BEYKOZ UNIVERSITY

GRADUATE PROGRAMS INSTITUTE

SEGMENT ANYTHING MODEL FOR CRATER DETECTION ON THE LUNAR SURFACE

Issa DAHDOULI

MASTER'S THESIS

İstanbul, June 2025

SEGMENT ANYTHING MODEL FOR CRATER DETECTION ON THE LUNAR SURFACE

Issa DAHDOULI

MASTER'S THESIS

Institute of Graduate Programs

COMPUTER ENGINEERING MASTER'S PROGRAM WITH THESIS

MASTER'S THESIS ADVISOR

Prof. Dr. Abdurazzag Alı A ABURAS

MASTER'S THESIS JURY MEMBERS

Asst. Prof. Dr. İnal Begüm TURNA DEMİREL

PREFACE

ACKNOWLEDGMENT

I want to express my gratitude and deepest appreciation to those who have stood by me throughout this journey. Your unwavering support and encouragement have been my source of strength and motivation for every step of this thesis.

To my father, Samir Dadoulis, and my mother, Nadia Refaieh, I offer my heartfelt thanks for your unconditional love and endless support, which have laid the foundation for all my accomplishments. I am forever grateful for your sacrifices and belief in me.

To Prof. Dr. Abdulrazzaq Ali Aburas, my advisor, I would like to extend my sincere thank you for your guidance and feedback, and the trust in my vision, your expertise and mentorship have been instrumental in shaping the direction and success in this research.

This thesis, titled “SEGMENT ANYTHING MODEL FOR CRATER DETECTION ON THE LUNAR SURFACE “, is a reflection not only of my academic dedication but also the love, wisdom, and encouragement of those around me.

With all my gratitude,

Issa Dahdouli (Γιουσά Νταδούλης)

Contents

ABSTRACT

SEGMENT ANYTHING MODEL FOR CRATER DETECTION ON THE LUNAR SURFACE

In this research we will discover how to implement Segment Anything Model for crater detection, Crater Detection is an essential part of planetary geology and scientific research, for mission planning and future lunar landing missions, this thesis will present a semi-automated crater detection pipeline, which will combine Segment Anything Model, which is a new state of the Art Deep learning model developed by Meta AI, we will implement classical geometric analysis, leveraging' SAM's promotable vision transformer, to generate initial crater mask candidates from 1024x1024 lunar surface images, due to the lack of datasets of the lunar surface these images will be generated using Three.js a program that uses WebGL to render 3D objects in the web browser, which then will make use of NASA's MoonKit to create an actual accurate sphere of the lunar surface with proper topographies and surface details and diverse lighting conditions, once our image is processed by SAM we will run a custom ellipse fitting algorithm, which then applies geometric filters, like area ratio, elongation and brightness to isolate crater like structures, it will use Least Squares Method for such operation, later we export the results in .PNG format showing our segmented image and Ellipse fitted image, in this research we will also discuss the difficulties and challenges of Lunar Crater Detection, shedding light on the different conditions of lunar surface, from lighting to shading, and it's challenges on image segmentation models like SAM, we will discuss the limitations of SAM in such regards for crater detection.

Keywords: Segment Anything Model, SAM, Image Segmentation, Crater Detection, Computer Vision, OpenCV, Ellipse Fitter, Least Squares Method, Deep Learning, Three.js

Date: July 2025

ÖZET

AY YÜZEYİNDE KRATER TESPİTİ İÇİN SEGMENT ANYTHING MODELİ

Bu çalışmada, krater tespiti için Segment Anything Modeli'nin (SAM) nasıl uygulanabileceği araştırılacaktır. Krater tespiti, gezegen jeolojisi ve bilimsel araştırmaların önemli bir parçası olup, görev planlaması ve gelecekteki Ay iniş görevleri için kritik öneme sahiptir. Bu tez, yarı otomatik bir krater tespit hattı sunmaktadır. Bu sistem, Meta AI tarafından geliştirilen yeni nesil bir derin öğrenme modeli olan Segment Anything Modeli ile klasik geometrik analizleri birleştirmektedir.

SAM'in yönlendirilebilir görsel dönüştürücüsü (promptable vision transformer) kullanılarak 1024×1024 çözünürlükteki Ay yüzeyi görüntülerinden ilk krater aday maskeleri üretilecektir. Ay yüzeyine dair etiketli veri setlerinin eksikliği nedeniyle bu görüntüler, WebGL tabanlı 3D görselleştirme motoru olan Three.js kullanılarak üretilecek ve NASA'nın MoonKit kütüphanesi aracılığıyla Ay'ın gerçekçi topografyasına ve yüzey detaylarına sahip küresel modeller oluşturulacaktır. Ayrıca, farklı ışık ve gölgeleme koşulları da simüle edilecektir.

SAM ile işlenen bu görüntülerden elde edilen maskeler üzerinde, özel olarak geliştirilen bir elips uydurma algoritması çalıştırılacaktır. Bu algoritma, alan oranı, uzama (elongation) ve parlaklık gibi geometrik filtreler uygulayarak krater benzeri yapıları izole edecektir. Bu işlem sırasında En Küçük Kareler Yöntemi (Least Squares Method) kullanılacaktır.

Sonuçlar, segmentasyon çıktısını ve elipslerle işaretlenmiş görüntüyü gösteren .PNG formatında dışa aktarılacaktır. Ayrıca bu çalışmada, Ay krateri tespitinin zorlukları ve karşılaşılan problemler de detaylı şekilde tartışılacaktır. Ay yüzeyinin değişken ışıklandırma ve gölgeleme koşulları gibi faktörlerinin segmentasyon modelleri üzerindeki etkisi vurgulanacak, Segment Anything Modeli'nin bu bağlamdaki sınırlamaları da ele alınacaktır.

Anahtar Kelimeler: Segment Anything Modeli, SAM, Görüntü Segmentasyonu, Krater Tespiti, Bilgisayarla Görü, OpenCV, Elips Uydurma, En Küçük Kareler Yöntemi, Derin Öğrenme, Three.js

Tarih: Temmuz 2025

SYMBOLS

Least Squares Method Related:

A : Coefficient of x^2 controls horizontal curvature
B : Coefficient of xy controls rotation/tilt of the ellipse
C : Coefficient of y^2 controls vertical curvature
D : Coefficient of x controls horizontal shift
E : Coefficient of y controls vertical shift
F : Constant term acts like an offset to position the conic in space
x : Horizontal Position of Pixels in 2D image
y : Vertical Position of Pixels in 2D image

Intersection over Union Related:

IoU : Intersection over Union
M : Current Mask
U : Union of Masks
$\cap$: Intersection
$|$: Cardinality or area
τ : Threshold

ABBREVIATIONS

API : Application Programming Interface

ALC : Albategnius

BERT : Bidirectional Encoder Representation from Transformers

BIO : Biot

CDA : Crater Detection Algorithms

CLIP : Contrastive Language Image Pre Training

CNN : Convolutional Neural Networks

CSV : Comma Separated Values file

DEM : Digital Elevation Model

GPT : Generative Pre Trained Transformers

GPU : Graphics Processing Unit

GUI : Graphical User Interface

IoU : Intersection over Union

KM : Kilometers

LROC : Lunar Reconnaissance Orbital Camera

MAE : Masked Autoencoders

MLP : Multilayer Perceptron

NASA : National Aeronautics and Space Administration

NLP : Natural Language Processing

NMS : Non Maximum Suppression

SAM : Segment Anything Model

SVM : Support Vector Machines

SOS : Sosigenes

TIF/F : Tagged Image File Format

TRI : Triesnecker

TYC : Tycho

ViT : Vision Transformer

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1. INTRODUCTION

Ever since humans gazed up the stars, curiosity followed, which is indeed a human trait, our ancestors looked up in awe, and as we began to understand what is around us, and gained knowledge and technology advanced, we began trying to set reach beyond our home, our first step was the moon, we sent orbiters and photographed and analyzed the surface, and before we can set foot, we had to find a good place to land, it was a tedious process, and we as humans it is in our nature, that we aspire for efficiency and speed, so we began to automate it, as the fields of Neural Networks and Machine evolved, it's potential use in space exploration was evident, and only a natural step. (Yue et al., 2023)

In Planetary Science Crater Detection is crucial task for deep space exploration, involving the identification of impact craters on lunar and planetary surfaces, primarily from the images of the surfaces of Celestial Bodies, craters are the most abundant landmarks on planets, and their characteristics provide valuable insights of a planet's geological history, or places for study or avoidance for a future space mission, or they could be used as landmarks for autonomous spacecraft navigation, enabling precise orbit determination for close flybys or low altitude orbiting. (Yue et al., 2023)

Traditional crater detection methods, involving human labeling craters manually, was a time consuming and tedious process, there is also a debate among experts on what is a crater and what is not, the wide variations in crater size, degradation levels, internal morphologies and imaging parameters, overlapping craters, and other geological features also makes the process tedious and time consuming, and this have led scientists to innovate, since we humans strive for efficiency and speed, since computers were invented to make things faster, humans began finding ways to involve them in tedious and repetitive tasks. (Lee & Hogan, 2021)

Many Crater Detection Algorithms (CDAs) were developed and they were tested on limited datasets, which restricts their broader applicability, and that leads to their gaps and vulnerabilities to be discovered, there is a lack of standard and universally accepted tool for automatic crater detections, tools that achieve consistent and reliable results across different datasets and imaging conditions, these algorithms, worked using traditional methods of feature extraction and edge detection and texture analysis. (Chaini & Jha, 2024)

One of these Methods, is Segment Anything Model or SAM, this model was released by Meta in 2023, it was one of the most advanced image segmentation models available today, this model can identify objects and images without requiring additional training, and in the context of this research we shall test it on its ability to detect craters, without specifically training a model. (Kirillov et al., 2023)

One of the challenges in developing CDAs is the lack of datasets, datasets are scarce, and there is a lack of annotated datasets which will make supervised learning approach difficult, due to the scarcity of quality data for training or testing (Fairweather et al., 2022) one way to deal with this issue is the generation of synthetic datasets, using NASA's MoonKit for example.

Developing said algorithms is not an easy task, and one of the main challenges is having to deal with computational constraints and striving for accurate results, once a CDA is developed it has to be tested on a dataset, which we discussed the challenges we have to deal with earlier, but during testing we have to deal with the detection accuracy, we have to deal with false positives, false negatives and missed detections, and that needs adjusting for parameters to ensure accurate results. (Tewari et al., 2023)

1.1. Background

Craters come in different shapes and sizes, and identifying said craters, manually using human labor is time consuming and tedious and repetitive process, and some human scientists may not detect all craters, while other humans have other definition of a crater, particularly how craters are identified.(Tewari & Khanna, 2024)

As it is a long and tedious process, humans began looking for ways, to automate and enhance the accuracy of crater detection, using image segmentation to find craters, by segmenting the image and assigning a label or category to every pixel in the image, labels that share visual characteristics like color intensity or texture belong to the same object or region(Lee & Hogan, 2021).

What makes such process hard and tedious, is how different scientists define what a crater is, craters can be hard to identify to several factors, like terrain, fading, lighting and shadows and overlapping with other craters and size(Zhou et al., 2023)

1.1.1. Lunar Surface Crater Characteristics

The most prevalent geological features on the Moon are craters, which are formed by high velocity impacts of meteoroids asteroids or comets, these craters have specific characteristics, studying them gives a lot of information about the history of lunar geology, age and history of the moon or the celestial surface and for planning safe landing zones for future lunar missions.(Yue et al., 2023)

Since Lunar Craters resulted from the high velocity impact processes that produced a characteristic circular depression, the result is that they vary in size and age and diameter,

depending on the energy of the impact and the size of the meteor(Hiesinger et al., 2023) , most notable types of craters are:

1- ALC-Albategnius C: Small Cup shaped with a diameter of around 10km or less, they have no central floor
2- BIO-Biot: similar to AL but their typical diameter is about 15km, they are small with flat floors
3- SOS-Sosigenes: wide and flat interior floor, no central peak, not terraced with a diameter of in the range of 15-25km
4- TRI-Triesnecker: size in range from 15-50km large enough that their inner walls have slumped to the floor
5- TYC-Tycho: larger than 50km, terraced inner walls and flat floors, yet they have a central peak formations.

Beyond couple of hundred kilometers in diameter, TYC central peak disappear, they start to be classed as basins (Francis.B, 2022)

1.1.2. Definition and Importance of Crater Detection

Crater Detection, is the process of identifying, localizing and characterizing impact craters on the surface of celestial bodies moons and planets alike, such as Moon and Mars and asteroids.(Yue et al., 2023) Crater detection helps in understanding the geological history of the moon, it helps giving us an idea for Surface Age Estimation, it also helps us for Geological mapping, it helps us understand the history of surface processes of a celestial body, we can study features like degraded rims, infill patterns or overlapping craters, to gain information on tectonic activity lava flows or erosion(Lin et al., 2022) , Crater Detection can also be used to help in Landing Site Selection, it helps in finding

suitable locations on the lunar surface that are scientifically valuable for both crewed robotic space missions, it can also be helpful in avoiding heavily cratered regions or sloped terrain which pose risk to landing missions(Silvestrini et al., 2022) Crater Detection is also used in Navigation for autonomous spacecraft navigation, modern guidance systems can map and use craters as reference for localization and relative terrain navigation as well as hazard avoidance when descending(Roberto & Alfredo, 2021) Crater Navigation can also be used to train AI on Datasets for accurate crater detection, by labeling datasets for training supervised learning models(Chaini & Jha, 2024)

1.1.3. Challenges in Lunar Image Analysis

There are many challenges in Lunar Image Analysis, in our case we are going to deal with crater detection, and this alone has got a lot of challenges and difficulties, from lighting to shadows to faded and overlapping craters, to what is considered a crater in the first place (Tewari et al., 2023)

Varying Illumination Conditions makes detecting craters difficult, and since the moon doesn't have an atmosphere to scatter sunlight, this results in images with deep shadows and extreme brightness, which means that craters will appear different from different angles, which will complicate their detection and classification, thus a crater might appear visible in low angle lighting but might be invisible under direct overhead illumination.(Christian et al., 2020)

Another Challenge is overlapping and degraded craters, which resulted from billions of years of impacts, and the older craters have been eroded or filled with lunar soil, and that makes it difficult to distinguish individual crater boundaries, especially in automated

segmentation methods, this can confuse edge based or mask based models, which may reduce segmentation accuracy.(Lin et al., 2022)

And another is the lack of ground truth tables, and resolution and scale variability, from limited annotated datasets which makes it difficult for supervised learning approaches due to the scarcity of high quality training data, this problem is amplified with the Lunar Datasets which consist of datasets captured at various resolutions, which means that craters may appear with vastly different pixel scales across different datasets, which will affect the generalization of detection models. (Fairweather et al., 2022) There also still however an ambiguity in crater definitions, even among human experts, especially for small or eroded or secondary craters, which leads to subjective labeling and inconsistencies in training and validation of datasets (Tewari et al., 2022) another problem when it comes to using deep learning in crater detection are features like rilles pits, boulders or ejecta rays, which may mimic crater like patterns which in turn results in false positives confusing the models of both traditional and deep learning based detection pipelines (Francis.B, 2022) Not mentioning the issue of Data Volume and Computational Complexity, High Resolution Lunar Datasets are enormous in size and require a lot of computational power(Jia et al., 2022), especially when using patch based segmentation or large models like SAM (Segment Anything Model)

1.1.4. SUMMARY

So to summarize in order to do lunar image analysis we must first understand the nature of the lunar surface, scientists and experts themselves disagreed on what is a crater the challenges of lighting and shadows makes using image segmentation or deep learning techniques difficult, this is amplified by the issue of not having enough annotated datasets

let alone the huge diversity of datasets that are available, which come in different resolutions and conditions, which makes training a model challenging, these datasets are also huge and are computationally intensive to analyze, however studying craters and identifying craters gives an insight about the Lunar Surface, about it's history and geological history and processes.

1.2. Image Segmentation

Image Segmentation is a computer vision task, that expands upon the idea of Object Detection, it is a process in which an image is segmented into fragments and having a label for each fragment, and this occurs on pixel level, to define the precise outline of an object within its frame and class, this is known as an output, the highlighted pixels are highlighted in one or more colors, depending on what type of segmentation is used (Patil & Aggarwal, 2023)

1.2.1. Image Segmentation Types

There are different types of Image Segmentation, mainly Instance Segmentation, Semantic Segmentation and Panoptic Segmentation, the most foundational definition of image segmentation is Panoptic Segmentation, in which pixels are identified grouped and labeled in an object, during those tasks for an image, a segmentation mask is created, which is the label of all pixels in the image.(Elharrouss et al., 2021) Another form of Segmentation is Instance Segmentation, which works not too different from Panoptic Segmentation objects are still detected by their bonds in the image, however each new object is labeled as a different instance, even if it would be in the same category(Cheng et al., 2022), for example Car A and Car B are different, even tho they are cars ! or take into account people in a picture, even tho they are all humans, but for instance segmentation they are different !

Figure 1.2.1-1 Shows how Instance Segmentation works, each detected segment is an instance, each person or cup is an instance for example, credits Super Annotate (2023, August 17)

The other type is Semantic Segmentation, in Semantic Segmentation masks represent labeled images, it means that all pixels in the image belong to some form of a category, so pixels with the same category are represented as a single segment(Jain et al., 2023), so for example Car A and Car B will always belong to Cars category, or take into account people in a picture, Semantic Segmentation will label them as human.

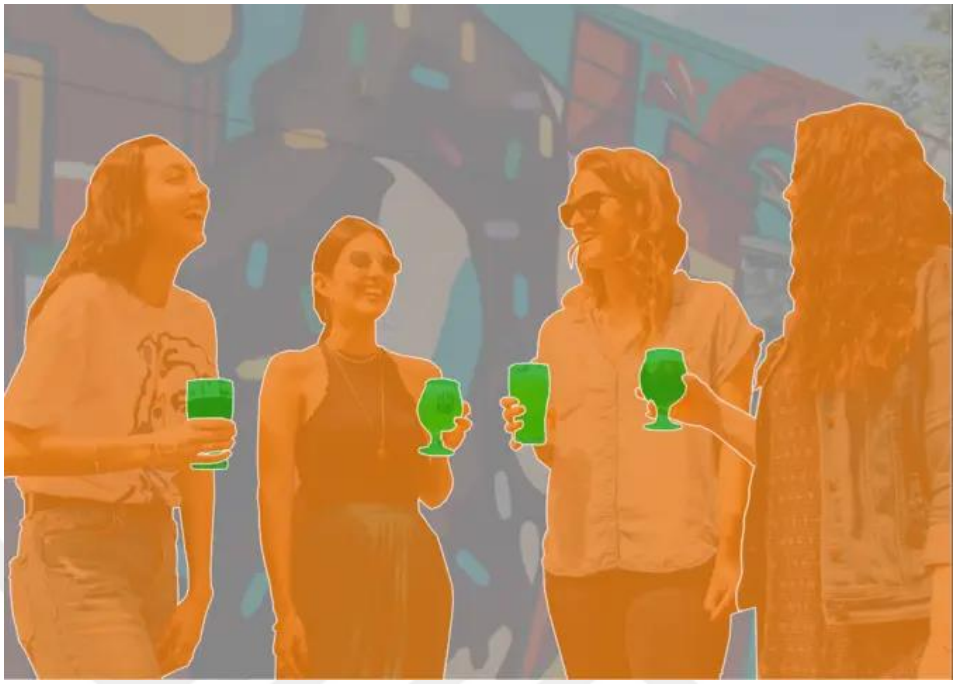


Figure 1.2.1-2 Shows how Semantic Segmentation Works, people are humans, cups are cups, credits Super Annotate (2023, August 17)

Of course both techniques are used for different purposes, for example Semantic Segmentation is used to distinguish between the background a category, to identify for example the exact shape of each instance in an image, while Instance segmentation takes into account the variety of objects in the segmentation process (Yu et al., 2023), this is a noteworthy difference, semantic segmentation may not be optimal for use in cases where we might need a more precise labeling, maybe we need to identify different types Animals or plants, this is vital in order to establish accurate labeling with minimal discrepancies.

Then we have Panoptic Segmentation, which combines both Instance and Semantic based segmentations, this way the entire image is labeled and different instances of pixels have different values even if they are in the same category, it is one of the complex computer vision tasks, it solves both problems of the semantic and instance segmentations

together, widely used in autonomous vehicles because cameras must provide full information that around.(de Carvalho et al., 2022)

Figure 1.2.1-3 Shows how Panoptic Segmentation works, labels the entire image, credits Super Annotate (2023, August 17)

1.2.2. Image Segmentation Techniques

There are many Image Segmentation Techniques, but for this research we will be focusing on introduction of the Traditional Region Based Segmentation and Edge detection segmentation, as well as the most important one for our research, the Deep Learning SAM Segment Anything Model.

Region Based, was the first technique, in which similarities are detected in the pixels of close proximity segments, so pixels that are nearest to each other are more likely to be a part of the same object, so this technique analyzes the similarities and differences between adjacent pixels and determines the boundaries of the given object, however its

flaws start to show when dealing with lighting and contrast within the image, which may lead inaccuracies when defining object parameters.(Kheradmandi & Mehranfar, 2022)

Edge Detection was developed in order to resolve the shortcomings of region based techniques, edge based segmentation algorithm emphasize the object edges, to achieve reliant results, which is done by determining and later classifying which parts, are "edge pixels" before anything else, this technique is best used with objects with clearly defined outlines, making them simple to implement for regular uses compared to other techniques which are more time consuming.(Sun et al., 2022)

Meta's Segment Anything Model SAM, this model was developed by META, it's ease of use makes nonmachine learning experts have access for using it for image segmentation, this model was trained on over 1 billion masks, thus making it able to give accurate predictions on any new dataset without additional training, SAM is also useful for complex semantic segmentation tasks, like space satellite images and other images like the ones in the medical sector (Zhang et al. 2023)

1.2.3. Limitations of Image Segmentation

Although there was remarkable progress in computer vision and deep learning, Image Segmentations still has got flaws, the problem with Image Segmentation in Lunar Image Analysis, mainly due to the lack of high quality annotated datasets, since Deep Learning is only as good as the data it learns from(Janiesch et al., 2021), Not mentioning the other issues of Illumination and Shadows, which leads to inconsistent detections especially in shaded or overexposed regions, it also has got difficulty detecting

overlapping features which may lead to over segmentation splitting a feature to multiple parts, or inaccurate results, where the true boundary of a crater are misrepresented, (Lin et al., 2022) Moreover, there is limited contextual awareness, which may lead to false readings of the surface features, being confused by ridges or valleys.

1.2.4. Summary

All in all using Image Segmentation to analyze Lunar Surfaces might be one novel way to detect Lunar surface terrain features, it does however need research on how good or viable such approach would be to analyze the lunar surface, and the results has to be studied thoroughly to validate it's use, for sure however deep learning methods would do better however compared to traditional methods.

1.3. Benefits of Research related to the industry

The research aims to help the Aerospace industry and the Space Exploration industry understand the geological history of the Moon and other celestial bodies, crater detection helps build a picture about the history of a celestial body, the soil composition or the volcanic activity, crater detection helps in optical based navigation of an autonomous space craft, using a crater as a landmark, crater detection also helps in either avoidance or in picking landing spots for future missions, it also helps in exploring the viability of Meta's deep learning SAM (Segment Anything Model) for crater detection and space exploration.

1.4. Project Problem Statement

The problem is the lack of standard crater detection tool to annotate datasets for future CNN models, crater detection is a crucial step for landing spot detections, and currently we have a problem in having to annotate datasets manually which is a very time consuming and labor intensive process, the other problem that comes with that is how accurate is Image Segmentation and is it a viable approach for dataset annotations and crater detections.

1.5. Project Goals and Objectives

The goal is to explore the viability of SAM in crater detection and how the modifiers can be optimized and improved for crater detection, it is meant to explore how good SAM is good in annotating datasets for developing and training a robust CNN later, how to increase SAMs True Positives and decrease False Positives and True Negatives, how to deal with the diverse shapes and sizes of craters.

1.6. Project Scope and Limitations

The limitations we have is the lack of capable hardware for mass annotations and a rather limited time scale for proper testing to make a robust model, another problem is how viable is the model for the surfaces of other planets, crater detection is one component for the detection of potential landing spots, and each component has to be robust enough before using them in a CNN program, that can accurately detect landing spots.

1.7. Project Proposed Solution

The proposed solution for mass annotation of datasets and for crater detection is to explore SAM for such task and make use of its modifiers and variables, for the dataset, Three.js software can be used alongside NASA's moon kit for lunar surface dataset generation.

1.8. Project Software Development Model

The Process starts with the requirements and analysis and the feasibility study, to identify the key functional needs for crater detection, we have an input image of the lunar surface, and we expect outputs of segmented or crater masks, these images will be obtained from NASA's MoonKit data, which will use Three.js synthetic rendering, images will be normalized in size and contrast will be enhanced to aid in segmentation accuracy, SAM will be integrated through python based environment, and it works by Apply prompt based segmentation to detect crater like structures and analyze the results, in the post processing phase, Ellipse filters will be run on segmented masks to isolate crater boundaries and filter non-crater features, maybe implement filters for overlapping shallow and large anomalies, and then testing and evaluation to compare our results to annotated test images using

metrics like precision, recall and intersection over union, and modify SAM parameters as needed, finally we document each phase's results for errors and improvements, interpret how well the model aligns with research objectives and scientific validity, thereby implementing an agile like approach.

1.9. Project Hardware and Software

1.9.1. Project Hardware

Our project's hardware consists of an Acer Nitro 5 Laptop with Intel Core I5 Gen 12 processor 12500H (16CPUs) at 3.1GHz and NVIDIA RTX3050m LAPTOP GPU with approximately of 12GB of memory

1.9.2. Project Software

As for Software we will be using a variety of programs from Anaconda Navigator to setup our Python Environment with the needed libraries for SAM, Three.js for Lunar Surface rendering and dataset extraction and preparation, ellipse filter to help in crater detection, Three.js will render the moon in our Web Browser and for this we are using Edge browser.

1.10. Summary

To sum it all up, we have a problem in our first step in developing an algorithm for proper crater detection which is a module in our big program to train a CNN to find potential landing spots, crater detection is challenging process, from finding datasets to annotating them, to dealing with the complex features of the lunar surface like lighting and shadows and eroded and overlapping craters, to the definition of that is a crater in the first place, however it is rewarding that it allows us to have information and idea about the history of the celestial body, and helps in many areas in the field of space exploration, from geology to autonomous navigation to hazard avoidance when picking a landing spot or if a crater has got enough value, it can be the place for a future landing spot in space exploration.

2. Literature Review

In this section we will discover various crater detection techniques from different sources, with the pros and cons of each one and the gaps of each one.

2.1. Review of Related Research Work

There is a lack of a standard crater detection algorithm or CDA due to challenges in crater detections, mainly due to the nature of craters, being size, degradation, morphology and overlapping features (Emami et al., 2019) they analyzed several CDAs and categorized the approaches into, Supervised, Unsupervised and Hybrid Techniques and they highlighted their role in improving lunar and planetary crater catalogs, unsupervised CDAs are CDAs that assume visual characteristics, by employing image processing and pattern recognition, like Hough Transform and are popular for detecting circular rims (Emami et al., 2019) among other methods, they noticed that Unsupervised approaches alone are insufficient for accurate detection and must require verification (Emami et al., 2019), which agrees with what discussed earlier in the introduction section of this thesis (Lin et al., 2022) (Tewari et al., 2022)

In Supervised Techniques, one notable example are decision trees, that use attributes for verifications of candidate regions, with Haar-Like features, that are also being used for crater region classification, Convolutional Neural Networks CNNs began gaining popularity due to their role and ability to learn features and other classification models during training (Emami et al., 2019), and they seem to be a better approach when dealing with crater detection, all in all Emami advises to use both combination of the two techniques for a hybrid approach, by using the information from both optical images and Digital Elevation Model DEM, which makes it for future research.(Emami et al., 2019)

to summarize it all from Emami, Unsupervised techniques alone are not enough and require verification using supervised techniques.

Now another study conducted by Zhou et tal, where they investigated the use of DEM data with the terrain analysis method for automatic detection of craters, they introduced a novel crater detection algorithm (CDA) which utilized digital elevation models or DEMs to enhance the accuracy of lunar impact crater identification, they argue that current mainstream CDAs of represent craters as circles which then leads to failure in reflecting their true shape and limits the measurement of characteristics such as circular shape and diameter (Zhou et al., 2018) this argument also agrees with the general idea of what we had discussed in the introduction (Tewari et al., 2023) their CDA incorporate a neighborhood mean algorithm, and reclassification method to obtain positive terrain, which leads to filtering out non-crater rim noises, then they filter out non-crater rim noises, and used morphological treatment to achieve complete true impact crater boundaries, and later conducted further denoising (Zhou et al., 2018) their study emphasized the importance of lunar crater research, stating, crater density determines the geological age of lunar surfaces, they argue that manual crater extraction is time consuming and be inaccurate due to image resolution or illumination angles, which agrees with what we have discussed in the introduction earlier (Lee & Hogan, 2021). (Zhou et al., 2023)

Existing algorithms often approximate craters as circles or ellipses, which can imprecise, algorithms using slope information, might identify various topographic features as craters by mistake, the study aims to solve these problems, by proposing a terrain analysis approach using 100m resolution DEM data from the Lunar

Reconnaissance Orbital Camera Dataset, their method uses a new indicator, SP(x,y), which combines SOA and positive terrain information to determine lunar crater rims, thereby reducing false rim detections and preserving true rim detections effectively (Zhou et al., 2018) the effectiveness of their approach was validated on the D'Alembert and Serenitatis regions, thereby achieving high detection and quality percentages, compared to the LU60645 Crater catalog and other algorithms, they maintained a relatively low false detection rate, they compared their proposed CDA with existing methods, to demonstrate improvements in detection rate and overall performance, they also showed high coherence with true crater forms, further confirming their algorithm's accuracy, however, their CDA have certain limitations and it's susceptibility to errors in areas where craters are faded or degraded, or craters with ambiguous rim also pose a challenge for correct identification, their algorithm's performance was affected by the reclassification parameters, which lead to missed detections of small diameter craters and shallow craters (Zhou et al., 2018) the same problems were common among other CDAs as we discussed in the introduction (Lin et al., 2022) (Tewari et al., 2022)

Another research, used something called CraterIDNet, which is a fully End to End Convolutional Neural Network developed for crater detection and identification, in remotely sensed planetary images , it deals with the critical task of detecting and identifying craters on planetary surfaces (Wang et al., 2018), which is essential for planetary studies and autonomous navigation as we have declared in the introduction section of this thesis (Lin et al., 2022) (Silvestrini et al., 2022) it aims to deal with the issues of traditional crater detection algorithms, which are categorized into supervised or unsupervised methods, unsupervised methods of course rely on the pattern recognition and template matching, it utilizes edge detection filtering and Hough transforms or template matching approaches for both lunar and Martian images (Wang et al., 2018) (Sun

et al., 2022) Supervised methods employ machine learning and have demonstrated a much higher accuracy with techniques like genetic programming and support vector machines (SVMs) (Wang et al., 2018) thus making CNNs appear the promising technique in computer vision or CV, with researchers utilizing them as binary classifiers to determine which regions of the image are craters or not, however many methods that only use CNN's s classifiers limit their ability to simultaneously detect and identify craters.

Crater Identification involves matching detected craters with surface landmarks in a known database, this has received less attention than crater detection, existing approaches include comparing crater pairs and triples with 3D models, or using non dimensional parameters to compare detected craters in a database, there is a still a lack however for simultaneous crater detection and identification for remotely sensed planetary images, using a small network architecture. (Wang et al., 2018)

Then we have an approach by Giannakis et al which introduces a novel and flexible crater detection scheme which makes use of the Segment Anything Model, like the one we are using, for planetary exploration, the limitation of Traditional Methods are known, they are manual and labor intensive and time consuming (Giannakis et al., 2024) and they are very inconsistent, needless to say existing ML models are often trained on specific data types, limiting their reliability when applied to different data sources, angles or setups, so the Authors proposed a promotable segmentation system capable of zero-shot generalization to unfamiliar objects and images, without requiring additional training, this flexibility makes the detection of crater like objects across diverse data types, from Satellite images to DEMs and other celestial bodies like the Moon and Mars and various angles, (Giannakis et al., 2024) detected masks can analyzed to filter out non-circular and

elliptical shapes using geometric indexes these circles or ellipses can be fitted to the remaining masks to determine the location and geometry of the detected craters, they demonstrate the effectiveness of this approach through several case studies, using lunar and Martian data, they noted that due to the size of SAM it makes it limited in its capabilities to detect small craters (Giannakis et al., 2024) this can be partially solved by zooming in to specific areas, their suggested future work involves fine tuning SAM with diverse, labeled datasets from various celestial bodies and data types to enhance the performance and deal with the limitations. (Giannakis et al., 2024)

It is worth mentioning our new approach deals with problem of detecting small craters.

2.2. Research Gap

The main gap in the said researchers, is that they didn't try to utilize new deep learning image segmentation methods like Segment Anything Model to detect craters or annotate datasets, or try to discover it's modifiers to improve detection results this makes for a rather interesting approach to look into when trying to utilize SAM to annotate databases and later to train CNN on, it's an approach worth looking into as SAM is easy to set up and is relatively low effort, and it can be improved upon, making it an interesting approach for our research.

3. Methodology

In this section we shall describe the tools and methods conducted for our research, namely we will be describing, Dataset Generation and Preparation through Three.js and NASA's moon kit, we shall describe Meta's Segment Anything Model in detail and we shall go for an overview of the Software used, and the challenges related to our approach.

3.1. Dataset Generation and Preparation

First things first, we shall be describing our method for generating dataset, before we start annotating and detecting craters, we need a set of images to conduct our analysis on, one way to do it is through the use of NASA's MoonKit and, NASA's MoonKit is a Lunar Data Tool Kit for scientific research, while Three.js is a powerful Javascript library to create 3D graphics and representations.

3.1.1. Three.js

Three.js is an open source JavaScript Library, it provides a highly flexible environment for creating and rendering interactive 3D objects in a Web Browser, in our case, we are going to create a sphere object to generate synthetic images of the lunar surface, to later use our SAM Image Segmentation CDA, three.js will allow us to control the lighting and height from different angles and resolutions and controlling other parameters like surface morphology, thus giving us the ability to generate quite a rich dataset, which help us mitigate the lack of enough real world lunar imagery, giving us the ability to test our SAM segmentation CDA in depth, our program for generating synthetic images works either by manually talking an image of the lunar surface, or automatically by pressing a button, that rotates the moon object, randomize lighting settings and randomize the height between 3000 KMs and 4000 KMs in order to have a comprehensive overview of the lunar surface and conditions in our generated images, each generated image is 1024x1024, ensuring that various details are captured within the image, the images are later fed as input for our image segmentation program.

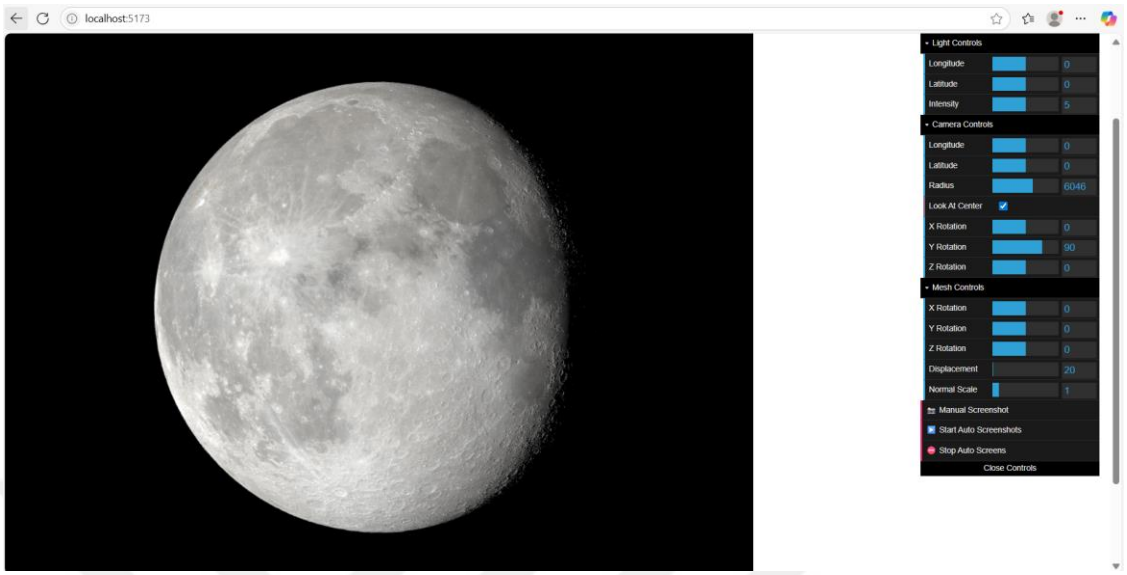


Figure 3.1.1-1 Shows how Three.js runs in a browser to render the Moon, textures were provided by NASA'S MoonKit

We will use this program to generate Synthetic images of the Moon's Surface to later run our analysis and segmentation models on. We automate this process by making our camera rotate the moon and take pictures of the moon from different heights and lighting angles, generating 1024x1024 images.

Below are examples of the said images:

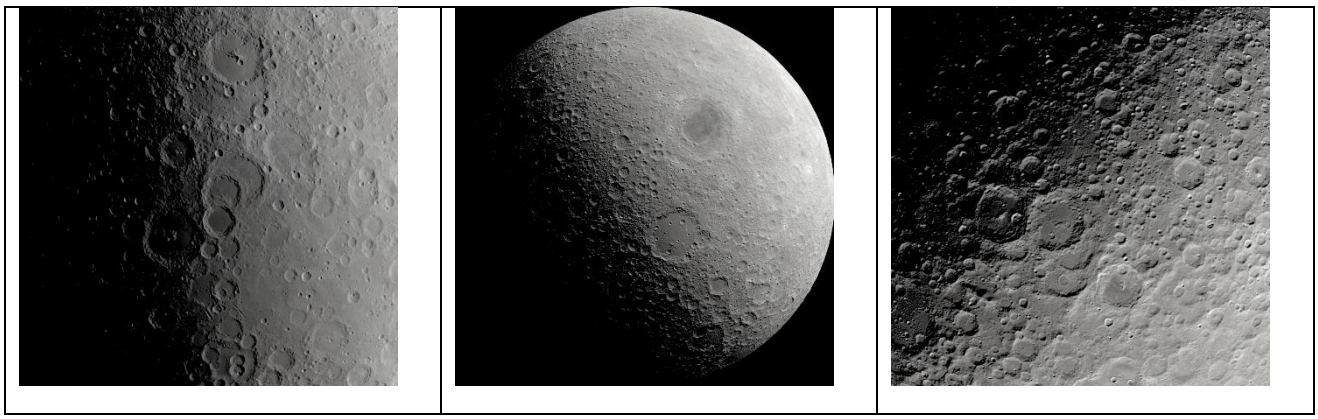


Figure 3.1.1-2 Showing Different Images Generated by Three.js

3.1.2. NASA's MoonKit

NASA's MoonKit, is a scientific tool for lunar surface research, it provides high resolution DEM and photometrically corrected images, we integrated this tool in our research to ensure that the synthetic images we are generating are realistic and provide robust hybrid dataset, rich in diversity and authenticity, because when combined with Three.js we can accurately represent the topographical and geological realities of the lunar surface, thus enhancing the authentic and the practical applicability of the results obtained, these images that are generated through Three.js are similar to ones by the Lunar Reconnaissance Orbiter Camera (LROC) thus making sure that our images look and feel authentic, the images are in .tif format or Tagged Image File format, and come into forms, the LROC Color and the DEM, with 27360x13680 and 23040x11520 pixels respectively, in order to use these images for our project, we must first run them through some processing, first we will have to convert them to .PNG format and rescale them and then we must generate the Normal map from the Displacement, Normal Maps are necessary for lighting and shadows, displacement map in this case comes from DEM, below are example images of NASA's MoonKit.

First we have the Moon Texture Map, this map represents the Lunar Surface, and it shows textures and craters and terrain details just like how you see them from orbit, now we have to run a Python Program on it to convert it to .png texture since it first comes in .tiff format, Second we have the Digital Elevation Model, which comes in .tiff format, again we will run a Python program on it to convert it to .png, this map provides height information of the lunar surface, brighter areas, represent higher elevation points, while darker areas represent lower elevations, we can't use it for our Three.js program directly since we have to process it and extract from it two vital maps, the Displacement Map and the Normal Map, so Displacement Map is a greyscale map, used to represent the detailed variations in lunar surface elevation, it is primarily in 3D modeling Software to make physical displacement to geometry, to create realistic terrain features, like craters and rims and depressions, Normal Map is used to represent the surface orientation, with each pixel having encoded data on the orientation of the lunar surface normal, which is necessary to

represent realistic lighting and shadows in synthetic images or as additional input data to improve crater detection algorithms.

Figure 3.1.2-1Shows the MoonTexture.PNG, which gives color for the model

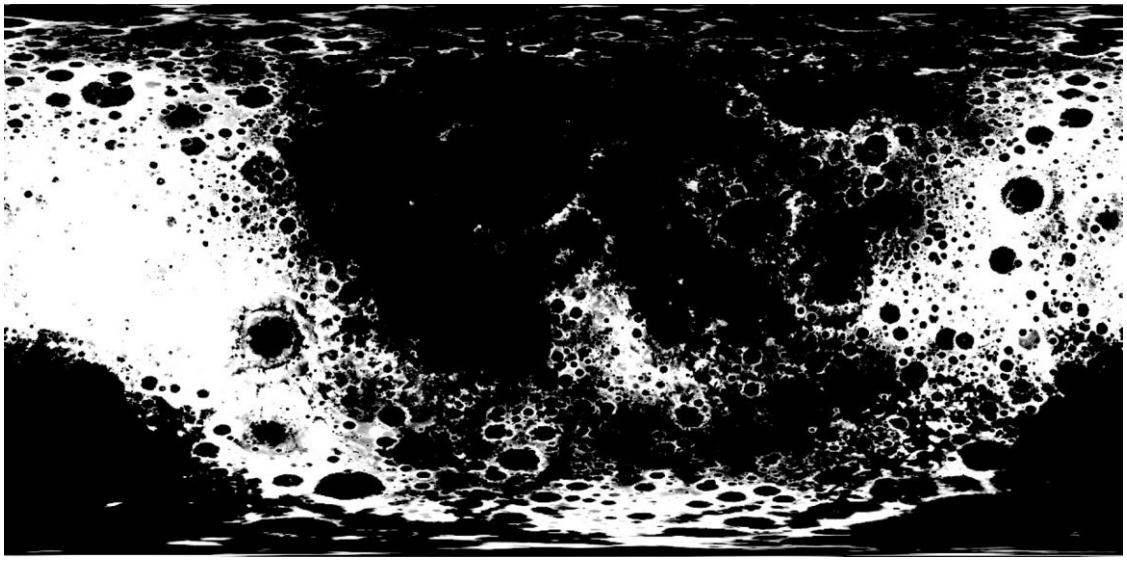


Figure 3.1.2-2 Shows the Digital Elevation Model Map that gives height and displacement for the model

Figure 3.1.2-3 Shows the Normal Map, generated from the Digital Elevation Model

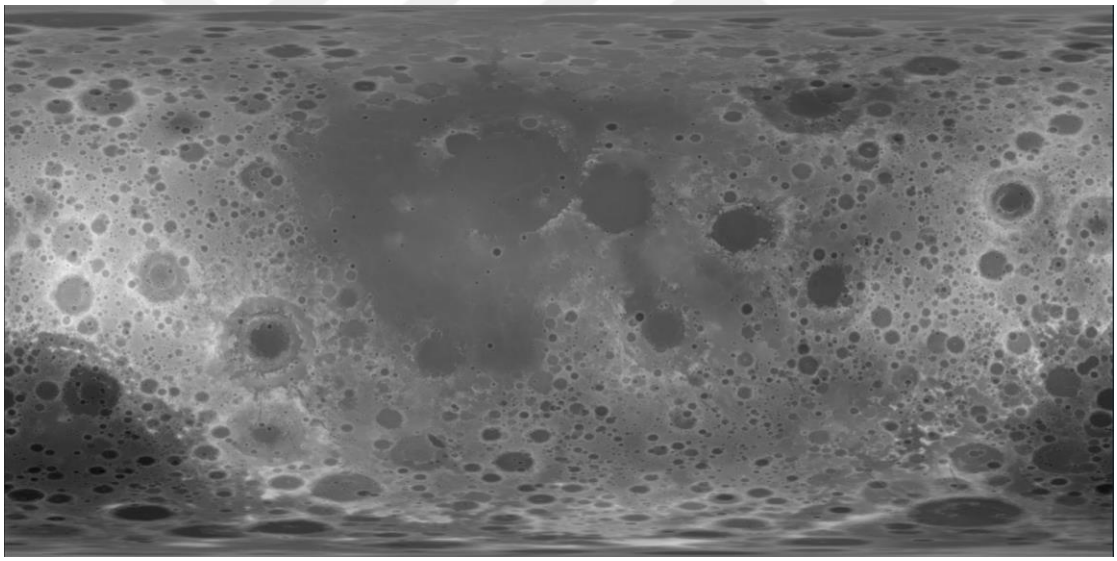


Figure 3.1.2-4 Shows the Displacement Map Generated from Digital Elevation Model

Normal Map Encodes the surface orientation, normal like RGB values, which simulates lighting and 3D surface details from a 2D image, thus representing the surface roughness and Lunar Topography based on the elevation or displacement at each pixel, Displacement map itself is a height function it's a 2D Array Z(x,y) where each pixel is given an elevation (heigh) at specific coordinates (x,y), pixel elevation values in our case will be converted into physical scale using the Moon's radius (1737.1 km)

assuming 64 pixels per degree, the slope is then calculated and Y is inverted to ensure the normal points outward in the image coordinate system, later tangent vectors and cross products are constructed to the surface, the normal vector will then be calculated through the cross product of these tangents, then each normal is converted to unit length so that all normal have magnitude of 1 normalizing them, and later the process to convert the normal vectors to RGBs begins and RGB normal map is exported as .png image, Normal maps are essential for lighting invariant crater detection, shape analysis and 3D visualization CNN training with geometric features.

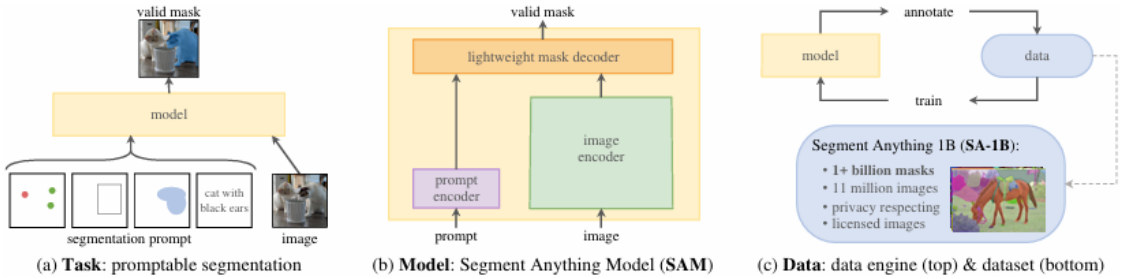
3.1.3. Challenges

The main challenges we encounter when using these two methods, are the computational constraints these, and that fact that Three.js using WebGL for rendering means we can't use our own hardware's true potential as we are limited by the software of the Web Browser.

3.2. Segment Anything Model Introduction

Segment Anything Model, is a powerful model made by Meta, it offers a novel approach to image segmentation, it offers a foundation for computer vision segmentation, akin to how large language models have revolutionized Natural Language Processing (NLP) (Kirillov et al., 2023), Segment Anything project, is designed to generate a valid segmentation mask, with various given prompts, like points, boxes or text, thus allowing the model Segment Anything Model, to perform zero-shot generalization to new image distributions and tasks through prompt Engineering, SAM's architecture consists of powerful image encoder, prompt encoder, and a lightweight mask decoder, allowing for real-time mask prediction and efficient reuse of the image's embeddings (Kirillov et al., 2023) to support this model and task, the developers of the project made a "data engine" to create massive dataset, known as SA-1B with over 1 billion masks on 11 million images, this dataset is collected through an iterative process of model-assisted

annotations, from manually annotated images, to fully automatic stages, significantly exceeding the scale of existing segmentation datasets (Kirillov et al., 2023) SAM's capabilities were tested through zero-shot transfer experiments on various downstream tasks, from edge detection to object proposal generation and instance segmentation it's a deep learning model not a tradition model, and it is competitive or fully surpasses fully supervised methods, even on tasks and data distribution not encountered during training, SAM and the SA-1B dataset are released under permissive licenses to foster further research in foundation models for Computer vision, thus making image segmentation more adaptable and composable component within larger AI systems.



(a) **Task**: promptable segmentation (b) **Model**: Segment Anything Model (**SAM**) (c) **Data**: data engine (top) & dataset (bottom)

Figure 1: We aim to build a foundation model for segmentation by introducing three interconnected components: a promptable segmentation *task*, a segmentation *model* (SAM) that powers data annotation and enables zero-shot transfer to a range of tasks via prompt engineering, and a *data* engine for collecting SA-1B, our dataset of over 1 billion masks.

Figure 3.1.3-1 shows the Architecture of SAM Segment Anything Model (Kirillov et al., 2023)

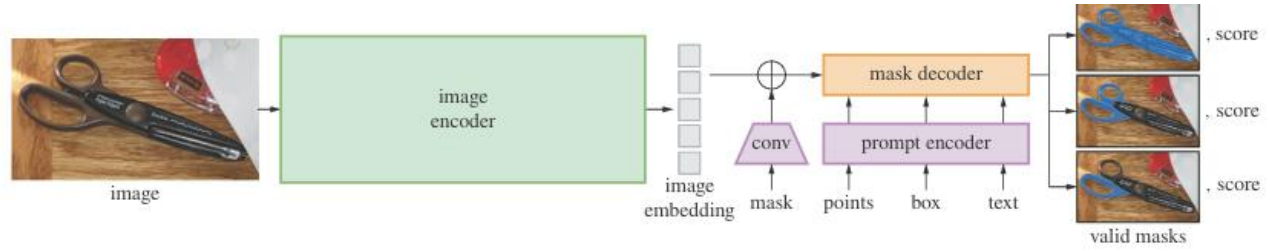


Figure 4: Segment Anything Model (SAM) overview. A heavyweight image encoder outputs an image embedding that can then be efficiently queried by a variety of input prompts to produce object masks at amortized real-time speed. For ambiguous prompts corresponding to more than one object, SAM can output multiple valid masks and associated confidence scores.

Figure 3.1.3-2 Shows how an input image is processed in SAM heavy weight image encoder (Kirillov et al., 2023)

SAM works through three main components and Image Encoder, Prompt Encoder, Mask Decoder so SAM is not a standard segmentation model, the main goal from developing the SAM project, was to develop a foundation model, for image segmentation, it was model developed on broad data that can be adapted to a wide range of downstream tasks, the current example of such models are BERT, CLIP, GPT-4 and BARD, these foundation models, are usually trained on a large amount of data, like billions of data points. (Kirillov et al., 2023)

Meta however for the nature of image segmentation data, made a data engine, data engine, is a model in the loop annotation strategy, and it collected training data in three stages, the first stage was called Assisted-Manual, and in this stage, Meta hired a team of professional annotators to label images with segmentation masks, these professional annotators labeled any object they could find in the order of prominence, and they were also assisted with a segmentation model, that was previously trained on standard segmentation datasets, the data engine at that stage collected over 4 million masks from 120 thousand images, the second stage was semi-automatic, and the goal was to increase the diversity of the dataset, which also involved human annotators, but this they human annotators were presented with already annotated images, and were asked to label anything else they could find, after that stage, the average number of masks per image, went from 44 to 72 masks, and data engine collected additional 5,9 million masks, and lastly the final stage was called fully-automatic, and it didn't involve human annotators anymore, here they prompted SAM with a 32x32 grid of points and, for each point, predicted a set of masks that may correspond to valid objects, they applied this process to 11 million high-resolution images that Meta previously collected from their provider and

got 1.1 billion high-quality masks and this became the final dataset called SA-1B(Kirillov et al., 2023), and Meta has made it publicly available under permissive license, it has 6x more images and 400x more masks than any existing segmentation dataset. (Kirillov et al., 2023)

Due to the uniqueness of SAM's architecture it is similar to ChatGPT's architecture, thus allowing for prompt engineering, and it also does zero-shot learning, SAM has got three main components, the first one is an image encoder, that takes an image and computes it's embedding, and this is the most computationally intensive step, now this happens only once, then the model can be prompted for segmentation masks, this is where the second component comes into play, which is the prompt encoder, a prompt can be taken, which can be a set of points (x,y fg/bg) , a bounding box (x1,y1) (x2,y2), another mask, or simply a descriptive text, like "cats" and outputs a prompt embedding, and then this embedding is combined with the image embedding and fed into the lightweight decoder that predicts our segmentation masks, so the architecture is promotable, and language models like ChatGPT show that prompting is a promising technique for zero-shot and few shot learning, but if want to use SAM as a general tool for segmentation, it has to be able to resolve ambiguity in prompts, to address this ambiguity SAM predicts multiple valid masks with a confidence score for each mask, using this method and some other modeling choices, we can now solve other downstream tasks by simply engineering appropriate prompts, the authors, evaluated its zero-shot capabilities on several tasks, one of them is edge detection, SAM however performs worse compared to the state of the art models trained on edge-detection, but it does compare well with other task-specific models, and it performs better than other zero-shot techniques.(Kirillov et al., 2023)

3.3. Overview of Software Used

For our Python environment we are going to import a lot of Libraries, most important ones are PyTorch, NumPy, OpenCV, Supervision, and the software that we are going to test our data preparation code is MS Edge Browser, through Three.js, which will be in a JavaScript file, the environment will be set up through Anaconda Navigator.

3.3.1. Anaconda Navigator

For our Python environment we are going to import a lot of Libraries, most important ones are PyTorch, NumPy, OpenCV, Supervision, and the software that we are going to test our data preparation code is MS Edge Browser, through Three.js, which will be in a JavaScript file, the environment will be set up through Anaconda Navigator.

3.3.2. PyTorch

PyTorch, an open source, deep learning framework, made and developed by MetaAI, it provides high-level abstractions used for making, training and evaluating deep learning methods, it also offers flexible for lower level tensor computations, PyTorch's dynamic computation graph allows researchers to modify models at runtime, thus making it suitable for experimental research tasks, such as segmentation model development for our research, we used PyTorch as a foundational framework to load and execute Segment Anything Model (SAM), SAM's pre-trained deep neural network models were built and distributed through PyTorch, which allows us for seamless integration of it's image encoder, prompt encoder and mask decoder components, the main key functionalities include, GPU acceleration for deep neural network operations, support for large tensor computations which are necessary to process high resolution lunar images, flexible integration with external libraries and custom processing pipelines.

3.3.3. NumPy

NumPy is a fundamental package necessary for scientific computing, with Python, providing the support necessary for large, multidimensional arrays and matrices, it also

offers a comprehensive collection of mathematical functions, necessary for operations on these arrays, for our research, NumPy was extensively used for preprocessing and manipulating image data, prior to feeding them to SAM for post processing of the resulting masks, key functions relevant for our research, is image normalization, efficient handling for large 2D and 3D datasets and vectorizing operations for rapid and fast numerical operations and computations, to sum it up, NumPy serves as a low level computational engine supporting both machine learning and computer vision operations, interfacing seamlessly with PyTorch and OpenCV.

3.3.4. OpenCV

OpenCV is a widely-used open source toolkit used widely for real-time computer vision, machine learning, and image processing. For our research OpenCV was employed for Image preprocessing tasks, contrast adjustments, histogram adjustments, noise reduction resizing and cropping, the conversion between different image formats and for other post processing tasks like applying morphological operations to refine the segmentation masks and for detecting contours for further shape analysis prior to ellipse fitting, OpenCV's suite of image manipulation algorithms, makes an essential tool for preparing lunar surface images for reliable segmentation and crater detection.

3.3.5. Super Vision

Supervision Library, is a library that holds an extensive suite of tools to manage visualize, post-processing, segmentation masks and annotations, it was originally made for general purpose Computer Vision tasks, Supervision library provides functionalities which are very useful for evaluating SAM outputs, including bounding box generation, mask filtering, overlap resolution and annotation export, this library simplifies integration into annotation pipelines, which should enable efficient quality control and facilitate more deeper analysis of the segmentation accuracy.

3.3.6. Ellipse Filter

Since craters, take the shape of an ellipse, and since our research revolves around crater detection, we are going to implement an ellipse filter to refine our crater detections which are produced by SAM, fitting an ellipse shape to segmentation masks, should serve as a

way to minimize false positives, but eliminating irregular or non-crater detections, to reduce the false positives and standardize crater boundary representations, ellipse filter integrates seamlessly with SAM outputs by identifying candidate regions, and applying geometric constraints, which should correspond the expected crater shapes, and then this process should be effective for handling overlapping craters or partially eroded crater rims, which are frequently encountered in lunar surface imagery.

4. Project Design and Architecture

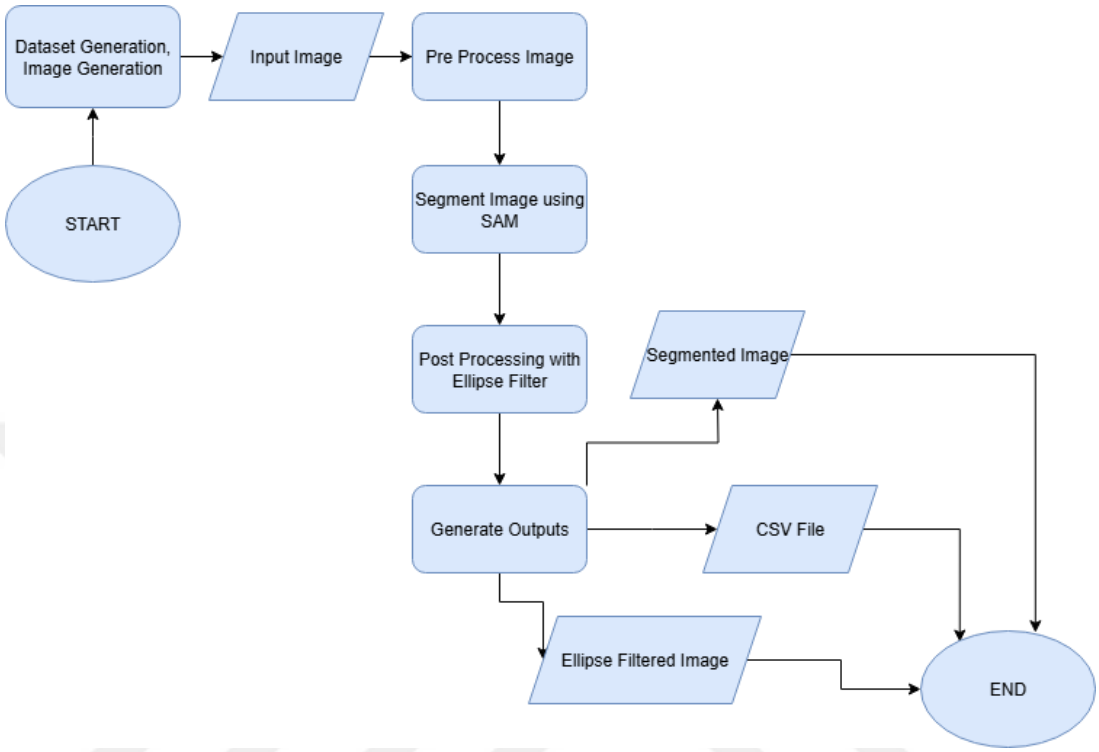
4.1. Project System Requirements Analysis

Since craters, take the shape of an ellipse, and since our research revolves around crater detection, we are going to implement an ellipse filter to refine our crater detections which are produced by SAM, fitting an ellipse shape to segmentation masks, should serve as a way to minimize false positives, but eliminating irregular or non-crater detections, to reduce the false positives and standardize crater boundary representations, ellipse filter integrates seamlessly with SAM outputs by identifying candidate regions, and applying geometric constraints, which should correspond the expected crater shapes, and then this process should be effective for handling overlapping craters or partially eroded crater rims, which are frequently encountered in lunar surface imagery.

4.1.1. System Architecture Overview

Our project is structured into 5 different modules, namely Dataset Generation Module, Preprocessing Module, Segmentation Module (SAM), Post Processing Module, Evaluation and Analysis Module.

Figure 4.1.1-1 The Flow Chart of the Crater Detection Program



4.1.2. Detailed Module Descriptions

For Data Generation Module, we are using Three.js, NASA's MoonKit, and our Primary Function of this module, Generation of Synthetic lunar images, simulating the diversity of the surface and lighting conditions of the Lunar Body, extraction and preparation of realistic lunar images from NASA MoonKit, with the output standardization is at 1024 x 1024 pixels for uniformity and consistency across datasets, for the Preprocessing Module, we use OpenCV and NumPY for the normalization of images, Contrast enhancement and noise reduction to improve the segmentation quality and the conversion of NASA's TIFF images to compatible .PNG format, then we have the Segmentation Module, with involved Segment Anything Model through PyTorch, with the primary function of deploying pre-trained SAM for zero-shot segmentation, application of segmentation using various prompt methods (points, bounding boxes or automatic) and the generation of initial segmentation masks to identify crater candidates, later we got the Post Processing Module, using Ellipse Filter, Supervision Library and OpenCV, with the primary functions of refining the SAM segmentation masks, using ellipse fitting algorithms, and

exploiting the typically elliptical nature of lunar craters, this filtering is necessary to remove false positives, irregular contours and overlapping detections, this should help in mask smoothing and improve accuracy, finally we have Evaluation and Analysis Module, with the primary function of Comparative Analysis of SAM-Generated masks against ground truth annotated images, this is done through the standard evaluation metrics, of using Precision, Recall, Intersection over union (IoU) and F1-Score, documentation of said SAM's performance in varied scenarios from illumination variations, to overlapping craters and other degraded features.

4.2. Callum Bruce's Crater Detection Algorithm Review

In this section we will talk about Callum Bruce's research and the use of Segment Anything Model to develop a CDA which is targeted at enabling autonomous crater-based optical navigation for spacecraft orbiting the Moon, his work, provides the basis for our thesis research project, his work employs a combination of Computer Vision and Neural Networks, by integrating SAM for crater rim segmentation, his pipeline uses Three.js to do image acquisition, then the image is processed using SAM and later an Ellipse Filter is used for extracting crater shapes and centroids, which will closely mirror our own system flow, his work demonstrates that SAM is indeed capable for accurate detection of crater rims, with ellipse fitting producing accurate and precise geometric characterization for navigation applications, the drawbacks however include sensitivity to prompt placement, and mask threshold settings, that will require careful parameter tuning, which will be similar to our challenges in our experiments, his work confirms the viability of SAM for Zero-shot crater segmentation in lunar imagery, it highlights the importance of prompt engineering for further mask refinement which we will investigate further, and confirms the importance of ellipse fitting to confirm its central role in post processing pipeline, his work however needs rigorous testing and evaluation across different lunar surface conditions, which we will investigate further in our research thesis, we will explore different parameters to improve detections and their impact on performance.

His program works by having an input image which is generated using three.js program that is then processed by SAM and later uses Ellipse filter to detect circular masks, it later outputs 3 files, 2 images and a .csv file, it lacks a user interface, and has to be run from a special Anaconda Python environment below is an example of its outputs :

```
(threejs_env) C:\Users\NETRO\threejs_synthetic_moon>python segment_image.py exp_6.png
135 masks
128 ellipses
Finished in 82.27 seconds

(threejs_env) C:\Users\NETRO\threejs_synthetic_moon>
```

Figure 4.1.2-1 Shows Bruce's Program performance

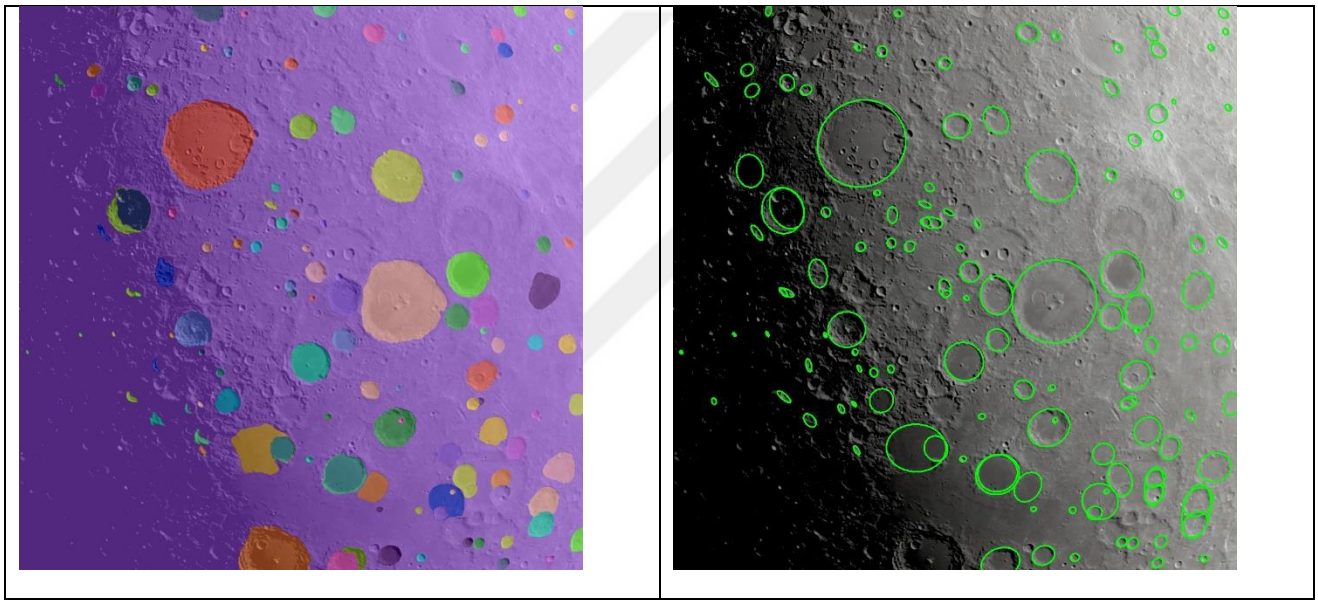


Figure 4.1.2-2 Shows the segmentation masks and Ellipses of Bruce's SAM program

And then we can notice how it took it 82.27 seconds roughly 1 minute and 20 seconds to detect 135 masks (some are overlapping) and later 128 ellipses after using ellipse filter to detect possible craters, it did miss obvious craters, and other small craters.

4.3. Our System's Workflow

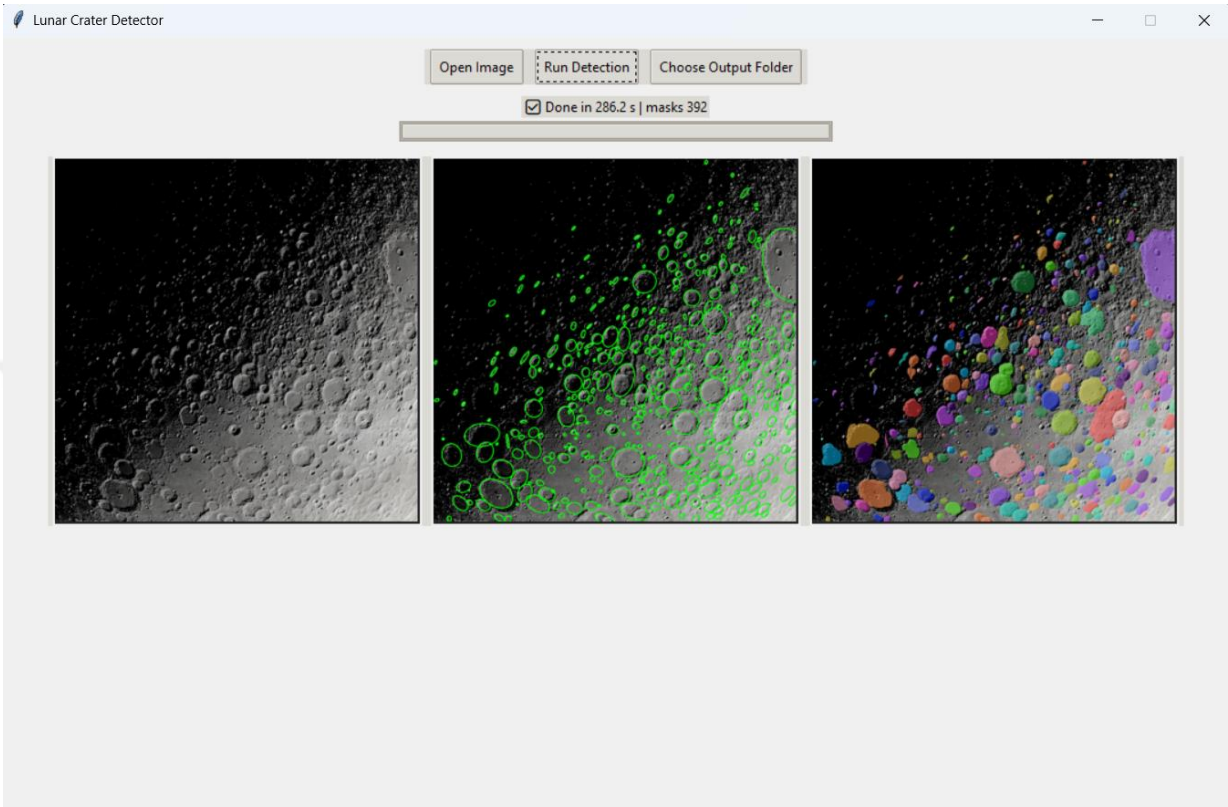
Figure 4.1.2-1 Showing our System Design Flow Chart

The Figure above shows how our program works it starts with the user, uploading an image for crater detections and choosing an output folder, then the user clicks Run Detection.

4.3.1. User Input

First the user inputs an image then chooses an output folder, the image is preferred to be 1024x1024 for optimal results, the image has to be in .PNG or .JPG format.

Figure 4.3.1-1 Showing the GUI of our program, the user inputs an image, chooses output folder and presses the Run Detections button. It later outputs 2 .png images the Ellipses and Segments and the time it took to output the image.



4.3.2. Preprocessing

The user's image is read using OpenCV in BGR Format, there is no other preprocessing done like resizing, or normalization, or grayscale conversion, as SAM will directly operate on the color image

4.3.3. Segmentation with SAM

The user's image is passed to the SAM Segmentor which then will load the ViT-H SAM model using a specified checkpoint, our SAM Segmentor will use different configurational parameters: such as points_per_side, pred_iou_thresh, crop_n_layers, min_mask_region_area, these parameters come from the SAM library.

- points_per_side: which will define resolution of our grid, which will be used to place the prompt points across the image, since SAM uses Prompt Engineering which we

will exploit to improve our detections, for example a value of 32 will result a grid that is 32x32 = 1024 prompt points. The higher the value the more accuracy and false positives and slower the processing becomes, SAM will then try to generate a mask around each point, lower values are faster, but may miss smaller craters.

- pred_iou_thresh: which determines the confidence threshold of the predicted masks, based on Predicted Intersection over Union scores, any mask below certain threshold will be discarded
- crop_n_layers: when it is set to 1 it will split the user's input image into multiple tiles and it will run segmentation on each tile separately, which then will improve detection in small regions but at the cost of additional processing time.
- min_mask_region_area: Which is used filter out the masks that cover very small regions needless to say lower values helps detecting small craters, it can still be useful to remove noise or trivial detections.

4.3.4. Ellipse Fitting

Now an Ellipse fitter is run on each segmentation mask, to determine whether it resembles a crater or not, based on the geometric features of each mask, there are several steps, first we need Contour extraction, then we do Ellipse fitting, both of these processes use OpenCV library, using findContours() and fitEllipse() functions respectively, the fitEllipse() function uses the Least Squares Method, Least squares fitting is mathematical procedure for finding the best fitting curve for a given set of points. The procedure tries to minimize the sum of the squares of the offsets of the points from the curve, the general conic representation of 2D ellipse is given by:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$ **Equation 4.3.4-1**

The use of the Least Squares Method will significantly improve our crater detections, it will reduce false positives by fitting precise ellipses on potential segments, will improve

crater localization thus increasing the accuracy of crater position and dimensions estimation, with the help of parameters will help in detecting small craters, often missed, and then to ensure our conic is an Ellipse the discriminant constraint must be satisfied as in :

$$B^2 - 4AC < 0$$ **Equation 4.3.4-2**

this fitting approach will minimize the algebraic distance subject to this constraint, having a negative discriminant signifies that the equation describes an ellipse rather than a parabola or hyperbola, which in case of ellipse fitting is crucial to ensure the fitted shape is an ellipse.

Explanation of the Least Squares Methods parameters:

A	The coefficient of x^2 controls horizontal curvature
B	Coefficient of xy controls rotation/tilt of the ellipse
C	Coefficient of $control's$ controls vertical curvature
D	Coefficient of x controls horizontal shift
E	Coefficient of y controls vertical shift
F	Constant term acts like an offset to position the conic in space

Table 4.3.4.1 shows the variables of the Least Squares Method Equation.

And *x* and *y* represent the coordinates on a 2D plane. With *x* being the horizontal position of pixels and *y* being the vertical position of pixels.

Next we will use Geometric filters, for the Minimum and Maximum Diameters threshold, the Elongation Constraints in which a ratio between the major and minor axis as he to be below a certain threshold then we have the area ratio, with the fitted ellipse area to be an actual contour the area must lie within acceptable bounds, with Solidity being used to measure contour compactness, then we implement the containment filter which removes nested or overlapping detections using the intersection over union comparison, to avoid overlapping or redundant detections we will use this Equation

$$\boldsymbol{IoU = \frac{|M \cap U|}{|M|}}$$ **Equation 4.3.4-3**

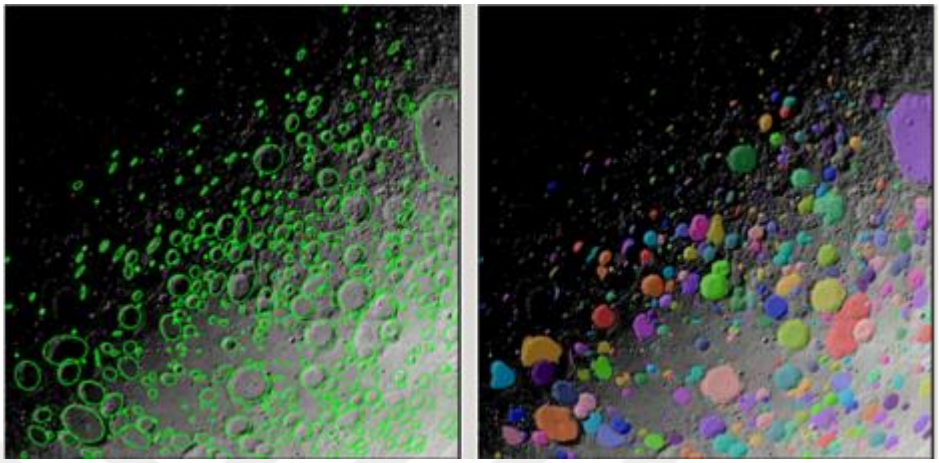
where M is the current mask, and U is the Union of the previously accepted masks, we keep only the ellipses satisfying the rule $IoU < \tau$ where τ is a threshold value defined by us, the intersection over union has to be lower than the threshold τ.

Once an ellipse passes all these tests will be recorded in a Data Frame, to store it's $x\ y$ coordinates, it's major and minor axis, the orientation angle and the optional quality metrics like area ratio and solidity, now these parameters are custom made by us, not from the OpenCV library, they are thresholds defined the EllipseFilter class, they are meant to control the filtering logic we apply after calling cv2.fitEllipse().

4.3.5. Our Program Output

Our output will be 2 .png images with Segments and another image with Ellipses and then a .CSV file.

Figure 4.3.5-1 An Example of our Program's Output, Left is Ellipses Image, and Right is SAM Segments Image, the colors don't mean anything special, they are simply represent an instance mask.



The Segmented mask image is a .png image color coded, which show all raw segmentation masks, the Ellipse overlay Image is another .png image, with the fitted ellipses overlaid on the user's input image, the CSV file is a table containing geometric parameters for each detected ellipse or crater, including the coordinates dimensions and angle, and finally the time it took is displayed on the GUI in seconds.

4.4. User Interface Design

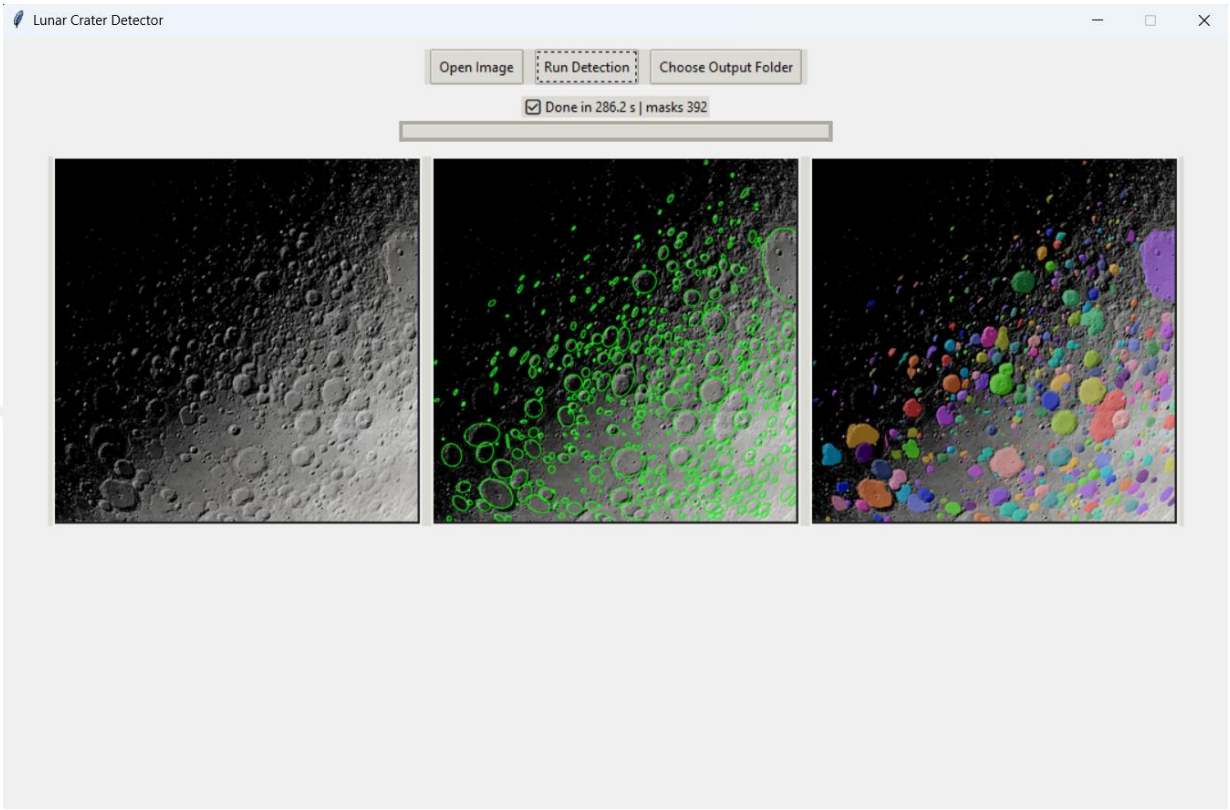


Figure 4.3.5-1 Showing the user interface of our program

Here we see the User Interface of our program, there are three buttons Open Image in which a user selects an image, Choose Output Folder, in which a user chooses the folder of the outputs and the Run Detection Button to start the crater detection operation, there are 3 images which reflect the input image, the output Ellipses Image and the output segments image.

The GUI was built using Tkinter a built in Python library for Graphical User Interface, it provides native looking widgets and windows management for different platforms, it will allow for a quick development of lightweight and interactive user interfaces, without the need for external dependencies.

5. Discussion

In this section, we are going to discuss the results of our programs, analyze the detections in the images, describe the results, compare our results with the previous program, discuss the limitations of our program, and provide recommendations for future work.

5.1. Interpretation of the Results

Our Subject image will be a 1024x1024, the image was generated using Three.js program that uses NASA's MoonKit in rendering of the Lunar Surface, under various heights and lighting conditions, our image was taken from a simulated height between 3000KM and 4000KM, with partially lit and dim and fully lit areas, to properly reflect the shadowing effect on the crater rims.

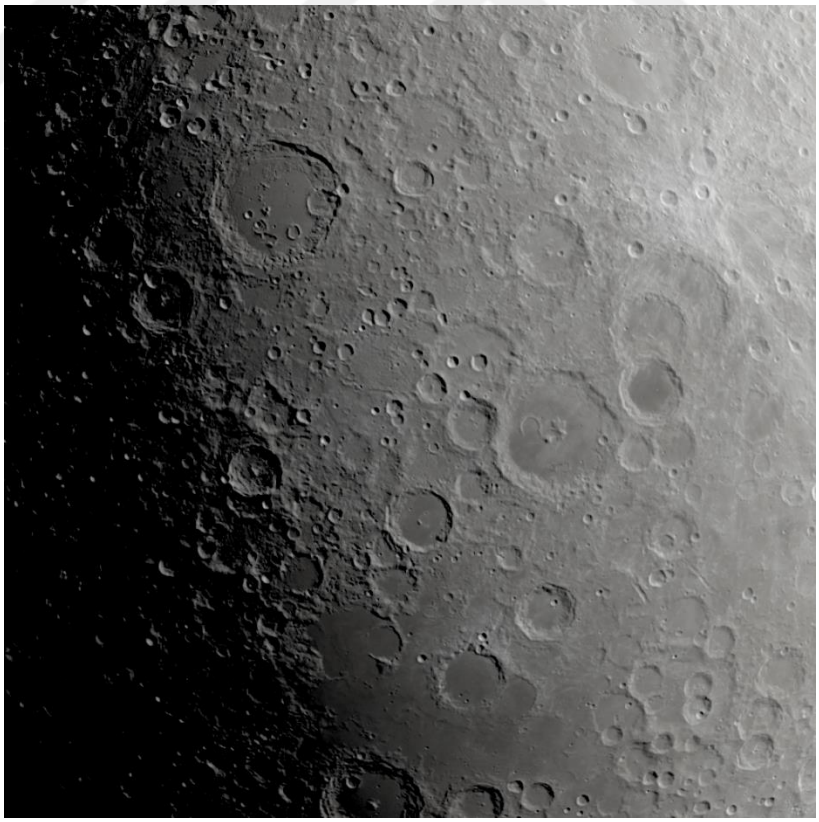


Figure 4.3.5-1 Subject Image where we will run our testing on

We insert this image into our program and choose the output folder and click on the Run Detection Button.

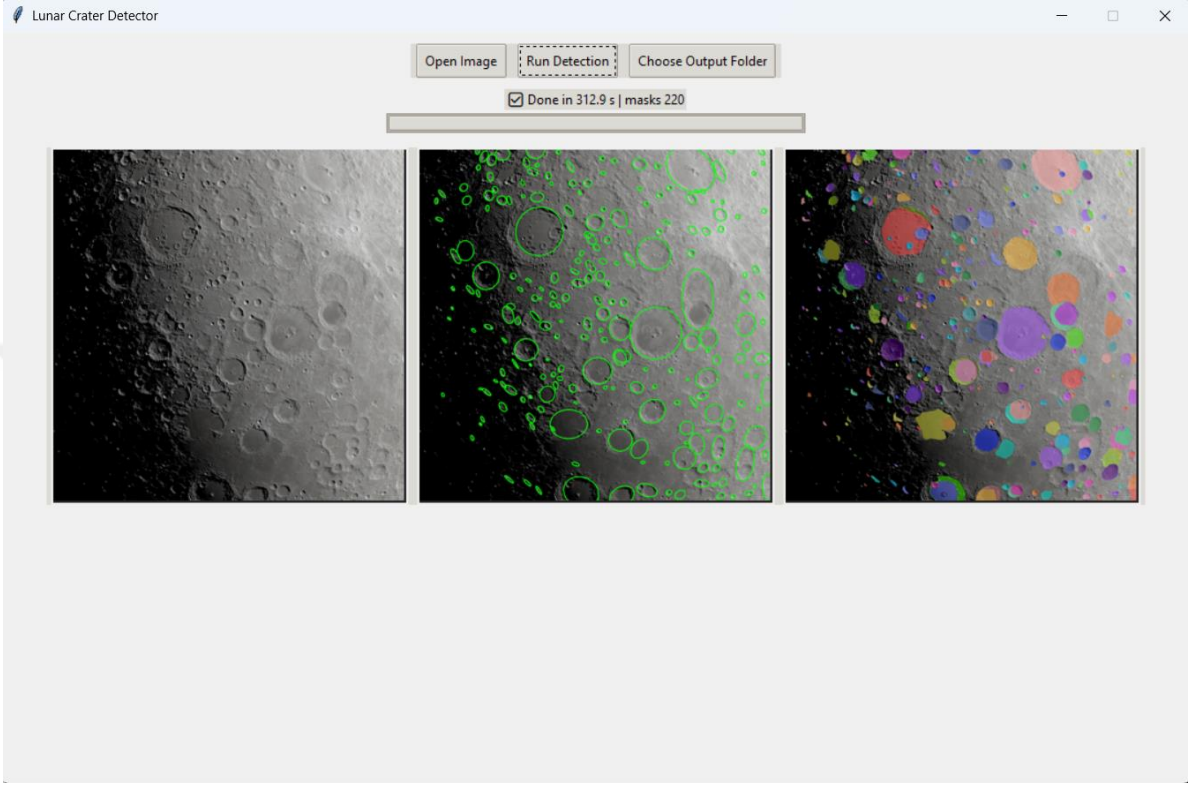


Figure 4.3.5-2 Showing the Results on our Subject image

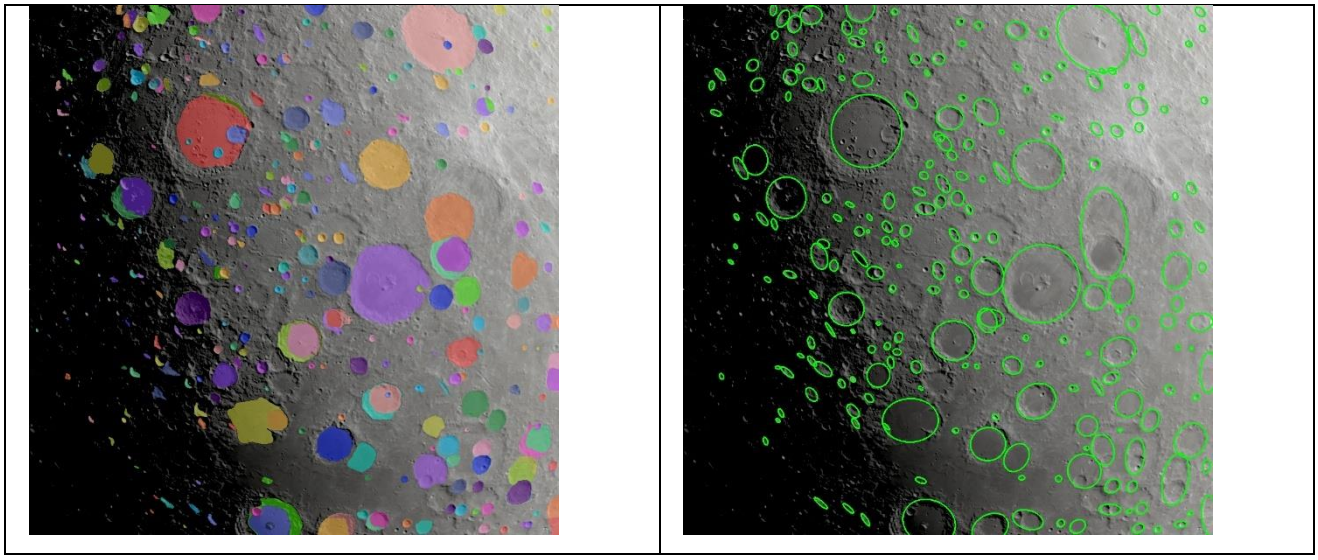


Figure 4.3.5-3 This figure shows the results of our program, Left are SAM Segments, Right are the results of our Ellipse Filter

To evaluate the effectiveness of our proposed Crater Detection Pipeline, we can see that the Majority of the prominent craters detected in the image, both large and midsize craters, with some challenges in overlapping craters, even under dynamic lighting, the Ellipses were fitted over the segmented masks and this shows robustness against segmentation noise, our implemented geometric features played a role in mitigating the segmentation noise, mainly due to our filters, Elongation Filter, Area Ratio Test, Minimum Diameter Threshold helped in lowering the false positives, still some false positives were detected and remained in the output, mainly elongated vertical shapes, the system does struggle with heavily shaded and sometimes with overlapping craters, near the terminator, which highlights one of the limitations of SAM in low contrast regions, most of the false positives include edge artifacts or shallow shaded features, examples in the figure below.

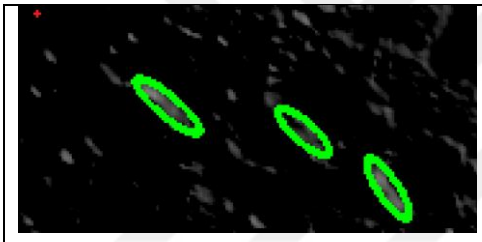
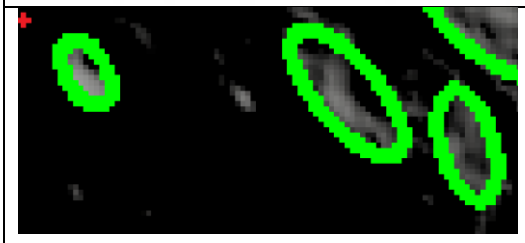
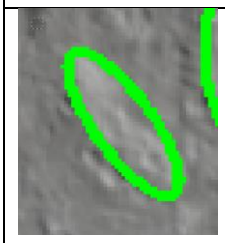
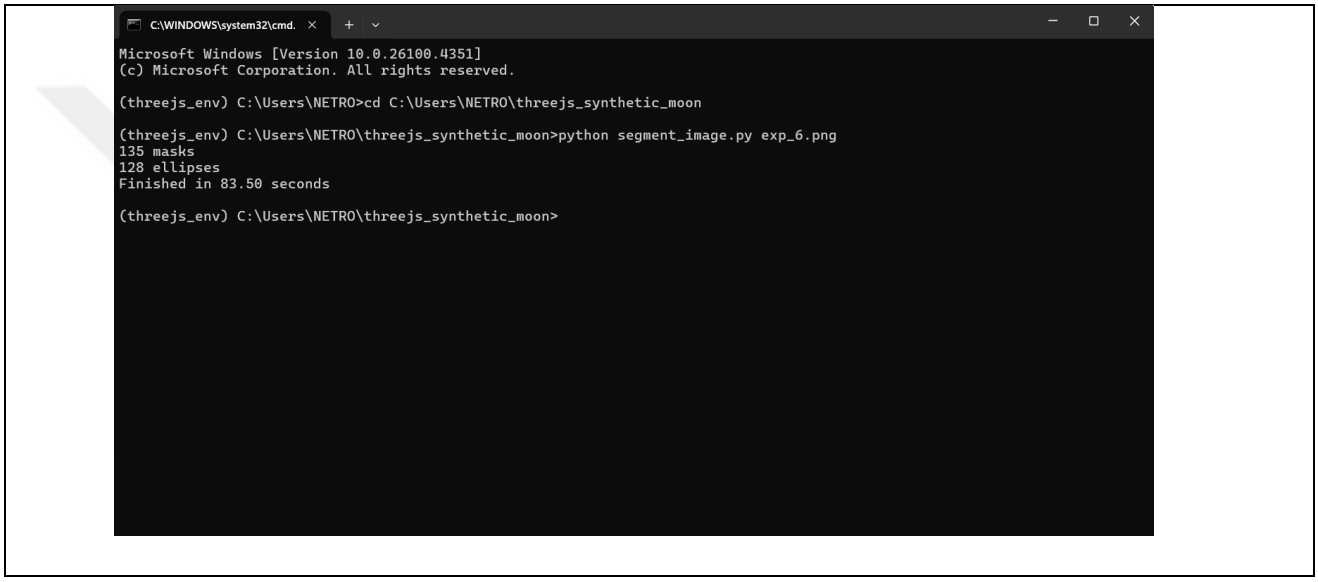
	Examples of false positives of Shallow Shaded Features.
	Another Shallow Shaded Features resulting in False Positives.
	Edge Detected falsely as a crater possibly due to lighting difference compared to the surrounded features.

Figure 4.3.5-4 Showing examples of False Positives in our Ellipse Detector Image

5.2. Comparison with the Old Pipeline

In this section, we will compare the performance of our pipeline with optimized parameters against that of the older one by Callum Bruce. We are going to run both programs on the same image and compare the results.

Here in this image, we can see how many segments there were, and how many elliptical masks were detected:



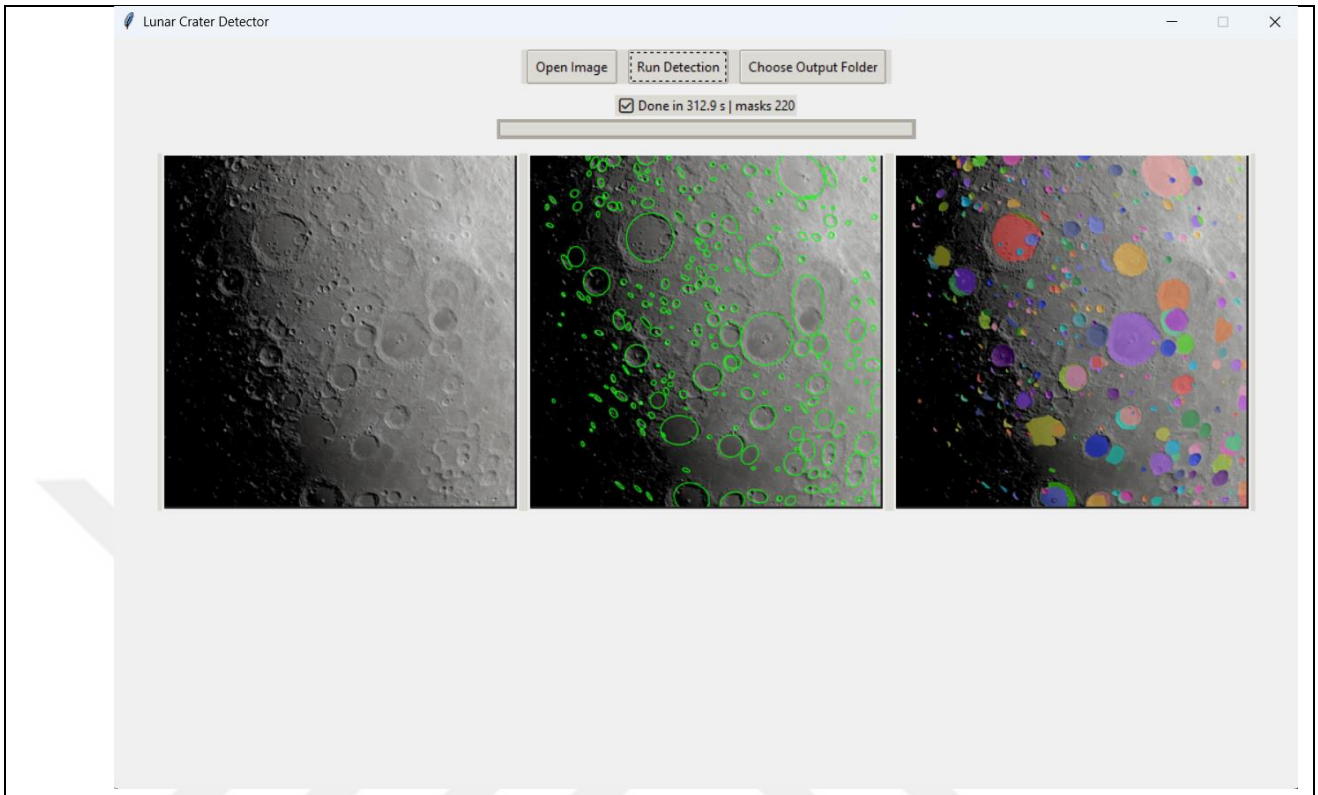
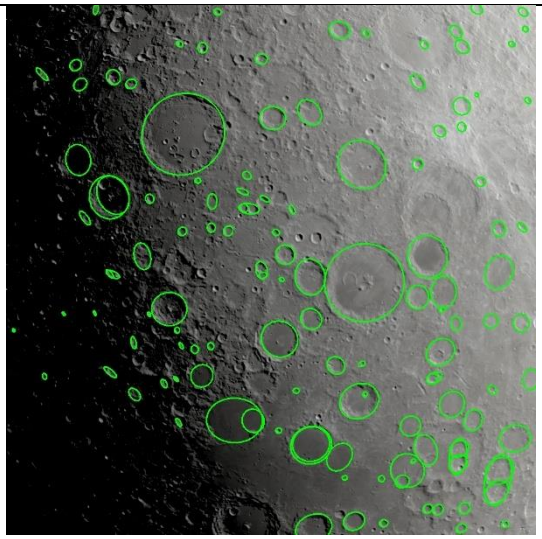


Figure 4.3.5-1 Comparison between old and new pipelines.

It seems to have detected 128 Ellipses compared to our 220 Ellipses. Also, our program took significantly longer, taking 312.9 seconds compared to 83.50 seconds; however, our program could detect many more craters, compared to the old pipeline, as we will see later on in the next figure.



This figure right here shows the Ellipse Filter Results of the Old Pipeline

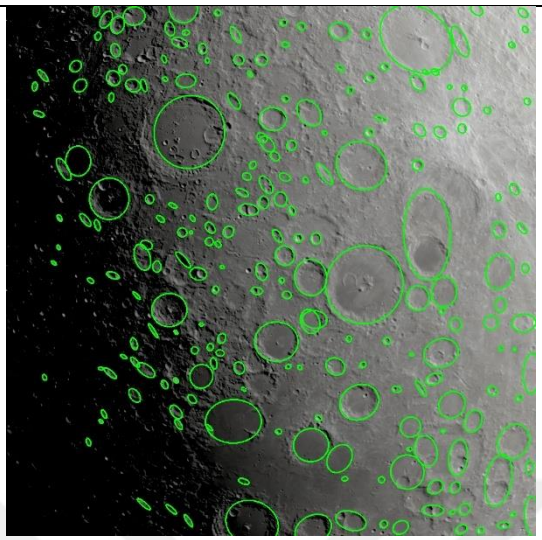
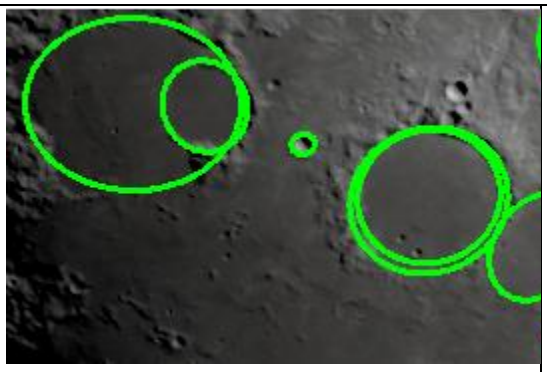
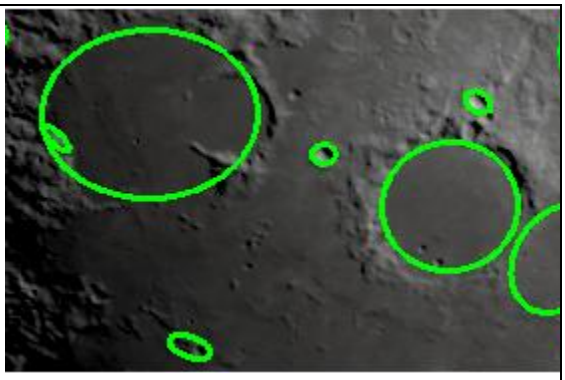
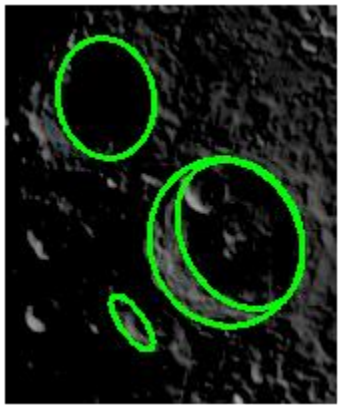
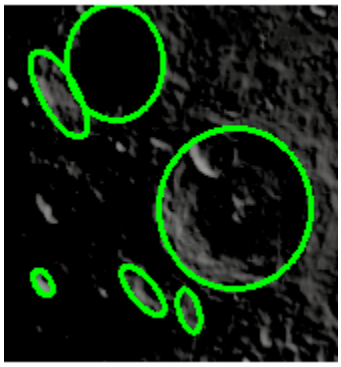
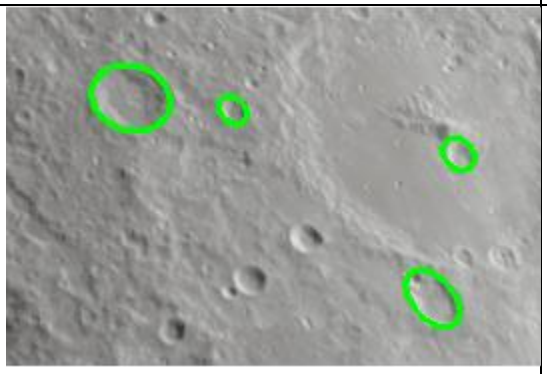
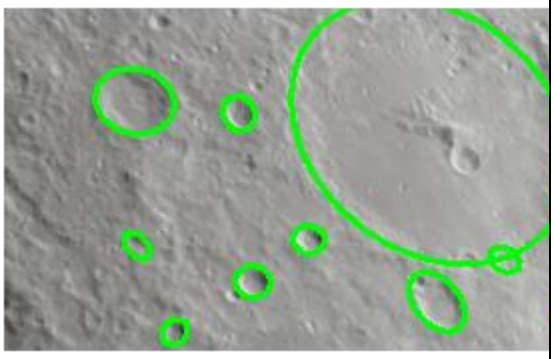
	This figure shows, the results of Ellipse Filter of our new Optimized Program

Figure 4.3.5-2 Showing comparison of old vs New Pipelines

As we can see, our program has indeed detected more craters with our optimizations. Let's look into some of the new results in detail, immediately, it seems our optimized pipeline was able to detect more of the small craters and some of the overlapping craters.

Figure 4.3.5-3 Shows a Detailed Comparison between the old and new pipelines

	
The old pipeline has a problem in detecting the same crater twice due to the changes in lighting on the rims	Our new pipeline has a built-in method to reject double detections and nested ellipses

	
The same problem can be seen again in dim light conditions, highlighting it as a lighting issue; it seems to have detected the same dimly lighted feature as a possible crater	Our new pipeline rejected the double detection and nested detection; however, due to aggressiveness, we can see some false positives, especially in the dimly lighted regions
	
Here we can see an old pipeline completely missing an obvious crater, along with other small craters	Here we can see our optimized pipeline detecting the large crater and some of the surrounding small craters; however, it detected two adjacent craters as a single crater

5.3. Limitation of SAM in Lunar Crater Detection

We can clearly see that SAM has got some potential in detecting craters, it is still not possible to fully determine the accuracy of our CDA from a single example, Lighting, Camera pose, and Camera position all affect the accuracy of the CDA, at the end SAM is a general purpose segmentation model, that even when optimized still lacks domain specific knowledge of lunar topology or geological features like craters, ridges or mare/highland boundaries, it still lacks an understand of shadows or illumination leading to countless false positives, craters heavily rely on shading and lighting cues, SAM treats them as flat color or textural regions, missing subtle craters near the terminator or misclassifying shadows as possible candidates, SAM is also sensitive to noise or resolution, which may lead to over segmentation, since it is tuned for the appearance of pixels not geometry, it cannot infer 3D shape or Depth, while post processing with ellipse filters and geometric heuristics does help in isolating crater like objects, it is still far from perfect, and more testing and tweaking is needed, we also need to acknowledge that SAM works per image only, and for large scale mapping or mosaicking it can’t enforce spatial continuation or crater persistence across tiles.

6. Conclusion

At the end we can notice that using SAM for crater detection does yield good results for annotating datasets and for crater detections, however, it's results are far from perfect and issues with lighting and shadowing remains an obstacle for robust detections, the computational intensity is also another problem which can be improved upon with better hardware, the false positives in detecting other terrain features as craters like ridges, remains a problem to be solved, however implementing an ellipse filter, seems to help us mitigate such issue, the use of an Ellipse Filter filters helps greatly in mitigating false positives and improving overall accuracy, giving us a better annotated datasets for developing and training a U-Net CNN later on, for either crater detection or as a module in a program for landing spot detections, SAM alone is not sufficient for robust crater detection under all lunar conditions unless, we have to optimize the parameters more, which by then brings us the computational limitation issue, it can however be integrated with domain specific filtering techniques, to present a viable hybrid approach, which can also lay the groundwork for further research into semi supervised annotation generation, it can further help in the research for domain adaptation of foundation models like SAM, and later development of automated lunar analysis tools to support possibly future space missions.

For Future Work, researchers can improve SAM's performance in crater detection by extensively tuning various parameters, this can have an effect in improving the robustness across multiple or diverse lunar terrains with varying lighting and shadow conditions, using custom ellipse fitters like what we did, or improving the parameters can have great impact on detection accuracy, to minimize false positives in misclassifying rocks, checking the diameter of an Ellipse to see if it fits a crater, while taking into account the angle on the lunar surface, can help mitigate stretched ellipse detections which were most of the false positives in our case, rocky features can be ignored through some parameters that deal with lighting of the pixels and surrounding pixels, but we need to take into account the computational intensity that comes with this approach, Researchers are recommended to continue research into hybrid frameworks to combine SAM's

segmentation capabilities with U-Net or other CNN which are advanced deep learning architectures, which would lead to more accurate and reliable lunar crater detections, thus supporting future planetary exploration missions.

REFERENCES

(1) A. Christian, J., Derksen, H., & Watkins, R. (2020). Lunar Crater Identification in Digital Images. 10.1007/s40295-021-00287-8

(2) Alfredo, R. (2021, June). A robust crater matching algorithm for autonomous vision-based spacecraft navigation. In 2021 IEEE 8th International Workshop on Metrology for AeroSpace (MetroAeroSpace) (pp. 322-327). IEEE. 10.1109/MetroAeroSpace51421.2021.9511670

(3) Anaconda Navigator https://www.anaconda.com/products/navigator

(4) Callum Bruce (2024) Lunar Crater Detection: Computer Vision in Space | Towards Data Science

(5) Chaini, C. & Jha, V. K. (2024). A review on deep learning-based automated lunar crater detection. Earth Science Informatics. DOI: 10.1007/s12145-024-01396-2

(6) Cheng, B., Parkhi, O., & Kirillov, A. (2022). Pointly-supervised instance segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2617-2626). [2104.06404] Pointly-Supervised Instance Segmentation

(7) de Carvalho, O. L. F., de Carvalho Júnior, O. A., Silva, C. R. E., de Albuquerque, A. O., Santana, N. C., Borges, D. L., ... & Guimarães, R. F. (2022). Panoptic segmentation meets remote sensing. Remote Sensing, 14(4), 965. https://doi.org/10.3390/rs14040965

(8) Elharrouss, O., Al-Maadeed, S., Subramanian, N., Ottakath, N., Almaadeed, N., & Himeur, Y. (2021). Panoptic segmentation: A review. arXiv preprint arXiv:2111.10250. https://doi.org/10.48550/arXiv.2111.10250

(9) Emami, E., Ahmad, T., Bebis, G., Nefian, A., & Fong, T. (2019). Crater Detection Using Unsupervised Algorithms and Convolutional Neural Networks. *IEEE Transactions on Geoscience and Remote Sensing*, *57*(8), 5373-5383 https://doi.org/10.1109/TGRS.2019.2899122

(10) Francis, B. (2022). Lunar Resources: Geosensing for Autonomous Exploration. Student Exemplar Repository

(11) Giannakis, I., Bhardwaj, A., Sam, L., & Leontidis, G. (2023). A flexible deep learning crater detection scheme using Segment Anything Model (SAM). *Icarus*. https://doi.org/10.1016/j.icarus.2023.115797

(12) Hiesinger, H., Van der Bogert, C. H., Michael, G., Schmedemann, N., Iqbal, W., Robbins, S. J., ... & Head III, J. W. (2023). The lunar cratering chronology. Reviews in Mineralogy and Geochemistry, 89(1), 401-451. https://doi.org/10.2138/rmg.2023.89.10

(13) SuperAnnotate. (2023, August 17). Image segmentation for machine learning: Techniques, applications, and tools. SuperAnnotate. https://www.superannotate.com/blog/image-segmentation-for-machine-learning

(14) Jain, J., Singh, A., Orlov, N., Huang, Z., Li, J., Walton, S., & Shi, H. (2023). Semask: Semantically masked transformers for semantic segmentation. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 752-761). https://doi.org/10.48550/arXiv.2112.12782

(15) Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. Electronic markets. https://doi.org/10.1007/s12525-021-00475-2

(16) Jia, Y., Liu, L., Peng, S., Feng, M., & Wan, G. (2022). An efficient high-resolution global–local network to detect lunar features for space energy discovery. Remote Sensing. https://doi.org/10.3390/rs14061391

(17) Kheradmandi, N. & Mehranfar, V. (2022). A critical review and comparative study on image segmentation-based techniques for pavement crack detection. Construction and Building Materials. https://doi.org/10.1016/j.conbuildmat.2021.126162

(18) Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., & Girshick, R. (2023). Segment Anything. *arXiv, 2304.02643*

(19) Lee, C. & Hogan, J. (2021). Automated crater detection with human level performance. Computers & Geosciences. https://doi.org/10.48550/arXiv.2010.12520

(20) Lin, X., Zhu, Z., Yu, X., Ji, X., Luo, T., Xi, X., Zhu, M., & Liang, Y. (2022). Lunar crater detection on digital elevation model: a complete workflow using deep learning and its application. Remote sensing. https://doi.org/10.3390/rs14030621

(21) Meta's Segment Anything Model Segment Anything | Meta AI

(22) NASA's MoonKit: https://svs.gsfc.nasa.gov/4720/

(23) NumPy https://numpy.org/

(24) OpenCV https://opencv.org/

(25) Patil, R. & Aggarwal, R. (2023). Comprehensive Review on Image Segmentation Applications. Available at SSRN 5227277.

(26) PyTorch https://pytorch.org/

(27) Silvestrini, S., Piccinin, M., Zanotti, G., Brandonisio, A., Bloise, I., Feruglio, L., ... & Varile, M. (2022). Optical navigation for Lunar landing based on Convolutional Neural Network crater detector. Aerospace Science and Technology, 123, 107503. 10.1016/j.ast.2022.107503

(28) Sun, R., Lei, T., Chen, Q., Wang, Z., Du, X., Zhao, W., & Nandi, A. K. (2022). Survey of image edge detection. Frontiers in Signal Processing, 2, 826967. https://doi.org/10.3389/frsip.2022.826967

(29) Supervision GitHub - roboflow/supervision: We write your reusable computer vision tools. 💜

(30) Tewari, A. & Khanna, N. (2024). Arbitrary Scale Super-Resolution Assisted Lunar Crater Detection in Satellite Images. https://doi.org/10.48550/arXiv.2402.05068

(31) Tewari, A., Prateek, K., Singh, A., & Khanna, N. (2023). Deep learning based systems for crater detection: A review. arXiv preprint arXiv:2310.07727. https://doi.org/10.48550/arXiv.2310.07727

(32) Tewari, A., Verma, V., Srivastava, P., Jain, V., & Khanna, N. (2022). Automated crater detection from co-registered optical images, elevation maps and slope maps using deep learning. Planetary and Space Science, 218, 105500. https://doi.org/10.48550/arXiv.2012.15281

(33) Three.js Javascript 3D Library https://threejs.org/

(34) Tkinter Python library for GUI https://docs.python.org/3/library/tkinter.html

(35) Wang, H., Jiang, J., & Zhang, G. (2018). CraterIDNet: An End-to-End Fully Convolutional Neural Network for Crater Detection and Identification in Remotely Sensed Planetary Images. *Remote Sensing*, *10*(7), 1067. https://doi.org/10.3390/rs10071067

(36) Yu, Y., Wang, C., Fu, Q., Kou, R., Huang, F., Yang, B., ... & Gao, M. (2023). Techniques and challenges of image segmentation: A review. Electronics, 12(5), 1199. https://doi.org/10.3390/electronics12051199

(37) Yue, Z., Shi, K., Di, K., Lin, Y., & Gou, S. (2023). Progresses and prospects of impact crater studies. Science China Earth Sciences. https://doi.org/10.1007/s11430-022-1009-0

(38) Zhang, C., Liu, L., Cui, Y., Huang, G., Lin, W., Yang, Y., & Hu, Y. (2023). A comprehensive survey on segment anything model for vision and beyond. arXiv preprint arXiv:2305.08196. https://doi.org/10.48550/arXiv.2305.08196

(39) Zhou, Y., Li, X., & Hua, B. (2023). Crater identification simulation using LiDAR on Lunar rover. Measurement. https://dx.doi.org/10.2139/ssrn.4250169

(40) Zhou, Y., Zhao, H., Chen, M., Tu, J., & Yan, L. (2018). Automatic detection of lunar craters based on DEM data with the terrain analysis method. *Planetary and Space Science, 160*, 1–11 10.1016/j.pss.2018.03.003

APPENDIX . Code Listings

A. Three.js Code used for generating the Synthetic Images main.js

```
import * as THREE from 'three';
import * as dat from 'dat.gui';

// Helper function to convert spherical coordinates to cartesian coordinates
function sphericalToCartesian(longitude, latitude, radius) {
    const phi = (90 - latitude) * (Math.PI / 180);
    const theta = (360 - longitude) * (Math.PI / 180);

    const x = radius * Math.sin(phi) * Math.cos(theta);
    const y = radius * Math.cos(phi);
    const z = radius * Math.sin(phi) * Math.sin(theta);

    return { x, y, z };
}

// Setup scene, renderer, camera and light
const scene = new THREE.Scene();

const renderer = new THREE.WebGLRenderer({
    antialias: true,
    encoding: THREE.sRGBEncoding,
    toneMapping: THREE.ACESFilmicToneMapping
});
renderer.setSize(1024, 1024);
document.body.appendChild(renderer.domElement);

const camera = new THREE.PerspectiveCamera(75, 1, 0.1, 10000);
camera.position.z = 3000;
camera.setFocalLength(35);
let cameraControlsObject = {
    longitude: 0,
    latitude: 0,
    radius: 3000,
    lookAtCenter: true,
    rotationX: 0,
```

```
    rotationY: 90,
    rotationZ: 0,
};

const light = new THREE.DirectionalLight(0xffffff, 5)
light.position.set(2000, 0, 2000);
light.castShadow = true;
light.shadow.mapSize.width = 2048;
scene.add(light);
let lightControlsObject = { longitude: 0, latitude: 0, intensity: 5};

// Setup moon mesh
const geometry = new THREE.SphereGeometry(1728.28, 4000, 2000);
const textureLoader = new THREE.TextureLoader();
const texture = textureLoader.load('/moon_texture.png');
const displacement = textureLoader.load('/scaled_moon_displacement.png');
const normal = textureLoader.load('/moon_normal_map.png');

texture.minFilter = THREE.LinearFilter;
displacement.minFilter = THREE.LinearFilter;
normal.minFilter = THREE.LinearFilter;

const scale = 19.87;

const material = new THREE.MeshLambertMaterial({
    color: 0xffffff,
    map: texture,
    displacementMap: displacement,
    displacementScale: scale,
    normalMap: normal,
    normalScale: new THREE.Vector2(1.0, 1.0),
    reflectivity: 0,
    shininess: 0
});

const mesh = new THREE.Mesh(geometry, material);
scene.add(mesh);
const meshControlsObject = { rotationX: 0, rotationY: 0, rotationZ: 0, displacementScale:
scale, normalScale: 1.0 };

// Variables for screenshot automation
let batchIndex = parseInt(localStorage.getItem('screenshotBatch') || '0') + 1;
localStorage.setItem('screenshotBatch', batchIndex);

const batchOffset = batchIndex * 1000;
```

```
let autoRunning = false;
let autoFrame = 0;
const totalFrames = 100;
const screenshotInterval = 10000; // 10 seconds
const meshRotationInterval = 2000; // 2 seconds
let lastRotationTime = 0;
let lastScreenshotTime = 0;

// Setup GUI controls
const gui = new dat.GUI();

// Light controls
const lightControls = gui.addFolder('Light Controls');
const lightLongitude = lightControls.add(lightControlsObject, 'longitude', -180,
180).name('Longitude');
const lightLatitude = lightControls.add(lightControlsObject, 'latitude', -90,
90).name('Latitude');
const lightIntensity = lightControls.add(lightControlsObject, 'intensity', 0,
10).name('Intensity');
lightControls.open();

// Camera controls
const cameraControls = gui.addFolder('Camera Controls');
const cameraLongitude = cameraControls.add(cameraControlsObject, 'longitude', -180,
180).name('Longitude');
const cameraLatitude = cameraControls.add(cameraControlsObject, 'latitude', -90,
90).name('Latitude');
const cameraRadius = cameraControls.add(cameraControlsObject, 'radius', 0,
10000).name('Radius');
const cameraLookAtCenter = cameraControls.add(cameraControlsObject,
'lookAtCenter').name('Look At Center');
const cameraRotationX = cameraControls.add(cameraControlsObject, 'rotationX', -180,
180).name('X Rotation');
const cameraRotationY = cameraControls.add(cameraControlsObject, 'rotationY', -180,
180).name('Y Rotation');
const cameraRotationZ = cameraControls.add(cameraControlsObject, 'rotationZ', -180,
180).name('Z Rotation');
cameraControls.open();

// Mesh controls
const meshControls = gui.addFolder('Mesh Controls');
const meshRotationX = meshControls.add(meshControlsObject, 'rotationX', -180,
180).name('X Rotation');
const meshRotationY = meshControls.add(meshControlsObject, 'rotationY', -180,
180).name('Y Rotation');
```

```
const meshRotationZ = meshControls.add(meshControlsObject, 'rotationZ', -180, 180).name('Z Rotation');
const displacementScale = meshControls.add(meshControlsObject, 'displacementScale', 0, 2000).name('Displacement Scale');
const normalScale = meshControls.add(meshControlsObject, 'normalScale', 0, 10).name('Normal Scale');
meshControls.open();

// Function to update light controls from GUI sliders
function updateLightControlsFromGUI() {
    const longitude = lightLongitude.getValue();
    const latitude = lightLatitude.getValue();
    const radius = 1;
    light.intensity = lightIntensity.getValue();
    const { x, y, z } = sphericalToCartesian(longitude, latitude, radius);
    light.position.x = x;
    light.position.y = y;
    light.position.z = z;
}

// Function to enable/disable camera rotation controls based on the value of lookAtCenter
function toggleCameraControls(enabled) {
    cameraRotationX.__li.style.pointerEvents = enabled ? 'auto' : 'none';
    cameraRotationY.__li.style.pointerEvents = enabled ? 'auto' : 'none';
    cameraRotationZ.__li.style.pointerEvents = enabled ? 'auto' : 'none';
}

// Function to update camera controls from GUI sliders
function updateCameraPositionControlsFromGUI() {
    const longitude = cameraLongitude.getValue();
    const latitude = cameraLatitude.getValue();
    const radius = cameraRadius.getValue();
    const { x, y, z } = sphericalToCartesian(longitude, latitude, radius);
    camera.position.x = x;
    camera.position.y = y;
    camera.position.z = z;
    const lookAtCenter = cameraLookAtCenter.getValue();
    if (lookAtCenter) {
        camera.lookAt(0, 0, 0);
    }
    toggleCameraControls(!lookAtCenter)
    camera.updateProjectionMatrix();
}

// Function to update camera rotation controls from GUI sliders
```

```
function updateCameraRotationControlsFromGUI() {
    const rotationX = cameraRotationX.getValue();
    const rotationY = cameraRotationY.getValue();
    const rotationZ = cameraRotationZ.getValue();
    camera.rotation.x = rotationX * Math.PI/180;
    camera.rotation.y = rotationY * Math.PI/180;
    camera.rotation.z = rotationZ * Math.PI/180;
    camera.updateProjectionMatrix();
}

// Function to update material controls from GUI sliders
function updateMeshControlsFromGUI() {
    const rotationX = meshRotationX.getValue();
    const rotationY = meshRotationY.getValue();
    const rotationZ = meshRotationZ.getValue();
    const normalScaleValue = normalScale.getValue();
    mesh.rotation.x = rotationX * Math.PI/180;
    mesh.rotation.y = rotationY * Math.PI/180;
    mesh.rotation.z = rotationZ * Math.PI/180;
    material.displacementScale = displacementScale.getValue();
    material.normalScale = new THREE.Vector2(normalScaleValue, normalScaleValue);
}

// Event listeners for GUI control changes
lightLongitude.onChange(updateLightControlsFromGUI);
lightLatitude.onChange(updateLightControlsFromGUI);
lightIntensity.onChange(updateLightControlsFromGUI);

cameraLongitude.onChange(updateCameraPositionControlsFromGUI);
cameraLatitude.onChange(updateCameraPositionControlsFromGUI);
cameraRadius.onChange(updateCameraPositionControlsFromGUI);
cameraLookAtCenter.onChange(updateCameraPositionControlsFromGUI);

cameraRotationX.onChange(updateCameraRotationControlsFromGUI);
cameraRotationY.onChange(updateCameraRotationControlsFromGUI);
cameraRotationZ.onChange(updateCameraRotationControlsFromGUI);

meshRotationX.onChange(updateMeshControlsFromGUI);
meshRotationY.onChange(updateMeshControlsFromGUI);
meshRotationZ.onChange(updateMeshControlsFromGUI);
displacementScale.onChange(updateMeshControlsFromGUI);
normalScale.onChange(updateMeshControlsFromGUI);

function saveScreenshot(filename) {
    renderer.render(scene, camera);
```

```
    renderer.domElement.toBlob(blob => {
        const a = document.createElement('a');
        a.href = URL.createObjectURL(blob);
        a.download = `screenshots/${filename}.png`;
        a.click();
    }, 'image/png');
}

// Automation logic (called in animate loop)
function automatedControlLoop(time) {
    if (!autoRunning || autoFrame >= totalFrames) return;

    // Screenshot every 10 seconds
    if (time - lastScreenshotTime >= screenshotInterval) {
        const lon = -180 + (autoFrame * 360 / totalFrames);               // longitude
sweeps full orbit
        const lat = Math.sin(autoFrame * 0.1) * 20;                        // latitude
oscillates ±20°
        const radius = 3500 + 500 * Math.cos(autoFrame * 0.1);            // radius
oscillates [3000, 4000]

        const camPos = sphericalToCartesian(lon, lat, radius);
        camera.position.set(camPos.x, camPos.y, camPos.z);
        camera.lookAt(0, 0, 0);

        const lightLon = (lon + 90) % 360;
        const lightLat = Math.cos(autoFrame * 0.1) * 45;
        const lightPos = sphericalToCartesian(lightLon, lightLat, 2000);
        light.position.set(lightPos.x, lightPos.y, lightPos.z);

        const name = `moon_${String(batchOffset + autoFrame).padStart(4, '0')}`;
        saveScreenshot(name);
        console.log(`Captured: ${name}.png`);
        autoFrame++;
        lastScreenshotTime = time;
    }

    // Rotate mesh every 3 seconds
    if (time - lastRotationTime >= meshRotationInterval) {
        mesh.rotation.y += THREE.MathUtils.degToRad(5); // smooth step
        lastRotationTime = time;
    }
}

// Start and stop automation buttons
```

```
gui.add({     takeScreenshot:     ()     =>     saveScreenshot("manual_capture")     },
'takeScreenshot').name('Manual Screenshot');
gui.add({
    startAutomation: () => {
        autoRunning = true;
        autoFrame = 0;
        lastScreenshotTime = 0;
        lastRotationTime = 0;
    }
}, 'startAutomation').name('Start Auto Screenshots');
gui.add({ stopAutomation: () => { autoRunning = false; } }, 'stopAutomation').name('⛔
Stop Auto Screens');

function animate(time) {
    requestAnimationFrame(animate);
    automatedControlLoop(time); // handles screenshot and mesh rotation
    renderer.render(scene, camera);
}
animate();
```

B. The Segmentor Code used to import the Segment Anything Model with parameters

```
# app/segmentor.py
from segment_anything import sam_model_registry, SamAutomaticMaskGenerator
import supervision as sv
import torch


class FastSamSegmentor:

    def __init__(self, checkpoint_path: str, device: str = "cuda"):
        sam = sam_model_registry["vit_h"](checkpoint=checkpoint_path)
        sam.to(device)

        self.mask_gen = SamAutomaticMaskGenerator(
            sam,
            points_per_side=48,
            crop_n_layers=1,
            pred_iou_thresh=0.86,
            crop_n_points_downscale_factor=2,
            min_mask_region_area=5,
        )

    # ------------------------------------------------------------------ #
    def segment(self, img_bgr, progress_callback=lambda p: None):
        """Return supervision.Detections."""
        with torch.inference_mode():
            masks = self.mask_gen.generate(img_bgr)
        progress_callback(1.0)
        return sv.Detections.from_sam(masks)
```

C. The Ellipse Filter used to run our operation on Segmented Masks using Least Squares Method

```python
# app/ellipse_filter.py
import numpy as np
import cv2, pandas as pd, supervision as sv


class EllipseFilter:
    """
    Rejects masks that are too small/large, too elongated, too irregular,
    too dark, or completely contained in a larger crater.
    """

    # ------------------------------------------------------------------ #
    # DEFAULT THRESHOLDS — adjust if you like
    LOWER_AREA_RATIO = 0.50
    UPPER_AREA_RATIO = 2.80
    MIN_DIAMETER_PX  = 6
    MAX_DIAMETER_PX  = 1_000
    MAX_ELONGATION   = 1.15
    CONTAINMENT_IOU  = 0.50
    MIN_BRIGHTNESS   = 0

    # ------------------------------------------------------------------ #
    def __init__(
        self,
        lower_area_ratio: float = LOWER_AREA_RATIO,
        upper_area_ratio: float = UPPER_AREA_RATIO,
        min_diameter: int = MIN_DIAMETER_PX,
        max_diameter: int = MAX_DIAMETER_PX,
        max_elongation: float = MAX_ELONGATION,
        containment_iou: float = CONTAINMENT_IOU,
        min_brightness: int = MIN_BRIGHTNESS,
    ):
        self.lower_area_ratio = lower_area_ratio
        self.upper_area_ratio = upper_area_ratio
        self.min_diameter     = min_diameter
        self.max_diameter     = max_diameter
        self.max_elongation   = max_elongation
        self.containment_iou  = containment_iou
```

```
        self.min_brightness    = min_brightness

    # ------------------------------------------------------------------ #
    def _bright_enough(self, mask, img_bgr):
        gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
        vals = gray[mask.astype(bool)]
        return vals.size and vals.mean() > self.min_brightness

    # ------------------------------------------------------------------ #
    @staticmethod
    def _ellipse(cnt):
        (x, y), (major, minor), angle = cv2.fitEllipse(cnt)
        return dict(x=x, y=y, majorAxis=major, minorAxis=minor, angle=angle)

    # ------------------------------------------------------------------ #
    def filter_ellipses(self, detections: sv.Detections, image=None):
        kept_idx, rows = [], []

        for idx, mask in enumerate(detections.mask):
            cnts, _ = cv2.findContours(mask.astype(np.uint8),
                                       cv2.RETR_EXTERNAL,
                                       cv2.CHAIN_APPROX_SIMPLE)
            if not cnts:
                continue
            cnt = max(cnts, key=cv2.contourArea)
            if len(cnt) < 5:
                continue

            ell  = self._ellipse(cnt)
            maj, min_ = ell["majorAxis"], ell["minorAxis"]

            # diameter / elongation tests
            if not (self.min_diameter <= maj <= self.max_diameter):
                continue
            if min_ == 0 or (maj / min_) > self.max_elongation:
                continue

            # roundness test
            area_e = np.pi * (maj / 2) * (min_ / 2)
            area_c = cv2.contourArea(cnt) or 1
            ratio  = area_e / area_c
            if not (self.lower_area_ratio <= ratio <= self.upper_area_ratio):
                continue

            # optional brightness gate
```

```
        if image is not None and not self._bright_enough(mask, image):
            continue

        kept_idx.append(idx); rows.append(ell)

    kept_idx, rows = self._suppress_nested(kept_idx, rows, detections)
    return detections[kept_idx], pd.DataFrame(rows)

# ------------------------------------------------------------------ #
def _suppress_nested(self, idx_list, row_list, detections):
    entries = sorted(zip(idx_list, row_list),
                     key=lambda t: t[1]["majorAxis"] * t[1]["minorAxis"],
                     reverse=True)
    final_idx, final_rows = [], []
    union = np.zeros_like(detections.mask[0], dtype=bool)

    for idx, row in entries:
        m = detections.mask[idx].astype(bool)
        inter = np.logical_and(union, m).sum()
        if inter / m.sum() < self.containment_iou:
            final_idx.append(idx)
            final_rows.append(row)
            union = np.logical_or(union, m)

    return final_idx, final_rows
```

D- Code for Generating Normal Map from Displacement Map Credit Callum Bruce

```
import sys
import numpy as np
from PIL import Image

Image.MAX_IMAGE_PIXELS = 1000000000

def calculate_normal_map(displacement_map):
    print('Calculating normal map...')
    # Calculate the gradient using central differences
    dz_dx = np.gradient(displacement_map, axis=1)
    dz_dy = np.gradient(-displacement_map, axis=0)

    # Create grid of coordinates
    x = np.arange(displacement_map.shape[0])
    y = np.arange(displacement_map.shape[1])
    xx, yy = np.meshgrid(x, y, indexing='ij')

    # Calculate the tangent vectors
    tangent_x = np.stack((np.ones_like(xx), np.zeros_like(xx), dz_dx), axis=-1)
    tangent_y = np.stack((np.zeros_like(yy), np.ones_like(yy), dz_dy), axis=-1)

    # Calculate the normal vector
    normal_vector = np.cross(tangent_x, tangent_y)
    normal_vector /= np.linalg.norm(normal_vector, axis=-1, keepdims=True)

    return normal_vector

def export_normal_map(normal_map, file_path):
    print('Exporting normal map...')
    # Scale normal vector components to range [0, 255]
    scaled_normal_map = (normal_map + 1) * 127.5

    # Convert to uint8 and create image
    normal_image = Image.fromarray(scaled_normal_map.astype(np.uint8))

    # Save the image
    normal_image.save(file_path + '.png')

def main(input_file, output_file):
    # Load the displacement map
```

```
    displacement_map = np.array(Image.open(input_file))

    # Scale the displacement map
    pixel_length_km = ((2 * np.pi * 1737.1) / 360) / 64 # 64 pixels per degree
    scaled_displacement_map = displacement_map / pixel_length_km

    # Calculate the normal map
    normal_map = calculate_normal_map(scaled_displacement_map)

    # Export the normal map
    export_normal_map(normal_map, output_file)

if __name__ == "__main__":
    input_file = sys.argv[1]
    output_file = sys.argv[2]
    main(input_file, output_file)
```