear interpolation. We will compare each operations result according to the 3 methods.

For each operation I will share original image and different 3 result of the solving methods. Firstly and shortly, I want to explain what is the forward mapping, backward mapping, and interpolation. And also explain how to work their logic. After that we can compare each operation with our methods.

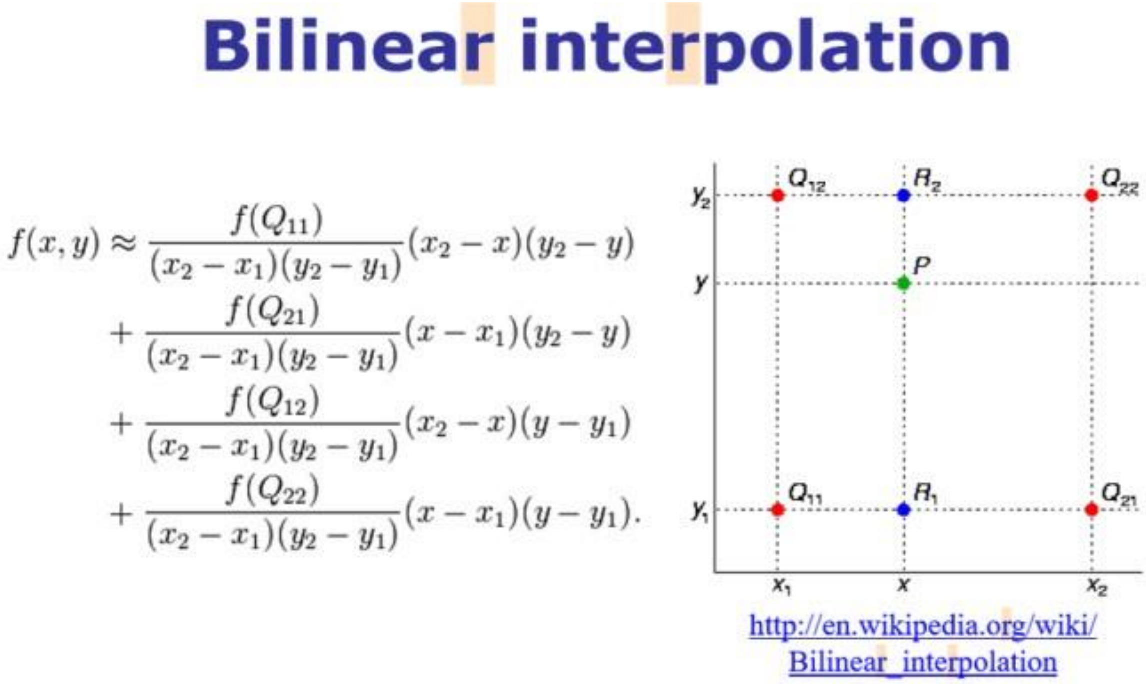
Affine Matrix, is used for manipulate image doing calculate each pixel new position. It is 2x3 matrix and their each values have different effect on the image. It is useful because our coordinate change linearly after calculation.

Forward Mapping, is direct operation of finding new image pixel coordinate. So we **multiply** affine matrix with source image coordinate and it result give us new coordinate of the new image.

Backward Mapping, is inverse of the forward mapping. Backward Mapping, try to find of each coordinate of the new image, it is **inverse of the affine matrix** operation. I will be explain in the example of operations. We make inverse operation because of that (AffineMatrix^-1 * tagetImage = sourceImage * AffineMatrix * AffineMatrix^-1) with this approach we gain target result coordinate.

Bilinear interpolation is kind of interpolation method. With this method, we avoid bluer image result that comes backward mapping. Idea is not getting target coordinate as a backward mapping, instead taking average of adjancent pixel. Here are bilinear interpolation formula.

Figure 1: Bilinear Interpolation



Meanwhile, I wanna show you difference between bilinear interpolation and without using bilinear interpolation. We get smooth pixel if we use bilinear interpolation. But you should zoom the image. Figure 2 and Figure 3 show that different.

Figure 2: Backward with interpolation

Figure 3: Backward without interpolation

We manipulate image that given below(Figure 3).

Figure 4: Original image

2. Zoom Operation

Zoom operation actually is a scale operation but new image size is same as a original image size. Our method will be change but affine matrix values are same.

For doing this operation our affine matrix operation values are:

a = 1.4, b = 0, tx = 1;

c = 0, d = 1.4, ty = 1;

2.1. Forward Mapping Method:

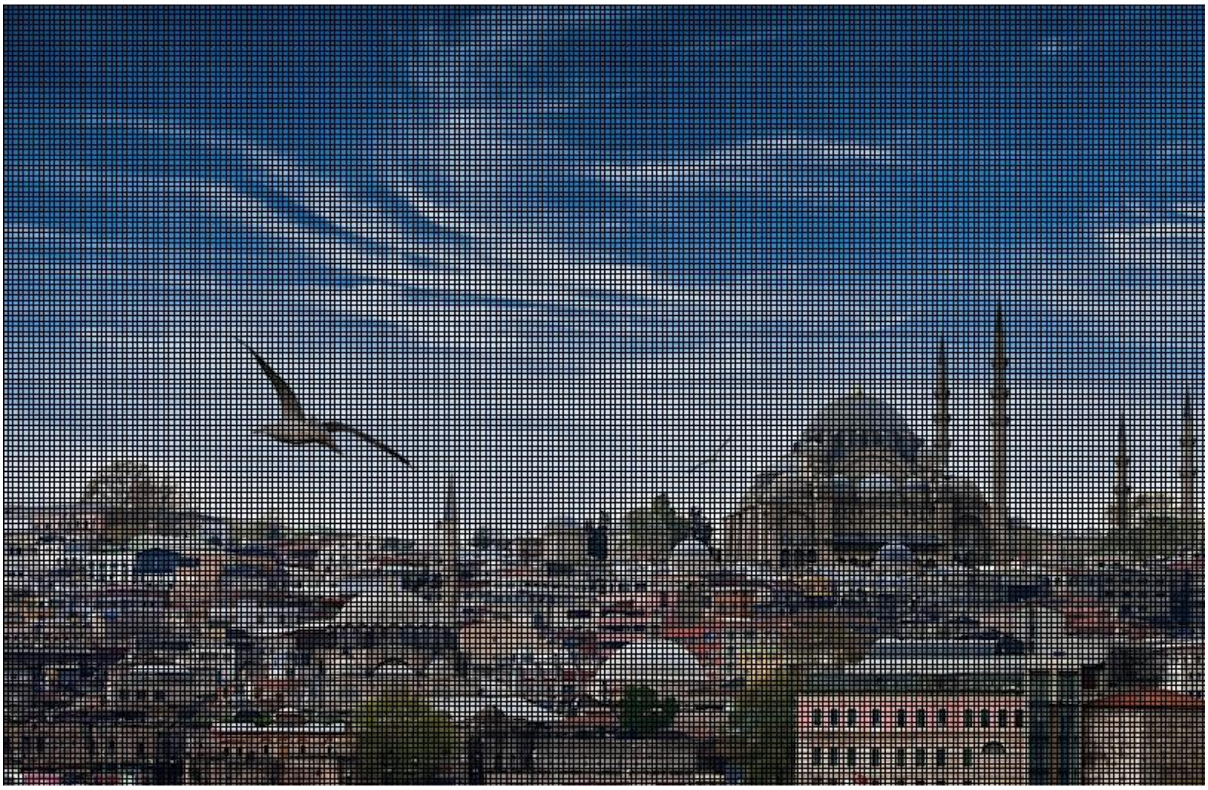
Forward Operation get original image y and x values and multiply with d and a value. After gain these result, I will get newY and newX positions. Forward mapping result have a lot blocks inside the image because of some of (y,x) coordinate lost when multiply each pixel coordinate.

Here are some explanation about how to find new position of each pixeal as a psudeu code;

newImageY = a * y;

newImageX = d * x;

Figure 5: Zoom operation with Forward Mapping



2.2. Backward Mapping Method:

When we use backward mapping, method don’t create squares as forward mapping. Because of we don’t use directly multiply source coordinate with our affine matrix. We want to investigate each of pixel of new image.

Therefore we use same multiply logic but with different approach. Here about calculation;

If (newImageY = a * y) => (y = newImageY / a). With this approach, we search for each pixel of new image. Of course some result don’t come as a integer, therefore we round it.

Figure 6: Zoom operation with Backward Mapping

2.3. Backward Mapping with Bilinear Interpolation:

Actually, we get it better result with backward mapping, instead of using forward mapping. But image have some bluer and in detail some pixel escape. For solving this idea, we use bilinear interpolation. This metod calculate adjacent of the coordinate and we get better result. To see difference between without using bilinear interpolation, you should zoom the image. You will find smooth pixels.

I also share how do you do calculate at the top of the page with an image.

Figure 7: Zoom operation with Backward Mapping with Bilinear Interpolation

3. Scale Operation

Our second operation is scale operation. It is different from zoom operation because we should change new image size after scaling. Otherwise we lose some part of image. But of course our affine matrix same with zoom affine matrix.

Here are our affine matrix is same with zoom affine matrix;

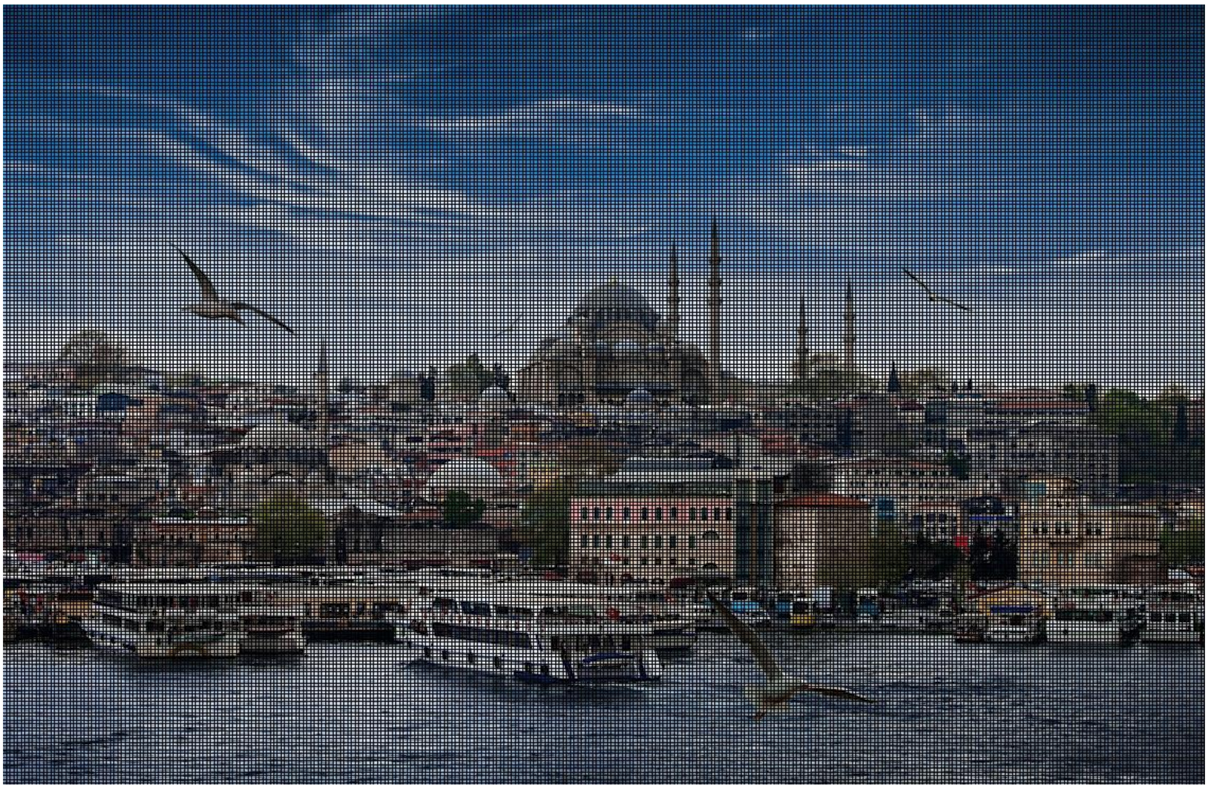
a = 1.4, b = 1, tx = 0;

c = 0, d = 1.4, ty = 0;

3.1. Forward Mapping Method:

We get full size of picture with little square blocks. Because same strategy applies to image and we lost some pixel coordinate of the new image size.

Figure 8: Scale Operation with forward mapping



3.2. Backward Mapping Method:

We get better result according to forward mapping and bluer look like less than also zoom image. Because we don't see details. We don't use interpolation and we look image far away. Therefore we cannot see approximated pixel filling.

Figure 9: Scale Operation with backwar mapping

3.3. Backward Mapping with Bilinear Interpolation Method:

When we use interpolation with backward Mapping method, we can see clear image. Also if we want to look detail of the new image, we will see a smooth slice of pixels instead of copying before coordinate.

Figure 10: Scale Operation with backward mapping with bilinear interpolation

4. **Horizontal Shear Operation**

Our third operation is share. Horizontal share mean is iterate pixels through x position.

For share operation we are manipulate b and d values of the affine matrix. When increase we b value of the affine matrix our image buttom goes wright. Here are our affine matrix;

a = 1, b = 1.4, tx = 1;

c = 0, d = 1, ty = 1;

4.1. Forward Mapping Method

Unlike of the other operations, when we do shear operation our new image doesn’t create a pattern inside the image. Because we are already transform the image, so maybe still we have black holes, but there are no any patter. Our image just look like broken.

Figure 11: Horizantal Share Operation with forward mapping

4.2. Backward Mapping and Backward Mapping with Bilinear Interpolation

Actually, both result look like similar like image that used forward mapping. We talked about how to calculate new position of new image and other differences. So I just visulise image.

Figure 12: Horizantal Share Operation with backward mapping

Figure 14: Horizantal Share Operation with backward mapping and bilinear interpolation

5. Rotation Operation

Our last operation is rotation. Rotation is more complex than other operations. Because actually, we dont rotate the image, we just shear image horizantally and vertically. For doing proper rotation we should shear both side simultaneusly. In addition for doing this operation image will be increase. For solving scale problem we should decrease size of image simultaneusly, too. For doing that we use wave transform of sin and cos.

5.1. Images of Rotation Operations

There are images of forward mapping, backward mapping, and backward mapping with bilinear interpolation. I just share the images because of their calculations are same. Here are rotation of affine matrix:

```
int centerX0 = width / 2;

int centerY0 = height / 2;

a = cos(radyan), b = -sin(radyan);

c = sin(radyan), d = cos(radyan);

tx = (newWidth / 2.0) - (centerX0 * a + centerY0 * b);

ty = (newHeight / 2.0) - (centerX0 * c + centerY0 * d);
```

Figure 15: Rotation Operation with forward mapping

Figure 16: Rotation Operation with backward mapping

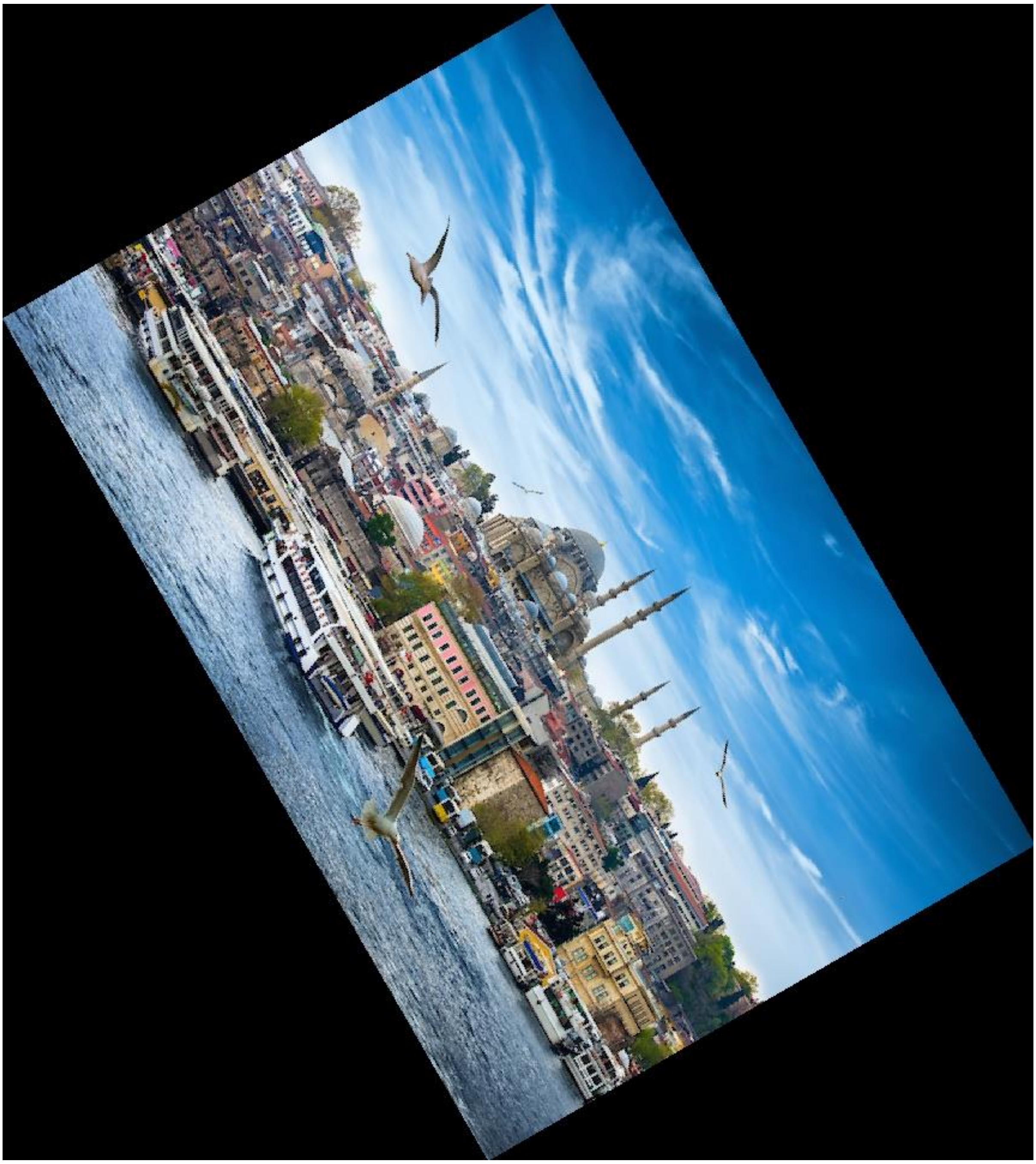
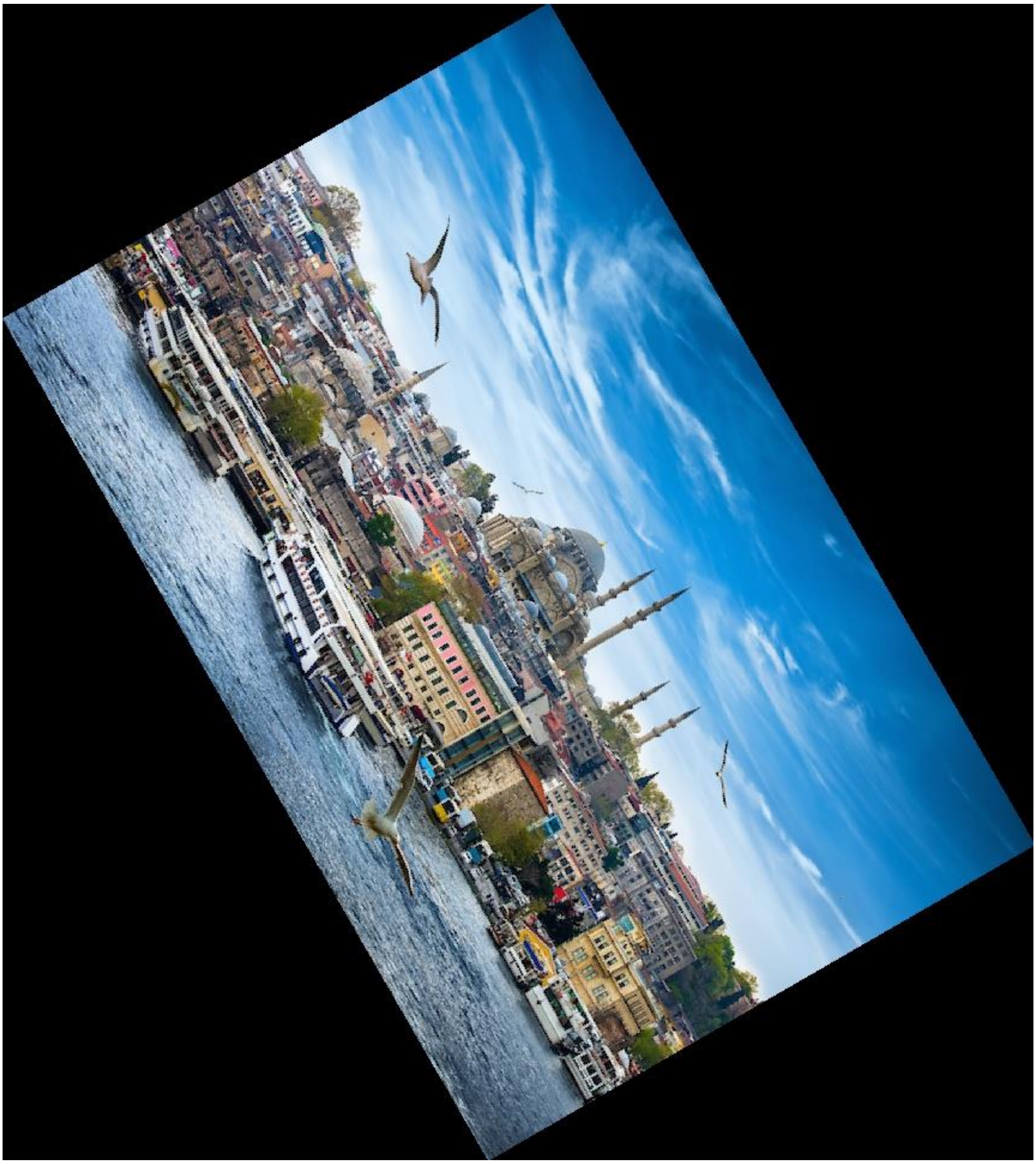


Figure 17: Rotation Operation with backward mapping and bilinear interpolation



6. Conclusion

To sum up, we manipulate the image with 4 different methods these scale, zoom (type of scale), shear, and rotation. For doing these operations we use 3 methods.

One of them is “Forward Mapping”; this method simply find new coordinate of image multiplying coordinates with affine matrix. Therefore, image have some hole.

Secondly we use “Backward Mapping”; this method better than first one. Because it is reach own pixel density from source image. Therefore, there are not any hole.

Lastly, we add bilinear interpolation method to “Backward Mapping”. This method increases the image quality. Because, normally we just getting nearest coordinate of the image. But with bilinear interpolation we calculate adjancent of the coordinate.

When we compare 3 result, we clearly see difference between forward and backward mappings. But if we don’t look at details of image, interpolation it comes a lot cost.

7. Appendix

7.1. Forward Mapping Function:

```
void forwardMapping(const Mat& image) {
    int width = image.cols;
    int height = image.rows;
    // affine matrix values
    double a, b, tx;
    double c, d, ty;

#if 0 // zoom image 1.4x

    a = 1.4, b = 0, tx = 1;
    c = 0, d = 1.4, ty = 1;
    Mat zoomAffineMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    // new Image
    Mat zoomImage(height, width, CV_8UC3, cv::Scalar(0, 0, 0));

    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            int newX = static_cast<int>(x * zoomAffineMax.at<double>(0, 0));
            int newY = static_cast<int>(y * zoomAffineMax.at<double>(1, 1));
            if (newX > 0 && newX < width && newY > 0 && newY < height) {
                zoomImage.at<cv::Vec3b>(newY, newX) = image.at<cv::Vec3b>(y, x); // Forward mapping
            }
        }
    }

    imshow("zoomFoward", zoomImage);
    if (!imwrite("zoomFoward.jpg", zoomImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif

#if 0 // horizotanl shear image 1.4x

    a = 1, b = 1.4, tx = 1;
    c = 0, d = 1, ty = 1;
    int newWidth = width + (height * b);
    Mat horizantalShearAffMax= (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    // new Image
    Mat shearImage(height, newWidth, CV_8UC3, cv::Scalar(0, 0, 0));

    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            int newX =  static_cast<int>(x + (y * horizantalShearAffMax.at<double>(0, 1)));
            int newY = static_cast<int>(y);
            if (newX >= 0 && newX < newWidth && newY >= 0 && newY < height) {
                // Forward mapping
                shearImage.at<cv::Vec3b>(newY, newX) = image.at<cv::Vec3b>(y, x);
            }
        }
    }

    if (!imwrite("HorizantalShearForward.jpg", shearImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif

#if 0 // scale image 1.4x

    a = 1.4, b = 1, tx = 0;
```

```cpp
    c = 0, d = 1.4, ty = 0;
    int newHeight = a * height;
    int newWidth = d * width;
    Mat scaleAffMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    // new Image
    Mat scaleImage(newHeight, newWidth, CV_8UC3, cv::Scalar(0, 0, 0));

    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            int newX = static_cast<int>(x * scaleAffMax.at<double>(0, 0));
            int newY = static_cast<int>(y * scaleAffMax.at<double>(1, 1));
            if (newX >= 0 && newX < newWidth && newY >= 0 && newY < newHeight) {
                // Forward mapping
                scaleImage.at<cv::Vec3b>(newY, newX) = image.at<cv::Vec3b>(y, x);
            }
        }
    }

    if (!imwrite("scaledIimageForward.jpg", scaleImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif

#if 1 // rotate image 60 degree

    int centerX0 = width/ 2;
    int centerY0 = height / 2;
    int degre = 60;
    double radyan = degre * 3.14159 / 180;

    a = cos(radyan), b = -sin(radyan);
    c = sin(radyan), d = cos(radyan);

    int newHeight = static_cast<int>(height * fabs(a)) + static_cast<int>(width * fabs(c));
    int newWidth = static_cast<int>(width * fabs(a)) + static_cast<int>(height * fabs(c));

    tx = (newWidth / 2.0) - (centerX0 * a + centerY0 * b);
    ty = (newHeight / 2.0) - (centerX0 * c + centerY0 * d);

    Mat scaleAffMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    // new Image
    Mat rotateImage(newHeight, newWidth, CV_8UC3, cv::Scalar(0, 0, 0));

    // rotate
    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            int newX = static_cast<int>((x * scaleAffMax.at<double>(0, 0)) + (y * scaleAffMax.at<double>(0, 1)) + scaleAffMax.at<double>(0, 2));
            int newY = static_cast<int>((x * scaleAffMax.at<double>(1, 0)) + (y * scaleAffMax.at<double>(1, 1)) + scaleAffMax.at<double>(1, 2));
            if (newX >= 0 && newX < newWidth +200 && newY >= 0 && newY < newHeight) {
                // Forward mapping
                rotateImage.at<Vec3b>(newY, newX) = image.at<Vec3b>(y, x);
            }
        }
    }


    if (!imwrite("rotated_imageForward.jpg", rotateImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif
    return;
```

```
}
```

7.2. Backward Mapping Function:

```
void backwardMapping(const Mat& image) {
    int width = image.cols;
    int height = image.rows;
    // affine matrix values
    double a, b, tx;
    double c, d, ty;

#if 0 // zoom image 1.4x

    a = 1.4, b = 0, tx = 1;
    c = 0, d = 1.4, ty = 1;
    Mat zoomAffineMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    // new Image wiht same size
    Mat zoomImage(height , width, CV_8UC3, cv::Scalar(0, 0, 0));

    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            int newX = static_cast<int>(x / zoomAffineMax.at<double>(0, 0));
            int newY = static_cast<int>(y / zoomAffineMax.at<double>(1, 1));
            if (newX >= 0 && newX < width && newY >= 0 && newY < height) {
                // Backward mapping
                zoomImage.at<cv::Vec3b>(y, x) = image.at<cv::Vec3b>(newY, newX);
            }
        }
    }

    if (!imwrite("zoomBackward.jpg", zoomImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif

#if 0 // horizotanl shear image 1.4x

    a = 1, b = 1.4, tx = 1;
    c = 0, d = 1, ty = 1;
    int newWidth = width + (height * b);
    Mat horizantalShearAffMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    // new Image
    Mat shearImage(height, newWidth, CV_8UC3, cv::Scalar(0, 0, 0));

    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < newWidth; ++x) {
            int imageX = static_cast<int>(x - (y * horizantalShearAffMax.at<double>(0, 1)));
            int imageY = static_cast<int>(y);
            if (imageX >= 0 && imageX < width && imageY >= 0 && imageY < height) {
                shearImage.at<cv::Vec3b>(y, x) = image.at<cv::Vec3b>(imageY, imageX); // Backward mapping
            }
        }
    }

    if (!imwrite("HorizantalShearBackward.jpg", shearImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif
```

```
#if 0 // scale image 1.4x

    a = 1.4, b = 1, tx = 0;
    c = 0, d = 1.4, ty = 0;
    int newHeight = a * height;
    int newWidth = d * width;
    Mat scaleAffMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    Mat scaleImage(newHeight, newWidth, CV_8UC3, cv::Scalar(0, 0, 0)); // new Image

    for (int y = 0; y < newHeight; ++y) {
        for (int x = 0; x < newWidth; ++x) {
            int imageX = static_cast<int>(x / scaleAffMax.at<double>(0, 0));
            int imageY = static_cast<int>(y / scaleAffMax.at<double>(1, 1));
            if (imageX >= 0 && imageX < width && imageY >= 0 && imageY < height) {
                scaleImage.at<cv::Vec3b>(y, x) = image.at<cv::Vec3b>(imageY, imageX); // Backward mapping
            }
        }
    }
    if (!imwrite("scaledImageBackward.jpg", scaleImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif

#if 0 // rotate image 60 degree

    int centerX0 = width / 2;
    int centerY0 = height / 2;
    int degre = 60;
    double radyan = degre * 3.14159 / 180;

    a = cos(radyan), b = -sin(radyan);
    c = sin(radyan), d = cos(radyan);

    int newHeight = static_cast<int>(height * fabs(a)) + static_cast<int>(width * fabs(c));
    int newWidth = static_cast<int>(width * fabs(a)) + static_cast<int>(height * fabs(c));

    tx = (newWidth / 2.0) - (centerX0 * a + centerY0 * b);
    ty = (newHeight / 2.0) - (centerX0 * c + centerY0 * d);

    Mat scaleAffMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    Mat rotateImage(newHeight, newWidth, CV_8UC3, cv::Scalar(0, 0, 0)); // new Image

    if (inverseOfAffineMax(scaleAffMax) == false) {
            std::cerr << "Affine matrix didnt turn inverse itself" << std::endl;
            return ;
        }

    // rotate
    for (int y = 0; y < newHeight; ++y) {
        for (int x = 0; x < newWidth; ++x) {
            int imageX = static_cast<int>((x * scaleAffMax.at<double>(0, 0)) + (y * scaleAffMax.at<double>(0, 1)) + scaleAffMax.at<double>(0, 2));
            int imageY = static_cast<int>((x * scaleAffMax.at<double>(1, 0)) + (y * scaleAffMax.at<double>(1, 1)) + scaleAffMax.at<double>(1, 2));
            //cout << "image x " << imageX;
            if (imageX >= 0 && imageX < width && imageY >= 0 && imageY < height) {
                rotateImage.at<cv::Vec3b>(y, x) = image.at<cv::Vec3b>(imageY, imageX); // Backward mapping
            }
        }
    }
```

```


    if (!imwrite("rotated_imageBackward.jpg", rotateImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif
    return;
}
```

7.3. Backward Mapping with Bilinear Interpolation

```
void backwardMappingWithInterpolation(const Mat& image) {
    int width = image.cols;
    int height = image.rows;
    // affine matrix values
    double a, b, tx;
    double c, d, ty;

#if 0// zoom image 1.4x

    a = 1.4, b = 0, tx = 1;
    c = 0, d = 1.4, ty = 1;
    Mat zoomAffineMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);

    Mat zoomImage(height, width, CV_8UC3, Scalar(0, 0, 0)); // new Image wiht new size

    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            double newX = x / (zoomAffineMax.at<double>(0, 0));
            double newY = y / (zoomAffineMax.at<double>(1, 1));
            if (newX >= 0 && newX < width && newY >= 0 && newY < height) {
                zoomImage.at<Vec3b>(y, x) =  calculateBilinearValue(image, newY, newX); // Backward mapping with interpolation
            }
        }
    }

    //imshow("zoomed image", zoomImage);
    if (!imwrite("zoomBackwardInterpol.jpg", zoomImage)) {
        std::cerr << "write error!" << std::endl;
    }

#endif

#if 0 // horizotanl shear image 1.4x

    a = 1, b = 1.4, tx = 1;
    c = 0, d = 1, ty = 1;
    int newWidth = width + (height * b);
    Mat horizantalShearAffMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    Mat shearImage(height, newWidth, CV_8UC3, cv::Scalar(0, 0, 0)); // new Image

    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < newWidth; ++x) {
            int newX = static_cast<int>(x - (y * horizantalShearAffMax.at<double>(0, 1)));
            int newY = static_cast<int>(y);
            if (newX >= 0 && newX < width && newY >= 0 && newY < height) {
                shearImage.at<cv::Vec3b>(y, x) = calculateBilinearValue(image, newY, newX);  // Backward mapping with interpolation
            }
        }
    }
```

```

    if (!imwrite("HorizantalShearInterpolation.jpg", shearImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif

#if 0 // scale image 1.4x

    a = 1.4, b = 1, tx = 0;
    c = 0, d = 1.4, ty = 0;
    int newHeight = a * height;
    int newWidth = d * width;
    Mat scaleAffMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    Mat scaleImage(newHeight, newWidth, CV_8UC3, cv::Scalar(0, 0, 0)); // new Image

    for (int y = 0; y < newHeight; ++y) {
        for (int x = 0; x < newWidth; ++x) {
            int newX = static_cast<int>(x / scaleAffMax.at<double>(0, 0));
            int newY = static_cast<int>(y / scaleAffMax.at<double>(1, 1));
            if (newX >= 0 && newX < width && newY >= 0 && newY < height) {
                scaleImage.at<cv::Vec3b>(y, x) = calculateBilinearValue(image, newY, newX); // Backward mapping
            }
        }
    }

    if (!imwrite("scaledIimageBackwardInterpolation.jpg", scaleImage)) {
        std::cerr << "write error!" << std::endl;
    }

#endif

#if 0 // rotate image 60 degree

    int centerX0 = width / 2;
    int centerY0 = height / 2;
    int degre = 60;
    double radyan = degre * 3.14159 / 180;

    a = cos(radyan), b = -sin(radyan);
    c = sin(radyan), d = cos(radyan);

    int newHeight = static_cast<int>(height * fabs(a)) + static_cast<int>(width * fabs(c));
    int newWidth = static_cast<int>(width * fabs(a)) + static_cast<int>(height * fabs(c));

    tx = (newWidth / 2.0) - (centerX0 * a + centerY0 * b);
    ty = (newHeight / 2.0) - (centerX0 * c + centerY0 * d);

    Mat scaleAffMax = (Mat_<double>(2, 3) << a, b, tx, c, d, ty);
    Mat rotateImage(newHeight, newWidth, CV_8UC3, cv::Scalar(0, 0, 0)); // new Image

    // before inverse

    if (inverseOfAffineMax(scaleAffMax) == false) {
        std::cerr << "Affine matrix didnt turn inverse itself" << std::endl;
        return;
    }

    // rotate
    for (int y = 0; y < newHeight; ++y) {
        for (int x = 0; x < newWidth; ++x) {
            int newX = static_cast<int>((x * scaleAffMax.at<double>(0, 0)) + (y * scaleAffMax.at<double>(0, 1)) +
scaleAffMax.at<double>(0, 2));
```

```
            int newY = static_cast<int>((x * scaleAffMax.at<double>(1, 0)) + (y * scaleAffMax.at<double>(1, 1)) + scaleAffMax.at<double>(1, 2));

            if (newX >= 0 && newX < width && newY >= 0 && newY < height) {
                rotateImage.at<cv::Vec3b>(y, x) = calculateBilinearValue(image, newY, newX); // Backward mapping
            }
        }
    }


    if (!imwrite("rotated_imageBackwardInterpol.jpg", rotateImage)) {
        std::cerr << "write error!" << std::endl;
    }
#endif
    return;
}
```

7.4. Bilinear Interpolation Function

```
Vec3b calculateBilinearValue(const Mat& image, const double &y, const double& x) {
    int x1 = static_cast<int>(x); // min round x
    int y1 = static_cast<int>(y); // min round y
    int x2 = min(x1 + 1, image.cols - 1); //  x1 + 1 is top border
    int y2 = min(y1 + 1, image.rows - 1); //  y1 + 1 is top border


    Vec3b Q11 = image.at<Vec3b>(y1, x1);
    Vec3b Q21 = image.at<Vec3b>(y1, x2);
    Vec3b Q12 = image.at<Vec3b>(y2, x1);
    Vec3b Q22 = image.at<Vec3b>(y2, x2);


    double div = ((x2 - x1) * (y2 - y1));

    if (div <= 0) {
        return Q11;
    }

    Vec3b resultDens;
    for (int i = 0; i < 3; ++i) {
        resultDens[i] = static_cast<uchar>(
            (Q11[i] * (x2 - x) * (y2 - y) / div) +
            (Q21[i] * (x - x1) * (y2 - y) / div) +
            (Q12[i] * (x2 - x) * (y - y1) / div) +
            (Q22[i] * (x - x1) * (y - y1) / div)
            );
    }
    return resultDens;
}
```

7.5. Inverse Affine Matrix Function:

```
bool inverseOfAffineMax(Mat& affineMax) {
    double a = affineMax.at<double>(0, 0);
```

```
    double b = affineMax.at<double>(0, 1);
    double tx = affineMax.at<double>(0, 2);
    double c = affineMax.at<double>(1, 0);
    double d = affineMax.at<double>(1, 1);
    double ty = affineMax.at<double>(1, 2);

    // calcukale determinant
    double det = a * d - b * c;
    if (det == 0) {
            return false;  // not found deteerminant
    }


    Mat inverseMat = (Mat_<double>(2, 3) <<
      d * (1 / det), -b * (1 / det), (b * ty - d * tx) * (1/det),
      -c * (1 / det), a * (1 / det), (c * tx - a * ty) * (1 / det)
      );

    inverseMat.copyTo(affineMax);
    return true;
}
```

8. Sources

1. https://chatgpt.com/
2. https://www.algorithm-archive.org/contents/bitlogic/bitlogic.html
3. Gonzalez, Rafael C. Digital Image 3.ed., Person, Processing-Prentice Hall, 2008