Synthetic Data for AUV Technical Vision Systems Testing

Aleksandr Kamaev[1], Sergey Smagin[1], Viacheslav Sukhenko[2], and Dmitry Karmanov[1]

[1]Computing Center of FEB RAS, Khabarovsk, Russia
[2]Pacific National University, Khabarovsk, Russia
kamaev_an@mail.ru

Abstract. Development of the autonomous underwater vehicle technical vision systems is impossible without precise debugging and testing. Due to the factors described in the paper, such testing in most cases cannot be implemented with the help of real AUV. Therefore, the usage of the procedurally generated virtual testing areas is suggested. The algorithm for generation and visualization of the seabed 3D model that is suitable for AUV technical vision systems testing and debugging is described. This algorithm allows building of high detailed surface of the seabed, where each part is absolutely unique and does not contain repeating texture patterns. Also, software system "AUV Vision Debugger" that consists of seabed generator and AUV simulator is considered. The simulator provides interaction between generated seabed, AUV model, and technical vision system that is currently being tested.

Keywords: AUV, computer vision, procedural, texturing, height map, tessellation, fractal noise, seabed, simulator.

Introduction

Complexity of the underwater navigation and inability to organize communication with high capacity data exchange between autonomous underwater vehicle (AUV) and operator leads to necessity of onboard technical vision systems development to improve underwater vehicle autonomy. High reliability and quality operation in various conditions is required from such systems. Their development and further usage is impossible without precise debugging and testing. Currently testing process organization and testing data obtaining becomes a serious problem.

At present time, testing areas with markers and targets located on seabed and specifically equipped pools are used for tasks of testing and debugging. Such methods are well suited for testing AUV devices and equipment, but they are not applicable for AUV technical vision systems testing and debugging because of following reasons.

1. Obtaining data from AUV is a time consuming and expensive process. It includes not only mission accomplishment itself, but also transportation to

testing area and device launching that is unacceptable especially at the early stages of development.
2. It is not always possible to obtain necessary data for testing from areas with different types of relief due to geographic reasons.
3. It is impossible to evaluate vision system accuracy while using data from real testing areas, because there are no other ways to measure seabed with the required accuracy level (up to cm) on large areas.
4. It is impossible to interfere in the vision system work at any required time.
5. There is a risk to lose the AUV due to technical vision system errors.

To avoid all mentioned problems of real data usage in AUV technical vision system testing and debugging processes, it is suggested to replace real experiments with tests in virtual environment. The software system "AUV Vision Debugger" that allows holding such tests is considered in the paper.

1 Related Work

Usage of the virtual simulations for tasks related to AUV is an actively developing research area. Currently a lot of modeling software for different purposes is developed. The workbench [1] supports 3D visualization and can be applied for missions planning. The simulator [2] is intended for AUV control systems developing. The modeling software system [3] is designed to be used for AUV operators learning. The system [4, 5] allows debugging of different systems and devices of AUV in virtual environment. Since AUV technical vision systems are using feature points and their descriptors [6–8], such requirements as high levels of detail (up to one pixel), absolute uniqueness of all relief parts and absence of repeated texture patterns are imposed to seabed model. These requirements make it impossible to use existing modeling software for AUV vision systems testing and debugging.

Generation and visualization of the seabed 3D model, that has good enough quality for AUV vision system testing and debugging, is the most complex task. Despite large number of existing methods for landscape generation [9], they cannot be directly applied for procedural seabed model synthesis. Methods based on fractal brushes [10] allow obtaining a landscape of needed form, but they require long time of artist work. Approaches based on multifractals [11, 12] do not provide enough control over the result that is necessary for generating the seabed of required form. Methods based on modeling of the physical process of landscapes formation [13, 14], do not take in account the specifics of the underwater relief formation processes. The methods of fractal interpolation [15] require basic constraints, but they do not solve problem of obtaining such constraints. Creation of the seabed 3D model that will be suitable for AUV technical vision system testing and debugging requires complex approaches and combinations of different generation methods.

2 Program System "AUV Vision Debugger"

Structural scheme of the developed program system for AUV technical vision algorithms testing and debugging – "AUV Vision Debugger" is shown in Fig. 1. This system consists of two programs: seabed 3D model generator and AUV simulator. The generator is a GUI program that allows building a seabed 3D model from the set of user defined parameters using approaches described in section 3. The simulator is a program that simulates AUV dynamic in virtual environment and allows user to observe all AUV movements by controlling the view camera. The physical model used by simulator is described in section 4.

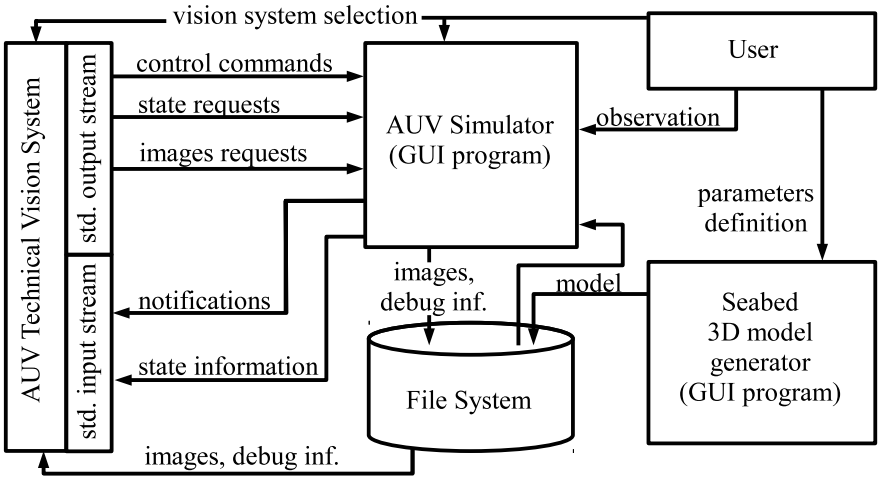

Fig. 1. Structural scheme of the program system "AUV Vision Debugger".

One of the simulator's capabilities is a selection of the controlling program – technical vision system combined with control system. The procedure of vision system testing and debugging may be different for different development stages and tasks that system must solve. Therefore, specific control system is required to implement testing of concrete vision system task. "AUV Vision debugger" does not impose any limitations on organization of technical vision and control systems and their interaction, and provides universal interface with simulator.

The simulator runs the controlling program and scans its standard output stream with approximately 50 Hz frequency to detect passed commands. As an answer to the commands the simulator writes messages to the standard input stream of the vision system and saves graphical information in the file system if required.

Commands supported by simulator can be divided into three groups: control commands, state requests and image requests. Control commands serve to

change such states of AUV parts (see section 4.1 for parts description) as position, orientation, power and etc. State requests can be used to know information about AUV position, orientation and velocity, distances to objects (if AUV equipped with sonar) and etc. As an answer to state requests, the simulator passes state information to the standard input stream of AUV technical vision system. Image requests are necessary for obtaining images with debug information from AUV onboard cameras. Images and debug information are not passed directly through standard input stream, but stored in file system. Instead, a notification that images are ready to use is passed to the input stream. The debug information contains accurate camera external and internal parameters and distances from camera to image pixels. Using this information, the controlling program can evaluate results produced by the technical vision system and make conclusions about the system's accuracy.

Thus, "AUV Vision Debugger" provides just a physical and visual model of environment and AUV. Development of the control system for testing AUV technical vision algorithms is the user's responsibility.

3 Generation and Visualization of the Seabed 3D Model

The algorithm that was developed for "AUV Vision Debugger" to generate and visualize the seabed 3D model has five stages: generation of low frequency relief map (section 3.1), fractal noise computation to increase level of details (section 3.2), 3D model mesh building (section 3.3), model refinement during visualization (section 3.4) and texturing (section 3.5).

3.1 Relief Map

Relief map $\mathbf{H} = (H_{ij}), i = 1, 2, ..., h, j = 1, 2, ..., w$, where w and h – map's size, defines relief height H_{ij} in point $(j \cdot res, i \cdot res)$, where res is map resolution specified in meters per point (m/p). The maps with $res = 10$ m/p and $w, h \leqslant 1000$ were used in the "AUV Vision Debugger". Such maps can describe the seabed with length up to 10 km that is more than enough for AUV technical vision algorithms testing and debugging.

The relief map contains such basic seabed elements as coast, shelf, continental slope, ocean bed, submarine canyons and mountains (Fig. 2a). These elements are defined by user through setting a few parameters' values, and they do not require specific artist knowledge.

Contours of the basic elements are specified with fractal lines and heights are interpolated, using different interpolation functions, based on distances to defined contour lines (Fig. 2b).

3.2 Fractal Noise

The relief details that have frequency more than $1/(2res)$ m^{-1} cannot be represented by $\mathbf{H}$ map. Meanwhile, to provide correct work of technical vision algorithms on distances from seabed that could be reached with onboard light

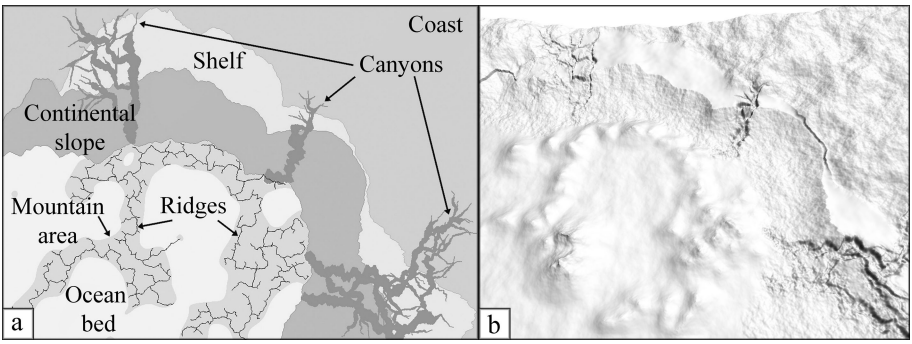

Fig. 2. Relief map: a – basic elements, b – low-frequency model.

equipment, accuracy up to millimeters is required. To improve relief level of details we will use fractal noise that consists of summed Perlin noise functions, taken at following frequencies:

$$\mathbf{f}_l = (f_{li}) = \frac{2^{i-1}}{res}, \qquad \mathbf{f}_h = (f_{hj}) = \frac{2^{j+n_l}}{res}$$

where $i = 0, 1, ..., n_l$, $j = 0, 1, ..., n_h$, $n_l = \lfloor \log_2(2 \cdot res) \rfloor$, $n_h = \lfloor \log_2(p_w^{-1} 2^{-n_l} res) \rfloor$, p_w – drawing fragment size (pixel of the screen or AUV camera) in meters. If fragment size could not be computed, the desired accuracy should be directly assigned to p_w. Low frequencies $\mathbf{f}_l$ will be added to seabed 3D model and high frequencies $\mathbf{f}_h$ will be utilized during visualization process.

Amplitudes on the same frequencies may be different for different seabed parts, depending on their slope roughness and some random factor. Let $\mathbf{S} = (S_{ij})$ be the slope map (Fig. 3a) and $\mathbf{R} = R_{ij}$ be the roughness map (Fig. 3b), where $i = 1, 2, ..., h$, $j = 1, 2, ..., w$,

$$R_{ij} = \min\left(1, \frac{\sigma(H_{ij})}{\sigma_{max}}\right),$$

$$S_{ij} = \frac{2 \arccos(n_{ij,z})}{\pi},$$

where $\sigma(H_{ij})$ is standard deviation of high frequency part of height map in 9×9 neighborhood (weighted with Gauss function with $\sigma = 2$) of the point H_{ij}, σ_{max} is a normalization factor and $\mathbf{n}_{ij} = (n_{ij,x}, n_{ij,y}, n_{ij,z})$ is a normal to low frequency part of height map in point H_{ij}.

We divide the frequencies range $(\mathbf{f}_l, \mathbf{f}_h)$ into three groups each of which is controlled by own basis function $\phi_i(f), i = 1, 2, 3$, – Fig. 4a. The logarithmic scale is used on the charts; at this scale functions $\phi_i(f)$ are piecewise-linear. As well we introduce functions of roughness and slope influences on the 1st and 2nd frequency groups:

$$\mathrm{I}_{R1}(r) = 0.45r^2 + 0.05, \quad \mathrm{I}_{S1}(s) = \begin{cases} 6s^2 - 8s^3, & s \leqslant 0.5 \\ (8s-2)(s-1)^2, & s > 0.5 \end{cases},$$

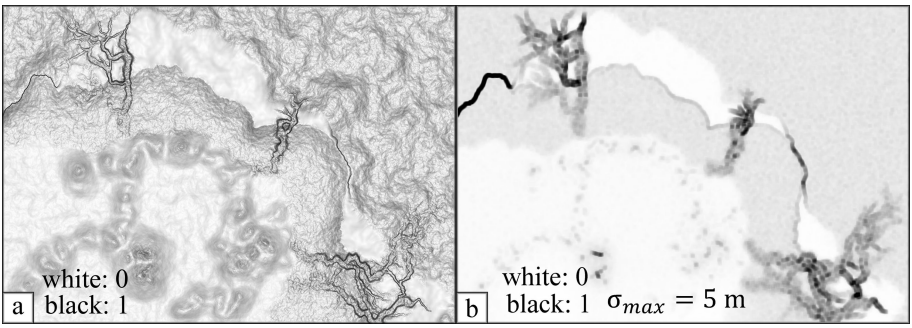

Fig. 3. a – slope map **S**, b – roughness map **R**.

$$\mathrm{I}_{R2}(r) = \frac{r}{2}, \quad \mathrm{I}_{S2}(s) = \frac{8}{9}\max(0, s - 0.25)^2.$$

Plots of these functions are presented at Fig. 4b, c. We define a united influence function for i-th frequency group that unites roughness, slope and random factors:

$$\psi_i(\mathbf{x}) = \max\left(0, \min\left(1, \mathrm{I}_{Ri}(\mathrm{R}(x_1, x_2)) + \mathrm{I}_{Si}(\mathrm{S}(x_1, x_2)) + \lambda_i \mathrm{P}_i(\mathbf{x})\right)\right), \qquad (1)$$

where $\mathbf{x} = (x_1, x_2, x_3)$ is a space point, $i = 1, 2$, $\mathrm{R}(x_1, x_2)$ and $\mathrm{S}(x_1, x_2)$ are the values obtained from **R** and **S** by means of bilinear interpolation, $\lambda \in [0..1]$ is user defined coefficient of the random influence on i-th group,

$$\mathrm{P}_i(\mathbf{x}) = \sum_{j=0}^{4} \frac{1}{2^j} \mathrm{P}\left(\frac{2^j}{res}(\mathbf{M}_i\mathbf{x} + \mathbf{T}_i)\right),$$

where $i = 1, 2, 3$, $\mathrm{P}(\mathbf{x})$ is a Perlin noise function [16], $\mathbf{M}_i$ is a random rotation matrix, and $\mathbf{T}_i$ is a random translation vector. The noise amplitude in $\mathbf{x}$ point corresponding to frequency f

$$\mathrm{A}(\mathbf{x}, f) = \frac{\gamma}{f}\left(\sum_{i=1}^{2} \phi_i(f)\psi_i(\mathbf{x}) + \phi_3(f)\left(\frac{\sin\left(2\pi \mathrm{P}_3(\mathbf{x})\right) + 1}{2}\right)\right), \qquad (2)$$

where γ – coefficient defining general roughness of height differences on all frequencies. Recommended value lies in range $\gamma \in [0.5..1]$. Using amplitude (2) we define noise in $\mathbf{x}$

$$\mathrm{N}(\mathbf{x}, t) = \sum_{i=0}^{n_t} \mathrm{A}(\mathbf{x}, f_{ti})\mathrm{P}(f_{ti}\mathbf{x}). \qquad (3)$$

3.3 Building of 3D Model

For mesh construction we will use approach [17], but unlike [17], we will build mesh during generation step on CPU and not at rendering time on GPU. We

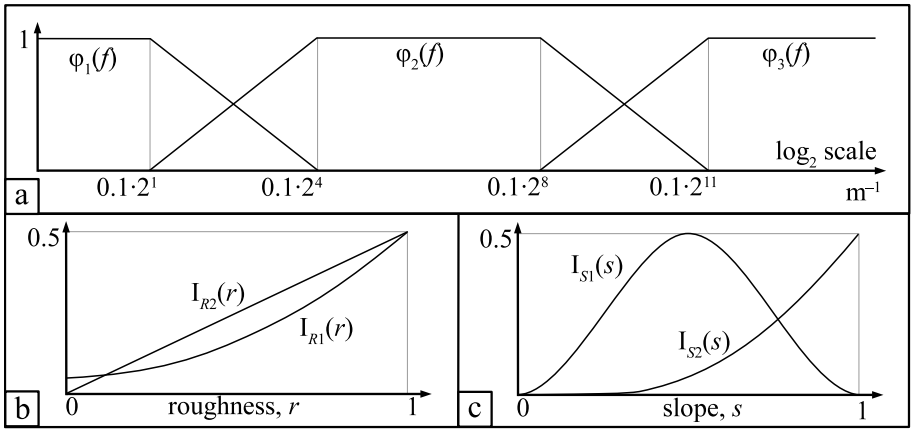

Fig. 4. a – functions $\phi_i(f)$, b – functions $\mathrm{I}_{R_1}(r)$,$\mathrm{I}_{R_2}(r)$, c – functions $\mathrm{I}_{S_1}(s)$, $\mathrm{I}_{S_2}(s)$.

define relief density in point $\mathbf{x} = (x_1, x_2, x_3)$

$$\rho(\mathbf{x}) = x_3 - \mathrm{H}(x_1, x_2) + \mathrm{N}(\mathbf{x}, l), \tag{4}$$

where value $\mathrm{H}(x_1, x_2)$ is obtained from $\mathbf{H}$ by means of bilinear interpolation. Surface $\rho(x) = 0$ defines seabed. To obtain 3D model of seabed surface $\rho(x) = 0$ is approximated using marching cubes algorithm [18]. Figure 5 depicts result of such approximation.

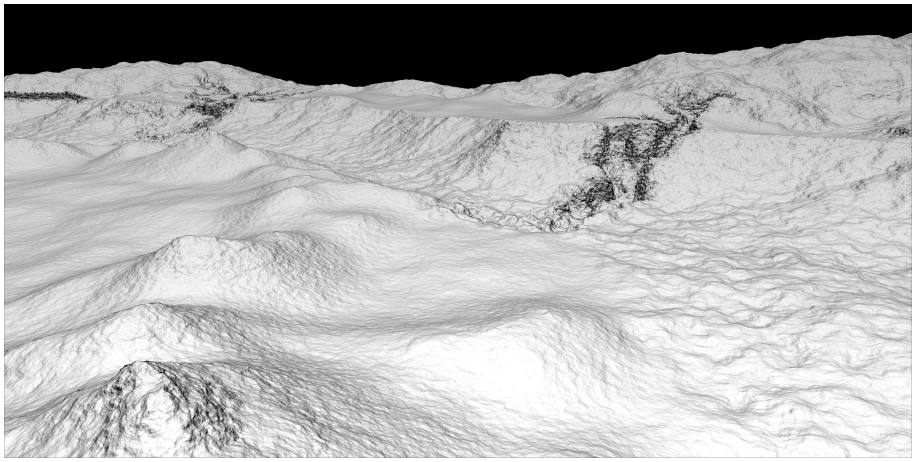

Fig. 5. Seabed 3D model.

3.4 Visualization

It is necessary to tessellate mesh during visualization to provide size close to one pixel for all visible triangles. The most of modern GPUs supported hardware accelerated tessellation with subdivision by up to 64 parts for each triangle side. Such subdivision will not be enough for mesh triangles located near the camera. Therefore, tessellation based on precalculated triangles sets (PTS) is suggested for these triangles.

Each triangle side could be divided into 2^{n_i} parts, where $i = 1, 2, 3$ is an index number of triangle side (counterclockwise numeration), and $n_i = 0, 1, ..., n_{max}$. AUV operation distance to seabed and onboard cameras resolution allows to choose $n_{max} = 9$ that leads to $N = (n_{max} + 1)^3 = 1000$ of different variants of tessellation. Let us consider j-th tessellation set, $j = 1, 2, ..., n$, that consists of N_j triangles. Each triangle of j-th set is described by three vertices $\mathbf{v}_{jkt} = (v_{jkt1}, v_{jkt2})$, $k = 1, 2, ..., N_j$, $t = 1, 2, 3$ (counterclockwise numeration). Values v_{jkt1} and v_{jkt2} are coordinates of t-th vertex of k-th triangle of j-th set, they defined in the coordinate system with basis vectors represented by 1st and 3rd sides of k-th triangle and origin in the intersection point of these sides – Fig. 6a.

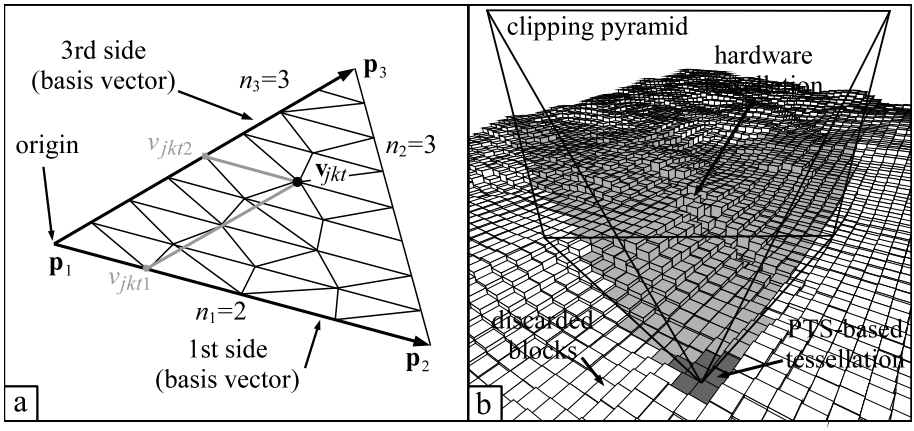

Fig. 6. a – PTS-based tessellation, b – invisible blocks clipping.

Using such coordinate system we can draw j-th tessellation set instead of some mesh triangle that have coordinates $\mathbf{p}_1$, $\mathbf{p}_2, \mathbf{p}_3$, by projecting local j-th set coordinates to the global space:

$$\mathbf{x}_{jkt} = v_{jkt1}(\mathbf{p}_2 - \mathbf{p}_1) + v_{jkt2}(\mathbf{p}_3 - \mathbf{p}_1) + \mathbf{p}_1. \quad (5)$$

The fact that tessellation sets store coordinates independent from mesh coordinates allows us to put these sets into video memory once, and then just use through one function call. The attempt to draw a triangle is replaced by suitable

tessellation set drawing under control of vertex shader that computes global vertices coordinates using (5). Coordinates $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ are passed to vertex shader as uniform parameters. The number of tessellation set j is defined based on triangle size and its remoteness from the camera to provide pixel accuracy in the screen space.

To speed up visualization process the seabed model is divided into rectangular blocks, so that blocks number and triangles number per block would allow iterating through them before each frame rendering. Blocks located behind camera's clipping planes are discarded. For rendering of the triangles located in blocks that are close to the camera, the PTS-based tessellation is used. If the block is far enough for using hardware accelerated tessellation only, it renders by single drawing function call. The process of seabed subdivision is shown in Fig. 6b.

After tessellation the fractal noise (3) is added to each vertex in the direction of the interpolated normal $\mathbf{n}$, computed in this vertex:

$$\hat{\mathbf{x}} = \mathbf{x} + \mathbf{n}\left(\mathrm{N}(\mathbf{x}, h) + \rho(\mathbf{x})\right),$$

where $\rho(\mathbf{x})$ is a density function (4). The example of tessellated and amplified with high frequency noise seabed model is presented in Fig. 7.

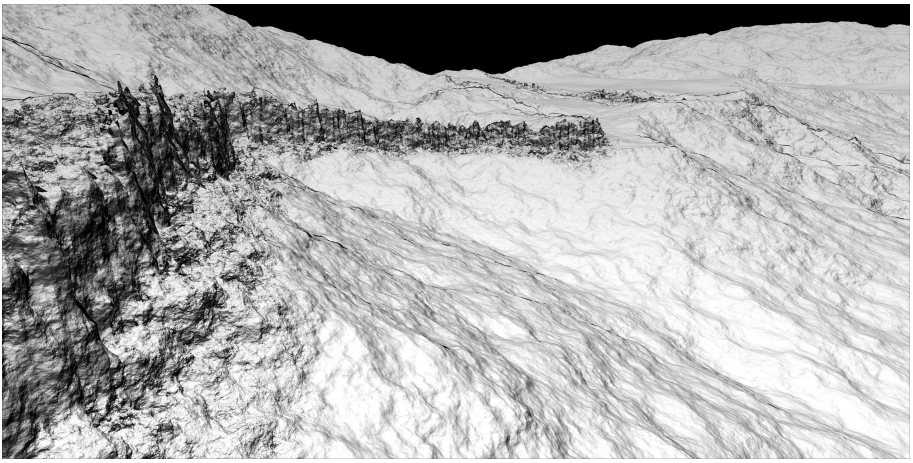

Fig. 7. Tessellated seabed 3D model.

3.5 Texturing

Texturing of landscapes, developing for testing AUV technical vision system has its own specific:

1. Only procedural textures could be used, because using of the bitmap images leads to the appearing of repeated texture patterns that makes impossible the testing of the algorithms based on feature points [6–8].
2. Complex relief makes impossible calculation of correct 2D texture coordinates, so just 3D textures could be used.
3. Textures should be correctly represented on all scale levels that are used during testing.

Surface type and, respectively, texture type is defined based on functions $\psi_1(\mathbf{x})$ and $\psi_2(\mathbf{x})$ (1). The sand procedural texture with weight α is mixed with the stone procedural texture with weight $1 - \alpha$, where

$$\alpha(\mathbf{x}) = \prod_{i=1,2} (1 - \text{smoothstep}(\psi_{ic} - \delta, \psi_{ic} + \delta, \psi_i(\mathbf{x}))) .$$

Smoothstep is a standard GLSL function. Constants ψ_{ic} define boundaries between relief types and δ is a transition width. These constants are set during seabed generation. The Fig. 9 is obtained with $\psi_{1c} = 0.2$, $\psi_{2c} = 0.12$ and $\delta = \max(0.0025, 0.05p_w)$, where p_w – is a drawing fragment size in meters.

Creation of procedural texture for each type of surface requires individual approach. Sum of Perlin noise functions or cellular basis [11] is used to create different unique texture patterns at different scales. When the screen's pixel size becomes too large to depict a pattern, this pattern is replaced by its average color and intensity. A single texture could have up to three different patterns at different scales. An example of sand and stone textures used in AUV Vision debugger is presented in Fig. 8.

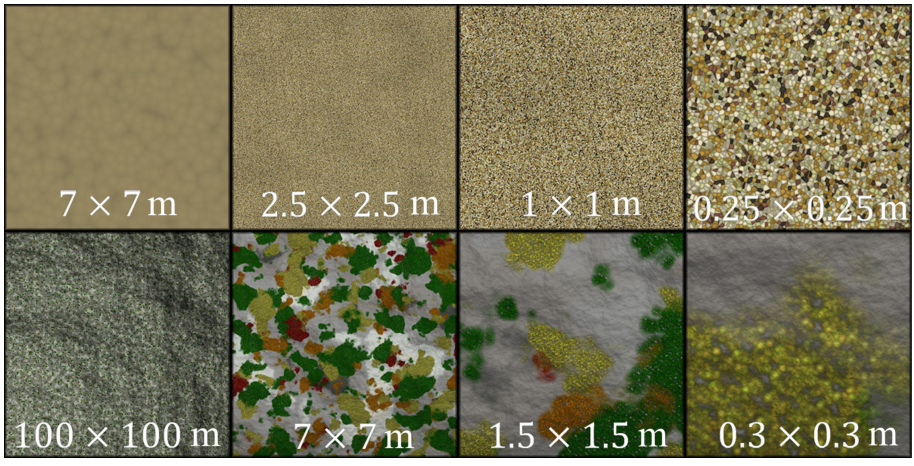

Fig. 8. Textures at different scale: sand(above) and stone covered with moss (below).

4 AUV Simulator

The main tasks of the simulator are visualization of seabed and AUV models, simulation of AUV dynamics, and computation of interactions with the environment. Simulator also provides the possibility to observe the mission process. "AUV Vision Debugger" was created mainly for testing and debugging of technical vision system and not for testing AUV control systems and devices, therefore simplified AUV model (section 4.1), its dynamics (section 4.2) and interactions with seabed (section 4.3) are used.

4.1 AUV Model

AUV model consists of parts, described in text file of the model. Each part has its own type: shell, engine, control surface, sonar, floodlight and camera. In the model description there could be only one shell and arbitrary number of parts having other type. Let us consider main parameters of different types:

1. Shell is defined by its weight and 3D model in 3DS format. The coordinate system of the shell is a basis for all other parts.
2. Engine and control surface are defined by their weights, 3D model, transformation matrix, describing position and orientation relative to shell. Directions and ranges of engine and control surface allowable movements and rotations are also defined. Thrust vector and its magnitude range could be defined for engine.
3. Sonar, floodlight and camera do not have their own weight and graphic representation. They defined by position and orientation relative to shell. Directions and ranges of allowable movements and rotations are also defined. Light power could be set for floodlight and focus distance, resolution, radial distortion, and lighting dependent errors could be set for camera.

Apart from mentioned parameters, each part has a name that allows the vision system to interact with this part.

4.2 AUV Dynamics

From position of dynamics AUV with all its parts is considered as one rigid body. To calculate its motion Newton–Euler equations are used. The details of motion computation process based on forces and torques acting on the rigid body are well described in [19]. The weight of one part (if part has weight) is uniformly distributed between all the part's points. The part's points correspond to vertices of the part's 3D model. All relative movement of AUV parts leads to redistribution of AUV weight and to recalculation of inertia tensor. Forces that lead to relative movements of parts, as well as forces induced by this movement, are not counted.

Simulator takes into account following forces: gravity force $\mathbf{F}_g$, thrust force of i-th engine $\mathbf{F}_{ti}$,$i = 1, 2, ..., n_e$, where n_e is engines number, pressure force $\mathbf{F}_p$

and hydrodynamic force $\mathbf{F}_h$. All forces except $\mathbf{F}_g$ induce torques relative AUV center of mass, that accounted in motion calculation. Let us consider how these forces act.

Gravity force is applied to AUV center of mass and pointed vertically down $\mathbf{F}_g = (0, 0, -mg)$, where m is sum of weight of all parts, and g is acceleration of gravity. Thrust force $\mathbf{F}_{ti}$ is applied to i-th engine center of mass and directed along thrust vector, described in AUV text file. If rotation is applied to i-th engine then its thrust vector is also undergo this rotation. Magnitude of $\mathbf{F}_{ti}$ is chosen by vision system from the range defined in model text file.

Forces $\mathbf{F}_p$ and $\mathbf{F}_h$ are calculated for each face of AUV 3D model if this face has nonzero area S, external unit normal vector $\mathbf{n}$ and located out of the shell. Forces $\mathbf{F}_p$ and $\mathbf{F}_h$ are applied to geometric center of the face $\mathbf{c} = (c_1, c_2, c_3)$:

$$\mathbf{F}_p = -\rho g |c_3| S \mathbf{n},$$

$$\mathbf{F}_h = F_{hx} \frac{-\mathbf{v}}{|\mathbf{v}|} + F_{hy} \frac{\mathbf{n} \times \mathbf{v} \times \mathbf{v}}{|\mathbf{n} \times \mathbf{v} \times \mathbf{v}|},$$

where ρ is a water density, $\mathbf{v}$ is velocity of $\mathbf{c}$ relative to environment, $\mathbf{F}_{hx}$ and $\mathbf{F}_{hy}$ are components of $\mathbf{F}_h$ in direction of $\mathbf{v}$ and normal to the $\mathbf{v}$ vector direction:

$$F_{hx} = |\mathbf{n} \cdot \mathbf{v}| S \frac{\rho |\mathbf{v}|}{2},$$

$$F_{hy} = S \frac{\rho |\mathbf{v}|}{\sqrt{2}} \mathbf{n} \cdot \mathbf{v} \sqrt{1 - \left(\mathbf{n} \cdot \frac{\mathbf{v}}{|\mathbf{v}|} \right)^2}.$$

Presented forces are sufficient to describe AUV dynamics accurate enough for technical vision systems testing and debugging.

4.3 Interactions with Seabed

Collision of AUV parts with seabed surface is a very dangerous situation that should be necessarily prevented by a technical vision system during visual navigation. Therefore, it is very important to detect such situation and to inform the vision system in the case it occurs. Informing is implemented by passing a message to the standard input stream of the vision system. We consider a process of detecting collision by the simulator and its reaction on this collision.

Collisions are tested on each simulation iteration for parts points. Let us consider collision detection process for point $\mathbf{x}$. The collision is tested for all seabed model triangles that are closer to $\mathbf{x}$ than one meter. If there are no such triangles then point $\mathbf{x}$ is above the seabed. Let $\mathbf{p}_i$ be the vertices of triangle that is closer to $\mathbf{x}$ than one meter, $i = 1, 2, 3$, and $\mathbf{n}_i$ be the normals to the model in these vertices. To detect a collision we should perform following steps:

Step 1: Computing face normal: $\overline{\mathbf{n}} = (\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)$.

Step 2: Computing intersection points between rays from $\mathbf{p}_i$ in directions $\mathbf{n}_i$ and plane $(\overline{\mathbf{n}}, -\overline{\mathbf{n}} \cdot \mathbf{x})$:

$$\mathbf{p}'_i = \mathbf{p}_i + \frac{\overline{\mathbf{n}} \cdot \mathbf{x} - \overline{\mathbf{n}} \cdot \mathbf{p}_i}{\overline{\mathbf{n}} \cdot \mathbf{n}_i} \mathbf{n}_i.$$

Step 3: If $\mathbf{x}$ does not lie inside triangle $(\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3)$, then no collision detected, else go to step 4.

Step 4: Finding of interpolated unit normal $\tilde{\mathbf{n}}$ in $\mathbf{x}$ point of triangle $(\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3)$ with vertex normals $\mathbf{n}_i$, by means of bilinear interpolation.

Step 5: Computing point of triangle $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$, corresponding to $\mathbf{x}$:

$$\tilde{\mathbf{x}} = \mathbf{x} + \frac{\overline{\mathbf{n}} \cdot \mathbf{p}_1 - \overline{\mathbf{n}} \cdot \mathbf{x}}{\overline{\mathbf{n}} \cdot \tilde{\mathbf{n}}} \tilde{\mathbf{n}}.$$

Step 6: Computing of $\mathbf{x}$ penetration distance into the seabed surface, using (3) and (4):

$$d = (\tilde{\mathbf{x}} + \tilde{\mathbf{n}}\,(\mathrm{N}(\tilde{\mathbf{x}}, h) + \rho(\tilde{\mathbf{x}})) - \mathbf{x}) \cdot \tilde{\mathbf{n}}.$$

If $d > 0$, then $\mathbf{x}$ point is located inside seabed, therefore collision happened. If value $d > 0$ computed for more than one triangle, maximum value should be chosen.

If collision of depth d for some point $\mathbf{x}$ with normal $\tilde{\mathbf{n}}$ is detected, then following force is applied to $\mathbf{x}$:

$$\mathbf{F}_c = (k_s d - k_d \mathbf{v} \cdot \tilde{\mathbf{n}})\tilde{\mathbf{n}},$$

where k_s is an elasticity coefficient, k_d is a damping coefficient, and $\mathbf{v}$ is a velocity of $\mathbf{x}$ point. Application of $\mathbf{F}_c$ force prevents AUV penetration into the seabed.

Conclusion

Methods and algorithms, suggested in the paper with developed program system "AUV Vision Debugger", allow testing and debugging of AUV technical vision systems in virtual environment that leads to following advantages:

1. high speed and low cost of testing data acquisition,
2. ability to obtain testing data from different surface types, starting from sand valleys and ending with rocky canyons,
3. ability to evaluate technical vision accuracy since the investigated seabed surface is precisely known,
4. ability to interrupt system working exactly at the moment when error occurs,
5. testing result repeatability.

Currently AUV Vision Debugger allows getting high detailed images of seabed, all parts of which are absolutely unique – Fig. 9. For further researches it is planned to add a possibility of procedural modeling and visualization of marine flora and fauna, including dynamically changing objects, to add more procedural textures and submarine caves network. It is also planned to learn ways of integrating our work with existing general purpose software modeling systems.

It is evident that using of synthetic tests in "AUV Vision Debugger" does not allow us to completely abandon the real experiments, but it significantly reduces their amount. As a result, the time required for AUV technical vision system development is decreased and reliability is increased.

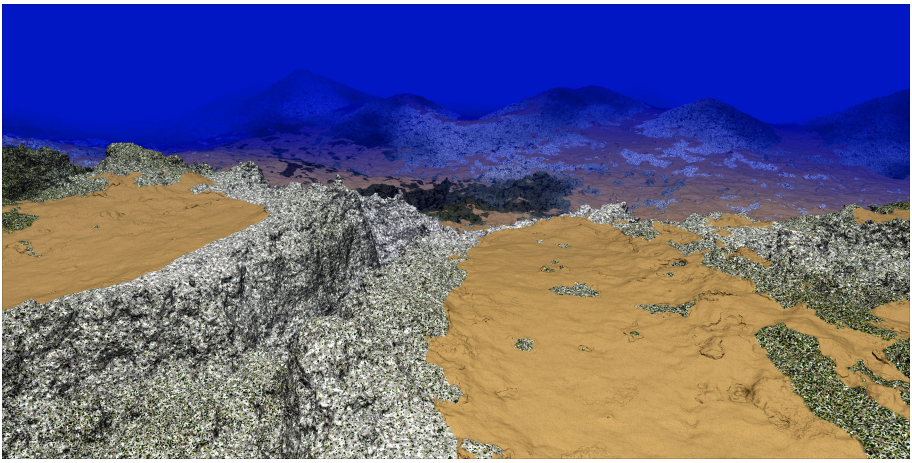

Fig. 9. Example of generated seabed image.

Acknowledgments. This work was supported by Russian Foundation for Basic Research (grant 16-31-00187 mol_a).

References

1. Davis, D.T, Brutzman, D.: The autonomous unmanned vehicle workbench: mission planning, mission rehearsal, and mission replay tool for physics-based x3d visualization. In: 14th International Symposium on Unmanned Untethered Submersible Technology (UUST), Autonomous Undersea Systems Institute (AUSI), pp. 21-24. Durham New Hampshire (2005)
2. Dantas, J.L.D., de Barros, E.A.: A real-time simulator for auv development. ABCM Symposium Series in Mechatronics vol. 4, 499–508 (2010)
3. Hanychev V.V. Trenazhorniy compleks dlya obuchenia operatorov teleupravlyaemykh neobitayemykh podvodnykh apparatov razlichnykh tipov In: 6-th Russian Conf. Tehnicheskie Problemi Osvoenia Mirovogo Okeana, pp. 50–60, Vladivostok (2015)
4. Inzartsev, A.V, Sidorenko, A.V., Senin, R.A., Matvienko, V.Y.: Kompleksnoye testirovanie programmnogo kompleksa na base imitacionnogo modeliruyuschevo kompleksa. In: Podvodnie issledovaniya i robototehnika vol. 1(7), 9–14 (2009)
5. Inzartsev, A.V. et al.: Integrirovannaya informatsionno-upravlyayuschaya i modeliruyushcaya sreda dlya avtonomnogo podvodnogo robota. In: 6-th Russian Conf. Tehnicheskie Problemi Osvoenia Mirovogo Okeana, pp. 129–133, Vladivostok (2015)
6. Herbert, B. et al.: SURF: Speeded Up Robust Features. CVIU, vol. 110, 346–369 (2008)
7. Mikolajczyk, K., Schmid C.: Scale & affine invariant interest point detectors. IJCV vol. 60, 63–86 (2004)
8. Verma A. et al.: A New Color SIFT Descriptor and Methods for Image Category Classification. In: IRAST International Congress CACS, pp. 819–822. Singapore (2010)

9. Smelik, R.M., de Kraker K.J., Tutenel. T.: A Survey of Procedural Methods for Terrain Modelling. In: CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS), pp. 25–34. Amsterdam (2009)
10. Giliam, J.P. de Carpentier, Bidarra, R.: Interactive GPU-based procedural heightfield brushes. In: 4th International Conference on Foundations of Digital Games pp. 55-62. ACM (2009)
11. Ebert, D.S. et al. Texturing and Modeling A Procedural Approach. Morgan Kaufmann, Sasn Francisco (2003)
12. Schneider J., Boldte T., Westermann R.: Real-Time Editing, Synthesis, and Rendering of Infinite Landscapes on GPUs In: Conf. on Vision, Modeling, and Visualization, pp. 145–152. Aachen, Germany (2006)
13. Belhadj, F., Audibert, P.: Modeling Landscapes with Ridges and Rivers: bottom up approach. In: 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, pp. 447–450. ACM (2005)
14. Genevaux, J.D. et al.: Terrain generation using procedural models based on hydrology. ACM Transactions on Graphics (TOG) vol. 32(4), p. 163. (2013)
15. Belhadj, F.: Terrain modeling: a constrained fractal model. In: 5th international conference on Computer graphics, virtual reality, visualization and interaction in Africa, pp. 197-204. ACM (2007)
16. Perlin, K.: Improving noise. ACM Transactions on Graphics (TOG) vol. 21(3), 681–682 (2002)
17. Geiss. R.: Generating Complex Procedural Terrains Using the GPU. In: GPU Gems 3, pp. 7-37. Addison-Wesley (2008)
18. Lorensen, W.E., Cline, H.E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. COMPUTER GRAPHICS vol 21(4), 163–169 (1987)
19. Baraff, D.: An Introduction to Physically Based Modeling Rigid Body Simmulation 1 — Unconstrained Rigid Body Dynamics. SIGGRAPH Course Notes (1997)