In [16]:
```
# ============================================================================
# CELL 1: Setup
# ============================================================================

import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim

print("✅ FrameShift V1.1 Ready with Advanced Features!")
print("   • ROI Selection")
print("   • Background Removal")
print("   • Edge Detection for Texture")
print("   • Text Filtering")
```

```
✅ FrameShift V1.1 Ready with Advanced Features!
   • ROI Selection
   • Background Removal
   • Edge Detection for Texture
   • Text Filtering
```

In [17]:
```
# ============================================================================
# CELL 2: Configuration - Set Your Options Here
# ============================================================================

# Configure analysis options
CONFIG = {
    'use_roi': False,           # True = Select specific region only
    'roi_coords': None,         # (x, y, w, h) or None for manual
    'remove_background': True, # True = Remove background before analysis
    'use_edge_detection': True,# True = Detect texture changes (tire wear)
    'filter_text_regions': True,# True = Ignore text labels
    'sensitivity': 0.01,
    'gen_image': False,
    # 0.1 = very sensitive, 0.2 = less sensitive
}

print("📋 Current Configuration:")
for key, value in CONFIG.items():
    print(f"   {key}: {value}")
```

```
📋 Current Configuration:
   use_roi: False
   roi_coords: None
   remove_background: True
   use_edge_detection: True
   filter_text_regions: True
   sensitivity: 0.01
   gen_image: False
```

In [18]:
```
# ============================================================================
# CELL 3: Load or Create Images
# ============================================================================

import tkinter as tk
```

```python
from tkinter import filedialog

# Option A: Create test images
def create_test_images():
    img1 = np.ones((400, 600, 3), dtype=np.uint8) * 200
    cv2.rectangle(img1, (50, 50), (150, 150), (100, 150, 200), -1)
    cv2.circle(img1, (400, 100), 50, (150, 200, 100), -1)
    cv2.rectangle(img1, (200, 250), (400, 350), (200, 100, 150), -1)

    img2 = img1.copy()
    cv2.rectangle(img2, (50, 50), (150, 150), (255, 150, 200), -1)
    cv2.circle(img2, (450, 100), 50, (150, 200, 100), -1)
    cv2.rectangle(img2, (450, 250), (550, 350), (255, 100, 100), -1)

    return img1, img2

# Option B: Select images using file dialog
def select_images_gui():
    """Open file dialogs to select two images"""
    root = tk.Tk()
    root.withdraw()  # Hide the main window
    root.attributes('-topmost', True)  # Bring dialogs to front

    print("📁 Select Image 1 (Before)...")
    img1_path = filedialog.askopenfilename(
        title="Select Image 1 (Before)",
        filetypes=[
            ("Image files", "*.jpg *.jpeg *.png *.bmp *.tiff"),
            ("All files", "*.*")
        ]
    )

    if not img1_path:
        print("❌ No image selected for Image 1")
        root.destroy()
        return None, None

    print("📁 Select Image 2 (After)...")
    img2_path = filedialog.askopenfilename(
        title="Select Image 2 (After)",
        filetypes=[
            ("Image files", "*.jpg *.jpeg *.png *.bmp *.tiff"),
            ("All files", "*.*")
        ]
    )

    if not img2_path:
        print("❌ No image selected for Image 2")
        root.destroy()
        return None, None

    root.destroy()

    # Load images
    img1 = cv2.imread(img1_path)
    img2 = cv2.imread(img2_path)
```

```
    if img1 is not None:
        print(f"✅ Image 1 loaded: {img1_path}")
    else:
        print(f"❌ Failed to load Image 1: {img1_path}")

    if img2 is not None:
        print(f"✅ Image 2 loaded: {img2_path}")
    else:
        print(f"❌ Failed to load Image 2: {img2_path}")

    return img1, img2

# Load images based on configuration
if CONFIG.get('gen_image', False):
    print("🎨 Using generated test images...")
    img1, img2 = create_test_images()
else:
    print("🖱️ Opening file dialogs...\n")
    img1, img2 = select_images_gui()

# Validate images loaded successfully
if img1 is None or img2 is None:
    error_msg = "\n❌ ERROR: Failed to load images!\n"
    error_msg += "\n💡 Options:\n"
    error_msg += "   1. Re-run this cell to try again\n"
    error_msg += "   2. Set CONFIG['gen_image'] = True to use test images\n"
    error_msg += "   3. Check that the selected files are valid images"
    print(error_msg)
    raise FileNotFoundError("Images not loaded successfully")

print(f"\n✅ Both images loaded successfully!")
print(f"   Image 1 shape: {img1.shape}")
print(f"   Image 2 shape: {img2.shape}")
```

```
🖱️ Opening file dialogs...

📂 Select Image 1 (Before)...
📂 Select Image 2 (After)...
✅ Image 1 loaded: C:/Users/wadhw/OneDrive/Desktop/FrameShift/approach1/test-image
s/sidepod1.png
✅ Image 2 loaded: C:/Users/wadhw/OneDrive/Desktop/FrameShift/approach1/test-image
s/sidepod2.png

✅ Both images loaded successfully!
   Image 1 shape: (831, 752, 3)
   Image 2 shape: (836, 749, 3)
```

In [19]:

```
# ============================================================================
# CELL 4: Optional - ROI Selection
# ============================================================================

def select_roi_manual(img, window_name="Select ROI"):
    """
    Interactive ROI selection
    Usage: Click and drag to select region
```

```
    """
    roi = cv2.selectROI(window_name, img, fromCenter=False, showCrosshair=True)
    cv2.destroyAllWindows()
    return roi  # Returns (x, y, w, h)

def apply_roi(img1, img2, roi_coords):
    """Extract ROI from both images"""
    if roi_coords is None or all(v == 0 for v in roi_coords):
        return img1, img2

    x, y, w, h = roi_coords
    return img1[y:y+h, x:x+w], img2[y:y+h, x:x+w]

# Apply ROI if enabled
if CONFIG['use_roi']:
    if CONFIG['roi_coords'] is None:
        print("🖱️  Select ROI by clicking and dragging...")
        print("   Press SPACE or ENTER when done, ESC to cancel")
        roi = select_roi_manual(img1, "Select Region of Interest")
        CONFIG['roi_coords'] = roi

    img1, img2 = apply_roi(img1, img2, CONFIG['roi_coords'])
    print(f"✅ ROI selected: {CONFIG['roi_coords']}")
else:
    print("ℹ️ Using full images (ROI disabled)")
```

```
ℹ️ Using full images (ROI disabled)
```

In [20]:

```
# ============================================================================
# CELL 5: Optional - Background Removal
# ============================================================================

def remove_background_grabcut(img):
    """
    Remove background using GrabCut algorithm
    Assumes subject is in center region
    """
    mask = np.zeros(img.shape[:2], np.uint8)
    bgd_model = np.zeros((1, 65), np.float64)
    fgd_model = np.zeros((1, 65), np.float64)

    # Define rectangle around subject (center 80% of image)
    h, w = img.shape[:2]
    rect = (int(w*0.1), int(h*0.1), int(w*0.8), int(h*0.8))

    # Run GrabCut
    cv2.grabCut(img, mask, rect, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)

    # Create mask: 0,2 = background, 1,3 = foreground
    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

    # Apply mask
    result = img * mask2[:, :, np.newaxis]

    return result, mask2
```

```python
def remove_background_simple(img, method='color'):
    """
    Simple background removal methods

    method='color': Remove gray/white backgrounds
    method='threshold': Keep only bright/colored regions
    """
    if method == 'color':
        # Convert to HSV
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

        # Mask for low saturation (gray/white background)
        lower = np.array([0, 30, 50])
        upper = np.array([180, 255, 255])
        mask = cv2.inRange(hsv, lower, upper)

    elif method == 'threshold':
        # Keep bright regions
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        _, mask = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY)

    # Apply mask
    result = cv2.bitwise_and(img, img, mask=mask)
    return result, mask

# Apply background removal if enabled
if CONFIG['remove_background']:
    print("🎨 Removing backgrounds...")
    try:
        img1_nobg, mask1 = remove_background_grabcut(img1)
        img2_nobg, mask2 = remove_background_grabcut(img2)
        print("✅ Background removed (GrabCut)")

        # Use masked versions
        img1 = img1_nobg
        img2 = img2_nobg

    except Exception as e:
        print(f"⚠️ GrabCut failed: {e}")
        print("   Falling back to simple method...")
        img1, mask1 = remove_background_simple(img1)
        img2, mask2 = remove_background_simple(img2)
        print("✅ Background removed (Simple)")
else:
    print("ℹ️ Using full images (background removal disabled)")
```

```
🎨 Removing backgrounds...
✅ Background removed (GrabCut)
```

In [21]:

```python
# ============================================================================
# CELL 6: Align Images
# ============================================================================

# Convert to grayscale
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
```

```
# Find features
orb = cv2.ORB_create(5000)
kp1, des1 = orb.detectAndCompute(gray1, None)
kp2, des2 = orb.detectAndCompute(gray2, None)

# Match features
matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = sorted(matcher.match(des1, des2), key=lambda x: x.distance)

# Align if enough matches
if len(matches) > 10:
    src_pts = np.float32([kp1[m.queryIdx].pt for m in matches[:50]]).reshape(-1, 1,
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches[:50]]).reshape(-1, 1,
    H, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC)

    h, w = img2.shape[:2]
    img1_aligned = cv2.warpPerspective(img1, H, (w, h))
    gray1_aligned = cv2.cvtColor(img1_aligned, cv2.COLOR_BGR2GRAY)
else:
    img1_aligned = img1
    gray1_aligned = gray1

print(f"✅ Aligned using {len(matches)} feature matches")
```

```
✅ Aligned using 884 feature matches
```

In [22]:

```
# ============================================================================
# CELL 7: Compute Differences (with Edge Detection option)
# ============================================================================

if CONFIG['use_edge_detection']:
    print("🔍 Using edge detection for texture analysis...")

    # Detect edges in both images
    edges1 = cv2.Canny(gray1_aligned, 50, 150)
    edges2 = cv2.Canny(gray2, 50, 150)

    # Compare edge maps
    edge_diff = cv2.absdiff(edges1, edges2).astype(float) / 255.0

    # Also get edge density
    kernel = np.ones((15, 15), np.float32) / 225
    edge_density1 = cv2.filter2D(edges1.astype(float) / 255.0, -1, kernel)
    edge_density2 = cv2.filter2D(edges2.astype(float) / 255.0, -1, kernel)
    density_diff = np.abs(edge_density1 - edge_density2)

    # Standard methods
    pixel_diff = cv2.absdiff(gray1_aligned, gray2).astype(float) / 255.0
    ssim_score, ssim_map = ssim(gray1_aligned, gray2, full=True)
    ssim_diff = 1 - ssim_map

    # Combine: more weight on edges for texture
    difference_map = (0.2 * pixel_diff +
                      0.2 * ssim_diff +
                      0.3 * edge_diff +
```

```
                    0.3 * density_diff)

    print("✅ Edge-enhanced difference computed")

else:
    # Standard method
    pixel_diff = cv2.absdiff(gray1_aligned, gray2).astype(float) / 255.0
    ssim_score, ssim_map = ssim(gray1_aligned, gray2, full=True)
    ssim_diff = 1 - ssim_map
    difference_map = (pixel_diff + ssim_diff) / 2

    print(f"✅ Standard difference computed (SSIM: {ssim_score:.3f})")
```

```
🔍 Using edge detection for texture analysis...
✅ Edge-enhanced difference computed
```

In [23]:

```
# ============================================================================
# CELL 8: Find Changed Regions
# ============================================================================

# Threshold
sensitivity = CONFIG['sensitivity']
binary = (difference_map > sensitivity).astype(np.uint8) * 255

# Clean up noise
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
binary = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
binary = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)

# Find contours
contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Get bounding boxes
changes = []
for contour in contours:
    area = cv2.contourArea(contour)
    if area > 100:
        x, y, w, h = cv2.boundingRect(contour)
        aspect_ratio = w / h if h > 0 else 0
        changes.append({
            'bbox': (x, y, w, h),
            'area': area,
            'aspect_ratio': aspect_ratio
        })

changes.sort(key=lambda c: c['area'], reverse=True)

print(f"✅ Found {len(changes)} potential changes")
```

```
✅ Found 19 potential changes
```

In [24]:

```
# ============================================================================
# CELL 9: Optional - Filter Text Regions
# ============================================================================

def is_text_region(change):
```

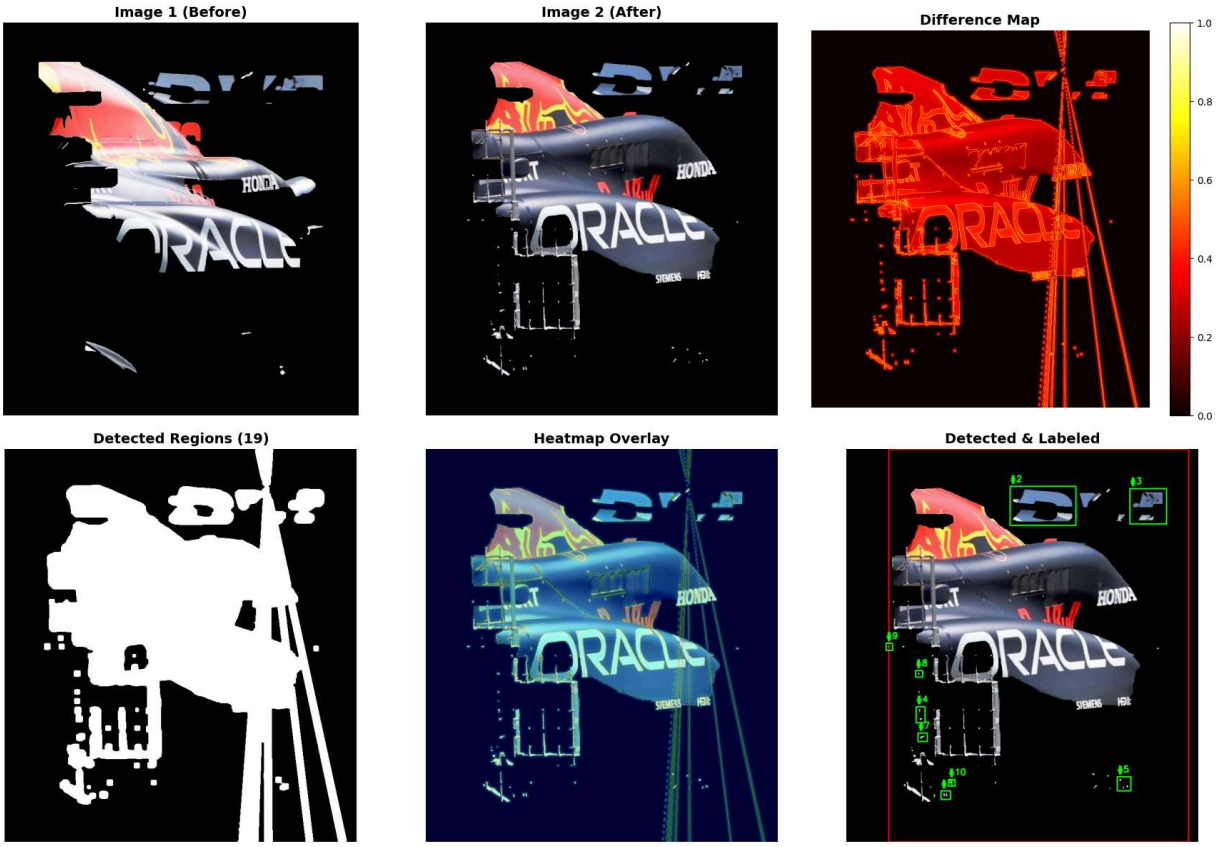
```
    """
    Heuristic to detect if region is likely text

    Text characteristics:
    - Elongated (high aspect ratio)
    - Small to medium size
    - Rectangular shape
    """
    area = change['area']
    aspect_ratio = change['aspect_ratio']

    # Text usually has aspect ratio > 2 or < 0.5
    is_elongated = aspect_ratio > 2.5 or aspect_ratio < 0.4

    # Text is usually small-medium size
    is_small_medium = 100 < area < 5000

    return is_elongated and is_small_medium

if CONFIG['filter_text_regions']:
    print("📝 Filtering text regions...")

    structural_changes = []
    text_regions = []

    for change in changes:
        if is_text_region(change):
            text_regions.append(change)
        else:
            structural_changes.append(change)

    print(f"   Structural changes: {len(structural_changes)}")
    print(f"   Text regions (filtered): {len(text_regions)}")

    # Use only structural changes
    changes = structural_changes
else:
    print(f"✅ Using all {len(changes)} detected changes")
```

```
📝 Filtering text regions...
  Structural changes: 19
  Text regions (filtered): 0
```

In [25]:

```
# ============================================================================
# CELL 10: Visualize Results
# ============================================================================

fig, axes = plt.subplots(2, 3, figsize=(18, 12))

# Row 1: Original images and difference
axes[0, 0].imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
axes[0, 0].set_title('Image 1 (Before)', fontsize=14, fontweight='bold')
axes[0, 0].axis('off')

axes[0, 1].imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
axes[0, 1].set_title('Image 2 (After)', fontsize=14, fontweight='bold')
```

```
axes[0, 1].axis('off')

im = axes[0, 2].imshow(difference_map, cmap='hot', vmin=0, vmax=1)
axes[0, 2].set_title('Difference Map', fontsize=14, fontweight='bold')
axes[0, 2].axis('off')
plt.colorbar(im, ax=axes[0, 2])

# Row 2: Binary, overlay, and annotated
axes[1, 0].imshow(binary, cmap='gray')
axes[1, 0].set_title(f'Detected Regions ({len(changes)})', fontsize=14, fontweight=
axes[1, 0].axis('off')

heatmap = cv2.applyColorMap((difference_map * 255).astype(np.uint8), cv2.COLORMAP_J
overlay = cv2.addWeighted(img2, 0.6, heatmap, 0.4, 0)
axes[1, 1].imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
axes[1, 1].set_title('Heatmap Overlay', fontsize=14, fontweight='bold')
axes[1, 1].axis('off')

result = img2.copy()
for i, change in enumerate(changes[:10]):
    x, y, w, h = change['bbox']
    color = (0, 0, 255) if i == 0 else (0, 255, 0)
    cv2.rectangle(result, (x, y), (x+w, y+h), color, 2)
    cv2.putText(result, f"#{i+1}", (x, y-10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

axes[1, 2].imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
axes[1, 2].set_title('Detected & Labeled', fontsize=14, fontweight='bold')
axes[1, 2].axis('off')

plt.tight_layout()
plt.show()

# Print statistics
print(f"\n📊 ANALYSIS RESULTS:")
print(f"   Changes detected: {len(changes)}")
print(f"   Sensitivity: {CONFIG['sensitivity']}")
print(f"   ROI mode: {'ON' if CONFIG['use_roi'] else 'OFF'}")
print(f"   Background removal: {'ON' if CONFIG['remove_background'] else 'OFF'}")
print(f"   Edge detection: {'ON' if CONFIG['use_edge_detection'] else 'OFF'}")
print(f"   Text filtering: {'ON' if CONFIG['filter_text_regions'] else 'OFF'}")

if changes:
    print(f"\n   Top 5 changes:")
    for i, change in enumerate(changes[:5]):
        x, y, w, h = change['bbox']
        print(f"   {i+1}. Location: ({x}, {y}), Size: {change['area']:.0f} px²")
```

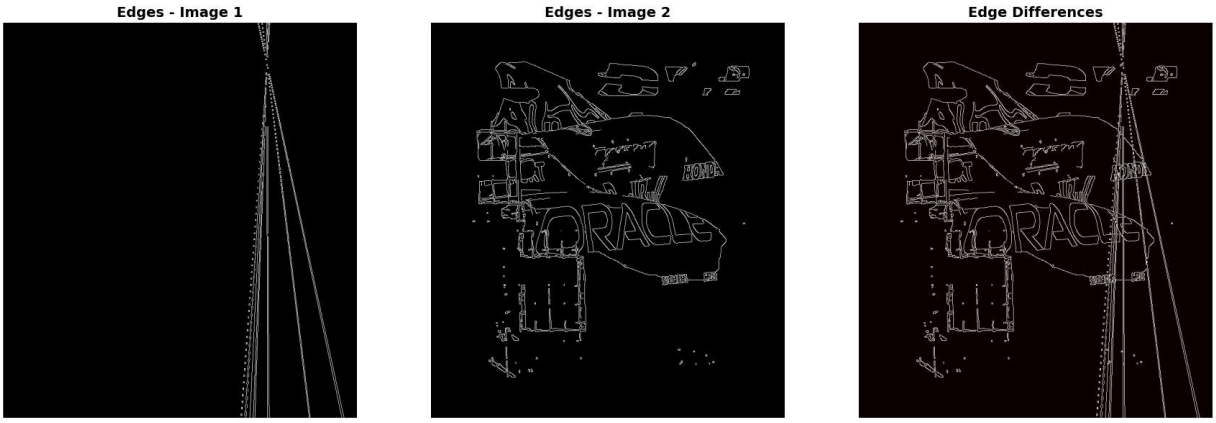


📊 ANALYSIS RESULTS:
Changes detected: 19
Sensitivity: 0.01
ROI mode: OFF
Background removal: ON
Edge detection: ON
Text filtering: ON

Top 5 changes:
1. Location: (90, 0), Size: 229050 px²
2. Location: (348, 79), Size: 9694 px²
3. Location: (603, 84), Size: 3751 px²
4. Location: (148, 548), Size: 460 px²
5. Location: (576, 697), Size: 414 px²

In [26]:
```
# ================================================================================
# CELL 11: Edge Detection Visualization (if enabled)
# ================================================================================

if CONFIG['use_edge_detection']:
    fig, axes = plt.subplots(1, 3, figsize=(18, 6))

    edges1 = cv2.Canny(gray1_aligned, 50, 150)
    edges2 = cv2.Canny(gray2, 50, 150)
    edge_diff = cv2.absdiff(edges1, edges2)

    axes[0].imshow(edges1, cmap='gray')
    axes[0].set_title('Edges - Image 1', fontsize=14, fontweight='bold')
    axes[0].axis('off')

    axes[1].imshow(edges2, cmap='gray')
```

```
    axes[1].set_title('Edges - Image 2', fontsize=14, fontweight='bold')
    axes[1].axis('off')

    axes[2].imshow(edge_diff, cmap='hot')
    axes[2].set_title('Edge Differences', fontsize=14, fontweight='bold')
    axes[2].axis('off')

    plt.tight_layout()
    plt.show()

    print("💡 Edge detection useful for:")
    print("   • Tire wear (texture changes)")
    print("   • Surface cracks/damage")
    print("   • Pattern modifications")
```




```
💡 Edge detection useful for:
   • Tire wear (texture changes)
   • Surface cracks/damage
   • Pattern modifications
```

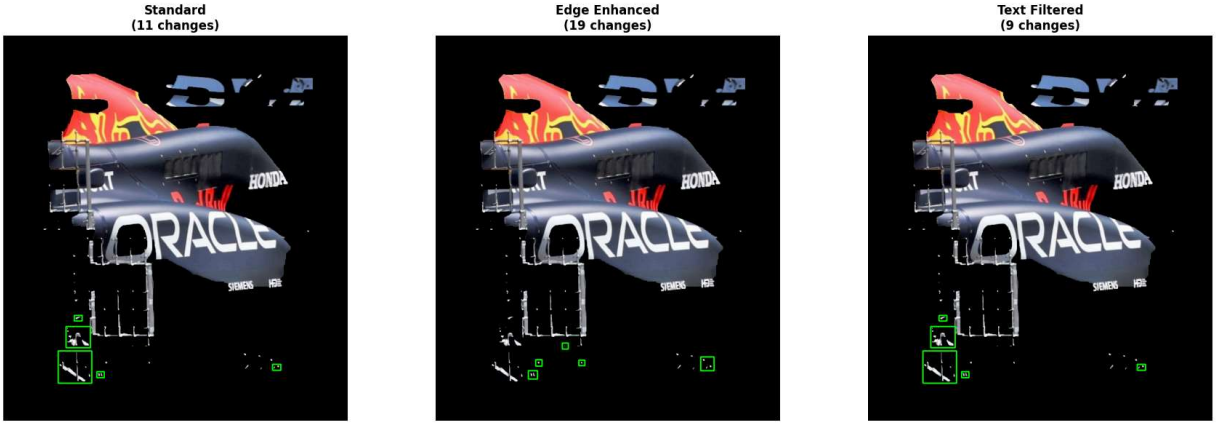
In [27]:

```
# ============================================================================
# CELL 12: Quick Test - Try Different Configurations
# ============================================================================

print("\n" + "="*70)
print("🧪 QUICK CONFIGURATION TESTS")
print("="*70)

test_configs = [
    {'name': 'Standard', 'use_edge_detection': False, 'filter_text_regions': False}
    {'name': 'Edge Enhanced', 'use_edge_detection': True, 'filter_text_regions': Fa
    {'name': 'Text Filtered', 'use_edge_detection': False, 'filter_text_regions': T
]

fig, axes = plt.subplots(1, len(test_configs), figsize=(18, 6))

for idx, test_cfg in enumerate(test_configs):
    # Update config temporarily
    old_edge = CONFIG['use_edge_detection']
    old_text = CONFIG['filter_text_regions']

    CONFIG['use_edge_detection'] = test_cfg['use_edge_detection']
    CONFIG['filter_text_regions'] = test_cfg['filter_text_regions']
```

```
    # Recompute difference map
    if CONFIG['use_edge_detection']:
        edges1 = cv2.Canny(gray1_aligned, 50, 150)
        edges2 = cv2.Canny(gray2, 50, 150)
        edge_diff = cv2.absdiff(edges1, edges2).astype(float) / 255.0

        kernel = np.ones((15, 15), np.float32) / 225
        edge_density1 = cv2.filter2D(edges1.astype(float) / 255.0, -1, kernel)
        edge_density2 = cv2.filter2D(edges2.astype(float) / 255.0, -1, kernel)
        density_diff = np.abs(edge_density1 - edge_density2)

        pixel_diff = cv2.absdiff(gray1_aligned, gray2).astype(float) / 255.0
        _, ssim_map = ssim(gray1_aligned, gray2, full=True)
        ssim_diff = 1 - ssim_map

        test_diff = (0.2 * pixel_diff + 0.2 * ssim_diff +
                     0.3 * edge_diff + 0.3 * density_diff)
    else:
        pixel_diff = cv2.absdiff(gray1_aligned, gray2).astype(float) / 255.0
        _, ssim_map = ssim(gray1_aligned, gray2, full=True)
        test_diff = (pixel_diff + (1 - ssim_map)) / 2

    # Find changes
    test_binary = (test_diff > CONFIG['sensitivity']).astype(np.uint8) * 255
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    test_binary = cv2.morphologyEx(test_binary, cv2.MORPH_CLOSE, kernel)
    test_contours, _ = cv2.findContours(test_binary, cv2.RETR_EXTERNAL, cv2.CHAIN_A

    test_changes = []
    for c in test_contours:
        area = cv2.contourArea(c)
        if area > 100:
            x, y, w, h = cv2.boundingRect(c)
            aspect_ratio = w / h if h > 0 else 0
            test_changes.append({'bbox': (x,y,w,h), 'area': area, 'aspect_ratio': a

    # Filter text if enabled
    if CONFIG['filter_text_regions']:
        test_changes = [c for c in test_changes if not is_text_region(c)]

    # Visualize
    test_result = img2.copy()
    for i, change in enumerate(test_changes[:5]):
        x, y, w, h = change['bbox']
        cv2.rectangle(test_result, (x, y), (x+w, y+h), (0, 255, 0), 2)

    axes[idx].imshow(cv2.cvtColor(test_result, cv2.COLOR_BGR2RGB))
    axes[idx].set_title(f"{test_cfg['name']}\n({len(test_changes)} changes)",
                        fontsize=12, fontweight='bold')
    axes[idx].axis('off')

    # Restore config
    CONFIG['use_edge_detection'] = old_edge
    CONFIG['filter_text_regions'] = old_text

plt.tight_layout()
```

```
plt.show()

print("\n💡 Compare configurations to see which works best for your images!")
```

```
======================================================================
🧪 QUICK CONFIGURATION TESTS
======================================================================
```




```
💡 Compare configurations to see which works best for your images!
```

In [28]:

```
# ============================================================================
# CELL 13: Usage Guide
# ============================================================================

print("\n" + "="*70)
print("📖 FRAMESHIFT V1.1 - USAGE GUIDE")
print("="*70)

print("""
🎯 FOR MCLAREN IMAGE (Different POV):
   CONFIG['use_roi'] = True
   CONFIG['remove_background'] = True
   → Run Cell 4 to select just the halo area
   → Ignores different backgrounds

🛞 FOR TIRE WEAR IMAGE:
   CONFIG['use_edge_detection'] = True
   CONFIG['remove_background'] = True
   → Detects texture changes (wear patterns)
   → Removes busy pit lane background

🏎️ FOR RED BULL SIDEPOD (Best case):
   CONFIG['filter_text_regions'] = True
   → Everything else default
   → Filters out year labels
   → Focuses on structural changes

⚙️ GENERAL TIPS:
   • Start with all options OFF
   • Enable one at a time to see effect
   • Adjust sensitivity (0.10-0.20 range)
   • Use ROI for misaligned images
   • Use edges for texture/wear analysis
```

```
🚀 MODIFY CONFIG IN CELL 2, THEN RE-RUN FROM CELL 3!
""")
```

```
================================================================================
📖 FRAMESHIFT V1.1 - USAGE GUIDE
================================================================================

🎯 FOR MCLAREN IMAGE (Different POV):
   CONFIG['use_roi'] = True
   CONFIG['remove_background'] = True
   → Run Cell 4 to select just the halo area
   → Ignores different backgrounds

🛞 FOR TIRE WEAR IMAGE:
   CONFIG['use_edge_detection'] = True
   CONFIG['remove_background'] = True
   → Detects texture changes (wear patterns)
   → Removes busy pit lane background

🏎️ FOR RED BULL SIDEPOD (Best case):
   CONFIG['filter_text_regions'] = True
   → Everything else default
   → Filters out year labels
   → Focuses on structural changes

⚙️ GENERAL TIPS:
   • Start with all options OFF
   • Enable one at a time to see effect
   • Adjust sensitivity (0.10-0.20 range)
   • Use ROI for misaligned images
   • Use edges for texture/wear analysis

🚀 MODIFY CONFIG IN CELL 2, THEN RE-RUN FROM CELL 3!
```

In [29]:

```python
# ================================================================================
# DONE!
# ================================================================================

print("\n" + "="*70)
print("✅ FRAMESHIFT V1.1 - READY FOR F1 ANALYSIS!")
print("="*70)
print(f"\n📊 Final Results: {len(changes)} changes detected")
print(f"⚙️ Configuration summary:")
print(f"   • ROI: {'Enabled' if CONFIG['use_roi'] else 'Disabled'}")
print(f"   • Background removal: {'Enabled' if CONFIG['remove_background'] else 'Di
print(f"   • Edge detection: {'Enabled' if CONFIG['use_edge_detection'] else 'Disab
print(f"   • Text filtering: {'Enabled' if CONFIG['filter_text_regions'] else 'Disa
```

```
======================================================================
✅ FRAMESHIFT V1.1 - READY FOR F1 ANALYSIS!
======================================================================

📊 Final Results: 19 changes detected
⚙️ Configuration summary:
   • ROI: Disabled
   • Background removal: Enabled
   • Edge detection: Enabled
   • Text filtering: Enabled
```

In [30]:

```
# ==============================================================================
# END
# ==============================================================================
```