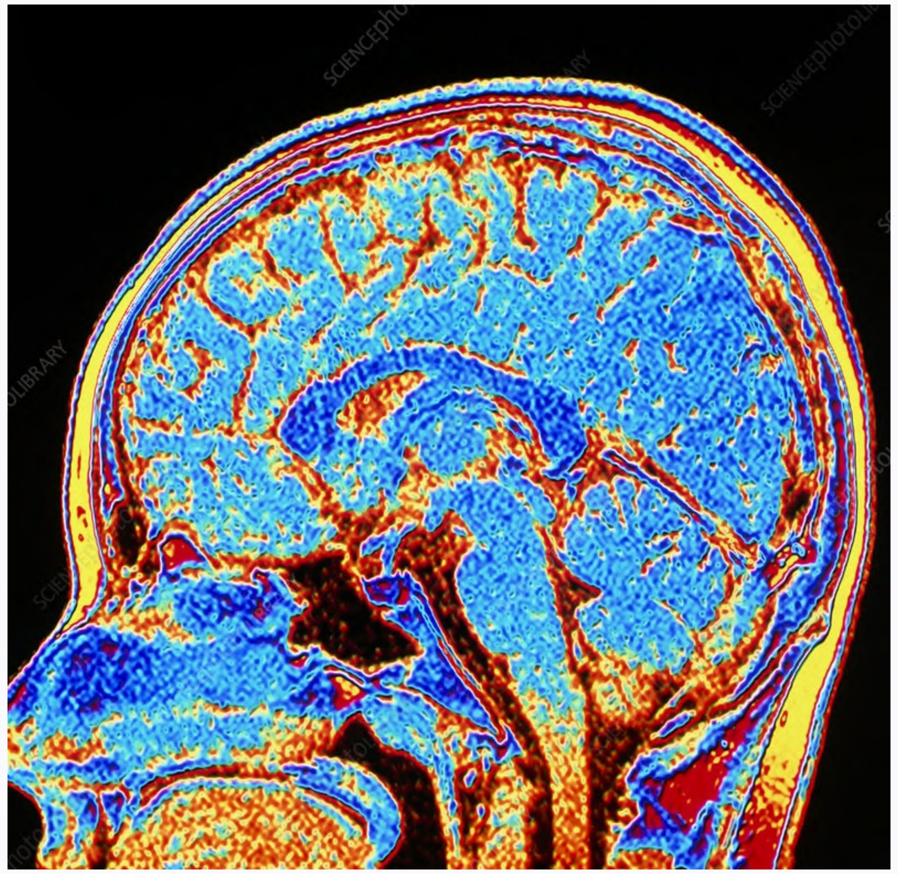
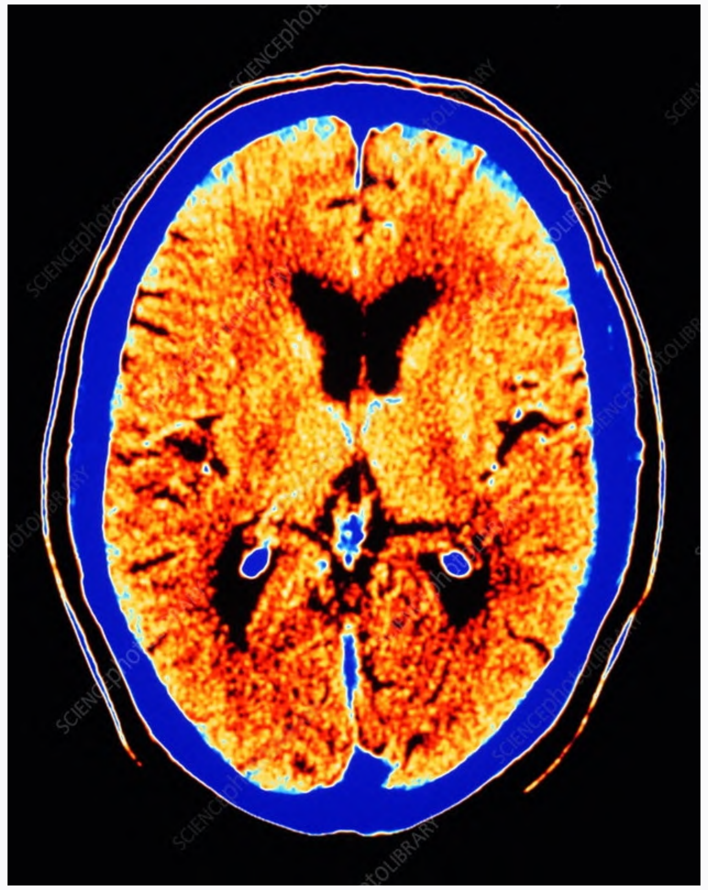
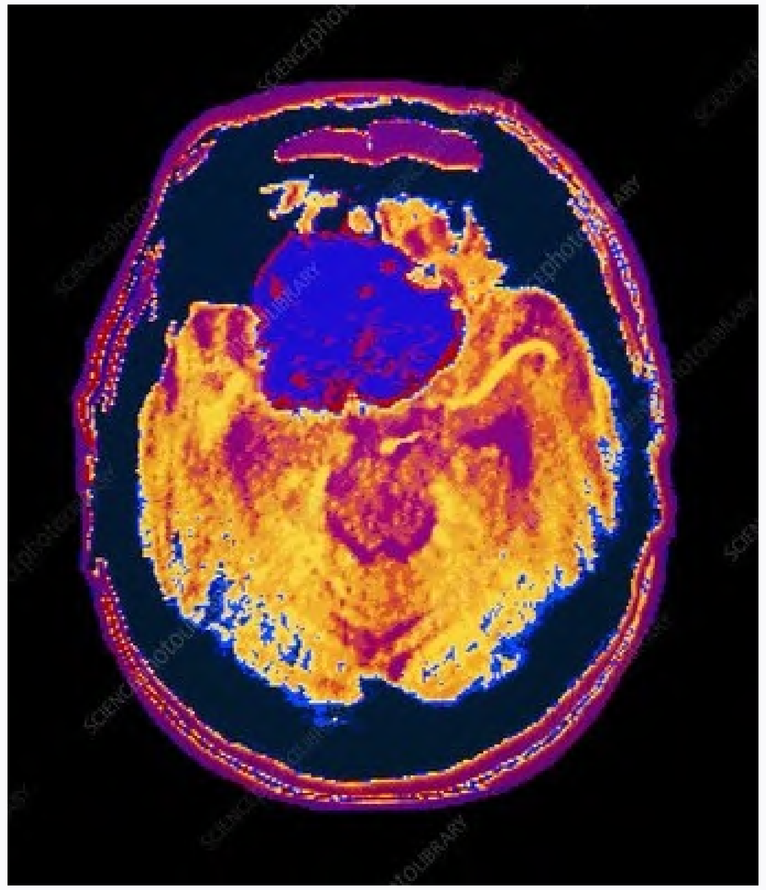
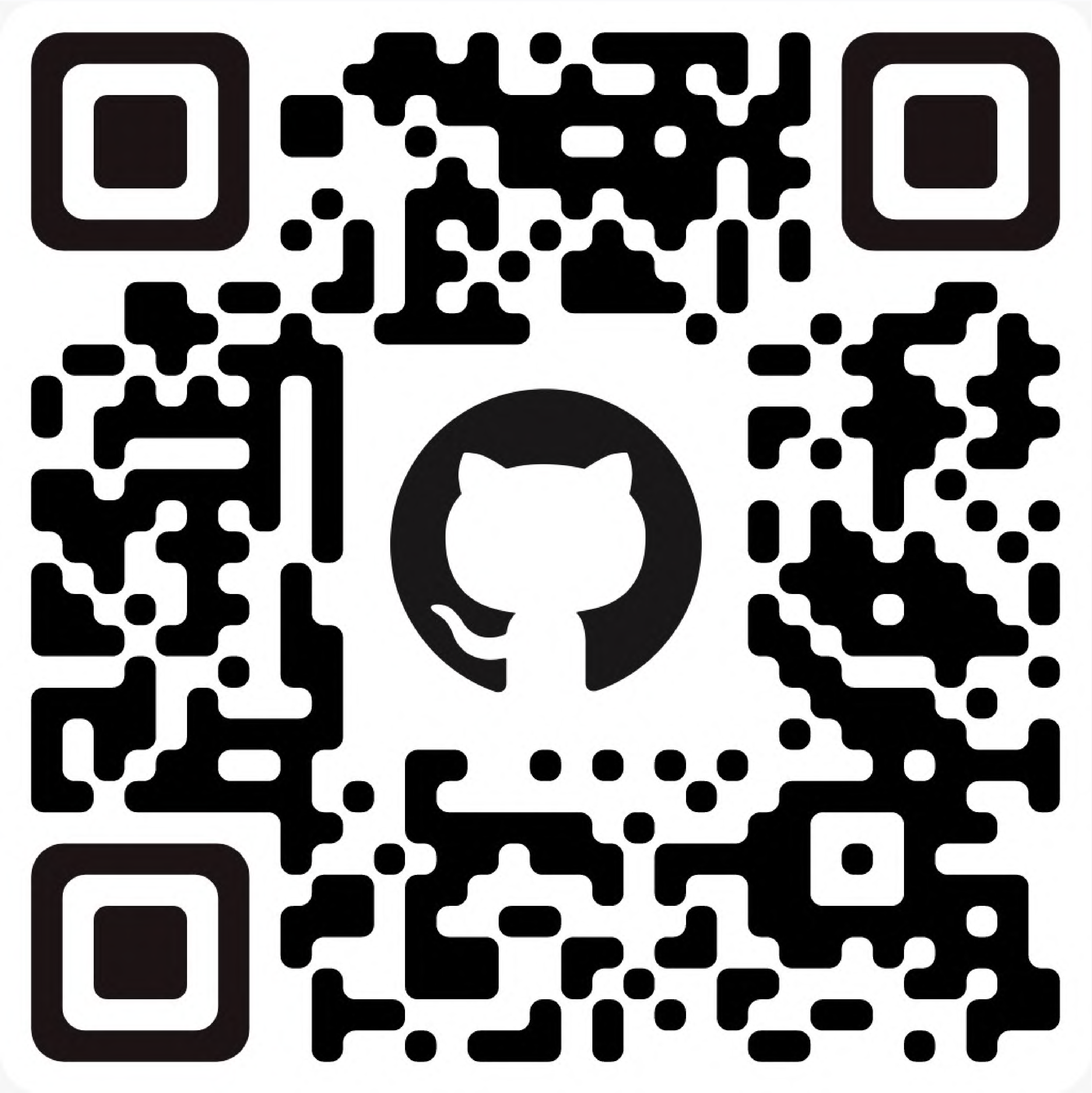
CT reconstruction
and
Windowing







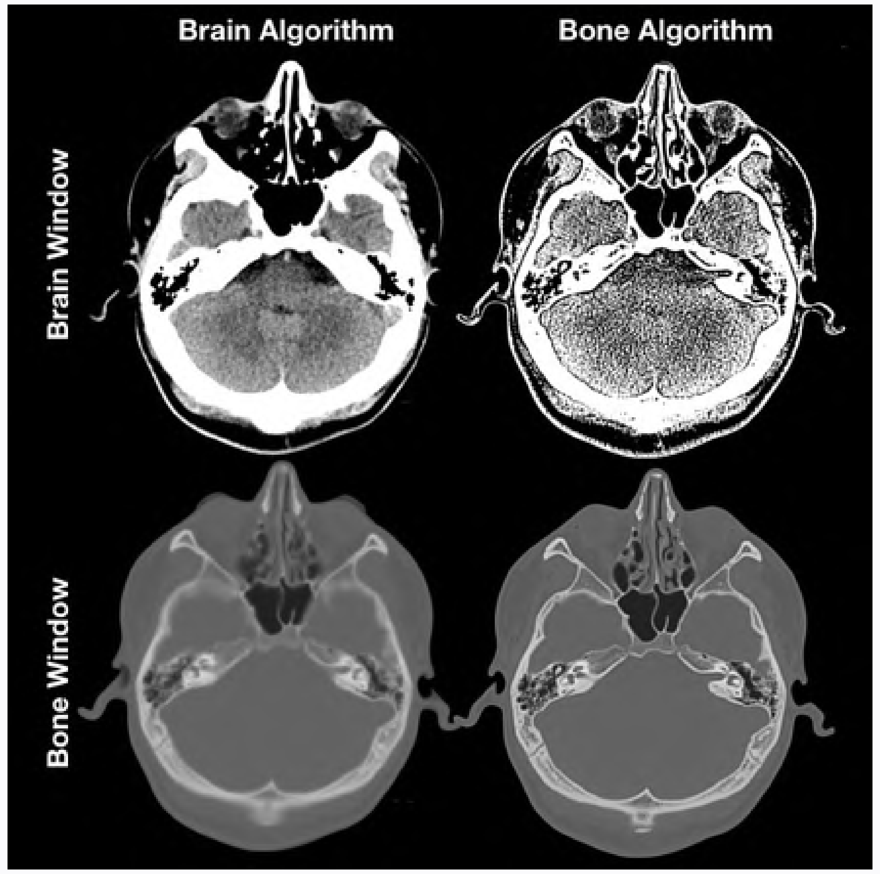
Github link for template code



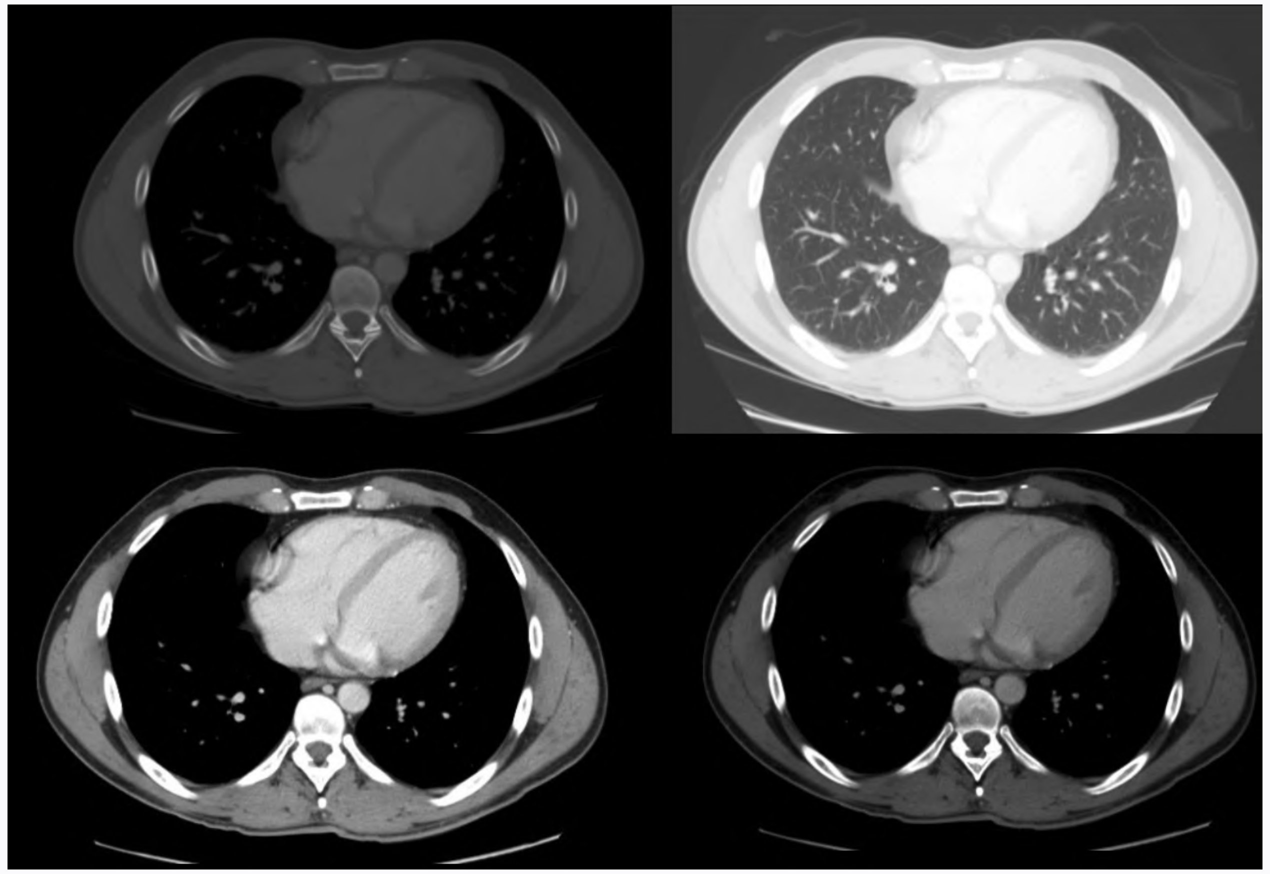
Windowing

- We choose a region of interest which lies within a given intensity range.
- Anything lying outside this intensity range is mapped to either extreme: 0 or 255
- We map the intensity values of the intensity 'window' to the range 0 to 255 via interpolation
- This allows distinction in contrast between the various regions

Brain CT scan

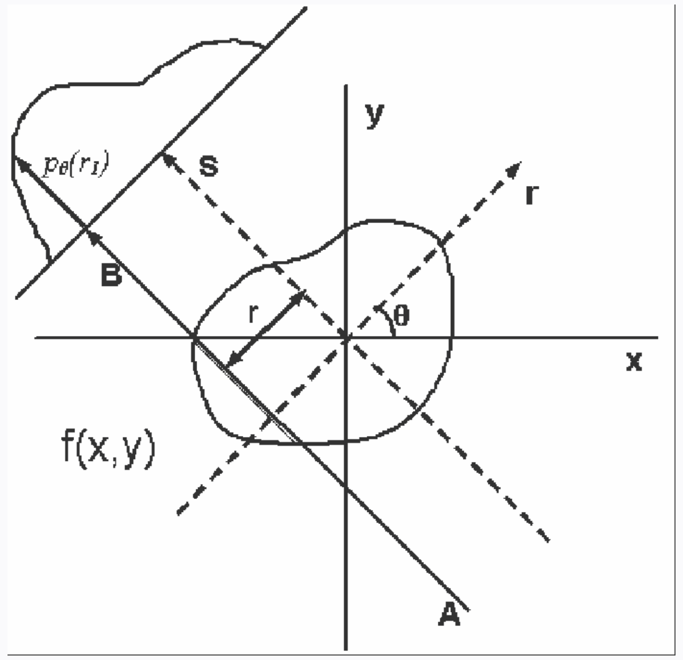


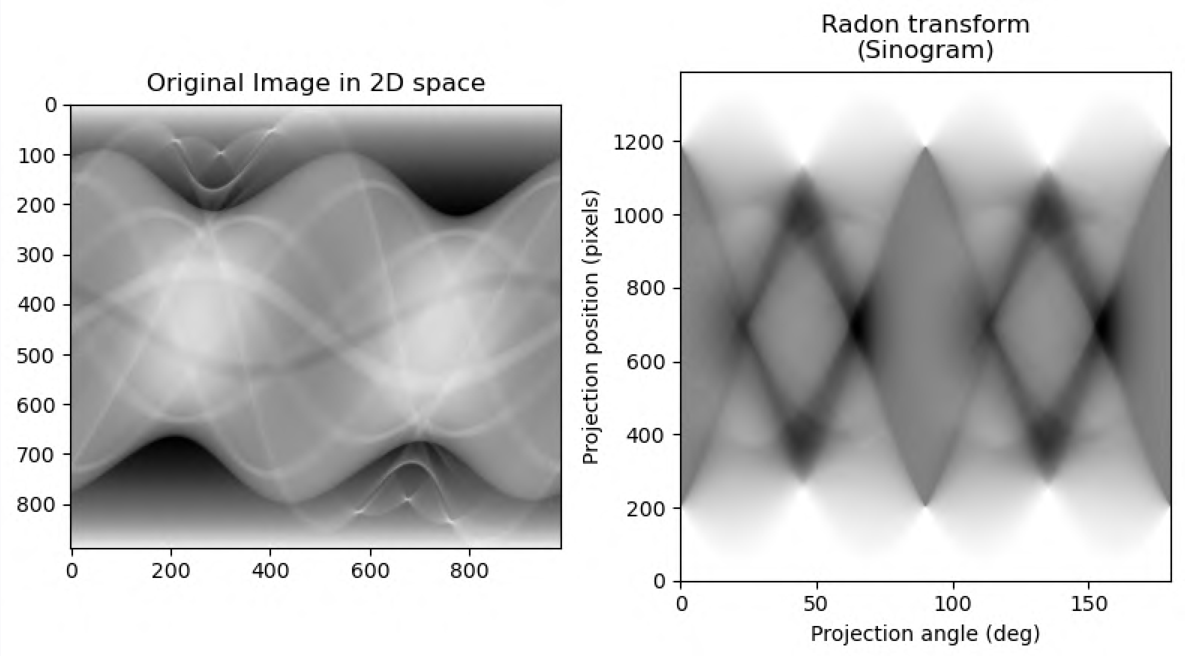
Lung CT scan

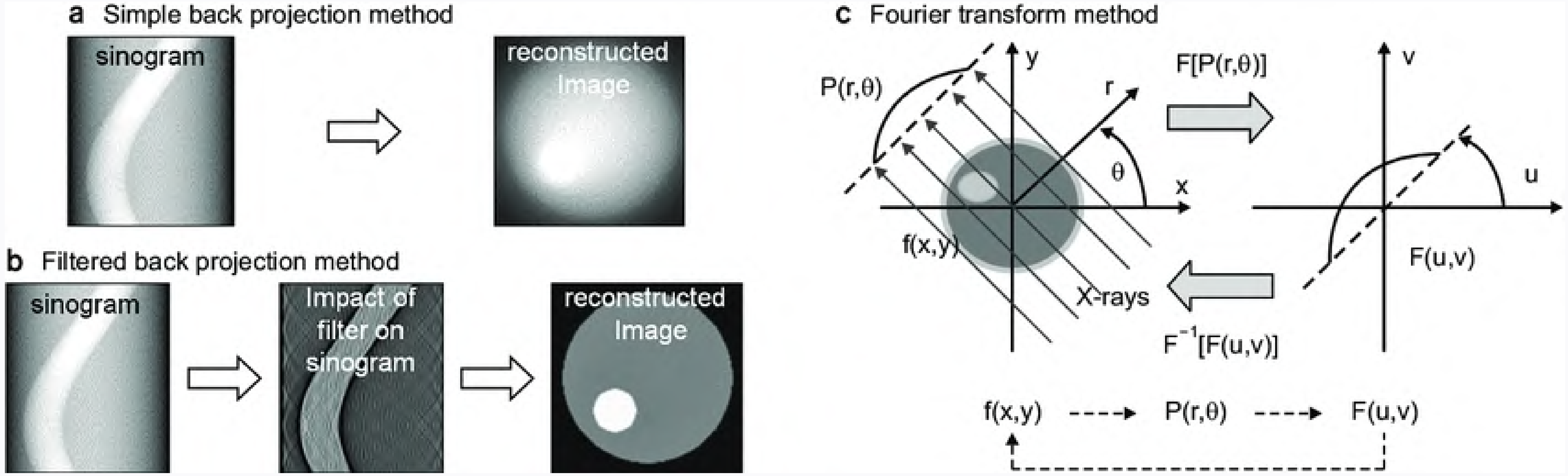


CT Reconstruction

Ray sums via Radon transform

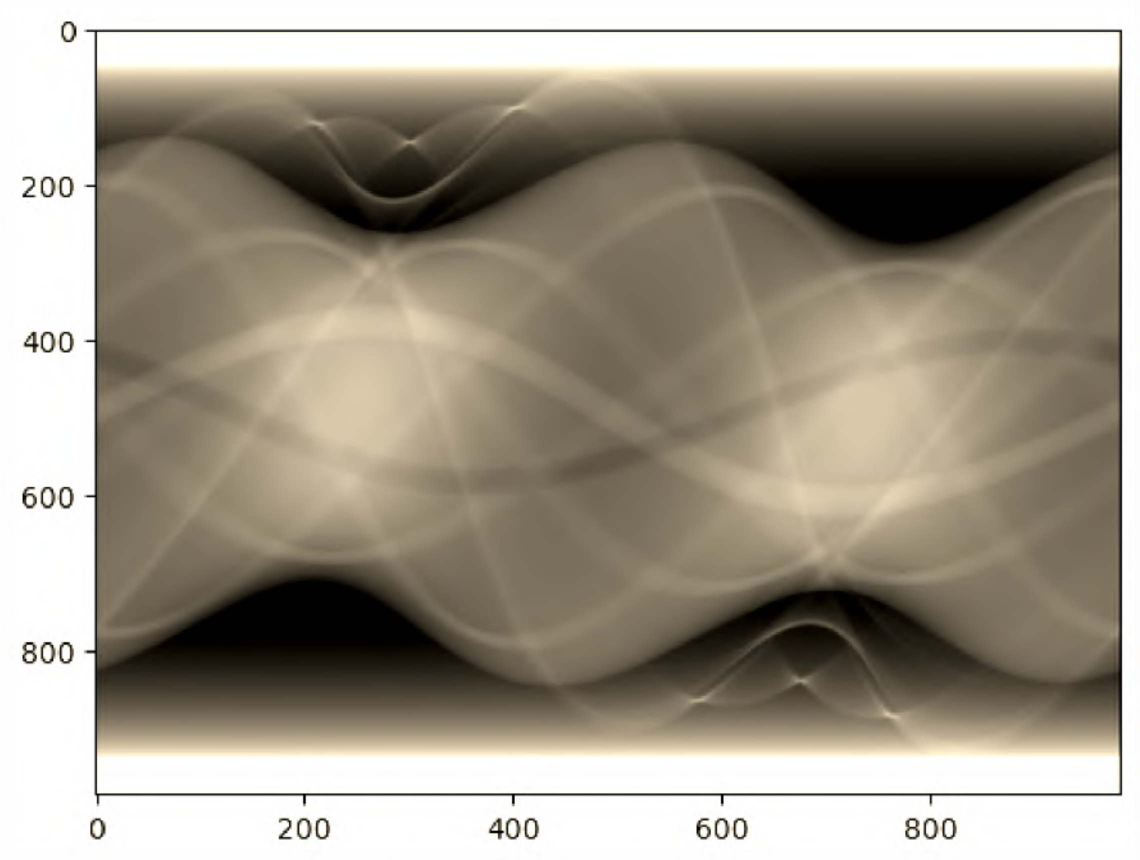




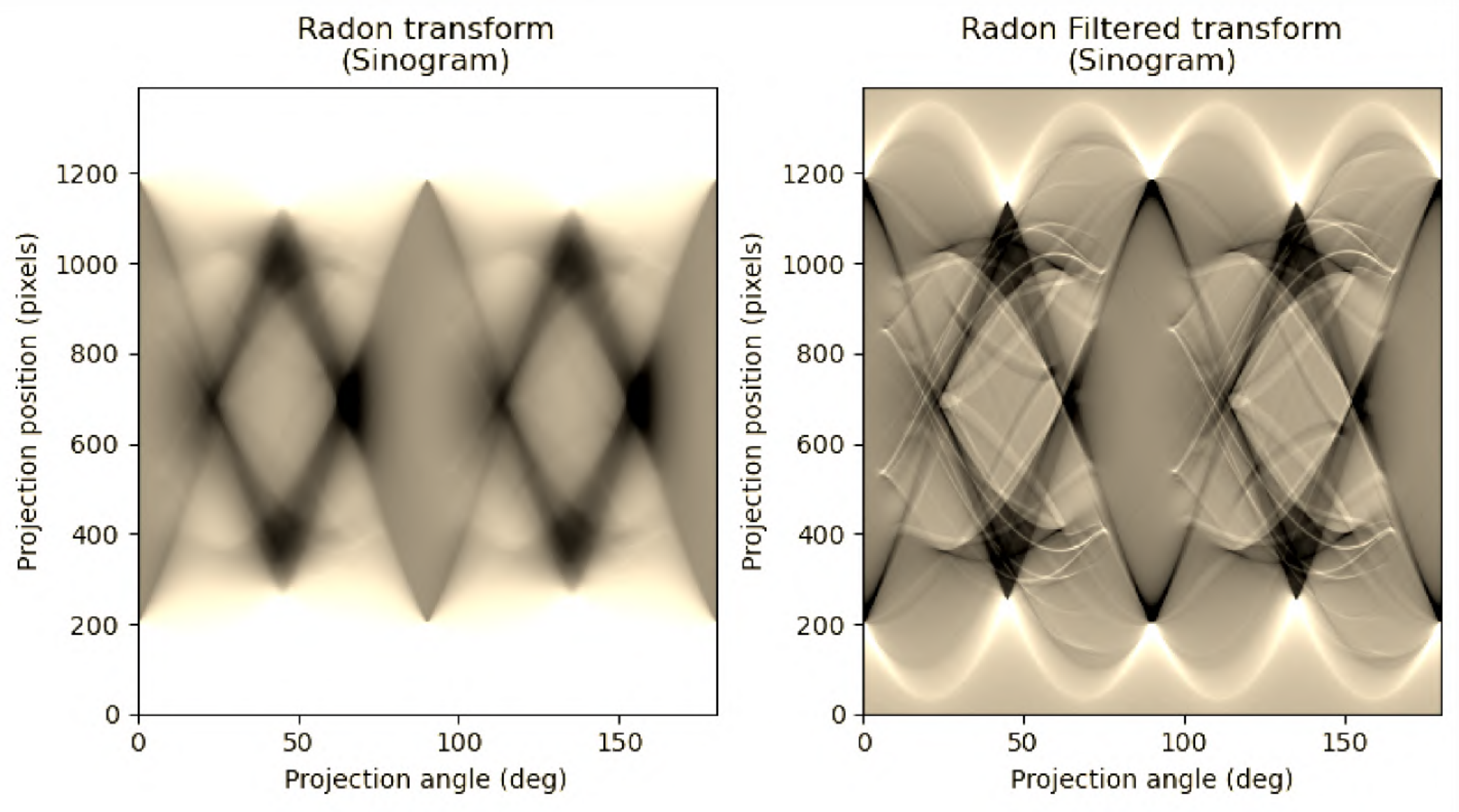


Filtering during Reconstruction

Original Fan beam data

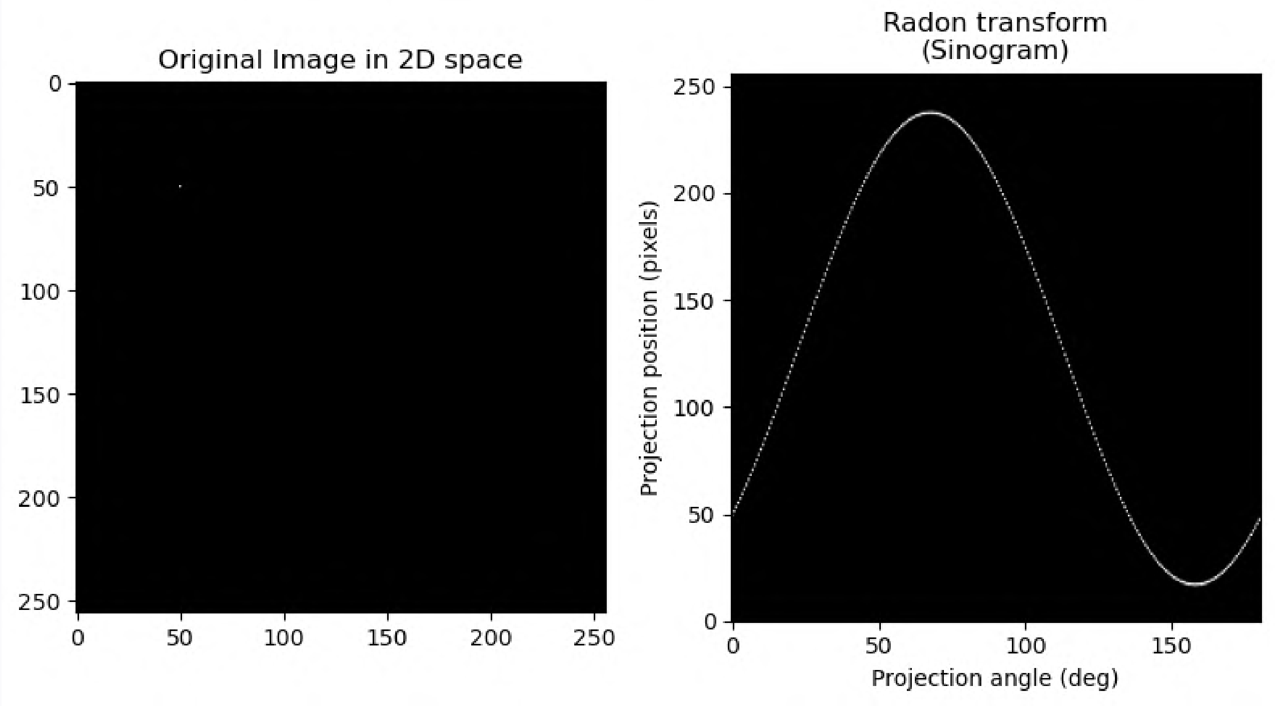


Effect of filtering on Sinogram

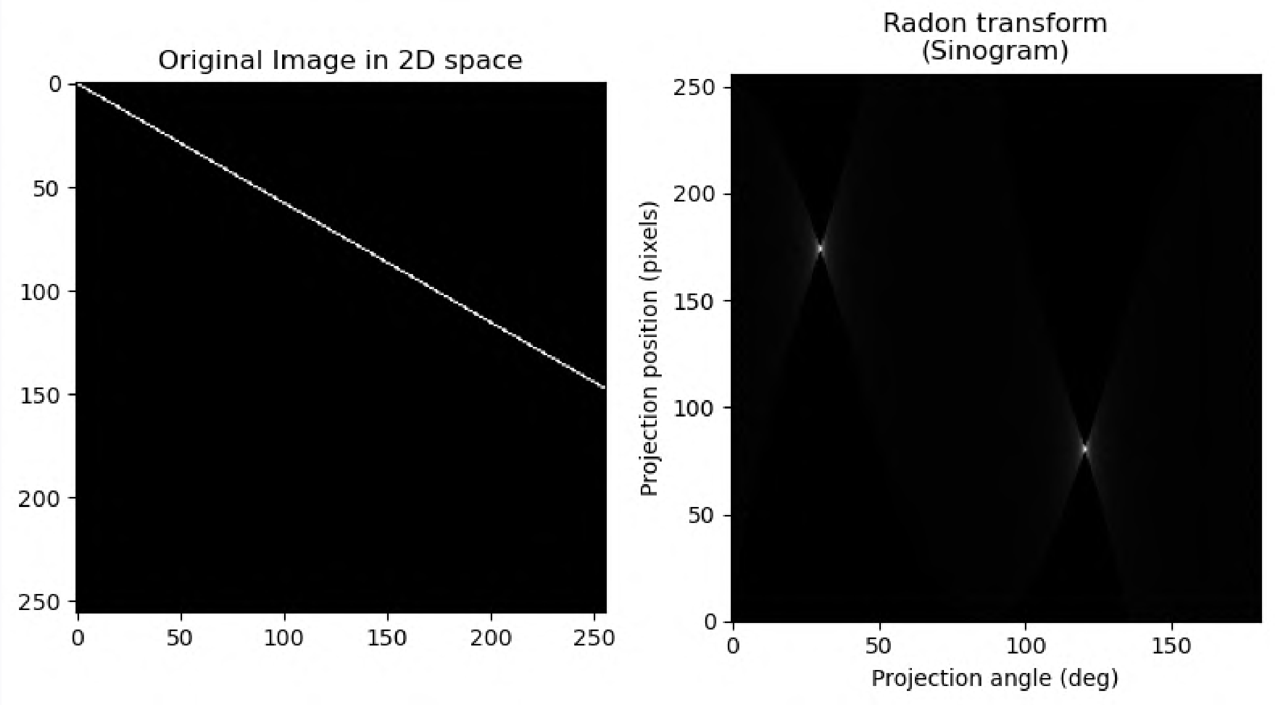


Radon Transform

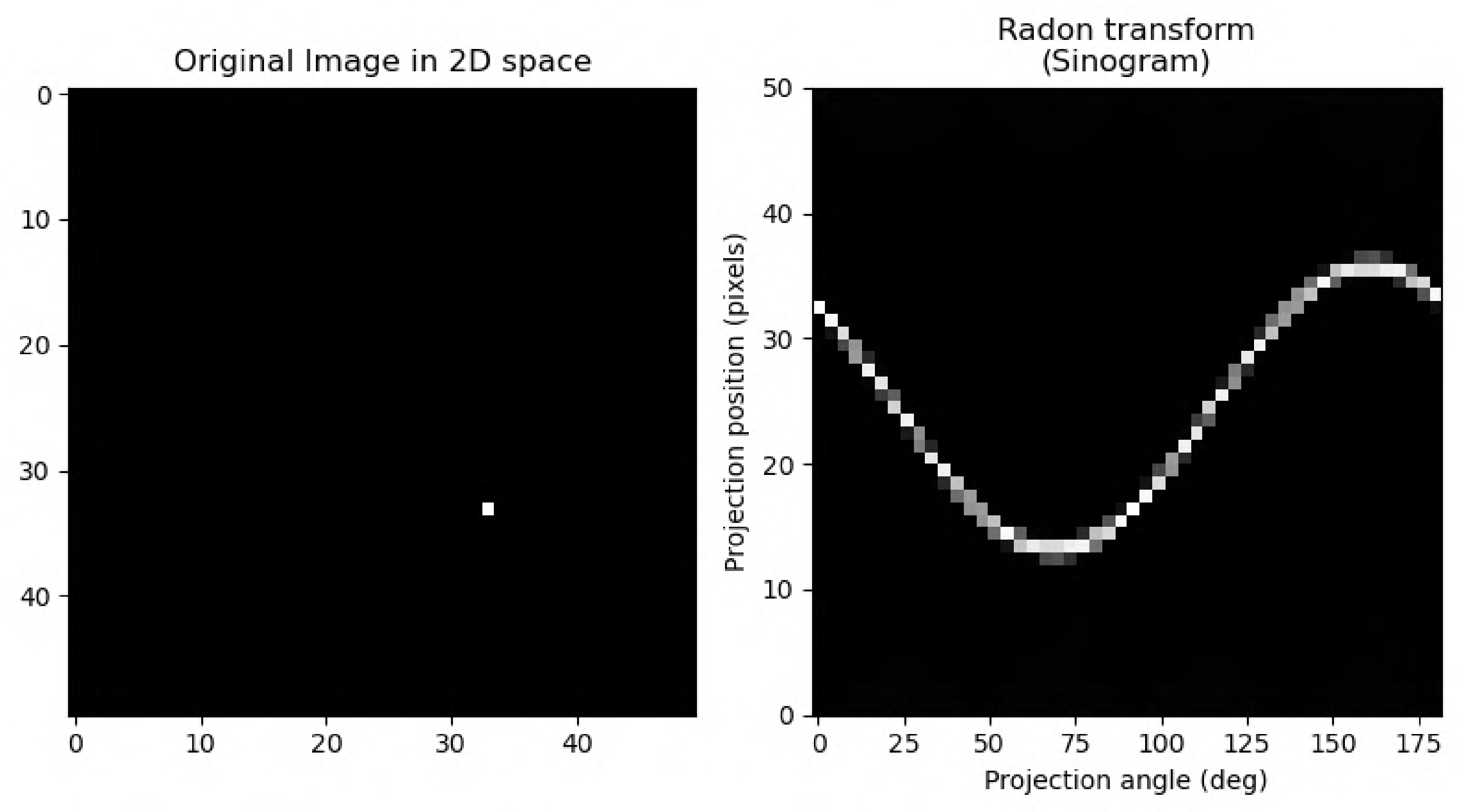
Radon transform of a point

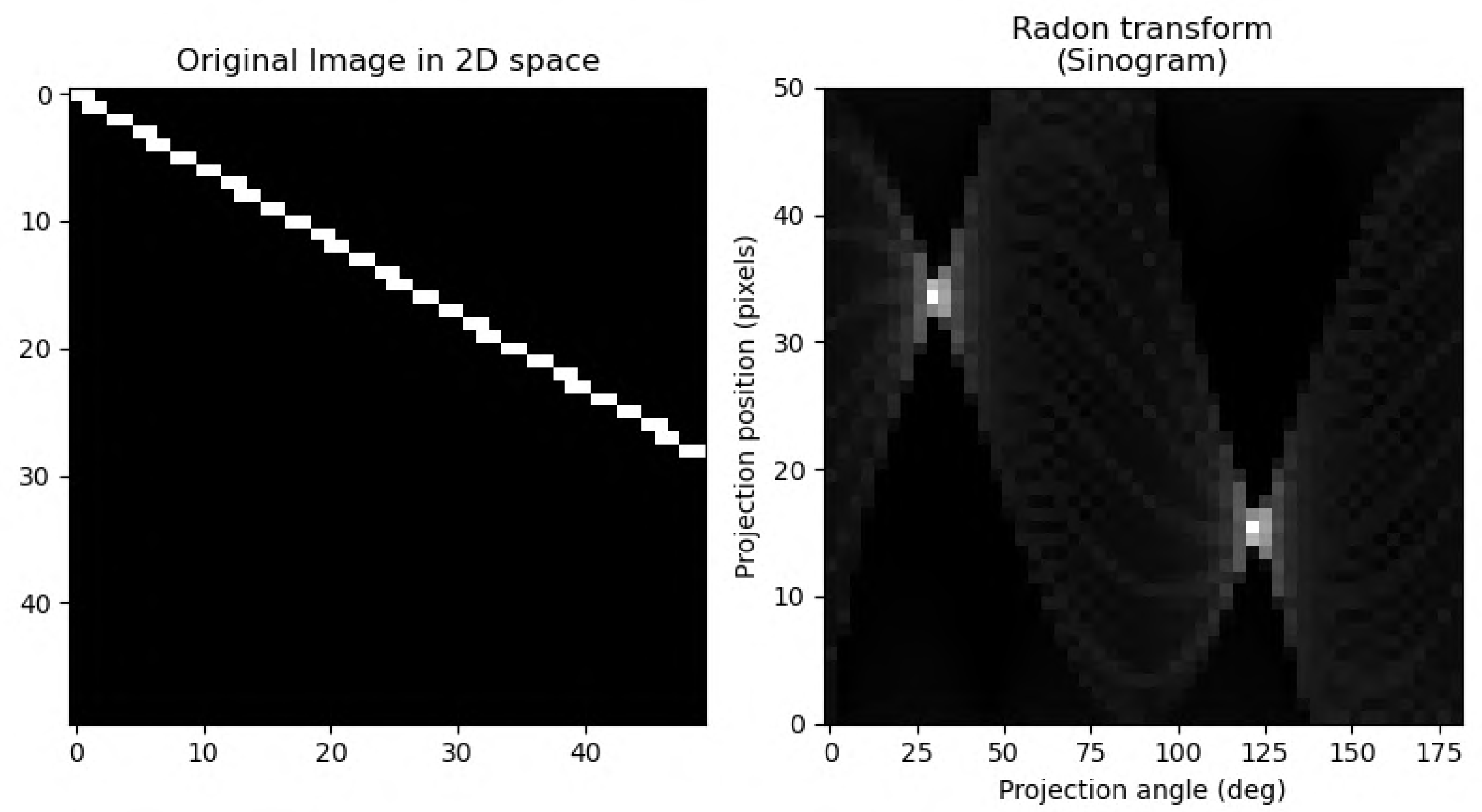


Radon transform of a line



Reducing number of detectors





CT Reconstruction: code pre-processing

```
def inv_radon_BP(radon_image, theta=None, output_size=256, interpolation="linear"):

    if theta is None:
        theta = np.linspace(0, 180, radon_image.shape[1], endpoint=False)

    angles_count = len(theta)
    if angles_count != radon_image.shape[1]:
        raise ValueError("The given ``theta`` does not match the number of "
                         "projections in ``radon_image``.")

    interpolation_types = ['linear', 'nearest', 'cubic']
    if interpolation not in interpolation_types:
        raise ValueError(f"Unknown interpolation: {interpolation}")
```

Initializing

```
radon_image = radon_image.astype(np.float32)
    dtype = radon_image.dtype

    img_shape = radon_image.shape[0]
    if output_size is None:
        # If output size not specified, estimate from input radon image
        output_size = int(np.floor(np.sqrt((img_shape) ** 2 / 2.0)))

    # Reconstruct image by interpolation
    reconstructed = np.zeros((output_size, output_size),
                             dtype=dtype)
    radius = output_size // 2
    xpr, ypr = np.mgrid[:output_size, :output_size] - radius
    x = np.arange(img_shape) - img_shape // 2
```

Reconstructing

```
    for col, angle in zip(radon_image.T, np.deg2rad(theta)):
        t = ypr * np.cos(angle) - xpr * np.sin(angle)
        if interpolation == 'linear':
            interpolant = partial(np.interp, xp=x, fp=col, left=0, right=0)
        else:
            interpolant = interp1d(x, col, kind=interpolation,
                                   bounds_error=False, fill_value=0)
        reconstructed += interpolant(t)

    return reconstructed * np.pi / (2 * angles_count)
```