***Appendix***

**Supplemental Information 1: Underwater Video Footage Collection**

Videos were collected by a SCUBA diver who swam 1-4 m above the patch reef in a boustrophedonic (i.e., lawnmower) pattern with cameras angled towards nadir followed by a second pass with cameras angled at approximately 45 degrees to obtain oblique views. Finally, the diver swam freely at various depths and distances from the patch reef, completely encircling it in an attempt to acquire footage of any occluded areas on the reef. Videos were recorded using two GoPro Hero 7 Blacks in 4K HD (2160 pixels x 3840 pixels) and at 24 frames per second (fps) in wide field-of-view mode with *HyperSmooth* stabilization set to active. Though we used a two-system camera, this is not necessary for performing SfM photogrammetry using Metashape; this setup simply allowed us to acquire data more quickly, and with ample horizontal overlap (> 80%). Twenty-three coded targets were strategically placed on and around the site to serve as ground control points and assist in estimating camera locations and the calibration coefficients during the photo alignment phase. Coded targets were obtained from Metashape Professional (Version 1.6) and were automatically detected within each image prior to the photo alignment phase using the ‘Detect Markers’ tool. Targets were printed on Xerox® NeverTear paper using a laser printer, and attached to tiles using water-proof duct tape. When using code targets, a good practice to ensure reliable results is to perform a preliminary test by first printing targets of various sizes and taking images of them using the intended survey camera from a distance that is representative of how far the sensor will likely be from the actual scene. Code targets that are consistently detected represent an optimal size for the given survey camera. The survey was conducted in July of 2019, when visibility was greater than 35ft, and used only ambient light. In total, 2180 images were extracted from the video footage by sequentially sampling one in every eight frames, allowing for enough forward overlap (> 60%) between successive images.

**Supplemental Information 2: Structure-from-Motion Photogrammetry**

As Structure-from-Motion photogrammetry (SfM) is used to reconstruct the 3-D model and also plays a crucial role in how semantic labels are assigned, this section serves as an overview of the reconstruction process and is meant to provide context to some of the more important details. SfM uses the fundamental principle of motion parallax to obtain some estimation of depth of an object or a scene captured from multiple overlapping images. By measuring the angle from the multiple viewpoints to the object while also estimating the distance between each viewpoint, the distance to the object can be calculated using basic trigonometry. Although not all SfM algorithms are identical, many use the same general principles that are described below.

*A. Feature Detection*

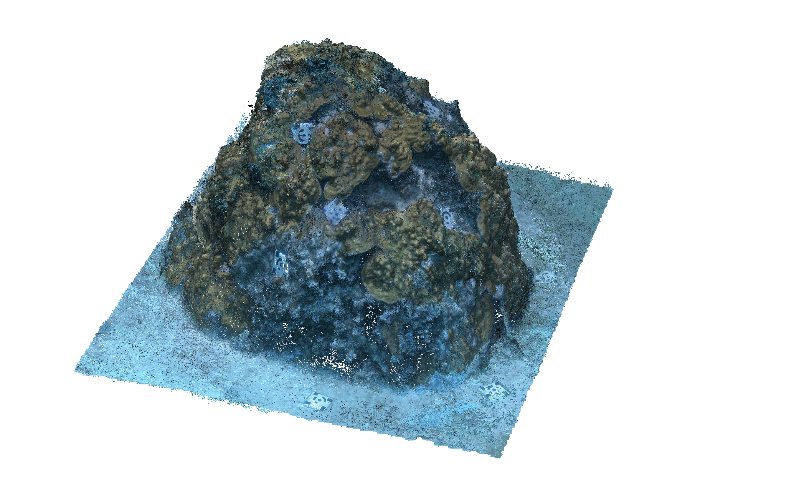
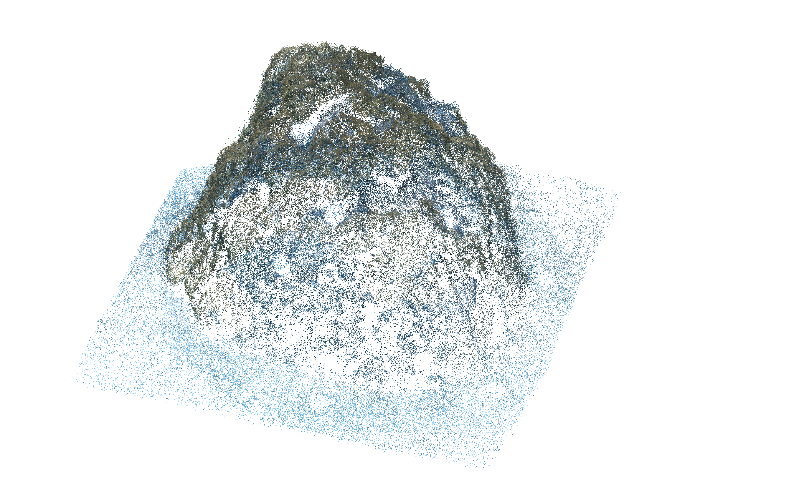
The first step in a typical SfM algorithm is feature extraction, which is used to detect specific parts within the object that can also be found in some of the other images. Key points represent local neighborhoods of pixel groupings in areas of an image with large changes in intensity in all directions (e.g., corners), and ideally are distinct and can be located within other images regardless of changes in scale, rotation and brightness. Once detected, information about those key points including a unique identifier, and their location in image space are stored in a file that is associated with the image that they were found in. Finally, an algorithm is used to match each of those key points with their corresponding points that were also found within other images.

*B. Camera Alignment*

The next step uses the key points to estimate the location of the camera at the time each image was taken. This process is sequential and starts by finding the two images that contain the most co-registered key points. Given the X, Y locations in image space of each key point and by assuming that all viewing rays to the optical sensor of the camera were straight and intersected at the time the image was taken, the Z-location for each key point can be estimated using trigonometry; consequently, this also provides an approximate location of the camera at the time the second image was taken relative to the first. This process is repeated for each additional image, estimating the location of the camera for subsequent images relative to those preceding it. However, due to refraction, and imprecise key point localization and camera calibration techniques, an error accumulates for each additional camera; camera locations are refined with a bundle adjustment algorithm, which uses projection matrices to simultaneously optimize camera and 3-D point locations.

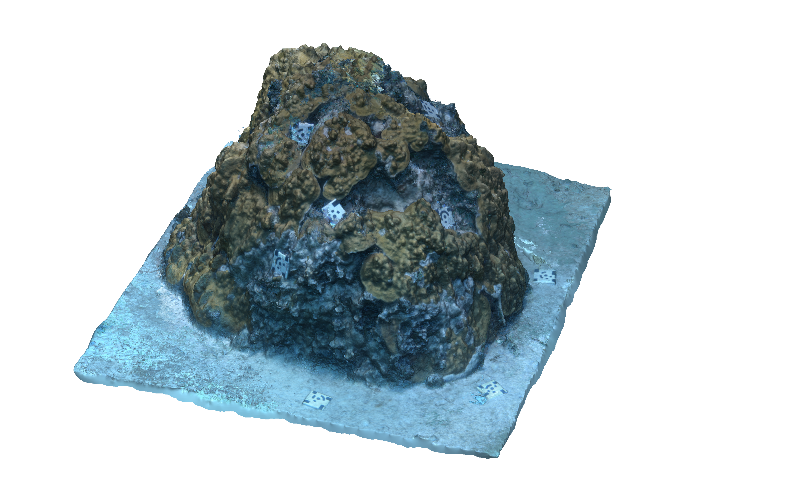
*C. Sparse Point Cloud*

Key points are projected into 3-D space to form a sparse point cloud, which primarily serves as an indication of how well cameras were aligned (Figure 9). Further refinements can be made to the point cloud by removing any points that are considered to be outliers as determined by their reconstruction uncertainty, re-projection error, and projection accuracy.

Figure 9 – The sparse point cloud generated from the 2180 aligned images. Each key point represents a corresponding location found within two or more images (i.e., co-registered), which is then projected into 3-D space. The sparse point cloud serves as an indicator as to how well the images were aligned, but it is not used directly to create the dense point cloud in the following step.

*D. Dense Point Cloud*

This point cloud is then densified by creating depth maps for every pair of images, which determines the location each pixel should be in 3-dimensional space (Figure 10). Each point in this dense cloud is assigned with an X, Y and Z location, as well the color components (i.e., RGB values) averaged from the pixels in the images that it originated from.

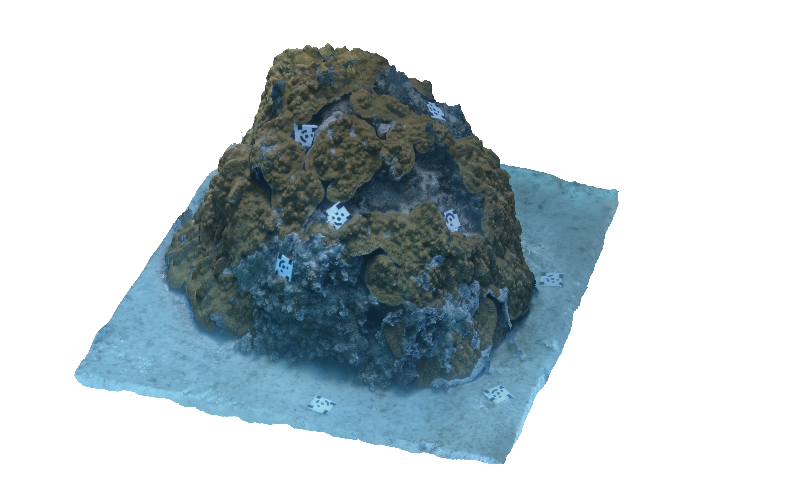
Figure 10 – The dense point cloud generated from the depth maps created as a result of the images being aligned. By estimating the relative location of the camera when each image was taken, trigonometry can be used to create depth maps for pairs of images, thus giving the pixels within each a location in 3-D space.

E. Shaded and Textured Mesh

From this point cloud a triangular mesh is created using the Poisson surface reconstruction algorithm (or some similar variant) to approximate the surface of the object being modeled (Figure 11). This mesh is made up of many elements (vertices, edges, faces, etc.) that also act as data structures storing the associated attributes such as location, color components, normal vectors, light reflectance values, and texture coordinates.

Figure 11 – The shaded mesh created from the dense point cloud. Color is provided to the mesh by using a weighted average of the color component values found in the associated pixel indices to the corresponding elements making up the mesh.

The texture or, “UV” coordinates represent the mesh in two dimensions, which conceptually can be thought of as flattening or unfolding the 3-D model to form a 2-D image. Through UV mapping, portions from the source images that were used in the reconstruction process are mapped onto the 2-D representation of the model creating what is referred to as a ‘texture atlas’; note that these textures are different from the color components and instead consist of groups of pixels in the shape of a triangle that are grafted from the source image onto the atlas. When the texture atlas is applied to the mesh and represented in 3-D it forms a textured mesh and is often used for display purposes (Figure 12); typically, the underlying mesh and the dense point cloud that were created in the previous steps are used for making precise measurements during various spatial analyses.

Figure 12 – The textured mesh created from the images that were used in the reconstruction process. Unlike the point clouds and the shaded mesh, the appearance of the textured mesh does not come directly from the color component values of the pixel indices. Instead, groups of pixels that are thought to best represent an area of the 3-D model are taken from the images and grafted onto it.

**Supplemental Information 3: Reconstruction of 3-D Patch Model:**

The patch reef—as seen in Figure 2—was reconstructed using all of the 2180 still images that were extracted from the video footage. The ‘Camera Calibration’ profile was set to ‘Fisheye lens’ to help account for the refraction caused by the GoPro’s wide-angle lens, and the ‘Detect Markers’ tool was used to automatically create control points for each coded target found within the image, drastically reducing much of the manual work needed; any of the coded targets not detected were marked manually. The rest of the procedure followed the outline set by Metashape and consisted of (1) photo alignment, (2) densification, (3) building a mesh and then (4) texturizing it. All quality settings were set to ‘Medium’ with the exception of photo alignment, which was set to ‘Highest’ resulting in 95% of images being aligned. When creating the textured mesh, the ‘Blending Mode’ within the ‘Build Texture’ tool panel was set to ‘mosaic’ as is recommended by Agisoft so that any available graphic processing units (GPUs) can be utilized. When determining the color component value for each element within the mesh, the ‘Build Texture’ tool first calculates a weighted-average by using all of the relevant images that are available for the low frequency component, and uses just a single image with the highest relative resolution as the high frequent component (Metashape). Therefore, to further enhance model results it is recommended to first disable low resolution images from a project by using the ‘Detect Image Quality’ tool and filtering images that fall below a user-defined threshold. The final model consisted of roughly 10 million triangular faces that approximated the surface of the patch reef, and was estimated to have a ground resolution of 0.278 mm/pixel and a reprojection error (i.e., root-mean square error) equal to 1.6 pixels. Absolute scale was provided to the model in Metashape by creating scale bars along the length and width of seven coded targets found within the model, and supplying them with the corresponding real-world dimensions (10 cm x 10 cm); the estimated accumulative error was reported to be approximately 1.4 mm.

**Supplemental Information 4: Fully Convolutional Network (FCN) Model Training**

Each of the EfficientNet encoders was initialized with ‘Noisy-Student’ weights, but was left frozen (i.e., immutable) for the entire training process, meaning only the weights within the decoder of the FCN were updated. Images were pre-processed in the same way as the images were when the original encoders were trained on the ImageNet dataset, whereas dense labels were converted into one-hot-encoded vectors forming a shape of (B x H x W x C) where B and C represent the batch size and the number of class categories, respectively. During preliminary analysis we found that heavier augmentation techniques (e.g., adding noise, blurring, sharpening, altering contrast) resulted in lower classification accuracies; instead, only augmentations in the form of simple affine transformations (flips, flops, rotations) were applied to each sample.

Each successive encoder within the EfficientNet family required an additional amount of memory to train due to their increasing architectural size and number of parameters. To accommodate the memory requirements of each of the encoders, all images were reduced in height and width by a factor of three resulting in dimensions of 736 pixels x 1280 pixels; this was the largest an image could be to work with all of the encoders, and consequently resulted in the training batch size having to be equal to one (i.e., a single image). Models were trained for 25 epochs with the 2180 images randomly split into a training (90%) and validation set (10%). The training set contains images the model learned from, whereas the validation set was used to ensure that the model did not over-fit to the training set. The models and the results from Fast-MSS were then evaluated using the 50 ground-truth dense labels that were manually created using LabelBox (see Table 3); these scores represent how accurate the models trained on Fast-MSS labels were compared to the ground-truth.

**Supplemental Information 5: Custom ‘Classify Vertices’ Tool**

Version 1.6 of Metashape did not offer a ‘Classify Vertices’ tool to correct the color component values for each vertex of the shaded mesh, therefore this study used a custom script written in Python that performed this task outside of Metashape, demonstrating that it could be done if needed. After the mesh was colorized using the ‘Colorize Vertices’ tool, it was exported as an OBJ file in ASCII format that stored the 3-dimensional coordinates of each vertex (X, Y, Z) and its color components (R, G, B) in an easily parsable format. When provided with the set of discrete color component values that denote each of the class categories used in the study, the script first checked if each vertex had one of the correct color components; if the values were not within the set, they were changed to the color components to which they are closest to in RGB color space as measured by their Euclidean distance. The script parsed the file line-by-line, therefore even large models could be corrected this way without having memory allocation errors. This same script could also be used to adjust each of the pixel indices in the classified textured atlas if the blending mode of the ‘Build Texture’ tool had been set to either ‘mosaic’ or ‘average’ instead of being disabled.