Weekly Report: Multi-View 2D Image-Based 3D Modeling of Objects

Project Number: 18

1 Introduction

This week, we focused on understanding **silhouettes** and their applications in 3D reconstruction. A silhouette is a binary image representing the outline of an object. It captures the shape of the object while ignoring color or texture information. Silhouettes are fundamental for visual hull reconstruction, where multiple silhouettes from different angles are combined to approximate a 3D object.

2 Methodology

The process of extracting silhouettes involved several key methods. Each method is explained in detail below.

2.1 Region of Interest (ROI) Selection

ROI selection is the process of selecting the area of an image that contains the object of interest and ignoring the background. **Implementation in Python:** We use OpenCV's interactive function `cv2.selectROI`, which opens the first image and allows the user to draw a rectangle around the object. The selected coordinates (x, y, width, height) are then applied to crop all images in the dataset.

Learning outcome:

- ROI reduces computational load by processing only relevant pixels.
- Cropping improves the quality of silhouettes by removing background noise.
- This method can be applied manually (interactive) or programmatically if coordinates are known.

2.2 Grayscale Conversion and Image Loading

BMP images are loaded and converted to grayscale because silhouette extraction relies on intensity, not color. **Implementation in Python:** We use `PIL.Image.open(path).convert('L')` to convert each image to a 2D array representing pixel intensity.

2.3 Binarization (Thresholding)

Binarization converts a grayscale image into a binary image where object pixels are 1 and background pixels are 0.

Implementation in Python:

```
binarized = (image < threshold).astype(np.uint8)
```

Pixels with intensity below a chosen threshold (e.g., 128) are considered part of the object.

Learning outcome:

- Separates object from background efficiently.
- Simple thresholding is fast and works well if object intensity is distinct from background.

2.4 Hole Filling / Gap Filling

Sometimes the binary silhouette may have small gaps or holes inside the object due to uneven illumination or noise. Hole filling ensures the silhouette is a solid object without gaps.

Implementation in Python:

```
from scipy.ndimage import binary_fill_holes
filled = binary_fill_holes(binarized).astype(np.uint8)
```

How it works:

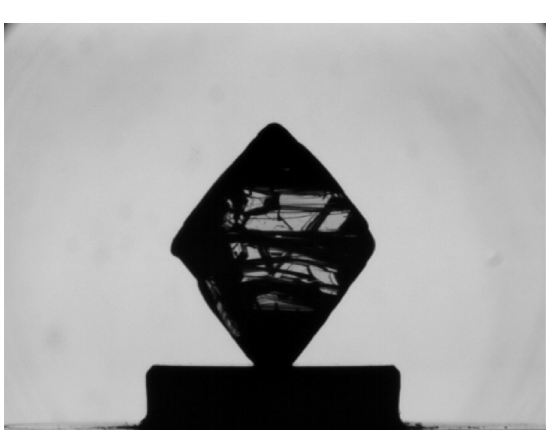
- The algorithm identifies connected regions of zeros (background) that are completely surrounded by ones (object).
- These internal zero pixels are then converted to ones, effectively filling holes.
- This ensures the silhouette represents the full shape without missing areas.

2.5 Silhouette Storage

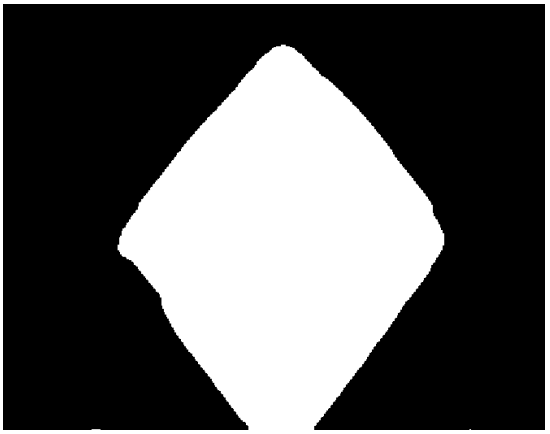
After processing, silhouettes are saved as PNG images for later analysis or 3D reconstruction. **Implementation:** Each binary array is multiplied by 255 and saved using `PIL.Image.fromarray`.

3 Results

The figure below shows a comparison between the original BMP image and the generated silhouette.



(a) Original BMP image



(b) Generated Silhouette

Figure 1: Side-by-side comparison of original image and generated silhouette.

4 Conclusion

This week, we learned and implemented:

- How to select ROI and why focusing on the object improves results.
- How to convert BMP images to grayscale for intensity-based processing.
- How to binarize images using thresholding to separate object from background.
- How hole filling ensures clean and complete silhouettes.
- How to save silhouettes for further processing in 3D reconstruction.

Understanding these processes gives us a strong foundation for building accurate visual hulls in future work.