# HW0 - Turn your favorite photo a rotating video

**Total Points: 100**

**Assignment Due: Sep 11th Thursday 11:59 PM EST**

In this assignment, you will:

- Implement rotation using **forward mapping** (origin & arbitrary center)
- Implement rotation using **inverse mapping** (origin & arbitrary center)
- Animate your image rotating and export as an MP4 video

Please submit the following files on Canvas:

- The original .ipynb notebook
- A PDF version of the .ipynb notebook
- rotation.mp4
- rotating_favorite_video.mp4
- rotating_favorite_creative_video.mp4

Allowed libraries: **numpy, matplotlib (plt), imageio, opencv-python (cv2)**

In [1]:

```
!pip install numpy matplotlib imageio opencv-python
```

```
Collecting numpy
  Downloading numpy-2.3.3-cp311-cp311-macosx_14_0_arm64.whl.metadata (6
2 kB)
Collecting matplotlib
  Downloading matplotlib-3.10.6-cp311-cp311-macosx_11_0_arm64.whl.metad
ata (11 kB)
Collecting imageio
  Downloading imageio-2.37.0-py3-none-any.whl.metadata (5.2 kB)
Collecting opencv-python
  Using cached opencv_python-4.12.0.88-cp37-abi3-macosx_13_0_arm64.whl.
metadata (19 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.3-cp311-cp311-macosx_11_0_arm64.whl.metadat
a (5.5 kB)
Collecting cycler>=0.10 (from matplotlib)
  Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.59.2-cp311-cp311-macosx_10_9_universal2.whl.m
etadata (109 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
```

```
  Downloading kiwisolver-1.4.9-cp311-cp311-macosx_11_0_arm64.whl.metada
ta (6.3 kB)
Requirement already satisfied: packaging>=20.0 in /Users/lauhityareddy/
Repos/CS485_584/.conda/lib/python3.11/site-packages (from matplotlib) (
25.0)
Collecting pillow>=8 (from matplotlib)
  Using cached pillow-11.3.0-cp311-cp311-macosx_11_0_arm64.whl.metadata
(9.0 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Using cached pyparsing-3.2.3-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in /Users/lauhityar
eddy/Repos/CS485_584/.conda/lib/python3.11/site-packages (from matplotl
ib) (2.9.0.post0)
Collecting numpy
  Using cached numpy-2.2.6-cp311-cp311-macosx_14_0_arm64.whl.metadata (
62 kB)
Requirement already satisfied: six>=1.5 in /Users/lauhityareddy/Repos/C
S485_584/.conda/lib/python3.11/site-packages (from python-dateutil>=2.7
->matplotlib) (1.17.0)
Downloading matplotlib-3.10.6-cp311-cp311-macosx_11_0_arm64.whl (8.1 M
B)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.1/8.1 MB 43.9 MB/s  0:00:
00
Downloading imageio-2.37.0-py3-none-any.whl (315 kB)
Using cached opencv_python-4.12.0.88-cp37-abi3-macosx_13_0_arm64.whl (3
7.9 MB)
Using cached numpy-2.2.6-cp311-cp311-macosx_14_0_arm64.whl (5.4 MB)
Downloading contourpy-1.3.3-cp311-cp311-macosx_11_0_arm64.whl (270 kB)
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.59.2-cp311-cp311-macosx_10_9_universal2.whl (2.
8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.8/2.8 MB 64.9 MB/s  0:00:
00
Downloading kiwisolver-1.4.9-cp311-cp311-macosx_11_0_arm64.whl (65 kB)
Using cached pillow-11.3.0-cp311-cp311-macosx_11_0_arm64.whl (4.7 MB)
Using cached pyparsing-3.2.3-py3-none-any.whl (111 kB)
Installing collected packages: pyparsing, pillow, numpy, kiwisolver, fo
nttools, cycler, opencv-python, imageio, contourpy, matplotlib
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10/10 [matplotlib]0 [matplo
tlib]on]
Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.59.2 i
mageio-2.37.0 kiwisolver-1.4.9 matplotlib-3.10.6 numpy-2.2.6 opencv-pyt
hon-4.12.0.88 pillow-11.3.0 pyparsing-3.2.3
```

Use `imageio.imread` to load the sample image.

In [33]:

```python
import imageio
import matplotlib.pyplot as plt

img = imageio.imread('Lenna.png')
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
/var/folders/mt/1m2_35k16877hdt1r69wsj1m0000gn/T/ipykernel_84089/360405
2135.py:4: DeprecationWarning: Starting with ImageIO v3 the behavior of
this function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as im
ageio` or call `imageio.v2.imread` directly.
  img = imageio.imread('Lenna.png')
```

# Part A. Forward Mapping (30 points)

## Explanation

- **Equation:**

$$x_d = R(\theta)\, x_s$$

where

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

- **Insight:** *Where will the pixel land in the destination? (Throwing pixels forward).*

- **Pseudocode:**

```
for each pixel (x_s, y_s) in source:
    [x_d, y_d] = R(theta) * [x_s, y_s]
    put source(x_s, y_s) into destination(round(x_d),
round(y_d))
```

## Q1-1. Implement `forward_mapping_origin` (15 points)

Write a function that rotates around the origin using forward mapping.

In [21]:

```
import numpy as np

def R(theta_rad):
    return np.array([[np.cos(theta_rad), -np.sin(theta_rad)], [np.sin(

def forward_mapping_origin(img, theta_deg):
    #convert the angle to radians
    theta_rad = np.deg2rad(theta_deg)

    R_func = R(theta_rad)

    #get the height and width of the image
    src_h, src_w = img.shape[:2]

    #create a new image with the same height and width
    dst_h = src_h
    dst_w = src_w
    out = np.zeros((dst_h, dst_w,3),dtype=np.uint8)

    #iterate over the image
    for i in range(src_h):
        for j in range(src_w):
            #get the coordinates of the pixel
            x_s = j  # column is x-coordinate
            y_s = i  # row is y-coordinate

            #apply the rotation matrix to the coordinates
            source_point = np.array([x_s, y_s])
            dest_point = R_func @ source_point
            x_d = int(round(dest_point[0]))
            y_d = int(round(dest_point[1]))

            #check bounds and put the pixel in the new image
            if 0 <= x_d < dst_w and 0 <= y_d < dst_h:
                out[y_d, x_d] = img[i, j]
    return out
```

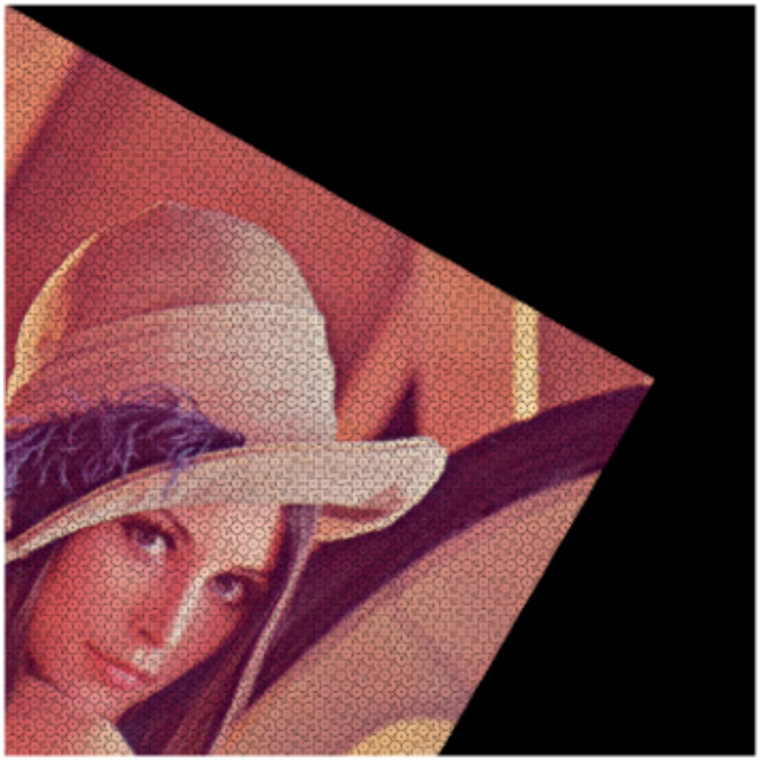
## Q1-2. Rotate by 30 degrees and display (5 points)

The image should look like it is rotated by -30 degree because in a Numpy array, the y-axis increases downward, whereas in Cartesian coordinates, the y-axis increases upward.

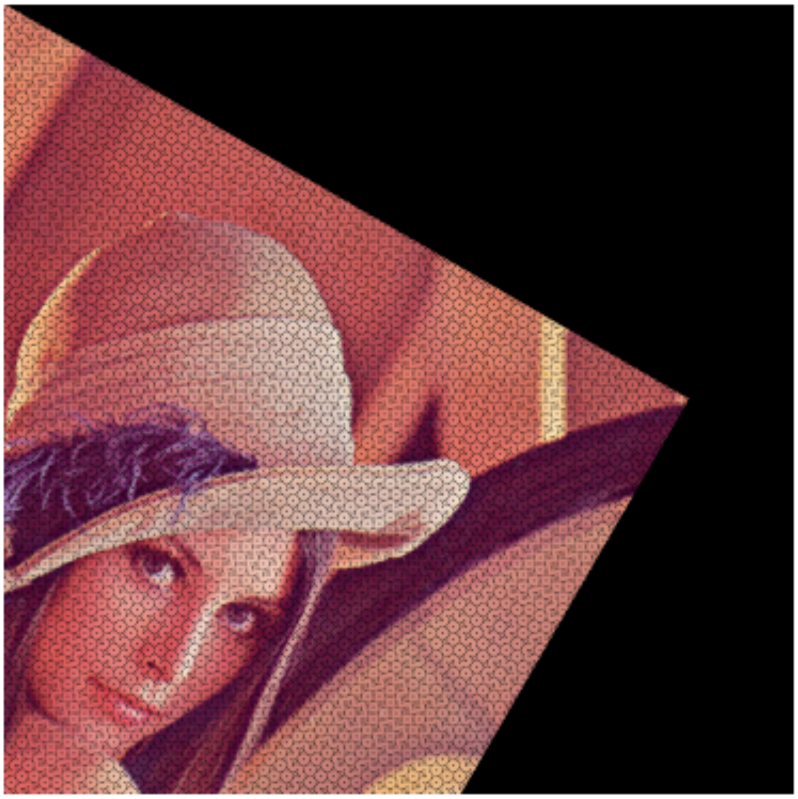
The expected output looks as follows:

In [ ]:
```
expected_output = imageio.imread('HW0_Q1_2_expected_output.png')
plt.imshow(expected_output)
plt.axis('off')
plt.show()
```

```
/tmp/ipython-input-188146722.py:1: DeprecationWarning: Starting with Im
ageIO v3 the behavior of this function will switch to that of iio.v3.im
read. To keep the current behavior (and make this warning disappear) us
e `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  expected_output = imageio.imread('HW0_Q1_2_expected_output.png')
```



In [22]:
```
# Plot the output here.
rotated = forward_mapping_origin(img, 30)
plt.imshow(rotated)
plt.axis('off')
plt.show()
```



## Q1-3. Implement rotation around arbitrary coordinate (15 points)

Hint: shift coordinates so that the chosen point is treated as the origin before rotation.

In [41]:

```python
def forward_mapping_arbitrary(img, theta_deg, cx, cy):
    # TODO: Implement forward mapping around arbitrary point (cx, cy)
    theta_rad = np.deg2rad(theta_deg)
    src_h, src_w = img.shape[:2]
    dst_h = src_h
    dst_w = src_w

    out = np.zeros((dst_h, dst_w,3),dtype=np.uint8)

    R_func = R(theta_rad)

    for i in range(src_h):
        for j in range(src_w):
            # Get source coordinates
            x_s = j
            y_s = i

            # Translate to make (cx, cy) the origin
            x_shifted = x_s - cx
            y_shifted = y_s - cy
            # Apply rotation around the new origin
            source_point = np.array([x_shifted, y_shifted])
            rotated_point = R_func @ source_point

            # Translate back to original coordinate system
            x_d = int(round(rotated_point[0] + cx))
            y_d = int(round(rotated_point[1] + cy))

            # Check bounds and put the pixel in the new image
            if 0 <= x_d < dst_w and 0 <= y_d < dst_h:
                out[y_d, x_d] = img[i, j]
    return out
```

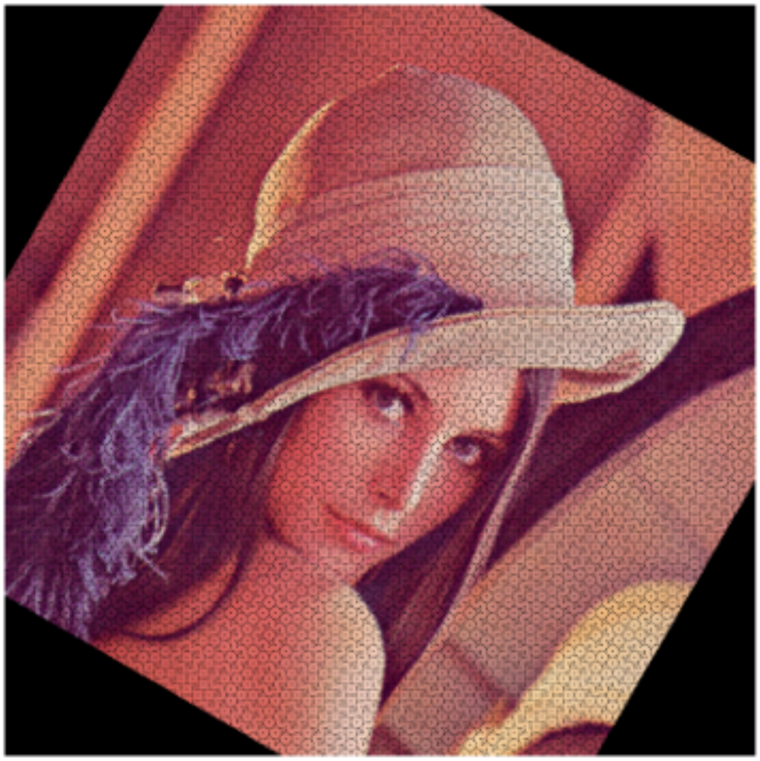
## Q1-4. Rotate with center (x=256, y=256), θ=30° and display (5 points)

The expected output looks as follows:

In [38]:

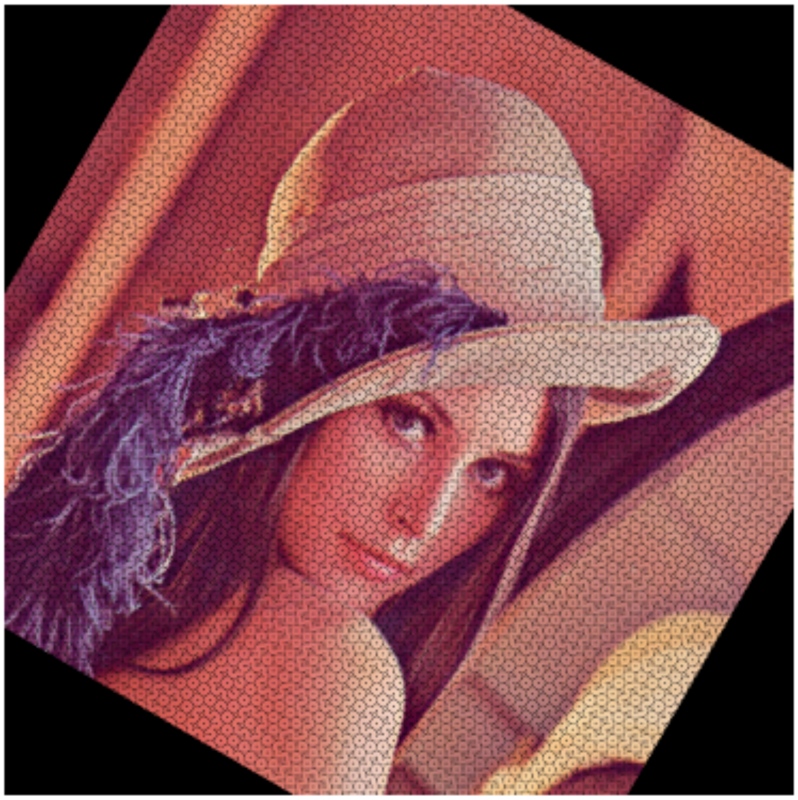
```
expected_output = imageio.imread('HW0_Q1_4_expected_output.png')
plt.imshow(expected_output)
plt.axis('off')
plt.show()
```

```
/var/folders/mt/1m2_35k16877hdt1r69wsj1m0000gn/T/ipykernel_84089/107794
347.py:1: DeprecationWarning: Starting with ImageIO v3 the behavior of
this function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as im
ageio` or call `imageio.v2.imread` directly.
  expected_output = imageio.imread('HW0_Q1_4_expected_output.png')
```



In [42]:

```
# Plot the output here.
rotated = forward_mapping_arbitrary(img, 30, 256, 256)
plt.imshow(rotated)
plt.axis('off')
plt.show()
```



**Problems with forward mapping**: holes (gap in the image) - they appear because some destination pixels are not assigned a value when multiple source pixels map to the same location or when no source pixel maps to a destination pixel. That is why we use **inverse mapping**.

# Part B. Inverse Mapping (30 points)

## Explanation

- **Equation:**

$$x_s = R(-\theta)\, x_d$$

where

$$R(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

- **Insight:** *Walk over each destination pixel and ask: where should I pull the color from in the source?*

- **Pseudocode:**

```
for each pixel (x_d, y_d) in destination:
    [x_s, y_s] = R(-theta) * [x_d, y_d]
    destination(x_d, y_d) = sample_from_source(x_s, y_s)
```

## Q2-1. Implement `inverse_mapping_origin` (15 points)

```
In [ ]: def R_inverse(theta_rad):
            return np.array([[np.cos(theta_rad), np.sin(theta_rad)], [-np.sin(

        def inverse_mapping_origin(img, theta_deg):
            # TODO: Implement inverse mapping rotation around origin
            theta_rad = np.deg2rad(theta_deg)
            src_h, src_w = img.shape[:2]
            dst_h = src_h
            dst_w = src_w

            R_inverse_func = R_inverse(theta_rad)

            out = np.zeros((dst_h, dst_w,3),dtype=np.uint8)

            for i in range(dst_h):
                for j in range(dst_w):
```

```
            # Get destination coordinates
            x_d = j
            y_d = i

            # Apply inverse rotation
            dest_point = np.array([x_d, y_d])
            source_point = R_inverse_func @ dest_point
            x_s = int(round(source_point[0]))
            y_s = int(round(source_point[1]))

            # Check bounds and sample from source to fill destination
            if 0 <= x_s < src_w and 0 <= y_s < src_h:
                out[i, j] = img[y_s, x_s]  # destination[i,j] gets sou

    return out
```

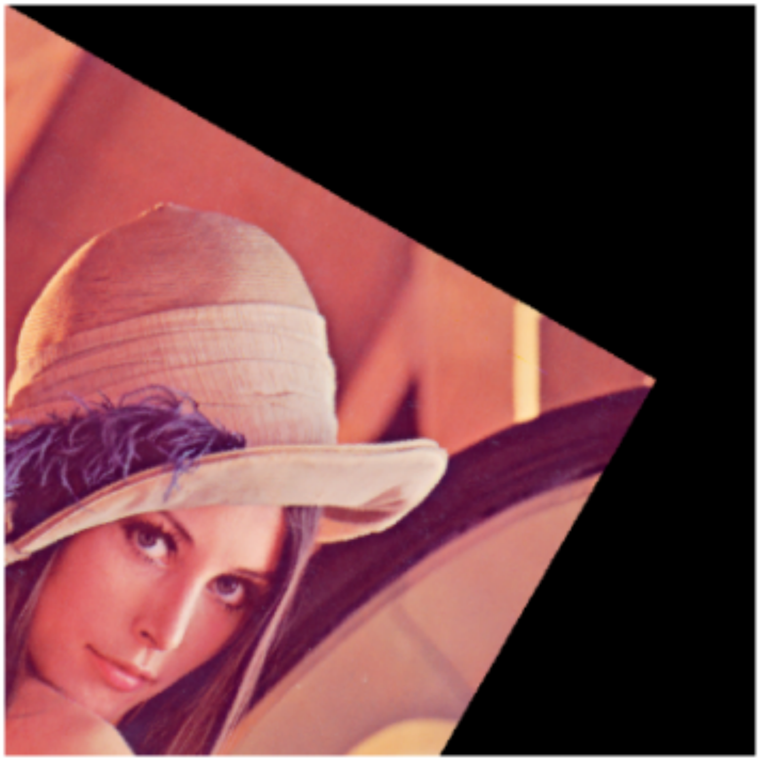
## Q2-2. Display rotation (θ=30°) (5 points)

The expected output looks as follows:

In [55]:

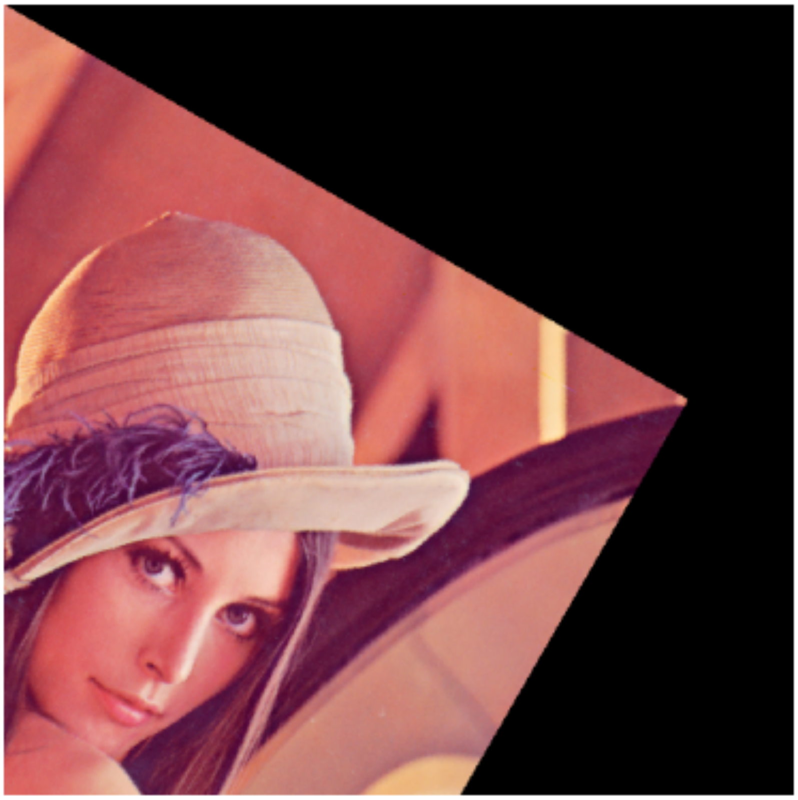
```
expected_output = imageio.imread('HW0_Q2_2_expected_output.png')
plt.imshow(expected_output)
plt.axis('off')
plt.show()
```

```
/var/folders/mt/1m2_35k16877hdt1r69wsj1m0000gn/T/ipykernel_84089/423286
5078.py:1: DeprecationWarning: Starting with ImageIO v3 the behavior of
this function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as im
ageio` or call `imageio.v2.imread` directly.
  expected_output = imageio.imread('HW0_Q2_2_expected_output.png')
```



In [56]:

```
# Plot the output here.
rotated = inverse_mapping_origin(img, 30)
plt.imshow(rotated)
plt.axis('off')
plt.show()
```



## Q2-3. Implement rotation around arbitrary center (15 points)

In [62]:

```
def inverse_mapping_arbitrary(img, theta_deg, cx, cy):
    # TODO: Implement inverse mapping around arbitrary point (cx, cy)
    theta_rad = np.deg2rad(theta_deg)
    src_h, src_w = img.shape[:2]
    dst_h = src_h
    dst_w = src_w
    R_inverse_func = R_inverse(theta_rad)

    out = np.zeros((dst_h, dst_w,3),dtype=np.uint8)

    for i in range(dst_h):
        for j in range(dst_w):
            # Get destination coordinates
            x_d = j - cx
            y_d = i - cy

            # Apply inverse rotation
            dest_point = np.array([x_d, y_d])
            source_point = R_inverse_func @ dest_point
            x_s = int(round(source_point[0]))
            y_s = int(round(source_point[1]))

            # Check bounds and sample from source to fill destination
            if 0 <= (x_s + cx) < src_w and 0 <= (y_s + cy) < src_h:
                out[i, j] = img[y_s + cy, x_s + cx]  # destination[i,j

    return out
```

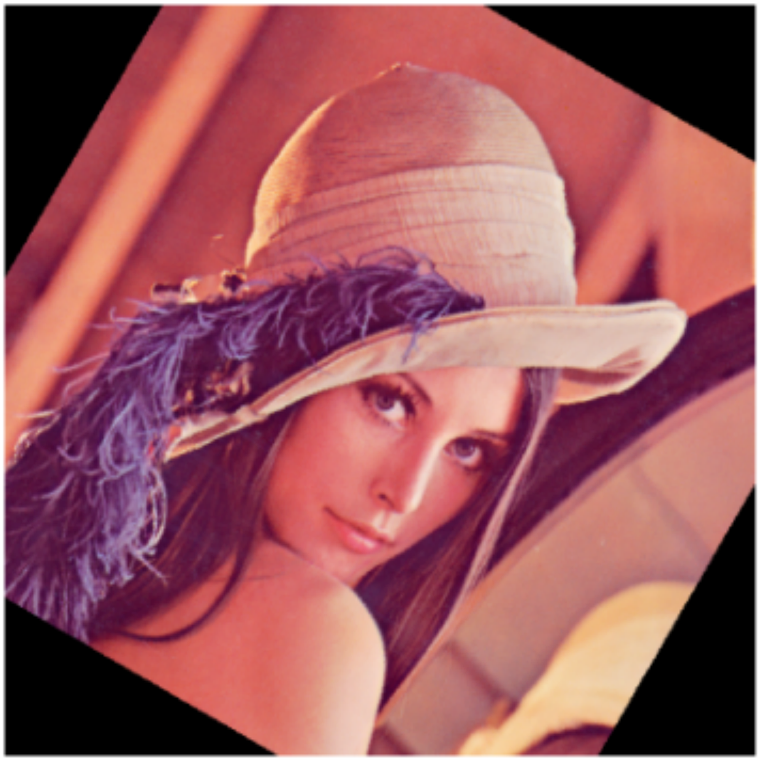
## Q2-4. Display result (x=256, y=256, θ=30°) (5 points)

The expected output looks as follows:

In [58]:

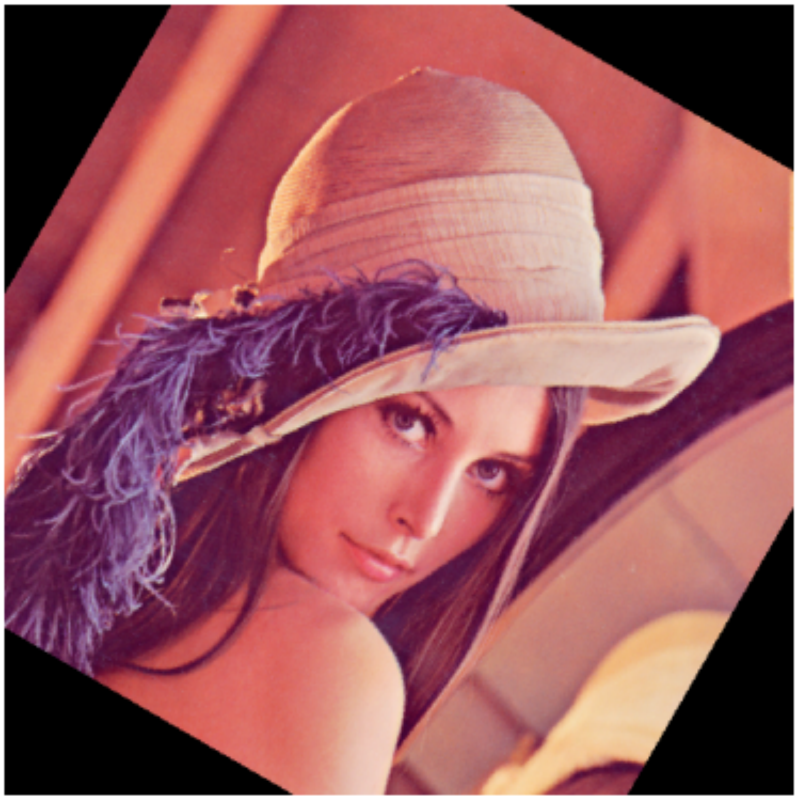
```
expected_output = imageio.imread('HW0_Q2_4_expected_output.png')
plt.imshow(expected_output)
plt.axis('off')
plt.show()
```

```
/var/folders/mt/1m2_35k16877hdt1r69wsj1m0000gn/T/ipykernel_84089/431391
595.py:1: DeprecationWarning: Starting with ImageIO v3 the behavior of
this function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as im
ageio` or call `imageio.v2.imread` directly.
  expected_output = imageio.imread('HW0_Q2_4_expected_output.png')
```



In [63]:

```
# Plot the output here
rotated = inverse_mapping_arbitrary(img, 30, 256, 256)
plt.imshow(rotated)
plt.axis('off')
plt.show()
```



# Part C. Animation (40 points)

## Q3-1. Rotation animation (15 points)

Implement a function to rotate the image continuously around the center and save as an MP4.

In [64]:

```
import cv2

def make_rotation_video(img, seconds=3, fps=24, out_file='rotation.mp4
    h, w = img.shape[:2]
    center = (w//2, h//2)
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    writer = cv2.VideoWriter(out_file, fourcc, fps, (w, h))

    # YOUR CODE HERE
    N = fps * seconds
    t = 3

    for t in range(N):

        theta = 360 * t / N

        frame = inverse_mapping_arbitrary(img, theta, *center)
        frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        writer.write(frame_bgr)
    writer.release()


make_rotation_video(img, seconds=3, fps=24)
```

## Q3-2. Turn Your Favorite Photo a Rotatinv Video (10 points)

In [66]:

```
# Load your own favorite photo
favorite_photo = imageio.imread('prison_shiv.png')
make_rotation_video(favorite_photo, seconds=3, fps=24, out_file='rotat
```

```
/var/folders/mt/1m2_35k16877hdt1r69wsj1m0000gn/T/ipykernel_84089/252689
7057.py:2: DeprecationWarning: Starting with ImageIO v3 the behavior of
this function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as im
ageio` or call `imageio.v2.imread` directly.
  favorite_photo = imageio.imread('prison_shiv.png')
```

## Q3-3. Creative Animation (15 points)

Create your own fun animation ('rotating_favorite_photo_creative.mp4'). For example:

- Image flying out of frame
- Zoom in/out while rotating
- Add trails or effects Be creative!

In [ ]:

```
def make_pizzazz_video(img, seconds=3, fps=24, out_file='rotating_favo
    h, w = img.shape[:2]
    center_y, center_x = h // 2, w // 2
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    writer = cv2.VideoWriter(out_file, fourcc, fps, (w, h))

    N = fps * seconds

    # Calculate the maximum radius needed to cover the entire image
    max_radius = int(np.sqrt((center_x)**2 + (center_y)**2)) + 1

    for t in range(N):
        # Create progress from 0 to 1
        progress = t / (N - 1)

        # Start with a small radius and grow to cover the whole image
        min_radius = 5  # Start with a small circle
        current_radius = int(min_radius + (max_radius - min_radius) *

        # Create a black canvas
        frame = np.zeros((h, w, 3), dtype=np.uint8)

        # Create a circular mask that grows over time
        for y in range(h):
            for x in range(w):
                # Calculate distance from center
                distance = np.sqrt((x - center_x)**2 + (y - center_y)*

                # If within the current radius, show the original pixe
                if distance <= current_radius:
                    frame[y, x] = img[y, x]

        # Convert to BGR for OpenCV
        frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        writer.write(frame_bgr)

    writer.release()

make_pizzazz_video(favorite_photo, seconds=3, fps=24, out_file='rotati
```

In [ ]: