STEM HAADF Image Analysis Tutorial

This tutorial demonstrates how to use the `stem_haadf_analysis` package for analyzing STEM HAADF images. The analysis includes:

1. **Image loading and preprocessing**
2. **PCA-based denoising**
3. **Diffraction pattern analysis**
4. **Power spectral density analysis**
5. **Peak detection and matching**

Import the required libraries

In [1]:
```
%matplotlib widget

import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from scipy.signal import find_peaks

import atomap.api as am
from pymatgen.core.structure import Structure
import copy

import stem_haadf_analysis as sha
plt.rcParams["figure.figsize"] = (8,8)
import importlib
```

Open files

- Heterostructure HAADF image .tif (16 bits grayscale)

In [2]:
```
# Load the main HAADF image
s, path = sha.load_image()
```

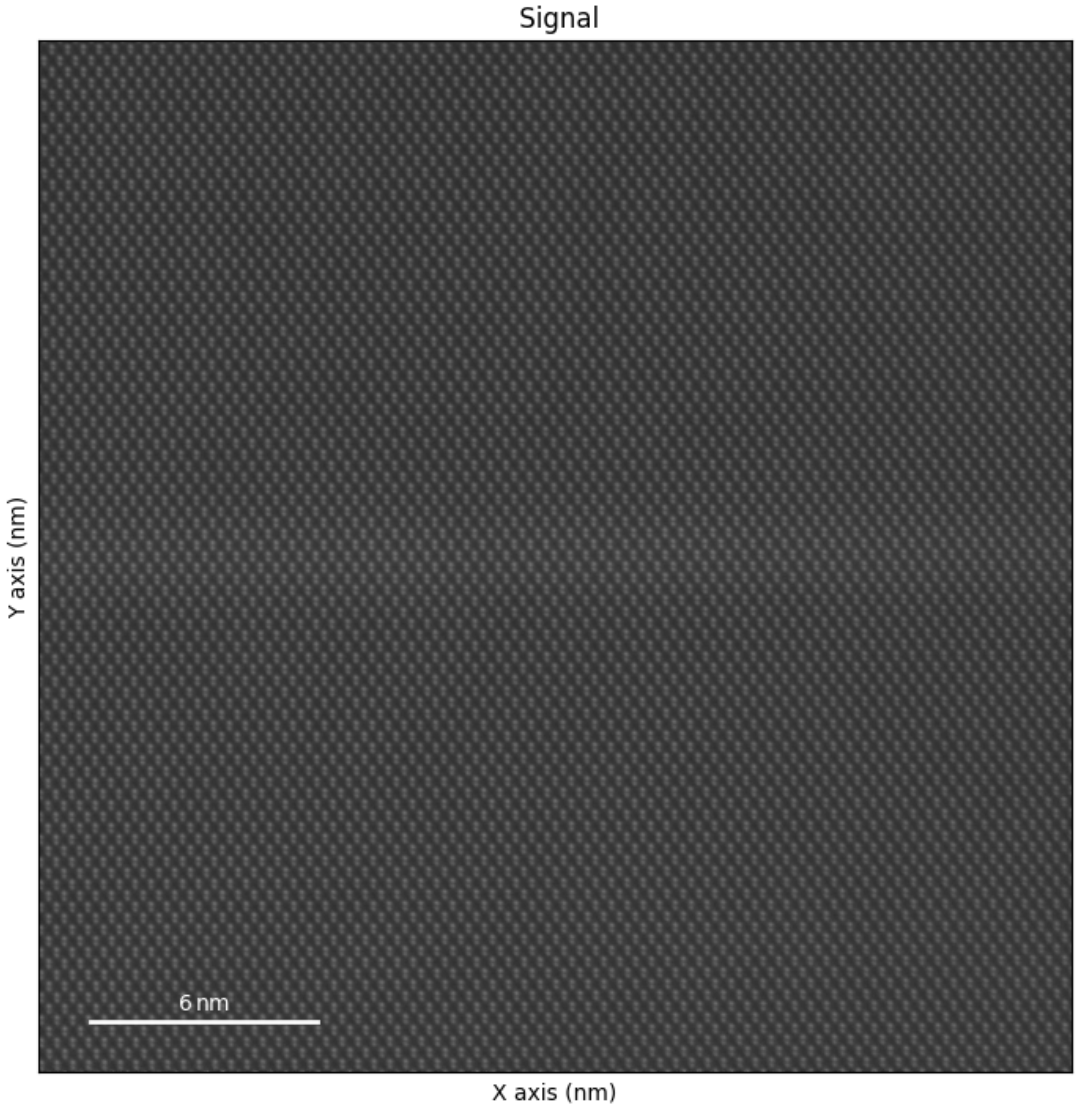
Axes scaling (pixel distance)

> **`pixel_size_pm`** : pixel size in picometers.
>
> An aspect ratio of 1 is assumed.

In [3]:

```
pixel_size_pm = 6.652

# Set up axes scaling using our utility function
s = sha.setup_axes_scaling(s, pixel_size_pm)
s.plot(colorbar=False)

ax = plt.gca()
norm = mpl.colors.Normalize(vmin=np.min(s.data), vmax=np.max(s.data))
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.15)
plt.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap="Greys_r"), ax=ax, pad=.
plt.tight_layout()
```

Figure Signal



Intensity scaling (pixel intensity)

- Detector image .tif (16 bits grayscale) - for the normalization to the impinging beam.

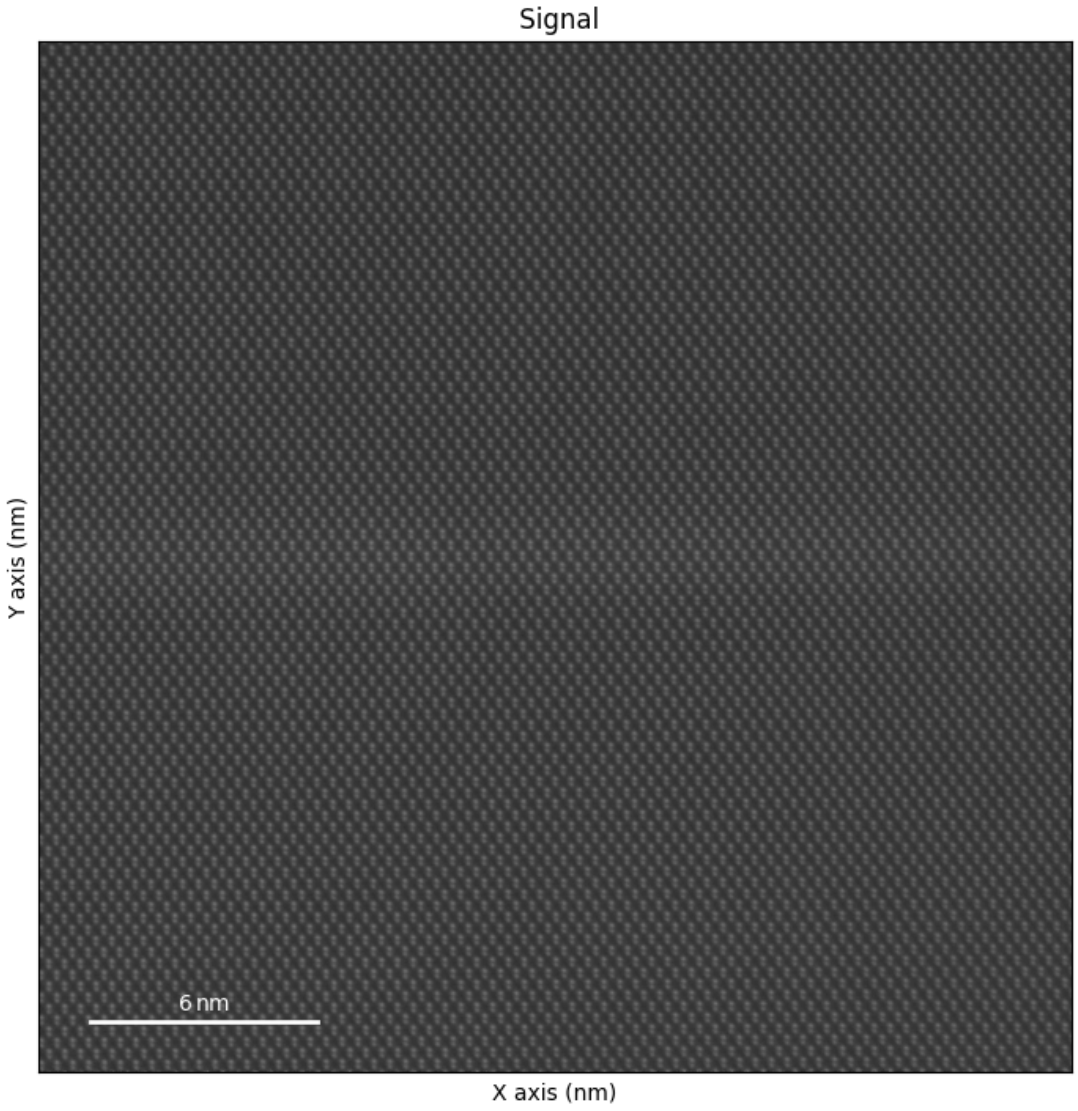
> **`inner_angle`** : inner collection angle in mrads.
> **`outer_angle`** : outer collection angle in mrads.

In [4]:

```
inner_angle = 159
outer_angle = 200

# Load detector image for normalization
det_image, _ = sha.load_image(create_dir=False)
s_normalised = am.quant.detector_normalisation(s, det_image, inner_angle=in
```

```
s_normalised.plot(colorbar=False)

ax = plt.gca()
norm = mpl.colors.Normalize(vmin=np.min(s_normalised.data), vmax=np.max(s_n
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.15)
plt.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap="Greys_r"), ax=ax, pad=.
plt.tight_layout()
```

Figure Signal

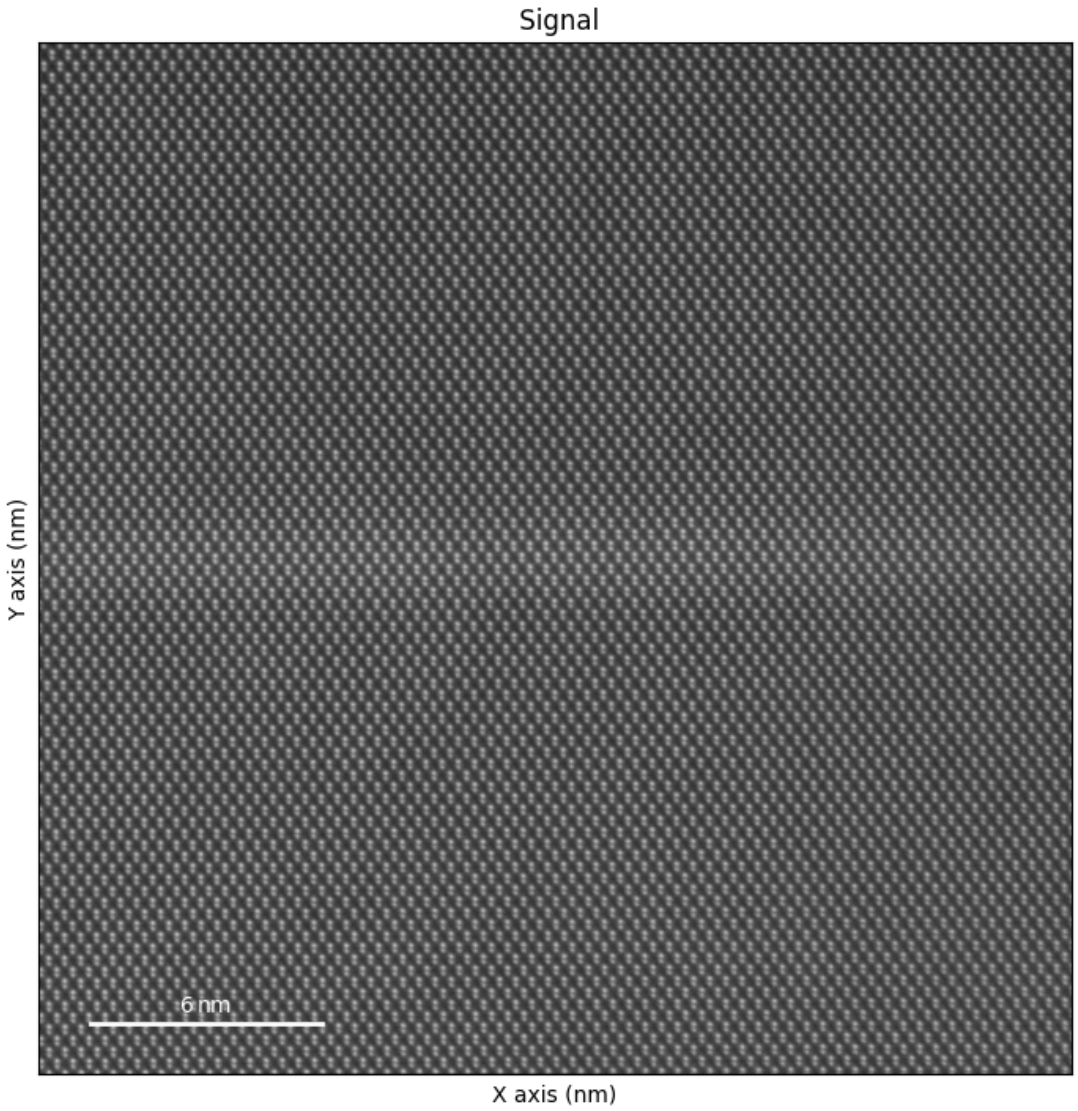


Denoising

> **`pixels_cropped`** : pixels to be cropped from the outer frame to remove side effects of the drift correction.

In [5]:
```
pixels_cropped = 32

original_imag = copy.deepcopy(s_normalised.data)
s_normalised = s_normalised.isig[pixels_cropped:-pixels_cropped, pixels_cro
s_normalised.plot(colorbar=False)

ax = plt.gca()
ax.set_title("Normalized image")
norm = mpl.colors.Normalize(vmin=np.min(s_normalised.data), vmax=np.max(s_n
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.15)
plt.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap="Greys_r"), ax=ax, pad=.
plt.tight_layout()
```

Figure Signal



In [6]:
```
# Save the normalized image
plt.savefig(rf"{path}\normalized_image.png", dpi=512, transparent=True, bbo
```

```
sha.save_image_with_metadata(
    s_normalised.data,
    rf"{path}\normalized_image.tif",
    s_normalised.axes_manager[0].scale
)
```

PCA (Principal Component Analysis):

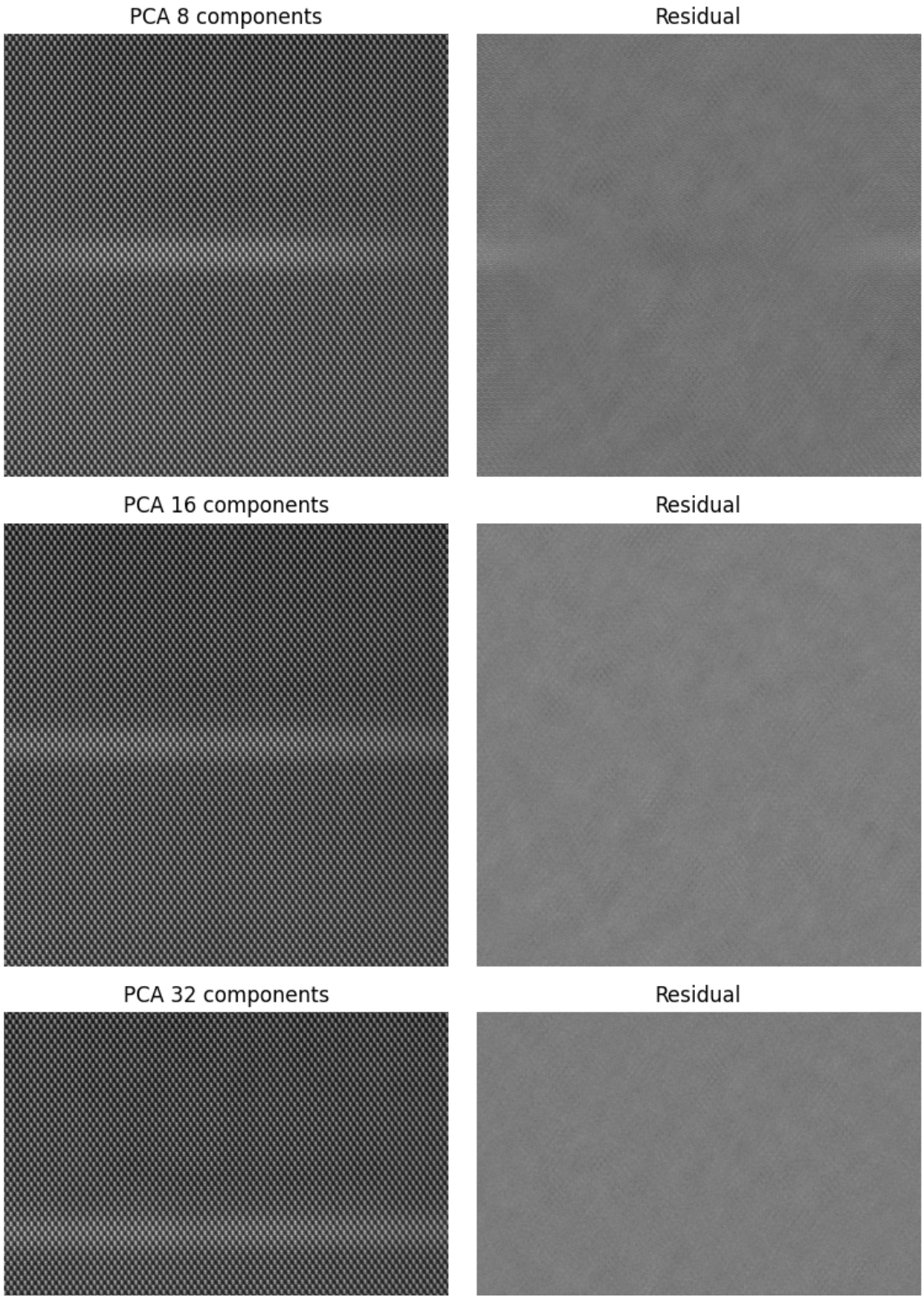
> **pca_list** : number of components in PCA.

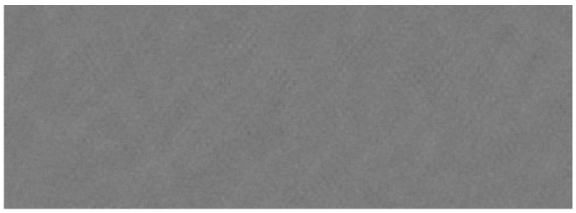
In [7]:
```
pca_list = [8, 16, 32, 64, 128, 256]

# Apply PCA denoising using our modular function
pca_results = sha.apply_pca_denoising_batch(
    original_image=original_imag,
    pixels_cropped=pixels_cropped,
    pca_components_list=pca_list
)
```

In [8]:
```
# Save individual PCA results
for idx, n in enumerate(pca_list):
    fig, axes = plt.subplots(1, 2, figsize=(8, 4))
    axes[0].imshow(pca_results[n]["pca_image"], cmap="gray")
    axes[0].set_title(f"PCA {n} components")
    axes[0].axis("off")
    axes[1].imshow(pca_results[n]["residual"], cmap="gray")
    axes[1].set_title("Residual")
    axes[1].axis("off")
    plt.tight_layout()
    plt.savefig(rf"{path}\PCA_{n}.png", dpi=512, transparent=True, bbox_inc
    plt.close(fig)

    # Save as TIFF with metadata
    sha.save_image_with_metadata(
        pca_results[n]["pca_image"],
        rf"{path}\PCA_{n}.tif",
        s_normalised.axes_manager[0].scale
    )

# Display all PCA results in one figure
fig, axs = plt.subplots(len(pca_list), 2, figsize=(8, 4*len(pca_list)))
for i, n in enumerate(pca_list):
    axs[i, 0].imshow(pca_results[n]["pca_image"], cmap="gray")
    axs[i, 0].set_title(f"PCA {n} components")
    axs[i, 0].axis("off")
    axs[i, 1].imshow(pca_results[n]["residual"], cmap="gray")
    axs[i, 1].set_title("Residual")
    axs[i, 1].axis("off")
```

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

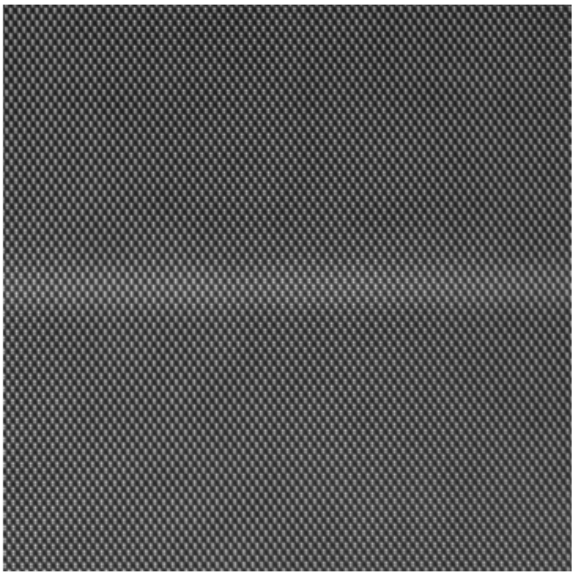
Figure

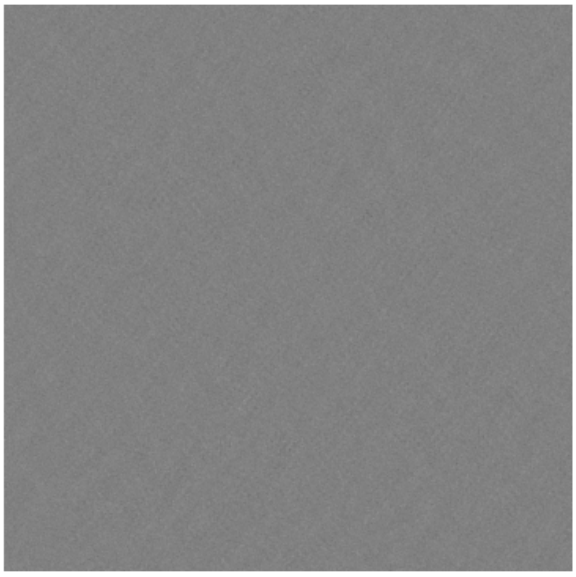




PCA 64 components

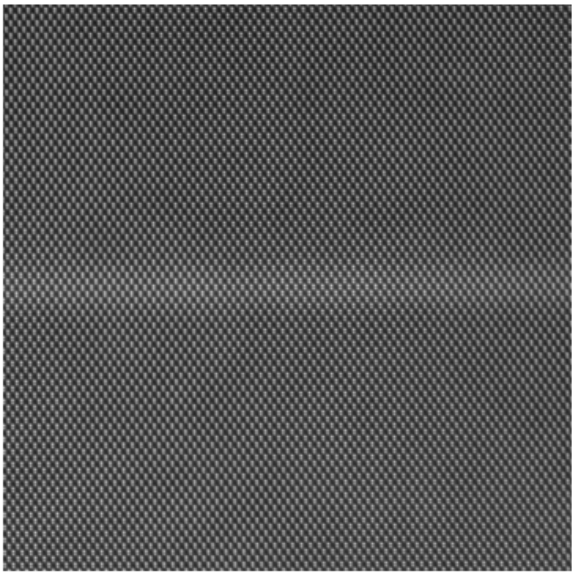
Residual

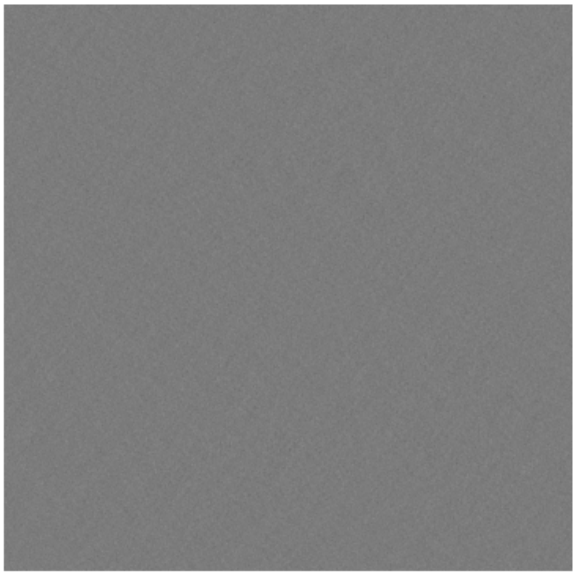




PCA 128 components

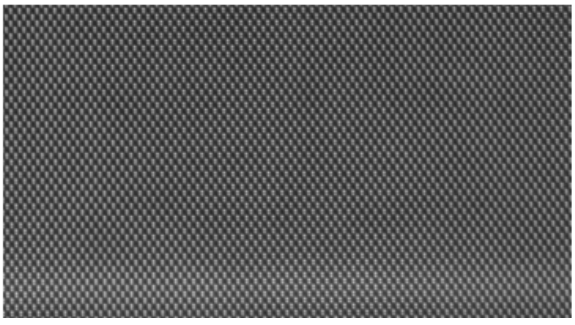
Residual

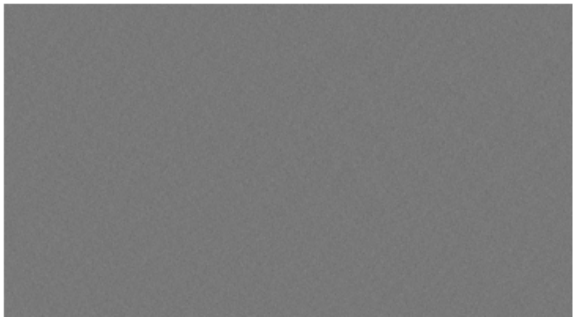


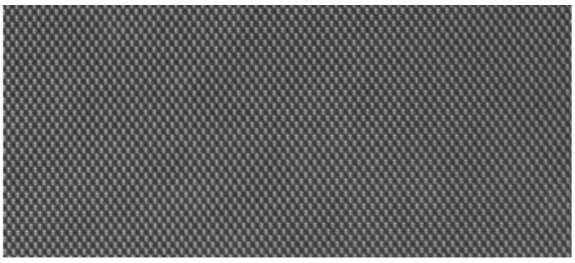


PCA 256 components

Residual







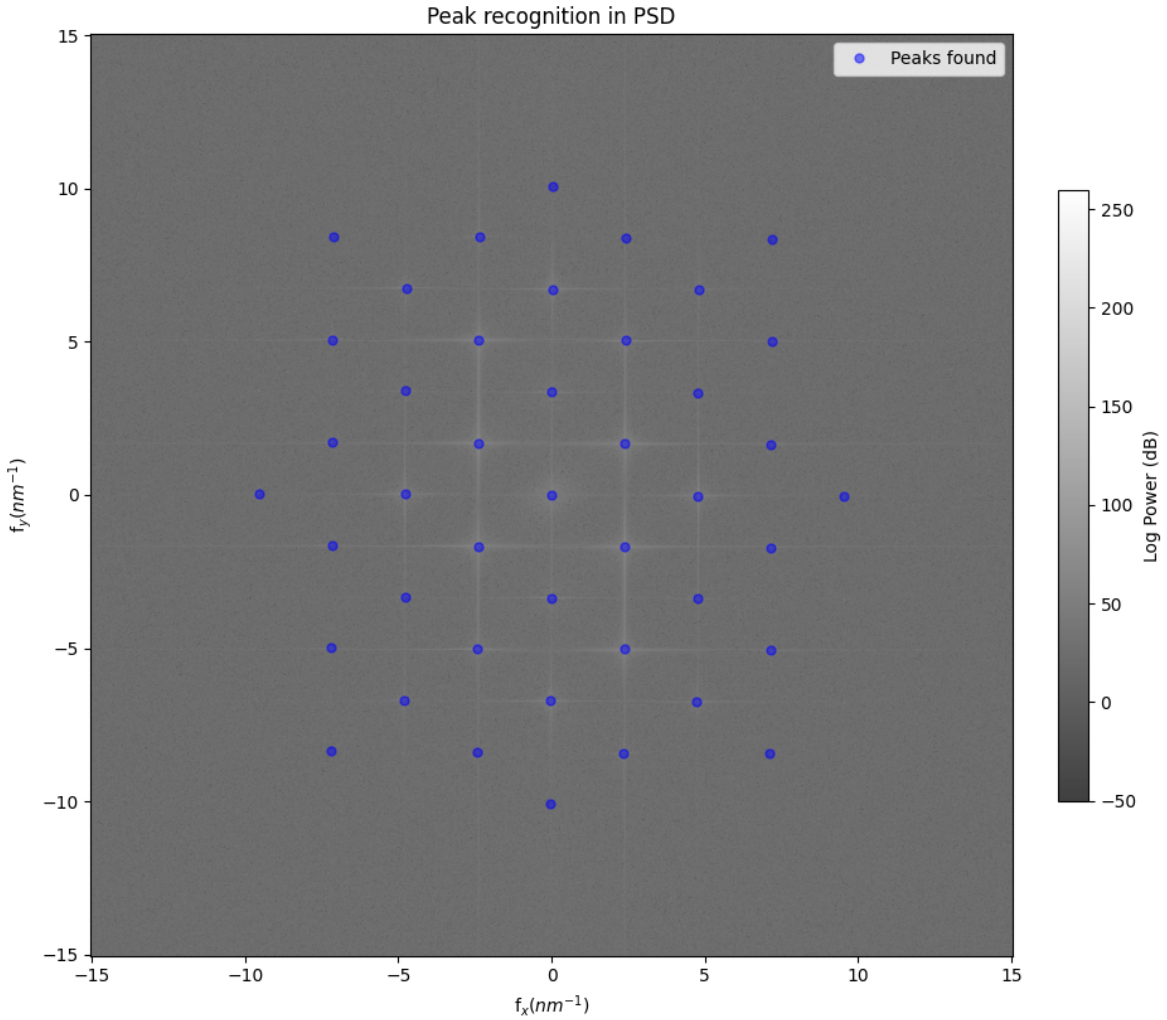

In [9]:
```
plt.savefig(rf"{path}\PCA_all.png", dpi=512, transparent=True, bbox_inches=
```

Diffraction Pattern Analysis

In [10]:
```
# Compute power spectral density
image_for_psd = original_imag[pixels_cropped:-pixels_cropped, pixels_croppe
psd2d, KX, KY = sha.compute_psd2d(image=image_for_psd, pixel_size=pixel_siz

kx_pixel_size = np.abs(KX[0,1] - KX[0,0])
g_max = 0.15 * KX.max()
r_int = sha.circular_mask(psd2d=psd2d, radius=g_max // kx_pixel_size)

# Find peaks and intensities in the diffraction pattern
refined_peaks = sha.find_refined_psd_peaks(
    psd2d, KX, KY,
    min_distance=40,
    threshold_abs=45,
    window_size=20,
    log_scale_find_pos=True
)
peak_avg_intensities = sha.calculate_avg_intensity_for_peaks(psd2d, refined

fig, ax = plt.subplots(figsize=(10, 10))
extent_X = KX.min()/(2*np.pi)*10, KX.max()/(2*np.pi)*10  # in nm^-1
extent_Y = KY.min()/(2*np.pi)*10, KY.max()/(2*np.pi)*10  # in nm^-1

im1 = ax.imshow(10 * np.log10(psd2d), extent=(extent_X[0], extent_X[1], ext
                origin="lower", cmap="gray", vmin=-50, vmax=260, alpha=0.75
ax.set_title("Peak recognition in PSD")
ax.set_xlim(0.2 * extent_X[0], 0.2 * extent_X[1])
ax.set_ylim(0.2 * extent_Y[0], 0.2 * extent_Y[1])

x_peaks, y_peaks = np.array(list(zip(*refined_peaks)))
x_peaks_nm, y_peaks_nm = np.array((x_peaks, y_peaks)) / (2 * np.pi) * 10
ax.plot(x_peaks_nm, y_peaks_nm, "bo", markersize=5, label="Peaks found", al

cbar = plt.colorbar(im1, ax=ax, pad=0.04, shrink=0.5)
cbar.set_label("Log Power (dB)")
ax.set_xlabel("f$_x (nm^{-1})$")
ax.set_ylabel("f$_y (nm^{-1})$")
```

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

Figure



In [11]:

```
plt.savefig(rf"{path}\PSD_peaks.png", dpi=512, transparent=True, bbox_inche

# Load crystal structure for diffraction simulation
file_path = sha.load_file()
structure = Structure.from_file(file_path)

# Simulate electron diffraction
reflections = sha.simulate_electron_diffraction(
```

```
    structure,
    g_max,
    zone_axis=np.array([0, 1, 1])
)

zone_axes_table = pd.DataFrame([
    {
        "g_x": r[0],
        "g_y": r[1],
        "distance_rad/A": np.linalg.norm((r[0], r[1])),
        "distance_A^-1": np.linalg.norm((r[0], r[1])) / (2 * np.pi),
        "label": r[2]
    }
    for r in reflections
])
display(zone_axes_table)
```

	g_x	g_y	distance_rad/A	distance_A^-1	label
0	7.455104e-16	6.556160e+00	6.556160	1.043445	(-6,0,0)
1	-1.545302e+00	5.463466e+00	5.677801	0.903650	(-5,-1,1)
2	1.545302e+00	5.463466e+00	5.677801	0.903650	(-5,1,-1)
3	-3.090603e+00	4.370773e+00	5.353082	0.851969	(-4,-2,2)
4	4.970069e-16	4.370773e+00	4.370773	0.695630	(-4,0,0)
5	3.090603e+00	4.370773e+00	5.353082	0.851969	(-4,2,-2)
6	-4.635905e+00	3.278080e+00	5.677801	0.903650	(-3,-3,3)
7	-1.545302e+00	3.278080e+00	3.624054	0.576786	(-3,-1,1)
8	1.545302e+00	3.278080e+00	3.624054	0.576786	(-3,1,-1)
9	4.635905e+00	3.278080e+00	5.677801	0.903650	(-3,3,-3)
10	-6.181207e+00	2.185387e+00	6.556160	1.043445	(-2,-4,4)
11	-3.090603e+00	2.185387e+00	3.785201	0.602433	(-2,-2,2)
12	2.485035e-16	2.185387e+00	2.185387	0.347815	(-2,0,0)
13	3.090603e+00	2.185387e+00	3.785201	0.602433	(-2,2,-2)
14	6.181207e+00	2.185387e+00	6.556160	1.043445	(-2,4,-4)
15	-4.635905e+00	1.092693e+00	4.762940	0.758045	(-1,-3,3)
16	-1.545302e+00	1.092693e+00	1.892600	0.301217	(-1,-1,1)
17	1.545302e+00	1.092693e+00	1.892600	0.301217	(-1,1,-1)
18	4.635905e+00	1.092693e+00	4.762940	0.758045	(-1,3,-3)

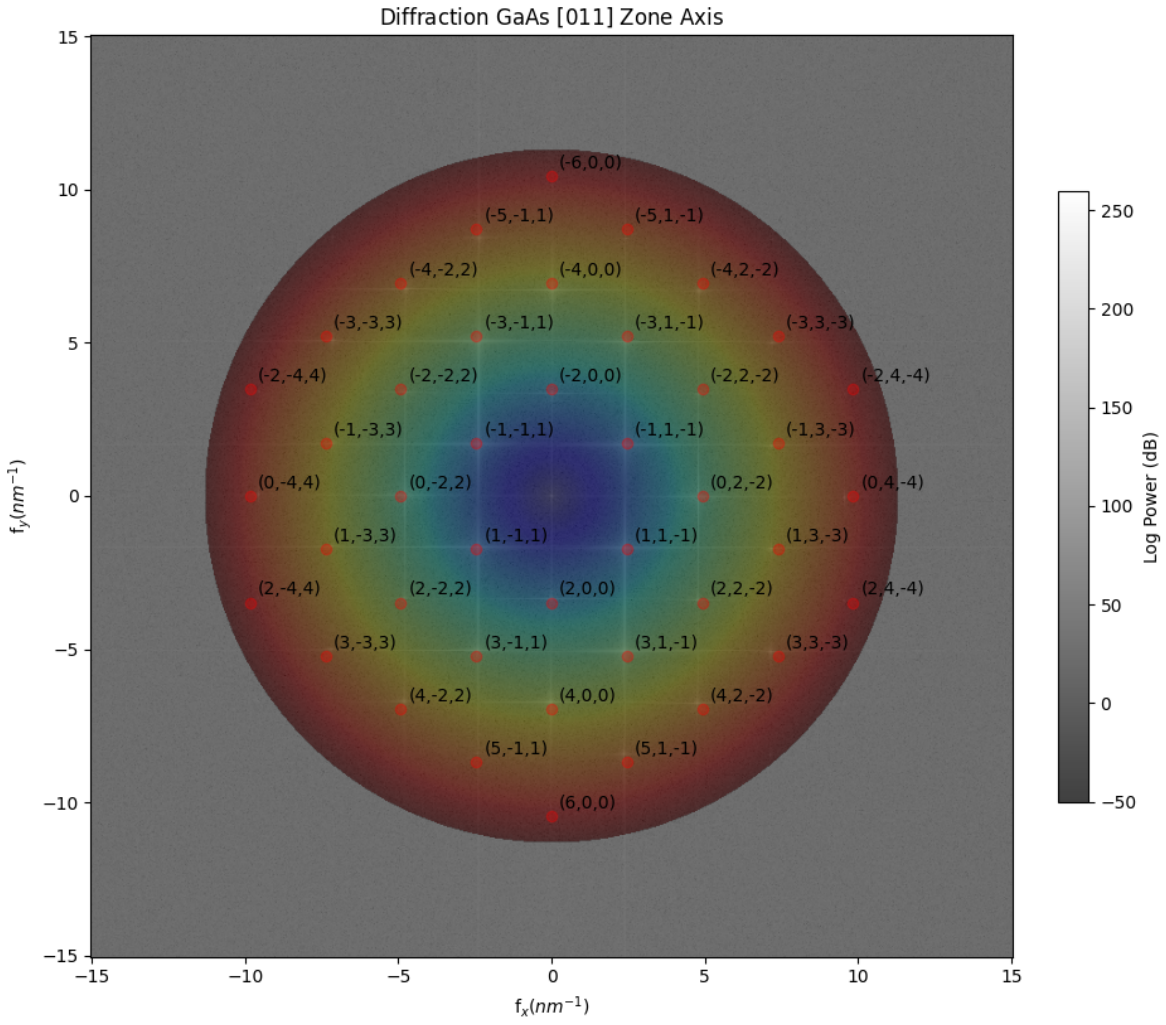
19	-6.181207e+00	2.676327e-16	6.181207	0.983770	(0,-4,4)
20	-3.090603e+00	1.338163e-16	3.090603	0.491885	(0,-2,2)
21	3.090603e+00	-1.338163e-16	3.090603	0.491885	(0,2,-2)
22	6.181207e+00	-2.676327e-16	6.181207	0.983770	(0,4,-4)
23	-4.635905e+00	-1.092693e+00	4.762940	0.758045	(1,-3,3)
24	-1.545302e+00	-1.092693e+00	1.892600	0.301217	(1,-1,1)
25	1.545302e+00	-1.092693e+00	1.892600	0.301217	(1,1,-1)
26	4.635905e+00	-1.092693e+00	4.762940	0.758045	(1,3,-3)
27	-6.181207e+00	-2.185387e+00	6.556160	1.043445	(2,-4,4)
28	-3.090603e+00	-2.185387e+00	3.785201	0.602433	(2,-2,2)
29	-2.485035e-16	-2.185387e+00	2.185387	0.347815	(2,0,0)
30	3.090603e+00	-2.185387e+00	3.785201	0.602433	(2,2,-2)
31	6.181207e+00	-2.185387e+00	6.556160	1.043445	(2,4,-4)
32	-4.635905e+00	-3.278080e+00	5.677801	0.903650	(3,-3,3)
33	-1.545302e+00	-3.278080e+00	3.624054	0.576786	(3,-1,1)
34	1.545302e+00	-3.278080e+00	3.624054	0.576786	(3,1,-1)
35	4.635905e+00	-3.278080e+00	5.677801	0.903650	(3,3,-3)
36	-3.090603e+00	-4.370773e+00	5.353082	0.851969	(4,-2,2)
37	-4.970069e-16	-4.370773e+00	4.370773	0.695630	(4,0,0)
38	3.090603e+00	-4.370773e+00	5.353082	0.851969	(4,2,-2)
39	-1.545302e+00	-5.463466e+00	5.677801	0.903650	(5,-1,1)
40	1.545302e+00	-5.463466e+00	5.677801	0.903650	(5,1,-1)
41	-7.455104e-16	-6.556160e+00	6.556160	1.043445	(6,0,0)

In [12]:

```
# Plot diffraction pattern with simulated spots
fig, ax = plt.subplots(figsize=(10, 10))
im2 = ax.imshow(r_int, extent=(extent_X[0], extent_X[1], extent_Y[0], exten
im1 = ax.imshow(10 * np.log10(psd2d), extent=(extent_X[0], extent_X[1], ext
                origin="lower", cmap="gray", vmin=-50, vmax=260, alpha=0.75
cbar = plt.colorbar(im1, ax=ax, pad=0.04, shrink=0.5)
cbar.set_label("Log Power (dB)")
ax.set_xlim(0.2 * extent_X[0], 0.2 * extent_X[1])
ax.set_ylim(0.2 * extent_Y[0], 0.2 * extent_Y[1])
```

```
for x, y, label in reflections:
    x_nm = x / (2 * np.pi) * 10
    y_nm = y / (2 * np.pi) * 10
    ax.plot(x_nm, y_nm, "ro", alpha=0.3)
    ax.text(x_nm + 0.25, y_nm + 0.25, label, fontsize=10)
ax.set_xlabel("f$_x (nm^{-1})$")
ax.set_ylabel("f$_y (nm^{-1})$")
plt.title(r"Diffraction GaAs $[011]$ Zone Axis")
ax.set_aspect("equal")
plt.tight_layout()
plt.show()
```

Figure



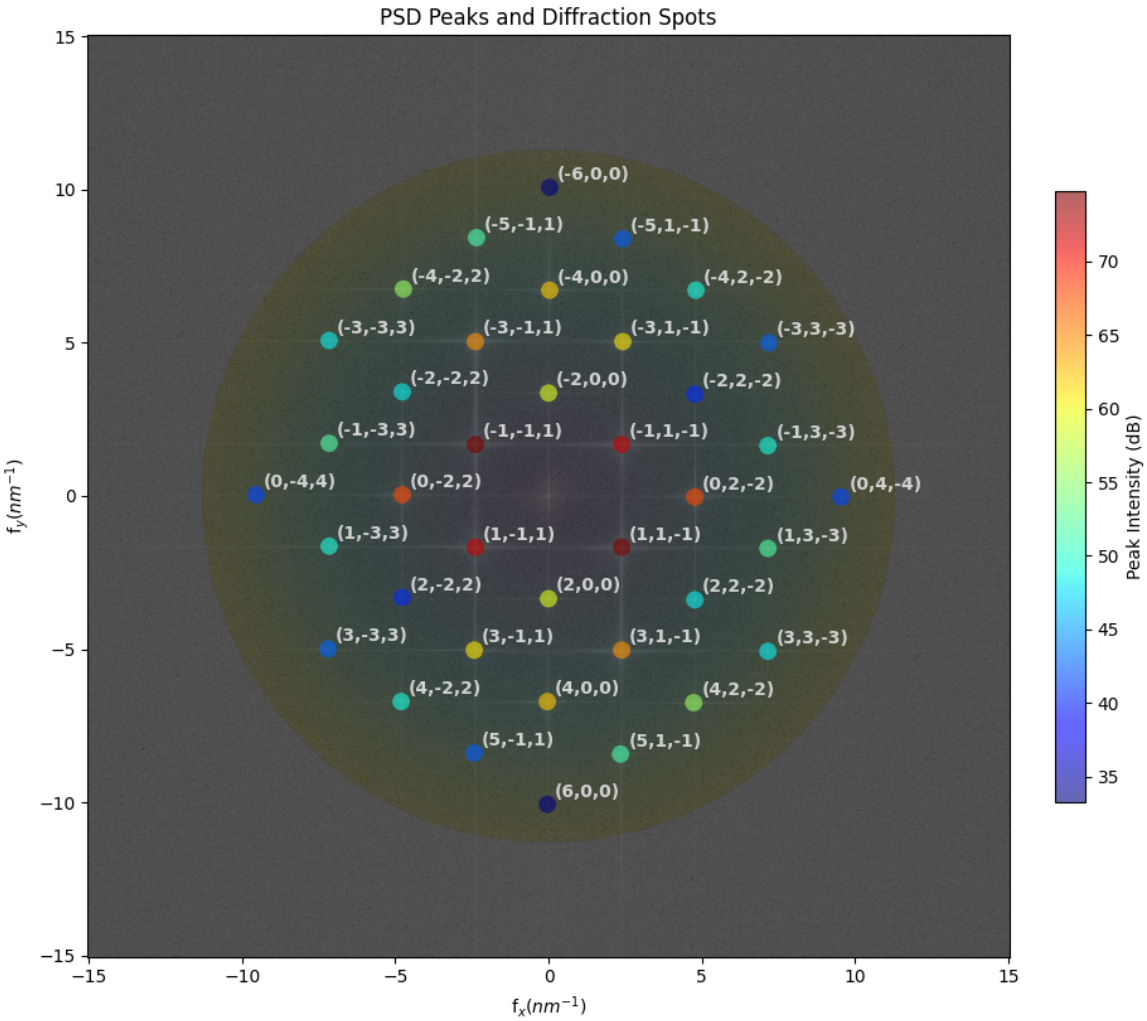
In [13]:
```
plt.savefig(rf"{path}\diffraction_GaAs.png", dpi=512, transparent=True, bbo

# Find best lattice constant
a_values = np.linspace(5.6, 6, 1000)
best_a, all_scores = sha.find_best_lattice_constant(
    structure, g_max, refined_peaks, a_values, tolerance=0.05
)
print("Best lattice constant:", best_a)
```

```
Best lattice constant: 5.919119119119119
```

In [14]:
```
# Match experimental and simulated peaks
experimental_reflections = [(gx, gy, intensity) for (gx, gy), intensity in
matches = sha.match_peaks(experimental_reflections, reflections, threshold=
matches_df = pd.DataFrame(matches)
```

In [15]:
```
# Create simplified matches dataframe and plot
matches_df_simple = matches_df[["hkl", "intensity"]].copy()
matches_df_simple["f_x"] = matches_df["exp_kx"] / (2 * np.pi) * 10
matches_df_simple["f_y"] = matches_df["exp_ky"] / (2 * np.pi) * 10
matches_df_simple["distance"] = np.linalg.norm(matches_df_simple[["f_x", "f
matches_df_simple["intensity_db"] = 10 * np.log10(matches_df_simple["intens

fig, ax = plt.subplots(figsize=(10, 10))
im2 = ax.imshow(r_int, extent=(extent_X[0], extent_X[1], extent_Y[0], exten
im = ax.imshow(10 * np.log10(psd2d), extent=(extent_X[0], extent_X[1], exte
               origin="lower", cmap="gray", vmin=-50, vmax=260, alpha=0.9)
sc = ax.scatter(matches_df_simple["f_x"], matches_df_simple["f_y"], c=match
               cmap="jet", s=100, edgecolor="None", label="PSD Peaks", alph

# Add hkl labels
for _, row in matches_df_simple.iterrows():
    ax.text(row["f_x"] + 0.25, row["f_y"] + 0.25, row["hkl"], fontsize=10,

cbar0 = fig.colorbar(sc, ax=ax, pad=0.04, shrink=0.5)
cbar0.set_label("Peak Intensity (dB)")

ax.set_xlabel("f$_x (nm^{-1})$")
ax.set_ylabel("f$_y (nm^{-1})$")
ax.set_title("PSD Peaks and Diffraction Spots")
ax.set_xlim(0.2 * extent_X[0], 0.2 * extent_X[1])
ax.set_ylim(0.2 * extent_Y[0], 0.2 * extent_Y[1])
ax.set_aspect("equal")
plt.tight_layout()
plt.show()
```

Figure



In [16]:

```
plt.savefig(rf"{path}\diffraction_GaAs_with_intensity.png", dpi=512, transp

# Group peaks by family
grouped, group_info = sha.group_peaks_by_family(matches_df_simple)
group_info_df_original = pd.DataFrame(group_info)
group_info_df_original = group_info_df_original.sort_values('avg_distance')
display(group_info_df_original)
```

	family	count	avg_distance	avg_intensity_db	hkl_list
0	(1, 1, 1)	4	2.917013	73.517237	[(1,1,-1), (-1,-1,1), (1,-1,1), (-1,1,-1)]
1	(0, 0, 2)	2	3.355597	59.088478	[(2,0,0), (-2,0,0)]
2	(0, 2, 2)	2	4.772550	67.914263	[(0,2,-2), (0,-2,2)]
3	(1, 1, 3)	4	5.578407	62.764957	[(3,1,-1), (-3,-1,1), (3,-1,1), (-3,1,-1)]
4	(2, 2, 2)	4	5.834107	44.295663	[(2,2,-2), (-2,-2,2), (2,-2,2), (-2,2,-2)]
5	(0, 0, 4)	2	6.711297	62.252478	[(4,0,0), (-4,0,0)]
6	(1, 3, 3)	4	7.352684	50.602344	[(1,3,-3), (-1,-3,3), (1,-3,3), (-1,3,-3)]
7	(2, 2, 4)	4	8.250347	52.250995	[(4,2,-2), (-4,-2,2), (4,-2,2), (-4,2,-2)]
8	(1, 1, 5)	4	8.739791	46.907077	[(5,1,-1), (-5,-1,1), (5,-1,1), (-5,1,-1)]
9	(3, 3, 3)	4	8.766394	45.480113	[(3,3,-3), (-3,-3,3), (3,-3,3), (-3,3,-3)]
10	(0, 4, 4)	2	9.544881	41.327297	[(0,4,-4), (0,-4,4)]
11	(0, 0, 6)	2	10.066859	33.301780	[(6,0,0), (-6,0,0)]

In [17]:

```
# Calculate matches analysis grouped by family for all PCA results
pca_family_results = sha.generate_pca_matches_analysis(pca_list, pca_result

for n, group_info_df in pca_family_results.items():
    print(f"\nPCA {n} components - Peak families:")
    display(group_info_df)

all_family_results_df = pd.DataFrame()
for n in pca_list:
    all_family_results_df = pd.concat([all_family_results_df, pca_family_re

# Save results
all_family_results_df.to_excel(path + "\\all_pca_families.xlsx", index=Fals
```

```
Processing PCA components:   0%|          | 0/6 [00:00<?, ?it/s]
PCA 8 components - Peak families:
```

	family	count	avg_distance	avg_intensity_db	hkl_list	pca_components
0	(1, 1, 1)	4	2.917013	73.524144	[(1,1,-1), (-1,-1,1), (1,-1,1), (-1,1,-1)]	8
1	(0, 0, 2)	2	3.355597	58.900494	[(2,0,0), (-2,0,0)]	8
2	(0, 2, 2)	2	4.772550	67.904662	[(0,2,-2), (0,-2,2)]	8
3	(1, 1, 3)	4	5.578407	62.266035	[(3,1,-1), (-3,-1,1), (3,-1,1), (-3,1,-1)]	8
4	(2, 2, 2)	4	5.834107	43.735905	[(2,2,-2), (-2,-2,2), (2,-2,2), (-2,2,-2)]	8
5	(0, 0, 4)	2	6.711297	61.855236	[(4,0,0), (-4,0,0)]	8
6	(1, 3, 3)	4	7.352684	45.355243	[(1,3,-3), (-1,-3,3), (1,-3,3), (-1,3,-3)]	8
7	(2, 2, 4)	4	8.250347	46.653888	[(4,2,-2), (-4,-2,2), (4,-2,2), (-4,2,-2)]	8
8	(1, 1, 5)	4	8.739791	43.596077	[(5,1,-1), (-5,-1,1), (5,-1,1), (-5,1,-1)]	8
9	(3, 3, 3)	4	8.766394	35.654965	[(3,3,-3), (-3,-3,3), (3,-3,3), (-3,3,-3)]	8
10	(0, 4, 4)	2	9.544881	37.393419	[(0,4,-4), (0,-4,4)]	8
11	(0, 0, 6)	2	10.066859	29.107507	[(6,0,0), (-6,0,0)]	8

PCA 16 components — Peak families:

	family	count	avg_distance	avg_intensity_db	hkl_list	pca_components
0	(1, 1, 1)	4	2.917013	73.522726	[(1,1,-1), (-1,-1,1), (1,-1,1), (-1,1,-1)]	16
1	(0, 0, 2)	2	3.355597	59.097066	[(2,0,0), (-2,0,0)]	16
2	(0, 2, 2)	2	4.772550	67.910345	[(0,2,-2), (0,-2,2)]	16
3	(1, 1, 3)	4	5.578407	62.667073	[(3,1,-1), (-3,-1,1), (3,-1,1), (-3,1,-1)]	16
4	(2, 2, 2)	4	5.834107	44.160198	[(2,2,-2), (-2,-2,2), (2,-2,2), (-2,2,-2)]	16
5	(0, 0, 4)	2	6.711297	62.213782	[(4,0,0), (-4,0,0)]	16
6	(1, 3, 3)	4	7.352684	49.448797	[(1,3,-3), (-1,-3,3), (1,-3,3), (-1,3,-3)]	16
7	(2, 2, 4)	4	8.250347	50.849913	[(4,2,-2), (-4,-2,2), (4,-2,2), (-4,2,-2)]	16
8	(1, 1, 5)	4	8.739791	46.058116	[(5,1,-1), (-5,-1,1), (5,-1,1), (-5,1,-1)]	16
9	(3, 3, 3)	4	8.766394	42.943106	[(3,3,-3), (-3,-3,3), (3,-3,3), (-3,3,-3)]	16
10	(0, 4, 4)	2	9.544881	37.729094	[(0,4,-4), (0,-4,4)]	16
11	(0, 0, 6)	2	10.066859	31.151282	[(6,0,0), (-6,0,0)]	16

PCA 32 components — Peak families:

	family	count	avg_distance	avg_intensity_db	hkl_list	pca_components
0	(1, 1, 1)	4	2.917013	73.517008	[(1,1,-1), (-1,-1,1), (1,-1,1), (-1,1,-1)]	32
1	(0, 0, 2)	2	3.355597	59.089050	[(2,0,0), (-2,0,0)]	32
2	(0, 2, 2)	2	4.772550	67.910626	[(0,2,-2), (0,-2,2)]	32
3	(1, 1, 3)	4	5.578407	62.744218	[(3,1,-1), (-3,-1,1), (3,-1,1), (-3,1,-1)]	32
4	(2, 2, 2)	4	5.834107	44.162804	[(2,2,-2), (-2,-2,2), (2,-2,2), (-2,2,-2)]	32
5	(0, 0, 4)	2	6.711297	62.231826	[(4,0,0), (-4,0,0)]	32
6	(1, 3, 3)	4	7.352684	50.493464	[(1,3,-3), (-1,-3,3), (1,-3,3), (-1,3,-3)]	32
7	(2, 2, 4)	4	8.250347	52.032393	[(4,2,-2), (-4,-2,2), (4,-2,2), (-4,2,-2)]	32
8	(1, 1, 5)	4	8.739791	46.703059	[(5,1,-1), (-5,-1,1), (5,-1,1), (-5,1,-1)]	32
9	(3, 3, 3)	4	8.766394	44.705282	[(3,3,-3), (-3,-3,3), (3,-3,3), (-3,3,-3)]	32
10	(0, 4, 4)	2	9.544881	38.849466	[(0,4,-4), (0,-4,4)]	32
11	(0, 0, 6)	2	10.066859	31.995553	[(6,0,0), (-6,0,0)]	32

PCA 64 components — Peak families:

	family	count	avg_distance	avg_intensity_db	hkl_list	pca_components
0	(1, 1, 1)	4	2.917013	73.517035	[(1,1,-1), (-1,-1,1), (1,-1,1), (-1,1,-1)]	64
1	(0, 0, 2)	2	3.355597	59.083180	[(2,0,0), (-2,0,0)]	64
2	(0, 2, 2)	2	4.772550	67.911375	[(0,2,-2), (0,-2,2)]	64
3	(1, 1, 3)	4	5.578407	62.751441	[(3,1,-1), (-3,-1,1), (3,-1,1), (-3,1,-1)]	64
4	(2, 2, 2)	4	5.834107	44.187088	[(2,2,-2), (-2,-2,2), (2,-2,2), (-2,2,-2)]	64
5	(0, 0, 4)	2	6.711297	62.240781	[(4,0,0), (-4,0,0)]	64
6	(1, 3, 3)	4	7.352684	50.530308	[(1,3,-3), (-1,-3,3), (1,-3,3), (-1,3,-3)]	64
7	(2, 2, 4)	4	8.250347	52.111937	[(4,2,-2), (-4,-2,2), (4,-2,2), (-4,2,-2)]	64
8	(1, 1, 5)	4	8.739791	46.761775	[(5,1,-1), (-5,-1,1), (5,-1,1), (-5,1,-1)]	64
9	(3, 3, 3)	4	8.766394	44.889041	[(3,3,-3), (-3,-3,3), (3,-3,3), (-3,3,-3)]	64
10	(0, 4, 4)	2	9.544881	39.541593	[(0,4,-4), (0,-4,4)]	64
11	(0, 0, 6)	2	10.066859	32.709268	[(6,0,0), (-6,0,0)]	64

```
PCA 128 components — Peak families:
```

	family	count	avg_distance	avg_intensity_db	hkl_list	pca_components
0	(1, 1, 1)	4	2.917013	73.517071	[(1,1,-1), (-1,-1,1), (1,-1,1), (-1,1,-1)]	128
1	(0, 0, 2)	2	3.355597	59.086501	[(2,0,0), (-2,0,0)]	128
2	(0, 2, 2)	2	4.772550	67.911991	[(0,2,-2), (0,-2,2)]	128
3	(1, 1, 3)	4	5.578407	62.754435	[(3,1,-1), (-3,-1,1), (3,-1,1), (-3,1,-1)]	128
4	(2, 2, 2)	4	5.834107	44.202949	[(2,2,-2), (-2,-2,2), (2,-2,2), (-2,2,-2)]	128
5	(0, 0, 4)	2	6.711297	62.245623	[(4,0,0), (-4,0,0)]	128
6	(1, 3, 3)	4	7.352684	50.547198	[(1,3,-3), (-1,-3,3), (1,-3,3), (-1,3,-3)]	128
7	(2, 2, 4)	4	8.250347	52.141920	[(4,2,-2), (-4,-2,2), (4,-2,2), (-4,2,-2)]	128
8	(1, 1, 5)	4	8.739791	46.799683	[(5,1,-1), (-5,-1,1), (5,-1,1), (-5,1,-1)]	128
9	(3, 3, 3)	4	8.766394	45.004788	[(3,3,-3), (-3,-3,3), (3,-3,3), (-3,3,-3)]	128
10	(0, 4, 4)	2	9.544881	40.145407	[(0,4,-4), (0,-4,4)]	128
11	(0, 0, 6)	2	10.066859	32.993970	[(6,0,0), (-6,0,0)]	128

PCA 256 components — Peak families:

	family	count	avg_distance	avg_intensity_db	hkl_list	pca_components
0	(1, 1, 1)	4	2.917013	73.517086	[(1,1,-1), (-1,-1,1), (1,-1,1), (-1,1,-1)]	256
1	(0, 0, 2)	2	3.355597	59.087154	[(2,0,0), (-2,0,0)]	256
2	(0, 2, 2)	2	4.772550	67.912945	[(0,2,-2), (0,-2,2)]	256
3	(1, 1, 3)	4	5.578407	62.758646	[(3,1,-1), (-3,-1,1), (3,-1,1), (-3,1,-1)]	256
4	(2, 2, 2)	4	5.834107	44.217456	[(2,2,-2), (-2,-2,2), (2,-2,2), (-2,2,-2)]	256
5	(0, 0, 4)	2	6.711297	62.248058	[(4,0,0), (-4,0,0)]	256
6	(1, 3, 3)	4	7.352684	50.568697	[(1,3,-3), (-1,-3,3), (1,-3,3), (-1,3,-3)]	256
7	(2, 2, 4)	4	8.250347	52.180353	[(4,2,-2), (-4,-2,2), (4,-2,2), (-4,2,-2)]	256
8	(1, 1, 5)	4	8.739791	46.841037	[(5,1,-1), (-5,-1,1), (5,-1,1), (-5,1,-1)]	256
9	(3, 3, 3)	4	8.766394	45.148742	[(3,3,-3), (-3,-3,3), (3,-3,3), (-3,3,-3)]	256
10	(0, 4, 4)	2	9.544881	40.608797	[(0,4,-4), (0,-4,4)]	256
11	(0, 0, 6)	2	10.066859	33.024213	[(6,0,0), (-6,0,0)]	256

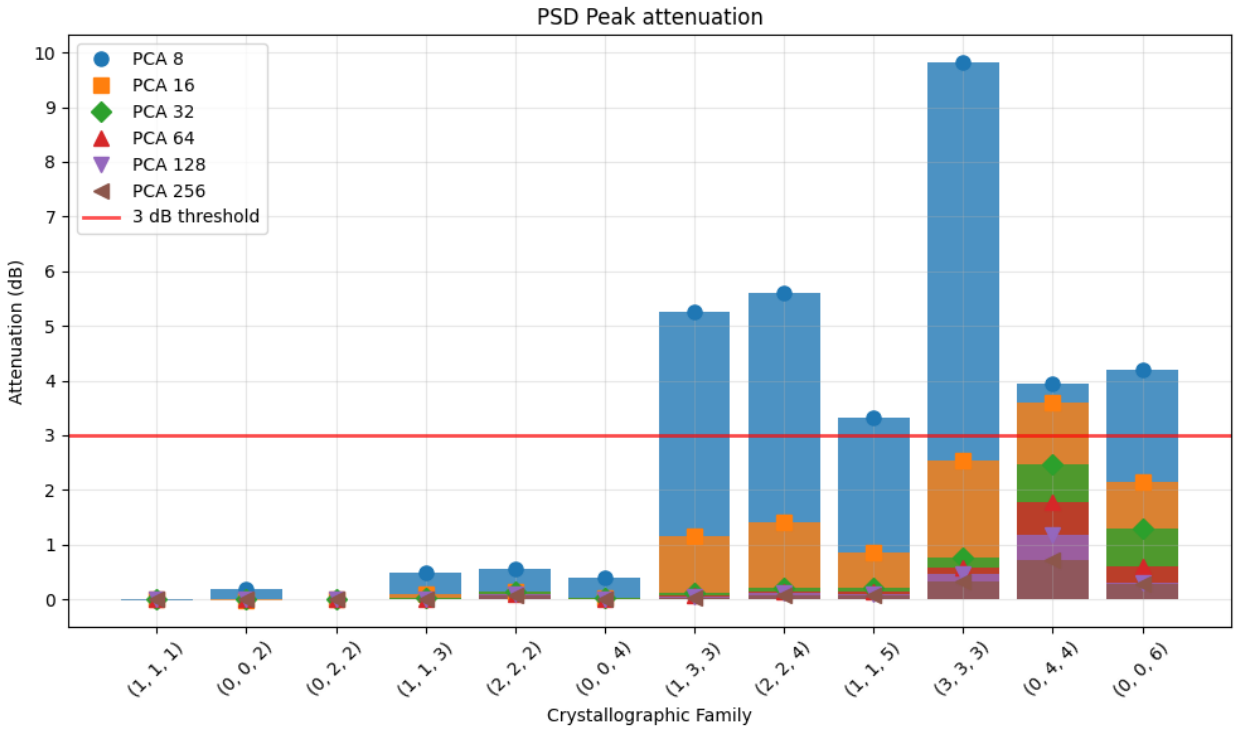
In [18]:
```
# Calculate attenuations for each PCA component
```

```
pca_attenuation_results = {}
all_attenuation_df = pd.DataFrame()

for n in pca_list:
    # Get the PCA family results
    pca_families = pca_family_results[n].copy()

    # Calculate attenuation by merging with original families
    attenuation_df = pca_families.merge(group_info_df_original, on='family'
    attenuation_df['attenuation_db'] = attenuation_df['avg_intensity_db_ori
    attenuation_df['pca_components'] = n

    # Store results
    pca_attenuation_results[n] = attenuation_df
    all_attenuation_df = pd.concat([all_attenuation_df, attenuation_df[['fa

all_attenuation_df.to_excel(path + "\\all_pca_attenuations_by_family.xlsx",
```

In [19]:

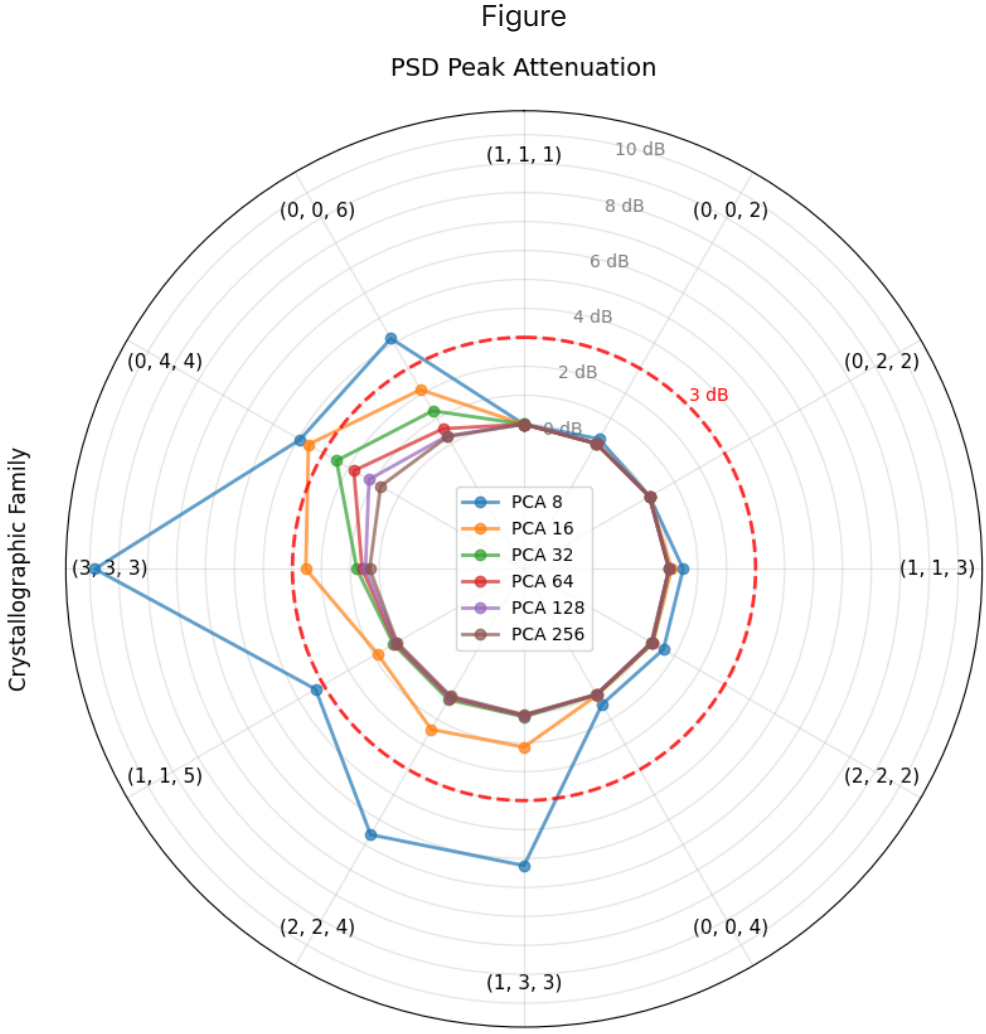
```
# Plot peak family attenuations across PCA components
plt.figure(figsize=(10, 6))

# Get unique families and PCA components
families = all_attenuation_df['family'].unique()
pca_components = all_attenuation_df['pca_components'].unique()

# Plot each PCA component as a separate line
for i, pca in enumerate(pca_components):
    pca_data = all_attenuation_df[all_attenuation_df['pca_components'] == p

    # marker shapes
    marker_shapes = ['o', 's', 'D', '^', 'v', '<', '>', 'p', '*', 'h', 'H',
    marker = marker_shapes[i % len(marker_shapes)]
    line, = plt.plot(pca_data['family'], pca_data['attenuation_db'],
             linestyle='', marker=marker, label=f'PCA {pca}',
             markersize=8)
    plt.bar(pca_data['family'].astype(str), pca_data['attenuation_db'],
            color=line.get_color(), alpha=0.8)

# Add horizontal line at 3 dB
plt.axhline(y=3, color='red', linestyle='-', alpha=0.7, linewidth=2, label=

plt.xlabel('Crystallographic Family')
plt.ylabel('Attenuation (dB)')
plt.title('PSD Peak attenuation')
plt.legend(loc='upper left')
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.yticks(np.arange(0, np.ceil(all_attenuation_df['attenuation_db'].max())
plt.tight_layout()
plt.show()
```

Figure



In [20]:

```
plt.savefig(rf"{path}\psd_peaks_attenuation_bar_chart.png", dpi=512, transp

fig, ax = sha.create_polar_radar_plot(all_attenuation_df, fill_polygons=Fal
fig.savefig(rf"{path}\psd_peaks_attenuation_radar_chart.png", dpi=512, tran
```



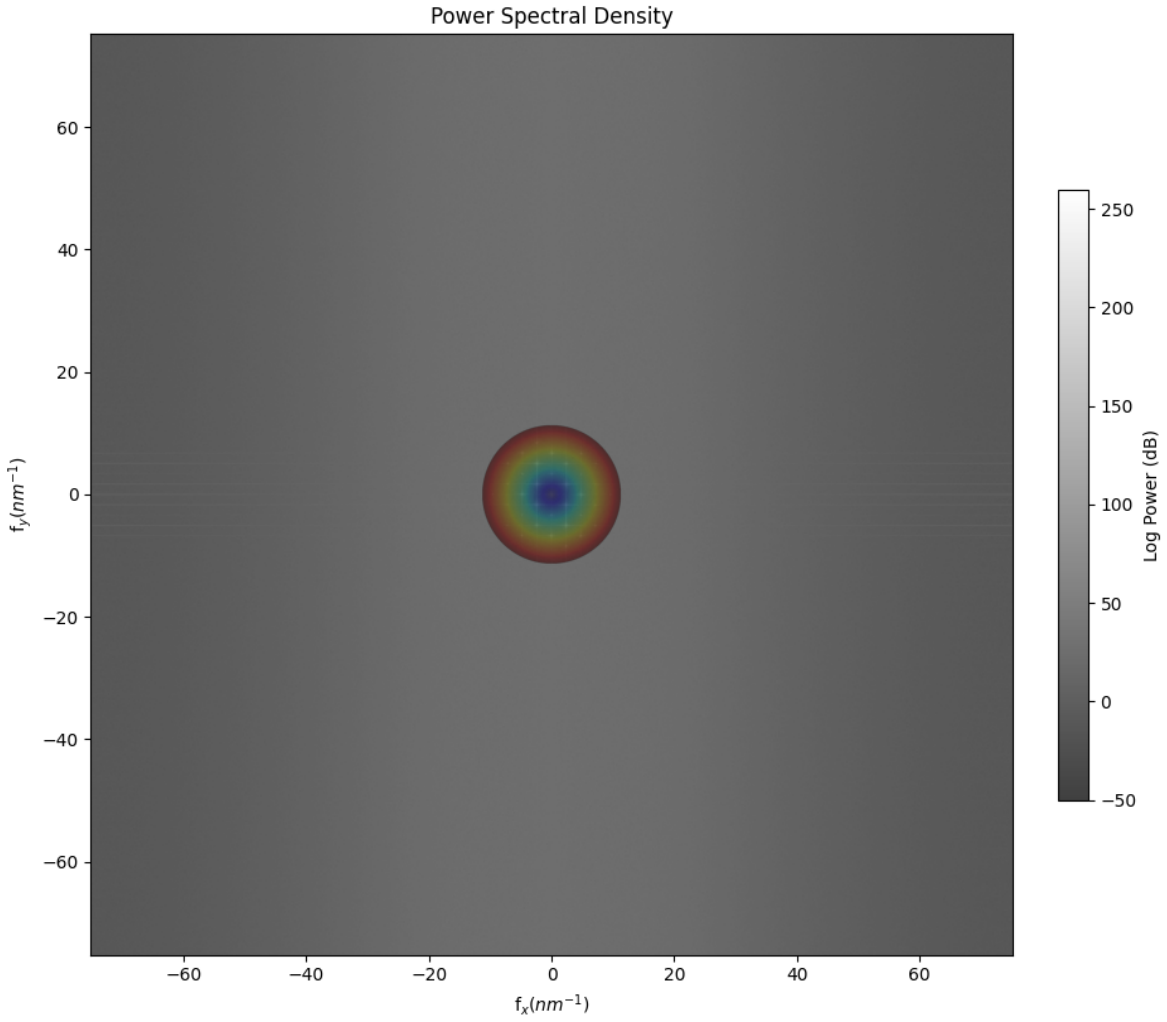
Radial profiles of PSD

In [21]:

```
# Display power spectral density
no_filter_image = original_imag[pixels_cropped:-pixels_cropped, pixels_crop
psd2d, KX, KY = sha.compute_psd2d(image=no_filter_image, pixel_size=pixel_s

kx_pixel_size = np.abs(KX[0,1] - KX[0,0])
g_max = 0.15 * KX.max()
r_int = sha.circular_mask(psd2d=psd2d, radius=g_max // kx_pixel_size)

fig, ax = plt.subplots(figsize=(10, 10))
extent_X = KX.min()/(2*np.pi)*10, KX.max()/(2*np.pi)*10  # in nm^-1
extent_Y = KY.min()/(2*np.pi)*10, KY.max()/(2*np.pi)*10  # in nm^-1
im2 = ax.imshow(r_int, extent=(extent_X[0], extent_X[1], extent_Y[0], exten
im1 = ax.imshow(10 * np.log10(psd2d), extent=(extent_X[0], extent_X[1], ext
                origin="lower", cmap="gray", vmin=-50, vmax=260, alpha=0.75

cbar = plt.colorbar(im1, ax=ax, pad=0.04, shrink=0.5)
```

```
cbar.set_label("Log Power (dB)")
#ax.set_xlim(0.2 * extent_X[0], 0.2 * extent_X[1])
#ax.set_ylim(0.2 * extent_Y[0], 0.2 * extent_Y[1])
ax.set_xlabel("f$_x (nm^{-1})$")
ax.set_ylabel("f$_y (nm^{-1})$")
plt.title("Power Spectral Density")
ax.set_aspect("equal")
plt.tight_layout()
plt.show()
```

Figure



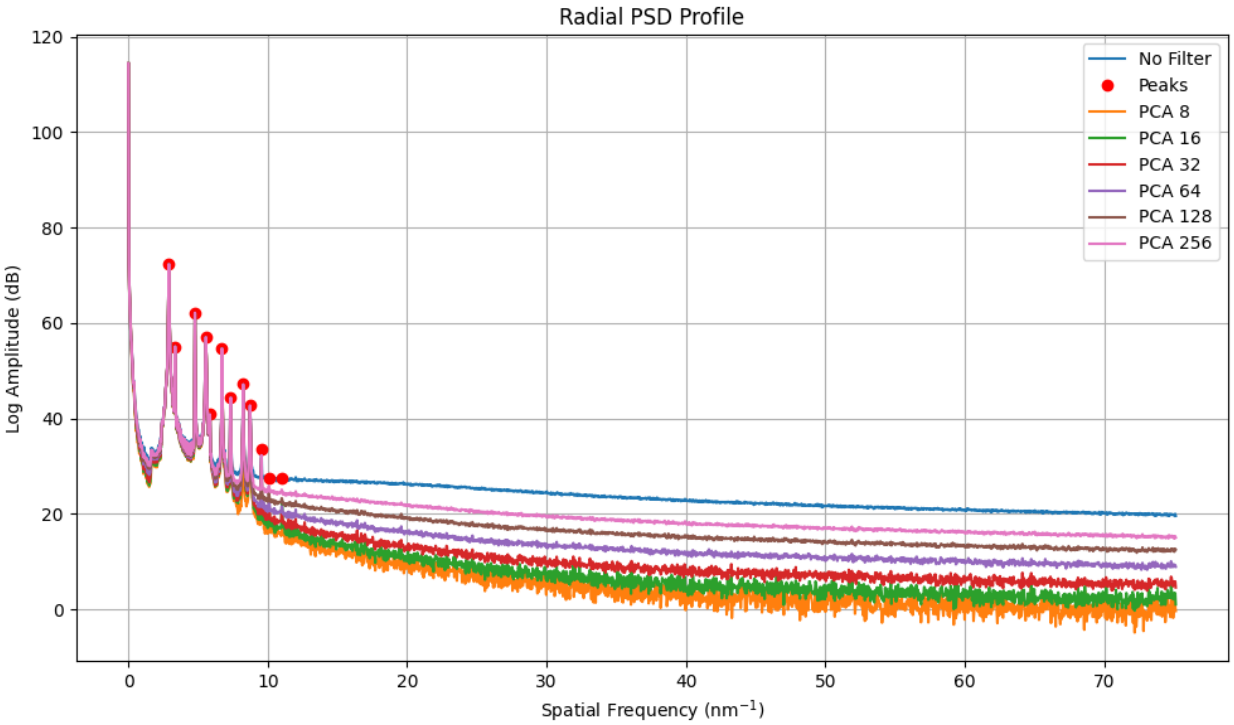
In [22]:

```
plt.savefig(rf"{path}\PSD.png", dpi=512, transparent=True, bbox_inches="tig
```

```
# Compute radial PSD profiles and analyze signal/noise

## We use PCA 16 to get all the peak positions
_, _, radial_psd_db = sha.compute_radial_psd(
    image=pca_results[16]["pca_image"],
    pixel_size=pixel_size_pm / 1000
)
peaks, _ = find_peaks(radial_psd_db, height=15, distance=8, prominence=2.8)

## Then we use these peak positions to analyze all images
radial_psd, spatial_freq, radial_psd_db = sha.compute_radial_psd(
    image=no_filter_image,
    pixel_size=pixel_size_pm / 1000
)
signal_power, noise_power = sha.compute_signal_noise_power(
    image=no_filter_image,
    information_limit=g_max // kx_pixel_size
)

signal_noise = {
    "Type": ["No Filter"] + [f"PCA {n}" for n in pca_list],
    "SignalPSD": [],
    "NoisePSD": []
}
signal_noise["SignalPSD"].append(signal_power)
signal_noise["NoisePSD"].append(noise_power)

plt.figure(figsize=(10, 6))
plt.plot(spatial_freq, radial_psd_db, label="No Filter")
plt.plot(spatial_freq[peaks], radial_psd_db[peaks], "ro", label="Peaks")

peak_table = {"Spatial Frequency (nm^-1)": spatial_freq[peaks]}
peak_table["No Filter"] = [radial_psd_db[p] for p in peaks]

for n in pca_list:
    image = pca_results[n]["pca_image"]
    radial_psd, spatial_freq, radial_psd_db = sha.compute_radial_psd(
        image=image,
        pixel_size=pixel_size_pm / 1000
    )
    signal_power, noise_power = sha.compute_signal_noise_power(
        image=image,
        information_limit=g_max // kx_pixel_size
    )
    signal_noise["SignalPSD"].append(signal_power)
    signal_noise["NoisePSD"].append(noise_power)

    plt.plot(spatial_freq, radial_psd_db, label=f"PCA {n}")
    peak_table[f"PCA {n}"] = [radial_psd_db[p] for p in peaks]

signal_noise = pd.DataFrame(signal_noise)
plt.title("Radial PSD Profile")
plt.xlabel("Spatial Frequency (nm$^{-1}$)")
```

```
plt.ylabel("Log Amplitude (dB)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Figure



In [23]:

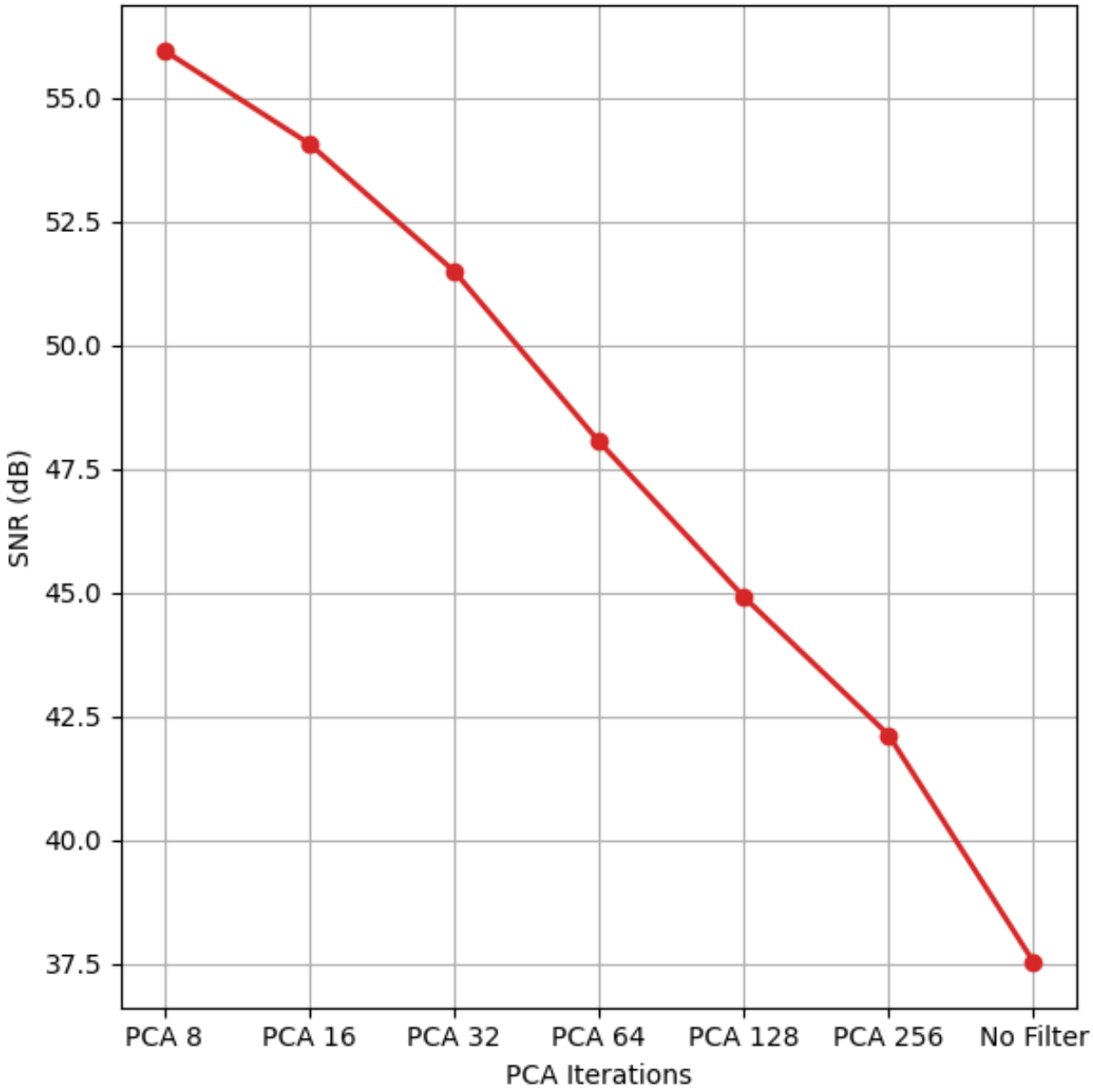
```
plt.savefig(rf"{path}\radial_psd.png", dpi=512, transparent=True, bbox_inch

# Display signal-to-noise analysis results
signal_noise["SNR"] = signal_noise["SignalPSD"] - signal_noise["NoisePSD"]
display(signal_noise)

df_peaks = pd.DataFrame(peak_table)
df_peaks_diff = df_peaks.copy()
for col in df_peaks.columns[1:]:
    if col != "No Filter":
        df_peaks_diff[col] = df_peaks["No Filter"] - df_peaks[col]
        df_peaks_diff.rename(columns={col: col + " PSD peak attenuation"},
```

	Type	SignalPSD	NoisePSD	SNR
0	No Filter	59.969941	22.423455	37.546486
1	PCA 8	59.962757	4.018230	55.944527
2	PCA 16	59.966461	5.900012	54.066449
3	PCA 32	59.967220	8.473831	51.493389
4	PCA 64	59.967635	11.914086	48.053549
5	PCA 128	59.968093	15.051617	44.916477
6	PCA 256	59.968692	17.841876	42.126816

In [24]:

```
# Plot signal-to-noise ratio analysis
plt.figure(figsize=(6, 6))
plt.plot(np.roll(signal_noise['Type'].values, shift=-1), np.roll(signal_noi
snr_vals = signal_noise['SNR'].values
plt.xlabel('PCA Iterations')
plt.ylabel('SNR (dB)')
plt.grid(True, which='both')
plt.tight_layout()
plt.show()
plt.savefig(rf"{path}\snr_pca.png", dpi=512, transparent=True, bbox_inches=
```

Figure



In [25]:

```
display(df_peaks_diff)
# Save results to Excel
df_peaks_diff.to_excel(path + "\\psd_peaks.xlsx", index=False)

# Replace first column of df_peaks_diff with family names from group_info_d
df_peaks_diff['Spatial Frequency (nm^-1)'] = group_info_df_original['family

# Reshape df_peaks_diff for analysis
df_peaks_melted = df_peaks_diff.melt(
    id_vars=['Spatial Frequency (nm^-1)'],
    var_name='PCA_Component',
    value_name='Attenuation'
)
```

```
# Extract PCA component numbers from column names (filter out non-PCA colum
df_peaks_melted = df_peaks_melted[df_peaks_melted['PCA_Component'].str.cont
df_peaks_melted['PCA_Components'] = df_peaks_melted['PCA_Component'].str.ex

# Create the reshaped dataframe with 3 main columns
df_peaks_reshaped = df_peaks_melted[['Spatial Frequency (nm^-1)', 'Attenuat
df_peaks_reshaped.columns = ['family', 'attenuation_db', 'pca_components']

display(df_peaks_reshaped)
```

	Spatial Frequency (nm^-1)	No Filter	PCA 8 PSD peak attenuation	PCA 16 PSD peak attenuation	PCA 32 PSD peak attenuation	PCA 64 PSD peak attenuation	PCA 12 PSD pea attenuati
0	2.908184	72.283235	-0.005479	-0.007573	-0.000395	0.000013	0.00005
1	3.355597	55.036248	0.233390	0.029290	0.021297	0.018622	0.01183
2	4.772404	62.084164	0.001745	0.003698	0.001653	0.001015	0.00082
3	5.555377	57.011328	0.835496	0.133356	0.024341	0.012629	0.01108
4	5.853652	40.941033	0.719296	0.478919	0.326067	0.254641	0.21925
5	6.711193	54.630367	0.424126	-0.002059	0.010207	0.008140	0.00588
6	7.345028	44.406806	5.860409	1.682882	0.191037	0.135108	0.1133
7	8.239854	47.138394	6.523985	0.510247	0.118979	0.094061	0.07214
8	8.724551	42.898388	5.458823	1.567414	0.549473	0.405969	0.31848
9	9.544808	33.419725	4.866360	4.440436	3.200595	2.466357	1.79088
10	10.104074	27.589422	8.045043	6.700308	5.871040	5.018570	3.77217
11	11.036184	27.432037	10.223811	5.081330	7.690873	6.316276	4.50263

	family	attenuation_db	pca_components
12	(1, 1, 1)	-0.005479	8
13	(0, 0, 2)	0.233390	8
14	(0, 2, 2)	0.001745	8
15	(1, 1, 3)	0.835496	8
16	(2, 2, 2)	0.719296	8
...	...	...	...
79	(2, 2, 4)	0.044485	256
80	(1, 1, 5)	0.208880	256
81	(3, 3, 3)	1.156179	256
82	(0, 4, 4)	2.252137	256
83	(0, 0, 6)	2.805741	256

72 rows × 3 columns