Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Procedural generation and real-time rendering of a marine ecosystem*

Rong LI[1], Xin DING[1], Jun-hao YU[2], Tian-yi GAO[1], Wen-ting ZHENG[†‡1], Rui WANG[1], Hu-jun BAO[1]

([1]*State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310027, China*)

([2]*PLA Unit 61741, China*)

[†]E-mail: wtzheng@cad.zju.edu.cn

Received Dec. 1, 2013; Revision accepted Feb. 18, 2014; Crosschecked June 16, 2014

Abstract: Underwater scene is one of the most marvelous environments in the world. In this study, we present an efficient procedural modeling and rendering system to generate marine ecosystems for swim-through graphic applications. To produce realistic and natural underwater scenes, several techniques and algorithms have been presented and introduced. First, to distribute sealife naturally on a seabed, we employ an ecosystem simulation that considers the influence of the underwater environment. Second, we propose a two-level procedural modeling system to generate sealife with unique biological features. At the base level, a series of grammars are designed to roughly represent underwater sealife on a central processing unit (CPU). Then at the fine level, additional details of the sealife are created and rendered using graphic processing units (GPUs). Such a hybrid CPU-GPU framework best adopts sequential and parallel computation in modeling a marine ecosystem, and achieves a high level of performance. Third, the proposed system integrates dynamic simulations in the proposed procedural modeling process to support dynamic interactions between sealife and the underwater environment, where interactions and physical factors of the environment are formulated into parameters and control the geometric generation at the fine level. Results demonstrate that this system is capable of generating and rendering scenes with massive corals and sealife in real time.

Key words: Procedural generation, Marine ecosystem, Biological feature, Graphic processing unit acceleration
doi:10.1631/jzus.C1300342 **Document code:** A **CLC number:** TP391

1 Introduction

A marine ecosystem is extremely complex in nature and plays a very important biological role in the Earth's environment (Mann and Lazier, 2005). Graphic or scientific applications require generating, simulating, and visualizing underwater environments realistically and very quickly. However, the diversity and complexity of underwater sealife have presented great challenges. First of all, sealife on a seabed must be distributed in a realistic manner so as to reflect their interactions with their environment. Second, geometric models of individual sealife must be synthesized with unique biological features in both shapes and topological structures. Third, to create believable and visually pleasant underwater scenes, dynamic interactions between the sealife must be considered. Finally and what is one of the biggest challenges, the scene, which may consist of billions of primitives, must be modeled and rendered efficiently.

In recent years, procedural modeling algorithms have been widely used in modeling vegetation on land (Deussen *et al.*, 1998) and for cities (Parish and

‡ Corresponding author

* Project supported by the Zhejiang Provincial Natural Science Foundation of China (No. LY13F020002), the National Natural Science Foundation of China (No. 61272301), the National Key Technology R&D Program of China (No. 2012BAH35B03), and the Fundamental Research Funds for the Central Universities, China

Fig. 1 Three views of an underwater scene. This scene consists of 128k corals, seaweed, and sealife (over 100M triangles), which are generated and rendered by the proposed system at above 30 frames per second

Müller, 2001). Such a technique has the advantage of creating complex but structural models at runtime, whereas no geometric needs are required to be stored or streamed beforehand. Compared with previous procedural modeling models, underwater sealife are different in both shapes and topological structures. Many underwater sealife are characterized by thorny skin and full detailed tips, which are rare in vegetation or buildings. To our knowledge, the procedural generation of underwater sealife remains an open problem.

In this study, we propose an efficient system that addresses the different challenges in the modeling and rendering of a marine ecosystem.

First, we propose a marine ecosystem simulation model in the proposed system to populate sealife in the ocean more naturally. The growth of species is simulated by considering different factors such as competition for space and sunlight, preference for wave energy, aging, death, and so on.

Second, we present a central processing unit-graphic processing unit (CPU-GPU) hybrid framework to the proposed procedural model for rendering the individual ocean sealife. A set of grammars are designed to define the very basic shapes of several underwater classes at a general level. Unique appearances like spiny skin in Echinoderms and sensory tentacles in Coelenterata are generated in different patterns to include the diversities identified in the undersea sealife. The proposed CPU-GPU hybrid framework has several good features:

1. Compact model representation on CPUs: Grammars are procedurally encoded to generate an individual model. Avoiding explicit geometry storage, this model is a rough representation of base meshes and tip nodes.

2. Fast generation of details on GPUs: The rough model is expanded to a detailed model at runtime by using hardware tessellation control and optimized instancing. Adaptive level-of-detail (LOD) geometries are created entirely on a GPU to generate massive sealife with different levels of geometric details in an underwater scene.

Third, the proposed system integrates dynamic interaction simulation with the underwater environment. Complex physical factors are simplified into a local model based on control points. The water flow and fish interaction are computed based on this model on a CPU. Swaying influences and collision results are sent to the GPU as parameters for geometry generation.

Finally, we use the proposed system to generate a complex marine ecosystem. Results demonstrate that this system is capable of generating and rendering scenes with massive corals and sealife in real time (Fig. 1).

2 Related work

Procedural generation plays an important role in vegetation modeling. L-systems (Lindenmayer, 1968) have been widely applied to both model and render plant ecosystems (Deussen *et al.*, 1998). The plant distribution is modeled on the basis of a fundamental phenomenon in plant population dynamics, i.e., self-thinning. In marine ecosystems, however, several other key factors affect the development of spatial distribution of underwater sealife, including preference for wave energy and competition for sunlight and space (Storlazzi *et al.*, 2002).

In modeling of individual plants, L-systems can be used to observe real plants (Prusinkiewicz and Lindenmayer, 1990). A tree can be further characterized into stem and leaf parts by using several positional functions (Weber and Penn, 1995). Levels of details can also be generated (Lluch *et al.*, 2003). The main benefit of L-systems is that each tree branch can be considered as a smaller complete tree, which makes the whole modeling process a single recursive step. Underwater sealife such as coral have devel-

oped into complex shapes and most of them have grown complex tips unlike their main bodies. It is hard to implement the original tree grammars to those shapes except for the tree-like ones. Meanwhile, full detailed geometry generation in this way can easily generate a large number of polygons of an individual model. The approach proposed by Marvie *et al.* (2012) introduces construction rules which can expand at runtime using hardware geometry generation. This approach can easily generate a large number of polygons and render complex scenes containing thousands of buildings and trees. However, it focuses on representing facade elements, but ignores a lot of unique biological features including spiny skin and full detailed tips.

Underwater illumination has become a research hotspot (mostly in the domain of particle and ray tracing algorithms) (Jensen, 2001). Meanwhile, real-time solutions have been devoted to classic underwater effects, such as caustics and godrays (Lanza, 2007; Papadopoulos and Papaioannou, 2009). Sakude *et al.* (2011) introduced an approach to modifying the graphic pipeline based on the programmable shading language to properly render the effects of the liquid environment, including loss of illumination and visibility, blur effects, etc. Unfortunately, there are very few methods for the modeling of underwater sealife. Large-scale underwater scenes with endless corals used in games and movies are mainly designed by artists. A common way for artists to construct a complex scene is to clone a single object many times to achieve the geometric complexity for the required realism. On a natural seabed consisting of many coral models, this common approach will result in a manufactured unnatural appearance, producing the same coral models and often the same branching structures. Besides the endless corals, thousands of animals are living on the seabed such as fishes and crabs. These animal-coral interactions bring about great challenges. Traditional procedurally generated vegetation scenes like forests are almost static except for a wind-like swaying. These wind animations for trees are usually based on a repeated function implemented by a vertex shader (Zioma, 2007). A real-time method is presented to animate thousands of trees under a user-controllable directional wind (Diener *et al.*, 2009). These methods are not suitable for a simulation of local dynamic scenes such as the classic interaction scene of clown fishes swimming in a giant sea anemone. This type of interaction requires the use of complex physical models to describe what is taking place. How to simplify this physical model to achieve convincing results at low cost is a great challenge.

3 Overview

The architecture of the proposed procedural modeling and rendering system is presented in Fig. 2. We propose a hybrid CPU-GPU framework to generate underwater scenes naturally and render them efficiently. *Xenia umbellata* is a typical underwater coral characteristic of pumping branches and spiny main bodies, which we will use as an example to describe our pipeline approach.

The modeling process begins with an ecosystem simulator that positions sealife on the seabed.

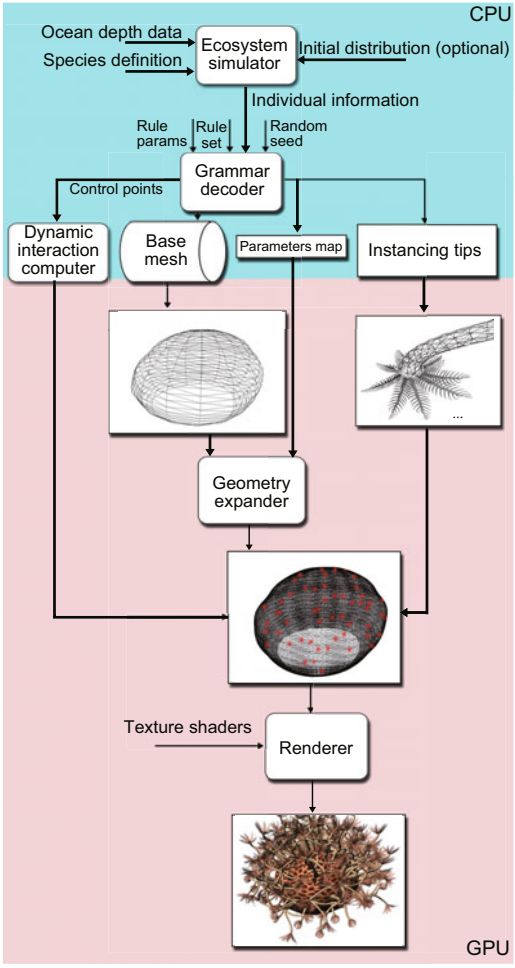

Fig. 2 Architecture of the procedural modeling and rendering system

An ecosystem simulation model is implemented to synthesize interactions between the sealife within the marine ecosystem. Taking the ocean depth data and the ecological characteristics of the species and, optionally, the initial distribution of the species as input, our simulator computes the distribution and growth of the sealife by considering the competition of the sealife for space and sunlight, preference for wave energy, aging, death, and so on. These simulations are performed using an L-system-based model, extended with the capability to simulate species-environment interactions. Given the type, position, and vigor parameters of individual sealife, we present a two-level procedure modeling and rendering pipeline system. At the initial rough level, a grammar decoder generates the base mesh, parameter map, and instancing tips for the detailed generation, as well as the control points for the dynamic interaction simulation. This process is computed on the CPUs to fully use the sequential and asynchronous computational power of CPUs. Then, based on such information, on a GPU, additional details are created with seamless LOD transitions by utilizing hardware tessellation and optimized instancing technologies. The GPU stage is designed to take advantage of massively parallel graphics hardware and expand undersea sealife into a fine level in the geometry expander. Finally, geometric and texture data is developed in the renderer.

Optionally, in a dynamic generator, the interactions between water flow and sealife are computed by a simplified physical model. The positions of the control points are computed to control the dynamic movement of the sealife. The results are sent to a GPU as parameters for generating dynamic geometry at the fine level.

The proposed system follows the basic principles initiated by Deussen *et al.* (1998). The process can be decomposed into several stages: specification of undersea sealife distribution, modeling of individual sealife, computing dynamic interaction with the environment, and finally development of the entire scene. Each stage operates at a specific level of abstraction, and provides a relatively high level input for the next stage. In this way, different stages can be computed efficiently and are well distributed on the CPU and the GPU.

4 Specification of sealife populations

The simulation and visualization of plant ecosystems on land has many theoretical and practical applications, but the studies on general simulation models of marine ecosystems are few. Considering the phenomena of self-thinning, plant succession, and plant propagation, a series of L-system based methods have accomplished a great deal of success in generating the spatial distribution of plants (Wonka *et al.*, 2011). However, factors relevant to underwater sealife distribution are more complicated. Apart from the resources competition between sealife, wave energy and sunlight attenuation play an important role in marine ecosystems (Storlazzi *et al.*, 2002).

4.1 Simulation model

According to fundamental research on marine ecosystems in ecology (Storlazzi *et al.*, 2002; Bryan and Metaxas, 2006), the underwater sealife distribution is simulated by considering these factors:

1. Competition: Sealife that can grow close in space need to compete for living resources, and the loser is inhibited and may disappear. This is the same as the self-thinning of plants on land (Wonka *et al.*, 2011).

2. Wave energy: Sealife that grow on a shallow seabed (depth < 3 m) are affected by wave energy. A smaller depth means more energy is dissipated by the seabed and the sealife, and thus more sealife are influenced by these actions. The wave energy is simulated by the wave model (WAM) (Storlazzi *et al.*, 2002), including a wave friction factor (Nielsen, 1992) and the peak bed shear stress under an oscillatory flow (Jonsson, 1966). The correlation between wave energy and seabed depth is shown in Fig. 3.

3. Sunlight attenuation: As sunlight is attenuated into the deep environment, less energy can be gained by the sealife in deep oceans, which determines the type of species and the density of the corals.

4. Attributes of species: Attributes are different among species. For example, Montipora Coral, the dominant species in shallow seabeds, can endure wave pressure, while high energy branching and encrusting species like *Pocillopora meandrina* and *Porites lobata* tend to live only with sufficient sunshine (Storlazzi *et al.*, 2002).

Considering these factors, a factor-driven model

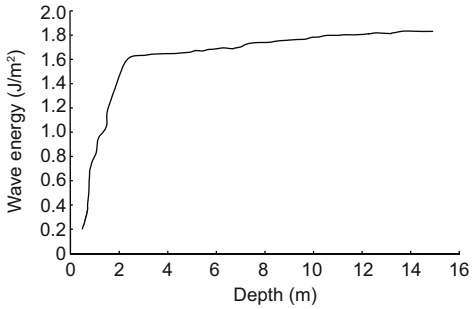

Fig. 3 Correlation between wave energy and seabed depth. A smaller depth means more energy is dissipated on the seabed

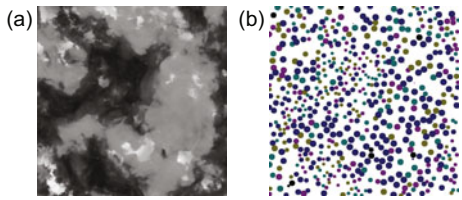

Fig. 4 In the input depth map (a), a darker color means deeper seabed. In the simulation result (b), different colors represent different species, while the size and density of the circles denote the growth situation. For example, the blue points represent *Pocillopora meandrina*, which lives only with sufficient sunshine (Note: references to color refer to the online version of this figure)

is constructed to simulate the marine ecosystem. In this model, each sealife is assigned with a growth factor, which indicates how well the sealife can grow and compete in this environment. The growth factor decreases when more wave energy is dissipated on the sealife, and it increases when sufficient sunlight can be received by the sealife. The growth factor of a species is specifically sensitive to wave energy and sunlight, depending on its attributes and tendencies within the environment. A larger growth factor means that sealife can grow faster and larger, and is therefore more competitive for living resources.

4.2 Simulation procedure and results

The simulation begins with a depth map as the input (Fig. 4a). Initially, different classes can be randomly spread or manually located on the seabed. Then the location of each sealife, associated with its depth, determines the growth factor. At each growth step, sealife expand or shrink as indicated by the growth factor. When sealife shrink to a limited size, they will disappear in the next step. When competition occurs, the sealife with a smaller growth factor loses and will gradually disappear. After several growth steps, the species are selected by the environment, and different densities of classes can be seen on different depths of the seabed. The result of simulation is shown in Fig. 4b.

5 Two-level procedural generation of individual underwater sealife

A traditional procedural pipeline is usually based on decoding a specific grammar iteratively, which causes a high computation overhead and creates too many triangles for detail (Fig. 5a). Generation of a full detailed tree can be processed in hundreds of milliseconds and even seconds. Thus, it is not suitable for a dynamic walk-through system without duplication. At least 100 000 known undersea species have been found worldwide. It is hard to summarize topology into a uniform grammar. In contrast, a series of grammars are designed to describe the very basic shape of several types of sealife. The cost of a CPU grammar decoder is low. The biological features are refined in the parameter map and tip nodes (Fig. 5b), which will be produced in the GPU for a finer geometry.

5.1 CPU grammar decoder

At a base level, the CPU grammar decoder converts the growth of an individual sealife into a rough representation. This representation creation must be fast as the user wants to start exploring the seabed without delay, but still all the information for refining the biological feature has to be generated. This process consists of three steps: first, we generate a base mesh of the main branch of an individual sealife; second, we produce a parameter map and a batch of tip nodes that can be reused for this species; finally we feed an appropriate number of base meshes into a render batch. By this step-by-step approach, the sequential and asynchronous computational power of CPUs is fully used.

5.1.1 Generation of base mesh and control points

A uniform grammar is unable to summarize the diversified sealife. Instead, a series of grammars are designed to describe a specific phylum of marine animals. Take Asteroidea, a class belonging to Echinoderms (Lawrence, 1987), as an example.

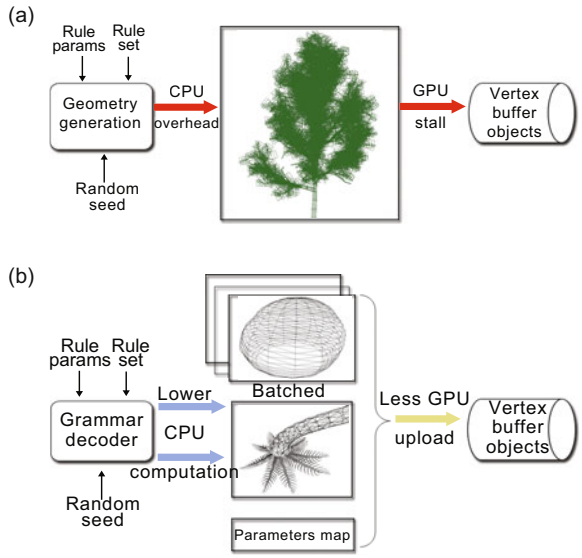

Fig. 5 (a) Classical tree generation pipeline in which the CPU grammar decoder creates coarse representation for GPU to generate details; (b) The CPU grammar decoder for *Xenia umbellata* with lower CPU computation overhead and upload stall

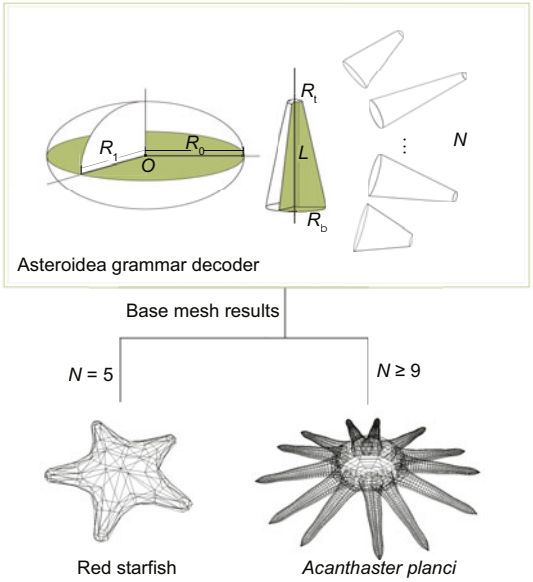

Fig. 6 Different species decoded by the same Asteroidea grammar with different seeds

The generation of a starfish is described as follows (Fig. 6). Since a starfish has a central disc and several arms, we will first create its major part. To build the body, an ellipsoid is created with point O as its center. Both major axis R_0 and minor axis R_1 of the ellipsoid are constrained to approximate a circle when viewing from the top. Totally N circular truncated cones are then extruded from the surface of the ellipsoid equidistantly to simulate the arms of the starfish. Radius of the top surface (R_t), radius of the bottom surface (R_b), and length of each circular truncated cone (L) can be adjusted according to different patterns to add to the diversity of this species. Grammars of other sealife are different from Asteroidea and will not be described in such depth. Meanwhile, within the species own soft branches, several control points are created from the beginning position to the ending position according to the length. These points are inputs for the dynamic interaction described in Section 6. All shape grammars are quite simple in order to perform a very basic mesh of the specific species.

5.1.2 Parameter map and tip nodes

Most sealife have skeletons made of bony plates (Castro and Huber, 2012). Some of the plates have spines that stick out to give these animals their spiny skin. This type of skin widely exists in Echinoderms (Lawrence, 1987), which all have a hard, spiny, or bumpy endoskeleton covered by a thin epidermis. We encode the details into a parameter map shared in the cluster to describe this feature of biological appearance. The parameter map is wrapped onto the surface and matches the texture (Fig. 5b), which can create adaptive geometric details at the GPU level.

Sealife in a Coelenterata are characteristic of sensory tentacles surrounding their main bodies. These tentacles are called moving tips. These tips are full of details that cannot be handled based on a texture method such as a canopy in vegetation. Each tip node is made of dozens to hundreds of polygons to achieve geometric details, which can easily cause triangular size overhead in both computation and rendering. We generated a branch of tip nodes shared in a cluster for instancing rendering on a GPU to lower the cost.

Applying these two features with individual base meshes allows us to fit the specific biological feature and makes the topology of the scene much more complex and realistic.

5.1.3 Batching

Modern GPUs can render a large number of triangles in real time as long as we divide them into reasonable batches (Wloka, 2003). Because a single

base mesh consists of hundreds of triangles at most, hundreds of them are packed into one draw call unit. After passing through view frustum culling, multiple objects in this unit can be generated and rendered at almost the same cost as a single object.

5.2 GPU geometry expander

After uploading the base mesh, the parameter map, and tip instance data, we carry out the geometry generation and rendering entirely based on a GPU scheme. At this fine level, additional details including spiny skin and full detailed tips are created procedurally in the adaptive LODs.

5.2.1 On-the-fly LOD generation on GPU

We propose an adaptive LOD approach entirely based on programmable hardware tessellation. In hardware tessellation, the compact representation and data independence of patch primitives are exploited to produce triangular data for immediate consumption at the rasterization stage. This means that only the vertices of a rough model need to be pre-computed into the CPU phase, while the GPU can amplify the rough geometry on the fly with very little memory bandwidth to produce a dense tessellation on the surface. The spiny skin (Fig. 7b) in Echinoderms is suitable in this way.

A two-phase process is used for each frame rendered on the GPU. In the first phase, a frustum culling test is executed on the tessellation control shader to avoid unnecessary subdivision. After passing, a factor is computed to change the tessellation density of a patch at runtime. This factor can be calculated by approximating the projection of the local patch to the near plane. In this way, the cost of object over-tessellation can be saved. At the second phase of the tessellation evaluation shader, the hardware tessellator unit is used to construct a Bezier surface to replace the original rough patch in the base mesh in order to smooth out a high polygon mesh. The parameter map is decoded in this phase to offset a local surface patch to obtain a spiny skin. The parameter map is encoded into a texture including offset direction and scalar. The point-normal triangles with adjacent edge normals (PN-AEN) model (McDonald, 2011) is used to create crack-free surfaces.

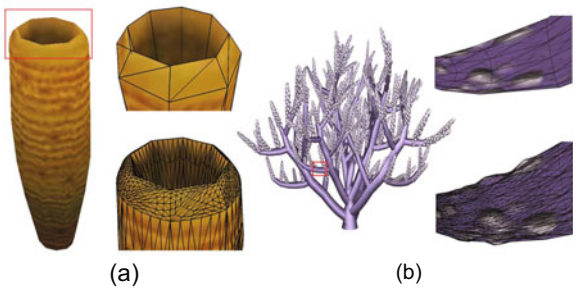

Fig. 7 (a) Further details are generated based on GPU tessellation to the smooth silhouette; (b) Create a spiny skin

The highest LOD substitute geometry is based on image-based impostors (Jeschke *et al.*, 2005). The classical way to create impostors is at the object level, but we use the batch in Section 5.1.3. The impostors are generated automatically by a GPU evaluation and simultaneously rendering the batch from several different views.

In this GPU-based approach, LODs are blended for seamless transitions. Smooth silhouettes (Fig. 7a) and lighting are generated. Since only the local subdivision is considered, the time and memory requirements are significantly lower than those in the traditional CPU scheme.

5.2.2 Two-pass optimization of instancing

Sealife in Coelenterata can have detailed tips like the pumping Xenia. As a single creature can develop hundreds of tips, a local cluster of this species possesses millions of orientations and scales. A hardware instancing technique (Carucci and Studios, 2005) is extremely suitable for such a situation. Unfortunately, the applicability of geometric instancing is strongly limited by several factors. One key factor is the culling of the instanced geometries. Sending the whole bunch of instances down the graphics pipeline may choke the vertex processor because of the high-poly tips and a large number of instances. Hierarchical culling techniques on a CPU usually break the batch down and reduce the benefits of geometric instancing. Therefore, a GPU-based alternative is increasingly needed (Rastergrid, 2010). In this study, this inconvenience is avoided by a two-pass based rendering technique (Fig. 8).

First, in the culling pass, the graphics pipeline is fed with information about the instanced tip nodes generated in Section 5.1.2. The culling shader is composed of a vertex shader and a geometry shader. The vertex shader determines whether the actual

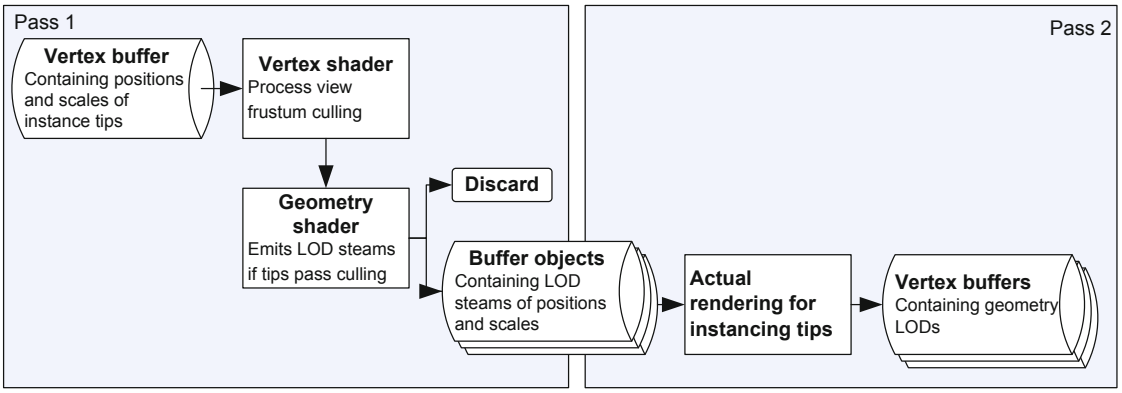

Fig. 8 Two-pass based workflow for improving hardware instancing performance

object instance's bounding volume is inside the view frustum and sends a flag about the culling to the geometry shader, which will emit the instance data to the destination buffer if it passes. Then, the primitives emitted by the geometry shader are captured by a transform feedback and sent to another buffer; meanwhile, further LOD determination can be done in the geometry shader to output data to separate primitive streams for an actual rendering pass.

In the actual rendering pass, with the pre-generated geometry and streams provided by an asynchronous query, each LOD buffer is drawn straightforward. Putting everything together, this two-pass optimization could maximize the hardware instancing ability by largely reducing the amount of vertex data sent through the graphics pipeline. The whole workflow on the GPU is shown in Fig. 8.

6 Dynamic interaction with the underwater environment

Interaction between the sealife is an important element in creating a believable and visually pleasant underwater scene. How to simulate coral motion in a water flow field with actions from the fishes is a complex and scientific topic. Some complex physical equations are required to apply local water force to individual branches of the coral at very high cost. Besides equations, various factors influence the deformation of the branches. Take an anemone as an example. The factors include stiffness and length of the branches, size and shape of the tips, conditions of local water flow field, and pulling force from the trunk. In this study, good results are achieved by simplifying these factors into a local model, which is described as follows.

In Fig. 9, the control points are bound by springs to form a physical model that represents a branch. With this model, we can examine how detailed the observed motions of the branch will be. We can deduce that the branches will show the swinging and waving motions, but not the swirling motion. In this model, it is important to analyze the forces acting on the control points that make up the branch.

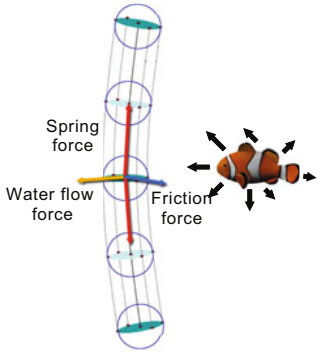

Fig. 9 A simple physical model which consists of control points and springs to represent a branch

The first force is spring tension between these points. The points are bound mutually in a pair sequence. Point 1 is bound to 2, 2 to 3, and 3 to 4. The springs are the sources of the force between two points. The spring force is formulated as

$$\boldsymbol{f}_{\mathrm{s}} = -k_1(\boldsymbol{l} - \boldsymbol{d}_1), \tag{1}$$

where k_1 is a constant that represents the stiffness of the branch spring, $\boldsymbol{l}$ is the distance to the point it is bound to, and $\boldsymbol{d}_1$ is a constant positive distance within which a spring stays steady. Eq. (1) clearly indicates that if the distance between two points is

equal to $\boldsymbol{d}_1$, no force will be applied. When $\boldsymbol{l}$ is larger than $\boldsymbol{d}_1$, the spring will stretch and otherwise, it will shrink.

The second force is the local water flow which is usually caused by fishes, formulated as

$$\boldsymbol{f}_{\mathrm{w}} = k_2/\boldsymbol{s}^2, \tag{2}$$

where k_2 is a constant that represents the stiffness of the branch swing caused by water flow, and $\boldsymbol{s}$ is the distance from the branch to the flow source. A minimum distance affected by water flow should also be defined to decide whether this force needs to be computed.

The third force is the friction part, i.e.,

$$\boldsymbol{f}_{\mathrm{f}} = -k_3\boldsymbol{v}, \tag{3}$$

where k_3 is a constant representing how much friction is in the water and $\boldsymbol{v}$ is the current velocity of the point. The friction equation can be written differently, but this one works well for the proposed soft branch model.

A local cluster comprises hundreds of branches, so the control points will inevitably smash each other. Collision detection is computed and the cost is low because of the point model. If the result of a collision is positive, we make the control points bounce off each other (Fig. 10). $\boldsymbol{d}$ is the initial velocity of the point, $\boldsymbol{s}$ is the projection of $\boldsymbol{d}$ onto the vector from the point to the point off it which is bouncing, and $\boldsymbol{d} - 2\boldsymbol{s}$ is the velocity of the point after the bounce. Clown fishes swimming in a giant anemone are one of the most complex interactions in the underwater environment. The sensory tentacles of the anemone are easily swung by the water flow caused by the fish. This scene is simulated to demonstrate the physical model. The positions of the base control points are computed in the CPU and updated on the GPU frame by frame; full detailed anemone branches are constructed by a geometry shader. High frame rate animation is achieved in this way (Fig. 11).

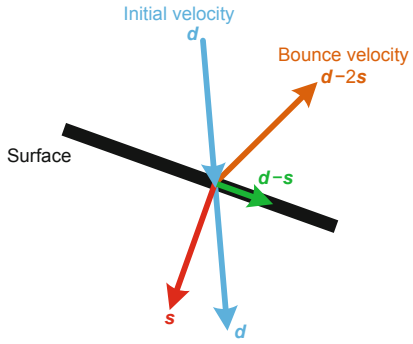

Fig. 10 Collision detection and bounce for control points

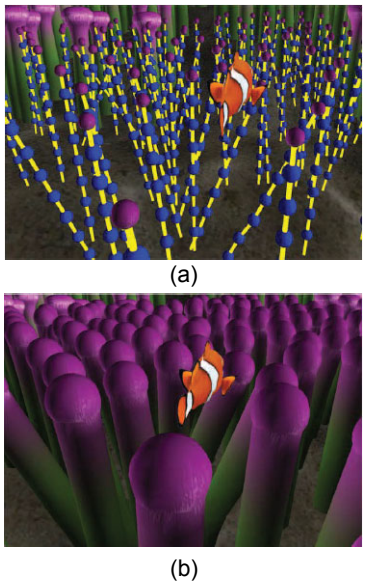

Fig. 11 A clown fish swimming in the giant sea anemone. The control points model is computed in CPU (a) and the geometry of detailed branches is created on the geometry shader in GPU (b)

7 Results

The multi-stage pipeline is implemented in a system coded in C++ and OpenGL. Several typical underwater species generated by our system are compared with photos in Fig. 12. The unique biological features such as the bump skin on brain coral (Fig. 12, row one) and the spiny skin on a starfish (Fig. 12, row two) are generated from the parameter map by tessellation. The moving tentacles of *Xenia umbellata* are generated and rendered by optimized instancing.

The massive bottom of the ocean scene comprises endless terrains and sealife. The species distribution is simulated by the ecosystem model discussed in Section 4. All images in this study are generated under real-time conditions at a resolution of 1920×1080 using a personal computer with a 2.93 GHz Intel i7-875 CPU and Nvidia GeForce GTX 580 GPU. Table 1 lists the number of rendered faces and the total time for generating and rendering a frame for the views in Fig. 13. This algorithm does not require storage except for the grammars and several

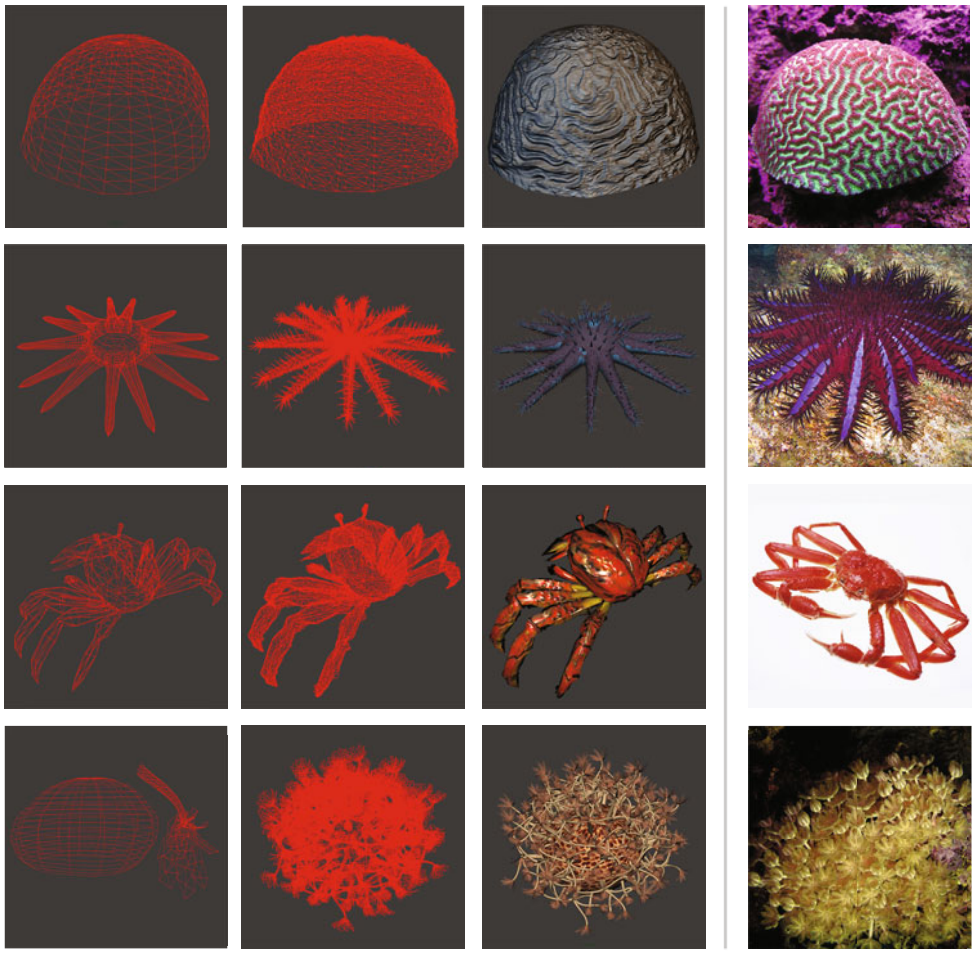

Fig. 12 Comparison between photos (the fourth column) and our results (the first three columns). The first column is the base mesh and tip node after CPU parsing grammar. The second column is the same model with unique biological details. These geometric details are generated by GPU on the fly. The models in the first two columns are in wireframe mode. The models in the third column are in texture and lighting mode

Fig. 13 Different views at the bottom of ocean scene

Table 1 Total rendering time and total number of triangles of the different views in Fig. 13

View	Number of triangles	Rendering time (ms)	View	Number of triangles	Rendering time (ms)
Seabed view 1	1 023 427	29.9	Crown fish view	853 798	5.4
Seabed view 2	987 526	28.0	Floating view	1 549 140	32.8

necessary textures. All the objects are generated and rendered on the fly. If the full scene of the floating view in Fig. 13 is stored, it will take about 337 MB. The representation of the detailed geometry for the entire seabed would far exceed the available graphics memory.

8 Conclusions

In this paper, we propose an efficient procedural modeling and rendering system to generate marine ecosystems for swim-through graphic applications. A multi-stage pipeline is proposed to generate realistic underwater scenes. First, the distribution of underwater species is determined by an ecosystem simulation, which considers the interactions among sealife and between sealife and the environment. Second, a CPU-GPU hybrid algorithm is developed to generate individual sealife. A rough representation of underwater sealife is first generated on a CPU, and then directly streamed across the graphics pipeline. Fine detailed models are generated in the GPU on the fly. Finally, interactions with the underwater environment are simulated by a simplified physical model to construct dynamic scenes. We believe our work is one step towards a solution to the much more complex and general problem of modeling and rendering a complete underwater environment.

References

Bryan, T.L., Metaxas, A., 2006. Distribution of deep-water corals along the North American continental margins: relationships with environmental factors. *Deep Sea Res. I*, **53**(12):1865-1879. [doi:10.1016/j.dsr.2006.09.006]

Carucci, F., Studios, L., 2005. Inside geometry instancing. *In*: Fernando, R., Pharr, M. (Eds.), GPU Gems 2. Addison-Wesley, Massachusetts.

Castro, P., Huber, M., 2012. Marine Biology. McGraw-Hill Companies, New York.

Deussen, O., Hanrahan, P., Lintermann, B., *et al.*, 1998. Realistic modeling and rendering of plant ecosystems. Proc. 25th Annual Conf. on Computer Graphics and Interactive Techniques, p.275-286. [doi:10.1145/280814.280898]

Diener, J., Rodriguez, M., Baboud, L., *et al.*, 2009. Wind projection basis for real-time animation of trees. *Comput. Graph. Forum*, **28**(2):533-540. [doi:10.1111/j.1467-8659.2009.01393.x]

Jensen, H.W., 2001. Realistic Image Synthesis Using Photon Mapping. A.K. Peters, Ltd., Natick.

Jeschke, S., Wimmer, M., Purgathofer, W., 2005. Image-based representations for accelerated rendering of complex scenes. EUROGRAPHICS, p.1-20.

Jonsson, I.G., 1966. Wave boundary layers and friction factors. Proc. 10th Int. Conf. on Coastal Engineering, p.127-148.

Lanza, S., 2007. Animation and rendering of underwater godrays. *In*: Engel, W.G. (Eds.), ShaderX5: Advanced Rendering Techniques. Cengage Learning, p.315-327.

Lawrence, J.M., 1987. A Functional Biology of Echinoderms. The Johns Hopkins University Press, Baltimore.

Lindenmayer, A., 1968. Mathematical models for cellular interactions in development: I. Filaments with one-sided inputs. *J. Theor. Biol.*, **18**(3):280-299. [doi:10.1016/0022-5193(68)90079-9]

Lluch, J., Camahort, E., Vivó, R., 2003. Procedural multiresolution for plant and tree rendering. Proc. 2nd Int. Conf. on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa, p.31-38. [doi:10.1145/602330.602336]

Mann, K., Lazier, J., 2005. Dynamics of Marine Ecosystems: Biological-Physical Interactions in the Oceans. Wiley-Blackwell.

Marvie, J.E., Buron, C., Gautron, P., *et al.*, 2012. GPU shape grammars. *Comput. Graph. Forum*, **31**(7):2087-2095. [doi:10.1111/j.1467-8659.2012.03201.x]

McDonald, J., 2011. Tessellation on any budget. Game Developers Conf.

Nielsen, P., 1992. Coastal Bottom Boundary Layers and Sediment Transport. World Scientific, Singapore.

Papadopoulos, C., Papaioannou, G., 2009. Realistic real-time underwater caustics and godrays. Proc. 19th Int. Conf. on Computer Graphics and Vision, p.89-95.

Parish, Y.I.H., Müller, P., 2001. Procedural modeling of cities. Proc. 28th Annual Conf. on Computer Graphics and Interactive Techniques, p.301-308. [doi:10.1145/383259.383292]

Prusinkiewicz, P., Lindenmayer, A., 1990. The Algorithmic Beauty of Plants. Springer-Verlag, New York.

Rsastergrid, 2010. Instance Cloud Reduction reloaded. Available from http://rastergrid.com/blog/2010/06/instance-cloud-reduction-reloaded/.

Sakude, M.T.S., Yano, E.T., Salles, P.S.C.R., 2011. Real time image generation for underwater simulation. Proc. Interservice/Industry Training, Simulation and Education Conf.

Storlazzi, C.D., Field, M.E., Dykes, J.D., *et al.*, 2002. Wave control on reef morphology and coral distribution: Molokai, Hawaii. *Ocean Wave Meas. Anal.*, **1**:784-793. [doi:10.1061/40604(273)80]

Weber, J., Penn, J., 1995. Creation and rendering of realistic trees. Proc. 22nd Annual Conf. on Computer Graphics and Interactive Techniques, p.119-128. [doi:10.1145/218380.218427]

Wloka, M., 2003. "Batch, batch, batch": what does it really mean? Presentation at Game Developers Conf.

Wonka, P., Aliaga, D., Müller, P., *et al.*, 2011. Modeling 3D urban spaces using procedural and simulation-based techniques. Proc. 38th Annual Conf. on Computer Graphics and Interactive Techniques, Article No. 9. [doi:10.1145/2037636.2037645]

Zioma, R., 2007. GPU-generated procedural wind animations for trees. *GPU Gems*, **3**:231-240.