Draft

Version 0.1

A Simulated Imagery Rendering Workflow using Blender for Photogrammetric 3D Reconstruction Accuracy Assessments

Prepared By:
Richard Slocum
Dr. Chris Parrish

December 31, 2016

Abstract

Structure from Motion (SfM) and MultiView Stereo (MVS) algorithms are increasingly being used to generate pointcloud data for various surveying applications, 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 a "sparse" pointcloud. A bundle adjustment is performed to optimize the matches. Without any additional information, the coordinate system is arbitrary in translation, rotation, and scale. To further constrain the problem and develop a georectified pointcloud Ground Control Points 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 be done 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 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 drastically effect the accuracy.

cite sift

cite: helmert uas

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 . 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 a scene with no uncertainty because it is simulated.

cite lidar accuracy studies

cite: sfm vs lidar

cite: sfm vs independent gcps

2 Computer Graphics for Remote Sensing Analysis

The field of computer graphics was first developed in the 19XXs and 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 into an image or movie is called a renderer. While there are many different renderers available with different algorithms, they all follow a basic workflow, or computer graphics pipeline.

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 popular way of creating detailed colors on an object is to add a "texture" to the object. A texture is an image which is overlaid on the image in a process called u-v mapping. The computer graphics definition for "texture", 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 it as a constant color or texture. 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 renderers 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, cameras are placed to create renders of the scene. The camera is set using exterior and interior orientations, though with slightly different terminology. 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 and performance of generating this 2D depiction of the scene is where most renderers differ.

There are many different rendering methodologies, but the one chosen for this research is Blender Internal Renderer which is a rasterization engine. The algorithm determines which parts of the scene are visible to the camera, and assigns each object color to the scene. This algorithm is very 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.

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

3 Renderer Accuracy Validation

There are many different renderers available to generate rendered imagery of a simulated scene. Before using a renderer to analyze surface reconstructions a series of validation experiments should be performed to ensure that the renderer 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 Renderer, but this validation methodology could be applied to any renderer. 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 renderer 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.

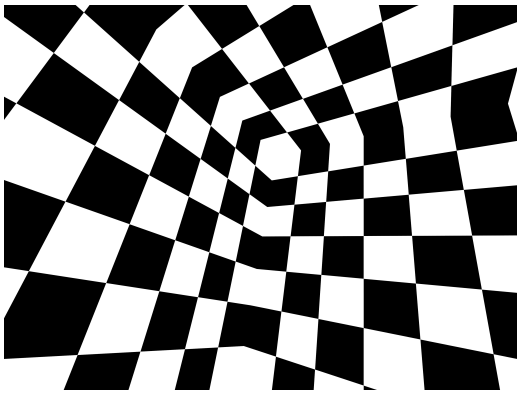

Figure 1: A cube with a 10x10 checkerboard pattern on each wall is used to validate the photogrammetric accuracy of the Blender Internal Renderer.

To validate the photogrammetric projection accuracy of the Blender Internal Renderer 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

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 renderer.

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

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.

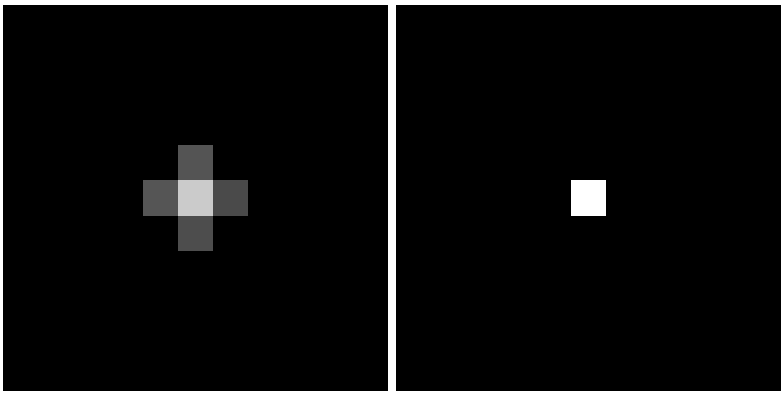

Figure 2: A sphere placed so it is encompassed by the viewing volume of only the central pixel is rendered with the default "distributed jitter" antialiasing (left) and with no antialiasing (right). In this instance, the image with no antialiasing is more photogrammetrically accurate.

To validate the point spread function of the Blender Internal Renderer, 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 Renderer (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 Renderer 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.

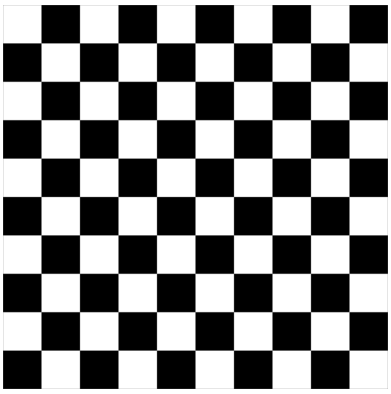

Figure 3: Each black and white square in this checkerboard represent one texel in the texture applied to the image. This rendered image demonstrates that the final texture that is rendered contains the full resolution of the desired texture, and that the Blender Internal Renderer is not artificially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

4 Proof of Concept

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.

A 100m x 100m DSM was manually generated in Blender using N vertices and N unique triangular planes to represent a hilly surface. The specular response of the surface was defined as perfectly lambertian, and the color of the surface was generated from two image textures. The first image texture is a XX Mp seamless grass image, that was taken from a UAV at XXX. This texture is repeated 5 times in both the X and Y dimensions, for a texel ground spacing of XX cm. The second texture is a XMp image taken from a UAV from at xxx. This texure is not repeated, and therefore has a texel ground spacing of XX cm. These images are mosaiced together using an intensity value of X for the repeating grass texture, and an intensity value of Y for the overview image. Both textures are interpolated in the renderer to avoid crisp edges or corners in between texels. A large 5m x 5m cube is placed in the center of the scene to observe the inability of SfM to generate crisp corners. The cube is textured using a XMp image of asphalt that was taken by a DSLR. The texel ground spacing is XX cm. Eleven 0.5m x 0.5m Black and White Checkered, lambertian GCPs are distributed in the scene to simulate being laid on the simulated hilly surface in an adhoc manner. To illuminate the scene, the sun is placed at a nadir angle for the first image, and its angle is linearly interpolated about the x axis so that the final image is rendered with a 30 degree solar incident angle. The ray tracing algorithm is set up so that shadows are cast by objects within the scene. An ambient light source for the environment is set to be 10% the value of the solar illumination, to prevent shadowed areas from rendering completely black.

A trajectory is generated in Matlab to simulate the motion of a small action camera on a UAS at an altitude of 25 meters flying a grid pattern across the scene with X sidelap and X overlap. To avoid imaging the edge of the simulated hilly topography, the outermost 10 meters of the scene are not considered in the simulated flight planning. The camera pose is nadir, but with a $\pm 5°$ random offset in roll, pitch, and yaw angles of the uas. The Blender Internal Render was used to render 100 images, which took approximately X minutes on a XX computer. To simulate more nonidealized imagery, the raw images were postprocessed to add gaussian blur, gaussian noise, salt and pepper noise, and nonlinear distortion governed by the brown distortion model using a Matlab script.

The postprocessed imagery was loaded into Agisoft Photoscan (version XX), and processed with settings shown in Table X. Notice that the GCP positions, camera positions, and camera interior orientation are input with no uncertainty, since the scene has been simulated. 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. The accuracy of the control points as reported by Photoscan are shown in Table X. Note that as a result of the unrealistic accuracy and of the data input into the program, the errors are reported in millimeters. This is not realistic to expect from a real world scenario,

but provides an example best case scenario for what to expect from SfM. The result of the processing is an orthophoto, with 1cm resolution, a sparse pointcloud with N points, and a dense pointcloud with N points. These data are then compared to the groundtruth data, which in this case is the model and a simulated orthophoto from Blender.

Cloud Compare is used to compare the points to the mesh using the points to plane tool. The resultant accuracies of the sparse and dense pointcloud are exported to Matlab, and gridded using an IDW . The variance of the accuracy is also visualized by binning the accuracy values into the nearest grid node and calculating the variance of the values in each node. These data are visualized in Figure X and summarized in Table X.

5 Discussion

5.1 Proof Of Concept Discussion

- The GCP accuracy is much better than the accuracy of the resultant surface... gcp accuracy is not necessarily indicative of pointcloud accuracy
- the accuracy in this experiment is closely coupled to the texture of the scene
- Cube has rounded edges in dense pointcloud
- Accuracy in shadows?
- Accuracy outside GCPs?
- Sparse accuracy as a function of number of images, reprojection error, etc
- Here I just want to give a taste of some preliminary results from this simple experiment to really spell out capabilities

5.2 Methodology Discussion and Implications

- methodology allows for robust experimentation and testing of numerous independent variables
- radiometric accuracy is not currently accounted for, as SfM only cares about texture... could incorporate later

6 Conclusion

- recap. potential future work.