The following supplements accompany the article

A novel three dimensional analysis of functional-architecture that describes the properties of macroalgae as a refuge

Colin Ware*, Jennifer A. Dijkstra, Kristen Mello, Andrew Stevens, Brandon O'Brien, William Ikedo

*Corresponding author: cware@ccom.unh.edu

Marine Ecology Progress Series 608: 93–103 (2019)

Supplement 1: Details of Seaweed Model Construction

This supplement provides details of the methods used for seaweed model construction. The code for *C.f.* spp. *fragile* and *D. japonica* was written in C++ and OpenGL. The model for *S. latissima* additionally used the Bullet Physics Software Developer's Kit.

Codium fragile spp. *fragile* Model

C.f. spp. *fragile* has a binary branching structure with all thalli having a similar lengths and diameters, except for growth buds. A sample of *C.f.* spp. *fragile* was photographed in a tank before being laid out in a shallow dish for detailed measurements. Its branching structure was fully characterized in a diagram with different levels of branches coded using a string of integers with 1 designating the root, 11 the left child, 12 right child, and so on (Figure S1_1). For each thallus, the length and the diameters at three locations (start, middle and end) were measured. Terminal thalli were also noted. From these measurements, mean lengths, diameters as well as the standard deviations of these variables were computed at each level of the tree structure. For *C.f.* spp. *fragile*, a plane through a pair of branches is roughly at right angles to a plane through a pair of branches one level closer to the holdfast. We based estimates of branch angles from tank observations in each case, viewing the sample in a direction orthogonal to the branch plane.

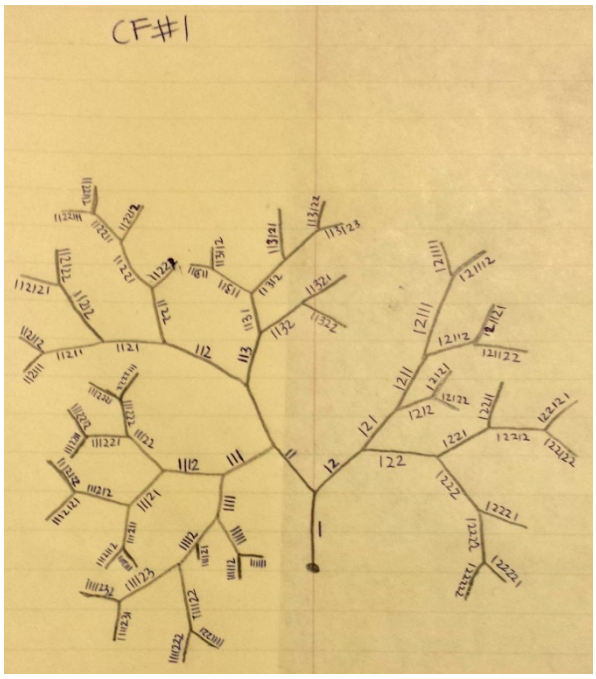

Figure S1_1. A sketch of the branching structure of a *C.f.* spp. *fragile* sample showing the coding scheme used.

The computer graphics model of *C.f.* spp. *fragile* is a simple recursive procedure with a binary branching. Individual thalli are modeled as curved tubes with hermit splines defining the central paths (see Figure 2a). Each thallus was capped with a hemisphere, although this was only visible for the terminii.

The model had the following parameters.

- Thallus length randomly determined between 1.0 and 1.5 cm.
- Thallus diameter of 0.33 cm expanding to 0.43 cm where it bifurcates.
- Child thalli, except for the terminal pairs had a 1:4 chance of being missing.
- Branching angle: 33 deg.
- Branching depth of 6

Dasysiphonia japonica **Model**

D. japonica has the branching structure illustrated in Figures 2 and S1_2. For the samples we examined, each thallus had up to 40 daughter thalli alternately branching to the left and right. The number of daughter branches was a function of the thallus length, although there was considerable variability in this parameter. In addition, there were many missing branches, particularly in the interior of the sample, near to its holdfast.

Because of the very large number of thalli in this fine filamentous seaweed, a hybrid strategy was used to characterize it. The first three levels of branches were all fully characterized in terms of the thallus length, diameter, number of sub-branches, and the spacing of sub-branches. A sampling strategy was adopted for branching levels four and above. To make these measurements a sample was spread out in a shallow dish and as far as possible it was untangled and flattened. The sample was then photographed both as a whole and at magnified scales. To characterize higher level branches, the sub-branches were randomly sampled, to obtain statistical estimates of length, number of sub-branches, etc. At level 5 and above many branches consisted of a single 'leaf', while others had two, three or four more branching levels.

The function

```
br = randProp(0.3)*(5.0*log(len*10.0) + 3.5);
```

was used to generate the number of branches as the fuction of thallus length. Where randProp returns a number uniformly distributed on the range [-0.7, 1.3]. The central panel of Figure 1 illustrates this function fitted to the measurement data.

Like *C.f.* spp. *fragile* the computer graphics model of *D. japonica* was implemented using a recursive algorithm that called itself for each right or left daughter branch. The diameter decreased by a factor of 0.8 at each level and the branch length decreased using a multiplier of 0.4.

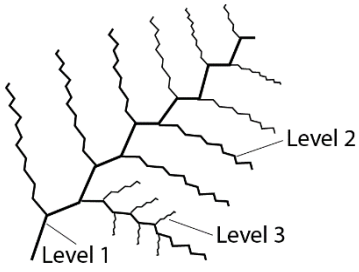

Figure S1_2. Each branch of *D. japonica* can have more than 20 sub-branches, alternating to the left and right as shown. However, there are also considerable irregularities such as missing branches. Beyond level three irregularities increase.

***Saccharina latissima* Model**

The *S. latissima* model was constructed using a very different method. *S. latissima* beds consist of many single bladed instances with each blade having the kind of structure illustrated in Figure S1_3. Whereas other two models consisted of recursive algorithms where self-intersection could occur (in those cases it was felt that this would not greatly affect the way the space was structured) the way individual blades of *S. latissima*, divide up space would be radically changed if they were allowed to pass through one another. Indeed, for *S. latissima* the structure of the space is to a large extent determined by the way the blades are packed and this is determined by forces on each blade exerted by neighboring blades. To accomplish this, a finite element model was developed, to both generate the morphology of *S. latissima* blades and to simulate the interactions between adjacent blades.

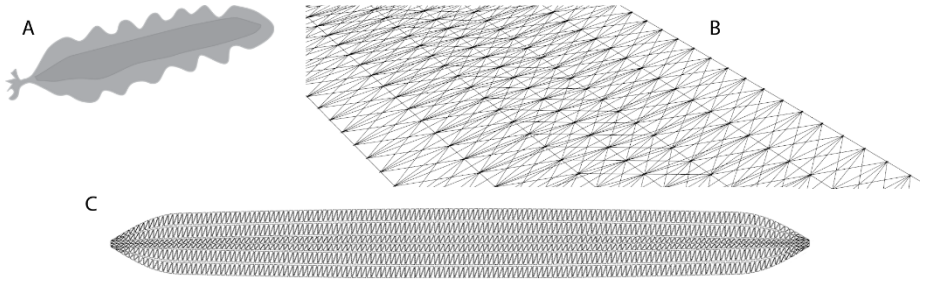

Figure S1_3. A) The blades of *S. latissima* have a central flat portion and a wavy lateral edges due to higher growth rates at the periphery. B) A spring mesh was used to construct each blade. C) The rendering mesh for a single blade.

A quadrat 50cm x 50cm x 50cm containing nine blades was photographed and measured *in situ*. Measurements were taken of holdfast locations, blade entry and exit points (if any) in relation to the quadrat boundaries. Blades were then harvested and measurements were made of each specimen's length, width, thickness, and the frequency of the blade-edge ripples. These measurements were used to generate 2D models of the *S. latissima* specimens which approximated their profile shapes when laid out flat.

The computer graphics model was implemented by starting with a mesh created by sampling the *S. latissima* profile shapes using a topological grid and simplifying the mesh so that vertices were at least 1 cm apart while preserving the profile shape. The meshes were used to create soft body objects via the Bullet Physics Software Developer's Kit (SDK) [www.bulletphysics.org], where each mesh vertex was assigned an equal mass per-specimen and these vertex nodes were linked to all neighboring nodes within a geodesic distance of two using two-way spring constraints. The soft body models were then positioned and oriented within a virtual environment relative to a virtual quadrat so that they roughly matched the observed configurations. The resting lengths of the node links were increased at and near the blade boundaries to achieve the characteristic rippling. Additional constraints were imposed by using spring-like anchors to attach holdfasts and parts of the soft body models to their entry and/or exit points in the quadrat. The simulation was allowed to run until the blades had reached a satisfactory conformation.

Supplement 2: Implementation Details for Spherical Space Analysis

This supplement provides implementation details for the efficient computation of inaccessible volume and area curves.

In outline the algorithm consists of the following six step process.

Step 1: Construct 3D voxel model of macroalgae sample within a 3D voxel volume

Step 2: From the outside-in employ a volume fill algorithm to fill all accessible parts (e.g. Feng & Soon 1998).

Step 3: Re-draw and voxelize the model enlarged by r

Step 4: Fill again from the outside this time marking the voxels in the outer shell in the enlarged model.

Step 5: For every boundary voxel in the enlarged model, label within radius r using a kernel

Step 6: Count the inaccessible volume and inaccessible area voxels

Figure S2_1 illustrates the first 4 steps. The goal of steps 2, 3 and 4 is to identify voxels within radius r of the macrophyte model. These are used for final filling step and since they typically make up less than 1% of the voxels in the volume, a more than 100 times speedup result. This process is repeated for many values of r to obtain inaccessible volume and inaccessible area curves. These curves serve as empirical functions.

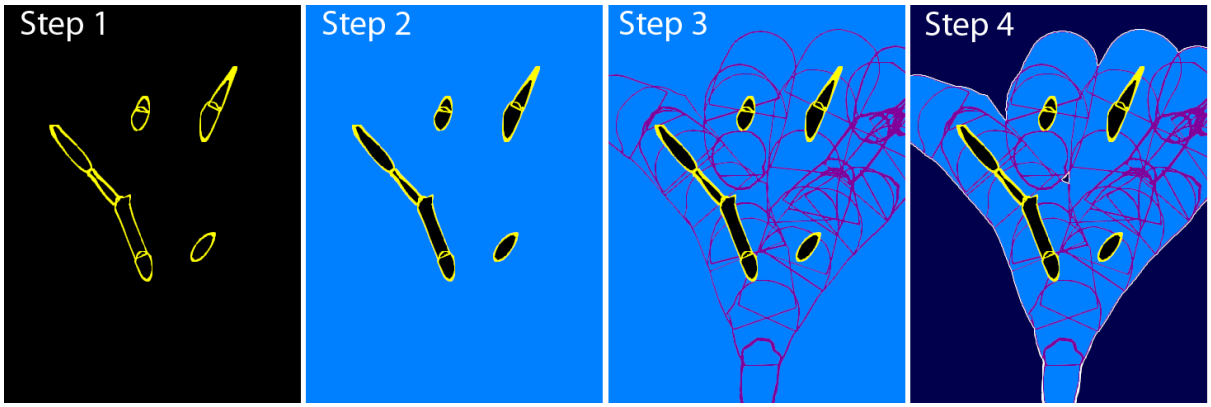

Figure S2_1. Steps 1 through 4 as implemented for a *C.f.* spp. *fragile* model. They all show the same slice through the voxel volume. Step 1: Following model capture. Step 2: Following volume fill. Blue represents water. Step 3: Following enlarged model capture. Step 4: Boundaries of enlarged model have been identified.

What follows is a more detailed account of the same six steps. In this account we use ***r*** to denote the radius in centimeters of the theoretical spherical organism, and ***R*** to denote the radius in voxels of the organism.

Step 1: ***Construct 3D voxel model of macroalgae sample within a 3D voxel volume***

Memory is allocated for a three dimensional array with one byte per voxel. A volume 600x600x600 bytes, for example is easily stored in the main memory of a modern desktop computer. Individual voxel bits are used to label the results of operations. The voxel volume is padded on all six sides by ***R***.

Step 1 of the algorithm requires that a voxelized model of the macrophyte be constructed within a larger three dimensional voxel volume. Unfortunately, standard computer graphics models produce three dimensional polygonal surfaces, not solid volumes. They give the appearance of three dimensional solids, but in fact are empty shells. Although development of a code or a software layer can be used to construct voxel-based models or polygonal surfaces within a 3D volume (e.g., Chandru et al. 1995), standard techniques from computer graphics cannot be employed. Consequently, we developed an alternative method that involved scanning a standard polygonal model using the near and far clipping planes to isolate slices through the model (see a standard computer graphics textbook such as (Marschner & Shirley 2015) for a discussion of projection and clipping). This process can be summarized as follows:

An orthographic projection (i.e., a representation of the 3D object into 2D) is used, with the display window having the same dimensions in pixels as the desired dimensions of the image in voxels.

The near and far clipping planes are set one voxel apart. This ensures that a single voxel-thick slice of the model will be displayed.

The model is translated backwards, in one voxel steps, and the frame buffer is captured at each rendering with a black (zero) background. Wherever the plant pixels appear bit 1 of the voxel volume is set at the corresponding location in the volume (yellow in Figure S2_1).

Because some of the polygons have a plane orthogonal to the screen and therefore are not rendered, a single scan of this type is insufficient. The solution to this is to repeat the scans after rotating the model by 90 degrees. First about a vertical axis and next about a horizontal axis. The effect is a set of scans perpendicular to the X, Y and Z axes. When all three scans are completed the result is a hole-free voxelation of the model surface. Figure S2_1 (Step 1) illustrates with a single slice through the volume.

Step 2: ***From the outside-in employ a volume fill algorithm to fill all accessible voxels***

A simple queue based breadth-first filling algorithm is used. Bit 2 of each voxel byte is used to encode the fill (blue in Figure S2_1). It is essential that the seed voxel is outside of the boundaries of the hollow macrophyte model, otherwise the inside might be filled as well as the outside and the purpose is to differentiate plant tissue from water.

Step 3: ***Re-draw and voxelize the model enlarged by r***

The macrophyte model is expanded by radius r, redrawn and captured using the same method as for Step 1. Bit 3 of each voxel byte is used to encode the enlarged model (purple in Figure S2_1).

Step 4: ***Fill again from the outside this time marking the voxels in the outer shell in the enlarged model***

This fill is a slight variation of that described in Step 2. Like Step 2 it fills up to the edge of the enlarged model (bit 4 is set for these voxels). In addition, when the fill reaches voxels on the boundary of the enlarged model, bit 5 is set for those boundary voxels (these are colored white in Figure S2_1).

Step 5: ***For every boundary voxel in the enlarged model, label within radius r using a kernel***

The goal of this step is to fill all voxels within radius r of the outer boundary voxels (previously set with bit 5). To make this more efficient a three dimensional array (kernel) is constructed containing voxels labeled as inside and outside of a sphere of radius r. However the effect is the same as if we simply test and label all voxels within radius r of each outer boundary voxel.

The three dimensional array kernel has size $2R + 1$ where R is the radius r expressed as voxels. Within this kernel we set all voxels <= r from the center voxel using bit 6, otherwise they are set to zero. As mentioned previously, the main voxel volume is padded by the kernel radius and this minimizes the number of tests when applying the kernel. This is the most computationally intensive part of the algorithm when the radius is large.

Step 6: ***Count the inaccessible volume and inaccessible area voxels***

The final step is accomplished by iterating over all voxels. If a voxel is labeled with bit 2 and not bit 6, it is counted as inaccessible volume. If a voxel is labeled with bit 1 and is adjacent to a voxel labeled with bit 2 and not bit 6, it is counted as inaccessible surface area. Figures 5 and 6 show examples of the inaccessible volumes rendered in 3D for *C.f.* spp. *fragile* and *D. japonica* with different values of r.

The code was implemented in C++ and Open GL and runs on any Windows PC. It can be made available to interested persons with the necessary computer expertise. However, because it relies on custom computer graphics models of seaweeds which must be compiled into the code, it cannot be used by non-programmers. We are exploring ways of making it more readily available to non-programmers.

LITERATURE CITED IN THE SUPPLEMENTS

Chandru V, Manohar S, Prakash CE (1995) Voxel-based modeling for layered manufacturing. IEEE Comp Graph Appl 15: 42-47

Feng L, Soon SH (1998) An effective 3D seed fill algorithm. Comput & Graph 22:641-644

Marschner S, Shirley P (2015) Fundamentals of computer graphics. CRC Press