VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Shaahid Ahmed N
RegNo.: 21BAI1087
Course & Course Code: Machine Vision Lab & BCSE417P
Lab & Date: Assesment-3 & 09-11-2024
Slot: L43+L44

By turning in this assignment, I agree and declare that all of this is my own work.

Imports

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

Load the video

```
video_path = '/content/example1.mp4'
cap = cv2.VideoCapture(video_path)

if not cap.isOpened():
    print("Error: Could not open video.")
else:
    print("Video loaded successfully!")
```

```
Video loaded successfully!
```

Frame Extraction

```
# Directory to save the extracted frames
output_folder = 'extracted_frames'
os.makedirs(output_folder, exist_ok=True)
```

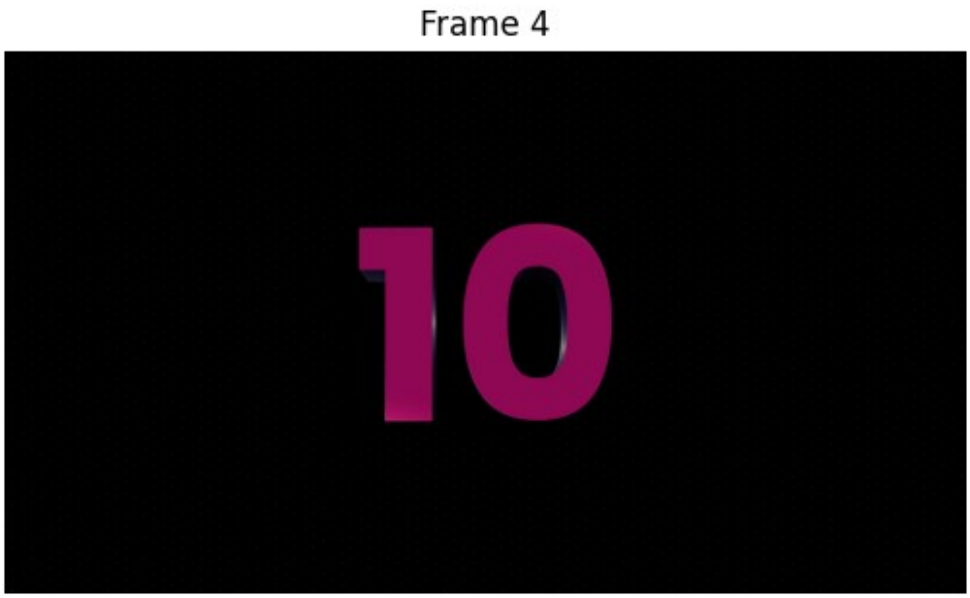
```
frame_count = 0

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Convert the frame from BGR to RGB for Matplotlib
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Display the frame using Matplotlib
    plt.imshow(frame_rgb)
    plt.title(f'Frame {frame_count}')
    plt.axis('off')
    plt.show()

    # Construct the frame filename
    frame_filename = os.path.join(output_folder,
f'frame_{frame_count:04d}.jpg')

    # Save the frame as an image
    cv2.imwrite(frame_filename, frame)

    frame_count += 1

    # Display only the first 5 frames
    if frame_count == 5:
        break

cap.release()
cv2.destroyAllWindows()

print(f"Total {frame_count} frames extracted to '{output_folder}'")
```

Frame 0

Frame 1

Frame 2

Frame 3




```
Total 5 frames extracted to 'extracted_frames'
```

Spatio-Temporal Segmentation

```
frame_count = 0
cap = cv2.VideoCapture(video_path)
# Parameters for optical flow
params = dict(
    pyr_scale=0.5, levels=3, winsize=15, iterations=3, poly_n=5,
poly_sigma=1.2, flags=0
)

ret, first_frame = cap.read()
# Convert first frame to grayscale for optical flow calculation
prev_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Convert frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Apply edge detection
    edges = cv2.Canny(gray, threshold1=100, threshold2=200)

    # Display the original frame and edge-detected frame side by side
    plt.figure(figsize=(12, 6))
```

```
    # Original frame display
    plt.subplot(1, 2, 1)
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    plt.imshow(frame_rgb)
    plt.title('Original Frame')
    plt.axis('off')

    # Edge-detected frame display
    plt.subplot(1, 2, 2)
    plt.imshow(edges, cmap='gray')
    plt.title('Edge Detection')
    plt.axis('off')

    plt.show()

    # Optical flow to track motion
    flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None,
**params)
    mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])

    # Threshold for identifying foreground motion
    mask = (mag > 2).astype(np.uint8) * 255

    # Display the motion mask (foreground regions)
    plt.figure(figsize=(6, 6))
    plt.imshow(mask, cmap='gray')
    plt.title('Foreground Motion Mask')
    plt.axis('off')
    plt.show()

    # Update previous frame
    prev_gray = gray

    frame_count += 1

    # Stop after processing a few frames
    if frame_count == 5:
        break

cap.release()
cv2.destroyAllWindows()
```

Original Frame

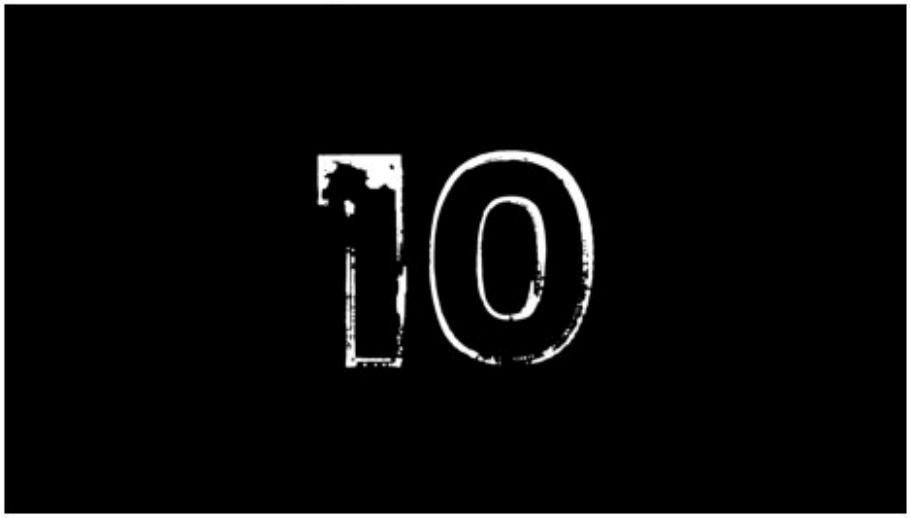
Edge Detection

Foreground Motion Mask

Original Frame

Edge Detection

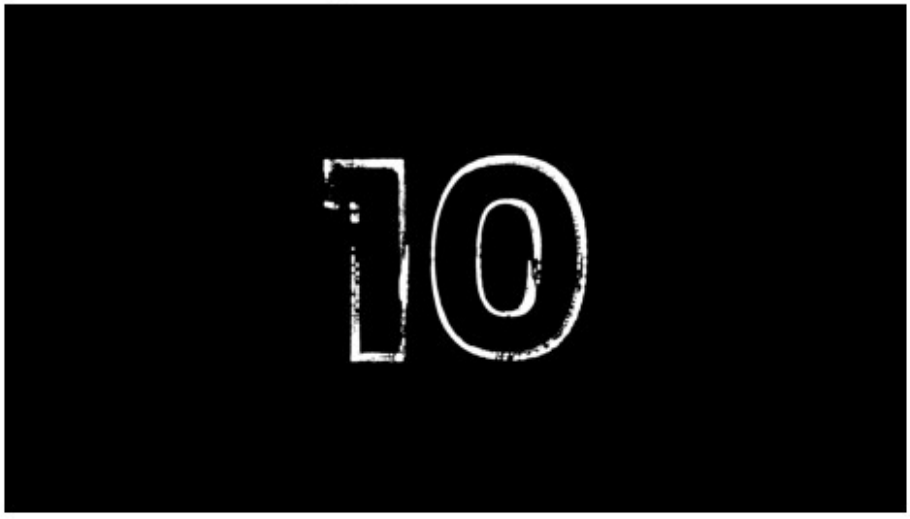
Foreground Motion Mask



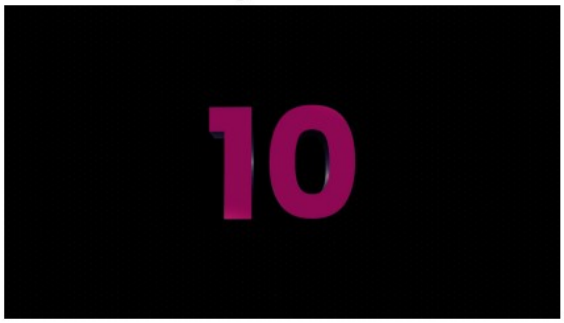
Original Frame

Edge Detection

Foreground Motion Mask

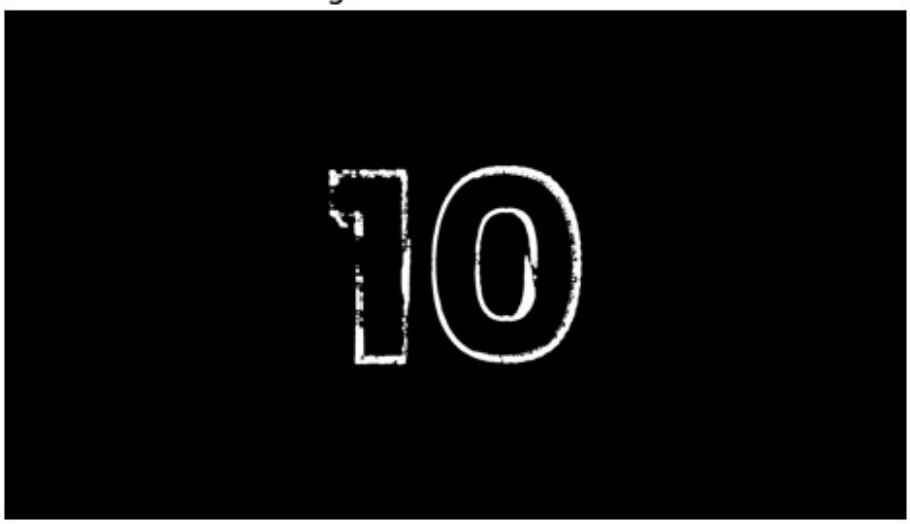


Original Frame



Edge Detection

Foreground Motion Mask



Original Frame

Edge Detection

Scene Cut Detection

```
cap = cv2.VideoCapture(video_path)
# Parameters for detection
hard_cut_threshold = 0.5  # Histogram difference threshold for hard
cuts
soft_cut_intensity_threshold = 5.0  # Intensity difference threshold
for soft cuts

frame_count = 0
scene_cuts = []
intensity_changes = []

# Read the first frame and calculate its histogram and intensity
ret, prev_frame = cap.read()
if not ret:
    print("Error: Could not read the first frame.")
    cap.release()
    exit()

# Convert to grayscale for initial processing
prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
prev_hist = cv2.calcHist([prev_gray], [0], None, [256], [0, 256])
prev_hist = cv2.normalize(prev_hist, prev_hist).flatten()
prev_intensity = np.mean(prev_gray)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break  # Exit when there are no more frames
```

```
    # Convert current frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Calculate current histogram and normalize
    hist = cv2.calcHist([gray_frame], [0], None, [256], [0, 256])
    hist = cv2.normalize(hist, hist).flatten()

    # Calculate histogram difference for hard cut detection
    hist_diff = cv2.compareHist(prev_hist, hist,
cv2.HISTCMP_BHATTACHARYYA)

    # Calculate intensity change for soft cut detection
    current_intensity = np.mean(gray_frame)
    intensity_diff = abs(current_intensity - prev_intensity)
    intensity_changes.append(intensity_diff)

    # Detect hard cut
    if hist_diff > hard_cut_threshold:
        print(f"Hard cut detected at frame {frame_count}")
        scene_cuts.append((frame_count, 'hard'))

    # Detect potential soft cut based on intensity changes
    if intensity_diff > soft_cut_intensity_threshold:
        print(f"Soft cut detected at frame {frame_count}")
        scene_cuts.append((frame_count, 'soft'))

    # Update previous frame's histogram and intensity for the next
iteration
    prev_hist = hist
    prev_intensity = current_intensity

    frame_count += 1

cap.release()

# Plot the detected scene cuts
hard_cut_frames = [frame for frame, cut_type in scene_cuts if cut_type
== 'hard']
soft_cut_frames = [frame for frame, cut_type in scene_cuts if cut_type
== 'soft']

plt.figure(figsize=(15, 5))
plt.plot(hard_cut_frames, [1] * len(hard_cut_frames), 'ro',
label='Hard Cuts')
plt.plot(soft_cut_frames, [2] * len(soft_cut_frames), 'bo',
label='Soft Cuts')
plt.title('Scene Cut Detection')
plt.xlabel('Frame Number')
plt.ylabel('Scene Change Type')
plt.yticks([1, 2], ['Hard Cut', 'Soft Cut'])
```

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

print(f"Total hard cuts detected: {len(hard_cut_frames)}")
print(f"Total soft cuts detected: {len(soft_cut_frames)}")
```

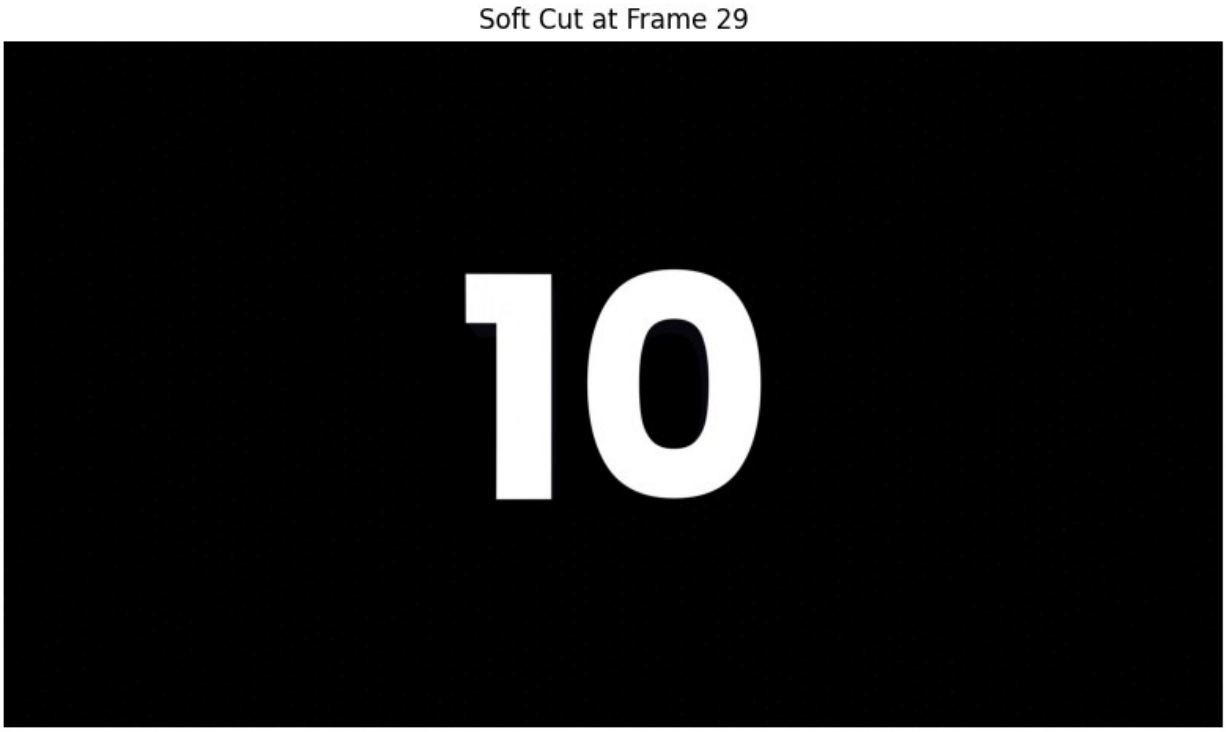
```
Soft cut detected at frame 29
Hard cut detected at frame 65
Hard cut detected at frame 66
Soft cut detected at frame 89
Hard cut detected at frame 209
Hard cut detected at frame 239
Hard cut detected at frame 270
Hard cut detected at frame 271
```


```
Total hard cuts detected: 6
Total soft cuts detected: 2
```

Mark Scene Cut

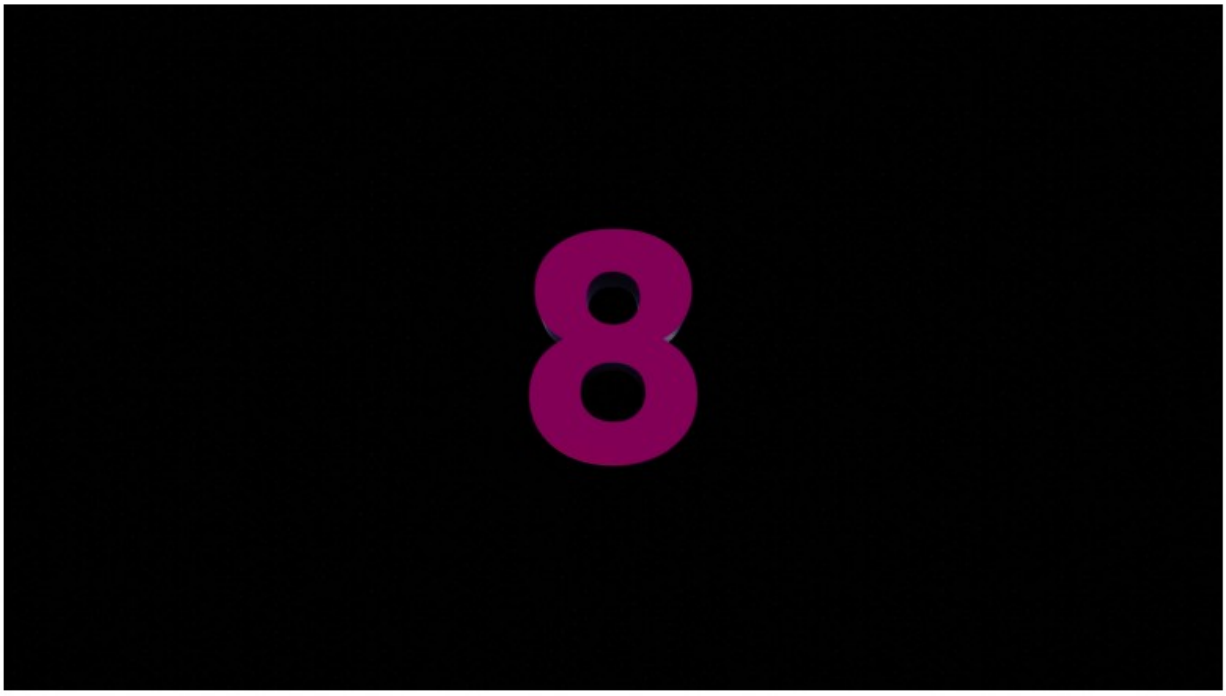
```
def display_frame(frame, title='Frame'):
    plt.figure(figsize=(10, 6))
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
    plt.show()

# Path to the video file
cap = cv2.VideoCapture(video_path)

# Display frames at detected scene cuts
for frame_num, cut_type in scene_cuts:
    cap.set(cv2.CAP_PROP_POS_FRAMES, frame_num)  # Move to the frame
```

```
position
    ret, frame = cap.read()
    if ret:
        display_frame(frame, title=f'{cut_type.capitalize()} Cut at
Frame {frame_num}')
    else:
        print(f"Error reading frame {frame_num}")

cap.release()
```

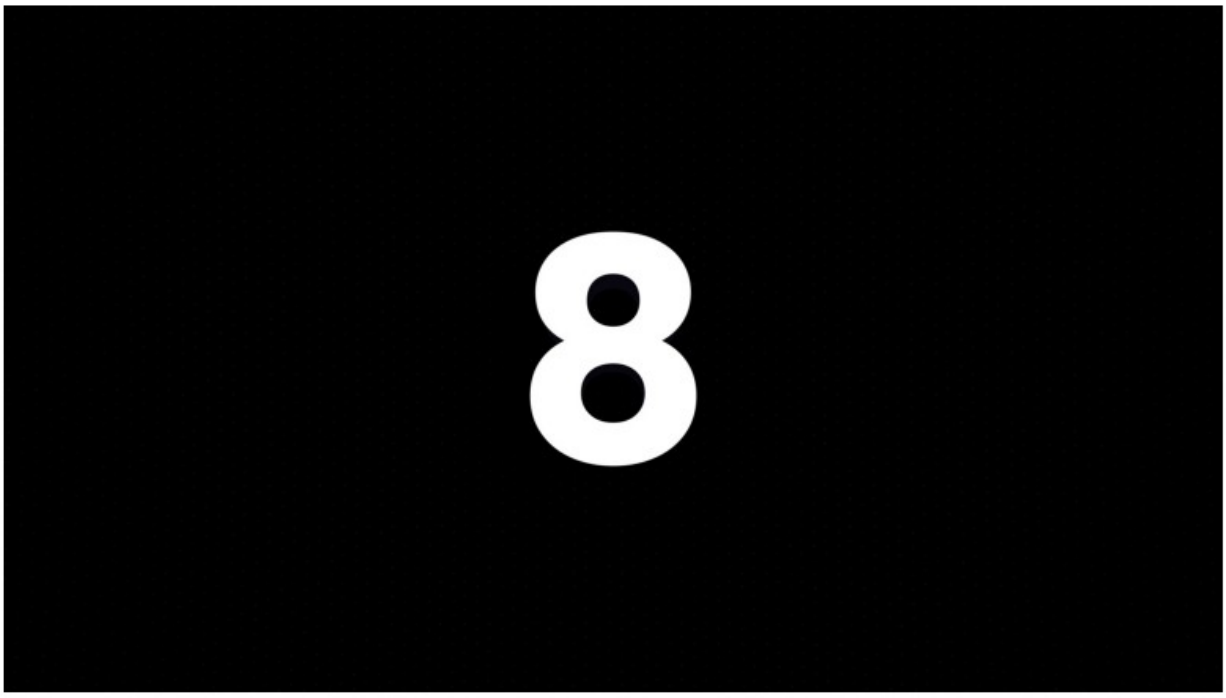


Hard Cut at Frame 65

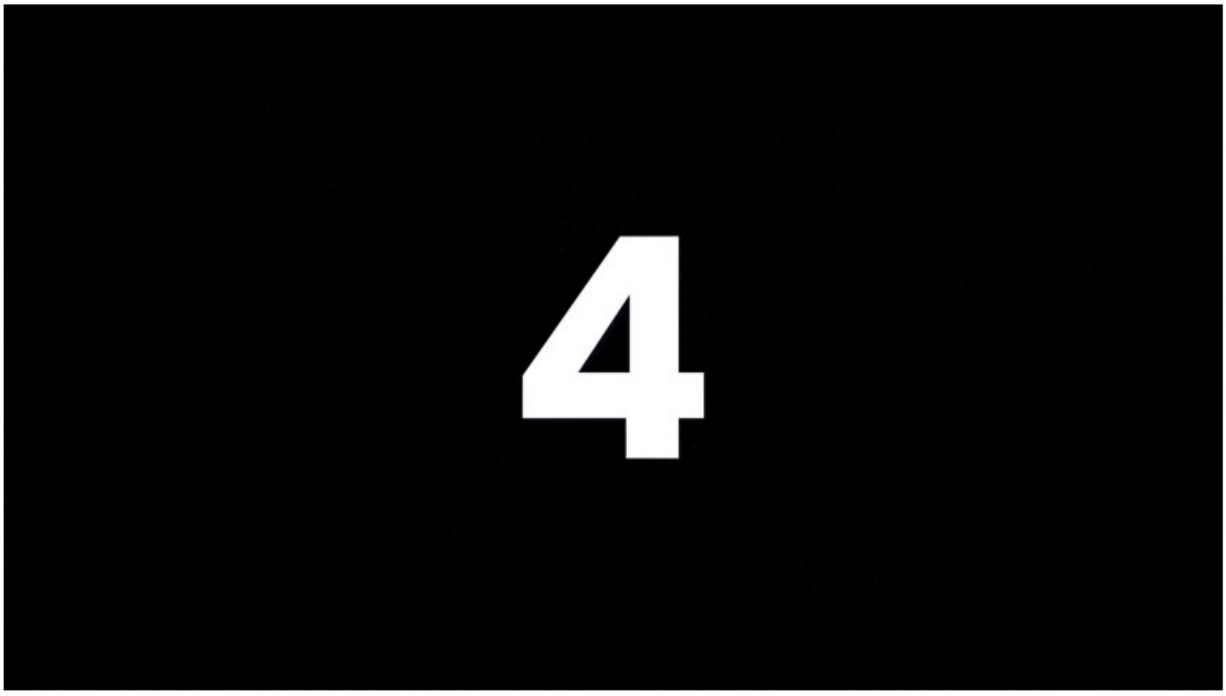
Hard Cut at Frame 66



Soft Cut at Frame 89

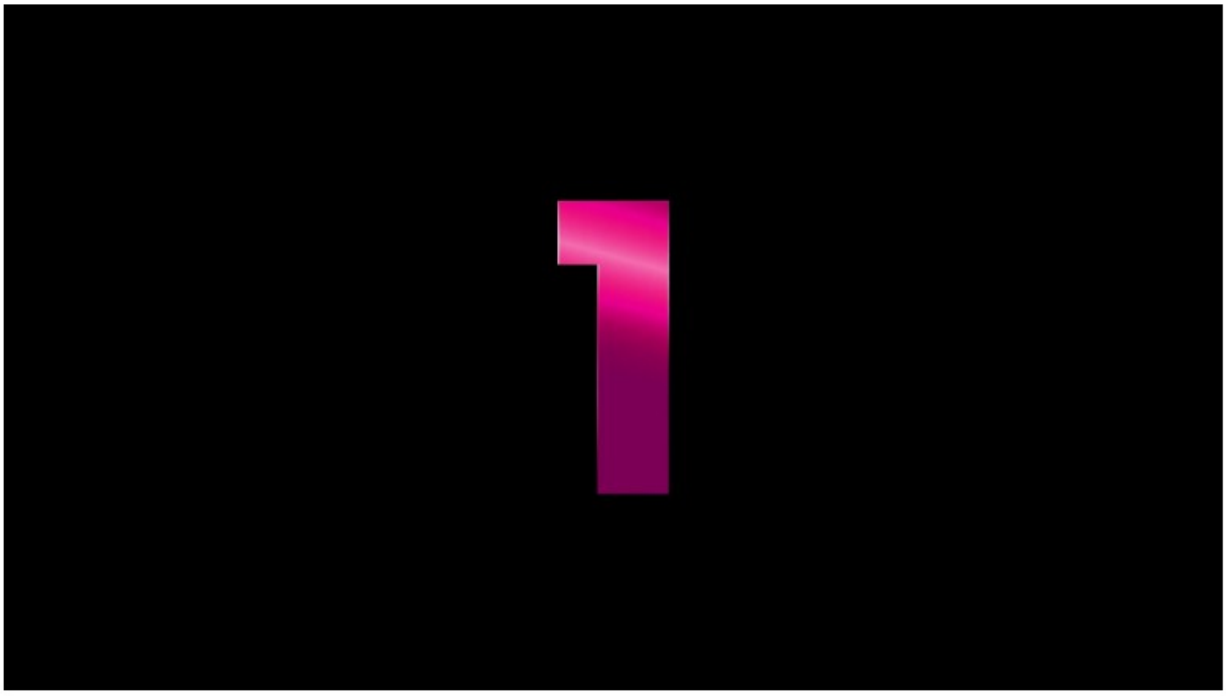


Hard Cut at Frame 209



Hard Cut at Frame 239

Hard Cut at Frame 270

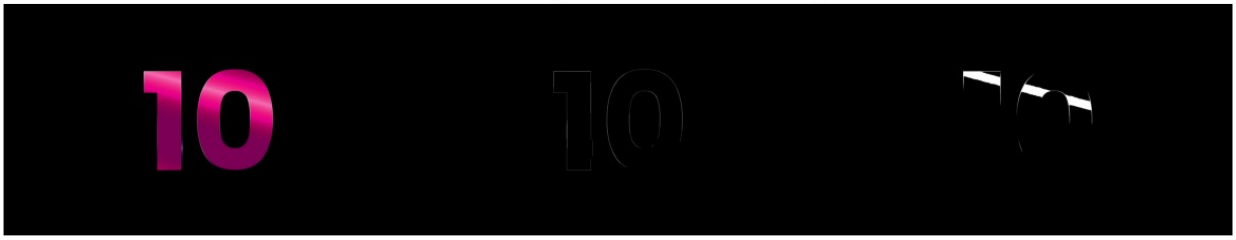


Result Visualization

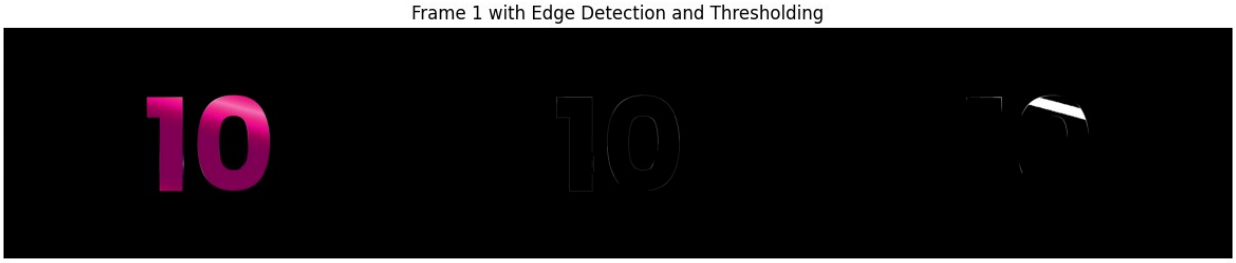
```
# Function to apply edge detection (Canny) and thresholding on a frame
def perform_edge_detection_and_thresholding(frame):
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Apply the Canny edge detection
    edges = cv2.Canny(gray, 100, 200)  # Threshold values can be
adjusted

    # Apply binary thresholding on the grayscale image
    _, thresholded = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    # Convert edges and thresholded image to 3-channel images for
overlaying
    edges_colored = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
    thresholded_colored = cv2.cvtColor(thresholded,
cv2.COLOR_GRAY2BGR)

    return edges_colored, thresholded_colored

# Function to display a frame with edge detection and thresholding
results
def display_frame_with_results(frame, edges_colored,
thresholded_colored, title='Frame with Edge Detection and
```

```
Thresholding'):
    # Convert the original frame to RGB for matplotlib
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Create side-by-side comparison of original frame, edge
detection, and thresholding
    combined = cv2.hconcat([frame_rgb, edges_colored,
thresholded_colored])

    # Display the result
    plt.figure(figsize=(15, 6))
    plt.imshow(combined)
    plt.title(title)
    plt.axis('off')
    plt.show()

# Path to the video file
cap = cv2.VideoCapture('/content/example1.mp4')  # Replace with your
video file path

# Check if video opened successfully
if not cap.isOpened():
    print("Error: Unable to open video file.")
else:
    # Process video frames (example for the first 10 frames)
    for frame_num in range(10):
        ret, frame = cap.read()

        if ret:
            # Apply edge detection and thresholding on the current
frame
            edges_colored, thresholded_colored =
perform_edge_detection_and_thresholding(frame)

            # Display the frame with edge detection and thresholding
results
            display_frame_with_results(frame, edges_colored,
thresholded_colored, title=f'Frame {frame_num} with Edge Detection and
Thresholding')
        else:
            print(f"Error reading frame {frame_num}")
            break

# Release the video capture object
cap.release()
```

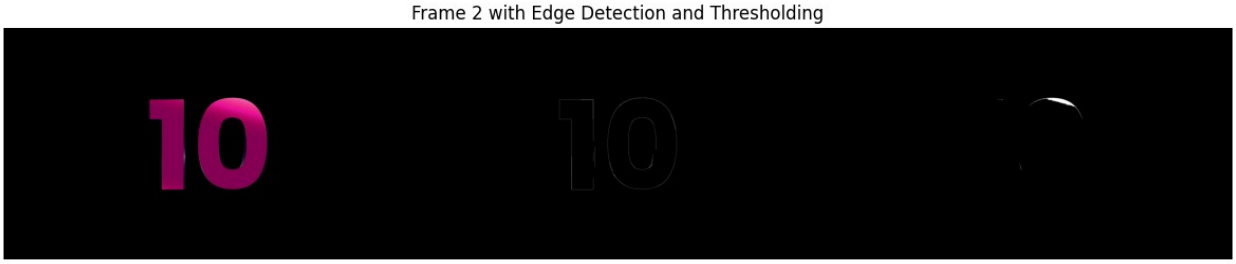
Frame 0 with Edge Detection and Thresholding



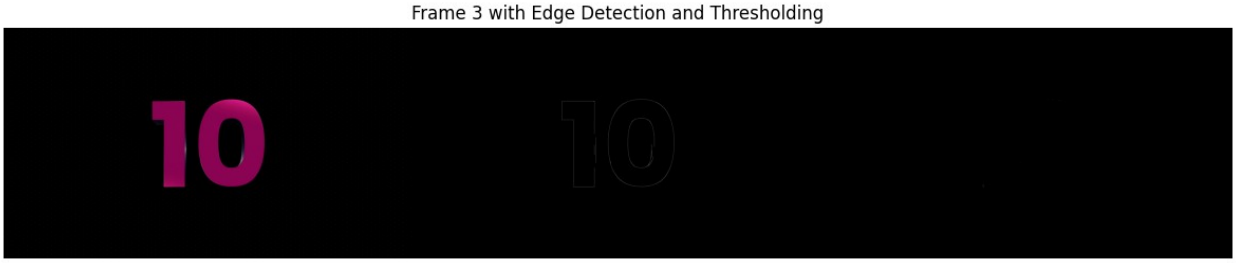
Frame 1 with Edge Detection and Thresholding



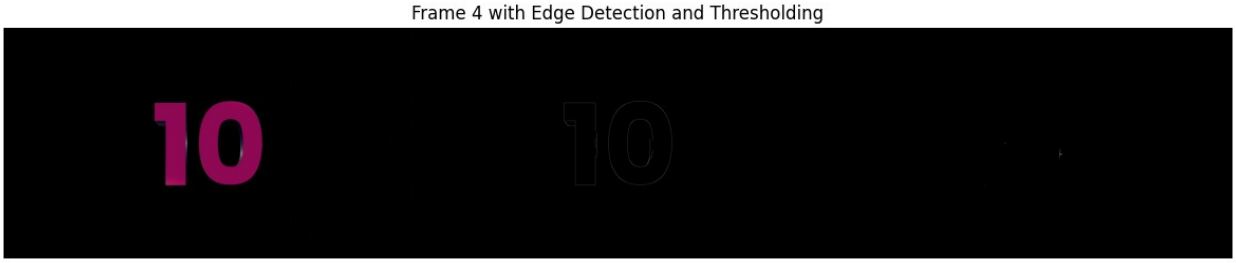
Frame 2 with Edge Detection and Thresholding



Frame 3 with Edge Detection and Thresholding

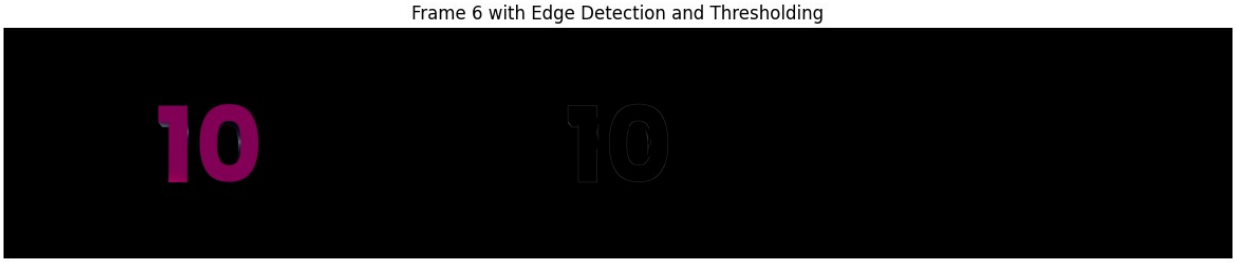


Frame 4 with Edge Detection and Thresholding

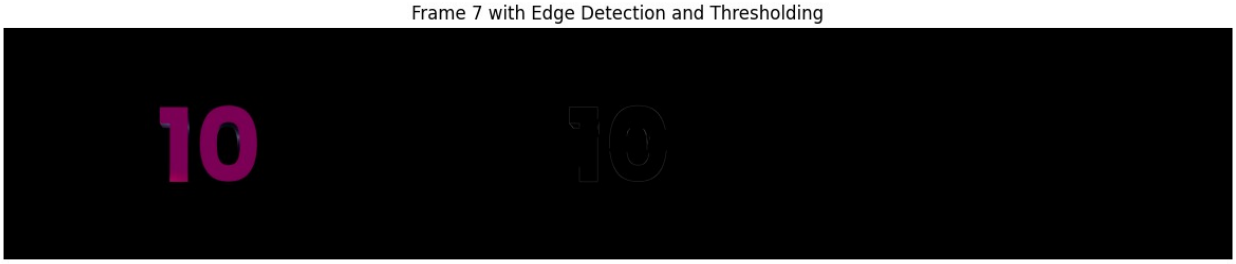


Frame 5 with Edge Detection and Thresholding

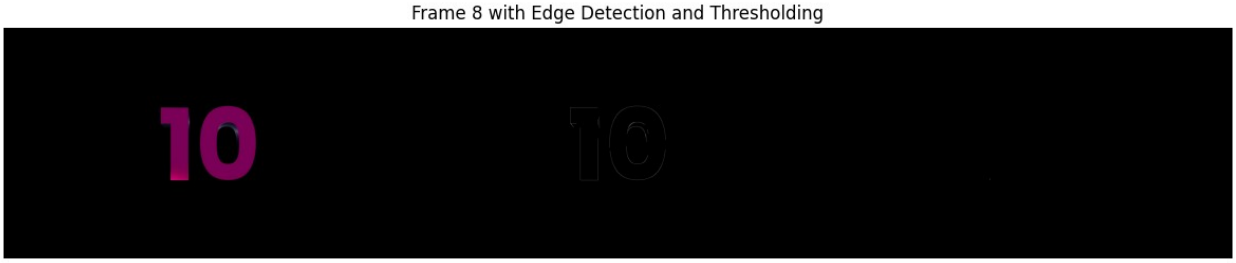
Frame 6 with Edge Detection and Thresholding



Frame 7 with Edge Detection and Thresholding



Frame 8 with Edge Detection and Thresholding



Frame 9 with Edge Detection and Thresholding