Proceedings of the 6th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'20)
~~Prague, Czech Republic~~ Virtual Conference – August, 2020
Paper No. CIST 107
DOI: 10.11159/cist20.107

Optimized 3D Segmentation Algorithm for Shelly Sand Images

Antonio Leonti[1], Joana Fonseca[2], Iren Valova[1], Ryan Beemer[3], Devin Cannistraro[1], Cynthia Pilskaln[4], Dylan DeFlorio[1], Grayson Kelly[1]
[1]Computer and Information Science Department University of Massachusetts Dartmouth
[3]Civil and Environmental Engineering, University of Massachusetts Dartmouth
[4]School for Marine Science and Technology, University of Massachusetts Dartmouth
285 Old Westport Rd, MA, USA
aleonti@umassd.edu; ivalova@umassd.edu; rbeemer@umassd.edu; dcannistraro@umassd.edu; cpilskaln@umassd.edu; ddeflorio@umassd.edu; gkelly1@umassd.edu
[2]Department of Civil Engineering, City University of London,
Northampton Square, London, UK
joana.fonseca.1@city.ac.uk

Abstract **–**There is much to be gained from analysing and studying calcareous sediment, with applications ranging from the study of climate change, rock dating, and even building offshore oil rigs and wind farms. One way of performing this analysis is to obtain a µCT scan of the sediment, allowing scientists and engineers to automate much of their analysis using software. Many existing and prospective analysis techniques require handling individual grains. Thus, fast and effective segmentation is an essential first step for any such analysis. Segmentation is non-trivial; these scans hold a lot of information, exhibit ambiguous boundaries between objects, and many objects are hollow, making it even more difficult to apply traditional watershed segmentation. Addressing these issues, in this paper we propose an optimized 3D segmentation (O3DS) algorithm based on watersheds. We make use of branch recursion, partition the image by height prior to segmentation, artificially reducing the size of the largest connected objects. These and additional changes are extremely effective in optimizing performance; O3DS reduces the time to segment a 659x925x932 scan of sediment by 95.4% and produces better or comparable results when compared to similar implementation by our co-author.

Keywords**:** Recursion, Watershed segmentation, 3D imaging, Microtomography, Calcareous sand

1. Introduction

Segmentation of 3D images of sediment, in particular, is difficult for a few reasons. One is the hollow nature of many shells, which can cause over-segmentation with traditional watershed algorithms [1]. Another is the interconnectedness of the objects in the scan (the original image has a meagre 1.9% the number of connected components in the segmented version). Lastly, since 3D images are quite large, the time and space complexity of the segmentation algorithm used can make or break its usefulness.

We start with a 3D microtomographic image of biogenic ("arising from life"—shells) sand. The 3D scan consists of 659 slices each with an aspect ratio of 925x932. The images can be used as standalone 2D images or "stacked" in memory to form a 3D image. We use the 3D representation in this work, as applying operations to the entire scan at once is faster than processing each image one at a time, and there is a lot of information lost if you ignore the 3D nature of the scan, such as pixel connections from one image to another.

Such scans have the potential to save micropaleontologists a considerable amount of work. For example, once the image is segmented, scientists and engineers can automatically calculate various properties of the grains. The individual grains can potentially be isolated/classified by a neural network [2]. Other methods of analysis not possible or feasible to do by hand can now be automated with relative ease using MatLab and similar software. Whatever the goal, most analysis techniques require viewing and analysing individual grains, which means the first step will be the segmentation of the scan.

Finally, to appreciate the size of these scans and using our own scan as an example, its dimensions are 659x925x932, which gives a whopping 568,123,900 (half a *billion*) pixels! It would be silly to assume future scans will not be even larger.

Let us explore some of the concepts of O3DS before diving into the methodology of the proposed approach.

1.1. Principles of binary image segmentation (pixel partitioning)

Image segmentation is the act of partitioning an image into sets of pixels. The most common form of binary image segmentation is simply creating sets of connected pixels. We call these sets "connected components" or "regions." This works well for images in which the subjects are cleanly separated from each other, such as Figure 1a.

(a) (b)

Fig .1: (a) Two disjoint circles which are easy to segment; (b) The same two circles, but one pixel closer to each other. They form one region (indicated by the color), despite being easily discerned by the eye.

For images with ambiguous boundaries (contours) between the subjects, such as the circles in Figure 1b, it does not perform as well. The reason is the algorithm assumes the existence of clear boundaries between the desired segments in the image. To partition an image lacking these boundaries, it is necessary to first carve them in. Watershed segmentation is the de facto approach to inserting these contours [3, 4, 5, 6].

1.2. Watershed segmentation

Watershed segmentation is a contouring method first introduced in [6]. It serves as a pre-processing step for conventional binary image segmentation techniques. The central idea of watershed segmentation is that a raindrop falling on a surface will take the path of steepest descent to a local minimum. Viewing the distance transformation of the target image as a *topography* map, a set of points leading to the same minimum is defined as a catchment basin. The watershed transform creates a binary mask of the original image containing only these catchment basins. The ridges separating the basins, or "drainage divides," are not included in the mask. These are the boundaries to be imposed between objects. While the analogy does not work for 3D images, we can still apply the abstract concept of using the inverted Euclidean distance map (IEDM) to create a segmentation mask.

Fig. 2: The original scan rendered in 3D.

Watershed segmentation of a grayscale image requires the binarization of the image, creating an IEDM, and finally applying the watershed transform. In addition to the boilerplate pre-processing steps, we use 3D filling, partitioning, and an IEDM modification step (H-minima transform [7]) to make the watershed transform more apt for this dataset and similar ones.

1.3. Scan origin

This work utilizes X-ray μtomography scan of the Browse #1 hemipelagic calcareous sand . The sample was collected, via box core, in the Browse Basin, offshore North West Australia, on October of 2017 by the CSIRO RV Investigator [8]. This biogenic sand is 89.75% calcium carbonate by mass and consists primarily of foraminifera tests, gastropods, pteropods, and their bioclasts. By the Unified Soil Classification System Browse #1 is a SP (poorly graded sand) and has a D_{50} of 390 μm [9]. The X-ray microtomography scans were conducted at the Centre for Microscopy, Characterisation & Analysis, The University of Western Australia with a Versa 520 XRM, Zeiss. An example of a 3D scan is presented in Figure 2.

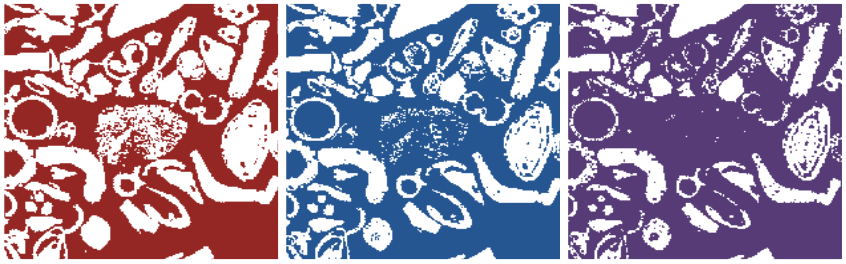
2. Methods and materials

Watershed segmentation involves binarizing the image, creating an IEDM, and performing the watershed transform on the IEDM. Using the watershed result, we can mask the original image's values, and the partitioning algorithm finally will have the desired effect. To make the watershed segmentation more effective for these complex 3D images, we have introduced the pre-processing steps of filling and temporarily partitioning the image before beginning the segmentation process. As a cornerstone feature of the proposed O3DS, we implement a tail- recursive control structure to the watershed algorithm which addresses the inherently hierarchical nature of splitting objects into an indefinite number of smaller objects.

2.1. Preparing the scan for watershed segmentation

The first step in preparing for watershed segmentation is to approximate the image via binarization. Each pixel is replaced with a 0 or 1 depending on whether it is less than or greater than or equal to a user-defined threshold. We use Otsu's [10] method to find a threshold value. It algorithmically finds the binarization threshold which minimizes the inter-class intensity variance (the classes being background (0) and foreground (1)).

Raising the binarization threshold to make the image easier to segment is intuitively an attractive possibility. The central assumption here is that the edges of things are *darker* (closer to zero) than their centres. If this were the case, raising the threshold would, in fact, be the easiest way of inserting the boundaries between objects. For this dataset, however, this is not the case—the edges are not reliably darker than the rest of the object. Raising the threshold *does* corrode some edges; it also dissolves some objects in undesirable ways, sometimes from the centre out. In extreme cases, raising the threshold destroys entire objects. Since we cannot recover data lost during binarization, raising the threshold too high could be detrimental to the segmented result. Figure 3 shows the effects of binarizing an image using progressively higher thresholds. An increased binarization threshold is clearly not a reliable method of increasing the space between objects. Some objects are untouched. Others become unrecognizable. The coral in the centre is annihilated.



(a) (b) (c)

Fig. 3: Slice binarized using (a) a 0%; (b) 10%; (c), and 20% threshold, respectively.

Many of the objects in these scans are hollow. The IEDM is most useful if we can approximate the centres of shapes via their *local minima*. This is not possible if the regions are full of holes. Therefore, we must fill the internal voids. The conventional way to do this is flood filling from a point which is known to be external void. The internal voids will be unaffected by this, so the internal-filled image F can be found from the binary image B and the external-filled image E as in (Eq.1):

$$F = B \cup E^c \tag{1}$$

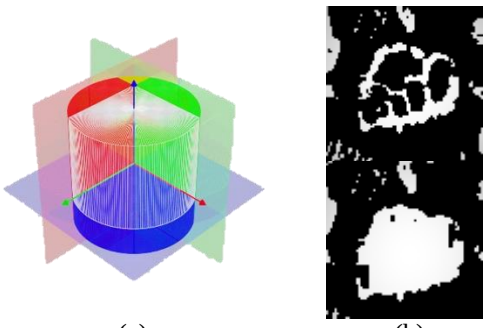
(where E^c is the complement of E).

As it turns out, filling these scans is not so simple. Many regions are hollow *without* a cleanly enveloped internal void (be it from breakage or intrinsic to the type of shell). The above procedure is ineffective on such regions, as the internal void will be filled by the external fill, removing the internal void from E^c.

The solution for this problem presented in [11] is to fill the scan via a plane passing through the image in each orthogonal direction. With each step of the plane, its intersection with the data is filled using the described flood fill and

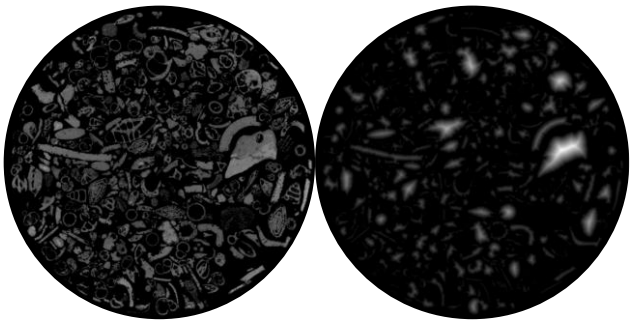
shown in Figure 4a. After this, the entire 3D image is filled at once using the same flood-fill method. This layered approach allows us to fill “leaky” internal voids, so long as the shape is enclosed in at least one orthogonal direction. method of filling does introduce some undesirable straight lines in some corner cases (such as multiple shells forming closed shape when viewed from a specific direction) as shown in Figure 4b, but we remove the pixels added by the procedure after segmentation anyway.

After filling, the 3D image is partitioned. Some of the connected components are quite large, consisting of thousands of would-be segments spanning the entire volume of the 3D image. Partitioning the 3D image, however, artificially decreases their size, allowing the large regions to be segmented like any other.



(a) (b)

Fig. 4: (a) Visualization of the described filling method . The cylinder represents the 3D image. The planes, the direction of their movement through the image, and the portions already filled are shown in colors corresponding to each orthogonal direction; (b) shows the same shell before and after the filling is applied.



(a) (b)

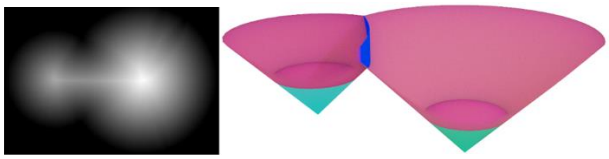
Fig. 5: Comparison between (a) a source image; and (b) its distance map.

With the filled binary image created and partitioned, we create an IEDM. A Euclidian distance map (or simply distance map) is a transformation which replaces all 1s with their respective distance to the nearest 0. An example of a distance map can be found in Figure 5b based on the source image in Figure 5a. Inverting the distance map means to multiply all its pixel values by -1. Note the distance map acts as a gradient from the centres of objects to their edges.

At this point, it becomes convenient to visualize the 2D distance map in 3D. The first two dimensions are the ones intrinsic to the matrix in which the image is stored (the location of each element relative to (0,0)). The third dimension is the scalar stored in each cell. In this manner, the image can be visualized like a topographical map.

In hydrology, a catchment basin is an area where precipitation gathers to a single body of water, such as a lake or river. Basins are analogous to the 3D shapes in the IEDM, in that if it were to metaphorically rain onto the IEDM, the water would collect in the minima where the slope of the topographic map leads to. Each minimum thus has an area of influence within which it collects all metaphorical precipitation. Where these areas of influence meet (“drainage divides”) are the contours which watershed segmentation introduces to the image.

Since watershed segmentation adds contours between each basin's area of influence, applying it to a raw IEDM will segment the image by its simple shapes. This is because all local minima have an area of influence, no matter how small. The things our proposed O3DS attempts to segment often consist of multiple shapes, however, so applying watershed segmentation directly to the IEDM will cause severe over-segmentation.



(a) (b)

Fig. 6: (a) Distance map of two touching circles; (b) 3D visualization of the inverted distance map. The plane inside each cone is an example of clipping from the bring-up [11] method (underneath, the original shape is shown in green). In blue, the drainage divide's projection onto the xy axis is shown. This projection will be the boundary introduced by the watershed transform.

Therefore, the final step in preparing the image for watershed segmentation is to modify the IEDM in a way that combines the basins of each thing. This is done via the bring-up [11] method. This method entails clipping (like in signal processing) each catchment basin by an amount ΔH (the "bring up amount"). One way of imagining it is filling the basin with concrete until the concrete is ΔH pixels deep or the basin is full, whichever comes first. The process is visualized in Figure 6b as it corresponds to the two touching circles in Figure 6a. We calculate ΔH by finding the reference depth (conventionally the deepest basin, H_{min}) and multiplying this reference depth by a user-defined constant $-k$.

$$H = \{\text{all basin depths sorted GTL}\} \tag{2}$$

$$R = H_{\text{min}} \tag{3}$$

$$\Delta H = -k * R \tag{4}$$

The method in [11] suggests using a different reference depth by determining the adjacent basins in H with the greatest difference. The larger of the two is the reference depth, and we calculate ΔH the same as before.

$$I = \text{argmax}_i \, |H_{i-1} - H_i| \tag{5}$$

$$R = H_I \tag{6}$$

$$\Delta H = -k * R \tag{7}$$

All basin depths are clipped to R. These modifications should allow *under*-segmentation. In this iterative manner, each region is segmented, one by one, into multiple regions, which are subsequently segmented into subregions, etc., until they no longer need to be segmented. Our proposed O3DS algorithm takes advantage of this natural recursive procedure to optimize performance and save time and memory at no expense to final result quality.

2.2. Recursive / branching mechanism of O3DS

The idea of recursive segmentation is quite attractive for this application and the proposed O3DS algorithm takes advantage as an optimization approach. Some regions require very little segmentation, while others need to be segmented thousands of times (it is not at all uncommon for the largest connected components to span all the slices of the entire 3D image). It would be impossible to find a value for k that fits the bill for every region.

To address this problem, when O3DS passes the segmentation function an image, it first partitions the image into its connected components. It then iterates through each region, creating an IEDM, modifying the IEDM using the bring-up [11] method, and finally performs watershed segmentation on the result of that.

If the watershed transform introduces a new contour to the region of interest, this means the region has split into at least two shapes. Thus, before moving onto the next region, the function calls itself, passing only the current region. The entire process is performed again on the sub-regions and sub-sub-regions until a call fails to introduce any new contours, which represents the *termination case* in our recursion.

```
Image segment(Image I){
    partition pixels & create label matrix from I

    for(regions in label matrix){
        IEDM = inverted Euclidean distance map of region
        make list of minima in IEDM

        if(# of minima >= 2){
            modify IEDM via bring-up method

            S = watershed(IEDM)

            if(there are new contours in S){
                // additional contours from recursion are brought
back here
                S = segment(S)}
            copy new contours to I}}
    return I}
```

Fig. 7: Pseudocode implementation of the recursive segmentation function.

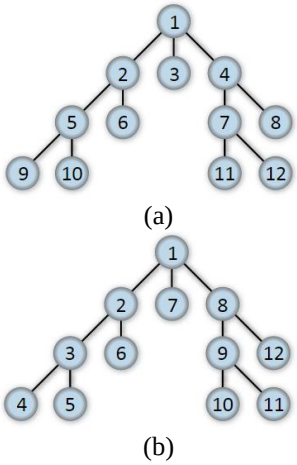


Fig. 8: (a) Processing order in breadth-first segmentation. (b) Processing order in depth-first segmentation.

Upon reaching the termination case, the call stack collapses into the next subroutine waiting to segment more regions. The region of interest is replaced with the segmented version, and the active subroutine carries on segmenting the next region. The pseudocode of O3DS is presented in Figure 7.

By using branch recursion and a depth-first control structure, the parent routine (the user's call to the segmentation function) looks at each region once, segments it to completion, and only then begins segmenting the next region. Comparatively, [11]'s implementation uses a breadth-first control structure, segmenting each region once and starting over. The breadth-first approach comes with significantly more overhead, as with each iteration, the image must be partitioned again, the connected components must be labelled, their bounds calculated, etc.; information from the previous iterations cannot be used to speed up the process. Figure 8 demonstrates the sequence in function call processing for both breadth-first and depth-first approach.

Once the algorithm has run on all partitions, the result is a *mask* of the original image. We want both the grayscale values from the source image *and* the boundaries added to the segmentation mask. We do this by taking the Hadamard product (or element-wise product) on the segmentation mask and the source image. Figure 9 shows an example of the Hadamard product of an image and a mask. We can finally partition the result and obtain the desired effect.

Fig. 9 Hadamard product ($I_1 \circ I_2$) of two images. All zeros in the mask are carried over to the result.

3. Results

We performed a timed test of O3DS and the algorithm from [11] on a PC with an NVIDIA GTX 2060 GPU, AMD Ryzen 5 2600 CPU, and 16 GB RAM in a B450 AORUS ELITE board.

The run-time performance comparison in presented in Table 1. The difference in run-time is vast; O3DS takes

only 7.6% of the time to process entire batch scan with dimensions of 659x925x932, or 568,123,900 pixels.

Table 1: Comparison of run times for the proposed optimized approach and the implementation offered in [11].

Algorithm	**Run-time for entire 3D batch scan**
Proposed O3DS	39 minutes and 17 seconds
Implementation from [11] as shared by the co-author	10 hours, 13 minutes, and 34 seconds

Figure 10 shows the different results from each. The figures in the left column are examples of [11] results, while the O3DS results are featured in the right column. The larger images represent a square sample of the 2D slice segmentation with a blown-up portion showing clear examples of over-segmentation (the marron object on the right compared to segmenting it as a group of 5 objects from the left). Likewise, the red stick-like object from the blown-up left column is over-segmented and shown as consisting of three objects in the blown-up right column. Finally, the handcuffs-like object on the bottom row of Figure 10 shows clear over-segmentation on the left versus the O3DS on the right. Therefore, the results from each implementation, while different, should be considered comparable (both results exhibit under and over segmentation in different and various parts).

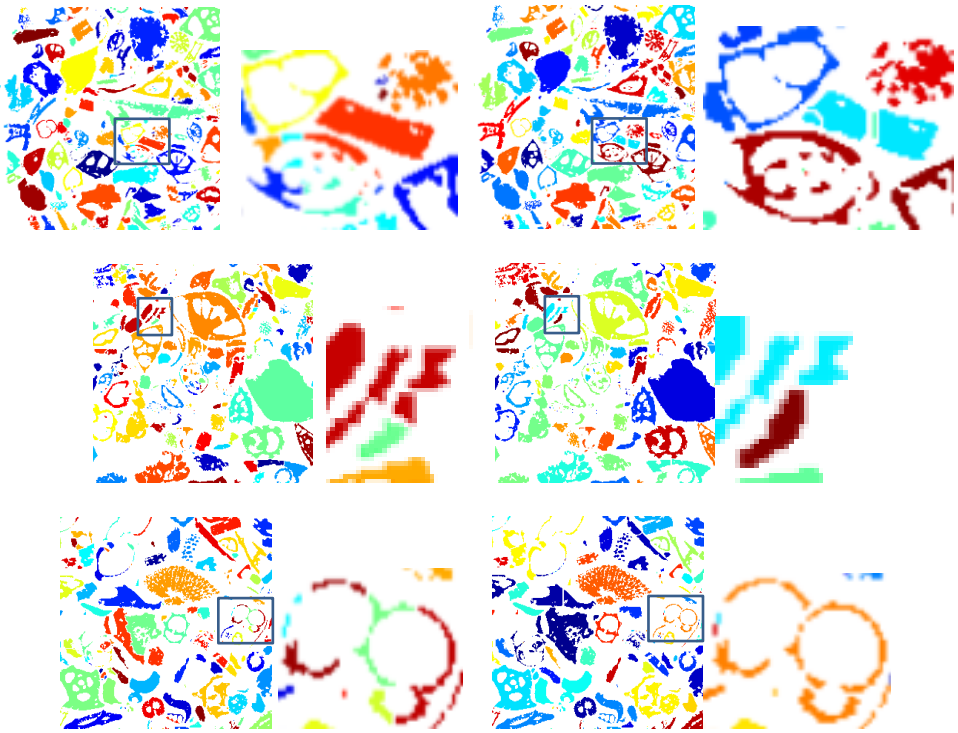


Fig. 10: Comparison of results from [11] (on the left) segmentation algorithm and O3DS (on the right).

4. Discussion and Conclusions

The segmentation of µCT scans opens the door to many automated analysis methods. In this paper we present a valuable improvement of the watershed segmentation algorithm for shelly sand [11]. As the name of the proposed algorithm implies - optimized 3D segmentation, or O3DS – the run-time complexity of this version of watershed-based segmentation has been extremely reduced due to the many improvements and optimization approaches described in the sections of this paper. We consider the employment of tail-recursion method with depth-first approach as one of the significant contributions to the optimization of O3DS. This requires initialization for the entire image once, with future

calls initializing on progressively smaller parts. In addition, the multitude of other contributions, such as a simple height-based partitioning pre-processing step; the calculation of a region's bounding box, as this procedure compounds over thousands of iterations, enables us to produce results of better or comparable quality in a markedly small fraction of the time.

The difficulty in finding an ideal value for k (one which does not cause over or under segmentation) is the optimizations presented in this paper. All results exhibit both under and over segmentation. This is a limitation of this algorithm, and perhaps watershed in general for this application. Improving the precision of the segmentation is important, as it allows for more precise analysis of the individual particles too. For now, the user must decide if over or under segmentation is more desirable for the needed outcome.

Other pre-processing or structural changes are needed to raise the ceiling of segmentation precision in crowded, complex 3D scans such as the one used in this paper. While we vastly improve the speed of this algorithm, future research may be better suited addressing the problem of precise segmentation with size variance, as with both algorithms, it is extremely difficult to produce a result which properly segments all objects.

Acknowledgements

The authors wish to thank the CSIRO Marine National Facility (MNF) for the support personnel, scientific equipment, and samples collected by the RV Investigator on Voyage IN2017_T01. All samples acquired on the voyage are made publicly available in accordance with MNF Policy. Contact: rbeemer@umassd.edu

The authors acknowledge Dr. Jeremy Shaw for the X-ray Microtomography scans in Figures 2, 3, 4, 5, and 10 and the facilities, scientific, and technical assistance of Microscopy Australia at the Centre for Microscopy, Characterisation & Analysis, The University of Western Australia, a facility funded by the University, State, and Commonwealth Governments.

References

[1] I. Arganda-Carreras and D. Legland, Modeling and Digital Imaging Group, Institut Jean-Pierre Bourgin, INRA Versailles, France, https://imagejdocu.tudor.lu/doku.php?id=plugin:segmentation:morphological_segmentation:start.

[2] M. Vania, D. Mureja and D.Lee, "Automatic spine segmentation from CT images using Convolutional Neural Network via redundant generation of class labels," *Journal of Comp. Design and Engineering*, 6:224 – 232, 2019.

[3] W. Wang, "Rock Particle Image Segmentation and Systems", in *Pattern Recognition Techniques, Technology and Applications*, Eds. P.Y. Yin, pp 197 – 226, 2008.

[4] Y. Tarabalka, J. Chanussot and J.A. Benediktsson, "Segmentation and classification of hyperspectral images using watershed transformation," *Pattern Recognition*, Vol. 43:7, pp 2367-2379, 2010.

[5] S. Lou, L. Pagani, W. Zeng, X. Jiang and P.J. Scott, "Watershed segmentation of topographical features on freeform surfaces and its application to additively manufactured surfaces", *Precision Engineering*, 63:177-186, 2020.

[6] S. Beucher and C. Lantuejoul, "Use of watersheds in contour detection". *Proc. Int. Workshop Image Process. CCETT*, Rennes, France, January, pp 2.1 - 2.12, 1979.

[7] N. H. Fauzi ismail, T. R. M. Zaini, M. Jaafar and N. C. Pin, "H-minima transform for segmentation of structured surface," in *MATEC Web of Conferences*, 74, 2016.

[8] H. Barker, A. Bowie, K. Walters, R. Beemer, V. Stavropoulos, B. Arthur, D. Steinberg, and M. Wreck, *Voyage Summary: IN2017_T01*. Hobart, Australia, 2017..

[9] ASTM International, "Standard Practice for Classification of Soils for Engineering Purpose (Unified Soil Classification System)" (Superseded), ASTM D2487–11, West Conshohocken, PA: ASTM International. https://doi.org/10.1520/D2487-11, 2011.

[10] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[11] D. Kong and J. Fonseca, "Quantification of the morphology of shelly carbonate sands using 3D images," *Géotechnique*, vol. 68, no. 3, pp. 249–261, 2018.