DIGITAL IMAGE PROCESSING

HOMEWORK 1 REPORT

ENES AYSU 1901042671

DIFFERENCE BETWEEN FORWARD AND BACKWARD MAPPING

Forward mapping and backward mapping are terms often used in the context of computer graphics, computer vision, and image processing.

Forward mapping involves transforming objects or points from their original coordinate space to a new coordinate space. It's like taking an object and figuring out where it will end up after a series of transformations. Think of it as starting with the initial state and applying transformations to get to the final state.

On the other hand, backward mapping is the opposite. It involves taking points or objects from the destination coordinate space and figuring out where they came from in the original coordinate space.

This report examines the difference between the concepts we mentioned on an image. Some affine transforms are applied to the image such as rotate, scale, shear and zoom.

FORWARD AND BACKWARD MAPPING COMPARISON

Forward Mapping:

Advantages:

1. Efficiency: Forward mapping can be more computationally efficient, especially when dealing with large datasets. You apply transformations once and get the result.
2. Predictability: It's generally easier to predict and control the outcome of forward mapping since you are moving from a known initial state to a known final state.

Disadvantages:

1. Loss of Information: Depending on the complexity of the transformations, you might lose some information during the process, especially if you're compressing or simplifying the data.
2. Not Always Reversible: Forward mapping might not always be reversible. Once you've applied transformations, it can be challenging to reconstruct the original state.

Backward Mapping:

Advantages:

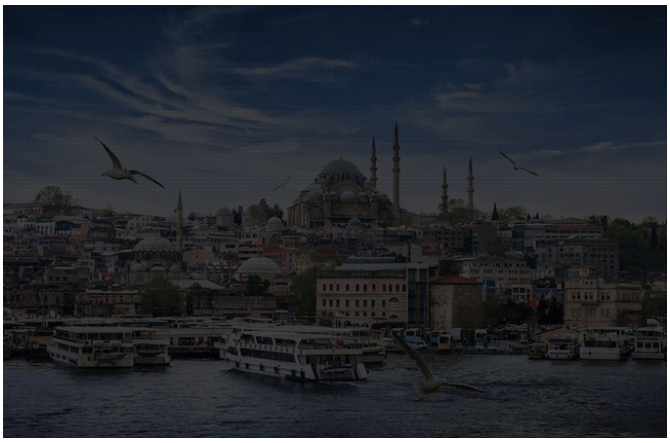
1. Reversibility: One of the main advantages is that backward mapping is inherently reversible. You can go back to the original state from the transformed state.
2. Preservation of Information: Since you're working from the transformed state back to the original, there's potentially less loss of information.

Disadvantages:

1. Computational Complexity: Backward mapping can be more computationally intensive, especially when dealing with complex transformations. It involves solving equations to find the inverse transformation.
2. Ambiguity: In some cases, there might be multiple possible original states that could lead to the same transformed state, leading to ambiguity.

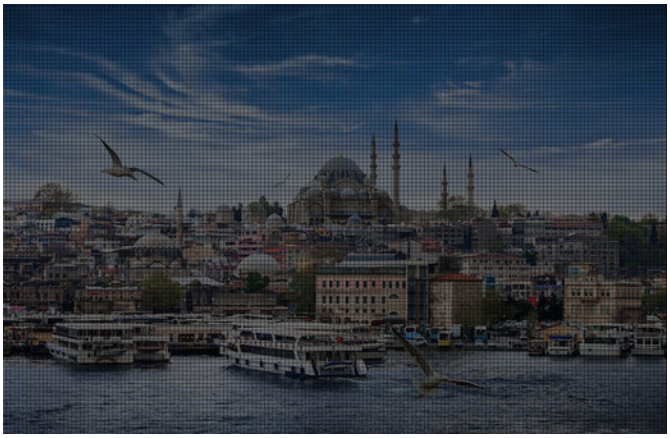
AFFINE TRANSFORMS WITH FORWARD MAPPING

rotated (30 degree)



scaled (x2)

sheared (0.5 on x-dimension)



zoomed (1.6 times)

AFFINE TRANSFORMS WITH BACKWARD MAPPING

rotated (30 degree)

scaled (x2)

sheared (0.5 on x-dimension)

zoomed (1.6 times)

AFFINE TRANSFORMS WITH BACKWARD MAPPING BY USING LINEAR INTERPOLATION

rotated (30 degree)

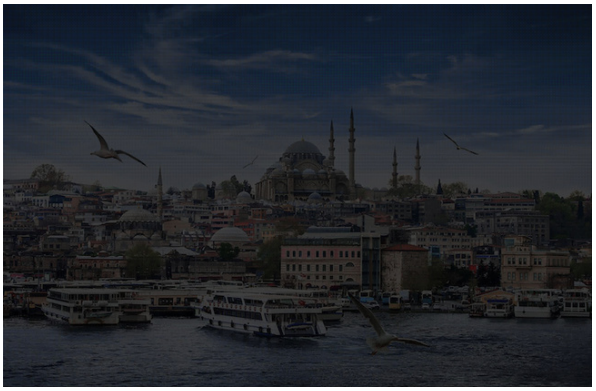
scaled (x2)

sheared (0.5 on x-dimension)

zoomed (1.6 times)

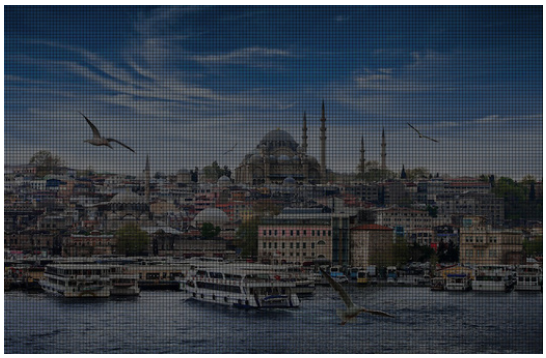
In the affine transformations applied with forward mapping, it was found that some pixels overlapped in the images that were exchanged during the operation, and therefore some pixels looked black in the newly formed image.

For example rotation ,scaling and zooming operations:



scaled (x2)

rotated (30 degree)



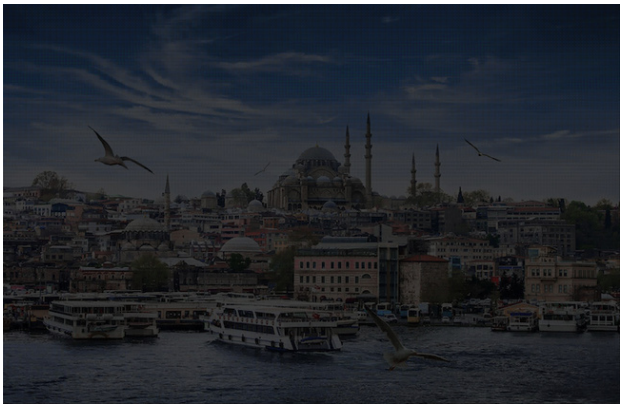
zoomed (1.6 times)

This is because some pixels do not have an integer value due to forward mapping, and therefore 1 or more than 1 pixels correspond to the same position in the new image, or in cases where the size of the image increases, such as scale operation, the number of pixels does not match the dimensions of the new image. In such cases, the visual clarity decreases and noise occurs.

In other forward mapping operations, alterations in image quality and loss of details have arisen due to variations in the dimensions of the image.

sheared (0.5 on x-dimension)

In backward mapping, according to forward mapping, there were no visual distortions, but the quality of detail in the original visual could not be maintained.



scaled with forward mapping (x2)

scaled with backward mapping(x2)

As can be seen in the images, there are missing pixels in the image transformed with forward mapping, while this situation was not observed in the backward mapping.

Compared to the original visual, it is observed that the details cannot be preserved.

original image

scaled image with backward mapping

As can be seen in the image, examining by the structure of the seagull and the dome, the smooth transition in the original image cannot be observed in a scaled form.

Finally, if the backward mapping method applied with bilinear interpolation is examined, it has been observed that images scaled back with affine transform with this usage maintain the quality of their original state better.

scaled (x2) with backward mapping (not interpolated)

scaled (x2) with backward mapping (interpolated)

As can be seen in the images, it is observed that the interpolated backward mapping applied visual has smoother transitions.

Because backward mapping with bilinear interpolation involves calculating pixel values by considering the weighted average of surrounding pixels, resulting in smoother transitions and better image quality.

Without bilinear interpolation, pixel values are determined without this weighted averaging, leading to a less refined representation with potential visual artifacts and a loss of image quality.

CODE IMPLEMENTATIONS

1) AFFINE TRANSFORMS WITH FORWARD MAPPING

1.1) ROTATING

```python
from PIL import Image
import numpy as np

image = Image.open("istanbul.jpg")

width, height = image.size

angle_degrees = 30
radian = np.radians(angle_degrees)

new_width = int(abs(np.cos(radian) * width) + abs(np.sin(radian) * height))
new_height = int(abs(np.sin(radian) * width) + abs(np.cos(radian) * height))

rotated_image = Image.new("RGB", (new_width, new_height))

center_x = new_width // 2
center_y = new_height // 2

rotated_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(height):
    for x in range(width):
        x_rel = x - width // 2
        y_rel = y - height // 2

        x_new = int(x_rel * np.cos(-radian) - y_rel * np.sin(-radian)) + center_x
        y_new = int(x_rel * np.sin(-radian) + y_rel * np.cos(-radian)) + center_y

        if 0 <= x_new < new_width and 0 <= y_new < new_height:
            rotated_data[y_new, x_new, :] = np.array(image.getpixel((x, y)))

rotated_image = Image.fromarray(rotated_data)
rotated_image.save("istanbul_rotated.jpg")
rotated_image.show()
```

1.2) SCALING

```python
from PIL import Image
import numpy as np

image = Image.open("istanbul.jpg")
width, height = image.size

scale_factor_x = 2
scale_factor_y = 2

new_width = int(width * scale_factor_x)
new_height = int(height * scale_factor_y)

scaled_image = Image.new("RGB", (new_width, new_height))

scaled_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(height):
    for x in range(width):
        x_new = int(x * scale_factor_x)
        y_new = int(y * scale_factor_y)

        if 0 <= x_new < new_width and 0 <= y_new < new_height:
            scaled_data[y_new, x_new, :] = np.array(image.getpixel((x, y)))

scaled_image = Image.fromarray(scaled_data)
scaled_image.save("istanbul_scaled.jpg")
scaled_image.show()
```

1.3) SHEARING

```python
from PIL import Image
import numpy as np

image = Image.open("istanbul.jpg")
width, height = image.size

sx = 0.5
sy = 0.0

new_width = width + int(height * sx)
new_height = height + int(width * sy)

sheared_image = Image.new("RGB", (new_width, new_height))

sheared_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(height):
    for x in range(width):
        x_new = x + int(y * sx)
        y_new = y + int(x * sy)

        if 0 <= x_new < new_width and 0 <= y_new < new_height:
            sheared_data[y_new, x_new, :] = np.array(image.getpixel((x, y)))

sheared_image = Image.fromarray(sheared_data)
sheared_image.save("istanbul_sheared.jpg")
sheared_image.show()
```

1.4) ZOOMING

```python
from PIL import Image
import numpy as np

image = Image.open("istanbul.jpg")
width, height = image.size

zoom_factor = 1.6

new_width = int(width * zoom_factor)
new_height = int(height * zoom_factor)

scaled_image = Image.new("RGB", (new_width, new_height))

scaled_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(height):
    for x in range(width):
        x_new = int(x * zoom_factor)
        y_new = int(y * zoom_factor)

        if 0 <= x_new < new_width and 0 <= y_new < new_height:
            scaled_data[y_new, x_new, :] = np.array(image.getpixel((x, y)))

scaled_image = Image.fromarray(scaled_data)
scaled_image.save("istanbul_zoomed.jpg")
scaled_image.show()

```

2) AFFINE TRANSFORMS WITH BACKWARD MAPPING

2.1) ROTATING

```python
from PIL import Image
import numpy as np

image = Image.open("istanbul.jpg")

width, height = image.size

angle_degrees = 30
radian = np.radians(angle_degrees)

new_width = int(abs(np.cos(radian) * width) + abs(np.sin(radian) * height))
new_height = int(abs(np.sin(radian) * width) + abs(np.cos(radian) * height))

rotated_image = Image.new("RGB", (new_width, new_height))

center_x = new_width // 2
center_y = new_height // 2

rotated_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(new_height):
    for x in range(new_width):
        x_rel = x - center_x
        y_rel = y - center_y

        x_old = int(x_rel * np.cos(radian) - y_rel * np.sin(radian)) + width // 2
        y_old = int(x_rel * np.sin(radian) + y_rel * np.cos(radian)) + height // 2

        if 0 <= x_old < width and 0 <= y_old < height:
            rotated_data[y, x, :] = np.array(image.getpixel((x_old, y_old)))

rotated_image = Image.fromarray(rotated_data)
rotated_image.save("istanbul_rotated_2.jpg")
rotated_image.show()

```

2.2) SCALING

```python
from PIL import Image
import numpy as np

image = Image.open("istanbul.jpg")
width, height = image.size

scale_factor_x = 2
scale_factor_y = 2

new_width = int(width * scale_factor_x)
new_height = int(height * scale_factor_y)

scaled_image = Image.new("RGB", (new_width, new_height))

scaled_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(new_height):
    for x in range(new_width):
        x_old = int(x / scale_factor_x)
        y_old = int(y / scale_factor_y)

        if 0 <= x_old < width and 0 <= y_old < height:
            scaled_data[y, x, :] = np.array(image.getpixel((x_old, y_old)))

scaled_image = Image.fromarray(scaled_data)
scaled_image.save("istanbul_scaled_2.jpg")
scaled_image.show()

```

2.3) SHEARING

```python
from PIL import Image
import numpy as np

image = Image.open("istanbul.jpg")
width, height = image.size

sx = 0.5
sy = 0.0

new_width = width + int(height * sx)
new_height = height + int(width * sy)

sheared_image = Image.new("RGB", (new_width, new_height))

sheared_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(new_height):
    for x in range(new_width):
        x_old = x - int(y * sx)
        y_old = y - int(x * sy)

        if 0 <= x_old < width and 0 <= y_old < height:
            sheared_data[y, x, :] = np.array(image.getpixel((x_old, y_old)))

sheared_image = Image.fromarray(sheared_data)
sheared_image.save("istanbul_sheared_2.jpg")
sheared_image.show()

```

2.4) ZOOMING

```python
from PIL import Image
import numpy as np

image = Image.open("istanbul.jpg")
width, height = image.size

zoom_factor = 1.6

new_width = int(width * zoom_factor)
new_height = int(height * zoom_factor)

scaled_image = Image.new("RGB", (new_width, new_height))

scaled_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(new_height):
    for x in range(new_width):
        x_old = int(x / zoom_factor)
        y_old = int(y / zoom_factor)

        if 0 <= x_old < width and 0 <= y_old < height:
            scaled_data[y, x, :] = np.array(image.getpixel((x_old, y_old)))

scaled_image = Image.fromarray(scaled_data)
scaled_image.save("istanbul_zoomed_2.jpg")
scaled_image.show()

```

3) AFFINE TRANSFORMS WITH BACKWARD MAPPING BY USING BILINEAR INTERPOLATION

3.1) ROTATING

```python
from PIL import Image
import numpy as np

def bilinear_interpolation(image, x, y):
    x0, y0 = int(np.floor(x)), int(np.floor(y))
    x1, y1 = x0 + 1, y0 + 1

    if x0 < 0 or y0 < 0 or x1 >= image.width or y1 >= image.height:
        return np.zeros(3, dtype=np.uint8)

    q11 = np.array(image.getpixel((x0, y0)))
    q21 = np.array(image.getpixel((x1, y0)))
    q12 = np.array(image.getpixel((x0, y1)))
    q22 = np.array(image.getpixel((x1, y1)))

    f_x1 = x - x0
    f_x0 = 1 - f_x1
    f_y1 = y - y0
    f_y0 = 1 - f_y1

    pixel_value = f_x0 * (f_y0 * q11 + f_y1 * q12) + f_x1 * (f_y0 * q21 + f_y1 * q22)

    return np.clip(pixel_value, 0, 255).astype(np.uint8)

image = Image.open("istanbul.jpg")

width, height = image.size

angle_degrees = 30
radian = np.radians(angle_degrees)

new_width = int(abs(np.cos(radian) * width) + abs(np.sin(radian) * height))
new_height = int(abs(np.sin(radian) * width) + abs(np.cos(radian) * height))

rotated_image = Image.new("RGB", (new_width, new_height))

center_x = new_width // 2
center_y = new_height // 2

rotated_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(new_height):
    for x in range(new_width):
        x_rel = x - center_x
        y_rel = y - center_y

        x_old = int(x_rel * np.cos(radian) - y_rel * np.sin(radian)) + width // 2
        y_old = int(x_rel * np.sin(radian) + y_rel * np.cos(radian)) + height // 2

        rotated_data[y, x, :] = bilinear_interpolation(image, x_old, y_old)

rotated_image = Image.fromarray(rotated_data)
rotated_image.save("istanbul_rotated_3.jpg")
rotated_image.show()

```

3.2) SCALING

```python
from PIL import Image
import numpy as np

def bilinear_interpolation(image, x, y):
    x0, y0 = int(np.floor(x)), int(np.floor(y))
    x1, y1 = x0 + 1, y0 + 1

    if x0 < 0 or y0 < 0 or x1 >= image.width or y1 >= image.height:
        return np.zeros(3, dtype=np.uint8)

    q11 = np.array(image.getpixel((x0, y0)))
    q21 = np.array(image.getpixel((x1, y0)))
    q12 = np.array(image.getpixel((x0, y1)))
    q22 = np.array(image.getpixel((x1, y1)))

    f_x1 = x - x0
    f_x0 = 1 - f_x1
    f_y1 = y - y0
    f_y0 = 1 - f_y1

    pixel_value = f_x0 * (f_y0 * q11 + f_y1 * q12) + f_x1 * (f_y0 * q21 + f_y1 * q22)

    return np.clip(pixel_value, 0, 255).astype(np.uint8)

image = Image.open("istanbul.jpg")
width, height = image.size

scale_factor_x = 2
scale_factor_y = 2

new_width = int(width * scale_factor_x)
new_height = int(height * scale_factor_y)

scaled_image = Image.new("RGB", (new_width, new_height))

scaled_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(new_height):
    for x in range(new_width):
        x_old = x / scale_factor_x
        y_old = y / scale_factor_y

        scaled_data[y, x, :] = bilinear_interpolation(image, x_old, y_old)

scaled_image = Image.fromarray(scaled_data)
scaled_image.save("istanbul_scaled_3.jpg")
scaled_image.show()

```

3.3) SHEARING

```python
from PIL import Image
import numpy as np

def bilinear_interpolation(image, x, y):
    x0, y0 = int(np.floor(x)), int(np.floor(y))
    x1, y1 = x0 + 1, y0 + 1

    if x0 < 0 or y0 < 0 or x1 >= image.width or y1 >= image.height:
        return np.zeros(3, dtype=np.uint8)

    q11 = np.array(image.getpixel((x0, y0)))
    q21 = np.array(image.getpixel((x1, y0)))
    q12 = np.array(image.getpixel((x0, y1)))
    q22 = np.array(image.getpixel((x1, y1)))

    f_x1 = x - x0
    f_x0 = 1 - f_x1
    f_y1 = y - y0
    f_y0 = 1 - f_y1

    pixel_value = f_x0 * (f_y0 * q11 + f_y1 * q12) + f_x1 * (f_y0 * q21 + f_y1 * q22)

    return np.clip(pixel_value, 0, 255).astype(np.uint8)

image = Image.open("istanbul.jpg")
width, height = image.size

sx = 0.5
sy = 0.0

new_width = width + int(height * sx)
new_height = height + int(width * sy)

sheared_image = Image.new("RGB", (new_width, new_height))

sheared_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(new_height):
    for x in range(new_width):
        x_old = x - int(y * sx)
        y_old = y - int(x * sy)

        sheared_data[y, x, :] = bilinear_interpolation(image, x_old, y_old)

sheared_image = Image.fromarray(sheared_data)
sheared_image.save("istanbul_sheared_3.jpg")
sheared_image.show()

```

3.4) ZOOMING

```python
from PIL import Image
import numpy as np

def bilinear_interpolation(image, x, y):
    x0, y0 = int(np.floor(x)), int(np.floor(y))
    x1, y1 = x0 + 1, y0 + 1

    if x0 < 0 or y0 < 0 or x1 >= image.width or y1 >= image.height:
        return np.zeros(3, dtype=np.uint8)

    q11 = np.array(image.getpixel((x0, y0)))
    q21 = np.array(image.getpixel((x1, y0)))
    q12 = np.array(image.getpixel((x0, y1)))
    q22 = np.array(image.getpixel((x1, y1)))

    f_x1 = x - x0
    f_x0 = 1 - f_x1
    f_y1 = y - y0
    f_y0 = 1 - f_y1

    pixel_value = f_x0 * (f_y0 * q11 + f_y1 * q12) + f_x1 * (f_y0 * q21 + f_y1 * q22)

    return np.clip(pixel_value, 0, 255).astype(np.uint8)

image = Image.open("istanbul.jpg")
width, height = image.size

zoom_factor = 1.6

new_width = int(width * zoom_factor)
new_height = int(height * zoom_factor)

scaled_image = Image.new("RGB", (new_width, new_height))

scaled_data = np.zeros((new_height, new_width, 3), dtype=np.uint8)

for y in range(new_height):
    for x in range(new_width):
        x_old = x / zoom_factor
        y_old = y / zoom_factor

        scaled_data[y, x, :] = bilinear_interpolation(image, x_old, y_old)

scaled_image = Image.fromarray(scaled_data)
scaled_image.save("istanbul_zoomed_3.jpg")
scaled_image.show()

```