Draft

Version 0.3

A Simulated Workflow Using Computer Graphics for UAS Based Photogrammetric 3D Reconstruction Accuracy Assessments

Prepared By:

Richard Slocum

Dr. Chris Parrish

January 14, 2017

Abstract

Structure from Motion (SfM) and MultiView Stereo (MVS) algorithms are increasingly being used to generate pointcloud data for various surveying applications from Unmanned Aerial System(UAS) based platforms, however the accuracy and sources of error in the resultant pointcloud across various use cases are difficult to realize without thorough experimentation. The acquisition of imagery and rigorous ground control data at field sites required for this experimentation is a time consuming and sometimes expensive endeavor. These experiments are also almost always unable to be perfectly replicated due to the numerous uncontrollable independent variables, such as solar radiation, sun angle, cloud cover, wind, objects in the scene moving, exterior orientation of cameras, and camera noise to name a few. The large number of independent variables creates a scenario where robust, repeatable experiments are cost prohibitive and the results are frequently site specific. Here, we present a workflow to render computer generated imagery using a virtual environment which can mimic all the independent variables that would be experienced in a real-world data acquisition scenario. The resultant modular workflow utilizes the open source software Blender for the generation of photogrammetrically accurate imagery suitable for SfM processing, with tight control on camera interior orientation, exterior orientation, texture of objects in the scene, placement of objects in the scene, and Ground Control Point (GCP) accuracy. The challenges and steps required to validate the photogrammetric accuracy of computer generated imagery are discussed, and an example experiment assessing accuracy of an SFM derived pointcloud from imagery rendered using a computer graphics workflow is presented.

1 Introduction

Three dimensional Geospatial pointcloud data with an accuracy <5cm and with a density greater than 10 points/m^2 has traditionally been acquired using either terrestrial or airborne lidar, but recent advances in Structure from Motion and MultiView Stereo algorithms have enabled their use to generate pointcloud products comparable to in density and accuracy to lidar data. SFM and MVS algorithms began development 30+ years ago (site) but have only begun to be utilized for commercial surveying applications recently due to the advances in camera hardware, Unmanned Aerial Systems (UAS), computer processing power, and commercial SFM-MVS software. Research into SFM and MVS in the geomatics community is currently focused on both the accuracy and potential applications of the commercial SFM and MVS software packages such as Agisoft Photoscan and Pix4d . The accuracy of SFM-MVS can vary greatly depending on a variety of factors , and the geomatics community is currently focused on determining potential use cases where SFM-MVS derived pointclouds can begin to replace lidar as an alternative surveying tool, without sacrificing accuracy. . The most common methodology for assessing the use cases and accuracy of SFM-MVS algorithms is to collect data in the field using a UAS and compare the data to lidar data. It is difficult to constrain many of the independent variables using this methodology, and the data acquisition can be expensive and time consuming. We propose a computer graphics based workflow to simulate various scenes and maintain full control over the ground-truth and the camera parameters. This workflow will allow geomatics engineers to perform more robust experiments to assess the feasibility and accuracy of SFM-MVS in various applications.

cite: OLD SFM-MVS

cite: Geomprph sfm paper

cite: accuracy vs factors

Should I cite specific use examples?

The 3D reconstruction methods used in most commercial software consist of an SFM algorithm first to solve for camera exterior and interior orientations, then an MVS algorithm to increase the density of the pointcloud. Unordered photographs are input into the software, and a keypoint detection algorithm such as SIFTis used to detect keypoints and correspondences between images. The SFM algorithm solves for camera interior orientation, exterior orientation, and generates a "sparse" pointcloud. A bundle adjustment is performed to optimize the matches. Without any additional information, the coordinate system is arbitrary in translation and rotation with an inaccurate scale. To further constrain the problem and develop a georectified pointcloud Ground Control Points(GCPs) and/or Initial Camera GPS Positions are used to constrain the solution. A camera calibration file can also be input to reduce the number of parameters that are solved for, however only inputting a camera calibration file will only help resolve the scale of the pointcloud coordinate system, not the absolute translation and rotation. These input data can be used to generate an absolute coordinate system either with a Helmert transformation after the pointcloud is generated, or using a nonlinear optimization within the bundle adjustment. The nonlinear optimization will generate more accurate results. The interior and exterior orientation for each image is used as the input

cite sift

cite: helmert uas

cite

to the MVS algorithm, which generates a more dense pointcloud. The MVS algorithm can generate more correspondences because it utilizes a 1D search along the epipolar line between two images due to the known interior and exterior orientation. For this reason, the accuracy of the MVS algorithm is highly dependent on the accuracy of the parameters calculated with the SFM algorithm. A detailed explanation of the various MVS algorithms can be found in paper. Each of these algorithms also assumes that the scene is rigid with constant lambertian surfaces, and deviations from these assumptions will effect the accuracy.

make sure this is true

cite MVS algorithms

Numerous studies have been performed to quantify the accuracy of the SFM-MVS algorithms in a variety of environments. The most common and robust method has been to compare the SFM-MVS derived pointcloud to a groundtruth terrestrial lidar survey . This comparison can exhibit errors in vegetated areas due to vegetation moving by blowing in the wind, and due increased uncertainty in the lidar pointcloud. Independent GPS control points have also been used, but result in fewer correspondences to compare with . The use of independent control points also can exhibit a bias when the points used are easily photo identifiable targets (eg. checkerboards). The highly textured independent control points will demonstrate a much better accuracy than a homogeneous surface, due to the lack of matching features, and therefore are not necessarily representative of the accuracy of the entire scene. The accuracy of SFM is adversely affected by: homogeneous scene texture, poor image overlapping, lens distortion not fully modeled by the nonlinear lens distortion equation, poor GCP distribution, inaccurate GCP or camera positions, poor image resolution, blurry imagery, noisy imagery, varying sun shadows, moving objects in the scene, user error in manually selecting image coordinates of GCPs, low number of images, or a low number of GCPs. With a computer graphics workflow, each of these variables can be constrained and varied in order to assess the effect on the overall accuracy of the scene. The accuracy of the resultant SFM-MVS pointcloud is also assessed by comparing it to an ideal groundtruth dataset with no uncertainty because it is all simulated data.

cite lidar accuracy studies

cite: sfm vs lidar

cite: sfm vs independent gcps

cite: sfm error sources

2 Computer Graphics for Remote Sensing Analysis

The field of computer graphics was first developed in the 19XXsand the advancement of the field has been predominantly driven by the desire to create more realistic video game and movie special effects. The package of software and algorithms to turn a simulated scene with various textures, material properties, and lighting into an image or sequence of images is called a render engine. While there are many different render engines available with different algorithms, they all follow a basic workflow, or computer graphics pipeline.

origin of CGI date

cite: computer graphics origins

First, A 3D scene is generated using vertices, faces, and edges. For most photo-realistic rendering, meshes are generated using an array of either triangular surfaces or quadrilateral surfaces to create objects. Material properties are applied to each of the individual surfaces to determine the color of the object. Most software

allow for the user to set a diffuse, specular, and ambient light coefficients as well as their associated colors in order to specify how light will interact with the object. The coefficient specifies how much diffuse, specular, and ambient light is reflected off of the surface of the object, while the color specifies the amount of red, green, and blue light that is reflected from the surface. The color using the material properties is only associated with each plane in the mesh, so for highly detailed coloring on objects there would need to be a large amount of faces. The more efficient way of creating detailed colors on an object without increasing the complexity of the surface of the objects is to add a "texture" to the object. A texture is an image which is overlaid on the mesh in a process called u-v mapping. The computer graphics definition for a "texture" object, is not to be confused with the SfM and photogrammetry definition, which is the level of detail and unique photo identifiable features in an image.

Once a scene is populated with objects and their associated material and texture properties, light sources and shading algorithms must be applied to the scene. The simplest method is to set an object material as "shadeless", which eliminates any interaction with light sources and will render each surface based on the material property and texture with the exact RGB values that were input. The more complex and photorealistic method is to place light sources in the scene. Each light source can be set to simulate different patterns and angles of light rays with various levels of intensity and range based intensity falloff. Most render engines also contain shadow algorithms, which perform ray tracing to determine what regions are shadowed by an object. Once a scene is created with light sources and shading parameters set, simulated cameras are placed to create renders of the scene. The camera translation, rotation, sensor size, focal length, and principal point are input and a pinhole camera model is used. The rendering algorithm generates a 2D image of the scene using the camera position and all of the material properties of the objects. The method, accuracy, and performance of generating this 2D depiction of the scene is where most render engines differ.

There are many different rendering methodologies, but the one chosen for this research is Blender Internal Render Engine which is a rasterization based engine. The algorithm determines which parts of the scene are visible to the camera, and assigns each object color to the pixel samples. This algorithm is fast, but is unable to perform some of the more advanced rendering features such as global illumination and motion blur. A more detailed description of common rendering engines and algorithms can be found XX.

cite: render engine summary

- Reference Others who have used CGI for remote sensing (dirsig, etc)

CGI photogrammetry

3 Renderer Accuracy Validation

There are many different render engines available to generate rendered imagery of a simulated scene. Before using a render engine to analyze surface reconstructions a series of validation experiments should be per-

formed to ensure that the render engine is generating imagery as expected. These validation experiments are performed to ensure that any resultant error in an uncertainty analysis is due to SFM algorithm, not inaccurate rendering. In this paper, we present an assessment and validation using Blender Internal Render Engine, but this validation methodology could be applied to any render engine. Note that there is no experiment presented to validate that accuracy of the lighting as the radiometric accuracy of the lighting is not the focus of this experiment. The authors recognize the render engine could also be validated by rigorously analyzing or developing the rendering source code, but believe these experiments provide a less time consuming methodology.

3.1 Photogrammetric Projection Accuracy

The first validation experiment ensures that the camera interior and exterior orientation are set accurately using a pinhole camera model. This experiment is performed by creating a simple scene consisting of a $1000m^3$ with a 10x10 black and white checkerboard on each wall, as depicted in Figure 1. The black and white corner of each checkerboard corner is at a known world coordinates. A series of images are rendered using various camera rotations, translations, focal lengths, sensor sizes, and principal point coordinates. To ensure that the images are rendered correctly, the coordinates of the checkerboard corners are calculated from the rendered imagery using a corner feature detector and compared to the expected coordinates of the targets using photogrammetric equations. The difference between the image derived coordinates and the photogrammetric equation derived coordinates should have a mean of 0 in both dimensions, and a subpixel variance on the order of the accuracy of the image corner feature detector. There should also be no correlation between the accuracy of the coordinate and the location of the coordinate in the image.

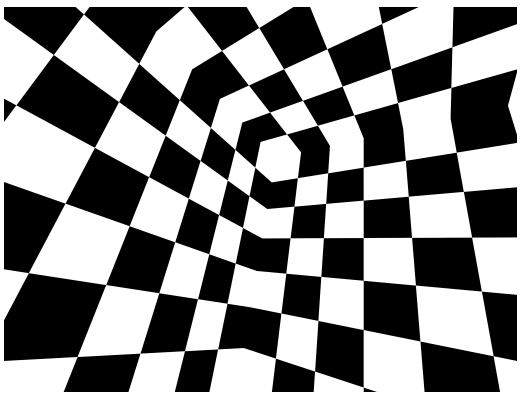

Figure 1: A cube with a 10x10 checkerboard pattern on each wall is used to validate the photogrammetric accuracy of the Blender Internal Render Engine.

To validate the photogrammetric projection accuracy of the Blender Internal Render Engine using this

experiment, a 1000m^3 cube was placed with the centroid at the origin. Five hundred images were rendered using five different interior orientations and random exterior orientations throughout the inside of the cube. These parameters were input using the Blender Python API, and their distributions are shown in Table 1. The accuracy of the imagery was observed qualitatively by plotting the photogrammetric equation calculated points on the imagery in Matlab to ensure a rough accuracy. Once the rough accuracy is confirmed, a nearest neighbor is used to develop correspondences between the Harris corner coordinates and the photogrammetric equation derived coordinates. The mean and variance of the differences between the correspondences in each experiment are shown in Table 2. The correlation between the difference in x, y, and radius versus several parameters shows no statistically significant correlation. The correlation results are summarized in Table 3.

Table 1: The position of the cameras used to render the imagery were uniformly distributed using parameters to full encompass every position and look angle within the box. Note that the translation was kept 1 meter away from the edge of the box on all sides.

Parameter	Minimum	Maximum	units
Translation X, Y, Z	-4	4	m
Rotation Theta, Phi	0	360	degrees
Rotation Omega	0	180	degrees

Table 2: The difference between the Harris Corner detected corners and the expected position of the corners from the photogrammetry equations is assessed to ensure that the rendering algorithm are working correctly. Note that the mean and variance of the difference between expected and detected are sub-pixel for each simulated camera, which suggests that the Blender Internal Renderer is accurate.

hFOV (°)	Horizontal	Vertical	# Corners	mean Δx	mean Δy	var Δx	var Δy
11.5	5184	3456	462	-0.0163	0.0035	0.0855	0.0827
29.0	3264	2448	3538	0.0050	0.0078	0.0915	0.0776
36.3	5456	3632	4093	0.0016	0.0116	0.0652	0.0715
36.9	4608	3456	4491	-0.0036	0.0041	0.0865	0.0705
46.8	4000	3000	7493	0.0033	0.0081	0.0797	0.0867

Table 3: The correlation between the errors and the location in the frame of the image is also calculated to ensure there is no systematic offset introduced due to the rendering. The Blender Internal Renderer shows no systematic biasing of the rendered image.

IV	DV	Slope	Bias	Different than 0 slope 0 bias?
x	dx			no
y	dx			no
r	dx			no
x	dy			no
y	dy			no
r	dy			no

To ensure that the variance is not an artifact of the rendering, an experiment was performed to determine

the expected accuracy of the Harris Corner detector. 1000 Simulated checkerboard patterns were generated with random rotations, translations, and skew to create a synthetic image dataset. The known coordinates of the corners were compared to the coordinates calculated with the Harris Corner feature detector, and the results are shown in in Table 4. From these results, the hypothesis that the variance of the rendered image coordinate error is statistically different than the variance of the simulated image coordinate error is rejected. Therefore, all the variance can be statistically attributed to the Harris Feature Corner detection algorithm, rather than the render engine.

Table 4: A checkerboard pattern is generated and then warped in Matlab before extracting the Harris Corner Features. The difference between the known checkerboard corners and the Harris corner features are on the same order of magnitude as the differences in the rendered imagery simulation. This suggests that the variance and mean offsets in the simulation are due to the uncertainty in the Harris Corner extraction, and not the renderer.

Name	# Correspondences	dX	dY	var X	var Y
Warped Imagery	390204	-0.0012	0.0075	0.0462	0.0474

3.2 Point Spread Function

The second validation experiment ensures that there is no blurring applied to the rendered image. Specifically, this test determines that the point spread function of the rendered imagery is a unit impulse. This test is performed by creating a white sphere placed at a distance and size such that it exists in only one pixel. The rendered image should therefore only contain white in the one pixel and not be blurred into any other pixels. This test is particularly important when antialiasing is performed, as the sampling and filter to combine the samples can sometimes create a blurring effect. For example, the default antialiasing in blender uses a "distributed jitter" pattern which "ensures that a certain amount of the sample color gets distributed over the other pixels as well." This effect can be seen in Figure 2, where the intensity of the white sphere is experienced in four of the neighboring pixels, even though the sphere should only be visible in one pixel. While this photogrammetric inaccuracy is minimal, the error could propagate into the resultant SFM derived pointcloud.

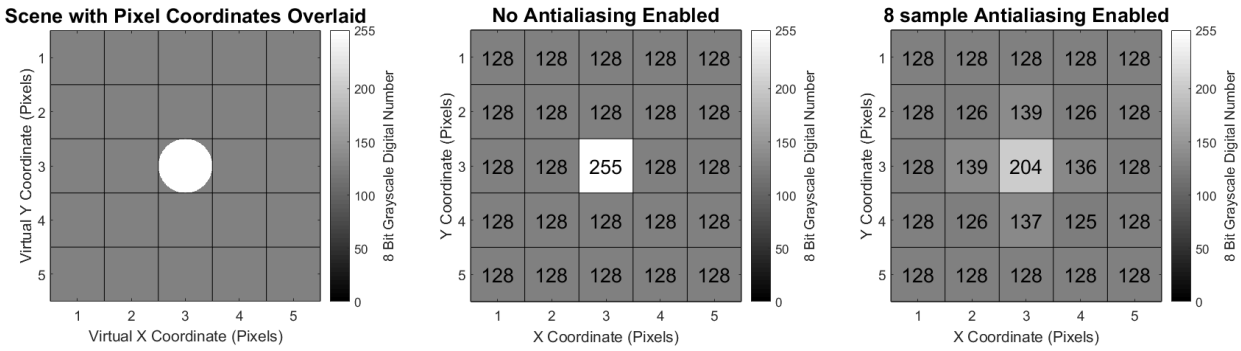

Figure 2: A circular plane placed so it is encompassed by the viewing volume of only the central pixel (right) is rendered with the no antialiasing (middle) and with 8 sample antialiasing (right).

To validate the point spread function of the Blender Internal Render Engine, a sensor and scene are set up such that the geometry of the sphere is only captured with one pixel in the render. This experiment ensures that any other pixels that contain white are an artifact of the rendering. Rendered imagery is shown with and without antialiasing. The antialiasing used is the default settings for the Blender Internal Render Engine (8 Samples, Mitchell-Netrevali filter). The rendered image with no antialiasing contains no blurring of the image, while the antialiased image contains a slight amount of blurring. The antialiased imagery renders a smoother, more photorealisitic imagery, and is deemed to be suitable for experimentation.

Need to word this better... I contradict myself by saying antialiasing is bad

3.3 Texture Resolution

The final validation experiment ensures that any textures applied to the objects in the scene are applied in a manner which maintains the resolution of the imagery. This validation experiment is performed by rendering a texture on a flat plane and rendering an image that contains a small number of the texture pixels. By qualitatively looking at the image, it should be clear that the desired number of pixels are in the frame, and no smoothing is being applied. When rendering textures in computer graphics there is an option to perform interpolation of the texture, which yields a smoother texture. An example of a texture with and without interpolation is shown in Figure X.

To validate the texture resolution of the Blender Internal Render Engine a black and white checkerboard pattern where each checkerboard square is 1x1 texel is applied to a flat plane such that each texel represents a 10cm x 10cm square. An image is rendered using a focal length and sensor size such that each texel is captured by 100 x 100 pixels. The rendered image is qualitatively observed to ensure that the checkerboard is rendered for each pixel.

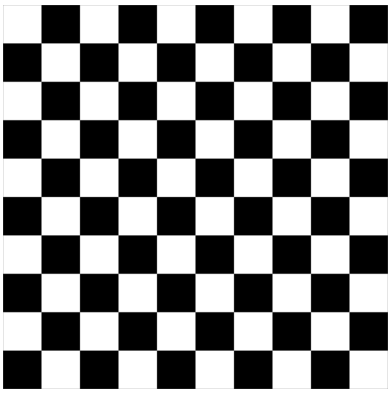

Figure 3: Each black and white square in this checkerboard represent one texel in the texture applied to the image. This rendered image demonstrates that the final texture that is rendered contains the full resolution of the desired texture, and that the Blender Internal Renderer is not artificially downsampling the texture.

4 Use Case Demonstration

An example experiment was generated as a proof of concept to demonstrate a potential workflow for testing the effect of various independent variables on SfM accuracy. This experiment specifically is generated to observe how the dense reconstruction quality setting in Agisoft Photoscan effects the resultant pointcloud accuracy. The results suggest that a higher quality dense reconstruction setting will generate more accurate results.

4.1 Experiment Design

A 200 m x 200m square surface was manually generated to simulate a topography with rolling hills using a 1 meter grid. A large 27m^3 meter cube was placed in the center of the scene to test surface reconstruction accuracy on regions with sharp corners and edges. Ten 1m x 1m x 0.05m square Ground Control Points (GCPs) are distributed evenly throughout the scene 0.25 meters above the ground surface. The material of all the objects in the scene are modeled as perfect lambertian surfaces. The topographic surface was textured using a combination of two textures. The first texture is a 7200x7200 pixel aerial image for an effective texel cite NZ footprint of 2.78cm square. The second texture is a 3456x3456 pixel image of grass was tiled ten times in both the x and the y dimension for an effective texel footprint of 0.58cm square. The image of grass was taken with a DSLR and manually edited to create a seamless texture for tiling. The aerial image and grass texture were merged together by setting the grass texture with an alpha of 0.15 and the aerial image layered beneath it with an alpha value of 1. The cube was textured using a 3456x3456 pixel seamless image of rocks that was derived from a DSLR image taken by the authors. This resulted in an effective texel footprint of 0.35cm. Each of the textures is set so that the coloring on the scene is interpolated between texels so that

there are no abnormal effects on the edge of pixels. An aerial image and oblique image of the scene are shown in Figure 4.

make this figure better. break into 2 figures. one shows layout of cameras and GCPs, one shows close up of gcp and box

The scene was illuminated using a "Sun" lamp where all of the light rays are parallel to each other. The light is initially directed at nadir and is linearly interpolated to a 30 degree rotation about the x axis for the final image. The sun has a constant intensity of 1, and emits white light with values of 1 for the R,G, and B. To improve lighting in shadowed regions, an ambient light source was added with an intensity value of 0.25 and values of 1 for R,G, and B.

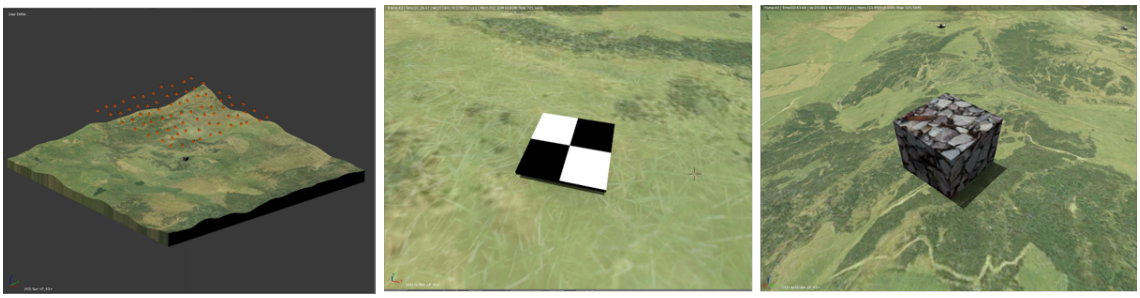

Figure 4: The scene was generated in Blender to represent a hilly topography with 10 Ground Control Points distributed throughout the scene and a 3m cube placed in the center.

A camera is created in Blender with parameters meant to emulate a Sony A5000 with a 16mm lens and 5456x3632 pixel sensor. A flight path is generated to create a Ground Sampling Distance (GSD) of 1.00cm and an overlap and sidelap of 75% each. In order to reduce imaging the edge of the simulated topographic surface, the inner 100m x 100m of the topography are selected as the Area of Interest. Using these constraints, a trajectory consisting of 77 images distributed across 7 flight lines is generated with nadir looking imagery, as shown in Figure 5. In order to generate imagery that is more representative of a real world scenario with a UAS, 1 sigma gaussian noise in meters is added to the camera translation in each of the three dimensions and 2 sigma gaussian noise in degrees is added to the camera rotation about each of the three axes. Imagery is then rendered using Blender Internal Render Engine with the default 8-sample antialiasing enabled.

how long did this take?

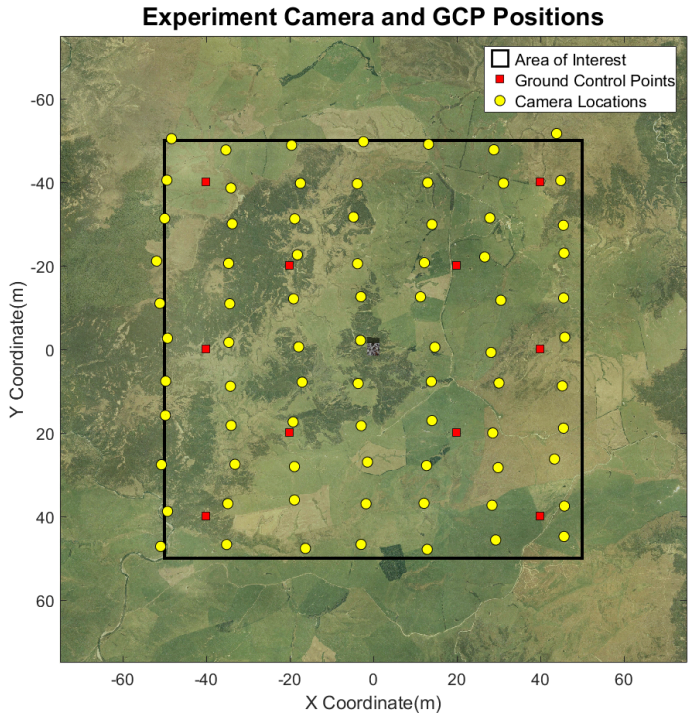

Figure 5:

The imagery that is output from Blender is postprocessed in Matlab to simulate nonlinear brown distortion , vignetting, salt/pepper noise, gaussian noise, and gaussian blur. In order to adequately apply fisheye distortion and gaussian blur, the imagery was rendered at a larger resolution than the sensor and then cropped after the filtering was applied. The constants used in this postprocessing are shown in Table 5

cite brown distortion

need to add equations for brown distortion, vignetting, add distortion eqn units, etc... this could get cumbersome...

Table 5: .

Parameter	Value	units
Distortion k_1	-4	f
Distortion k_2	-4	f
Distortion k_3	-4	f
Distortion k_4	-4	f
Distortion p_1	-4	f
Distortion p_2	-4	f
Vignetting v_1	10	DN
Vignetting v_2	0.2	DN
Vignetting v_3	0	DN
Salt Noise Probability	0.01	%
Pepper Noise Probability	0.01	%
Gaussian Noise Mean	0	DN
Gaussian Noise Variance	0.02	DN
Gaussian Blur Sigma	1	

4.2 Processing Methodology

The resultant imagery was processed using the commercial software Agisoft Photoscan Pro using the settings shown in . The dataset was processed by inputting the position of the cameras, the position of the GCPs, and the camera calibration file with no uncertainty or error added. Additionally, the pixel coordinates of the GCPs, which are traditionally clicked by the user by varying degrees of accuracy are calculated using photogrammetric equations and input into the program. A nonlinear adjustment is performed using the "optimize" button, and the reported total RMSE for the GCPs is 0.38mm. This represents an idealized scenario which is currently unrealistic for a real world scenario as traditionally there will be error in the GCP surveyed markers, the UAS position, the user clicking of pixel coordinates for GCPs, and in the calculation of the camera calibration.

cite table

A dense reconstruction is performed using the "aggressive" filtering and each of the quality settings available in Photoscan. The quality settings downsample the imagery for faster processing, and ultrahigh becomes quickly unattainable to users without purpose built CPUs with a large amount of RAM. . The processing time and number of points for each pointcloud is shown in Table 6. The computer used for this experiment is a Windows 7 Desktop PC with a Intel Xeon CPU E5-1603 @ 2.80GHz, GeForce GTX 980 Graphics card (4Gb), and 32Gb of RAM.

better describe quality and cite photoscan manual

Table 6: Add caption

Pointcloud Name	Time (Hours:Minutes)	Number of Points	RMSE Mean (m)	RMSE std (m)
sparse	00:36	22,214	-0.0001	0.0028
dense lowest	00:03	716,331	-0.0066	0.0323
dense low	00:09	2,886,971	-0.0020	0.0154
dense medium	00:30	11,587,504	-0.0005	0.0077
dense high	02:19	46,465,218	-0.0002	0.0044
dense ultrahigh	11:54	186,313,448	-0.0002	0.0026

Each of the dense pointclouds is processed using CloudCompare and compared to the groundtruth blender mesh using the "point to plane" tool. This tool calculates the signed distance of every point in the pointcloud to the nearest surface on the mesh, using the surface normal to determine the sign of the error. Each pointcloud was then exported and analyzed in Matlab to determine how the dense reconstruction quality setting effects the pointcloud error.

cite

cite

4.3 Results

The error was first visualized spatially for each reconstruction by gridding the pointcloud elevation and error using a nearest neighbor gridding technique. The number of points and standard deviation of points in each

grid cell are also visualized. The results for the medium quality dense reconstruction are shown in Figure 6. These plots are useful to begin to explore the spatial variability in both the density and the errors in the data. One initial observation for this dataset is that there is a larger standard deviation of error at the edges of the pointcloud outside the extents of the AOI. This is due to the poor viewing geometry at the edges of the scene, and suggests that in practice these data points outside of the AOI should be either discarded or used cautiously.

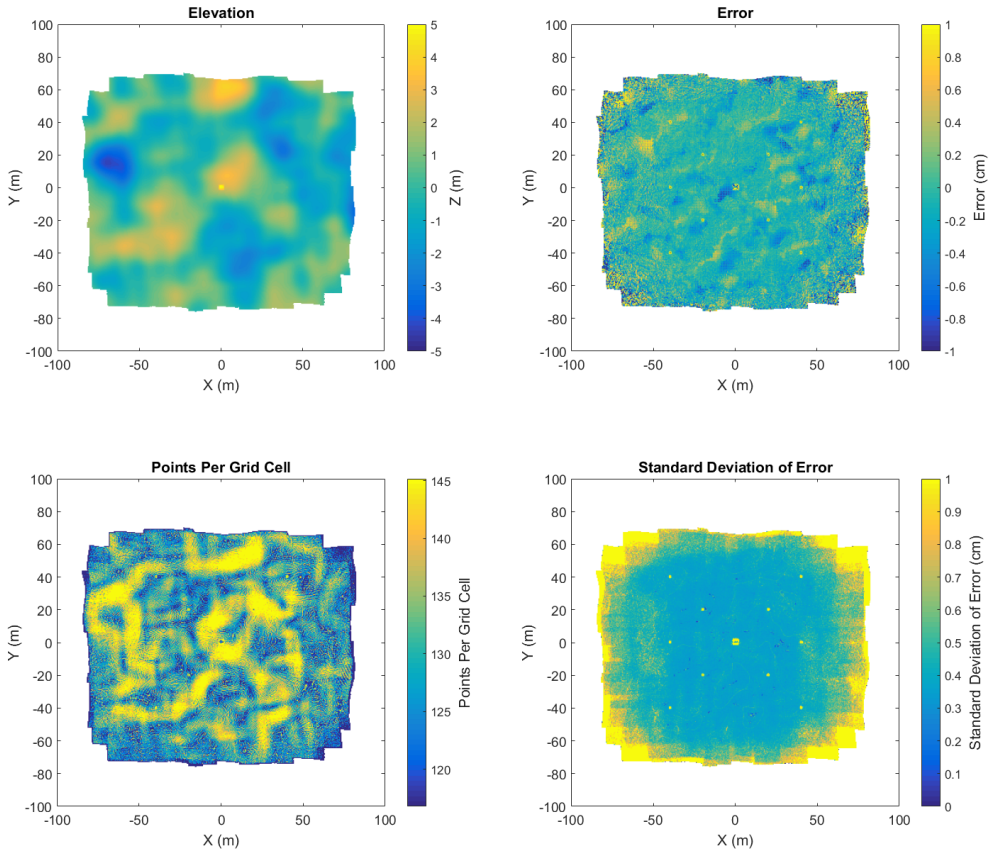

Figure 6: The elevation, error, number of points, and standard deviation of error are gridded to 0.5 meter grid cells using a nearest neighbor gridding algorithm and visualized.

To qualitatively observe the effect of different quality dense reconstructions, a plot showing the true surface and the points from each construction in a 0.5 meter wide section of the 27m^3 box is shown in Figure 7. Notice that the accuracy of each pointcloud at the sharp corners of the box improves as the quality of the reconstruction increases. This observation suggests that higher quality dense reconstruction settings will increase accuracy in regions with sharp corners.

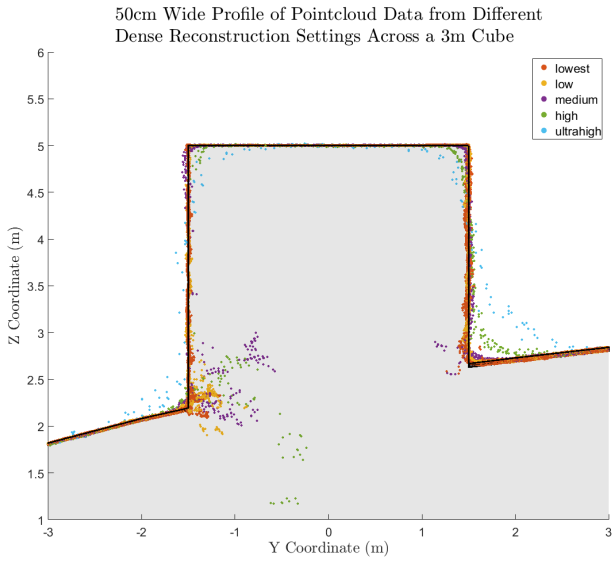

Figure 7: A half meter wide section of the scene containing a box is shown with the dense reconstruction pointclouds overlaid to demonstrate the effect of pointcloud dense reconstruction quality on accuracy near sharp edges.

A more quantitative, statistical assessment is performed to assess the error throughout the entire scene by calculating a histogram for the distribution of error in each pointcloud, as shown in Figure 8. These distributions bolster the conclusion derived from the box profile plot, which is that a higher quality dense reconstruction settings yield more accurate results than a lower quality reconstructione. While the accuracy of the GCPs as provided in Photoscan were an average of 0.38mm RMSE, the accuracy of the dense reconstruction ranges in standard deviation from 2.6mm to 32.3mm. This observation indicates that the GCP accuracy table is insufficient as a metric to depict the accuracy of the resultant dense pointcloud. While these conclusions suggest general trends. further experimentation is required in order for the accuracy and magnitudes of the distribution to be generalized.

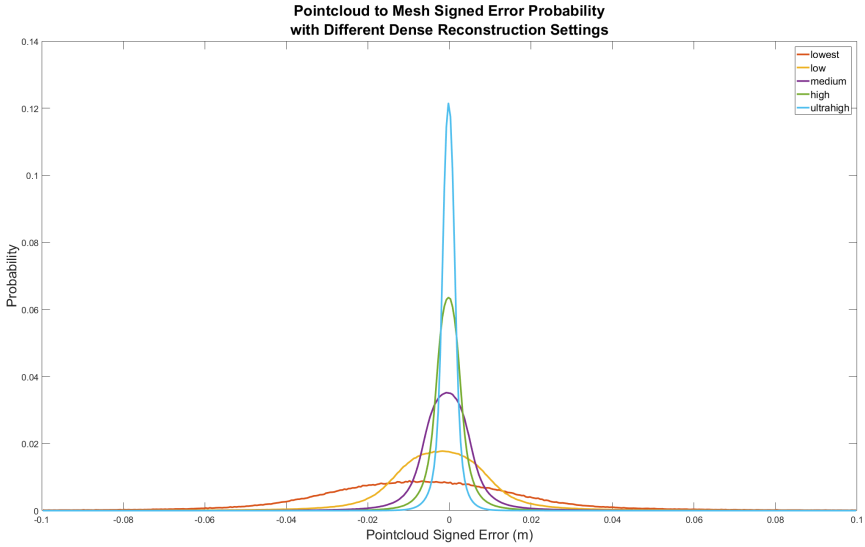

Figure 8: The signed error probability distribution for each of the calculated dense pointclouds suggests that the higher that dense reconstruction quality, the better the accuracy of the pointcloud.

5 Discussion

The use case demonstration provides an example of the type of rigorous analysis that can be obtained by utilizing computer generated imagery, rather than field data. It is important to note that the data and results associated in this experiment are closely coupled to the texture and topography of the scene. Future work will vary these independent variables in order to assess their effect on pointcloud accuracy.

The first conclusion from this experiment is that the error and standard deviation of error are larger for points outside of the area of interest, which in this experiment was -50m to 50m in both the x and y directions. This is shown in the spatial error plot in Figure 6. The cause of this error is the poor viewing geometry of these points, where they are only seen by a few cameras at oblique angles. In practice, these points should be included in the final data product with caution, as it is shown here that the errors can be significantly greater than in the AOI.

The second conclusion from this experiment is that a higher quality dense pointcloud results in a more accurate pointcloud, as shown qualitatively in Figure 7 and quantitatively in Figure 8. The quality settings in Photoscan determines the amount of downsampling of the imagery that should occur before performing the reconstruction algorithm. The downsampling of the imagery removes some of the finer texture details in the imagery, and therefor reduces the quality of the keypoint matching. The authors recommend using the highest quality setting that the computer processing the dataset can handle. For this experiment, a relatively small number of images (77) were used to create the dense pointcloud, which took almost 12 hours for the

highest pointcloud setting. The resultant pointcloud for this setting also contained 186 Million points, which caused some pointcloud data viewers and processing to fail due to memory issues. For this reason, ultra high may not be a possible solution for all experiments.

The third conclusion is that the RMSE of the GCP control network as shown in Agisoft Photoscan is insufficient to characterize the accuracy of the resultant dense pointcloud. In this extremely idealized experiment, where the GCP positions, pixel coordinates of GCPs, camera positions, and camera calibration were all input precisely, the GCP control network 3D RMSE was 0.38mm. The smallest standard deviation of the pointcloud error, using the "ultra high" quality setting, is 2.6mm and the largest standard deviation, using the "lowest" setting, is 32.3mm, as shown in Table 6. Further experimentation is needed to determine the relationship between the GCP total RMSE and the statistics of the dense pointcloud, and this workflow is well suited to perform the experimentation required to make accurate generalized conclusions with confidence.

5.1 Methodology Implications

This methodology generates photogrammetrically accurate imagery rendered using a pinhole camera model of a scene with various textures and lighting which is then processed to assess SfM pointcloud accuracy. The rendered imagery can be postprocessed to add noise, blur, nonlinear distortion, and other effects to generate imagery more reminiscent of that from a real world scenario. The accuracy of the camera trajectory, GCP position, camera calibration, and GCP pixel coordinates in each image can also be systematically adjusted to simulate uncertainty in a real world scenario. The ability to adjust these parameters enables a user to perform a sensitivity analysis with numerous combinations of independent variables.

While this methodology enables the user to perform repeatable, accurate experiments without the need for time consuming field work, there are currently some limitations in the experiment methodology when utilizing the Blender Internal Render Engine. First, the internal render engine does not handle global illumination, and therefore light interactions between objects are not modeled. A second limitation of the lighting schema is that the radiometric accuracy has also not been independently validated. There are a few methods within the render engine which effect the exposure and gamma of the resultant imagery. For this experiment, these exposure and gamma values were set at the default and provided imagery that was not over or underexposed. While the lighting in the scene using the Blender Internal Render Engine does not perfectly replicate physics based lighting, the absolute color of each surface of an object is constant and perfectly lambertian. The keypoint detection and SfM algorithms utilize gradients in colors and the absolute colors of the scene, and the accuracy should not be effected by this inaccuracy.

Another more broad source of inaccuracy in the Blender Internal Render Engine methodology is the

methodology to convert the scene to pixel values relies in an integration over a finite number of subpixel super-sampling. This deviates from a real world camera where the pixel value is a result of an integration over all available light. The Blender Internal Render Engine uses the term "antialiasing" to describe a supersampling methodology for each pixel, which can supersample a up to 16 samples per pixel. This small, finite number of samples per pixel can induce a small amount of inaccuracy when mixed pxiels are present. These small inaccuracies, though, are small enough to be deemed negligible for this experiment.

A potential source of uncertainty induced into the system is the use of repeating textures to generate a scene. In the use case provided in Section X, the grass texture is repeated 10 times in both the x and y direction. This repeating pattern was overlaid onto another image, to create different image color gradients in an attempt to generate unique texture features. Despite this effort, it is possible that keypoint detection and matching algorithms could generate false positives which may bias the result if not removed as outliers. In this experiment, this effect was not observed, but if the scene is not generated carefully, these repeating textures could induce a significant amount of inaccuracy in the SfM processing step.

fix

6 Conclusion

The methodology presented where a UAS based image acquisition is simulated using the Blender Internal Render Engine enables research into the performance of SfM and MVS algorithms without the need for costly field experiments. The accuracy of the simulated groundtruth data enhances the confidence in sensitivity analyses and allows for numerous repeat experiments. An example use case is presented to explore the effect of the Agisoft Photoscan dense reconstruction setting on the accuracy of the output pointcloud. Results suggest that the errors decrease as the dense reconstruction setting is increased. Secondary results suggest that the data points outside of the AOI should be either discarded or used with caution, as the accuracy of those points is higher than that of the pointcloud within the AOI.

As the application of SfM-MVS algorithms continue to expand into new fields, it is important to first test the accuracy of the pointcloud when performing new experiments. This methodology could also be used from alternative sources of imagery, such as handheld or vehicle based imagery. Future work will focus on various sensitivity analyses to assess the accuracy of new algorithms and applications.