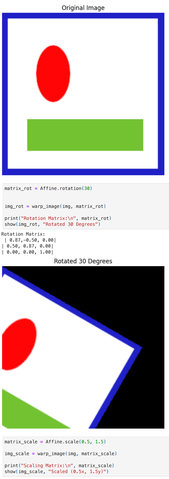
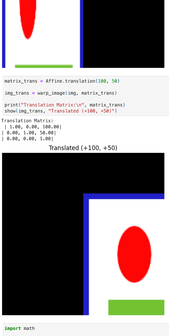
Task 8.1

In [33]:
```
import rasterio
from rasterio.transform import Affine
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import affine_transform

def warp_image(image, affine_matrix):
    matrix_inv = ~affine_matrix

    mat_2x2 = np.array([[matrix_inv.a, matrix_inv.d],
                        [matrix_inv.b, matrix_inv.e]])
    offset = [matrix_inv.f, matrix_inv.c]


    channels = []
    for i in range(image.shape[0]):
        result = affine_transform(
            image[i],
            mat_2x2,
            offset=offset,
            output_shape=image.shape[1:],
            order=1,
            mode='constant',
            cval=0
        )
        channels.append(result)

    return np.array(channels)


with rasterio.open('abstract.png') as src:
    img = src.read()


def show(arr, title):
    plt.figure(figsize=(6,6))
    plt.imshow(np.transpose(arr, (1, 2, 0)))
    plt.title(title)
    plt.axis('off')
    plt.show()

show(img, "Original Image")
```



In [34]:
```
matrix_rot = Affine.rotation(30)


img_rot = warp_image(img, matrix_rot)

print("Rotation Matrix:\n", matrix_rot)
show(img_rot, "Rotated 30 Degrees")
```

```
Rotation Matrix:
| 0.87,-0.50, 0.00|
| 0.50, 0.87, 0.00|
| 0.00, 0.00, 1.00|
```



In [35]:
```
matrix_scale = Affine.scale(0.5, 1.5)

img_trans = warp_image(img, matrix_scale)

print("Scaling Matrix:\n", matrix_scale)
show(img_scale, "Scaled (0.5x, 1.5y)")
```

```
Scaling Matrix:
| 0.50, 0.00, 0.00|
| 0.00, 1.50, 0.00|
| 0.00, 0.00, 1.00|
```

In [36]:
```
matrix_trans = Affine.translation(100, 50)

img_trans = warp_image(img, matrix_trans)

print("Translation Matrix:\n", matrix_trans)
show(img_trans, "Translated (+100, +50)")
```

```
Translation Matrix:
| 1.00, 0.00, 100.00|
| 0.00, 1.00, 50.00|
| 0.00, 0.00, 1.00|
```

In [37]:
```
import math


with rasterio.open('osm_1.png') as s1: osm1 = s1.read()
with rasterio.open('osm_2.png') as s2: osm2 = s2.read()

s  = 1.2     # Scale
r  = -15.0   # Rotation (Degrees)
tx = 550     # Translation X (Right)
ty = 175     # Translation Y (Down)


M_align = Affine.translation(tx, ty) * Affine.rotation(r) * Affine.scale(s, s)

# 2. Warp
inv = ~M_align
inv_affine = (inv.a, inv.d, inv.b, inv.e)
off = (inv.f, inv.c)

aligned_layers = []
for i in range(osm2.shape[0]):
    aligned_layers.append(
        affine_transform(osm2[i], mat, offset=off, output_shape=osm1.shape[1:], order=1, cval=0)
    )
aligned_final = np.array(aligned_layers)

# 3. Visualization
plt.figure(figsize=(12, 12))
plt.imshow(np.transpose(osm1, (1, 2, 0)))

mask = np.any(aligned_final > 0, axis=0)
plt.imshow(np.transpose(aligned_final, (1, 2, 0)), alpha=0.6)

plt.title(f"Alignment Check\nScale: {s}, Rot: {r}°, Offset: ({tx}, {ty})")
plt.show()

print("Final Matrix:\n", M_align)
```


```
Final Matrix:
| 1.16, 0.31, 550.00|
|-0.31, 1.16, 175.00|
| 0.00, 0.00, 1.00|
```

Task 8.2

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
import rasterio
from scipy.ndimage import gaussian_filter

def fill_gaps(image, method):
    """
    Fills gaps (0 values) in an image using the specified interpolation method.
    """

    mask = (image != 0)


    coords_y, coords_x = np.where(mask)
    values = image[mask]


    grid_y, grid_x = np.mgrid[0:image.shape[0], 0:image.shape[1]]

    # 1. Standard Scipy Methods
    if method in ['nearest', 'linear', 'cubic']:
        filled = griddata((coords_y, coords_x), values, (grid_y, grid_x), method=method)
        if np.isnan(filled).any():
            filled[np.isnan(filled)] = griddata((coords_y, coords_x), values,
                                                (grid_y[np.isnan(filled)], grid_x[np.isnan(filled)]),
                                                method='nearest')

        return filled

    # 2. Bilinear (Approximation using Linear)
    elif method == 'bilinear':

        return fill_gaps(image, 'linear')

    # 3. Gaussian (Convolution based filling)
    elif method == 'gaussian':

        sigma = 1.0
        blurred = gaussian_filter(image.astype(float), sigma=sigma)


        filled = image.astype(float).copy()
        filled[~mask] = blurred[~mask]

        k_size = int(2*4*sigma + 1)

        return filled

    # 4. Lanczos (Resampling kernel)
    elif method == 'lanczos':
        print("Note: Lanczos is a resampling kernel. Using Cubic as close approximation for gap filling.")
        return fill_gaps(image, 'cubic')

with rasterio.open('satellite.png') as src:
    sat_img = src.read(1)


methods = ['nearest', 'linear', 'cubic', 'gaussian']

plt.figure(figsize=(15, 10))

plt.subplot(2, 3, 1)
plt.imshow(sat_img, cmap='gray')
plt.title("Original (with gaps)")
plt.axis('off')


for i, m in enumerate(methods):

    try:
        res = fill_gaps(sat_img, m)

        plt.subplot(2, 3, i+2)
        plt.imshow(res, cmap='gray')
        plt.title(f"Method: {m.capitalize()}")
        plt.axis('off')
    except Exception as e:
        print(f"Error with {m}: {e}")

plt.tight_layout()
plt.show()
```

Comparison of Methods:

1. Nearest Neighbor: Fast but produces blocky/pixelated artifacts. Best for categorical data (e.g. classification maps), worst for continuous imagery.
2. Linear/Bilinear: Smooths the data but can blur edges. Good balance of speed and quality.
3. Cubic: Produces smoother gradients than linear but can introduce 'overshoot' (ringing) artifacts near sharp edges. Computationally more expensive.
4. Gaussian: Effectively blurs the missing areas. Good for noise reduction but loses high-frequency detail.
5. Lanczos: (Similar to Cubic) theoretically best for preserving detail while suppressing aliasing, but computationally heavy and prone to ringing.

Generally Cubic (or Bicubic) is considered best for visual interpretation of continuous satellite imagery because it preserves continuity and gradients better than linear methods without the blockiness of nearest neighbor.

Nearest Neighbor is the worst for visual images because it introduces severe pixelation artifacts.