Benthia: benthic image analysis software for the sedimentation event sensor

Cordelia Sanborn-Marsh, Stanford University

Mentors: Rich Henthorn

Summer 2014

Keywords: software, sedimentation event sensor, SES, image analysis, benthic, Bayer16, Prosilica, percent coverage, attenuation.

ABSTRACT

Changes in climate have resulted in reduced cycling of nutrients, especially in organic carbon, the primary source of food for deep-sea communities (1). In order to assess the repercussions of climate change on deep-sea ecosystems, benthic instruments, such as the Sedimentation Event Sensor (SES), measure trends in marine snow fall. Marine snow, composed of decaying organisms and fecal matter, sinks down through the water column, enriching the ecosystems below. Studies have equated trends in sedimentation to benthic activity (2,3). To collect data on marine snow is therefore to monitor benthic ecosystem productivity. The program Benthia expedites and standardizes this image analysis of the data generated by the SES. Written in C++ and run in terminal, Benthia will facilitate research both in situ and post-deployment. This program generates two values for each image: the percent coverage and the attenuation of the pixels on the slide. In situ, Benthia directs the SES to adjust the length of the intervals in between pictures, according to the amount of coverage. Post-deployment, graphing the data values over time will reveal insights on trends in ocean activity and potentially the effects of climate change on benthic zone productivity.

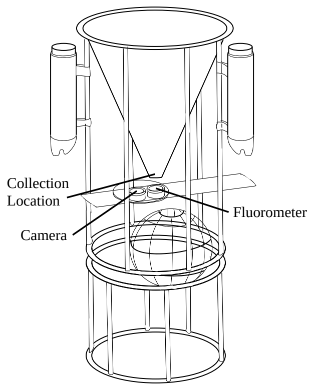


Figure 1: The Sedimentation Event Sensor (SES) proves an efficient and reliable way to collect data on ocean activity. Deployed on the ocean floor, it concentrates marine snow with a collection cone onto a slide. At regular intervals, it takes a picture of the slide before clearing it.

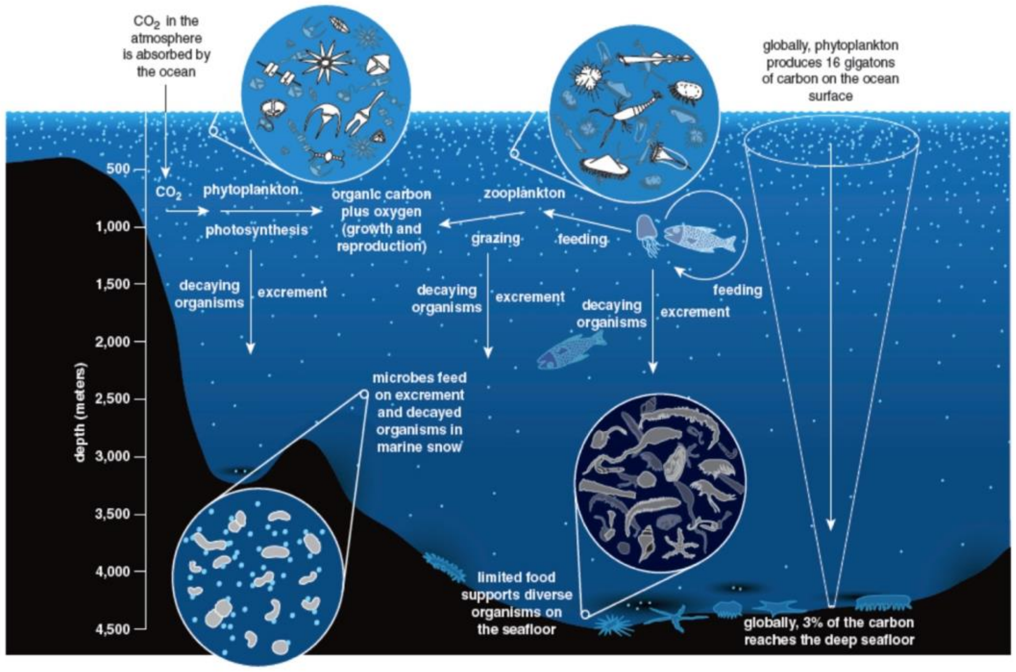
INTRODUCTION

1.1 THE BENTHIC ZONE AND MARINE SNOW

The oceans, which comprise 71% of the Earth's surface, rely on surface zone productivity for the synthesis of organic carbon, the foundation of food webs that stretch down to deep-sea communities. The ocean, with an average depth of 3,700 m, harbors a wide range of communities. Deep-sea regions, located at depths greater than 2,000 m, cover 61% of the Earth's surface. Such deep-sea ecosystems risk starvation should the cycling processes deteriorate between the surface zone and deep-sea communities. Other anthropogenic factors, such as climate change, mineral and hydrocarbon extraction, and commercial fishing, also have the potential to disturb the deep-sea floor, which is covered with soft sediment (1).

Changes in the climate have resulted in warmer upper ocean temperatures and a reduction in vertical cycling of nutrients. Deep-sea communities rely on the energy provided by the sediment from the trophic zone: a portion of the organic matter, yielded through photosynthesis at the surface, sinks to the sea floor. The productivity of the food-barren benthic zones is therefore limited to the activity and mixing of these nutrients (1). Studies have shown a direct correlation between trophic zone activity and the deep-sea ecosystems below (2,3). More precisely, respiration, used as an approximation for organic carbon consumption, increases as the amount of nutrients falling to the deep-sea floor rises (4,5).

Studies are now examining the effects of climate change on deep-sea community productivity. The austerity of the conditions there, such as the high hydrostatic pressure and the corrosive nature of the ocean water, constrains research to the capabilities of technology. More challenging yet, sampling the deep sea over long periods of time poses additional issues. Due to the long-term effects of climate change, long time-series sampling is essential (1). The Monterey Bay Aquarium Research Institute (MBARI) has refined such technologies, including the Sedimentation Event Sensor (SES).



CO_2 in the atmosphere is absorbed by the ocean
globally, phytoplankton produces 16 gigatons of carbon on the ocean surface
CO_2
phytoplankton
photosynthesis
organic carbon plus oxygen (growth and reproduction)
zooplankton
grazing
feeding
feeding
decaying organisms
excrement
decaying organisms
excrement
decaying organisms
excrement
microbes feed on excrement and decayed organisms in marine snow
limited food supports diverse organisms on the seafloor
globally, 3% of the carbon reaches the deep seafloor
depth (meters)
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500

Figure 2: Marine cycling of nutrients plays a critical role in providing food for the benthic zone. Trophic zone organisms and their fecal matter decay and sink to the bottom of the ocean, enriching benthic communities (6).

1.2 TECHNOLOGIES: THE SEDIMETNATION EVENT SENSOR

The SES, which originated at the SCRIPPS Research Institute, has since been redeveloped and updated at MBARI. The SES differs from standard sedimentation collectors, which collect sedimentation data samples in bottles. Given the finite number of collection bottles a sedimentation collector can carry, this approach limits the length of the deployment and therefore the amount of data collectable. Additionally, by funneling marine snow into a bottle, the sedimentation collectors increase the likelihood of mixing the marine snow in ways that alter the stratification and skew the data.

The SES addresses these design flaws by instead collecting marine snow on a slide. It uses the same collection cone as a sedimentation collector to concentrate sediment. The SES regularly captures images of the sediment deposited on the SES slide with a Prosilica camera and then clears the slide. The process is facilitated by a rotating platform into which the slide is set. The rotating platform stops the slide underneath the collection cone, underneath the camera, and then cleans the slide by running it through the brushes. By taking pictures instead of collecting samples, the SES generates more data over a longer period of time.

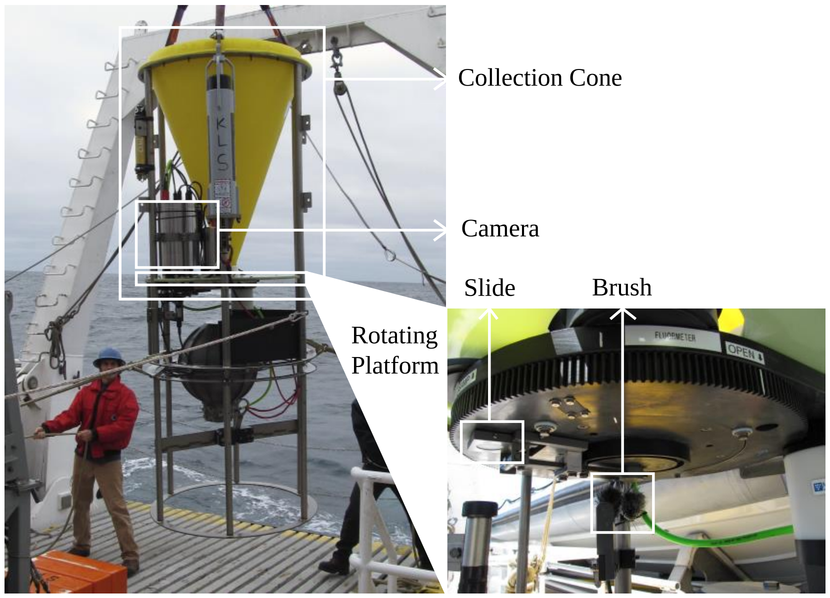


Figure 3: The structure of the SES differs from that of other sedimentation collectors, improving the efficiency and accuracy of data collection. The rotating platform stops the slide underneath the collection cone, underneath the camera, and then cleans the slide by running it through the brushes.

This approach provides a more accurate and precise method of monitoring deep-sea activity. Each deployment, lasting several months, generates hundreds of images, all of which before had to be analyzed visually by humans. Not only a time-consuming process, this method risks human error. Benthia eliminates both of these setbacks, inherent to the original data analysis method.

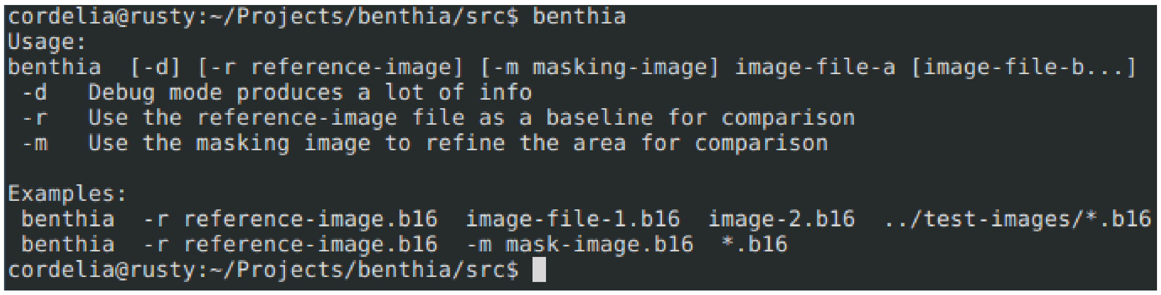
Sedimentation Event Sensor Deployments: Pulses 60 – 62

Pulse Name	Pulse Start Date	Pulse End Date	Duration of Pulse (months)	Collection Time (hours)	Frequency of Collections per Day	Number of Images Collected
Pulse 60	Jun 2012	Nov 2012	5	12	2x	304
Pulse 61	Nov 2012	Mar 2013	5	3	8x	
Pulse 62	Jun 2013	Feb 2014	8	2	12x	1,420

Table 1: Each deployment by the Ken Smith lab at MBARI is numbered as a Pulse, The SES's first deployment took place during Pulse 60. This instrument has been deployed four times for varying durations and took pictures of data at different intervals.

1.3 IMAGE ANALYSIS SOFTWARE: BENTHIA

Benthia is written in C++ for seamless integration into the SES software, an instrument run by code written in Ruby. Benthia was created on a Linux platform in order to eliminate dependencies. The program, intended to be run on terminal in order to reduce graphical interfaces, aims to simplify the image analysis process. A comprehensive troubleshooting menu is available to the user at every step of the program. Refer to the Appendix for the complete Benthia code.



```
cordelia@rusty:~/Projects/benthia/src$ benthia
Usage:
benthia  [-d] [-r reference-image] [-m masking-image] image-file-a [image-file-b...]
 -d   Debug mode produces a lot of info
 -r   Use the reference-image file as a baseline for comparison
 -m   Use the masking image to refine the area for comparison

Examples:
 benthia  -r reference-image.b16  image-file-1.b16  image-2.b16  ../test-images/*.b16
 benthia  -r reference-image.b16  -m mask-image.b16  *.b16
cordelia@rusty:~/Projects/benthia/src$
```

Figure 4: Benthia accomplishes its goal of simplifying image analysis by running on terminal, which minimizes the graphical interface.

MATERIALS AND METHODS

2.1 DATA TYPE

The Prosilica camera on the SES takes pictures in a Bayer16 format. Bayer16 files are a raw data type that has not been compressed in order to preserve information. Benthia exclusively accepts Bayer16 format files. Benthia first distinguishes between Bayer16 and non-Bayer16 data files. Next, the program parses out unimportant information.

Paul McGill provided insight on how to check for the correct file type. Each Bayer16 image contains 5,018,522 2-byte words. Paul McGill reasoned that the product of the dimensions of a Bayer16 image—2,448 x 2050 pixels—makes up the data portion of each Bayer16 image. The number of 2-byte words in the header is the difference between the number of total 2-byte words and those in the data portion of the file (5,018,522 - 5,018,400 = 122). The First 122 2-byte words do not contain the intensities of the pixels in the image. Unfortunately, the makers of Prosilica have not made public how to interpret the values in the header. The header potentially provides information about the dimensions of the image and shutter speed, for example.

Benthia first checks for valid file type by counting the number of 2-byte words in the file. If that number is 5,018,522, the program next parses out the header, or first 122 2-byte words. Then it creates an array from the remaining 5,018,400 2-byte words, or pixel values. The relevant pixel values are later used in calculating data values that describe the composition of the matter on the slide.

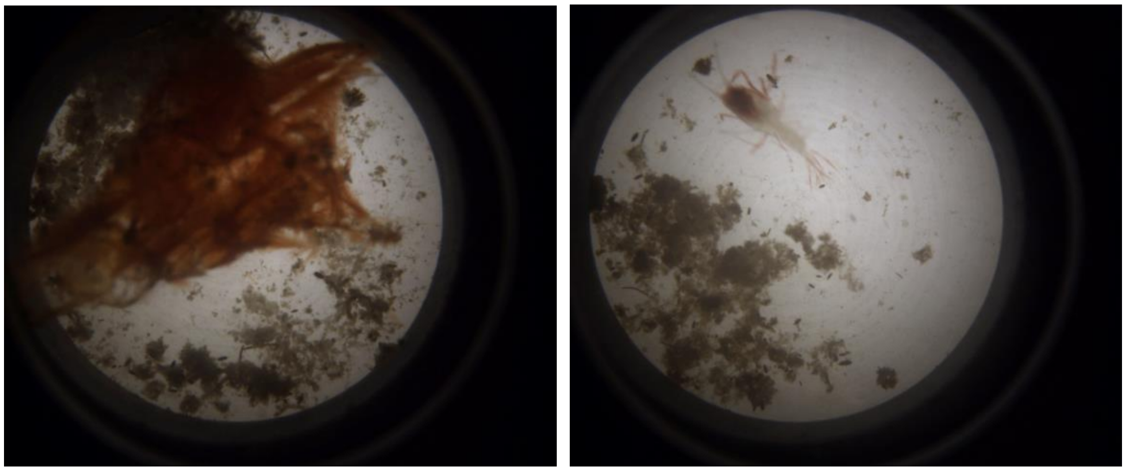


Figure 5: Two examples of data images taken by the SES: on the left, a squid beak (Pulse 61: 04/16/13), and on the right, an arthropod (Pulse 61: 01/01/13).

2.2 OVERVIEW: COMPARING IMAGES

Benthia operates by comparing a blank, reference slide to every other data slide in a dataset. The program cycles through each pixel of the reference slide and its corresponding pixel in the data slide, comparing the values of the pixels. The values of the pixels represent the intensities of the pixels at each specific point in the image.

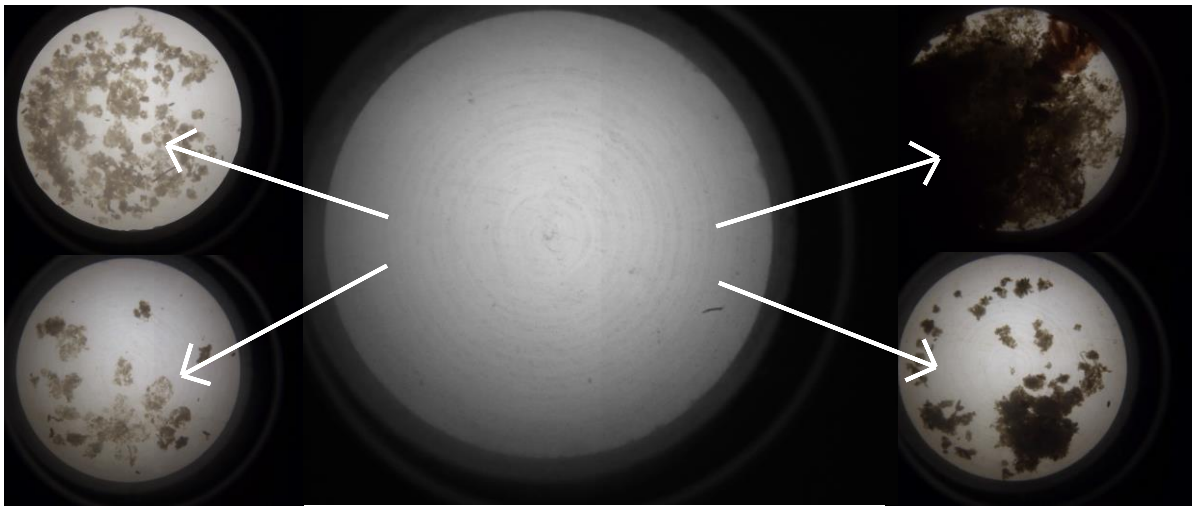


Figure 6: Benthia works by comparing each data file to a blank, reference slide, pixel by corresponding pixel. The data files shown here were generated during Pulse 61 (top left: 01/29/13, bottom left: 12/23/12, top right: 04/08/13, bottom right: 04/07/13).

2.3 MASKING: BINARIZATION AND THRESHOLDS

Only parts of the images should be compared to one another. More specifically, Benthia should only cycle through the circular data slide area. The light emitted through the sediment on the slide could affect calculations if the area outside the slide is included in the comparisons. Depending on how much matter falls on the slide, more or less light pervades the darkness of the region outside of the slide.

In order to mask out the area outside the slide, the reference file is first grayscaled and binarized. As a result, all the pixels counted as part of the slide have a value of 1. All the other pixels outside the slide have a value of 0. Benthia cycles through the entire data file, but only compares pixels at the points where the mask pixels have a value of 1.

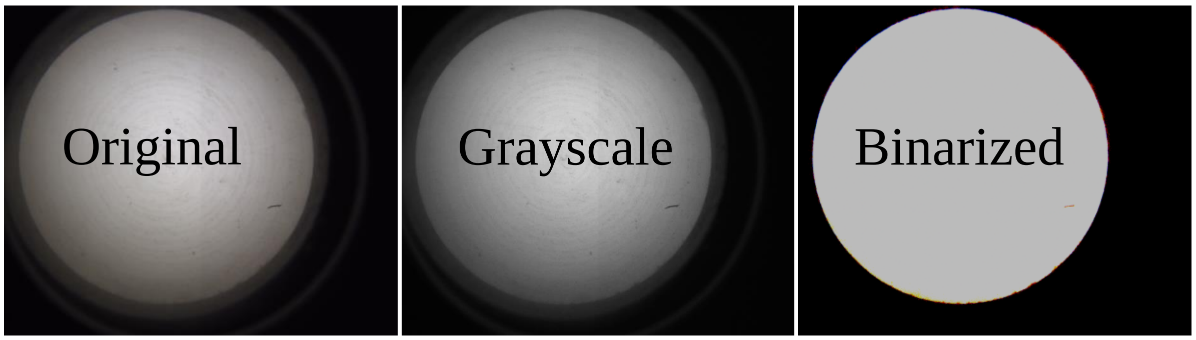


Figure 7: In order to create a mask, the blank reference slide is first grayscaled and then binarized. The mask helps reduce error when comparing different data files to the reference file.

When binarizing, a threshold must be specified. The threshold dictates how to allocate the pixels with intensities that might be counted either as black or white. In the case of the reference slide image, a threshold of 23 best approximates the accurate area of the slide within the image. Lower thresholds, such as the one of 10 shown in Figure 8, allocate too many pixels to the slide, resulting in a larger slide and a halo effect. Not all of these pixels should be compared between the data and reference files. On the other hand, when the threshold is too great, for example, 30, as in Figure 8, too few pixels are allocated to the slide area. As a result, Benthia will not cycle through all the pixels that should be compared between the data and reference slides. A threshold of 23 eliminates the fuzziness seen around the slide's exterior in the other two images.

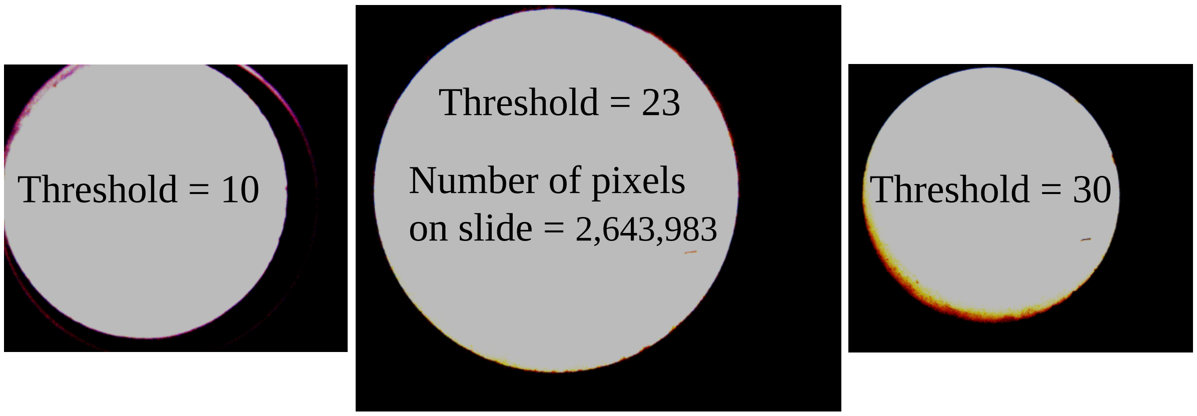


Figure 8: Binarization requires a specified threshold value; in this case, a threshold of 23 best approximates the number of pixels in the slide. Allocating the right number of pixels to count as the slide affects later data calculations.

Binarizing achieves the goal of masking the images and therefore reducing the error resulting from variation in the pixels outside of the slide. Benthia will only cycle through the pixel values within the slide and compare the intensity of each data slide pixel to the intensity of the corresponding pixel in the reference slide. Masking eliminates the effects of the varying amounts of light permeating the sediment and changing the intensities of the pixels of the area around the slide.

2.4 DATA VALUES: PERCENT COVERAGE AND ATTENUATION

The next challenge in writing Benthia was determining how to characterize the composition of the sediment on the SES slide in just a couple data values. Percent coverage and attenuation of the slide were identified as the most efficient means. Percent coverage reveals the amount of the slide with sediment deposited on it. Attenuation, or the loss of intensity through a medium, quantifies the density or darkness of the matter. As shown in Figure 9, it is important to distinguish between the two: percent coverage and attenuation both express different characteristics of the composition of the sediment deposited on each data image.

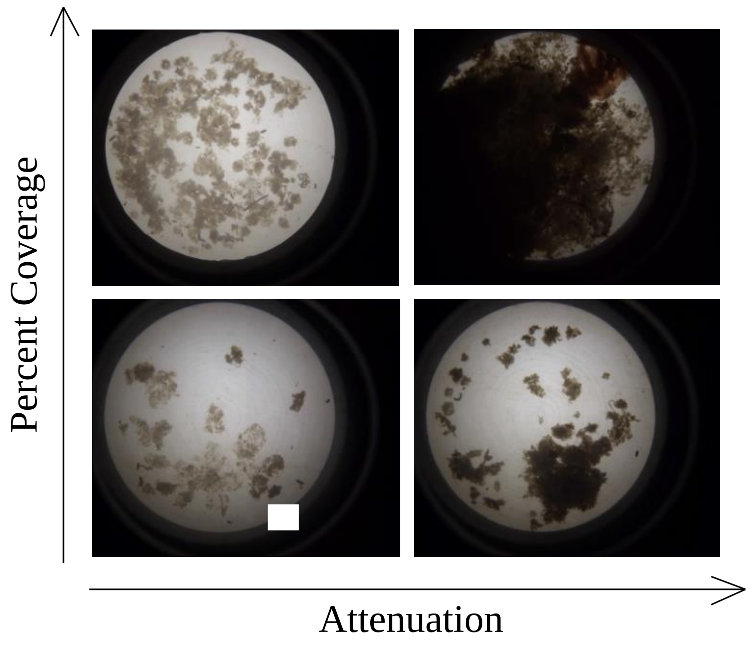


Figure 9: It is important to distinguish between the two data values generated by Benthia, percent coverage and attenuation, as the two values reveal different qualities about the sediment deposited on the slide. The data files shown here were generated during Pulse 61 (top left: 01/29/13, bottom left: 12/23/12, top right: 04/08/13, bottom right: 04/07/13).

COVERAGE

Benthia cycles through all the data files and compares them each to the reference file, pixel by corresponding pixel. To calculate the coverage, the program counts each time the corresponding pixel values differ between the reference and the data file. It returns a percent value (0% - 100%) of the number of pixels in each data file that differ from their corresponding reference file pixels.

The user has the option to enter a threshold value from 0% to 100%. Benthia translates the percentage into an intensity value (0 to 4096). In order for a pixel to be considered "different" and count towards coverage, the difference between the intensity of the reference image and the data image must be greater than the intensity threshold. The default intensity threshold is 0% or an intensity of 0, meaning that any difference in the reference and data images' corresponding pixels counts towards the coverage value. Increasing the threshold may therefore decrease the coverage value of an image or leave it unchanged, but a greater threshold will never increase the coverage value.

ATTENUATION

To calculate attenuation, Benthia cycles through all the pixels of the data file and that of the reference file included by the mask. It compares corresponding pixels' intensities. This function also counts the number of the pixels in the data file with intensity values that differ from the intensities of their corresponding pixels in the reference file. For the pixels whose intensities differ, it takes the difference between the pixel intensities and adds the difference to an intensity sum. Finally, the program averages these differences in pixel intensities and converts the average intensity difference value (0 to 4096) to a percentage (0% to 100%).

TESTING COVERAGE

After some initial debugging, Benthia produced reasonable coverage percentage values. A known number of pixels (in the form of a black rectangle) on the blank reference slide were modified. The modified reference image was then run through Benthia with the original reference slide. If the percent difference of the modified reference image matched the fraction of modified pixels versus the total number of pixels allocated to the slide and included by the mask, then the percent coverage values would be accurate.

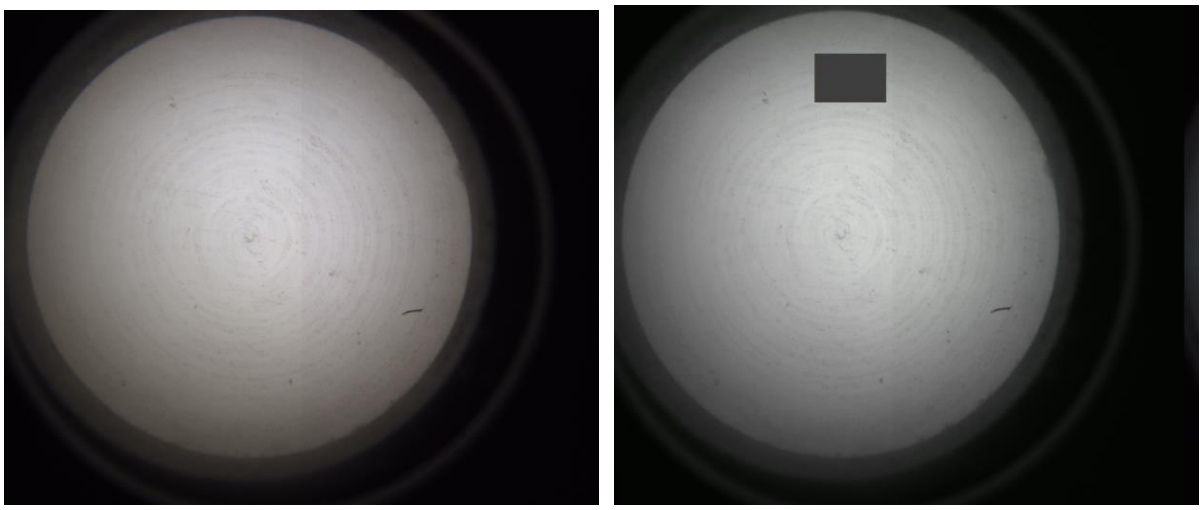
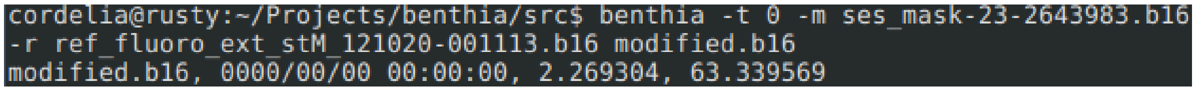


Figure 10: The reference file (left) was modified by a known amount to yield the modified reference file (right) for testing.

Below, Benthia is called in terminal to compare the two images shown above in Figure 9. The program uses a threshold of 0 when comparing each pixel, meaning that any difference greater than 0 counts as a pixel intensity difference. The first file, preceded by “-m,” “ses_mask-23-2643983.b16,” is the mask file with a threshold of 23. The next file, “-r,” “ref_fluoro_ext_121020-001113.b16,” is the original reference file. The “modified.b16” is the modified reference file that acts as a data file used for testing. Benthia calculates that 2.269304% of the pixel intensities differed between the reference file and the modified one.

The reference file and the modified reference file differ from one another by a black rectangle with dimensions 900 to 1200 (range of 300) and from 200 to 400 (range of 200). In total, 60,000 pixels (200 * 300) were modified to have intensities that were lesser than those of their corresponding pixels in the reference image. The simple calculation below proved Benthia’s coverage percentage values accurate.

$$\frac{\text{Modified Pixels}}{\text{Total Number of Pixels}} = \frac{100 * 60{,}000}{2{,}643{,}983} = 2.269304\ \%$$



```
cordelia@rusty:~/Projects/benthia/src$ benthia -t 0 -m ses_mask-23-2643983.b16
-r ref_fluoro_ext_stM_121020-001113.b16 modified.b16
modified.b16, 0000/00/00 00:00:00, 2.269304, 63.339569
```

Figure 11: The reference file (left) and the modified reference file (right) were put through Benthia to yield a coverage percentage value. The two values generated by Benthia are the percent coverage on the left: 2.2269304 and the attenuation on the right: 63.339569.

TESTING ATTENUATION

As shown below, Benthia compared the binarized mask file ("-r ses_mask-23-2643983.b16") and a modified binarized mask file ("modified-bin.b16") to test for attenuation. The modified binarized mask file had the same 200-by-300 pixel rectangle, where each pixel had an intensity of 0. As a result, the difference between the white pixels (intensity of 4096) of the binarized mask file (acting as the reference file) and the pixels in the black rectangle (intensity of 0) in the modified binarized mask file was 4096 for each of the 60,000 pixels.

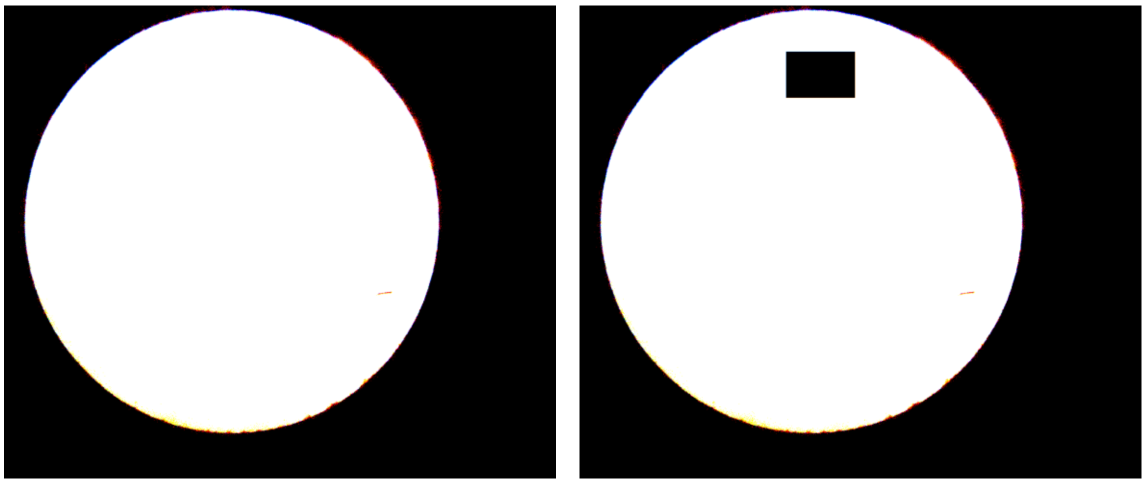
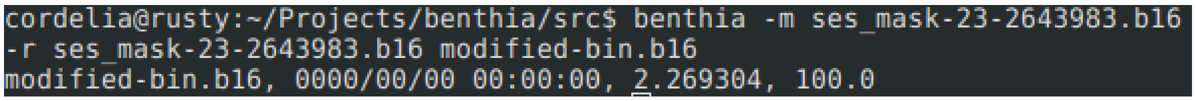


Figure 12: Benthia compared the binarized mask file and a modified binarized mask file to test for attenuation.

Averaged, the attenuation yielded that the average difference in intensity between all the pixels with different intensities was 4,096. However, Benthia scales the difference to a percent scale from 0% to 100%. Divided by 4,096, the attenuation was 100%, the correct and expected attenuation for the binarized mask and the modified binarized mask.

$$\frac{\text{Sum of Differences Between Pixel Intensities}}{\text{Number of Pixels with Different Intensities}} = \frac{100 * (60{,}000 * 4{,}096)}{(60{,}000 * 4{,}096)} = 100\%$$

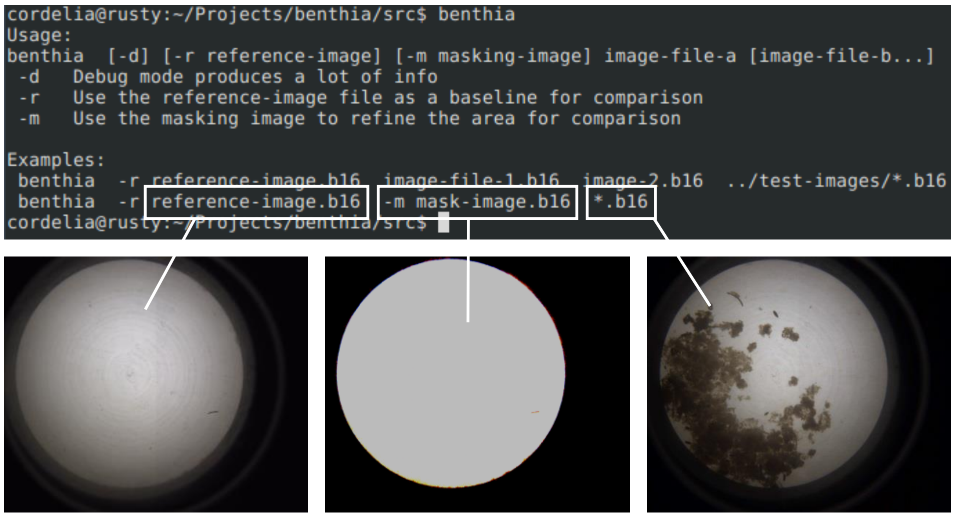


```
cordelia@rusty:~/Projects/benthia/src$ benthia -m ses_mask-23-2643983.b16
-r ses_mask-23-2643983.b16 modified-bin.b16
modified-bin.b16, 0000/00/00 00:00:00, 2.269304, 100.0
```

Figure 13: Benthia compared the binarized mask and the modified binarized mask code to correctly returned the attenuation: 100 % (right).

RESULTS

In its final form, Benthia accepts a threshold (optional: if unspecified, the default threshold is 0), a reference file, a mask file (optional: recommended to maximize data's accuracy), and one or more data files.



```
cordelia@rusty:~/Projects/benthia/src$ benthia
Usage:
benthia  [-d] [-r reference-image] [-m masking-image] image-file-a [image-file-b...]
 -d   Debug mode produces a lot of info
 -r   Use the reference-image file as a baseline for comparison
 -m   Use the masking image to refine the area for comparison

Examples:
 benthia  -r reference-image.b16  image-file-1.b16  image-2.b16  ../test-images/*.b16
 benthia  -r reference-image.b16  -m mask-image.b16  *.b16
cordelia@rusty:~/Projects/benthia/src$
```

Figure 14: Benthia accepts a threshold, a reference slide, a mask (optional), and one or more data slides, all of which must be in a bayer16 format.

As shown in Table 2, the different components have different requirements for placement and arguments within the terminal command line. Benthia was designed to allow the user to tailor the program to accomplish specific analysis.

Components of the Benthia command line in terminal

Component	Argument	Position
Threshold	-t	--
Reference File	-r	--
Mask File	-m	--
Data File(s)	--	At end

Table 2: Some of the components of the command line must be entered in a specific order or must be preceded by a specific argument.

Benthia operates at 1-2 seconds per image on an Intel processor that runs at more than 1GHz. The TS-7200 on the SES, on the other hand, runs at 200 MHz. Assuming that the TS-7200 is one order of magnitude slower, we can predict that the SES will take 10-20 seconds to process each image. These times have not been tested for accuracy and should be before Benthia is formally incorporated into the SES image analysis. However, since the TS-7200 would only be processing one image at a time to generate coverage values (which would determine whether to adjust the duration of the collection time), 10-20 seconds is fast enough.

In order to analyze trends with in ocean activity with respect to Benthia analysis, Pulse 60 and Pulse 61 were run through Benthia. The following data was generated:

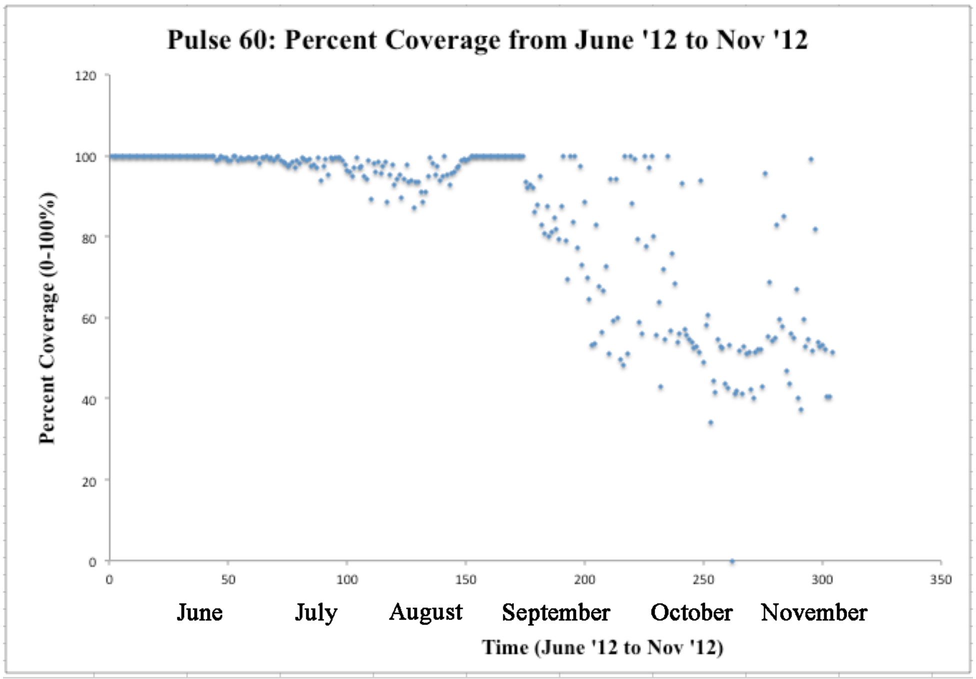


Figure 15: The percent coverage of Pulse 60 remained at or near 100% for the beginning of the deployment.

The percent coverage of Pulse 60 showed that the SES slide was nearly to almost covered for most of the first half of the deployment. The second half of the deployment exhibited more variation and lower percent coverage values. The percent coverage never dropped below 30%.

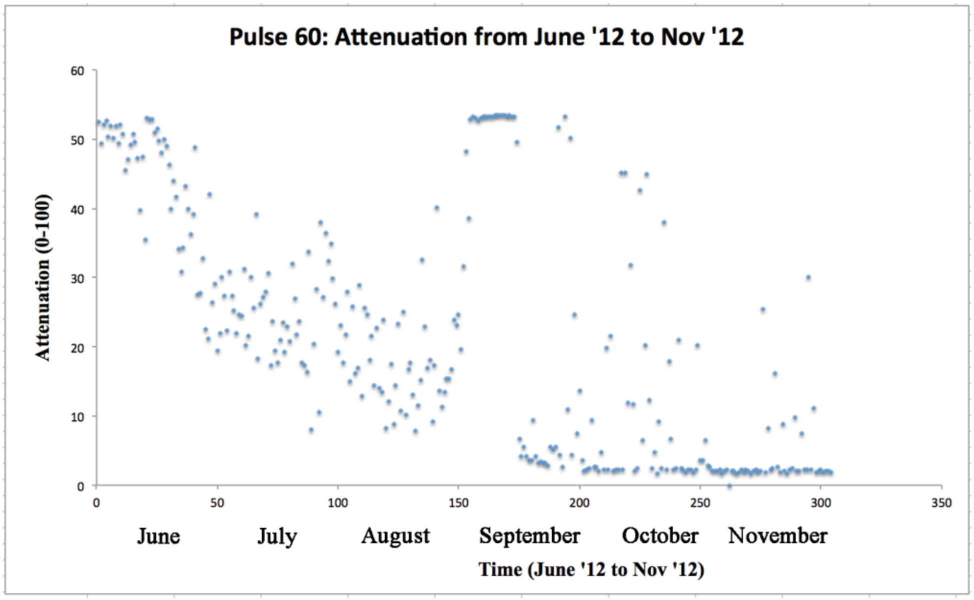


Figure 16: The Pulse 60 attenuation dropped to near 0 by the end of the five-month deployment.

The Pulse 60 attenuation never exceeded 55 throughout the entire deployment. During the first half of Pulse 60, the attenuation steadily dropped. In between August and September, the deployment experienced a period of high attenuation, followed by a drastic drop to a near value of 0 for the rest of the deployment.

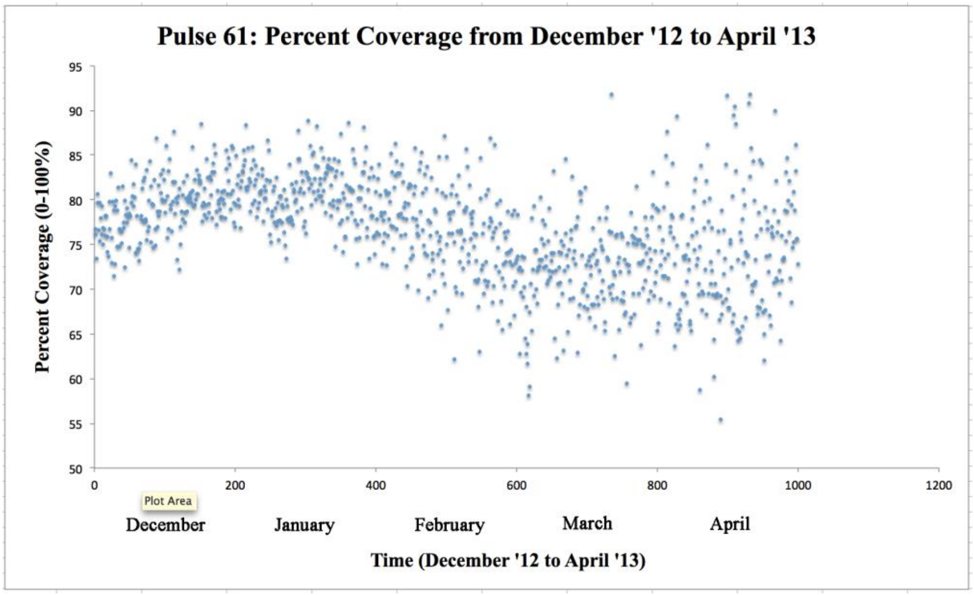


Figure 17: The Pulse 61 percent coverage showed a slight increase during the first half of the deployment, followed by a decrease in the second half.

The Pulse 61 percent coverage experienced an increase during the first half of the deployment. The percent coverage dropped during the second half of Pulse 61. In addition, the variation of the data increased toward the end of the deployment. Overall, the percent coverage ranged between close to 95% and just above 55%.

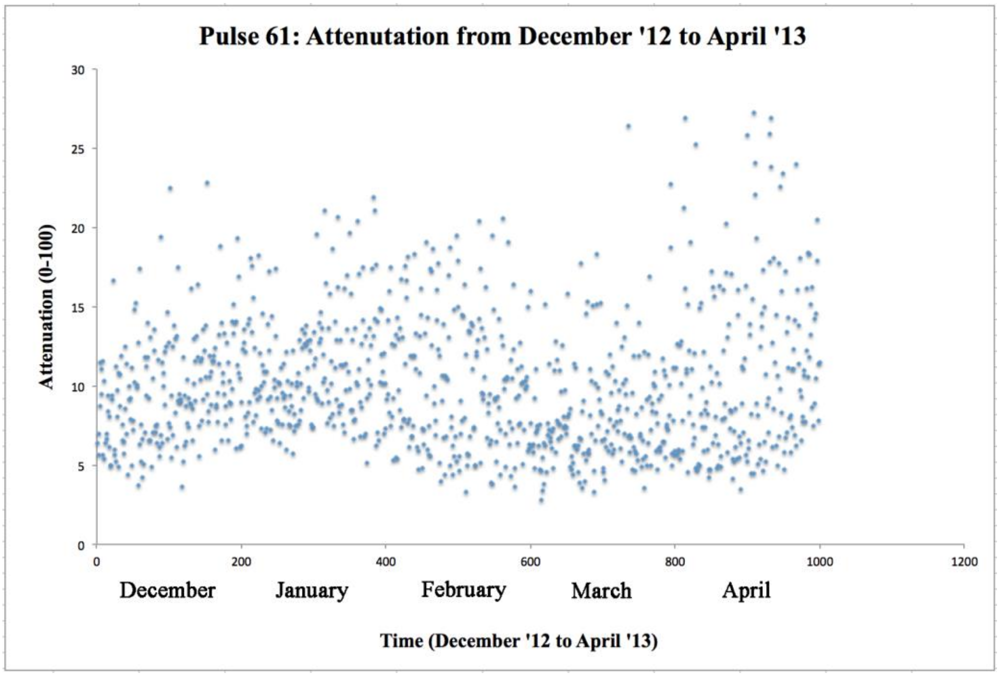


Figure 18: The Pulse 61 attenuation remained below 20 for the majority of the deployment.

The Pulse 61 attenuation did not exhibit a strong trend. The variation in data increased slightly in mid-February and again more strongly in April, at the end of the deployment. Attenuation never exceeded 30 and never dropped below 3.

DISCUSSION

It is well known that the phytoplankton blooms occur in the spring. The mixing of nutrients during the winter, followed by their stratification, which results in an increased intensity of light pervading the trophic zone, creates the ideal environment for a phytoplankton bloom. Next, the populations of zooplankton that eat the phytoplankton surge. Studies have shown that it takes two to three months for the marine snow to sink 4,000 meters to where the SES is deployed (7).

We would therefore expect there to be both greater percent coverage and attenuation during the months after the plankton bloom. The lag accounts for the sinking time of the marine snow. Comparing the graphs of the percent coverage and attenuation of Pulse 60 and Pulse 61 supported this conjecture, since Pulse 60 took place right after the plankton bloo at Pulse 61 did not.

The percent coverage graph of Pulse 60 (June ’12 to November ’12) showed a percent coverage of near 100% from June to mid-September. Afterwards, the data dropped as far as 40%, intermittently returning to 100%. On the other hand, the percent coverage graph of Pulse 61 (December ’12 to April ’13) rarely surpassed a 90% coverage. The majority of the data points ranged from 65% to 85%. The months after the springtime plankton bloom showed more marine snow.

The attenuation graph of Pulse 60 (June ’12 to November ’12) began at just above 50%. From late-June to mid-August, the attenuation gradually dropped. There was a resurgence of attenuation to 65% from the beginning to the middle of September. In contrast, the majority of the attenuation graph of Pulse 61 (December ’12 to April ’13) only ranged from 5% to 15%. The density of the matter falling from the trophic zone was greater following the spring plankton bloom as well.

It was interesting to compare the percent coverage and attenuation graphs of the Pulse 60 (June ’12 to November ’12). The attenuation graph dropped more quickly than did the percent coverage graph, insinuating that the denser particles of marine snow (that led to a greater attenuation) must have fallen more quickly through the water column. The lighter particles took longer and therefore maintained the high percent coverage values for longer. These particles may have been too fine to contribute to a greater attenuation. Finally, the early-September to mid-September surge in both percent coverage and attenuation must mean that the marine snow captured on the SES then was denser.

CONCLUSION

As the effects of climate change become more apparent and the capabilities of our technologies increase, it will become more and more important to develop image analysis to keep up with data generated in studies. Data values, such as percent coverage and attenuation have already revealed trends in ocean activity. Benthia is but a step towards an understanding of the dynamics, changes, and future of the marine nutrient cycling. This system is a vital foundation to the oceans and our atmosphere. The more it is researched the better we can anticipate possible threats to it.

RECOMMENDATIONS

In October of 2014, the SES will be deployed at Station M. The researchers will be employing Benthia for image analysis both in situ and on shore, post-deployment. Although Benthia’s code is complete, it has yet to be incorporated in and tested on in situ image analysis on the Sedimentation Event Sensor. If successful, in situ, coverage data yielded by Benthia will prompt the SES to adjust the duration of sedimentation collection. Further testing is necessary.

The approach delineated in this paper is just one method of completing the important task of expediting and standardizing image analysis of the SES. Ten weeks was just enough to build Benthia’s functionality. Further work should investigate ways of optimizing the code. Speeding up the image analysis process is worth exploring in order to reduce energy consumption. The power consumed by the TS-7200 is constant, meaning that the total electrical energy used to process an image on the SES is directly related to how long it takes.

ACKNOWLEDGEMENTS

I’d first like to recognize the Ken Smith Lab. I’m indebted to Rich Henthorn’s amazing mentorship—he made this summer the incredible experience that it was. Between teaching me the difference between “bang” (!) and “splat” (*) and choosing a project of the perfect scope for my abilities, Rich was a better mentor than I could have hoped for. I’d also like to thank Paul McGill for fielding all my existential crises about MATLAB. Johnny Ferreira made the summer even more special when he let me put o-rings into the benthic rover. Thank you to Ken Smith for humoring my excitement during lab meetings about matters that probably weren’t that exciting—like putting o-rings into benthic rovers.

I’m also grateful to George Matsumoto and Linda Kuhnz. Without their hard work, this internship would not have run as smoothly, and the interns would not have had as many incredible opportunities. The 2014 summer interns taught me so much—thank you. Finally, I want to recognize the David and Lucile Packard Foundation for making the internship possible—it’s been a magical summer.

References:

[1] B. J. Bett, D. S. M. Billett, R. S. Lampitt, H. A. Ruhl, R. S. Kaufmann, and K. L. Smith, Jr. (2009). Climate, carbon cycling, and deep-ocean ecosystems. *Proceedings of the National Academy of Sciences*, 46: 19211–19218.

[2] Rowe GT (1971). *Fertility of the Sea*, ed Costlow JD (Gordon and Breach, New York), pp 441– 445.

[3] Johnson NA, et al. (2007). The relationship between the standing stock of deep-sea macrobenthos and surface production in the western North Atlantic. *Deep-Sea Res I* 54:1350 –1360.

[4] Smith KL, Jr (1987). Food energy supply and demand: A discrepancy between particulate organic carbon flux and sediment community oxygen consumption in the deep ocean. *Limnol Oceanogr* 32:201–220.

[5] Pfannkuche O (1993). Benthic response to the sedimen- tation of particulate organic

matter at the BIOTRANS station, 47°N, 20°W. *Deep-Sea Res II* 40:135–149.

[6] McClain, Craig. "Main Menu." *Deep Sea News*. N.p., n.d. Web. 14 Aug. 2014.

[7] R.J. Baldwin, M. Kahru, R.S. Kaufmann, B.G. Mitchell, H.A. Ruhl, K.L. Smith (2006). Climate Effect on Food Supply to Depths Greater than 4,000 Meters in the Northeast Pacific. *Limnology and Oceanography* 51:166-176.

APPENDIX: CODE

1. ImageFile.h

```
/******************************************************************************
 * Copyright 1990-2014 MBARI
 * MBARI Proprietary Information. All rights reserved.
 ******************************************************************************
 * Summary  : Example: implements interface to ACM 2-D Current Meter
 * Filename :
 * Author   :
 * Project  : Example: Benthic Rover
 * Version  :
 * Created  : Example: Nov 08
 * Modified :
 *****************************************************************************/
const unsigned int HDRSIZE = 122;
const unsigned int IMGSIZE = 5018400;
const unsigned int SLDSIZE = 2643983;

 class ImageFile
 {
   public:

   // Ctor
   ImageFile(const char* img_filename, bool debug);
```

```
    // Dtor
    ~ImageFile();


    // Member functions or methods
    unsigned int load();
    unsigned int dump(const char* filename);
    float calc_coverage(ImageFile *reference, ImageFile *mask);
    float calc_coverage(ImageFile *reference, ImageFile *mask, unsigned int threshold);
    float calc_intensity(ImageFile *reference, ImageFile *mask);
    unsigned int extract_timestamp();


    // Member variables or class attributes
    bool  _debug;
    char* _filename;
    unsigned short int _header[HDRSIZE];
    unsigned short int _data[IMGSIZE];
    float _coverage;
    float _intensity;

    // File timestamp
    unsigned int _year;
    unsigned int _month;
    unsigned int _day;
    unsigned int _hour;
    unsigned int _min;
    unsigned int _sec;


    unsigned int _cols, _rows;

  };
```

2. ImageFile.cpp

```
/*******************************************************************************
 * Copyright 1990-2014 MBARI
 * MBARI Proprietary Information. All rights reserved.
 *******************************************************************************
 * Summary  : Example: implements interface to ACM 2-D Current Meter
 * Filename :
 * Author   :
 * Project  : Example: Benthic Rover
 * Version  :
 * Created  : Example: Nov 08
 * Modified :
 ****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include "ImageFile.h"

ImageFile::ImageFile(const char* img_filename, bool debug)
{
  _debug = debug;
  _filename = strdup(img_filename);
}

ImageFile::~ImageFile()
{
  if (_filename) delete _filename;
}

float ImageFile::calc_coverage(ImageFile *reference, ImageFile *mask)
{
        return calc_coverage(reference, 0);
}


float ImageFile::calc_coverage(ImageFile *reference, ImageFile *mask, unsigned int threshold)
{
  // Define function calc_coverge
  float coverage_counter;
  unsigned int pixs = 0;

  for (unsigned int i = 0; i < IMGSIZE; i++)
  {
    // Skip only if there's a masking image and the masking data at this pixel equals 0
    //
    if (mask && (mask->_data[i] == 0))
          continue;
    else
          pixs++;

    // darker pixels have lower values
    if ( (_data[i] + threshold) < reference->_data[i] )
```

```
      {
         //printf("samp data[%d] = %u      ref data[%d] = %u\n", i, _data[i], i, reference->_data[i]);
        coverage_counter += 1.0;

         // if (_debug) printf("Image %4u + %3d < Ref %4u by %4u\n", _data[i], threshold,
         reference->_data[i],
        // reference->_data[i] - (_data[i] + threshold));
           }
         }

         _coverage = (coverage_counter / pixs) * 100;
   if (_debug) printf("%f/%ld = %f\n", coverage_counter, pixs, _coverage);

         return _coverage;
}

float ImageFile::calc_intensity(ImageFile *reference, ImageFile *mask)
{
   // Define function calc_intensity
   float intensity_counter = 0.;
   float intensity_sum = 0.;
   unsigned int pixs = 0;

   for (unsigned int i = 0; i < IMGSIZE; i++)
   {
      // Skip only if there's a masking image and the masking data at this pixel equals 0
      //
      if (mask && (mask->_data[i] == 0))
         continue;
      else
         pixs++;


      if (_data[i] < reference->_data[i] )
           {
              intensity_sum += reference->_data[i] - _data[i];
                     intensity_counter++;
           }
         }

   if (_debug) printf("Using %ld pixels in intensity image\n", pixs);
         if (intensity_counter > 0) _intensity = (intensity_sum / intensity_counter) / 40.96;

         return _intensity;
}

unsigned int ImageFile::load()
{
```

```
// Check for access first and exit if we don't
if (0 != access(_filename, R_OK))
{
  if (_debug) printf("ImageFile::load - cannot access %s\n", _filename);
  return 1;
}

// Open file and read header
int fd = open (_filename, O_RDONLY);

int nbytes = read (fd, _header, sizeof(_header));

// Report debug
if (_debug) printf("ImageFile::load - read %d bytes in header\n",
                                        nbytes);

// Check to see if right amount of bytes were read
if (nbytes < sizeof(_header))
{
        printf("ImageFile::load - error: header size too small\n");
        return 1;
}

if (_debug)
{
        for (int i = 0; i < HDRSIZE; i++)
        {
          if (_debug)
                    if (_header[i] != 0) printf("Header [%d] = %d\n", i, _header[i]);
        }
  printf("Is this a time stamp: %ld\n", _header[0]);
  printf("Is this a time stamp: %ld\n", _header[8]);
  printf("Is this a time stamp: %ld\n", _header[54]);
}

// Location of image dimensions within header
_cols = _header[38];
_rows = _header[40];

// Verify image dimensions
if ((_cols * _rows) != IMGSIZE)
{
        printf("ImageFile::load - error bad dimensions: %d x %d\n",
                              _cols, _rows);
        return 2;
}

// Read in image data
```

```
  nbytes = read (fd, _data, sizeof(_data));
  if (_debug) printf("ImageFile::load - read %d bytes of data\n",
                                          nbytes);

  // Check to see if right amount of bytes were read
  if (nbytes < sizeof(_data))
  {
        printf("ImageFile::load - error: file size too small\n");
        return 1;
  }

  extract_timestamp();

  return 0;
}


// Use the file name to extract the timestamp of the image,
// trying to match known file naming methods.
// Return 0 if successful.
//
unsigned int ImageFile::extract_timestamp()
{
  unsigned int ret_val = 0;
  _year = _month = _day = _hour = _min = _sec = 0;

  // Attempt to extract the date and time from the filename.
  // Assuming filename format something like:
  // some_image_description_YYMMDD-HHMMSS.b16
  //
  // First, find the extension .b16
  // strstr(haystack, needle) finds and returns a pointer to a substring
  // within a string, or returns NULL if not found.
  // Don't bother unless the extension is found and the
  // length of the filename is long enough
  //
  char *extension = NULL;
  if (strlen(_filename) >= strlen("YYMMDD-HHMMSS.b16")) extension = strstr(_filename,
".b16");

  if (NULL != extension)
  {
    // Look for naming method file_name_yymmdd-hhmmss.b16
    //
    char *datetime = extension - strlen("YYMMDD-HHMMSS");   // Back up to the start of the date
and time
    int n = sscanf(datetime, "%2d%2d%2d-%2d%2d%2d",
      &_year, &_month, &_day, &_hour, &_min, &_sec);
    _year += 2000; // This century

    if (_debug) printf("%d items => %04d/%02d/%02d %02d:%02d:%02d\n",
      n, _year, _month, _day, _hour, _min, _sec);

    // If the date and time was scanned correctly, n will equal 6
    // and the date/time components will contain the values.
    //
    // If the date and time was not parsed correctly, reset the values
    // and return a non-zero value.
    //
    if (n != 6)
    {
      _year = _month = _day = _hour = _min = _sec = 0;
      ret_val = 2;   // Not a recognized naming method
    }
  }
  else
  {
    ret_val = 1;      // File name too short or not a .b16 file
  }

  return ret_val;
}


// Write the data to a file, possibly with mods
//
unsigned int ImageFile::dump(const char* new_filename)
{
  if (0 == strcmp(_filename, new_filename))
  {
    printf("Not going to overwrite image file: Use a name other than %s\n", _filename);
    return 1;
  }
  int fd = open (new_filename, O_WRONLY|O_CREAT|O_TRUNC,
S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP);
  if (_debug) perror("");

  int nbytes = write(fd, _header, sizeof(_header));
  printf("Wrote %d bytes to %s\n", nbytes, new_filename);

  // Write a rectangle into the image
  //
  float mask_level = 0.23;
  unsigned long counter = 0;
  unsigned int level = mask_level * 4096;
  for (int i = 0; i < IMGSIZE; i++)
  {
    if (_data[i] <= level)
      _data[i] = 0;
    else
    {
      _data[i] = 4090;
      counter++;
    }

  }
  nbytes = write(fd, _data, sizeof(_data));
  printf("Wrote %d bytes to %s, %ld non-zero values\n", nbytes, new_filename, counter);

  close(fd);
  printf("Wrote modified version to %s\n", new_filename);

  return 0;
}
```

Benthia.cpp

```
/*****************************************************************************
* Copyright 1990-2014 MBARI
* MBARI Proprietary Information. All rights reserved.
*****************************************************************************
* Summary  : Example: implements interface to ACM 2-D Current Meter
* Filename :
* Author   :
* Project  : Example: Benthic Rover
* Version  :
* Created  : Example: Nov 08
* Modified :
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <math.h>

#include "ImageFile.h"
```

```
// Function shows a concise help
//
void usage()
{
  const char* help =
  "Usage:\n"
  "sesia  [-d] [-r reference-image] [-m masking-image] image-file-a [image-file-b...]\n"
  " -d   Debug mode produces a lot of info\n"
  " -r   Use the reference-image file as a baseline for comparison\n"
  " -m   Use the masking image to refine the area for comparison\n"
  "\nExamples:\n"
  " sesia  -r reference-image.b16  image-file-1.b16  image-2.b16  ../test-images/*.b16\n"
  " sesia  -r reference-image.b16  -m mask-image.b16  *.b16\n";

  printf(help);
}


int main(int argc, char** argv)
{
  // We will eventually set the debug flag using an option
  // sesia -d filename
  //
  bool debug = false;

  unsigned int threshold = 0;

  //
  char* ref_filename  = NULL;
  char* mask_filename = NULL;
  char* mod_filename  = NULL;
  int error = 0;
  int c;
  int fnames = 1;
  while ((c = getopt (argc, argv, "?w:dt:r:m:")) != -1)
  {
    switch (c)
    {
      case 'd':
        debug = true;
        fnames += 1;
        break;

         // Set the threshold value, entered as a percentage (0-100),
         // which is then converted into a pixel intensity (0-4096 = 2**12)
      case 't':
                 if ((atoi(optarg) < 0) || atoi(optarg) > 100)
```

```
                {
                        printf("Invalid old value. Threshold value ranges from 0 to 100.");
                        error = 1;
                }
        threshold = atoi(optarg) * 40.96;
        fnames += 2;
        break;

    // Identify the reference file
      case 'r':
        ref_filename = strdup(optarg);
        if (strlen(ref_filename) <= 0)
        {
          printf ("Reference file: bad filename length!\n");
          error = 1;
        }
        fnames += 2;
        break;

    // Identify the masking file
      case 'm':
        mask_filename = strdup(optarg);
        if (strlen(mask_filename) <= 0)
        {
          printf ("Masking file: bad filename length!\n");
          error = 1;
        }
        fnames += 2;
        break;

      case 'w':
                  mod_filename = strdup(optarg);
                        if (strlen(mod_filename) <= 0)
                        {
          printf ("Output file: bad filename length!\n");
                          error = 1;
                        }
        fnames += 2;
                        break;

                case '?':
                        error = 1;
                        break;

                default:
                printf ("bug\n");
        return (1);
```

```
       }
  }

  // Check for other errors (threshold limit, reference filename, etc)
  //
  if (error) {
    printf("Usage: sesia -r ref_filename imfile...]\n");
    return (2);
  }

  // Let's get it started
  //
  if (debug) printf("sesia processing...\n");

  // Check input. Show some help if invalid
  //
  if (argc < 2)
  {
    usage();
    return 1;
  }

  // Load the reference file
  ImageFile *ref = new ImageFile(ref_filename, debug);
  ref->load();
  if (mod_filename != NULL)
  {
   printf("Writing modified %s to %s\n", ref_filename, mod_filename);
         ref->dump(mod_filename);
  }

  // Load the mask file, if any
  //
  ImageFile *mask = NULL;
  if (mask_filename)
  {
    mask = new ImageFile(mask_filename, debug);
    if (0 != mask->load())
    {
      delete mask;
      mask = NULL;
    }
  }
  // Process the arguments
  //
  for (int i = fnames; i < argc; i++)
  {
    if (debug) printf("processing... %s\n", argv[i]);
```

```
      // Create an ImageFile object for this file
      ImageFile *ifile = new ImageFile(argv[i], debug);

      // Tell object to load file
      ifile->load();
      //for (int j = 0; j < 2448*2050; j++) printf("%d \n", ifile->_data[j]);

      // Tell object to process the file
      //float cov = ifile->calc_coverage(ref);
      //if (debug) printf("Coverage of %s is %f\n", argv[i], cov);
      float cov_t = ifile->calc_coverage(ref, mask, threshold);
      float intense_t = ifile->calc_intensity(ref, mask);

      printf("%s, %04d/%02d/%02d %02d:%02d:%02d, %f, %f\n", argv[i],
        ifile->_year, ifile->_month, ifile->_day, ifile->_hour, ifile->_min,
        ifile->_sec, fabs(cov_t), intense_t);

      delete ifile;
    }
    // do_stuff();

    // All done
    //
    return 0;
  }
```