# A Near Real Time Space Based Computer Vision System for Accurate Terrain Mapping

Caleb Adams
University of Georgia Small Satellite Research Laboratory
106 Pineview Court, Athens, GA 30606
CalebAshmoreAdams@gmail.com

**Faculty Advisor:** Dr. David Cotten
Center for Geospatial Research, UGA Small Satellite Research Laboratory

## ABSTRACT

The Multiview Onboard Computational Imager (MOCI) is a 3U cube satellite designed to convert high resolution imagery, 4K images at 8m Ground Sample Distance (GSD), into useful end data products in near real time. The primary data products that MOCI seeks to provide are a 3D terrain models of the surface of Earth that can be directly compared to the Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) v3 global Digital Elevation Model (DEM). MOCI utilizes a Nvidia TX2 Graphic Processing Unit (GPU)/System on a Chip (SoC) to perform the complex calculations required for such a task. The reconstruction problem, which MOCI can solve, contains many complex computer vision subroutines that can be used in less complicated computer vision pipelines.

## INTRODUCTION

This paper does not seek to describe the entire satellite system; it seeks to describe, in detail, the complex computation system that MOCI will utilize to generate scientific data on orbit. An overview of the satellite system and optical system are provided for clarity and context. A detailed explanation of the subroutines in MOCI's primary computer vision pipeline are described in detail over the course of this paper.

### *System Overview*

The MOCI satellite primarily uses Commercial Off the Shelf (COTS) hardware so that the focus can be on payload development. The MAI-401 with a star-tracker is utilized to achieve the necessary pointing requirements. The GomSpace BP4 P60 Electrical Power System (EPS) is used. The F'Sati Ultra High Frequency (UHF) transceiver and F'Sati S-Band transmitter are used for communications of commands, telemetry, and science data. A Clyde Space On Board Computer (OBC) is used as the main flight computer. The payload uses the Nvidia TX2 SoC as the high-performance computation unit and a custom optical system developed by Ruda-Cardinal that produces 4K images from at 8m GSD from a 400km orbit.

### *Surface Reconstruction Pipeline*

In our case, a computer vision pipeline, sometimes referred to as a workflow[1], consists of a set of chained computer vision subroutines where the output of the previous is the input to the next. Subroutines are often referred to as stages in this sense[1]. MOCI implements a Surface Reconstruction Pipeline with the initial inputs being a set of images, the position of the spacecraft per image, and the orientation of the spacecraft per image. The first stage in the pipeline is the feature detection stage.

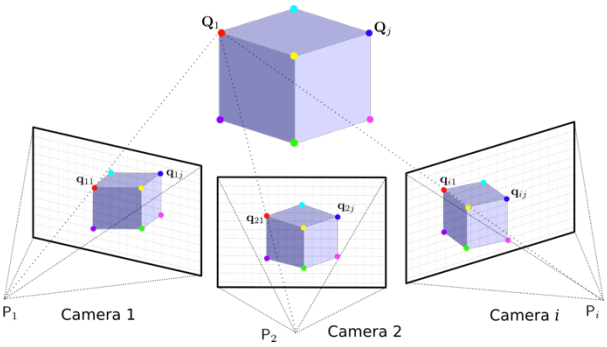


**Figure 1: Multiview Reconstruction[2]**

The feature detection stage identifies regions within each image that should be considered for feature description. This stage produces a set of points at location $(x, y)$, scale $s$, and rotation $\theta$. The feature description stage takes this set and encodes the information from local regions into a feature vector $f$. The next stage is the feature matching stage, which seeks to find the best correspondence, or minimum difference, between the set of points in the images. Once points have been matched in the image they need to be placed into $\mathbb{R}^3$ from $\mathbb{R}^2$. for reprojection. Vectors are made at the position of each matched point and used to calculate the point of

minimum distance between all projected lines, which is the calculated point of intersection. The output of the reprojection is a set of points in $\mathbb{R}^3$. After the initial reprojection a Bundle Adjustment is performed as the next stage in the pipeline. This takes the sets of points and uses camera data to estimate the reprojection error and remove it from the generated points. The result after the Bundle Adjustment is a more accurate point cloud. The next stage is the final stage, which is a surface reconstruction. First normal are calculated for the set of points to make an oriented point set. The oriented point set is then used for a Poisson Surface reconstruction to make the final data product. Any additional computer vision subroutines discussed here are not part of MOCI's primary pipeline.

## RELATED WORK

The techniques relayed here are not new, but are built from well understood algorithms and computer vision subroutines. The implementations of these well understood principals are built from previous work that the University of Georgia (UGA) Small Satellite Research Laboratory (SSRL) has done. The adaptation of structure from motion and real time mapping for aerial based photogrammetry and autonomous robotics is commonplace.

### *Multiview Reconstruction*

GPU accelerated mechanics Structure from Motion have been implemented on many occasions. Chang Chang Wu's research to develop an incremental approach to Structure from Motion demonstrated that it was possible to solve the reconstruction problem in $O(n)$ rather than $O(n^4)$, greatly improving efficiency and speed[3]. Additional research has shown that the triangulation problem can be achieve a 40x speed up[4] when utilizing Compute Unified Device Architecture (CUDA) capable Nvidia GPUs. Additionally, multicore GPU Bundle Adjustment has been shown to achieve a 30x speed up over previous implementations. GPU accelerated feature detectors and descriptors are now commonplace. This can typically lead to the identification and extraction of features within a few milliseconds[5], which allows the near real time extraction of input information into the pipeline. A standard Poisson Reconstruction, fundamentally limited by the octree data structure, can run two orders of magnitude[6] faster when implemented on a CUDA capable GPU.

### *Surface Normal Calculation*

A key problem in a surface reconstruction or structure from motion pipeline is the generation of an oriented point set. Recent research, testing the feasibility of generating oriented point sets from cube satellites, has claimed that normal estimation is only accurate between $5^o$ and $29^o$ when utilizing 2m GSD imagery[7]. Additionally, new techniques have recently been demonstrated showing that a Randomized Hough Transform can preserve sharp features, improving the accuracy of point normal while being almost an order of magnitude faster[8]. It is expected the more efficient point normalization methods will improve the accuracy of the 3D models generated by MOCI.

### *Cloud Height and Planetary Modeling*

With previous studies, we have shown that image data from the International Space Station (ISS) High Definition Earth-Viewing System (HDEV) can produce accurate cloud height models within 5.926 – 7.012 km[9]. Additionally, available structure from motion pipelines and surface reconstruction, in conjunction with the SSRL's custom simulation software, have been used to demonstrate that a 3D surfaces of mountain ranges can be reconstructed. The SSRL has demonstrated, that the proposed pipeline can generate 3D models of large geographic features within 68.2% accuracy of ASTER v3 global DEM data[10], resulting in an approximately 10m resolution surface model.

## PAYLOAD SYSTEM OVERVIEW

A simple overview of the system is provided in this section to make later sections about scientific computations more clear. The payload sits at the top of the electronics stack of the MOCI system and contains the Nvidia TX2, an optical assembly, an e-con systems See3CAM_CU135 with an AR1335 image sensor from ON Semiconductor, Core GPU Interface (CORGI) Board to connect all the subsystems together and allow them to communicate over a standard PC104+ bus.

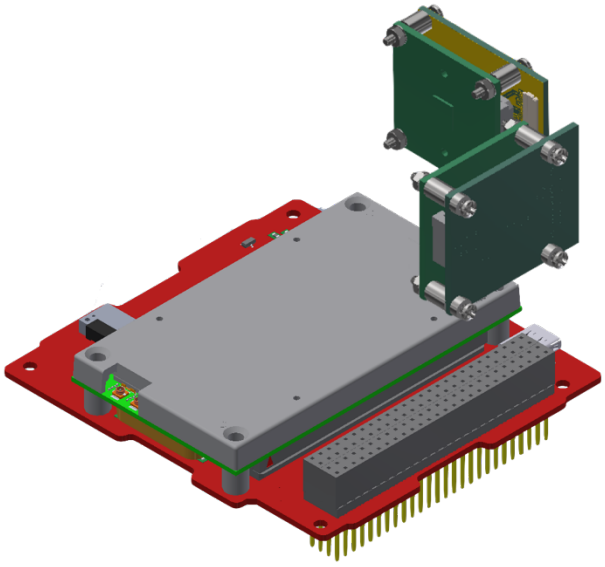

**Figure 2: Payload Electronics**

### *Data Interfacing and Power*

The Nvidia TX2 is interfaced to the CORGI Board via a 400 pinout connector. The CORGI Board connected to the See3CAM_CU135 interface board via a Universal Serial Bus type C connector (USB-C) allowing for 4K image data to be streamed to the GPU. The CORGI also routes an Inter-Integrated Circuit ($I^2C$) bus, Ethernet, and a Serial Peripheral Interface (SPI) into the satellite's PC104+ bus. The TX2's maximum power draw is 7W, but current computations are only running at approximately 4.5W.

### *Thermal Properties*

For a worst case, thermal analysis, an unrealistic power draw of 14W is used. Additionally, a system with 0% efficiency was also assumed as a worst-case scenario. A maximum temperature of approximately 51$^o$ C was simulated with these conditions. The TX2 is attached to a Thermal Transfer Plate and simulations have shown that the max operating temperature is sustainable for the system. Further, more detailed, research will soon be published on how we have managed these thermal conditions.

### *Optical System*

The SSRL is partnering with Ruda-Cardinal to make a custom optical assembly capable of generating images at a resolution of at most 8m GSD. The optical system has a 4.5$^o$ Field of View (FOV) and an effective focal length of 120mm.

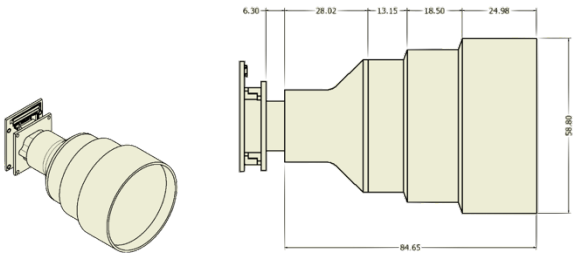


**Figure 3: MOCI Optical Assembly**

## GPU SYSTEM OVERVIEW

The GPU (Nvidia TX2) is a complete SoC, capable of running GNU/Linux on an ARM v8 CPU with a Tegra GPU running the Pascal Architecture. The TX2 is has 256 CUDA cores, 8 GB of 128 bit LPDDR4, and 32 GB of eMMC.

### *Radiation Mitigation*

The primary concerns in LEO are Single Event Upsets (SEU), Single Event Functional Interrupts (SEFI), and Single Event Latchups (SEL)[11]. These are certainly concerns for a dense SoC like the TX2. Thus, MOCI will utilize aluminized capton as a thin layer of protection for the payload. Software mitigation is also implemented. The Clyde Space OBC contains hardware-encoded ECC and could flash a new image onto the TX2 if necessary. The TX2 also utilizes a custom implementation of software-encoded error correction coding (ECC). Further, more detailed, research will soon be published on how we have managed and characterized these radiation conditions.

### *Compute Unified Device Architecture*

Currently the TX2 utilizes CUDA 9.0. CUDA capable GPUs can parallelize tasks, leading to computational speeds orders of magnitude higher than those of a CPU system performing the same operations. CUDA's computational parallel model is comprised of a grid that contains blocks made up of threads. The TX2 can handle up to 65535 blocks per dimension, leading to a potential total of 2.81 x 1014 blocks each containing a maximum of 1024 threads. The potential for parallelization here is substantial, and is the key to developing a near real time computer vision system.

### *GPU Accelerated Linear Algebra*

In Hartley's survey paper on optimal algorithms in Multiview Geometry, every algorithm he identifies benefits greatly from hyper optimized matrix operations that are made possible by the massive parallelization that CUDA enables[12]. Furthermore, widely available linear algebra libraries, such as the Basic Linear Algebra Subsystem (BLAS) have been accelerated with CUDA[13]. These modified libraries, such as cuBLAS and cuSOLVER are critical to the improving the functions needed in complex computer vision pipelines.

## FEATURE DETECTION AND DESCRIPTION

After images have been acquired from the payload system, the first step in the pipeline is feature detection. Typically, feature detection attempts to identifies regions within an image that should be considered for feature description[14]. Feature descriptions are only given to candidate regions/points that meet the requirements of the algorithm. For our purposes, we utilize the Scale Invariant Feature Transform (SIFT) developed by Lowe. The SIFT algorithm, which contains several standard subroutines, has become a standard in computer vision

### *Detection of Scale Space Extrema*

To detect feature that are scale-invariant, the SIFT algorithm uses a Difference of Gaussians (DoG) to identify local extrema in scale-space. The Laplacian of Gaussians (LoG) is often used to detect stable features in scale-space[14]. The convolution of an image with a

Gaussian kernel is defined by a function $L(x, y, \sigma)$, which is produced by the convolution of a scale space Gaussian[15], $G(x, y, \sigma)$, and input image, $I(x, y)$ and $*$ is a convolution operation between functions:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (1)$$

An efficient way to calculate the DoG function, $D(x, y, \sigma)$ is to simply compute the difference two nearby scales with a separation of $k$.

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) \quad (2)$$
$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \quad (3)$$

The Gaussian kernel is convolved with the input image to form a Gaussian Scale Pyramid.

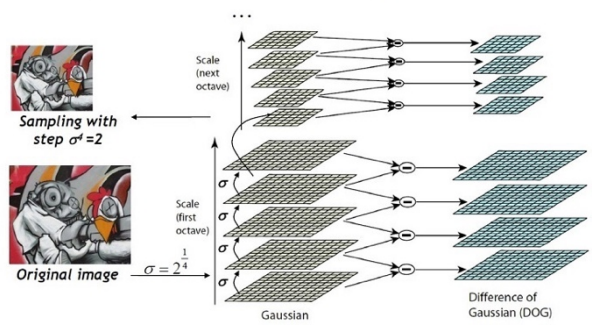


**Figure 4: The DoG in Scale-Space[15]**

To detect the local minima and maxima of the DoG function, the candidate point is compared to the 8 local neighbors of its current scale, the 9 neighbors above its scale, and the 9 neighbors below its scale in the DoG pyramid. For ease of computation, the scale separation of $k$ is chosen to be represented as $k = 2^{1/s}$ . where $s$ is chosen to be an integer number such that a doubling of $s$ results in a division of the scale space $\sigma$ in the next octave[15].

***Keypoint Localization and Filtering***

Given a set of candidate points, given by the detection of scale-space extrema from the DoG, the challenge is to localize the point by determining the ratio of principal curvature. A method proposed by Brown uses a Taylor expansion of the scale-space function, $D(x, y, \sigma)$, where the origin is at the center of the sample point[16]:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \quad (4)$$

The derivatives are located at the center of the sample point and the offset from the sample point is defined as $x = (x, y, \sigma)^T$. The local extreme $\hat{x}$ is given by taking the derivative of the function with respect to $x$ and setting it equal to zero:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \quad (5)$$

Often additional calculations are preformed to eliminate unstable extrema and edge responses. To eliminate strong edge responses, which a DoG will often produce, the principal curvature is computed from a Hessian matrix containing the partial derivatives of the DoG function, $D(x, y, \sigma)$:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (6)$$

Harris and Stephens have shown that we only need be concerned with the ratios of eigenvalues[17]. We let $\alpha$ represent be the eigenvalue with the largest magnitude and $\beta$ be the smaller magnitude. Then we compute the trace of $H$ and the determinate.

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta \quad (7)$$

$$Det(H) = D_{xx} D_{yy} - (D_{xx}{}^2) = \alpha\beta \quad (8)$$

We then let $r$ be the ratio of the between the largest and smallest eigenvalues such that $\alpha = r\beta$. Then we discover:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \quad (9)$$

We can then use this ratio as a cut off for undesired edge points. Typically, a value of $r = 10$ is used[15] to eliminate principal curvatures greater than $r$.

***Orientation and Magnitude Assignment***

To achieve rotation invariance, so that we can identify the same keypoints from any rotation, we must assign an orientation to the keypoints from the previous step. We want these computation to occur in a scale invariant manner as well, so we select the Gaussian smoothed image $L(x, y)$ at scale $\sigma$ where the extrema was detected. We can use pixel differences to compute the gradient magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, to assign:

$$m(x, y) = \sqrt{\begin{matrix}(L(x+1, y) - L(x-1, y))^2 \\ +(L(x, y+1) - L(x, y-1))^2\end{matrix}} \quad (10)$$

$$\theta(x, y) = tan^{-1} \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \quad (11)$$

### *Feature Description*

For each keypoint, the SIFT algorithm will start by calculating the image gradient magnitudes and in a 16 × 16 region around each keypoint using its scale to select the level of Gaussian blur for the image. A set of oriented histograms is created for each 4 × 4 region of the image gradient window[15].

A Gaussian weighting function with $\sigma$ equal to half the region size assigns weights to each sample point. Given that there are 4 × 4 and 8 possible orientations, the length of the generated feature vector is 128. In other words, there are 128 elements describing each point in the final output of the SIFT algorithm.

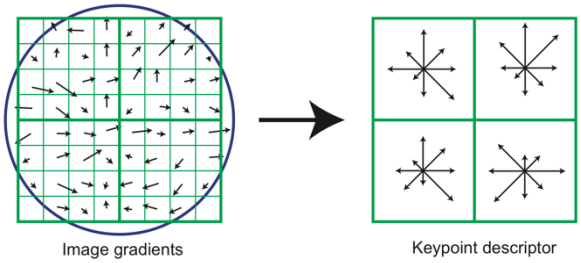


**Figure 5: A SIFT Feature Descriptor[15]**

## FEATURE MATCHING

Feature matching can be thought of as a simple problem of Euclidean distance. First, sets of points are eliminated that do not fit within a radius $r$, a set of close points is generated with the simple Euclidean distance $d$, where each feature has a coordinate $(x, y)$ on the image.

$$d = \sqrt{(y_2 - y_1) + (x_2 - x_1)} \tag{12}$$

We iterate through each point in image one, $I_1$, and image two, $I_2$, and accumulate potential matches where $d < r$.

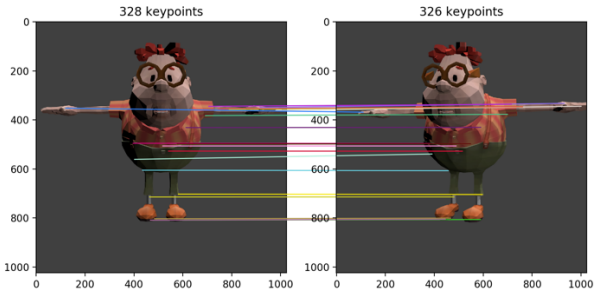


**Figure 6: SIFT feature matching**

For each value in the feature vector, $f$, we find the minimum 128 dimensional Euclidean distance, $m$.

$$m = \sqrt{\sum_{0}^{127} (f_{I1} - f_{I1})} \tag{13}$$

The resulting "matched" points should also be checked against some maximum threshold. If the minimum Euclidean distance is more than that threshold, the match should be discarded.

## MULTIVIEW RECONSTRUCTION

Once the features have been identified for each image and the features between images have been matched, the image planes must be placed into $\mathbb{R}^3$. The matched keypoints and camera information must be used to triangulate the location of the identified feature in $\mathbb{R}^3$.

### *Moving into 3D space*

The first step to moving a key point into $\mathbb{R}^3$. is to place it onto a plane in $\mathbb{R}^2$. The coordinates $(x', y')$ in $\mathbb{R}^2$ require the size of a pixel $dpix$, the location of the keypoint $(x, y)$, and the resolution of the image $(xres, yres)$ to yield:

$$x' = dpix \left(x - \frac{xres}{2}\right) \tag{14}$$

$$y' = dpix \left(\frac{yres}{2} - y\right) \tag{15}$$

This is repeated for the other matching keypoint. The coordinate $(x', y', z')$ in $\mathbb{R}^3$ of the keypoint $(x', y')$ in $\mathbb{R}^2$ is given by three rotation matrices and one translation matrix. First we treat $(x', y')$ in $\mathbb{R}^2$ as a homogenous vector in $\mathbb{R}^3$ to yield $(x', y', 1)$. Given a unit vector representing the camera, in our case the spacecraft's camera's, orientation $(r_x, r_y, r_z)$ we find the angle to rotate in each axis $(\theta_x, \theta_y, \theta_z)$. In a simple case, we find the angle in the $xy$ plane with:

$$\theta_z = \cos^{-1} \frac{[1\ 0\ 0] \cdot [r_x\ r_y\ r_z]}{\sqrt{[r_x\ r_y\ r_z] \cdot [r_x\ r_y\ r_z]}} \tag{16}$$

Rotations for all planes are generated in an identical way. Now, given a rotation in each plane $(\theta_x, \theta_y, \theta_z)$ we calculate the homogeneous coordinate $(r_x, r_y, r_z, 1)$ in $\mathbb{R}^3$ using linear transformations. The values $(T_x, T_y, T_z)$ represent a translation in $\mathbb{R}^4$ and use camera position coordinates $(C_x, C_y, C_z)$, the camera unit vectors representing orientation $(u_x, u_y, u_z)$, and focal length $f$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_x & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \quad (17)$$

$$\begin{bmatrix} C_x - (x_r + f * u_x) \\ C_y - (y_r + f * u_y) \\ C_z - (z_r + f * u_z) \\ 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \\ 1 \end{bmatrix} \quad (18)$$

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \quad (19)$$

***Point and Vector format***

The resulting transformation in equations 17 and 19 should be performed for all matched points. This should result in $n$ homogeneous points of the form $(x_n, y_n, z_n, 1)$. Each point has a corresponding camera, which is already known by the camera coordinate, $(C_{xn}, C_{yn}, C_{zn}, 1)$. From this find a vector $V_n$ from the camera position:

$$V_n = [C_{xn} - x_n \quad C_{yn} - y_n \quad C_{zn} - z_n] \quad (20)$$

$V_n$ should then be normalized so that it is a unit vector.

***N-view Reprojection***

Now the point cloud can finally be generated. To do such a thing, the goal of the n-view reprojection is to find the point, $B$, that best fits a set of lines. Traa shows we can start with the distance function, $D$, between our ideal point, $B$, and a parameterized line with vector, $V$, and point, $P$. We can think of distance function as a projector onto the orthocomplement of $V$, giving[18]:

$$D(B; P, V) = I - VV^T \quad (21)$$

Equation 21 should be thought of as a projecting vectors $B$ and $P$onto the space orthogonal to $V$. The challenge is solving this least squares problem given only matching sets of points $P_n$ and their vectors $V_n$. Let the set of matched points/vectors be represented by the set $L = \{(P_0, V_0), \dots, (P_n, V_n)\}$. We can view this set $L$ as a set of parameterized lines. We should minimum the sum of squared differences with the equation:

$$D(B; P, V) = \sum_{j=0}^{L} D(B; P_j, V_j) \quad (22)$$

To produce the point $\hat{b}$, the equation to minimize is:

$$\hat{b} = \min_B(D(B; P_n, V_n)) \quad (23)$$

Taking both derivatives with respect $B$ to we receive:

$$\frac{\partial D}{\partial B} = \sum_{j=0}^{L} -2\left(I - V_j V_j^T\right)(P_j - B) = 0 \quad (24)$$

We then obtain a linear of the form $Rp = q$, where:

$$R = \sum_{j=0}^{L}\left(I - V_j V_j^T\right), \quad q = \sum_{j=0}^{L}\left(I - V_j V_j^T\right) P_j \quad (25)$$

Traa shows that we can either solve the system directly or apply the Moore-Penrose pseudoinverse:

$$\hat{b} = R^{\dagger} q \quad (26)$$

The resulting $\hat{b}$ is the point of best fit for the members of set $L$. Once $\hat{b}$ is calculated, the next set of point/vector matches is loaded. The computation is repeated until all points best fit points $\hat{b}$ have been calculated. At the end of this stage in the pipeline, the point cloud has been generated.

***Bundle Adjustment***

All the calculations to this point have been in preparation for a reprojection, which is a simple triangulation. The Bundle Adjustment is used to calibrate the position of features in $\mathbb{R}^3$ based on a camera calibration matrix and minimize the projection error[19]. The camera calibration matrix, $K$, is stored by the spacecraft at the time of image acquisition. Given the location of an observes feature is $(x, y)$ and the real location of the feature $(\hat{x}, \hat{y})$ then the reprojection error, $r$, for that feature is given by:

$$r = (x - \hat{x}, y - \hat{y}) \quad (27)$$

The camera parameters include the focal lengths $(f_x, f_y)$, the center pixel $(C_x, C_y)$, and coefficients $(k_1, k_2)$ that represent the first and second order radial distortion of the lens system. The vector $S$ contains those 6 camera parameters, the feature's position in $\mathbb{R}^3$given in equation 19 as $(x', y', z')$, and the camera's position and orientation represented by $(C_x, C_y, C_z)$ and $(u_x, u_y, u_z)$, as previously equation 18. The goal is to minimize the function of vector $S$:

$$\min f(S) = \frac{1}{2} r(S)^T r(S) \quad (28)$$

MOCI's system will supply a pre-estimation of camera data from sensors onboard, allowing for a bound

constrained bundle adjustment that will greatly improve the speed of the computation. Levenberg-Marquardt (LM) algorithm is utilized to by iteratively solving a sequence of linear least squares and minimize he problem. A constrained multicore implementation of the bundle adjustment would only be a slight modification of the parallel algorithm proposed and described by Wu[19]. This is the last step of the point cloud generation process, when the reprojection error is minimized the point cloud computation is considered done. Additional research is necessary to determine if MOCI needs to perform a bundle rather than a real time custom calibration step before a standard n-view reprojection.

## POINT CLOUD NORMALIZATION

Once the point set has been generated it must be oriented so that a more accurate surface reconstruction can take place.

### *Finding the Normals of a Point Set*

The coordinates of the points in the point cloud and the camera position $(\boldsymbol{C_x, C_y, C_z})$ which generated each point. The problem of determining the normal to a point on the surface is approximated by estimating the tangent plane of the point and then taking the normal vector to the plane. However, the correct orientation of the normal vector cannot directly be inferred mathematically, so an additional subroutine is needed to orient each normal vector. Let the points in the point cloud be members of the set $P = \{p_0, p_1, \dots, p_n\}$ where $p_n = (x_n, y_n, z_n)$. The normal vector of $p_n$ is $n_n = (x_n, y_n, z_n)$, which we want to compute for all $p_n \in P$. Lastly, the camera position corresponding to $p_n$is denoted, in vector form, $C_n = \boldsymbol{(C_{nx}, C_{ny}, C_{nz})}$.

An octree data structure is used to search for the nearest neighbors of point $\boldsymbol{p_n}$. The $k$ nearest neighbors are defined by the set $Q = \{q_0, q_1, \dots, q_k\}$. The centroid of $Q$ is calculated by:

$$\boldsymbol{m} = \frac{\boldsymbol{1}}{\boldsymbol{k}} \sum_{\boldsymbol{q} \in \boldsymbol{Q}} \boldsymbol{q} \tag{29}$$

Let A be a k × 3 matrix as follows:

$$A = \begin{bmatrix} q_0 \\ q_1 \\ \vdots \\ q_k \end{bmatrix} \tag{30}$$

Now we factor matrix $A$ using singular value decomposition (SVD) into $A = UV^T$. Where $U$ is a (k × k) orthogonal matrix, $V^T$ is a (3 × 3) orthogonal matrix, and is a (k × 3) diagonal matrix, where the elements on the diagonal, called the "singular values" of $A$, appear in descending order. Note that the covariance matrix, $ATA$, can be easily diagonalized using our singular value decomposition:

$$\boldsymbol{A^T A} = (\boldsymbol{V \Sigma^T U^T})(\boldsymbol{U \Sigma V^T}) = \boldsymbol{V(\Sigma^T \Sigma) V^T} \tag{31}$$

The eigenvectors for the covariance matrix are the columns of vector $V$. The eigenvalues of the covariance matrix are the elements on the diagonal of $\Sigma^T\Sigma$, and they are exactly the squares of the singular values of matrix $A$. In this formula, both $V$ and $\Sigma^T\Sigma$ are (3 × 3) matrices, just like the covariance matrix $A^TA$. For randomly ordered diagonal elements $(\sigma_i)^2 \in \Sigma^T\Sigma$ we keep only the maximum $r$ many of them, along with their corresponding eigenvectors in matrix $V$. To produce the best approximation of a plane in $\mathbb{R}^3$ we would take the two eigenvectors, $(v_1, v_2)$, of the covariance matrix with the highest corresponding eigenvalues. Thus, the normal vector $n_n$ is simply the cross product of these eigenvectors, $n_n = v_1 \times v_2$.

### *Orienting a Point Set*

Orientation of all the normals begins once we have computed the normal for every point $\boldsymbol{p_n} \in \boldsymbol{P}$. We also want the normals of neighboring points to be consistently oriented. For the simple case where only a single viewpoint $C'$ is used to generate a point cloud, we can simply orient our normal vector such that the following equation holds:

$$(C' - p_n) \cdot n_n < 0 \tag{32}$$

If the equation does not hold for a computed normal vector, we simply "flip" the normal vector by taking $-n_n = (\boldsymbol{-x_n, -y_n, -z_n})$. If the dot product between the two vectors is exactly 0, then additional methods need to be applied. We need to account for the fact that multiple camera positions may apply to each point in the point cloud. Let $C = \{C_{i0}, C_{i1}, \dots, C_{in}\}$ be the set of all $N$ many camera position vectors for point $\boldsymbol{p_n}$.

We define $n_n$ to be an "ambiguous normal" if:

1. There exists a $C_{in} \in C$ such that $(C_{in} - p_n) \cdot n_n > 0$ AND
2. There exists a $C_{in} \in C$ such that $(C_{in} - p_n) \cdot n_n < 0$

Normals are assigned to all points that do not generate ambiguous normals. The way to make sure all normals are consistently correctly is to orient all the non-ambiguous normals, adding them to a list of finished normals, while placing all ambiguous normals into a queue. We then use the queue of ambiguous normals and try to determine the orientation by looking at the neighboring points of pi. If the neighboring points of pi

have already finished normals, we orient $n_n$ consistently with the neighboring normals $m_n$ by setting $n_n m_n > 0$. If the neighboring points do not have already finished normals, move $n_n$ to the back of the queue, and continue until all normals are finalized.

The point normal generation process needs significant improvements and can be aided greatly by the implementation of an octree data structure in CUDA[6].

## SURFACE RECONSTRUCTION

MOCI implements a Poisson Surface Reconstruction algorithm that is parallelized with CUDA. The input to this algorithm is an oriented set of points and the output is a 3D modeled surface. This surface is the final end-product of MOCI's computer vision pipeline and is stored in the Stanford PLY format.

### *Poisson Surface Reconstruction*

Thanks to Kazhdan, Bolitho, and Hoppe, an implementation of Poisson surface reconstruction on any GPU has become feasible[20]. A Poisson surface reconstruction takes in a set of oriented points, $\vec{V}$, and generates a surface.

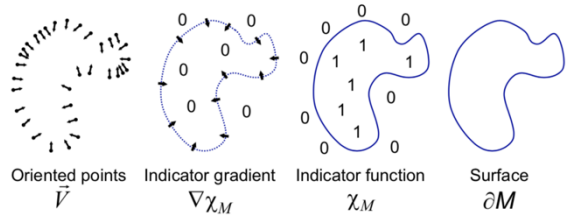


**Figure 7: Stages of Poisson Reconstruction in 2D**

Poisson computes an indicator gradient for the oriented points set by first finding the function $\chi$ that best approximates the vector field $\vec{V}$. When the divergence operator is applied the it becomes a Poisson problem where the goal is to compute the scalar function $\chi$ whose Lapacian equals the divergence of vector field vector field $\vec{V}$:

$$\Delta\chi \equiv \nabla \cdot \nabla\chi = \nabla\vec{V} \qquad (33)$$

## CONCLUSION AND INITIAL RESULTS

### *Comparison to ASTER data*

When compared with ASTER data MOCI is already meeting minimum mission success: MOCI can generate Digital Elevation Models within one sigma of accuracy relative to ASTER models.

Accuracy is calculated by a percent pixel difference. A simple program is used to project the 3D surface onto a plane, essentially rasterization. This plots a histogram of how likely it is that a given elevation is off. The percent difference is calculated from the minimum and maximum elevations. In other words, the percent pixel difference is the inverse of the magnitude of the elevation error.

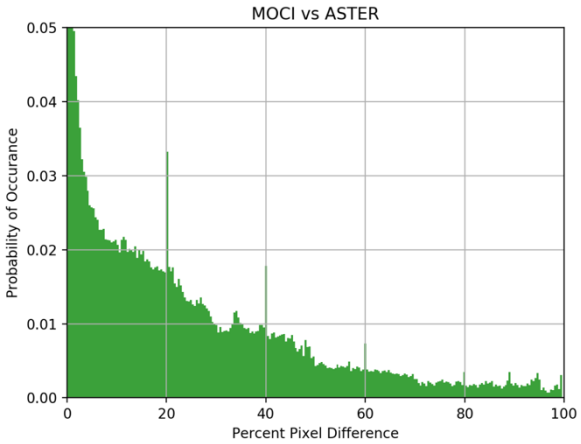


**Figure 8: A comparison of mount a simulated Mount Everest from MOCI with ASTER data**

MOCI's accuracy is likely to increase as better methods are implemented in the computer vision pipeline.

### *2-view Reconstruction*

A simple 2-view projection has been fully implemented and is often used to test the point cloud reconstruction portion of the pipeline. When supplied with near perfect keypoint pairs, the algorithm can reconstruct a point cloud with 86% accuracy. Accuracy is calculated by a percent pixel difference, which plots a histogram of how likely a given elevation is to be a given distance off.

A CPU implementation of 2-view reconstruction runs on a 50,000 point set in approximately 25 minutes. It is expected that this stage, when optimized for the GPU, will take less than 90 seconds[19].

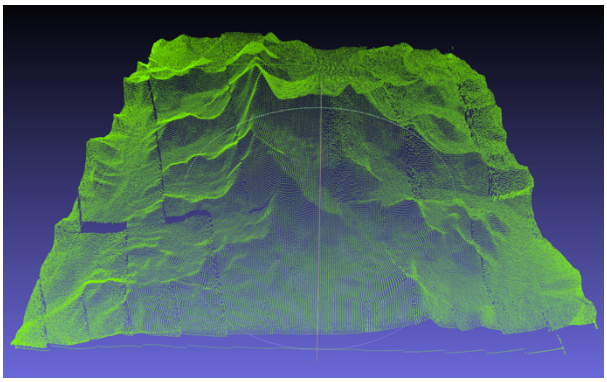

**Figure 9: A point cloud of Mount Everest generated from a simulation of a 2-view reconstruction**

### *Simulation of Data Acquisition*

The simple blender workflow that was demonstrated in the initial feasibility study has been expanded, improved, and is now included in a custom simulation package[10]. This simulation package has the capability to simulate a satellite with variable imaging payloads. This was so the SSRL could discover the optical lens systems, GSD, focal length, and sensor for the mission requirements. The simulation software allows the user to edit these as parameters - GSD is calculated. The simulation also allows for variable orbits, variation of ground targets, custom target objects, and more. A list of generated and input variables, seen to the left as a json file, shows some of the current capabilities of the simulation. The SSRL is currently working on porting these simulations to the supercomputing cluster available at UGA. Once the user has created a json file with the variables and environment that they would like to simulate, they can run the image acquisition simulation in a terminal[10].

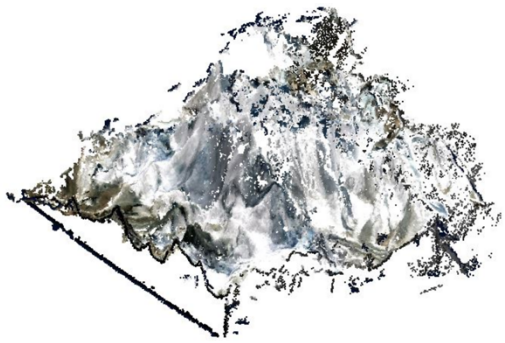

**Figure 10: A point cloud of Mount Everest generated from a simulated orbit over the region.**

Simulated data acquisition can be piped into reconstruction any algorithm. The position and orientation of the camera, as well as the image set, are all part of the standard output.

## FUTURE WORK

### *Testing and Simulation*

While initial results are promising and terrestrial technologies have shown that MOCI's computer vision pipeline is successful, more tests are needed to understand the limitations and capabilities of MOCI's computer vision system.

### *N-view Reconstruction vs. Bundle Adjustment*

It is unclear if a full Bundle Adjustment is necessary given MOCI's knowledge of its camera parameters. It may be possible to calibrate the first and second order radial distortion of the lens system once for all images and feature matches instead of calculating it every time a reprojection occurs. It may also be the case that, despite somewhat accurate knowledge of camera parameters, MOCI's multi-view reconstruction stage will benefit from improved accuracy with a bound constrained Bundle Adjustment.

## IMPLICATIONS

The complex computation system on the MOCI cube satellite may show that it is worth performing more complex computations in space rather than on the ground. In MOCI's case, it is beneficial because the 40 or more 4K images that it takes to generate a 3D model contain much more data than the final 3D model. This could also be the cause for real time data analysis, with a GPU accelerated system it may be possible to analyze data onboard a spacecraft to determine which data is the most useful and prioritize the downlink of that data. This has clear applications for autonomous space system or deep space missions. During a deep space mission, it would be possible to implement an AI to decide what data is worth sending back to Earth. In general, it's likely that Neural Networks will be easily implemented for space based applications on the TX2, or a system like MOCI's.

### *Acknowledgments*

A significant thank you to Aaron Martinez, who helped me understand most of the mathematics of MOCI's computer vision pipeline. The same thanks goes to Nicholas (Hollis) Neel, who helped me understand the concepts relating to the SIFT algorithm. Thanks to Jackson Parker for being the person who usually has to experiment with writing these algorithms in CUDA. Last, but not least, I would like to thank Dr. David Cotton, who has helped guide the SSRL to where it is today.

***References***


1. Rossi, Adam J., "Abstracted Workflow Framework with a Structure from Motion Application" (2014). Thesis. Rochester Institute of Technology. Accessed from http://scholarworks.rit.edu/theses/7814
2. Michot J., Bartoli A., Gaspard F., "Algebraic Line Search for Bundle Adjustment"- British Machine Vision Conference (BMVC)
3. Wu C, "Towards linear-time incremental structure from motion.", 3D Vision-3DV 2013, 2013 International conference on …, 2013
4. J Mak, M Hess-Flores, S Recker, JD Owens, KI Joy, "GPU-accelerated and effiecient multi-view triangulation for scene reconstruction" Applications of Computer Vision (WACV), 2014 IEEE, 2014
5. Aniruddha Acharya K and R. Venkatesh Babu, "Speeding up SIFT using GPU," 2013 Fourth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG), Jodhpur, 2013, pp. 1-4.
6. K. Zhou, M. Gong, X. Huang and B. Guo, "Data-Parallel Octrees for Surface Reconstruction," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 5, pp. 669-681, May 2011.
7. J. Stoddard, D. Messinger and J. Kerekes, "Effects of cubesat design parameters on image quality and feature extraction for 3D reconstruction," *2014 IEEE Geoscience and Remote Sensing Symposium*, Quebec City, QC, 2014, pp. 1995-1998.
8. Alexandre Boulch, Renaud Marlet. *Fast Normal Estimation for Point Clouds with Sharp*. *Features* using a *Robust* Randomized Hough Transform. Computer Graphics Forum, Wiley. 2012, 31 (5), pp.1765-1774. HAL Id: hal-00732426 https://hal-enpc.archives-ouvertes.fr/hal-00732426
9. Adams, C., Neel N., "Structure from Motion from a Constrained Orbiting Platform Using ISS image data to generate cloud height models.", presented at the NASA/CASIS International Space Station Research and Development Conference, Washington D.C., 2017.
10. Adams, C., Neel N., "The Feasibility of Structure from Motion over Planetary Bodies with Small Satellites", presented at the The AIAA/Utah State Small Satellite Conference - SmallSat, Logan Utah, 2017.
11. Likar J. J., Stone S. E., Lombardi R. E., Long K. A., "Novel Radiation Design Approach for CubeSat Based Missions." Presented at 24th Annual AIAA/USU Conference on Small Satellites, August 2010
12. Hartley R., Kahl F. (2007) Optimal Algorithms in Multiview Geometry. In: Yagi Y., Kang S.B., Kweon I.S., Zha H. (eds) Computer Vision – ACCV 2007. ACCV 2007. Lecture Notes in Computer Science, vol 4843. Springer, Berlin, Heidelberg
13. Chrzeszczyk, Andrzej & Chrzeszczyk, Jakub. (2013). Matrix computations on the GPU, CUBLAS and MAGMA by example.
14. Hassaballah, M & Ali, Abdelmgeid & Alshazly, Hammam. (2016). Image Features Detection, Description and Matching. 630. 11-45. 10.1007/978-3-319-28854-3_2.
15. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. 60(2), 91–110 (2004)
16. M. *Brown* and D. *Lowe*. Invariant Features from Interest Point Groups. In David Marshall and Paul L. Rosin, editors, Proceedings of the British Machine Conference, pages 23.1-23.10. BMVA Press, September *2002*.
17. Harris, C. and Stephens, M. (1988) A Combined Corner and Edge Detector. Proceedings of the 4th Alvey Vision Conference, Manchester, 31 August-2 September *1988*, 147-151.
18. Traa J., "Least Square Intercetion of Lines", UIUC 2013, http://cal.cs.illinois.edu/~johannes/research/LS_line_intersect.pdf
19. Wu C., Agarwal S., Curless B., SM Seitz. "*Multicore bundle adjustment"*. Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on …, 2011. 574, 2011.
20. Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing* (SGP '06). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 61-70.