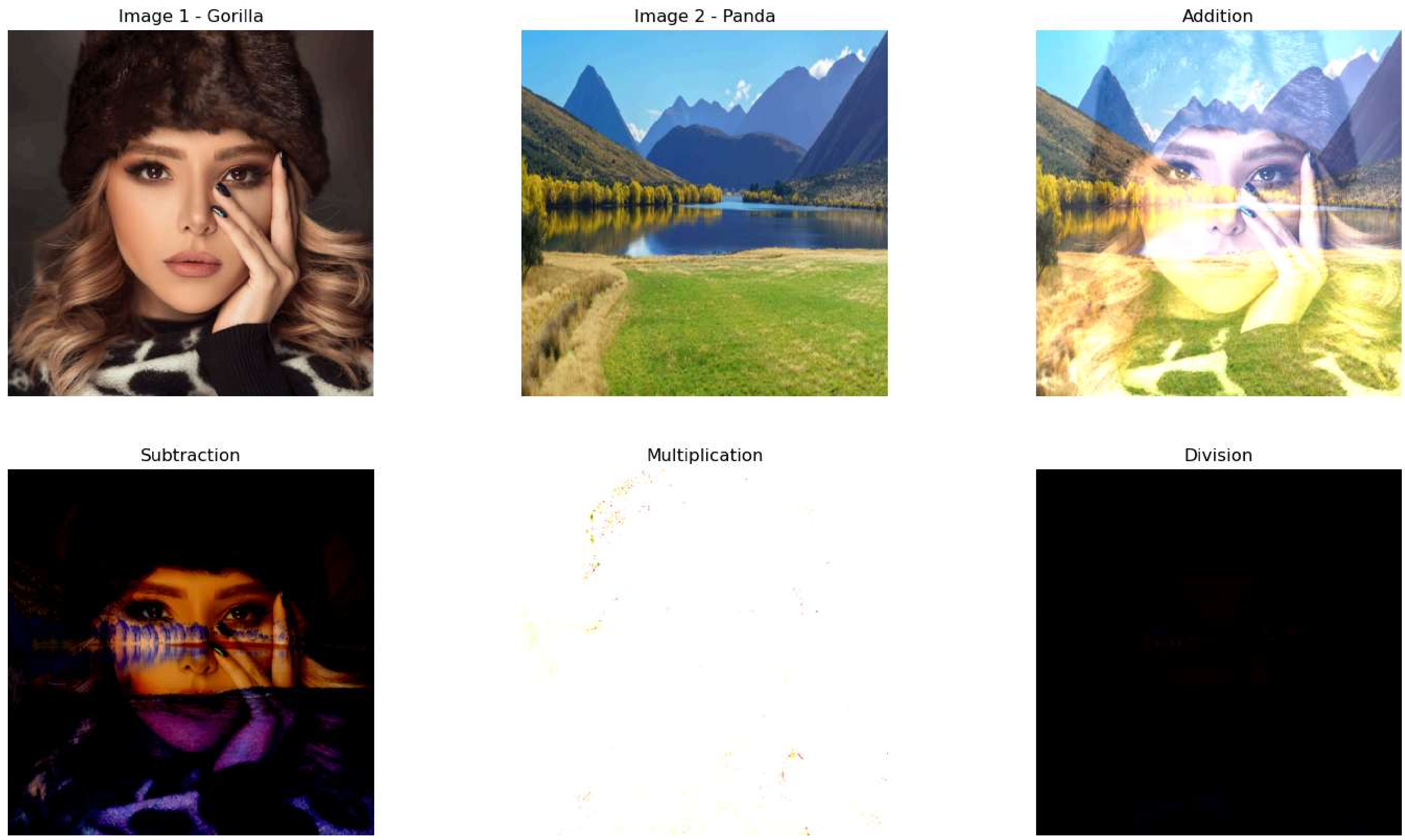
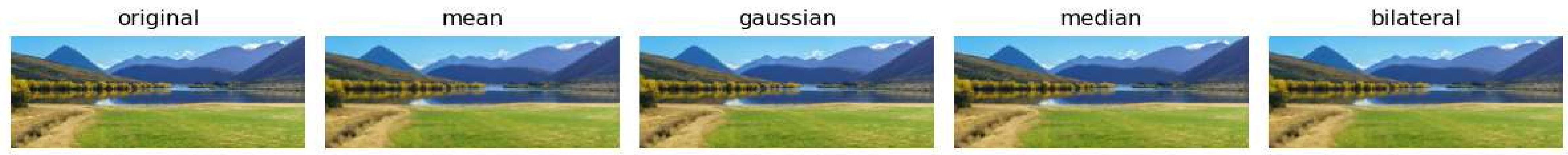
In [4]:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img1 = cv2.imread("girl.jpg")

img2 = cv2.imread("scene.jpg")

img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))

add = cv2.add(img1, img2)
sub = cv2.subtract(img1, img2)
mul = cv2.multiply(img1, img2)
div = cv2.divide(img1, img2)

rows, cols = img1.shape[:2]
M_trans = np.float32([[1, 0, 100], [0, 1, 50]])
translation = cv2.warpAffine(img1, M_trans, (cols, rows))

M_rot = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 1)
rotation = cv2.warpAffine(img1, M_rot, (cols, rows))

transform_images = [translation, rotation]
transform_titles = ['Translation', 'Rotation 90°']

def convert_bgr_to_rgb(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

images = [img1, img2, add, sub, mul, div]
titles = ['Image 1 - Gorilla', 'Image 2 - Panda', 'Addition', 'Subtraction', 'Multiplication', 'Division']

# Plot the images
plt.figure(figsize=(18, 10))
for i, (img, title) in enumerate(zip(images, titles)):
    plt.subplot(2, 3, i+1)
    plt.imshow(convert_bgr_to_rgb(img))
    plt.title(title)
    plt.axis('off')

```



In [7]:

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("scene.jpg")

mean = cv2.blur(img, (5, 5))
gaussian = cv2.GaussianBlur(img, (5, 5), 0)
median = cv2.medianBlur(img, 5)
bilateral = cv2.bilateralFilter(img, 9, 75, 75)

images = [img, mean, gaussian, median, bilateral]
titles = ['original', 'mean', 'gaussian', 'median', 'bilateral']

plt.figure(figsize=(12, 6))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
    plt.title(titles[i])
    plt.axis('off')

plt.tight_layout()
plt.show()

```

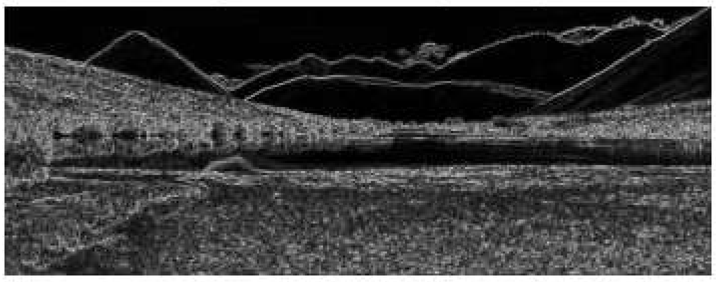


In [8]:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image in grayscale
image = cv2.imread("scene.jpg", cv2.IMREAD_GRAYSCALE)

# Apply Sobel operator in x and y directions
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)

# Calculate gradient magnitude
sobel_magnitude = np.uint8(np.absolute(cv2.magnitude(sobel_x, sobel_y)))

# Sharpen image by adding edge magnitude to original
sharp_image = cv2.add(image, sobel_magnitude)

# Plotting the results
plt.figure(figsize=(10, 10))

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(sobel_magnitude, cmap='gray')
plt.title("Edge Detection (Sobel)")
plt.axis('off')

plt.figure(figsize=(6, 6))
plt.imshow(sharp_image, cmap='gray')
plt.title("Sharpened Image")
plt.axis('off')

plt.show()

```

Original Image

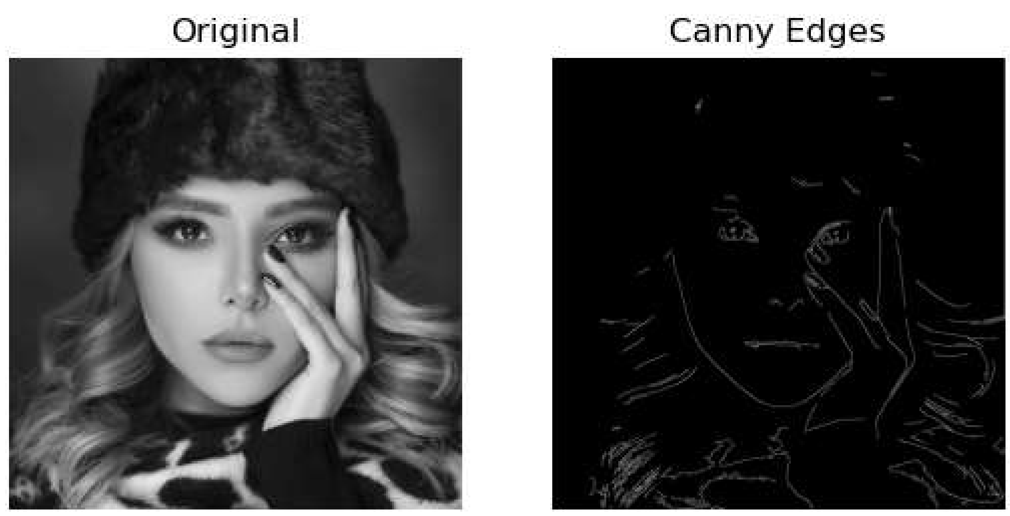
Edge Detection (Sobel)



Sharpened Image

In [9]:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image in grayscale
image = cv2.imread("girl.jpg", cv2.IMREAD_GRAYSCALE)

# Apply Gaussian blur before Canny to reduce noise
edges = cv2.Canny(cv2.GaussianBlur(image, (5, 5), 1.5), 50, 150)

# Plot original and edge-detected images
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(edges, cmap='gray')
plt.title('Canny Edges')
plt.axis('off')

plt.show()

```



In [13]:

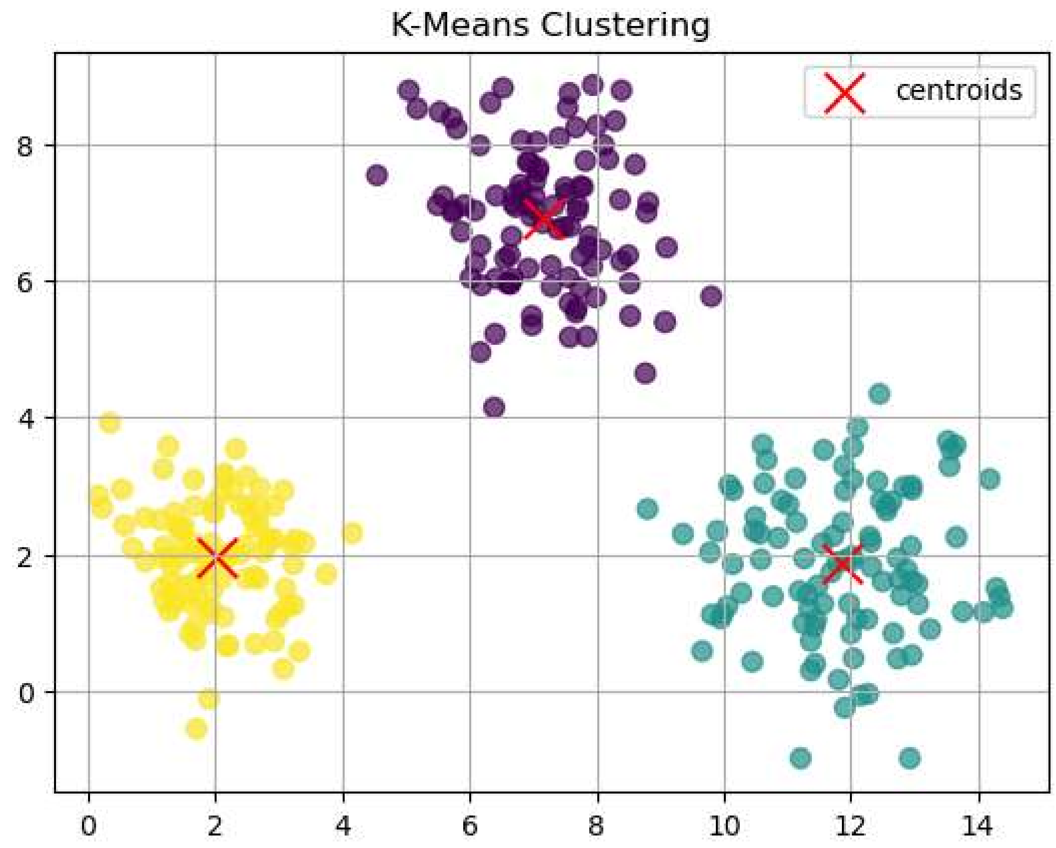
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread("girl.jpg", cv2.IMREAD_GRAYSCALE)
equalized = cv2.equalizeHist(image)
stretched = np.uint8(((image - np.min(image)) * 255) / (np.max(image) - np.min(image)))
for i, (img, title) in enumerate([...]):
    plt.subplot(2, 3, 2*i + 1)
    plt.imshow(img, cmap='gray')
    ...
    plt.subplot(2, 3, 2*i + 2)
    plt.hist(img.ravel(), bins=256, range=(0, 256))

```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 7
      5 equalized = cv2.equalizeHist(image)
      6 stretched = np.uint8(((image - np.min(image)) * 255) / (np.max(image) - np.min(image)))
----> 7 for i, (img, title) in enumerate([...]):
      8     plt.subplot(2, 3, 2*i + 1)
      9     plt.imshow(img, cmap='gray')

TypeError: cannot unpack non-iterable ellipsis object
```

In [20]:

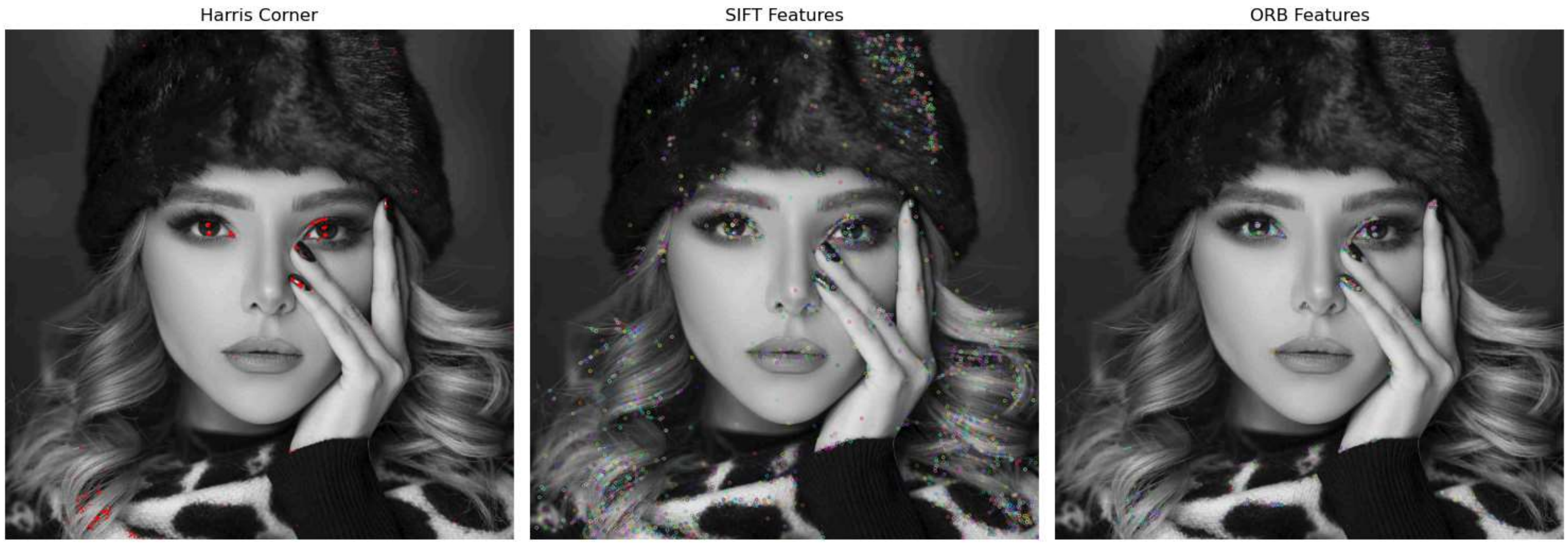
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
X = np.vstack([
    np.random.normal(loc=[2, 2], scale=0.8, size=(100, 2)),
    np.random.normal(loc=[7, 7], scale=1.0, size=(100, 2)),
    np.random.normal(loc=[12, 2], scale=1.2, size=(100, 2))
])
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis', s=50,alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],c='red',marker='x',s=200,label ="centroids")

plt.title('K-Means Clustering')
plt.legend()
plt.grid(True)
plt.show()
```

```
E:\Anaconda3\envs\tensorflow\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
E:\Anaconda3\envs\tensorflow\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to have a memory lea
k on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=2.
  warnings.warn(
```



In [21]:

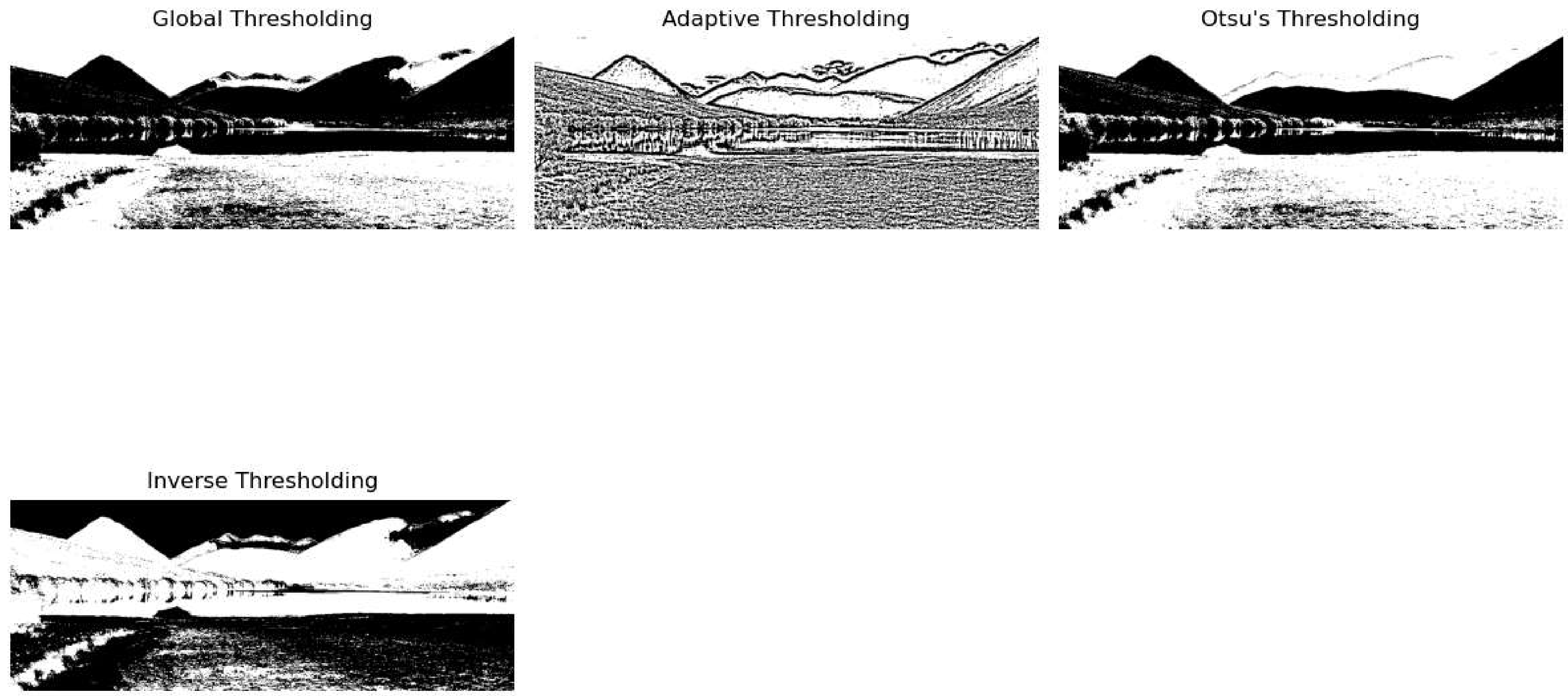
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load image in grayscale
image = cv2.imread("girl.jpg", cv2.IMREAD_GRAYSCALE)

# Harris Corner Detection
harris = cv2.cornerHarris(image, 2, 3, 0.04)
harris = cv2.dilate(harris, None)
image_harris = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
image_harris[harris > 0.01 * harris.max()] = [0, 0, 255]

# SIFT Feature Detection
sift = cv2.SIFT_create()
kp_sift, des_sift = sift.detectAndCompute(image, None)
image_sift = cv2.drawKeypoints(image, kp_sift, None)

# ORB Feature Detection
orb = cv2.ORB_create()
kp_orb, des_orb = orb.detectAndCompute(image, None)
image_orb = cv2.drawKeypoints(image, kp_orb, None)

# Combine images and titles
images = [image_harris, image_sift, image_orb]
titles = ['Harris Corner', 'SIFT Features', 'ORB Features']

# Plot results
plt.figure(figsize=(15, 8))
for i, (img, title) in enumerate(zip(images, titles), 1):
    plt.subplot(1, 3, i)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')

plt.tight_layout()
plt.show()

# Print number of keypoints
print(f"SIFT Keypoints: {len(kp_sift)} | ORB Keypoints: {len(kp_orb)}")

```




```
SIFT Keypoints: 2016 | ORB Keypoints: 500
```

In [22]:

```python
import cv2
import matplotlib.pyplot as plt

# Load image in grayscale
image = cv2.imread("scene.jpg", cv2.IMREAD_GRAYSCALE)

# Apply various thresholding techniques
global_thresh = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)[1]
adaptive_thresh = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                        cv2.THRESH_BINARY, 11, 2)
otsu_thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
inverse_thresh = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)[1]

# Titles and images for display
titles = ['Global Thresholding', 'Adaptive Thresholding', "Otsu's Thresholding", 'Inverse Thresholding']
images = [global_thresh, adaptive_thresh, otsu_thresh, inverse_thresh]

# Display the images
plt.figure(figsize=(12, 8))
for i, (title, img) in enumerate(zip(titles, images)):
    plt.subplot(2, 3, i + 1)
    plt.imshow(img, cmap='gray')
    plt.title(title)
    plt.axis('off')

plt.tight_layout()
plt.show()

```



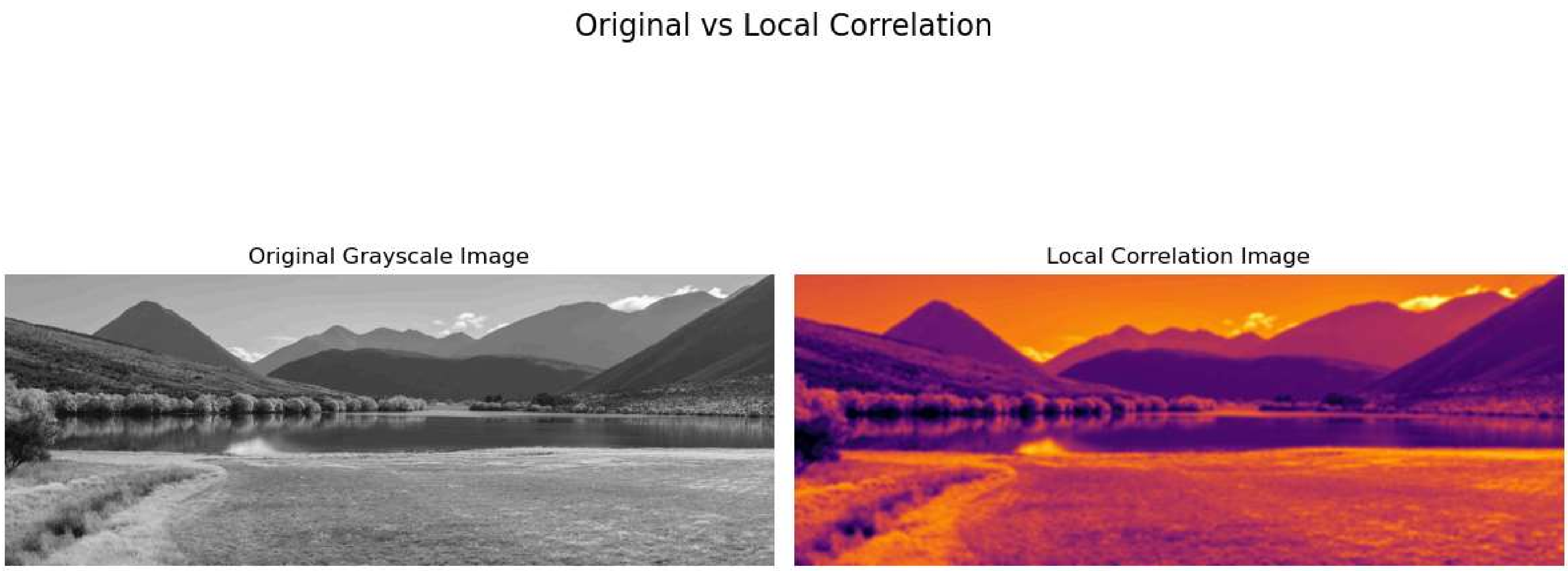
In [23]:

```python
import cv2
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

# Load grayscale image
image = cv2.imread("girl.jpg", cv2.IMREAD_GRAYSCALE)

# Create a second image filled with zeros, same shape as the original
image2 = np.zeros_like(image)

# Flatten both images and use them as features
X = np.array([image.flatten(), image2.flatten()])
y = [0, 1]  # Labels for classification

# Initialize and train the K-Nearest Neighbors classifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X, y)

# Predict the class of the input image
y_pred = knn.predict([image.flatten()])

# Display the image with predicted class
plt.imshow(image, cmap='gray')
plt.title(f'Predicted Class: {y_pred[0]}')
plt.axis('off')
plt.show()
```

Predicted Class: 0

In [25]:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img_local = cv2.imread("scene.jpg")

gray_local = cv2.cvtColor(img_local, cv2.COLOR_BGR2GRAY)
kernel = np.ones((5, 5), np.float32) / 25
local_corr = cv2.filter2D(gray_local, -1, kernel)

images = [gray_local, local_corr]
titles = ['Original Grayscale Image', 'Local Correlation Image']
cmaps = ['gray', 'inferno']

plt.figure(figsize=(12, 6))
for i, (img, title, cmap) in enumerate(zip(images, titles, cmaps), 1):
    plt.subplot(1, 2, i)
    plt.imshow(img, cmap=cmap)
    plt.title(title)
    plt.axis('off')

plt.suptitle('Original vs Local Correlation', fontsize=16)
plt.tight_layout()
plt.show()

```



In [ ]:

```python

```