# 3D Reconstruction from Multi-View Stereo: Implementation Verification via Oculus Virtual Reality

Andrew Moran
MIT, *Class of 2014*
andrewmo@mit.edu

Ben Eysenbach
MIT, *Class of 2017*
bce@mit.edu

## Abstract

*We propose a novel approach to visualizing computer vision pipelines. We demonstrate how an Oculus Rift can be used to visualize the output of various spatial algorithms, allowing researchers to better understand how existing algorithms work and where they fail. Then, using our 3D visualization techniques, we visually experiment with and evaluate our implementation of a multi-view stereo pipeline.*

## 1. Introduction

Computer vision pipelines are complex. Constructing one requires choosing a set of algorithms, and for each, selecting a number of parameters.

Once a pipeline has been compiled, it is very difficult to figure out how changes and minor adjustments will affect performance. Thus, most researchers visualize the output of each step of their pipeline. This lets them decouple the system and choose optimal algorithms and parameters for each step individually. Evaluating each step can be done quantitatively or qualitatively.

By visualizing a multi-view stereo pipeline, we demonstrate how our techniques for 3D visualization can be used to for qualitative evaluation.

## 2. Current Approaches

### 2.1. Quantitative Methods

Quantitative approaches usually work best when ground truth values are known. Without ground truth data, an error function encapsulating what a 'good' output looks like must be constructed. It is difficult to construct such a function without relying on the same assumptions as the pipeline.

Even when the ground truth is known, algorithms may perform poorly (according to some metric) for various reasons. In this case, it may be possible to take ideas from multiple low performing pipelines and combine them to create a high performing pipeline. However, by only looking at their (low) performance scores, it would be impossible to do this.

### 2.2. Qualitative Methods

Qualitative approaches to evaluating algorithms can work in some scenarios where quantitative approaches fail. By visualizing the output of an algorithm, researchers can validate and visually compare outputs when parameters are tweaked. Likewise, on broken pipelines, visualization helps with debugging.

Existing techniques for visualizing the output of algorithms which operate in two dimensions work well. For example, it is easy to evaluate the performance of panorama stitching, a 2D process, by viewing the constituent images superimposed on top of each other.

For algorithms which operate in three dimensions, such as stereo reconstruction, it can be difficult to understand the performance of an algorithm when visualized in two dimensions. Using software which allows researchers to manipulate a 3D model of the output helps, but information is still lost when projecting the 3D output to a two dimensional screen.

## 3. Virtual Reality

### 3.1. Research Potential

Virtual Reality exploits human ability to understand, navigate, and interact with 3D worlds. It makes sense to try to represent 3D data in a way which mimics objects and scenes humans encounter in the real world. [14, 3].

### 3.2. Our Approach

Wanting to visualize information in a 3D environment, we used an Oculus Rift, a 3D, head-mounted, virtual reality display.[1] To render 3D scenes, our project utilizes the Unity3D game engine [2], which supports stereoscopic displays, like the Rift.

[1] The techniques described would work equally well for other 3D displays.

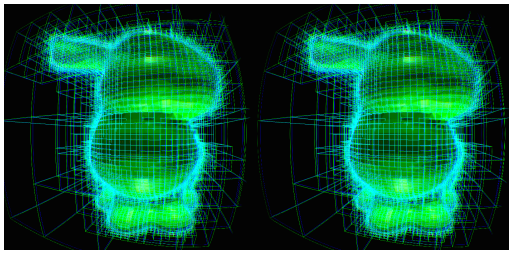

Figure 2. Stereoscopic view depicted by the Oculus Rift.

## 4. Pipeline Visualization

We propose visualizing the multi-view stereo pipeline using a 3D virtual reality display. This allows users to navigate around an object and inspect different areas as they would in the real world with a physical object. This natural interaction makes it easier to evaluate the performance of processes within a pipeline, and compare outputs when parameters are changed. For multi-view stereo, we highlight camera pose estimation, dense reconstruction, and surface reconstruction.

### 4.1. Camera Pose Estimation

#### 4.1.1 Feature Matching

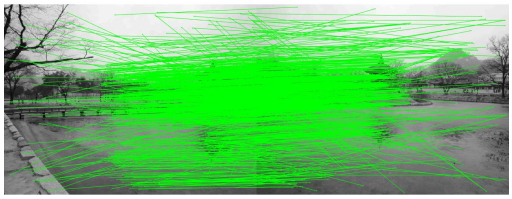

Figure 3. Visualization of keypoint matching. Because this is a 2D process, it is easy to visualize (in two dimensions).

The first step in most multi-view stereo pipelines is to find correspondence points between images. Once these points are known, the relative position of the cameras which took the images can be determined.

First, particular features of interest in each image are selected. There exist numerous algorithms for selecting such points (SIFT, SURF). The general idea of these approaches is to find features which are unique enough such that finding a similar feature in another image indicates with high probability that the two features correspond to the same object.[5]

Once these features have been extracted from each image, pairwise matches must be found. Not all features will be matched. A common criteria for whether to accept a match is to check if the two features are each the best match for the other. Another approach is to match one feature to another if the best alternative match is much worse than the best match.[5]

Feature matching is a two dimensional process, so it makes sense to visualize this step in 2D. We recommend using existing 2D visualization tools, comparing images side-by-side, to verify the output of feature matching algorithms and tune parameters.

#### 4.1.2 Bundle Adjustment

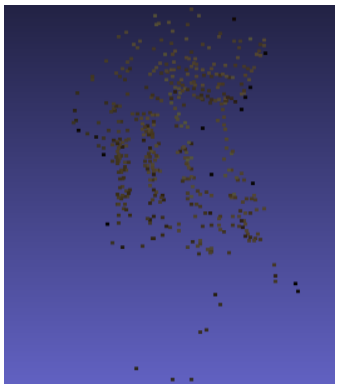

Figure 4. Example of a sparse reconstruction of Temple.[19] Notice that it is difficult to verify that the points are in the correct locations in 3D space.

The next step is to using the correspondence points to estimate the the pose (location and direction) from which each image was taken. If we only wanted to find the orientation of one camera relative to another, we could use RANSAC to fit a homography (with the Discrete Linear Transform).[7] However, applying this technique to all pairs of images would not give a globally consistent pose graph.[6]

Instead, we want to estimate camera poses and feature point locations simultaneously. One approach, known as Bundle Adjustment, incrementally alters camera poses and feature point positions to minimize *reprojective error*. This process minimizes a series of nonlinear equations; the Levenberg Marquardt Algorithm for nonlinear least-squares is commonly used as a subroutine. At the end of this process, Bundle Adjustment outputs a pose for each camera and a position for each feature point, both with respect to a global coordinate system.[20, 17]

The output of this process can be difficult to evaluate quantitatively. A researcher can examine the reprojective error of optimal locations, but that can be difficult to comprehend because it is highly dependent on image resolution, on image scale, and on how feature matching was performed. If ground truth information is known, the output of Bundle Adjustment can be compared against it. However, few large datasets are equipped with camera poses.

Viewing the camera poses and feature points in 3D solves many of these problems. Given a set of images, a human can easily figure out how images were taken with respect to one another. Then, if she is put in a virtual 3D environment displaying camera poses and feature points, it is easy for her to check whether camera poses and feature point locations are consistent with the input images.

## 4.2. Dense Reconstruction

Now, a scene can be reconstructed from calibrated cameras. Common approaches to this problem include: (1) searching along equipolar lines; (2) starting with a volume which encloses the region of interest, and removing *voxels* which are not photoconsistent; and (3) generating stereo depth maps for pairs of images, and then fusing them together.[9, 8]

Similar to Bundle Adjustment, it's difficult to quantitatively evaluate the output of dense reconstruction without ground truth data. Human observers often intuitively know the 3D geometry of the object of interest, but are unable to translate their mental model of the object into a quantitative model to evaluate their algorithm against. Nonetheless, if they are able to see the resulting point cloud in 3D, it is easy for them to verify that the dense reconstruction algorithm has performed as desired. If not, they can pinpoint where their point cloud is incorrect and make adjustments to their algorithm.

## 4.3. Surface Reconstruction

Many multi-view stereo algorithms output a set of points which lie on the surface of objects in the scene. These points can be analyzed at this stage, but it is often preferable to reconstruct the 3D surface(s) from which the points were sampled. This sort of problem also arises when analyzing LIDAR scans.

We used our visualization technique to analyze one surface reconstruction algorithm using poisson methods.[13] This algorithm estimates normals for every point in the point cloud and then constructs a function which is positive inside the object and negative outside the object. The surface where the function is zero is extracted, and taken to be the surface of the object.

### 4.3.1 Octree Representation

Before reconstructing the surface, we organize the point cloud into a spatial data structure. Multi-dimensional trees such as octrees and kd-trees are frequently used because geometric algorithms, such as nearest neighbor, can be efficiently implemented with them.[13] These tree data structures recursively partition 2D/3D space. For octrees, each node in the tree corresponds to a cuboid. Each has eight children, corresponding to partitioning along the $x$, $y$, and $z$ axes. Two parameters determine the final structure of the tree: a criteria for when to partition a cuboid, and a maximum tree height. Visualizing using a Rift allows users to verify that the octree was constructed correctly and helps them understand how the octree changes when parameters are altered.

### 4.3.2 Normal Estimation

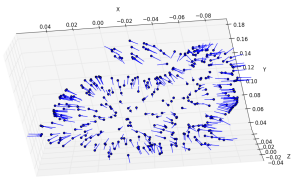

Figure 7. Normals computed for the Stanford Bunny point cloud. It is difficult to tell if the normals are correct because doing so required knowing the 3D position of the points, which cannot be extracted from this 2D image.

Many surface reconstruction methods require that each point has an estimated normal. This presents a Chicken-and-Egg scenario: normals are needed to construct a surface; however, the underlying surface is required to calculate normals.

Nonetheless, the surface normal at a point can be estimated by looking at neighboring points. These points form a probability distribution over point locations. Level surfaces of this distribution (e.g. 90% of points fall within this surface) form ellipsoids. Remarkably, the axes of these ellipsoids are the eigenvectors of the covariance matrix, scaled by their eigenvalues. Then, we can approximate the surface normal by taking the eigenvector with the smallest eigenvalue.

Eigenvectors are invariant to scale, so normals must then be oriented such that they all point from interior to exterior. This is completed by fixing the orientation of a known point (at an extrema), and propagating orientation information to neighboring points. Specifically, we find the neighboring point which has the normal most parallel to a point with a known normal orientation. We flip the first point's normal if necessary. This method is analogous to creating a minimum spanning tree over the points, where the distance $d$ between two points, $p_1, p_2$ is defined as:

$$d(p_1, p_2) = 1 - p_1.normal \cdot p_2.normal$$

Similar to previous steps in the pipeline, it is easier to evaluate the orientation of the normals when the normals and the minimum spanning tree are visualized in 3D.

#### 4.3.3 Function Construction

Poisson surface reconstruction then creates a function $V : \mathbb{R}^3 \rightarrow \mathbb{R}$ which is positive inside the object and negative outside the object. In poisson methods, we view the surface normals of our point cloud $P$ as the gradients of $V$:

$$\forall p \in P, \quad \nabla V(p) \approx p.normal$$

Applying the divergence operator to both sides, we get a discrete poisson equation:

$$\forall p \in P, \quad \nabla \cdot \nabla V(p) = \Delta V(p) \approx \nabla \cdot p.normal$$

[22, 13]

#### 4.3.4 Computing the Zero-Surface

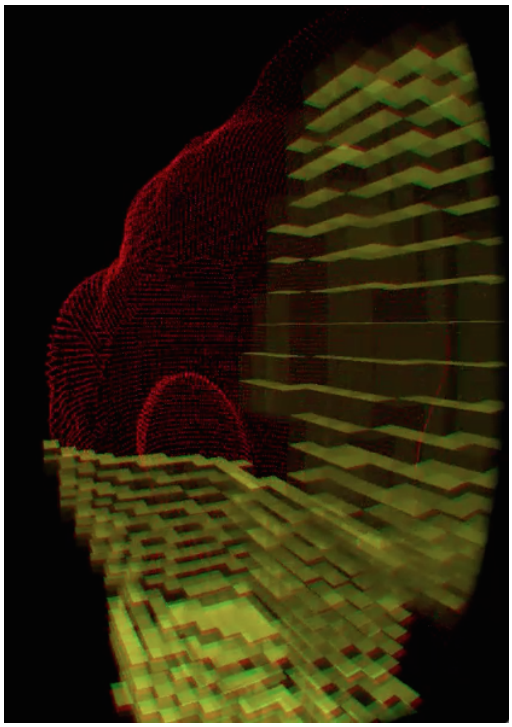

Figure 8. Marching cubes applied to the "Stanford Bunny."

Given $V$, we want to extract the surface where $V = 0$. Marching cubes is one common approach. We partition the space of points into cubes; this can be taken directly from the leaves of the octree storing the point cloud. Then, for each cuboid, we evaluate the function at each vertex. If they are all the same sign, we conclude that node is fully inside or outside the surface. Otherwise, the cube has some edge with positive and negative endpoints. We find the intermediate point along that edge where the function would be zero, assuming it were linear. We connect all such points of the cube/tetrahedron to form triangular faces. Taking all such faces from all cubes, we get the complete reconstructed surface.[4]

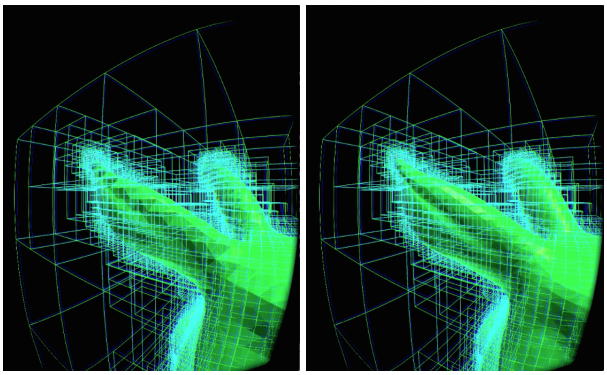

Figure 9. Noticeable refinement in surface reconstruction at the "Standford Bunny" ear at octree depths of 6 (Left) and 8 (Right).

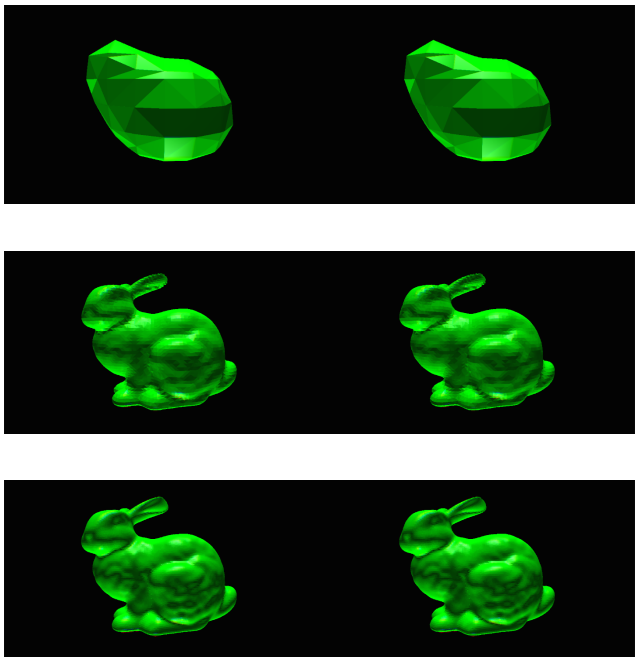

Figure 10. Stereoscopic view of marching cubes surface reconstruction with octree depths of 3, 6 and 8 respectively.

## 5. Discussion

Our visualization of a multi-view stereo reconstruction pipeline can be extended to include more algorithms at each step, allowing comparison between them. Additional steps, such as filtering and smoothing, can be added. The techniques shown here can also be applied to other types of compute vision pipelines, such as visual SLAM.[6]

In addition to passively viewing the output of each step

in the pipeline, humans could actively assist the algorithms in certain steps of the pipeline. For example, a human could provide a coarse estimate for surface reconstruction, which could then be incrementally improved by an algorithm.

## References


[1] Meshlab. http://www.3d-coform.eu/index.php/tools/meshlab.

[2] Unity3d - game engine. http://www.unity3d.com. 1

[3] P. H. R. K. G. L. S. M. E. V. C. D. M. G. A. M. F. S. C. W. 4. Djorgovski, S. G. and C. Lope. The mica experiment: Astrophysics in virtual worlds. 29 Jan. 2012 - 9 May 2014. 1

[4] P. Bourke. Polygonising a scalar field, 1994. Accessed: 12-6-2014. 4

[5] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007. 2

[6] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014. 2, 4

[7] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 2

[8] S. Fuhrmann and M. Goesele. Fusion of depth maps with multiple scales. In *ACM Transactions on Graphics (TOG)*, volume 30, page 148. ACM, 2011. 3

[9] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(8):1362–1376, 2010. 3

[10] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328–341, 2008.

[11] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992.

[12] H. Hoppe, T. Derose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proceedings of the 20th annual conference on Computer Graphics and interactive technologies*. ACM, 1993.

[13] M. Kazhdan, B. Matthew, and H. Hugues. Poisson surface reconstruction. In *Geometry Processing, 2006 Eurographics Symposium on*. 3, 4

[14] D. A. Keim. Information visualization and visual data mining. In *Visualization and Computer Graphics, IEE Transactions*, pages 1–8. 2012. 1

[15] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000.

[16] S. C. G. Laboratory. Stanford 3d scanning repository. http://graphics.stanford.edu/data/3Dscanrep/. Accessed: 12-8-2014.

[17] M. I. Lourakis and A. A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):2, 2009. 2

[18] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[19] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 519–528. IEEE, 2006. 2

[20] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. *ACM transactions on graphics (TOG)*, 25(3):835–846, 2006. 2

[21] R. Szeliski. *Computer vision: algorithms and applications*. Springer, 2010.

[22] R. Tang, S. Halim, and M. Zulkepli. Surface reconstruction algorithms: Review and comparison. In *Proceedings of the 20th annual conference on Computer Graphics and interactive technologies*. ISDE, 2013. 4