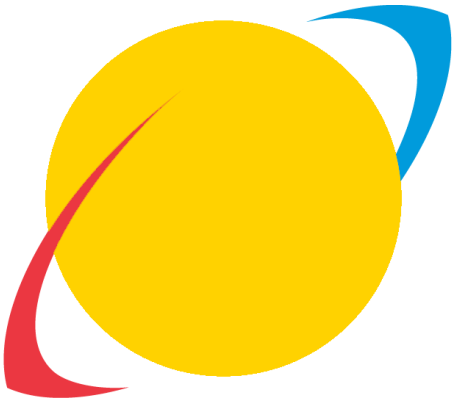
COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Lunar Outpost

Quincy Breaux
Evan Moore
Kadin Le
Alexander Viola

Revised June 21, 2026



CSCI 370 Summer 2026
Prof. Donna Bodeau

Table 1: Revision history

Revision	Date	Comments
New	5/20/26	
Rev – 2	5/21/26	Added initial introduction, functional requirements, non-functional requirements, risks, definition of done, team profile.
Rev – 3	5/29/2026	Added Risk Register and System Architecture
Rev – 4	6/2/2026	Added Software Test and Quality plan and Ethical Considerations
Rev – 5	6/14/2026	Updated Software testing results Added project completion status, future work, lessons learned, and acknowledgements
Rev - 6	6/21/2026	Updated Software Architecture, Appendix, Tests, References, grammar based on feedback from peer review

Table of Contents

I. Introduction

Simulation plays a big part in making sure rovers operate how companies expect them to on the lunar surface. Proper simulation reduces uncertainty and allows operational characteristics to be predicted. Our team's project is to use images of the lunar surface from previous space missions to get a system of metrics of the lunar surface so our client can upload a simulated 2D photo and apply these metrics to that photo. The metrics we are looking for included but are not limited to color, texture, and influence of lighting angles. This filter that we are aiming to build will help the simulation team of Lunar Outpost run a simulation of their new rover on the lunar surface with greater accuracy.

II. Functional Requirements

The functional requirements of this project revolve around the goal of creating a 2D image filter that improves the realism of a 2D simulated image of the moon's surface. These requirements include collecting a database of 50+ lunar surface images, writing tooling to generate metrics regarding what makes a 2D image realistic, and finally a program to apply a realism filter to simulated images.

Database functional requirements:

- Real photographs from the lunar surface
- 50+ photographs
- Color photographs
- High resolution photographs
- Does not include man-made equipment
- No lens flare
- No overexposure or underexposure

Metric generation requirements:

- Compares simulated images to real images
- Generates quantitative metrics

Final filter functional requirements:

- Inputs an image, outputs an image

III. Non-Functional Requirements

Throughout the duration of this project our employers Lunar Outpost have made it clear that they want proper documentation of the whole project. This means having detailed reports on how we completed each section and why we made the choices we did. They also specified that they want us to have a proper task organizer to keep us on track while working on the project. Also, the code needs to be easy to read and contain comments to explain its functionality. Finally, we need to have unit tests to make sure our metric generation is working properly.

IV. Risks

Our project comes with a set of risks that is important to keep in mind. Errors in metric generation could lead to misrepresentation of simulation fidelity, resulting in unfounded confidence in visual models. Also, the introduction of unrealistic artifacts from the lunar filter might result in autonomy algorithms being tested using bad data, potentially resulting in dangerous autonomy system behavior when deployed to the lunar surface. It is imperative that our contributions to the rover do not lead to a fatal error on a mission.

Table IV.1 gives a numerical description of the possible risks of this project. Our risks magnitudes range from 0-25 depending on likelihood and impacts. Because the accuracy of this filter is a top priority, our risk tolerance is low, ideally under 5. With our control methods, we believe most of our risks' likelihoods can be mitigated to an acceptable level.

Table IV.1: Risk Register

Risk Description	Risk Impact	Likelihood	Impact	Risk Magnitude	Treatment	Control	Mitigated Risk Magnitude
Metric Generation implementation errors	Would mean the Lunar outpost team would change their simulation based on incorrect metrics	Medium	Serious	9	Reduce Likelihood	Unit tests, peer review, comparison against expected outputs.	3
Filter introduces unrealistic artifacts	Would create an unrealistic environment for Lunar Outpost's simulation, preventing correct training for rovers.	Medium	Serious	9	Reduce likelihood	Give limited controls over filter settings to the filter generation algorithm.	2
Dataset contains images with man-made objects	Would alter the metrics of the filter to not accurately describe the moon surface and instead would have areas that could look man made	low	Serious	6	Reduce likelihood	Crop out the man made object or just do not use them at all	2
Filter generation implementation errors	Filters may crash, apply effects incorrectly, or fail to process images consistently.	Medium	Doomsday	15	Reduce likelihood	Code reviews, integration testing, regression testing.	3
Metric generation generates unnecessary metrics	The team spends significant time developing metrics that provide little value to Lunar Outpost.	Medium	Medium	6	Reduce likelihood	Client feedback at sprint reviews.	1

V. Definition of Done

Our project will be considered complete once several core objectives are achieved. First, we must create a baseline database of high quality lunar surface images collected from reliable mission sources. Next, we will create a metric generation system capable of analyzing and comparing important visual characteristics within lunar imagery, such as texture, lightning, shadows, and terrain features. Finally, the team will fabricate a computer-based filtering system that takes 2D lunar images and then applies a lunar simulation filter designed to simulate/replicate the realistic visual appearance and environmental conditions of the Moon.

VI. System Architecture

Figure VI.1 depicts the diagram of our overall system architecture for the moon filter. The first step is to compile a database of real lunar images and simulated lunar images. Then, these images will be fed through the system which first preprocesses the images to only contain the lunar surface, and these processed images are fed into a metric generation algorithm. This algorithm aims to generate certain defined metrics for the real dataset and the simulated dataset for the image comparison algorithm which can then compare the "realism" of a simulated lunar surface image and send that data to the Lunar Surface Filter Pipeline. This final step applies a filter to simulated images to improve its "realism" based on generated metrics.

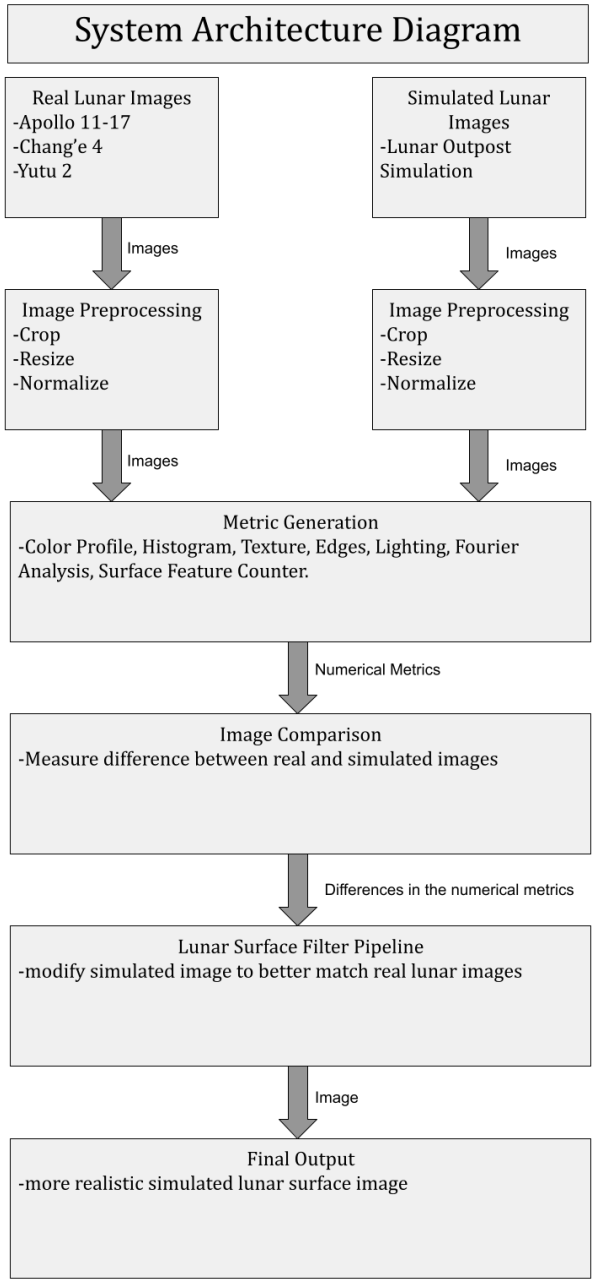


Figure VI.1: System Architecture

Metrics Generation

Figure VI.2 is an Architecture on how we generate our metrics for our filter. Our main program takes in real images from the lunar surface from our data base and simulated images from the lunar surface. Then the metric generation architecture happens inside the box. First the metric generator takes in a real image from our database and the simulated images. Then it runs the surface mask on the image to isolate the lunar surface of the image to run metric generation tools on. Then each of our 4 metrics take in the image and the mask and run their own program to get metrics on frequency, color, features and horizon line. Then the main program goes on to use these metrics to apply a filter over the simulator image.

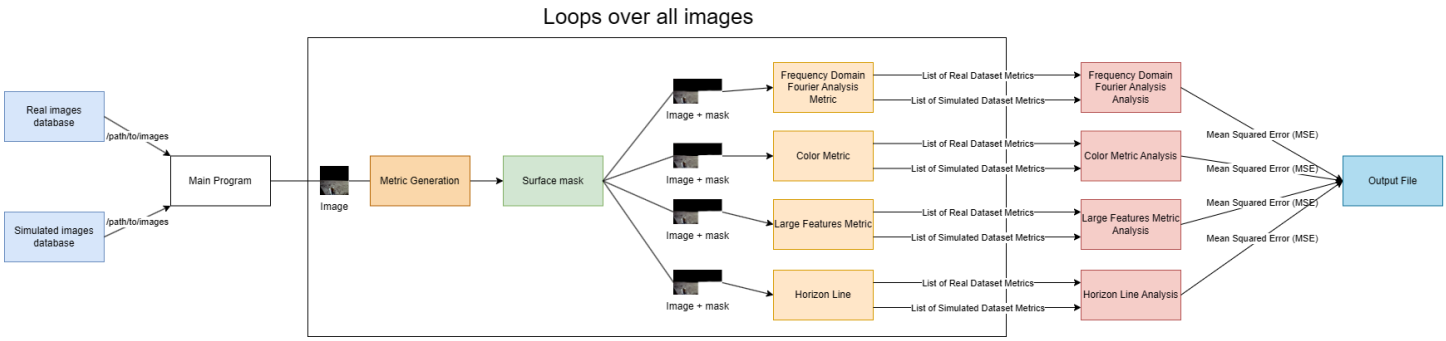


Figure VI.2: Metric Generation Architecture

Lunar Filter

Our final filter architecture involves a custom implementation of CycleGAN and a LoRA fine tuned Stable Diffusion model.

CycleGAN excels at learning patterns between two unpaired image datasets, and then applying the features from one dataset to the other.[1] In our case, CycleGAN processes a real dataset and simulated dataset, learns the features which makes a real photo look “real” compared to simulated photos, and apply these features to any simulated photo the user wants. The custom CycleGAN implementation targets consumer hardware with a minimum of 12 GB of RAM or 8GB of VRAM.

To do this, we developed a custom CycleGAN training algorithm optimized for lunar surface photos and consumer hardware. The training loop consists of converting a simulated image to a real image then back to a simulated image and converting a real image to a simulated image then back to a real image. Then, it evaluates the amount of loss in process to determine the effectiveness of the current model using structural loss functions, the metric generation algorithm, and an adversarial discriminator. The metrics generation algorithm compares real and simulated->real images to determine a numerical error between the two, representing the magnitude of loss. The adversarial discriminator trains alongside and attempts to determine which images are truly real and which are simulated images with a filter applied. Using this knowledge, it will tweak its weights to improve the realism of the filter and repeat the process for a user-specified number of loops, also known as epochs. After the specified epoch is reached, the model is saved to a file to be used to apply a filter to any image.

To enhance the details of the lunar surface, a custom Lunar LoRA fine-tuned Stable Diffusion XL model is applied to the lunar surface via ControlNet. The LoRA training process adapts Stable Diffusion to our database of lunar images to make the final product similar to Apollo and Chang’e missions.[2] The pretrained LoRA model is trained on the Chang’e images in the database. ControlNet uses Canny Edge Detection to create an outline of the surface’s position and important details, such as craters and rocks, to preserve important information to help Stable Diffusion preserve the original image’s main features.[3] This process yields a very high quality and realistic lunar surface in the final image at the expense of diffusion processing time.

After all CycleGAN training and LoRA training has been completed, the model files are used to apply a filter to any simulated image. Figure VI.3 depicts the final filter pipeline with both CycleGAN and Stable Diffusion.

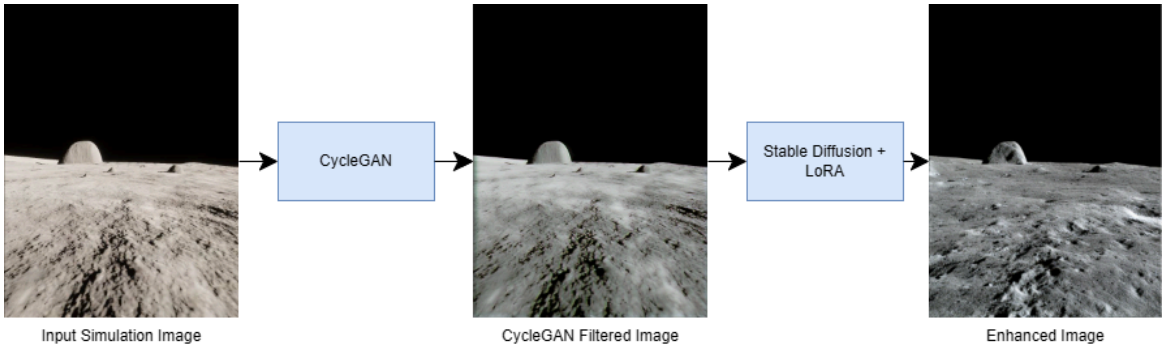


Figure VI.3: Filter Pipeline

VII. Software Test and Quality

As discussed in the introduction, the purpose of this project is to improve the accuracy of simulated lunar images through the addition of a 2D filter. The main elements of our quality control consists of Unit Testing, code reviews, and user acceptance testing.

Code Reviews

Code reviews are conducted either in person or virtually. This is where we go one at a time and walk through the code we wrote for our part. This gives us a chance to group debug problems we have been stuck on. It also helps us better understand the program as a whole and how different parts are implemented. It also helps with brainstorming, having all 4 group members' perspectives.

Unit Testing

For the metric generation deliverable, the algorithms used for metric generation must be reliable. To assure that the metric generation math is correct, Unit Tests are used for generating sample inputs and checking if the outputs are expected. The framework used for testing python code is pytest, and this will automatically discover the testing files and run all tests with one command.

User Acceptance Testing

For user acceptance testing, the metric generation algorithm can be used to quantitatively evaluate the similarity of the final image produced by the program. Using client feedback and numerical metrics, we can compare both sources of feedback to determine a numerical “error” threshold that the filter must meet to make the simulated images “realistic.”

Integration Testing

We have a file named metrics_main.py, and when we run this file, it runs all of our metric generators, and returns results for each of the metric generators. We have been using this as more of Integration testing and bulk testing the project in 1 go.

Performance Testing

Parts of the metrics algorithm require accuracy. The horizon line needs to detect the horizon correctly, and the surface mask needs to isolate the surface without including any sky pixels or cropping part of the surface. The performance can be tested by running these algorithms through the full database of images and evaluating each image on a pass/fail basis.

Quality Dependent Use Cases

Table VII.1 introduces 5 possible user stories that rely on the quality of this image filter software. Initial testing methodologies are discussed and will be elaborated on in the following sections.

Table VII.1: Software Quality User Stories

User Story	Test/Acceptance Criteria
As a Simulation Expert, I want a filter that can make my simulation look more lunar like, so I can do better tests for our rover launch	Given a simulated image, when the filter is applied, then the output image should be saved successfully and visually preserve terrain structure while improving lunar qualities such as contrast, shadows, and texture. Integration testing and user acceptance testing can be used to integrate into rovers.
As a moon rover, I want a realistic example of the environment I am going into, so I don't make bad decisions while crawling on the moon.	Given a simulated lunar image, when the filter and metric pipeline are applied, then the output should visually preserve important terrain features such as craters, shadows, rocks, and sloped terrain texture, while producing metrics that are closer to real lunar images than the original simulation image. Integration testing and user acceptance testing can be used to integrate into rovers.
As a mechanical engineer working on rover design, I want to be able to see and understand the terrain we are working with, so the rover can be designed to operate on that terrain.	The terrain must look realistic enough to give an idea of the terrain. User acceptance testing and unit testing can be used to evaluate realism of images.
As a Computer Vision expert, I want to be able to get accurate lunar image metrics from images from lunar missions, so I can compare them to other images.	Given an image path, when the program loads the image, then it should confirm the image exists, load it without error, and return its resolution, color channels, and pixel data type. Unit testing can ensure that the algorithms are correct.
As an Employee that works in a aerospace company, I want a database of pure lunar surface Images that I can use in future projects, so I can save time when starting these projects	Given an image lunar database, when someone opens the image database it should have all the necessary metadata such as: unique ID, filename, source link, resolution, format and notes so the dataset can be reused without needing to revalidate images.

Surface Mask Performance Test

This test involves running the mask over the full database of images and manually inspecting each mask on a pass/fail basis. The criteria for success are as follows:

- The full sky is covered by the mask
- No surface features is covered by the mask
- Any imperfection, regardless of size, is a failure

After running through the full database of 187 images, only 32 generated masks contain noticeable errors which amounts to a passable 82% success rate. These errors range from masking out large mountains to tiny rocks near the horizon. After discussion with the client, an 82% success rate is good enough, and masking out large shadowing areas can help this specific use case. Figure VII.1 illustrates the few cases where the surface mask contained noticeable issues.

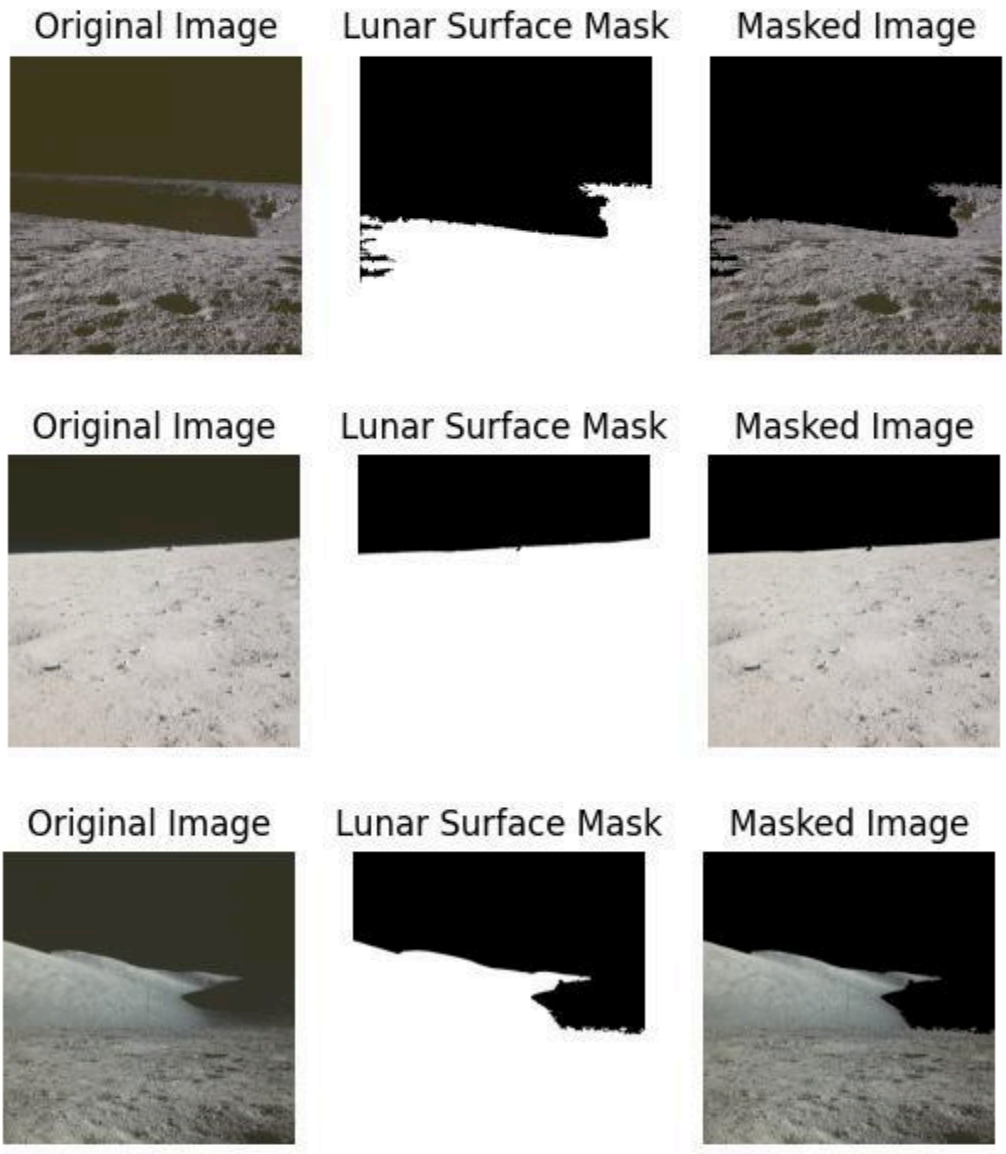
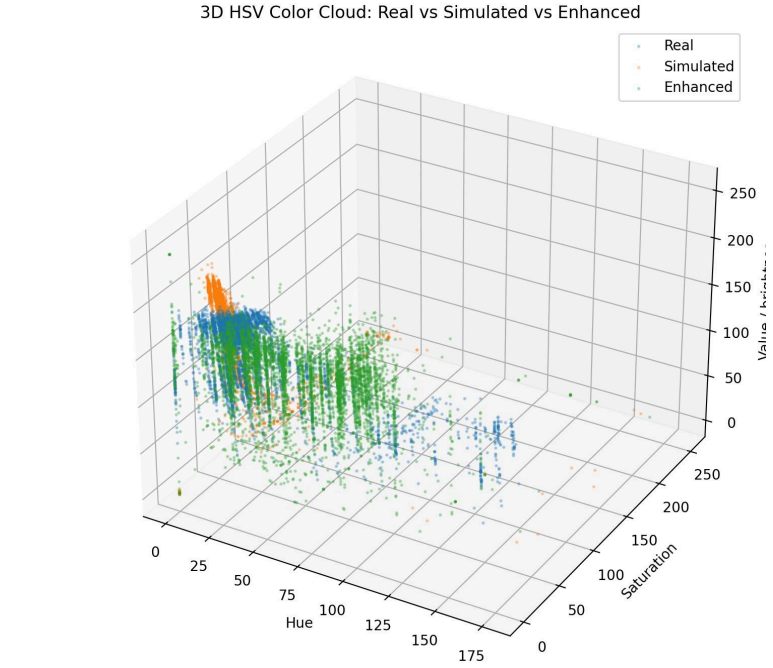


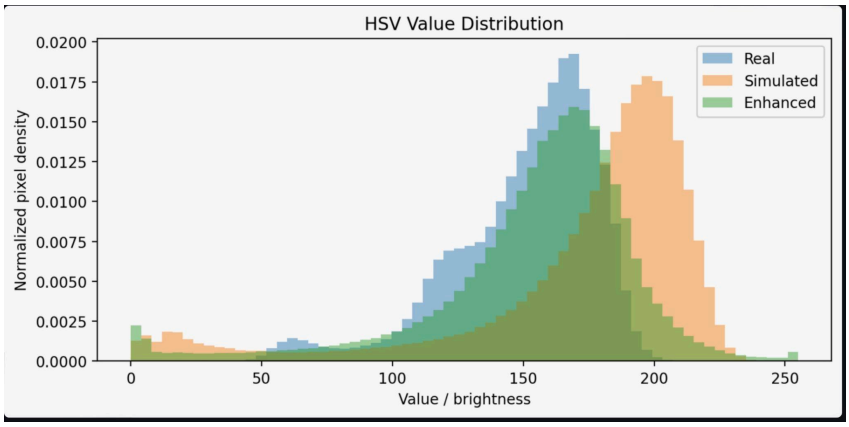
Figure VII.1: Surface Mask Failures

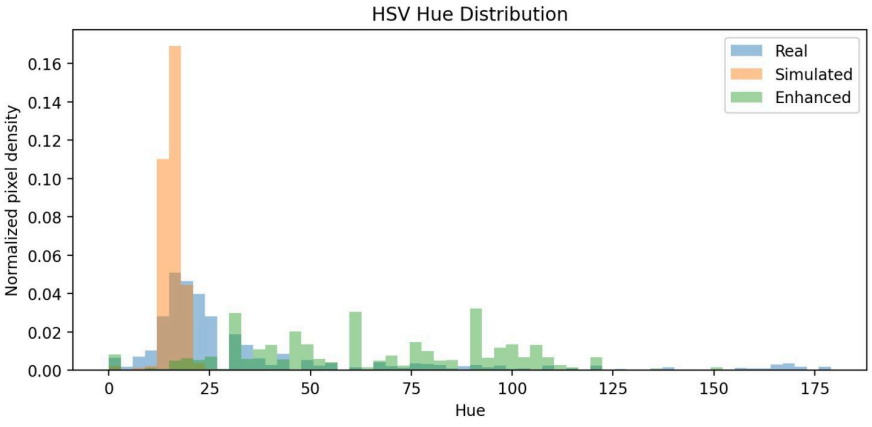
Lunar Filter Performance and User Acceptance

Using the custom metrics discussed in the Software Architecture section, the performance of the filter can be compared in before and after results.

For the color matrix metric, the simulated images show concentrated hue and saturation, indicating lack of color variety, compared to real images. After the filter is applied to the simulation images, the hue and saturation are more distributed, similar to real images. The filter also helps lower the brightness, the V in HSV, the better match real images. This indicates that the filter is effective at improving the realism of colors. Figure VII.2 depicts the distribution of colors in the HSV color model for real images, simulated images, and filtered images.


3D HSV Color Cloud: Real vs Simulated vs Enhanced
Real
Simulated
Enhanced
Hue
Saturation
Value / brightness


HSV Value Distribution
Real
Simulated
Enhanced
Normalized pixel density
Value / brightness


HSV Hue Distribution
Real
Simulated
Enhanced
Normalized pixel density
Hue

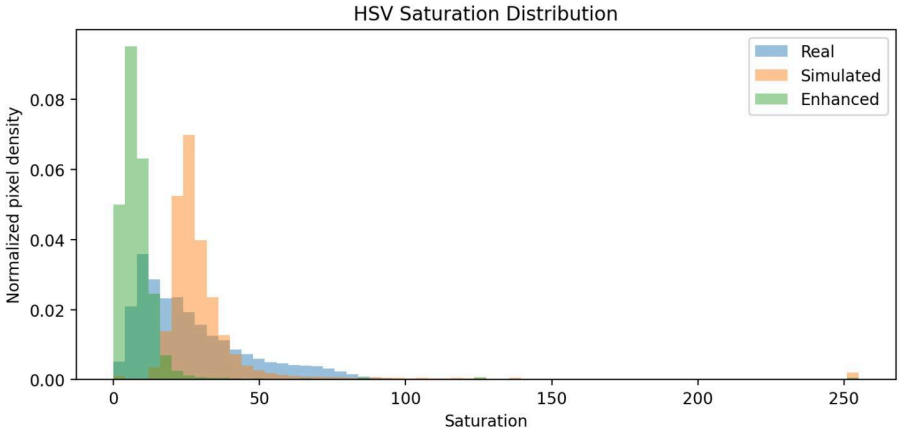


Figure VII.2: HSV Real vs. Simulated vs. Enhanced Comparisons

After the filter is applied to simulation images, a noticeable reduction in metric errors is observed. During CycleGAN training, the average differences between real and filtered simulated images drops below the baseline, indicating an improvement in overall realism across all 4 metrics discussed in System Architecture. Figure VII.3 demonstrates the overall improvement in filtered simulation images compared to the baseline simulation images as CycleGAN training proceeds.

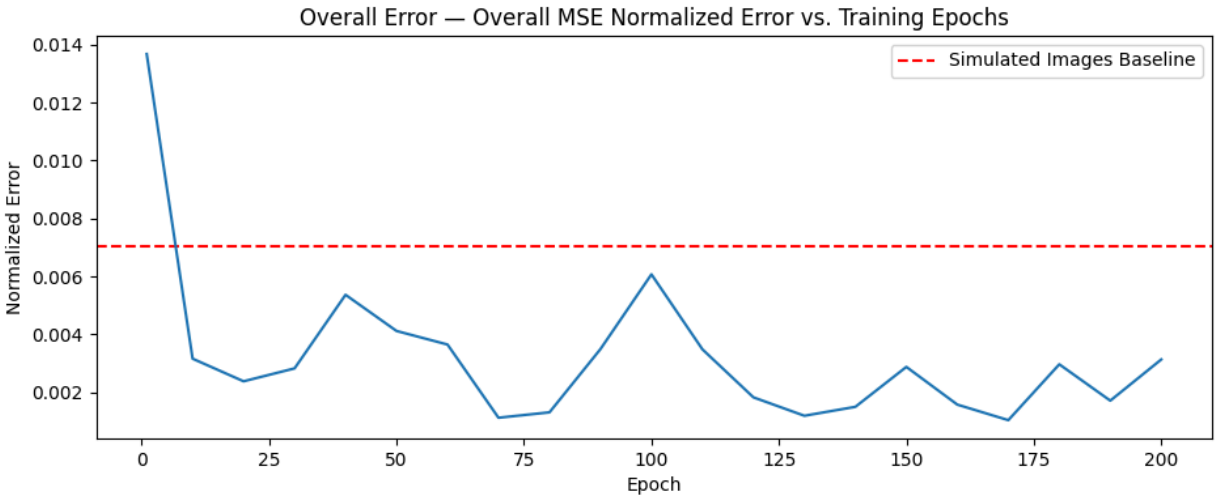


Figure VII.3: Metrics Mean Squared Error vs. CycleGAN Training Epochs

VIII. Project Ethical Considerations

Relevant ACM/IEEE Principles

The ACM/IEEE Principle that is most relevant to our project is **Principle 3: Product**, which states that software engineers should make sure their products meet high professional standards. Since our project creates metrics that compare simulated lunar images to real lunar images, it is important that our software is accurate, tested, and well documented. If our metrics are incorrect, the results could mislead Lunar Outpost when evaluating their simulations.

Another important principle is **Principle 1: Public**. Even though our software is mainly being developed for a client, the results may influence future engineering decisions. Because of this, we have a responsibility to ensure that our findings are accurate and honestly reported.

Principles Most at Risk

The principle most at risk of being violated is **Principle 3: Product**. Our project depends heavily on image datasets and metric calculation. If the metrics are implemented incorrectly or if our testing is not thorough enough, the software could produce inaccurate results.

This could cause Lunar Outpost to make decisions based on misleading information. In the worst case, it could lead to unrealistic simulations and wasted development time.

Michael Davis Ethical Tests

Harm Test

One option would be to release our metrics without fully validating them in order to save time. However, this could result in incorrect measurements and unreliable conclusions.

A better option is to spend extra time testing and validating the metrics before presenting results. While this requires more effort, it reduces the chance of harming the client through inaccurate information.

Publicity Test

The Publicity Test asks how a decision would look if it became public knowledge.

For example, if our team knew that parts of our dataset were flawed but chose not to disclose that information, it would reflect poorly on both our team and the project. Being transparent about limitations and possible sources of error would be a much more ethical choice.

Ethical Considerations if the Quality Plan Fails

If the software quality plan is incomplete or poorly executed, it could lead to several of the following ethical considerations,

- **Inaccurate or Misleading Outputs:** Poor validation could cause the filter to generate unrealistic textures, incorrect lighting, or biased color transformations. Users may unknowingly rely on flawed data.
- **Bias in Training Data**: If the real lunar images used for statistical modeling are not representative of all lunar features, the filter may embed systematic bias into all outputs.
- **Lack of Reproducibility** : Without proper version control, documentation, and testing, different versions of the filter may produce inconsistent results, undermining scientific reliability.
- **User Misunderstanding**: If the quality plan does not ensure clear documentation, users may misinterpret the filter as a scientific truth generator rather than a statistical style transfer tool.
- **Erosion of trust**: Poor quality control can lead to errors that damage trust in the software and Lunar Outpost.

Security Considerations

Even though the project does not involve personal data, several security concerns still apply:

- **Data Integrity**: Real lunar images and simulation datasets must be protected from corruption or unauthorized modification. Altered datasets could compromise the statistical models and produce misleading outputs.

- **Model Tampering**: If someone modifies the color matrices, frequency-domain signatures, or noise models, the filter could intentionally or unintentionally produce deceptive results.
- **Protection of Proprietary Simulation Data:** The simulated lunar images are part of a research pipeline so they must be protected from unauthorized access or misuse.

IX. Project Completion Status

We have completed Part 1, the image database, of the project by compiling a comprehensive database of real lunar surface images. Part 2, metric generation, is also finished, where we used these images to generate several analytical metrics, including color, frequency, contrast, and horizon metrics. Part 3, the lunar filter, required the most time, but it is now complete as well. Using the metrics from Part 2, we developed a CycleGAN model capable of comparing real and simulated lunar images and applying a realism enhancing filter to the simulated ones. These metrics were incorporated directly into the model's loss function, and as a result, we are now producing outputs that show promising levels of realism.

X. Future Work

While this project successfully developed a framework for evaluating the realism of simulated lunar imagery, there are several opportunities for future improvement. One area of future work would be expanding the metric suite to include additional terrain characteristics such as crater distributions, rock density, shadow behavior, and surface roughness. These metrics could provide a more complete understanding of lunar realism and improve the accuracy of comparisons between real and simulated images. Another potential direction is improving the filter pipeline using machine learning techniques that could automatically adjust simulated images based on metric feedback rather than only evaluating them.

A more ambitious long-term goal would be procedural terrain generation. By learning patterns from real lunar datasets, future systems could generate realistic lunar landscapes and unique terrain configurations for rover testing. This would allow autonomous systems to train on a much wider variety of environments than currently available. The framework could also be extended beyond lunar applications to evaluate or improve simulations of Earth, Mars, or other planetary surfaces. Overall, this project provides a strong foundation for future research in simulation realism, terrain generation, and computer vision-based environment analysis.

XI. Lessons Learned

Throughout the software development process, our team learned the effects of competition in the government space contracts, the importance of daily meetings with our client and software architecture diagrams, and the role of AI and Large Language Models (LLMs) in software development today.

One of our first roadblocks was the ability to source images from Lunar Outpost's proprietary simulation software. This led to a discussion of the nature of the space industry in the U.S. Because there are a handful of companies vying for the same contract and international competition for lunar missions, everything is locked behind strict rules, even 27 snapshots of an unfinished simulation rendering.

To help guide our progress, Lunar Outpost planned daily meetings with the team. These meetings proved to be highly valuable because it prevented work from veering off course. Any questions were quickly answered, any new ideas could be discussed before implementation, and the client was able to give feedback on day to day work. At the same time, we learned that communicating the structure of the program through code is nearly impossible. This is why Software Architecture Diagrams, depicted in Section VI, and UML diagrams are incredibly useful for large projects such as this. The diagrams helped communicate the idea to our client and will help maintainability in the future.

Another important lesson from our experience was the use of AI. Contrary to courses in college, LLMs are an integral part of development today. In this project, the use of LLMs accelerated the brainstorming process and assisted with development. It helped research the CycleGAN algorithm, understand each other's code, and provided an introduction to new technologies we used. With its help, a few stretch goals could be achieved by the end of the Field Session.

XII. Acknowledgments

We want to say thank you to Professor Donna Bodeau, our faculty advisor for helping us with the project. She made us think about problems and solutions in a way during our meetings and she was always willing to discuss the details with us. Professor Donna Bodeau also helped us stay on track.

We as a team also want to thank Shelby O'Callaghan and Lunar Outpost for letting us work with them. The advice and feedback they gave us during our daily Zoom meetings really helped us make progress on the project. Shelby was always happy to answer our questions, give us feedback and help us understand the real world challenges and issues related to the project. We also appreciate that they were able to get the surface simulation photos to us. We really appreciate the time and effort that Lunar Outpost put into supporting our team and helping us do a job.

We also want to thank the School of Mines and the Computer Science department for giving us this opportunity and providing us with the resources we needed throughout the project.

XIII. Team Profile

Quincy Breaux:

- Senior @ Colorado School of Mines
- Computer Science B.S. Focus in Computer Engineering
- Home Town: Denver, Colorado
- Client Liaison

Kadin Le:

- Junior @ Colorado School of Mines
- Computer Science
- Hometown: Parker, CO

- Robotics focus
- Advisor Liaison

Evan Moore:

- Senior

- Computer Science

- Two Minors in applied mathematics and statistics, and business

- Hometown: Houston, TX

- Documentation Lead

Alexander Viola:

- Senior @ Colorado School of Mines
- Computer Science Major

- Mech E minor
- Hometown: Vail, CO

References

[1] GeeksforGeeks, “Cycle Generative Adversarial Network (CycleGAN),” *GeeksforGeeks*, Jul. 20, 2020. https://www.geeksforgeeks.org/machine-learning/cycle-generative-adversarial-network-cyclegan-2/

[2] P. Cuenca and S. Paul, “Using LoRA for Efficient Stable Diffusion Fine-Tuning,” *huggingface.co*, Jan. 26, 2023. https://huggingface.co/blog/lora

[3] lllyasviel, “ControlNet,” *GitHub*, May 16, 2023. https://github.com/lllyasviel/ControlNet

Appendix A – Key Terms

Term	Definition
CycleGAN	*Cycle Generative Adversarial Network - An architecture for unpaired image to image translation.*
LoRA	*Low Rank Adaptation - An efficient fine tuning technique for Large Language Models and Image Diffusion models*