Nom : Ghomari

Prénom : Alae

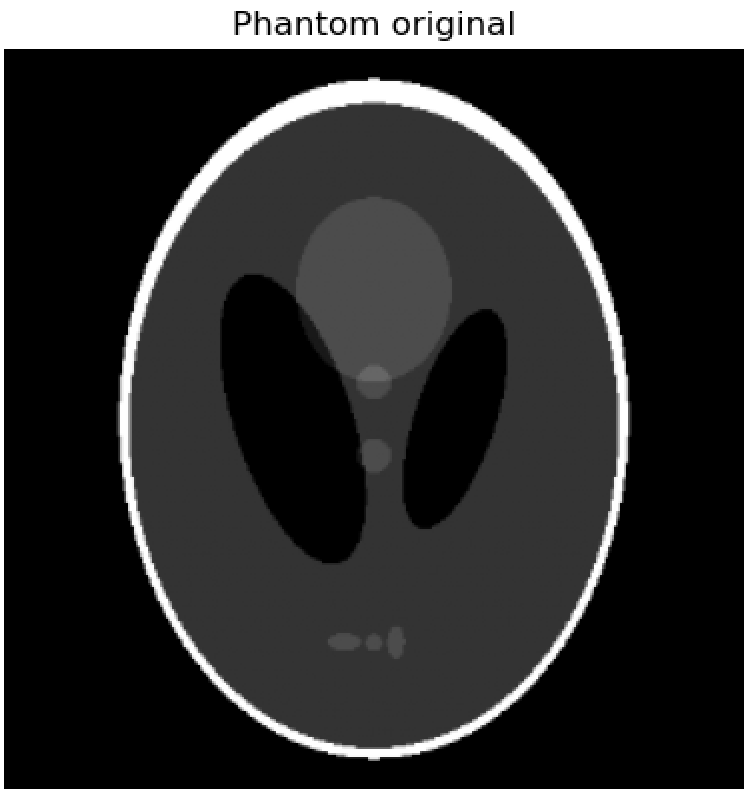
Spécialité : IAA

Groupe : [02]

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage.data import shepp_logan_phantom
from skimage.transform import resize
from skimage.filters import median, gaussian, threshold_otsu
from skimage.metrics import structural_similarity as ssim
from skimage.util import random_noise
from scipy.fft import fft2, ifft2
from scipy.signal import wiener
import pywt
from skimage.filters import threshold_local
```

Affichage du phantom de Shepp-Logan redimensionné à 256x256 pour visualisation.

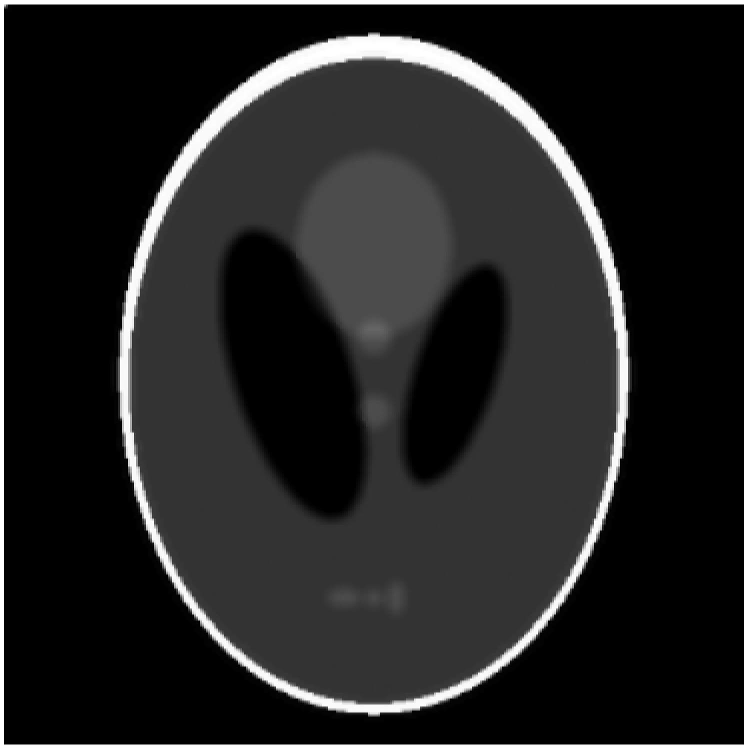
In [3]:
```
size = 256
phantom = resize(shepp_logan_phantom(), (size, size))
plt.imshow(phantom, cmap='gray')
plt.title("Phantom original")
plt.axis('off')
plt.show()
```



Transformation en k-space avec ajout de bruit Poisson, reconstruction de l'image IRM, puis débruitage avec un filtre de Wiener (5x5).

In [5]:
```
# Fourier → k-space
kspace = fft2(phantom)

# Bruit Poisson
noise = random_noise(np.abs(kspace), mode='poisson')
kspace_noisy = kspace + noise

# Reconstruction
irm_recon = np.abs(ifft2(kspace_noisy))

# Débruitage Wiener
irm_denoised = wiener(irm_recon, (5,5))

plt.imshow(irm_denoised, cmap='gray')
plt.title("IRM ")
plt.axis('off')
plt.show()
```

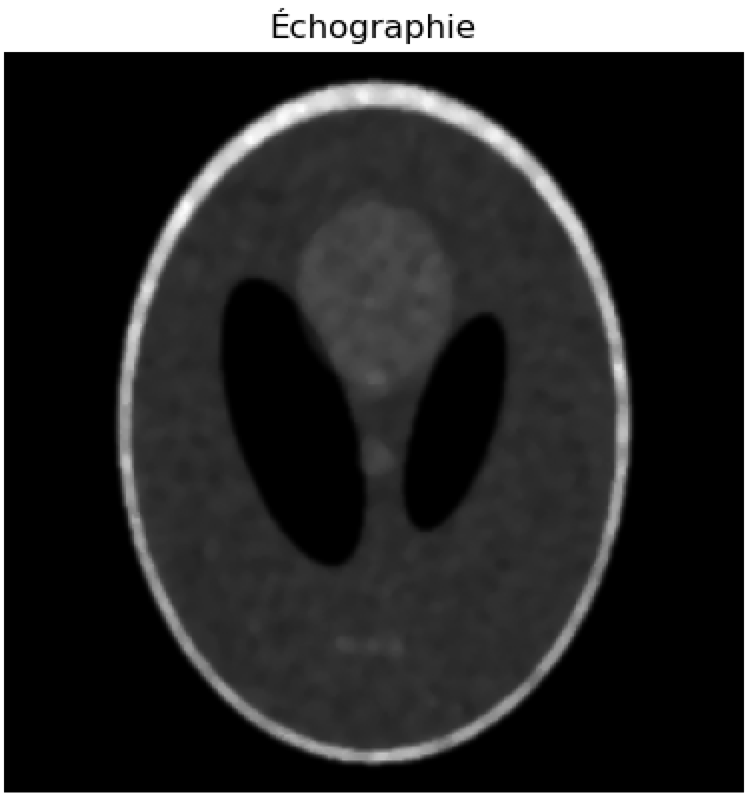
IRM



Ajout de bruit speckle sur le phantom, filtrage gaussien simulant le beamforming, puis débruitage médian pour l'image d'échographie.

In [6]:
```
# Bruit speckle multiplicatif
echo = phantom + phantom * np.random.randn(*phantom.shape) * 0.4

# Filtre gaussien = beamforming simplifié
echo_gauss = gaussian(echo, sigma=1)
```

```python
# Débruitage médian
from skimage.morphology import disk
echo_denoised = median(echo_gauss, disk(3))

plt.imshow(echo_denoised, cmap='gray')
plt.title("Échographie")
plt.axis('off')
plt.show()
```

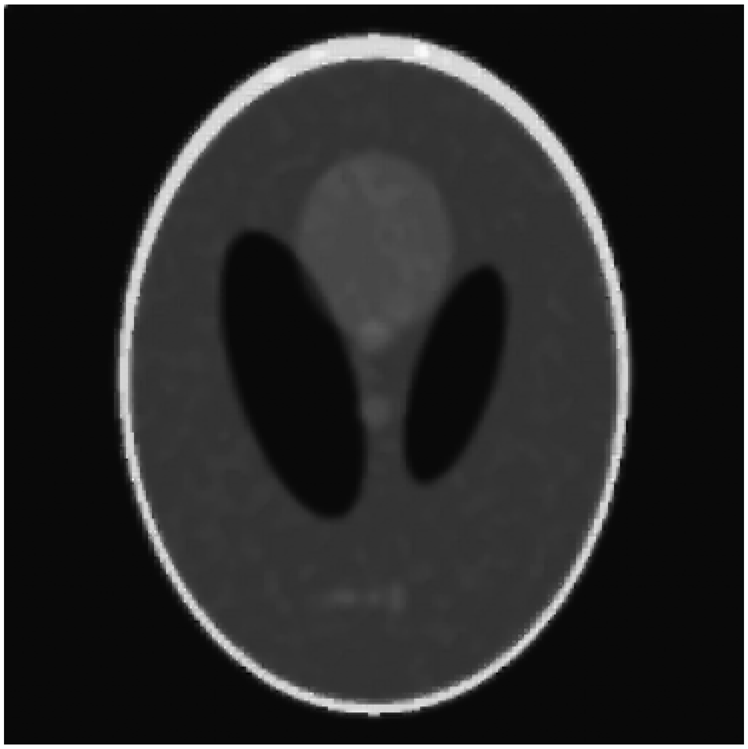


Fusion des images IRM et échographie via ondelettes (max des coefficients), puis reconstruction et normalisation de l'image finale.

In [13]:
```python
# Décomposition ondelettes
coeffs1 = pywt.dwt2(irm_denoised, 'db1')
coeffs2 = pywt.dwt2(echo_denoised, 'db1')

cA1, (cH1, cV1, cD1) = coeffs1
cA2, (cH2, cV2, cD2) = coeffs2

# ✅ Fusion par maximum (meilleure que moyenne)
cA = np.maximum(cA1, cA2)
cH = np.maximum(cH1, cH2)
cV = np.maximum(cV1, cV2)
cD = np.maximum(cD1, cD2)

# Reconstruction
fused_image = pywt.idwt2((cA, (cH, cV, cD)), 'db1')

# ✅ Normalisation
```

```
fused_image = (fused_image - fused_image.min()) / (fused_image.max() - fused_image.
fused_image = np.clip(fused_image, 0, 1)

plt.imshow(fused_image, cmap='gray')
plt.title("Image Fusionnée")
plt.axis('off')
plt.show()
```

Image Fusionnée



Segmentation des images par Otsu et adaptative, puis évaluation de la fusion via Dice et IoU par rapport au phantom.

Dice / IoU par rapport au Phantom :

- IRM : Dice=0.998, IoU=0.996
- Écho : Dice=0.913, IoU=0.840
- Fusion : Dice=0.119, IoU=0.063

> On remarque que la fusion a un score très faible.

In [17]:
```
def segment(img):
    t = threshold_otsu(img)
    return img > t

phantom_seg = segment(phantom)
irm_seg = segment(irm_denoised)
echo_seg = segment(echo_denoised)

# ✅ Segmentation fusion (Otsu normalisé)
t = threshold_otsu(fused_image)
```

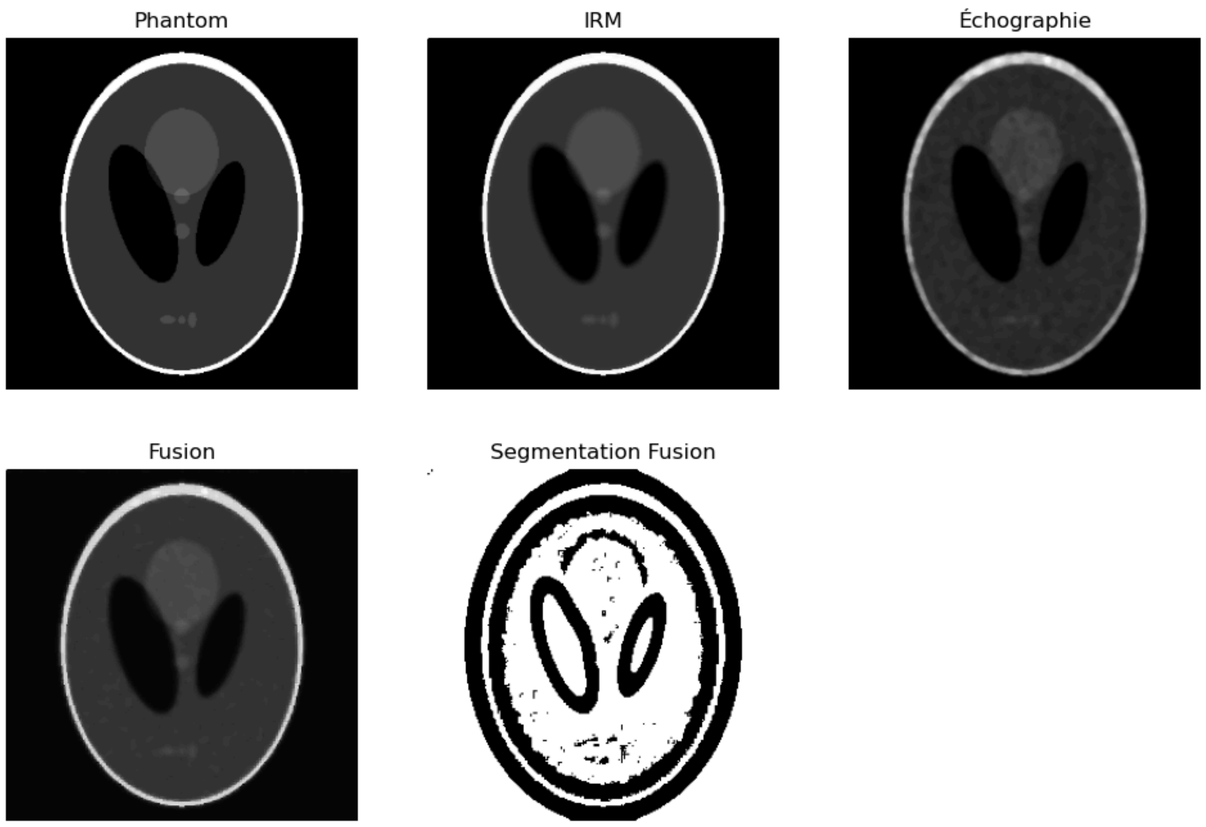
```python
fused_seg_otsu = fused_image > t

# ✅ Segmentation adaptée (meilleure si l’autre est noire)
adaptive_thresh = threshold_local(fused_image, block_size=35, offset=0.01)
fused_seg_adaptive = fused_image > adaptive_thresh

# ✅ Choisis la meilleure des deux :
fused_seg = fused_seg_adaptive   # << tu peux changer en fused_seg_otsu si Otsu mar

def dice_coef(a, b):
    a, b = a.astype(bool), b.astype(bool)
    return 2 * np.sum(a & b) / (np.sum(a) + np.sum(b))

def iou(a, b):
    a, b = a.astype(bool), b.astype(bool)
    return np.sum(a & b) / np.sum(a | b)

print("Dice / IoU par rapport au Phantom :\n")
print(f"IRM      : Dice={dice_coef(phantom_seg, irm_seg):.3f},  IoU={iou(phantom_se
print(f"Écho     : Dice={dice_coef(phantom_seg, echo_seg):.3f}, IoU={iou(phantom_se
print(f"Fusion   : Dice={dice_coef(phantom_seg, fused_seg):.3f}, IoU={iou(phantom_s
```

```
Dice / IoU par rapport au Phantom :

IRM      : Dice=0.998,  IoU=0.996
Écho     : Dice=0.913, IoU=0.840
Fusion   : Dice=0.119, IoU=0.063
```

Affichage comparatif : phantom, IRM, échographie, image fusionnée et segmentation de la fusion pour visualiser les résultats.

In [18]:

```python
fig, axes = plt.subplots(2, 3, figsize=(12,8))

axes[0,0].imshow(phantom, cmap='gray'); axes[0,0].set_title("Phantom"); axes[0,0].a
axes[0,1].imshow(irm_denoised, cmap='gray'); axes[0,1].set_title("IRM"); axes[0,1].
axes[0,2].imshow(echo_denoised, cmap='gray'); axes[0,2].set_title("Échographie"); a

axes[1,0].imshow(fused_image, cmap='gray'); axes[1,0].set_title("Fusion"); axes[1,0
axes[1,1].imshow(fused_seg, cmap='gray'); axes[1,1].set_title("Segmentation Fusion"
axes[1,2].axis('off')

plt.show()
```

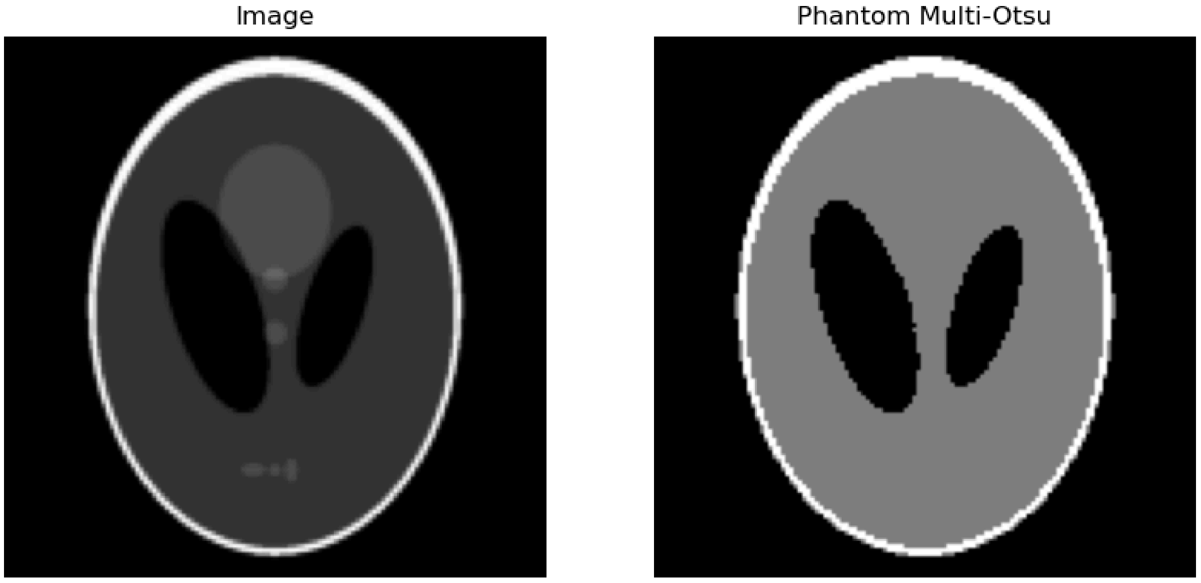


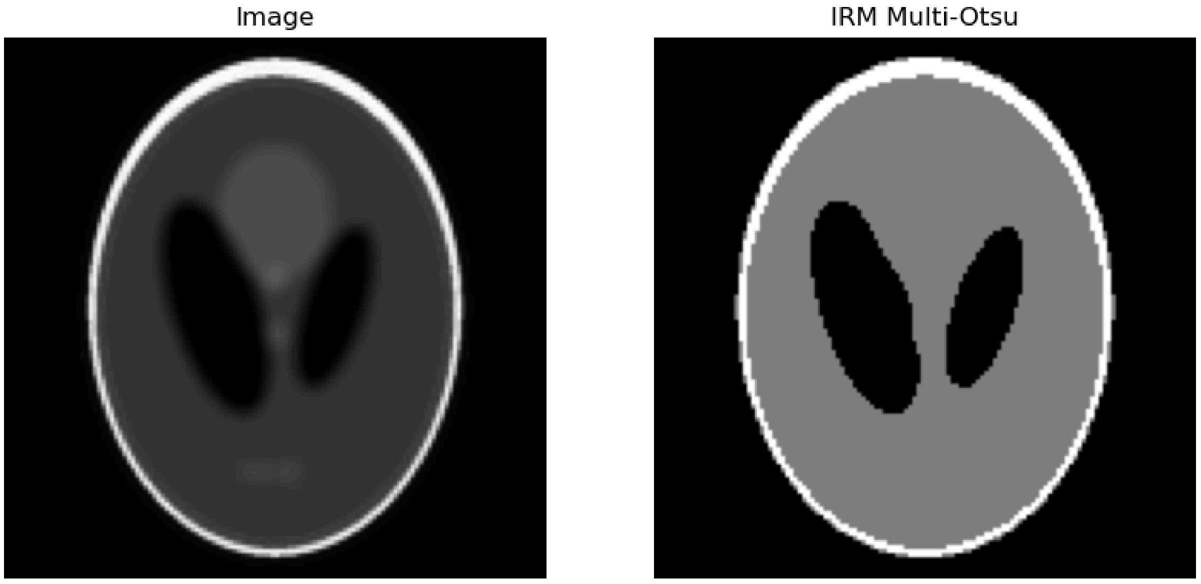
Segmentation multi-classes (3 classes, Multi-Otsu) sur Phantom, IRM, échographie et fusion, avec calcul des scores Dice et IoU par classe.

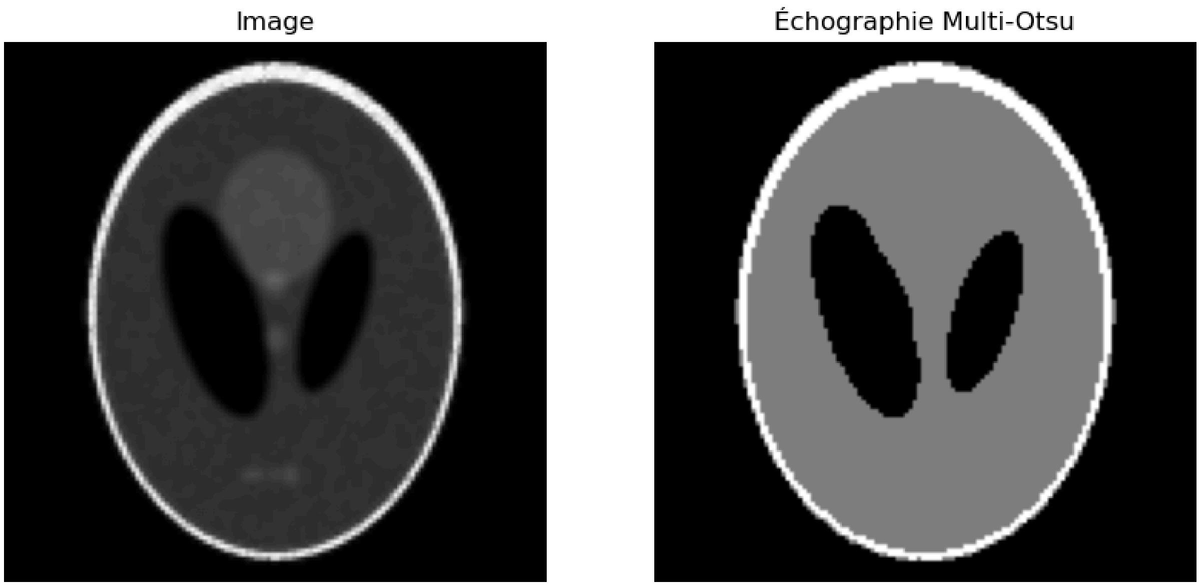
Avec la segmentation Multi-Otsu, la fusion a donné un **très bon résultat**, comme le montrent les métriques Dice et IoU, bien supérieures à celles obtenues avec la simple fusion binaire.

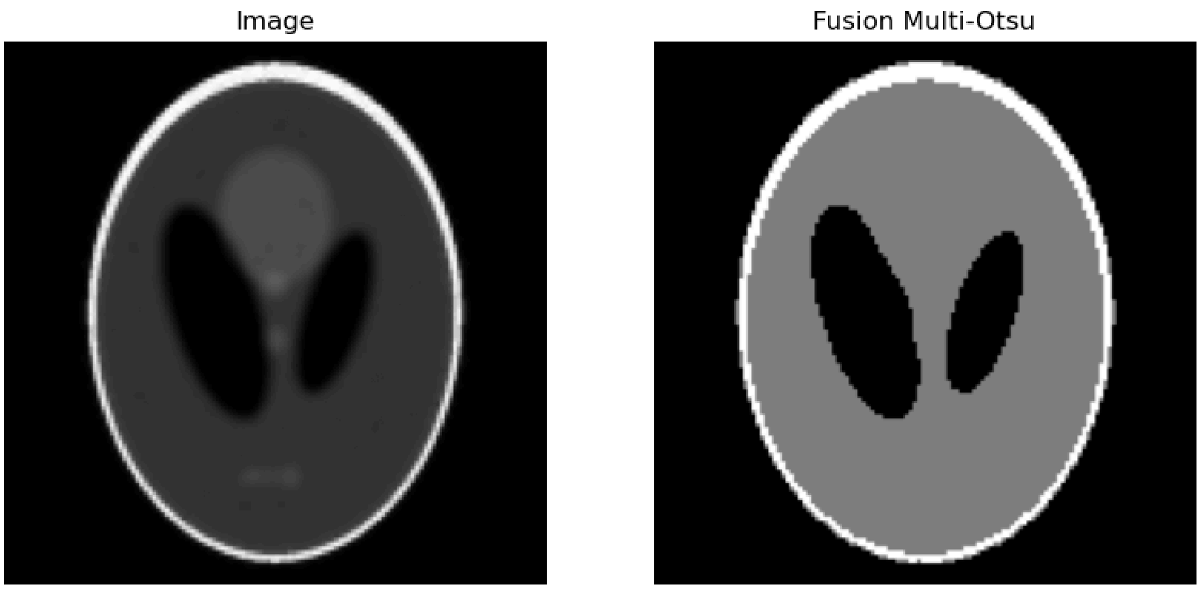
In [22]:
```
from skimage.filters import threshold_multiotsu
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def multiotsu_seg(img, classes=3):
    thresholds = threshold_multiotsu(img, classes=classes)
    seg = np.digitize(img, bins=thresholds)
    return seg

phantom_seg_multi = multiotsu_seg(phantom, 3)
irm_seg_multi = multiotsu_seg(irm_denoised, 3)
echo_seg_multi = multiotsu_seg(echo_denoised, 3)
fused_seg_multi = multiotsu_seg(fused_image, 3)


def dice_coef_multilabel(a, b, labels):
    dice_scores = {}
    for label in labels:
        a_label = (a == label)
        b_label = (b == label)
        if np.sum(a_label) + np.sum(b_label) == 0:
            dice_scores[label] = 1.0
        else:
            dice_scores[label] = 2 * np.sum(a_label & b_label) / (np.sum(a_label) +
    return dice_scores

def iou_multilabel(a, b, labels):
    iou_scores = {}
    for label in labels:
        a_label = (a == label)
        b_label = (b == label)
        if np.sum(a_label | b_label) == 0:
            iou_scores[label] = 1.0
        else:
            iou_scores[label] = np.sum(a_label & b_label) / np.sum(a_label | b_labe
    return iou_scores

labels = [0, 1, 2]  # 3 classes

# Calcul des métriques
dice_irm = dice_coef_multilabel(phantom_seg_multi, irm_seg_multi, labels)
iou_irm = iou_multilabel(phantom_seg_multi, irm_seg_multi, labels)

dice_echo = dice_coef_multilabel(phantom_seg_multi, echo_seg_multi, labels)
iou_echo = iou_multilabel(phantom_seg_multi, echo_seg_multi, labels)

dice_fused = dice_coef_multilabel(phantom_seg_multi, fused_seg_multi, labels)
iou_fused = iou_multilabel(phantom_seg_multi, fused_seg_multi, labels)

print("Dice / IoU par classe :\n")
print("IRM      :", dice_irm, iou_irm)
print("Écho     :", dice_echo, iou_echo)
print("Fusion   :", dice_fused, iou_fused)


cmap = ListedColormap(['black', 'gray', 'white'])  # 0=fond, 1=intermédiaire, 2=cla

def show_segmentation(img, seg, title="Segmentation"):
    fig, axes = plt.subplots(1,2, figsize=(10,5))
    axes[0].imshow(img, cmap='gray')
    axes[0].set_title("Image")
    axes[0].axis('off')

    axes[1].imshow(seg, cmap=cmap)
    axes[1].set_title(title)
    axes[1].axis('off')

    plt.show()

# Afficher chaque segmentation
show_segmentation(phantom, phantom_seg_multi, "Phantom Multi-Otsu")
show_segmentation(irm_denoised, irm_seg_multi, "IRM Multi-Otsu")
show_segmentation(echo_denoised, echo_seg_multi, "Échographie Multi-Otsu")
show_segmentation(fused_image, fused_seg_multi, "Fusion Multi-Otsu")
```

```
Dice / IoU par classe :

IRM      : {0: np.float64(0.9972844896437889), 1: np.float64(0.9952023028946105), 2:
np.float64(0.9939230249831195)} {0: np.float64(0.9945836873406967), 1: np.float64(0.
9904504217730383), 2: np.float64(0.9879194630872483)}
Écho     : {0: np.float64(0.9982958781552881), 1: np.float64(0.9955228653661656), 2:
np.float64(0.9838056680161943)} {0: np.float64(0.9965975544922914), 1: np.float64(0.
991085641515441), 2: np.float64(0.9681274900398407)}
Fusion   : {0: np.float64(0.9979229908931139), 1: np.float64(0.9957644050187805), 2:
np.float64(0.9905277401894452)} {0: np.float64(0.9958545918367347), 1: np.float64(0.
9915645392328506), 2: np.float64(0.9812332439678284)}
```









In [20]:
```
labels = [0, 1, 2]  # 3 classes


dice_irm = dice_coef_multilabel(phantom_seg_multi, irm_seg_multi, labels)
iou_irm = iou_multilabel(phantom_seg_multi, irm_seg_multi, labels)

dice_echo = dice_coef_multilabel(phantom_seg_multi, echo_seg_multi, labels)
iou_echo = iou_multilabel(phantom_seg_multi, echo_seg_multi, labels)

dice_fused = dice_coef_multilabel(phantom_seg_multi, fused_seg_multi, labels)
iou_fused = iou_multilabel(phantom_seg_multi, fused_seg_multi, labels)

def print_metrics_rounded(dice, iou, method_name):
    dice_r = {k: round(v, 3) for k,v in dice.items()}
    iou_r  = {k: round(v, 3) for k,v in iou.items()}
    print(f"{method_name} : Dice={dice_r}, IoU={iou_r}")


print("=== Dice / IoU par classe \n")
```

```
print_metrics_rounded(dice_irm, iou_irm, "IRM")
print_metrics_rounded(dice_echo, iou_echo, "Écho")
print_metrics_rounded(dice_fused, iou_fused, "Fusion")
```

```
=== Dice / IoU par classe

IRM : Dice={0: np.float64(0.997), 1: np.float64(0.995), 2: np.float64(0.994)}, IoU=
{0: np.float64(0.995), 1: np.float64(0.99), 2: np.float64(0.988)}
Écho : Dice={0: np.float64(0.998), 1: np.float64(0.996), 2: np.float64(0.984)}, IoU=
{0: np.float64(0.997), 1: np.float64(0.991), 2: np.float64(0.968)}
Fusion : Dice={0: np.float64(0.998), 1: np.float64(0.996), 2: np.float64(0.991)}, Io
U={0: np.float64(0.996), 1: np.float64(0.992), 2: np.float64(0.981)}
```

In [ ]: