3D Reconstruction of fly photoreceptors

Kunal Jain and Kiran Sandilya
{kunal.jain001},{kiran.sandilya001}@umb.edu
University of Massachusetts Boston

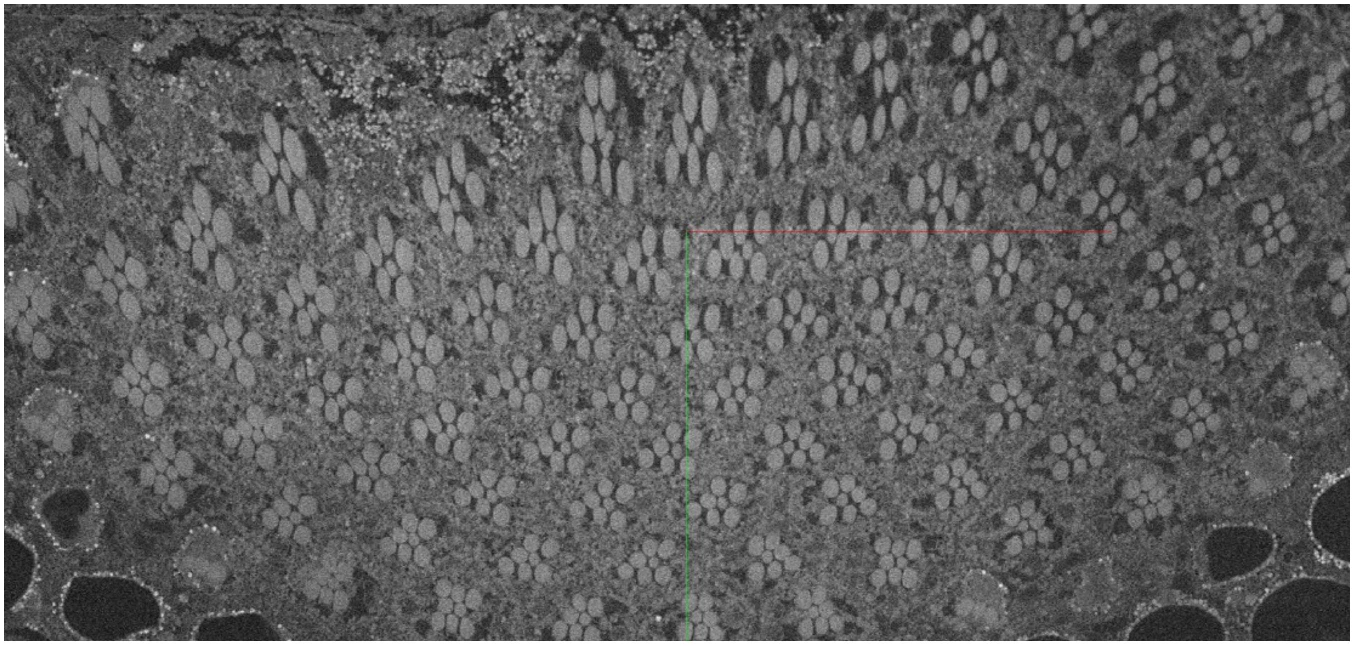


Figure 1: Hexagonal open Rhabdomere structure(7 facets).

ABSTRACT

In this project our objective is to study about the Drosophila eyes which are a useful model for studying the principles of vision and the neural basis of perception due to its relatively simple nervous system and genetic tractability.

KEYWORDS

Drosophila, Photoreceptor, Rhabdomeres, Visualizing, Neuroglancer, Neuroglancer Volumes, Python, Segmentation, Normalization, Thresholding, Regions, 3D Mesh

ACM Reference Format:
Kunal Jain and Kiran Sandilya. 2022. 3D Reconstruction of fly photoreceptors. In *CS460: Computer Graphics at UMass Boston, Fall 2022*. Boston, MA, USA, 7 pages. https://CS460.org

1 INTRODUCTION

Drosophila, or fruit flies, have a pair of simple eyes called ommatidia that are similar in some ways to the eyes of humans and other animals. Drosophila and human eyes have a lens that focuses light onto a light-sensitive layer of cells called the retina.

The main visual structure of an insect is the "COMPOUND EYE" (Mosaic vision). This compound eye consists of around 100-1000 tiny facets called Ommatidium.

Ommatidium is an insect's eye's basic structural and functional unit, divided into two parts. The light gathering part and the Light detecting part. The light-gathering part consists of cornea and crystalline cones responsible for gathering the light and channelizing them into the light-detecting part.

In humans, the retina consists of photoreceptor cells called rods and cones, which also contain pigments that absorb light and trigger a neural response. In Drosophila, the retina comprises photoreceptor cells (light-gathering part) called rhabdomeres. Rhabdomeres contain pigments that absorb light and trigger a neural response. In the open structure, the Rhabdomeres are separated, forming a polygon pattern. In the close Rhabdomere structure, it fuses in the center. Flies have a hexagonal open Rhabdomere structure(7 facets). We are trying to understand these structures better through this project.

Drosophila and human eyes have a neural pathway that sends information about the visual scene to the brain for processing. In Drosophila, this pathway consists of a series of neurons called lamina neurons, which are connected to the photoreceptors and send information about the visual scene to higher brain centers. In humans, this pathway consists of the optic nerve, which carries information about the visual scene from the retina to the brain.

There are also some differences between the eyes of Drosophila and humans. For example, Drosophila ommatidia are much smaller and simpler than human eyes and do not have a complex structure like the human eye's iris and pupil. In addition, Drosophila eyes cannot focus on objects at different distances, as human eyes do. Despite these differences, Drosophila eyes are still a valuable model for studying the principles of vision and the neural basis of perception due to its relatively simple nervous system and genetic tractability.

3D reconstruction of Drosophila photoreceptors typically involves imaging the cells at multiple angles and using computer algorithms to combine the images into a single 3D model. The 3D model can provide detailed information about the shape and structure of the cells, which can be useful for understanding their function and role in the nervous system.

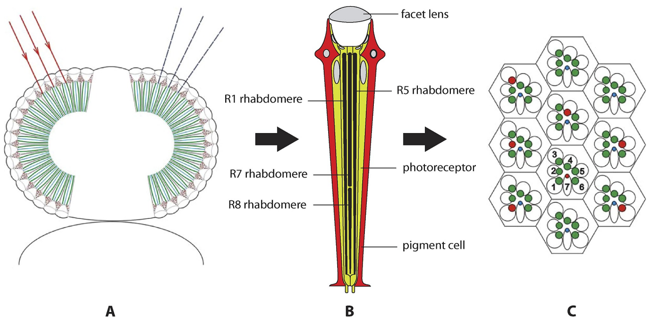


Figure 2: Image of rhabdomere

2 EXISTING WORK

The Drosophila visual system is a well-studied model system in neuroscience due to the high degree of genetic conservation between flies and mammals and the relatively simple and well-defined structure of the fly visual system.

One model where Drosophila visual system is well studied is the Connectome Based Hexagonal Lattice Convolutional Network Model of the Drosophila Visual System by Fabian David Tschoop and others - https://arxiv.org/pdf/1806.04793.pdf

A connectome-based hexagonal lattice convolutional network model is a computational model of the Drosophila visual system that aims to simulate the neural connections and processing within the fly's visual system.

In this model, the fly's visual system is represented as a hexagonal lattice, with each hexagonal unit representing a neural processing unit or "neuron." The connections between these units are based on the known anatomy of the fly visual system, including the connections between different layers of neurons and the connections between different visual areas of the brain.

The model uses convolutional neural network (CNN) techniques to simulate visual information processing within the visual system. CNNs are a type of neural network that is particularly well-suited to processing and recognizing image patterns.

The connectome-based hexagonal lattice convolutional network model can be used to study the neural basis of visual processing in the Drosophila visual system and may provide insights into the neural mechanisms underlying visual perception in other species, including humans.

3 METHOD

Neuroglancer is a web-based tool for visualizing and interacting with 3D imaging data, such as volumes and meshes, widely used in neuroscience and related fields. It allows users to view and explore large datasets in three dimensions and offers a range of features for navigating, annotating, and interacting with the data.

Researchers at Indiana University have used electron microscopy to image the photoreceptors of Drosophila. These images provide detailed information about the structure and organization of photoreceptors at the cellular level, including their shape, size, and ultrastructure.

Electron microscopy is a technique that uses a beam of electrons to create images of samples at high resolution and can be used to visualize the structure of cells and tissues in great detail. It is possible to use electron microscopy to image photoreceptors, the cells in the eye that detect light and play a crucial role in vision, in a variety of organisms, including flies.

It is evident from the dataset that the photoreceptors are organized into a regular, hexagonal array and that they are connected to other cells in the eye through thin, branching processes called dendrites. Along with photoreceptors, images also contain numerous organelles, including mitochondria, endoplasmic reticulum, and Golgi apparatus, which are involved in various cellular processes such as energy production and protein synthesis.

3.1 Implementation

3.1.1 Conversion of the data. The conversion of given raw data into neuroglancer-compatible data consists of 4 steps:

Step-1: Copying the images from the input folder into a temporary folder for safe conversion and not tampering with the original data

```
for image in images:
    new_path = only_images_path + "/" + image
    shutil.copy(path + "/" + image, new_path).
```

Step 2: Converting the copied image files, which are in TIFF format, into JPEG using the Imagemagick tool.

```
images = [f for f in sorted(os.listdir(only_images_path))
    if '.tif' in f.lower()]

for i in images:
    img_input_path = only_images_path + "/" + i
    img_output_path = jpeg_path + "/" +
        i.replace("tif", "jpeg")
```

Step-3: To read the info from .info of the image file to generate Neuroglancer JSON File.

```
with open(path + "/" + images[0]) as f:
    lines = f.readlines()
    da = [i for i in lines if "pixelsize"
        in i.lower()]
    la = [i for i in lines if "tif" in i.lower()]
    ofs = [i for i in lines if "offset" in i.lower()]
    offset = ofs[0].split()
    offset_a = offset[1]
```

Step 4: After reading the required data from .info files, we align the data with the standard Neuroglancer JSON format before starting the conversion.

```
counter = [f for f in sorted(os.listdir(jpeg_path))
    if ".jpeg" in f.lower()]
slices = len(counter)
image = mahotas.imread(jpeg_path + "/" + counter[0])
height = image.shape[0]
width = image.shape[1]
if image.ndim == 2:
    channels = 1 # grayscale
if image.ndim == 3:
    channels = image.shape[-1]
```

The Neuroglancer JSON file includes information about the volume data, such as the file format and location of the data, the dimensions and resolution of the volume, and any additional metadata associated with the data. It also includes information about how the data should be displayed and interacted with in Neuroglancer, such as the default views and layers and any custom annotations or labels that will be added to the data.

Example of Neuroglancer JSON file format -

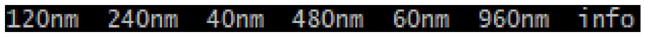
```
{
"type": "image",
"data_type": "uint8",
"num_channels": 1,
"encoding": "raw",
"resolution": [10, 10, 40],
"size": [6100, 4100, 800],
"voxel_offset": [0, 0, 0]
}
```

The Neuroglancer JSON file helps to load and display the volume data in Neuroglancer. Neuroglancer allows users to view and explore the data in three dimensions and to interact with and annotate the data using Neuroglancer's various tools and features.

The Neuroglancer script then reads the images and JSON files and converts them into a neuroglancer readable stack. A Neuroglancer stack is a collection of volumetric data that users can visualize and interact with in Neuroglancer. Neuroglancer supports several different data formats, including the Multiscale Volume Format (MVF), specifically designed for efficient visualization and exploration of large volumetric data sets. To convert our data into an MVF stack that Neuroglancer can read, we use the Neuroglancer Precomputed Python library. This library provides tools for creating and manipulating MVF stacks, as well as tools for visualizing the data in Neuroglancer.

```
os.system("generate-scales-info {}/data.json {}"
    .format(json_path, OUTFOLDER))
os.system("slices-to-precomputed --flat
    --input-orientation RPS {}{}"
    .format(jpeg_path, OUTFOLDER))
os.system("compute-scales --flat {}".format(OUTFOLDER))
```

The Multiscale Volume Format (MVF) is a file format developed specifically for efficient visualization and exploration of large volumetric data sets in Neuroglancer. MVF files are typically used to store and transfer volumetric data too large to be stored in memory or transmitted over the network simultaneously. MVF files consist of a hierarchy of scale levels, each representing a progressively downsampled data version. The highest scale level represents the full resolution of the data, while the lower scale levels represent progressively lower resolutions. Neuroglancer can access and visualize the data in an MVF file by loading only the scale levels needed to display the data at the desired resolution.



```
120nm  240nm  40nm  480nm  60nm  960nm  info
```

Figure 3: MVF files generated after a successful conversion.

3.1.2 Segmentation Pipeline. We started with preprocessing the data to segment photoreceptor cells in the fly EM images. Depending on the quality of the images, we applied various preprocessing techniques.
1. We first load the image using python's Mahotas image processing library.

```
z_img = mh.imread(os.path.join(DATA_DIR,img))
```

2. The image in our dataset is very high resolution. Each image slice is 6100 X 4000 px in dimensions. To segment photoreceptors, we first crop the image. The dimension that we crop the image to

is 1500 X 500 px.

```
z_img_cropped = z_img.copy()[1500:2000, 2500:4000]
```

3. Once cropped, we normalize all pixel values. Normalization involves scaling the values of the pixels in an image to a specific range, 0 to 1 in our case. We perform the Min-Max Normalization technique on the image. This method scales the values of the pixels so that the minimum value becomes 0 and the maximum value becomes 1. This can be done by subtracting the minimum value from all the pixels and then dividing by the range (max-min). After Normalization, we multiply all pixel values with 255.

```
# load images
slices = []
for z in range(601,603):
    img = '000000_000000_000'+str(z)+'_000000.tif'
    z_img = mh.imread(os.path.join(DATA_DIR,img))
    z_img_cropped = z_img[1500:2000, 2500:4000]
        .astype(np.float)
    z_img_cropped /= z_img_cropped.max()
    z_img_cropped *= 255
    slices.append(z_img_cropped)
```

4. As part of our segmentation pipeline, we first apply a Gaussian Filter on the image. A Gaussian filter is a low-pass filter commonly used in image processing to smooth images and reduce noise. It works by convolving the image with a kernel generated from a Gaussian function. We are using the gaussian filter function from the Mahotas library here.

5. After applying the Gaussian filter, we perform thresholding on the image. Thresholding involves dividing an image into two or more classes based on the intensity values of the pixels. It is used for segmenting images by creating a binary image that contains pixels with intensity values above a specified threshold value and pixels with intensity values below the threshold value. As evident from the shape of rods and cones, we apply Binary Thresholding on the image. Binary thresholding is the most basic form of thresholding. We divide the image into two classes: pixels with intensity values above the threshold and pixels with intensity values below the threshold. Pixels above the threshold are typically set to 1, and pixels below the threshold are set to 0. For this segmentation pipeline, we set Threshold value to 100. Every pixel below the intensity of 100 is set to 0, or the background, and all pixels above the intensity of 100 are set to 1.

6. After thresholding, we do region labeling, also known as Connected Component Analysis. The technique involves identifying and labeling distinct objects or regions in an image. We use the mahotas.label() function to label regions in the binary image. We pass the binary image as an input to the function. The function then scans the image and assigns a unique label to each connected region of pixels. A connected region is defined as a group of adjacent pixels that have the same intensity value (usually 1). The function returns a labeled image where each region is assigned a different intensity value.

7. Once labeled, we filter regions based on label size. Filtering involves selecting only certain regions in the image based on their size. This is often used to remove small noises or to focus on specific features in the image. We use the labeled image to measure the size of each region, and the labels are modified or removed based on the size of the region they correspond to. Mahotas provides a function labeled-size() function for filtering based on region size. The function takes an array of labels and returns an array of the same shape but with the labels replaced by the size of the region they correspond to. Once we have the array of region sizes, we use the remove-regions() function from the Mahotas library to remove regions less than a certain threshold value. For this segmentation pipeline, we set the threshold value to 1000. Every region below the size of 1000 pixels is removed from the labeled image.

8. We then create a binary mask using the labeled image. For creating a binary mask, we set all pixel values with intensity greater than 0 as 1. This creates a black background and a white foreground. The foreground consists of photoreceptors of the drosophila eye.

9. Once we have the binary mask, we then work to remove the noise from the image to get a better-quality mask. We perform a Closing morphological operation on the mask. Closing is a type of dilation followed by erosion that is used to fill small holes and smooth rough boundaries in an image. We perform this operation using the close-holes() function provided in the Mahotas library

10. After removing the inner noise, we perform the close region morphological operation on the image. We use the Mahotas function close() to apply this operation.

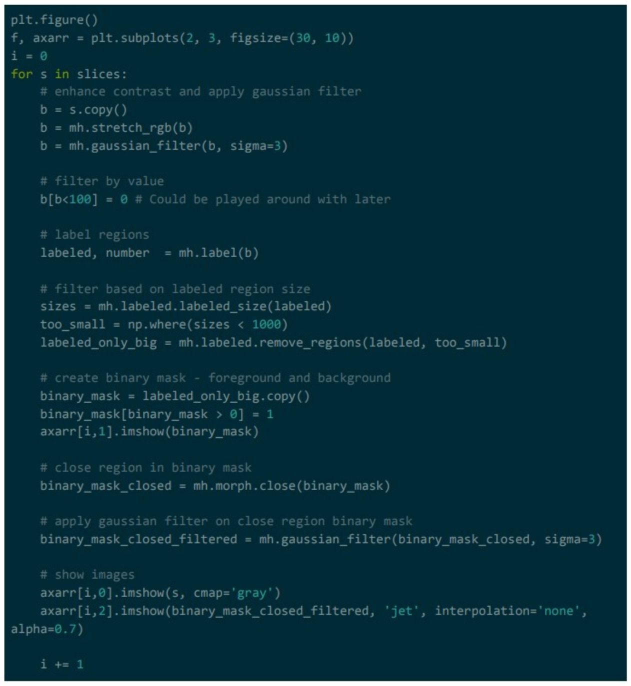
11. Finally, to remove small island noise, we use dilation and erosion techniques. Dilation expands the size of objects by adding pixels to the boundaries of the objects, while erosion shrinks the size of objects by removing pixels from the boundaries. For optimum results, we dilate the image four times and then erode the image the same number of times.

12. Lastly, after all the above operations, we again smoothen the image and reduce noise using the Gaussian Filter

We have performed numerous experiments with the segmentation pipeline. We used Jupyter notebooks for all of our work. Jupyter notebooks are interactive documents that allow the combination of code, text, and visualizations in a single document. They are used for data analysis, machine learning, and scientific computing.

3.2 Milestones

3.2.1 Milestone 1. After the successful data conversion, it is tested on the neuroglancer tool to explore more in-depth for better understanding. Moreover, the iframes of the neuroglancer workspace are stored on the mpsych.org/rister webpage. The iframes allow users



```
plt.figure()
f, axarr = plt.subplots(2, 3, figsize=(30, 10))
i = 0
for s in slices:
    # enhance contrast and apply gaussian filter
    b = s.copy()
    b = mh.stretch_rgb(b)
    b = mh.gaussian_filter(b, sigma=3)

    # filter by value
    b[b<100] = 0 # Could be played around with later

    # label regions
    labeled, number  = mh.label(b)

    # filter based on labeled region size
    sizes = mh.labeled.labeled_size(labeled)
    too_small = np.where(sizes < 1000)
    labeled_only_big = mh.labeled.remove_regions(labeled, too_small)

    # create binary mask - foreground and background
    binary_mask = labeled_only_big.copy()
    binary_mask[binary_mask > 0] = 1
    axarr[i,1].imshow(binary_mask)

    # close region in binary mask
    binary_mask_closed = mh.morph.close(binary_mask)

    # apply gaussian filter on close region binary mask
    binary_mask_closed_filtered = mh.gaussian_filter(binary_mask_closed, sigma=3)

    # show images
    axarr[i,0].imshow(s, cmap='gray')
    axarr[i,2].imshow(binary_mask_closed_filtered, 'jet', interpolation='none',
alpha=0.7)

    i += 1
```

Figure 4: The code of segmentation pipeline.

to easily access the neuroglancer workspace to view the preloaded data from the servers in neuroglancer. Here, users can use the neuroglancer tool's controls to navigate the volume and view it from different angles and perspectives.

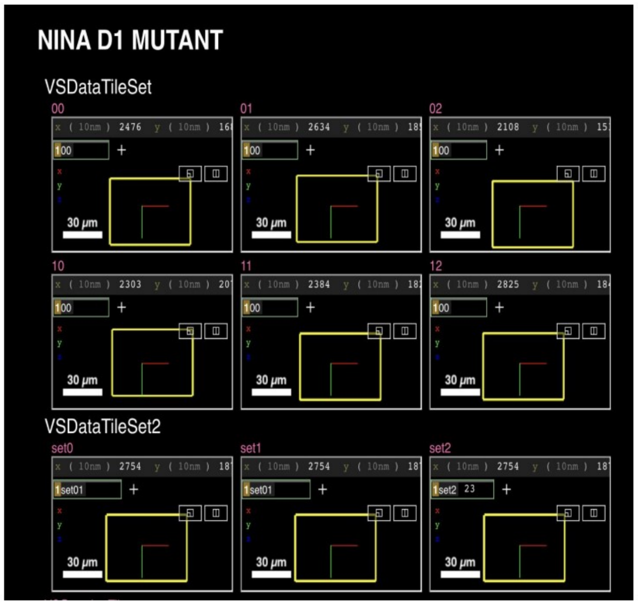


Figure 5: A cropped image of the iframes thumbnails from mpsych.org/rister.

3.2.2 *Milestone 2.* The results look good for some images and could be better for some samples. The varying results we get from the segmentation pipeline are because the images in the dataset have varying contrast levels. We can use contrast enhancement for this. Contrast enhancement is a technique used in image processing to improve an image's visual appearance by adjusting its pixels' intensity levels. The goal of contrast enhancement is to stretch the range of intensity values in an image so that the pixels are more evenly distributed across the full range of intensity values, resulting in an image that is more visually appealing and easier to interpret.

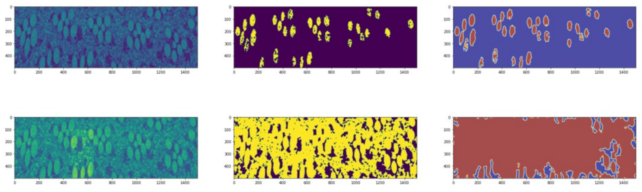


Figure 6: Outputs of the segmentation pipeline

3.3 Challenges

- Challenge 1: Overcoming the formidable challenge of transferring the massive amount of data from the fly server to our lab machines proved a formidable task, as the pipelines repeatedly broke during the transfer, requiring us to start the process from the beginning. The difficulty is compounded by the fact that the same issues arose when attempting to upload the converted data sets back onto the fly server, adding a layer of complexity to the already daunting task.
- Challenge 2: The segmentation pipeline presented a particularly challenging one for us as we struggled to distinguish between good- images that met our requirements and bad- images ones that would compromise the overall integrity of the 3D construction. Sifting through the vast quantity of data and making these critical determinations proved to be an exceptionally trying and time-consuming process, testing the limits of our patience and fortitude

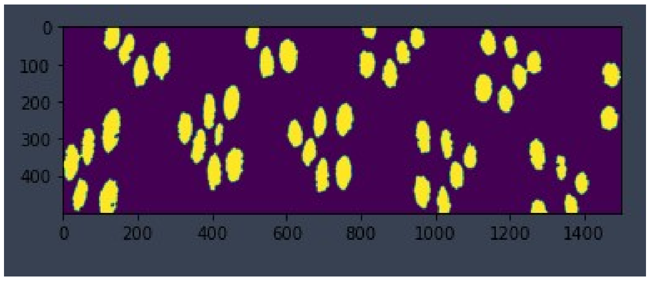


Figure 7: example of a good image

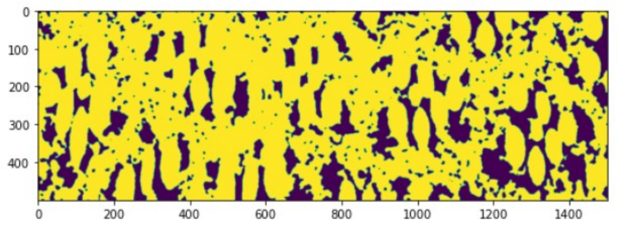


Figure 8: example of a bad image

4 RESULTS

There are multiple techniques of contrast enhancement. The technique we are trying to perform on our dataset is Histogram equalization. This method redistributes the intensity values of an image such that the resulting histogram is distributed evenly. Once we get even contrast images, we believe that our pipeline should work with much better accuracy

The results also show little island noise in some images, even after dilation and erosion. A way to handle this problem could be to use size filtering again on the generated binary mask. Apart from size filtering, we can also look into other filtering methods like Median and Adaptive filters. Median filtering is a method that replaces each pixel in the image with the median value of a set of neighboring pixels, which can help to smooth out isolated pixels. Adaptive filters are the filters that adjust the strength of the filtering based on the local characteristics of the image, which can help to preserve essential image details while still reducing noise.

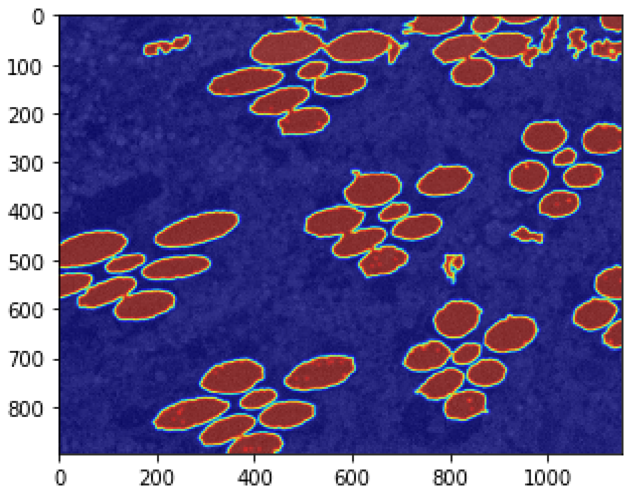


Figure 9: An example image with small island noise

Once we had good results from the segmentation pipeline and better segmentation accuracy, we reconstructed the segmented dataset 3D using 3D slicer software.

The 3D Slicer software is an open-source platform for medical image informatics, image processing, and three-dimensional visualization. It provides several tools and features that can be used to visualize and analyze medical image data, including the ability to load and display mesh data stored in the NRRD file format. To create a mesh from a segmentation in 3D Slicer, we use the "Make Model" effect in the "Segment Editor" module. This effect converts the selected segmentation into a 3D model, which can then be displayed and manipulated in the 3D Slicer viewer window.

To use 3D Slicer to visualize a mesh, we follow these steps:

1. Open 3D Slicer and select the "File" menu.
2. Choose "Add Data" and select the NRRD file that contains the data.
3. Load the image data into 3D Slicer and create a segmentation using the tools in the "Segment Editor" module.
4. Select the segmentation we want to convert into a mesh in the "Segments" list.
5. Click the "Make Model" button in the "Segment Editor" module.
6. In the "Make Model" dialog, we specify the desired settings for the model, such as the material, color, and level of detail.
7. Click "Apply" to create the model. The resulting mesh can be seen in the 3D Slicer viewer window

Overall, for the 3D construction work we have done so far, It is essential to carefully evaluate the performance of our segmentation approach, using metrics such as accuracy, precision, and recall, to ensure that it is producing high-quality results which we later use to generate the 3D Construction of fly photoreceptors. We also need to consider the scalability and efficiency of our approach, as well as any specific constraints or limitations imposed by the data.

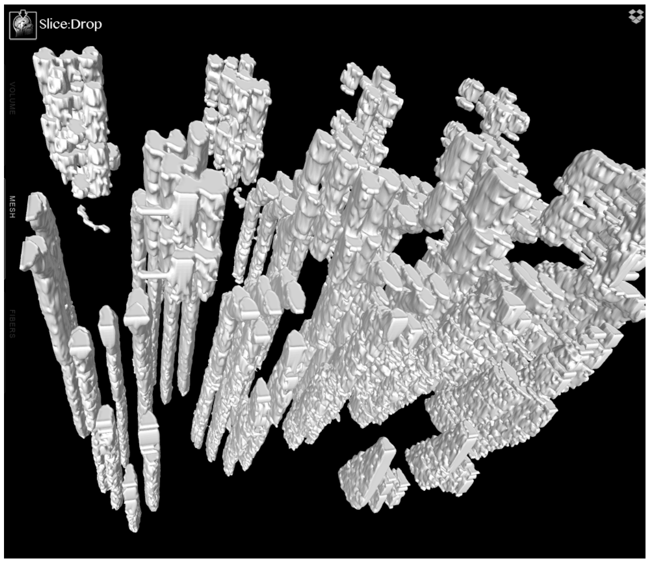


Figure 10: 3D view of mesh slices.

5 CONCLUSIONS

In conclusion, electron microscopy is a technique that is used to image photoreceptors in the eye at high resolution. In this study, the photoreceptors are observed to be organized into a regular hexagonal array and connected to other cells in the eye through dendrites. The raw data from the electron microscopy images are converted into a Neuroglancer-compatible format through a series of steps involving converting TIFF images to JPEG, reading relevant information from .info files, and aligning the data with the standard Neuroglancer JSON format. The resulting Neuroglancer stack allowed for the visualization and exploration of the volumetric data in three dimensions and the ability to interact with and annotate the data using Neuroglancer's tools and features.

The segmentation pipeline achieved good results with high accuracy, and the 3D reconstruction was performed using the Slicer 3D, specifically the segement editor tool that we used to create mesh from slices. It is important to continue evaluating the performance of the segmentation approach and ensure that it produces high-quality results for the 3D construction of fly photoreceptors. Scalability and efficiency should also be considered, as well as any specific constraints or limitations of the data.

REFERENCES

1. Mechanisms of vitamin A metabolism and deficiency in the mammalian and fly visual system - https://www.sciencedirect.com/science/article/pii/S0012160621000762
2. A Connectome Based Hexagonal Lattice Convolutional Network Model of the Drosophila Visual System - https://arxiv.org/pdf/1806.04793.pdf
3. Fy rhabdomeres - https://figshare.com/articles/figure/_Diagrams_of_the_Eye_and_Retinal_Organization_of_the_Fruit_Fly_Drosophila_/603114/1
4. Google Neuroglancer - https://github.com/google/neuroglancer
5. Neuroglancer volumes - https://github.com/google/neuroglancer/blob/master/src/neuroglancer/datasource/precomputed/volume.md
6. Neuroglancer scripts - https://github.com/HumanBrainProject/neuroglancer-scripts/blob/master/docs/script-usage.rst
7. Neuroglancer python integration - https://pypi.org/project/neuroglancer/
8. Mahotas Python - https://arxiv.org/abs/1211.4907
9. Numpy Python - https://numpy.org/doc/stable/
10. Image Processing Algorithms are the basis for Image Computer Analysis and Machine Vision - https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f312bf0664bec582128ef8d2fa1164af631a2d31
11. Geeks for geeks Computer Vision techniques - https://www.geeksforgeeks.org/computer-vision-introduction/
12. Mahotas documentation - https://buildmedia.readthedocs.org/media/pdf/mahotas/release-1.0/mahotas.pdf
13. Pynrrd documentation - https://pynrrd.readthedocs.io/en/stable/
14. Slicer 3D - https://ieeexplore.ieee.org/abstract/document/1398617
15. Image segmentation based on 3D slicer - https://www.extrica.com/article/21263