A Constructive Approach for the Generation of Underwater Environments

Ryan Abela
Institute of Digital Games
University of Malta
ryan.abela.01@um.edu.mt

Antonios Liapis
Institute of Digital Games
University of Malta
antonios.liapis@um.edu.mt

Georgios N. Yannakakis
Institute of Digital Games
University of Malta
georgios.yannakakis@um.edu.mt

ABSTRACT

This paper introduces *Coralize*, a library of generators for marine organisms such as corals and sponges. Using constructive algorithms, *Coralize* can generate stony corals via L-system grammars, soft corals via leaf venation algorithms and sponges via nutrient-based mesh growth. The generative algorithms are parameterizable, allowing a user to adjust the parameters in order to create visually appealing 3D meshes. Such meshes can be used to automatically populate a seabed or reef, in order to create a biologically realistic and aesthetically pleasing underwater environment.

1. INTRODUCTION

Procedural content generation (PCG) has been used in the game industry for decades, both for creating unpredictable, ever-changing gameplay experiences but also for speeding up game development. Although several commercial games have received acclaim for their (usually game-specific) generators, arguably the most successful application of PCG in the game industry — at least in terms of ubiquitousness of use — is *SpeedTree* (IDV 2011). *SpeedTree* is a middleware tool, compatible with several popular game engines, which is used during development to create vast, detailed forest scenes or other virtual biomes with rich vegetation. The appeal of PCG middleware tools is that they alleviate designer and artist effort for what is essentially optional content [17]; such content is taxing to create manually but only constitutes a backdrop for the scene where the actual game action takes place. In order to be as generalizable and attractive to a broad user base, PCG middleware must use sophisticated and configurable generative algorithms, ensuring the generation of both realistic (e.g. resembling real-world objects) and visually appealing artifacts which can be customized by designers to suit their needs.

Motivated by the lack of appropriate tools for and PCG studies on *underwater* ecosystems, this paper introduces *Coralize*, a middleware tool for the generation of underwater environments. *Coralize* is a plugin for *Unity3D* (Unity Technologies 2015), and comes with several algorithms for generating different marine organisms commonly found in coral reefs. *Coralize* has been built on the vision of an underwater *SpeedTree* application; in that sense it is a complete mixed-initiative co-creation tool [19] for underwater environments, with several subcomponents. This paper, however, focuses on the methods used to generate 3D meshes of hard corals (via L-systems), soft corals (via leaf venation algorithms) and sponges (via accretive growth).

The key goal of *Coralize* is to enhance the visual quality and realism of underwater scenes, in a quick yet controllable fashion. Speed is ensured by the constructive methods implemented, which often include a simplified (and computationally efficient) model of complex biological processes, while customizability is ensured by the numerous parameters which can be configured by the designer; different parameter setups result in quite visually disparate meshes. *Coralize* also comes with a user interface for the placement of generated corals and sponges in an underwater scene. Placement can be done manually, or it can simulate the real-world growth of reefs and can be impacted by water currents.

2. RELATED WORK

This section situates the generative methods implemented in *Coralize* in the context of the academic research and commercial praxis of PCG, and provides an overview of coral ecosystems as well as on the coral generation and leaf venation algorithms that the methods of *Coralize* build on.

2.1 Procedural Game Content Generation

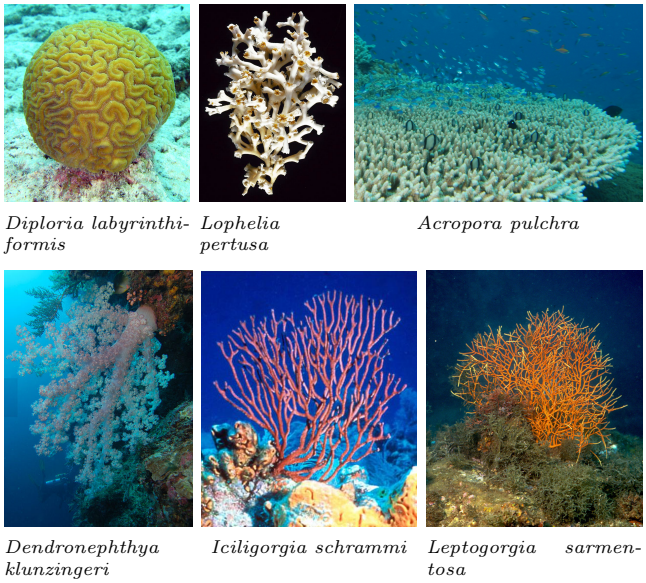
Game content has often been generated procedurally for the purposes of speeding up development and reducing the effort of artists. An example of this is *SpeedTree*, which is able to generate unique yet realistic trees (along with their 3D meshes, textures, and foliage) within minutes. *SpeedTree* has been used extensively in a plethora of large commercial games, as it relieves artists from the task of hand-crafting 3D meshes and textures of trees — a task which is cumbersome and lengthy yet of minor importance to the game's identity and style. Other tools which automate cumbersome yet trivial tasks include terrain generators, which allow for the creation of vast swathes of terrain meshes (often including realistic geological simulations such as erosion and river generation) quickly and effortlessly. Such generators are usually highly customizable, allowing the designers and artists to tweak the algorithmic parameters in order to ensure that the content produced fit the intended purpose (e.g. an island landmass) and style (e.g. an alpine treeline).

Approaching PCG as an academic field, Togelius et al. provide a taxonomy of generators, making a distinction between *search-based* approaches and *constructive* approaches [17]. Search-based approaches target a specific objective, from time spent in combat in generated shooter levels [2] to visual appeal of generated spaceships [11]. Constructive approaches use algorithms which are carefully tuned to create desirable results, but do not test the quality of those results after generation is complete; examples of such approaches include grammar-based generators of *SpeedTree* or cellular automata for generating caves [8]. The algorithms described in this paper follow constructive approaches; evaluating the quality of the generated marine organisms is beyond the scope of this project.

Another distinction made by Togelius et al. is between *necessary* and *optional* content [17]. Necessary content, such as game levels or the avatar's statistics, can not be omitted without making the game unplayable; it is important that such content are of adequate quality or at least satisfy certain constraints on playability, e.g. that the hero is able to reach the exit of a generated dungeon [10]. Optional content, on the other hand, serve a secondary purpose (e.g. as background objects) and if omitted the game experience may be poorer but still enjoyable; examples of optional content include generated textures [6] or generated rocks [5]. The corals and sponges generated by *Coralize* are similarly optional, acting as an aesthetically pleasing backdrop to an underwater setting; as optional content, the constructive methods suffice since there is no risk of rendering the game unplayable.

2.2 Coral Reef Ecosystems

Coral reefs are fascinating environments brimming with sea life and awash with striking colors, which are often found in the tropics. These reefs are home to thousands of different organisms and sea plants, making them one of the most biodiverse environments. A detailed overview on corals and reefs can be found in *The Encyclopedia of Modern Coral Reefs* [7]. As their name suggests, the defining feature of these underwater environments are corals: corals are calcium carbonate structures usually made up of layers of excrement. Calcium carbonate is excreted over time by thousands of tiny organisms called polyps. Polyp biology may vary between one family of coral to another. Polyps in tropical corals photosynthesize, requiring clear waters which are exposed to significant amounts of sunlight. In contrast, deep water corals feed by preying on zooplankton which drifts past, and can survive without sunlight. Due to the need for a steady stream of water, rich in zooplankton, water currents are an important factor for deep water corals [13]. Despite the large diversity of shape, size and types, corals can be classified into two main orders: (a) Hard corals *(Scleractinia)* such as brain, star, staghorn, elkhorn and pillar corals, and (b) Soft corals *(Alcyonacea)* such as sea fans, sea whips, and sea rods. Figure 1 shows examples of real-world corals of both orders. Sponges are also commonly found in reef systems; sponges contribute to coral growth by excreting nutrients vital to the polyps' survival. Sponges are among the simplest multicellular organisms as they lack a nervous, digestive or circulatory system. Sponges obtain food, oxygen and dispose of waste through their numerous small pores. Since sponges are very different from corals, all underwater organisms studied in this paper are referred to as *marine sessile organisms*.



Diploria labyrinthiformis *Lophelia pertusa* *Acropora pulchra*

Dendronephthya klunzingeri *Iciligorgia schrammi* *Leptogorgia sarmentosa*

Figure 1: Top: examples of hard corals (*Scleractinia*); Bottom: examples of soft corals (*Alcyonacea*). All images shown are in the public domain.

2.3 PCG for Marine Sessile Organisms

Unlike other organisms in nature such as vegetation, procedural content generation has paid little to no attention to the generation of marine organisms. While underwater environments have not been the target of procedural generation for games, there have been some previous attempts to visually model marine sessile organisms and reefs, either in 2D or in 3D.

2.3.1 Mesh Generation of Marine Sessile Organisms

In their book *"The Algorithmic Beauty of Seaweeds, Sponges and Corals"*, Kaandorp and Kübler analyzed in detail the morphology of various marine sessile organisms, focusing on how they grew and what affects their structural form [9]. Among the examples in the book, stony corals like *Pocillopora damicornis* were shown to change their structure depending on their environment: stony corals exposed to currents have a much denser structure, due to the fact that they are exposed to more nutrients.

Kaandorp and Kübler consider several different techniques to model coral and sponges, including L-systems [16], but the most successful results were obtained by the accretive growth model. This technique involves growing a 3D mesh outwards by influencing it via external environmental features like light or water current. Kaandorp and Kübler modeled nutrient distribution in fluid flow based on Lattice Boltzmann techniques [4] with impressive results. The only downside of these techniques is that they are computationally intensive and are not scalable [12] (e.g. can create a single coral at a time).

Using an approach different to Kaandorp and Kübler, Meister [12] studied the procedural generation of 3D representations of a whole reef (rather than a single coral). Studying the structure of the cold water coral *Lophelia pertusa*,

Meister derived an L-system grammar which produced 3D models visually similar to the corals he was studying. The hard coral generative process used in this paper is largely based on the grammar of [12].

2.3.2 *Leaf Venation*

For the procedural generation of vegetation, considerable attention has been given to algorithms which create the veins of leaves (leaf venation). While leaf venation may seem out of context for the purposes of marine organisms, the structure of many soft corals is similar to that of leaf veins: the algorithms in this paper for soft coral generation were inspired by leaf venation.

Leaf venation is concerned with the pattern of veins on a leaf blade. Leaf veins are used to supply water and minerals originating from the root of the plant to the leaf. While several biological theories explain the formation of leaf veins, the most recognized theory is the *canalization hypothesis* [15]. According to this hypothesis, a hormone (auxin) found in the leaf blade flows towards veins, creating a canal trail analogous to water streams carving river beds.

A class of biologically motivated algorithms have been suggested by Runions et al. [14]:

Open Leaf Venation which models open leaf veins in 2D. In these leaf patterns, veins branch out of a root seed in the middle of the leaf blade, and do not rejoin. The veins are represented as nodes in a directed acyclic graph; new vein nodes are created on each iteration by 'extending' a vein node towards the auxins which are influencing it. This is done by iterating through all the auxins and finding out the closest vein node to each source. If a vein has one (or more) sources influencing it, a new child vein node is created in the direction of the auxin(s) affecting it.

Closed Leaf Venation which creates closed leaf veins, i.e. veins growing towards the same auxin source. Runions et. al. formalize this concept by using the *relative neighborhood* [18] of the auxins.

3. GENERATING MARINE ORGANISMS

Coralize is a middleware tool for the *Unity3D* game engine for the purposes of creating underwater scenes through procedurally generated corals, sponges and other marine sessile organisms. At its current iteration, *Coralize* can procedurally generate 3 different types of marine sessile organisms: stony corals, soft corals and sponges.

3.1 Stony Corals

The generative algorithms for stony coral 3D models are largely based on Meister's grammars for *Lophelia pertusa* [12]. The self similarity observed in many hard corals makes L-systems an ideal algorithm for generating this type of structures. Towards that end, the L-system C# library[1] was modified and integrated within *Unity3D*. The L-system library creates 3D meshes via turtle graphics, with the turtle receiving the following commands: move forward (`F`), turn (`+` or `-`), pitch (`^` or `&`), roll (`/` or `\`), start a branch (`[`), end a branch (`]`) and change thickness (`#`). The commands of the core library were modified to support stochasticity in the generative process, allowing for grammars to specify an

[1]https://code.google.com/p/lsystems-csharp-lib

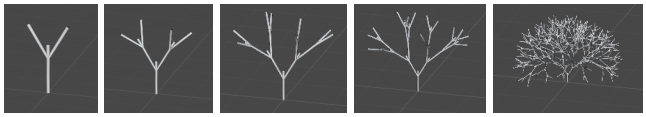


Figure 2: Stony coral growth via an L-system grammar.

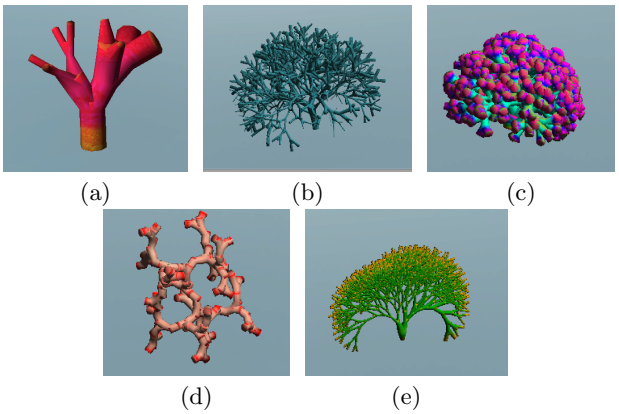


Figure 3: Different stony corals generated using the 1st (a, b, c), 2nd (d) and 3rd (e) L-system grammar.

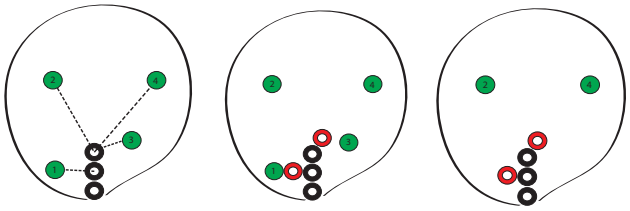
additional parameter representing *variance*. For instance, rather than `F(5)` to move the turtle 5 steps forward, the grammar can contain a `F(5,1)` command which moves the turtle between 4 and 6 steps forward. The introduction of the variance parameter ensures that while grammars produce the same coral structure, small variations make each stony coral different from the other.

The L-System grammar library outputs a string of commands after production rules are applied. A 3D mesh is derived out of this string via turtle graphics: the initial state of the cursor (turtle) is pushed in a stack, including its translation, rotation (initially [0,0,0]) and width parameters. The string of turtle commands is then parsed character by character. If the character is `F` (forward command), a horizontal cylinder mesh is constructed with the base at [0,0,0] and the top at [F_{steps}, 0 , 0]; F_{steps} is a configurable parameter (see below) and the cylinder's width is stored in the current state. The cylinder is rotated and translated to the values of the current state, and the current state is updated to match the position of the cylinder's top. In case of rotation commands (`+`, `-`, `^`, `&`,`/`, `\`), the rotation of the current state is updated, while change thickness commands (`#`) update the cylinder's width at the current state. If the character is a `[`, a copy of the current state is pushed onto the stack, while if the character is a `]` the current state is popped out of the stack. Figure 2 shows an example of turtle graphics growing the mesh as the L-system iterations increase.

Coralize allows a user to customize the generative process of stony corals by adjusting the following parameters:

Grammar: *Coralize* uses three different grammars for the L-systems of stony coral generation: two variations of the *Lophelia pertusa* grammar used in [12] and a third variation for generating flatter corals.

Iterations: the number of L-system productions performed.



(a) (b) (c)

Figure 4: Vein growth in the open venation algorithm.

Thickness: affects the thickness of certain commands.

Thickness Variance: affects the thickness variance of certain commands.

Mesh Detail: the number of edges of the 3D cylinder produced during a forward (`F`) command.

Figure 3 shows some different structures which can be created with the stony coral generation algorithm.

3.2 Soft Corals

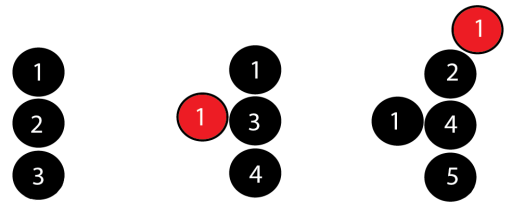
Although the biological processes which form these patterns might be different, patterns in most soft corals are visually similar to those of leaf veins. Both open and closed venation patterns of leaves (described in Section 2.3.2) can be found in soft corals. *Coralize* uses an algorithm suggested by Runions et al. [14], adapting it to work in 3D — instead of 2D lines in the original implementation.

Veins are represented as nodes in a directed acyclic graph $G = \langle V, E \rangle$ where V is the set of vein nodes (representing points of the vein) and E is the set of edges connecting these vein nodes. The root of graph G is the initial seed specified by the user. Each auxin is represented by a point in 3D space; these points are generated on each iteration at random positions within the edges of the leaf. Auxins are removed if they are near vein nodes or other auxins.

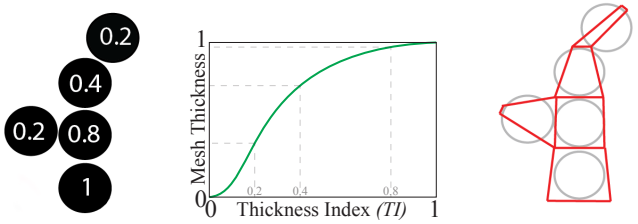
Figure 4 shows how the algorithm grows veins (black connected nodes) towards auxins (green circles): (a) for each auxin, the closest vein node is found; (b) new vein nodes (red circles) are created in the direction of influencing auxins; (c) the two auxins are too close to at least one vein node and are removed while the others remain.

Coralize extends the algorithm of Runions et al. to generate 3D meshes, allowing auxin sources to be placed in 3D space with veins growing towards them. Veins in the 3D mesh are represented as cylinders of variable thickness. Calculating vein thickness builds on the assumption that any vein that spawns a child vein becomes slightly thicker. Every vein node has a thickness index TI (initialized to 1); when a new vein node is created, all its ancestors (up to the root vein node) increase their TI by 1, as illustrated in Fig. 5a. The root always has the highest TI value. TI is translated into the width of the cylinders' bases via a process illustrated in Fig. 5: the thickness indexes are normalized (through division with the root's TI), and then are mapped to a thickness curve resulting to the actual width of each cylinder's base.

Coralize allows a user to customize the generative process of soft corals by adjusting the following parameters:

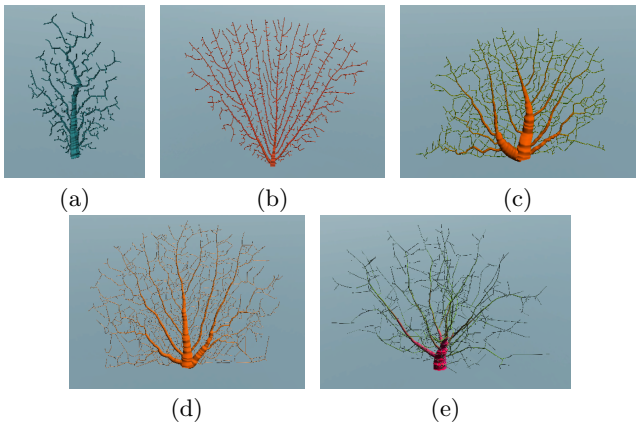


(a) Newly created nodes (red) increase the thickness index of their ancestor nodes.



(b) Normalizing TI (c) Curve for mapping TI to mesh thickness. (d) Final mesh thickness.

Figure 5: Calculating the thickness of the coral structure.



(a) (b) (c) (d) (e)

Figure 6: Different soft corals generated via open venation (a, b) and closed venation (c, d) using a a 2D box for placing auxins and using a 3D box (e).

Open Ended: specifies whether the open leaf venation algorithm (if true) or the closed leaf venation is used.

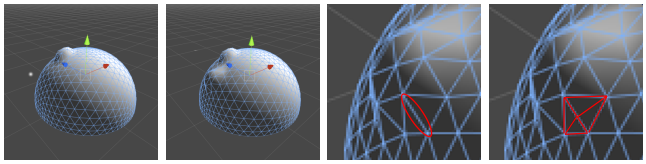
Iterations: the number of iterations of vein growth. Unlike the L-systems algorithm, the venation algorithm usually converges and after a certain number of iterations the mesh does not appear visually different.

Auxins per iteration: how many new auxin sources are generated on the leaf blade per iteration.

Initial width: the initial width of the leaf blade.

Width increment: the rate that the leaf blade grows outwards in width per iteration (note that marginal growth was used to grow the leaf).

Auxin Kill Radius: an auxin is consumed when a vein comes within the specified radius from this auxin.



(a) (b) (c) (d)

Figure 7: Sponge growth and mesh splitting process: a nutrient (white particle) approaches a vertex (a), which grows upon collision as the nutrient is removed (b). As the length of the red edge (c) is above the splitting threshold, it splits (d) to create two more vertices.

Vein Radius: the vein's growth rate in length per iteration.

Mesh detail: the number of edges in a vein's 3D cylinder.

2D: specifies whether the coral grows in a streamlined 2D structure or in a 3D box.

Thickness: a parametric curve specifying how the cylinder's mesh thickness changes from the root of the structure to the extreme leafs of the structure (see Fig. 5).

Shape: two bezier curves specifying the edges of the leaf (a 2D representation of the curved 3D resulting mesh).

Figure 6 shows some different structures which can be created with the soft coral generation algorithm.

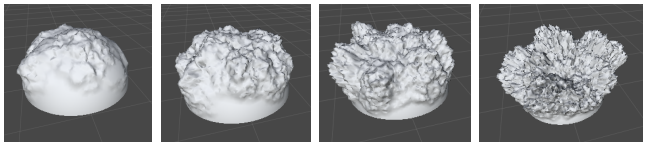
3.3 Sponges

Coralize generates sponges through a process inspired by the accretive growth model of [9], but without using a model of fluid dynamics in order to reduce computational overheads. The simplified generative process used in *Coralize* starts from a hemispherical mesh, where each vertex represents a polyp waiting to be fed. The hemisphere is bounded by a cube, and on every iteration a number of particles (or nutrients), of radius r, are spread randomly along the top surface of this cube. On every iteration, the particles move downwards; when a vertex of the hemisphere collides with a particle (using sphere collision), a nutrient is presumed to be consumed. When this happens, the colliding vertex grows outwards (based on its normal) by a fraction of its edge length to other vertices; neighboring vertices also grow outwards slightly. Neighbors' growth is controlled by a normal function of the distance, so the further away from the colliding vertex, the less the neighboring vertex grows outwards. Figure 8 illustrates the growth process as the iterations increase. As in [9] the mesh is optimized after each vertex growth. Once a vertex grows, the system checks whether its edges exceed a split threshold; if they do, the edge splits in half, creating another vertex (see Fig. 7). If two edges are smaller than a collapse threshold, they collapse into one by removing a vertex.

Coralize allows a user to customize the generative process of sponges by adjusting the following parameters:

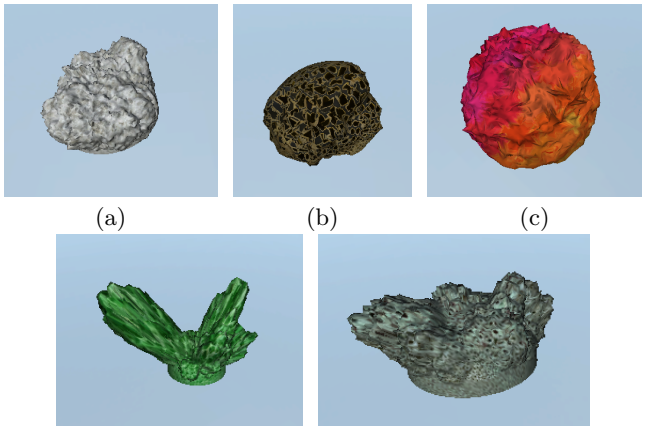
Iterations: the number of iterations of sponge growth.

Feeding iterations: the number of neighboring points which grow along the vertex colliding with a nutrient.



1 iteration 4 iterations 10 iterations 25 iterations

Figure 8: Sponge growth as iterations increase.



(a) (b) (c) (d) (e)

Figure 9: Different sponges generated in *Coralize*.

Splitting threshold: the edge threshold length, above which it splits in two.

Collapse threshold: the length of two adjacent edges, below which they collapse into one.

Nutrients per iteration: the number of particles generated at the cube's top edge per iteration.

Nutrient radius: the radius of each particle (particles are spheres).

Particle feed: the upper limit of points (of the sponge mesh) that the nutrient can collide with before it dies.

Nutrient lifetime: the number of iterations a nutrient can survive.

Mesh detail: the initial hemisphere's number of vertices.

Figure 9 shows some different sponges which can be created with the sponge growth algorithm.

4. POPULATING THE REEF

The generative algorithms of *Coralize* can be used to populate a reef, allowing a level designer to create an underwater environment. The user can generate corals and sponges, adjusting the algorithms' parameters as described in Section 3 and assign materials to them, thus adding color to the meshes. *Coralize* is accompanied by several textures appropriate for marine sessile organisms; users can also create custom materials using the *Unity3D* material editor. The generated 3D assets are added to a *coral pool* accompanied by the parameter set used to generate them and any materials selected.

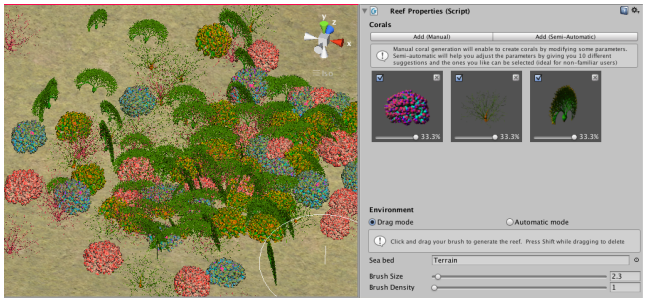
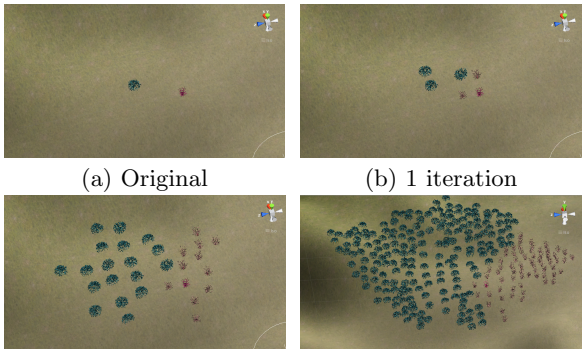


Figure 10: Interface for adding corals and sponges to a seabed mesh using a brush (left); the coral pool is shown on the right.



(a) Original (b) 1 iteration

(c) 5 iterations (d) 10 iterations

Figure 11: Iterations of reef growth, starting from two corals in Fig. 11a. Reef growth results in "territories" of corals.

4.1 Brush Functionality

The user interface of *Coralize* offers a brush functionality (inspired by the *Unity prefab brush*[2]), which enables the user to create copies of the 3D meshes over a specified unity terrain (or any other unity game object) simply by dragging the mouse over it (see Fig. 10). Dragging the mouse over a designated mesh which acts as the seabed causes randomly chosen corals or sponges from the user's coral pool to be placed at random points within the brush's radius. The brush adjusts the height and orientation of the new assets, so that they follow the topology of the terrain. The user can control the distribution and variation of the added organisms by customizing the following brush parameters:

Brush Size: the area around the brush populated with corals.

Brush Density: how many corals are instantiated on each drag/click event.

Coral Random Bias: a percentage specified per coral, indicating the likelihood of a particular coral being selected from the random pool.

Coral Orientation and Rotation: orientation can be forced not to follow the terrain's topology, and any axis can be set to rotate randomly.

Coral Size: a range which will be used to randomly scale a coral.

The *standard* brush of *Coralize* instantiates the same materials and meshes found in the coral pool, although it can rotate or scale them at random to provide the semblance of variation. The repetition of the same meshes in the coral pool allows for more designer control over the appearance of the corals and sponges and is faster to design with (less latency as no new organisms are generated) and when rendering (as one mesh detail needs to be stored).

In order to provide more variation in the seabed environments, the *automatic brush* allows for new meshes to be generated and placed on the seabed. Similar to the standard brush, upon dragging the brush over a mesh designated as a seabed, corals and sponges from the coral pool are selected; variants of these corals and sponges are generated, using the same values of their generative parameters (e.g. thickness, iterations, etc.) and applying the same material as the original meshes. Since generating new meshes may be computationally demanding, the brush initially places placeholder cubes on the seabed, replacing the cubes with the new meshes when their generation is complete. The automatic brush, while slower to work with due to the waiting time for new meshes to be generated, creates more interesting and diverse scenes; the designer's control over the scene's appearance is still retained to a large degree, since newly generated meshes still use the same parameters and materials specified by the user.

[2]https://www.assetstore.unity3d.com/en/content/21321

4.2 Growing the Reef

Beyond designating the areas populated by corals and sponges, *Coralize* allows for more realistic placement of these organisms which simulates reef growth. Competition for space among marine sessile organisms is a common occurrence that we find in coral reef ecosystems [3]. Reef growth is modeled in *Coralize* using a simple algorithm which spreads an initial set of marine organisms along the sea bed. Once a set of corals or sponges has been placed (e.g. using the automatic brush), the user can choose to grow the reef: every time the growth command is issued, for each organism the neighboring positions along the four cardinal directions are checked. For each neighboring position which does not have another organism nearby, another instance of the same organism (generated using the same parameters and materials as per the automatic brush) is placed on the seabed. Figure 11 illustrates how corals expand as growth iterations increase. This algorithm also takes into consideration the depth (derived after the user specifies the sea plane), and only grows corals in depths between 50cm to 20m below sea level. In future versions of *Coralize* this depth will be specified per coral type.

4.3 Using Water Currents

Coralize also allows the placement of water currents, which the level designer specifies as arrows on the scene, along with the current's strength and nutrition range. Since real-life sponges and corals rely on nutrients to be delivered through water streams, marine organisms exposed to a water current tend to have a denser and stronger structure than those in sheltered areas [9]. *Coralize* currently affects the generated corals' thickness parameter, based on their proximity to the water current; future work can explore more ways of visu-

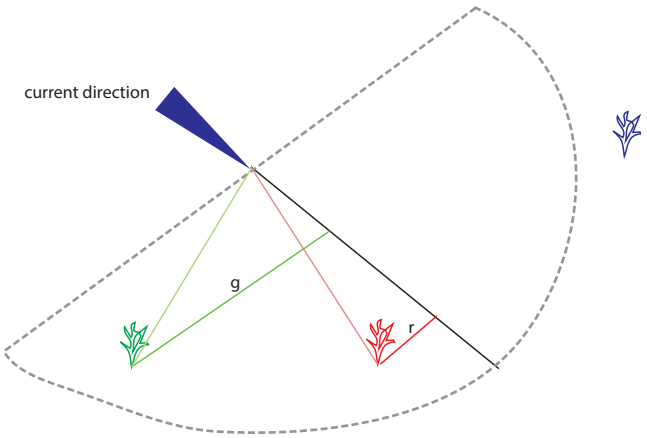


Figure 12: The water current model, with a hemisphere of radius equal to the current's strength deciding which corals are affected. The red coral is closer (based on r) to the current's ray than the nutrition range and is fully exposed; the green coral is not within nutrition range and is therefore exposed less (by a factor inversely proportional to its distance from the current's ray, i.e. g); the blue coral is outside the hemisphere and is not exposed at all.

alizing a denser structure in both corals and sponges. Each coral traces a ray to the source of the water current; in case there is a collision with the seabed then the coral is in a sheltered area and thus not exposed to the current. The current considers non-sheltered corals within a hemisphere, the radius of which is equal to the current's strength. Calculating exposure to water currents is illustrated in Figure 12. In short, corals within the current's nutrition range (with distance calculated from the water current's ray) are fully exposed to the current. Corals not within the current's nutrition range (but within the hemisphere) are exposed less; the further away such a coral is from the current's ray, the less it is exposed. Corals outside of the hemisphere, or in sheltered areas, are not exposed. The more exposed a newly added coral is to a water current, the higher its thickness parameter in the generative algorithm (see Fig. 13). Since previously placed corals are not regenerated, this necessitates that the level designer places water currents before using the brushes to add corals. If a coral is exposed to multiple currents, the growth effect is aggregated resulting in even thicker corals.

5. DISCUSSION & FUTURE WORK

Coralize incorporates a number of generative algorithms for generating several types of marine sessile organisms appropriate for populating a biologically realistic underwater scene in a game or virtual world. The processes are highly parameterizable by the user, allowing for a broad expressive range in the visual appearance of results as evidenced by Fig. 3, 6 and 9. Using popular constructive PCG methods such as L-systems and biological simulations such as leaf venation and sponge growth, *Coralize* allows for the quick generation of 3D meshes for hard corals, soft corals and sponges by simplifying and streamlining the algorithms

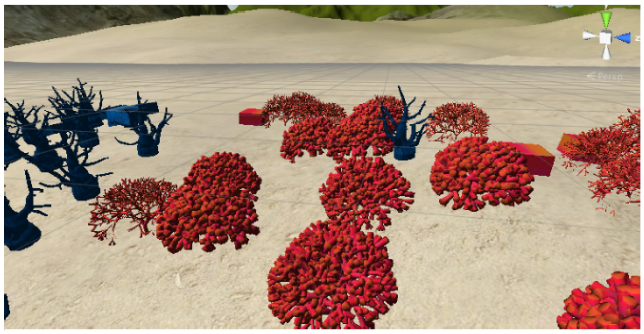


Figure 13: Corals along the water current (right) are thicker than those further away (left). The cubes shown here are placeholder elements from the automatic brush, and are replaced with corals once those are generated.

(with a controlled loss of biological and visual accuracy).

Coralize also provides a graphic user interface in *Unity3D* for adding the premade or newly generated marine sessile organisms into a 3D scene. Additional features such as reef growth and water currents create a more realistic, if less controllable, reef appearance, while traditional brushes (including an automatic brush which creates new meshes for each instance) allow for more designer control over the placement of corals and sponges.

During the development of *Coralize* and the implementation of its generative algorithms, it was soon obvious that one cannot possibly model all the structures of organisms using one PCG technique. Despite the different techniques used currently in *Coralize*, there are many more types of corals and marine sessile organisms which cannot be generated by e.g. adding another L-systems grammar. Although one can find lots of similarities in marine sessile structures, each family of organisms needs to be studied individually in order to design and implement a technique for modeling its processes and generating its structure algorithmically. For instance, hard corals such as *Diploria labyrinthiformis* (see Fig. 1) can arguably not be modeled via L-systems due to their compact nature. Future work in *Coralize* will attempt to refine existing generative techniques and explore new methods to create more varied types of marine sessile organisms found in nature.

Future work on *Coralize* includes improvements of the existing generative techniques on many levels. To start with, several optimization methods can be applied to the generative algorithms to speed up mesh generation and reduce the number of vertices which need to be rendered in the final underwater scene. In particular, Delaunay triangulation and Voronoi diagrams can be used, as suggested by [14], to dramatically increase the speed of leaf venation algorithms. The sponge generation algorithm can be enhanced through the use of octrees for collision detection with nutrients, instead of iterating through all the vertices of the mesh; this would make collision detection possible in logarithmic time and increase the speed of generating sponges. While the L-system algorithm is very fast when generating stony corals, the appearance of the generated mesh could be greatly improved by applying a smoothing algorithm in the branch intersections. An inspiration comes from *SpeedTree* which

uses a welding method to modify the geometry of a branch mesh to intersect seamlessly with another branch.

Another area of improvement is the materials of the generated meshes. Currently *Coralize* uses materials only for the purpose of coloration; however, materials could be exploited to generate more realistic corals. Bump maps can also be used to create the bumpy effect which is usually found in most corals. Moreover, certain corals (e.g. the *Diploria labyrinthiformis* brain coral) could be modeled simply by placing a bump map with the coral's pattern over a smooth hemisphere. The potential of using similar algorithms to those presented in this paper to generate the textures of such materials will be explored in future work.

Finally, the reef creation methods can be enhanced by modifying the seabed mesh based on the amount of *Scleractinia* (reef building) corals that are present; this is already explored in the literature [1]. A more precise water current model can also be implemented using fluid dynamics, ensuring that sheltered corals may still exhibit some growth depending on the topology of the seabed mesh. The computational overhead of such algorithms is considerable, but it may be alleviated somewhat via a pre-rendering step which calculates water movement in a particular area of the reef.

6. CONCLUSION

This paper introduced *Coralize*, a tool which generates 3D meshes of realistic marine sessile organisms. The parameterizable generative algorithms integrated in the tool allow for a diverse set of stony and soft corals and sponges to be generated and placed on a scene. The constructive algorithms used (L-systems, leaf venation and sponge growth) produce new high-detail meshes quickly and in a controllable manner. This allows a level designer to populate underwater scenes with unique, visually appealing objects without needing to buy, hand-craft or re-use 3D assets.

7. ACKNOWLEDGEMENTS

The research was supported, in part, by the FP7 ICT projects C2Learn (project no: 318480) and ILearnRW (project no: 318803), and by the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665).

8. REFERENCES

[1] H. Bosscher and W. Schlager. Computer simulation of reef growth. *Sedimentology*, 39(3):503–512, 1992.

[2] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi. Evolving interesting maps for a first person shooter. In *EvoApplications (1)*, pages 63–72, 2011.

[3] N. E. Chadwick and K. M. Morrow. Competition among sessile organisms on coral reefs. In *Coral Reefs: an ecosystem in transition*, pages 347–371. Springer, 2011.

[4] S. Chen, Z. Wang, X. Shan, and G. D. Doolen. Lattice Boltzmann computational fluid dynamics in three dimensions. *Journal of Statistical Physics*, 68(3-4):379–400, 1992.

[5] I. M. Dart, G. De Rossi, and J. Togelius. Speedrock: procedural rocks through grammars and evolution. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, 2011.

[6] D. S. Ebert, F. K. Musgrave, and D. Peachey. *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 2002. 3rd Edition.

[7] D. Hopley. *Encyclopedia of Modern Coral Reefs: Structure, Form and Process*. Springer, 2011.

[8] L. Johnson, G. N. Yannakakis, and J. Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the Workshop on Procedural Content Generation in Games*. ACM, 2010.

[9] J. A. Kaandorp and J. E. Kübler. *The algorithmic beauty of seaweeds, sponges and corals*. Springer, 2001.

[10] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius. Procedural personas as critics for dungeon generation. In *Proceedings of Applications of Evolutionary Computation*, 2015.

[11] A. Liapis, G. N. Yannakakis, and J. Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, 2012.

[12] M. Meister. *Interactive Visualization in Interdisciplinary Applications*. PhD thesis, University of Erlangen-Nuremberg, 2008.

[13] A. Rogers. *The biology, ecology and vulnerability of deep-water coral reefs*. International Union for Conservation of Nature, 2004.

[14] A. Runions, M. Fuhrer, B. Lane, P. Federl, A.-G. Rolland-Lagan, and P. Prusinkiewicz. Modeling and visualization of leaf venation patterns. In *ACM Transactions on Graphics*, volume 24, pages 702–711. ACM, 2005.

[15] T. Sachs. The control of the patterned differentiation of vascular tissues. *Advances in botanical research*, 9:151–262, 1981.

[16] J. Togelius, N. Shaker, and J. Dormans. Grammars and L-systems with applications to vegetation and levels. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. 2015.

[17] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[18] G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern recognition*, 12(4):261–268, 1980.

[19] G. N. Yannakakis, A. Liapis, and C. Alexopoulos. Mixed-initiative cocreativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games*, 2014.