GENERATING RANDOM TERRAINS IN 3D GAMES BY AN APPROXIMATED VERSION OF GAUSSIAN PROCESS

Viet-Hung Dang [(1), (2)], Van Nhan Vo [(1), (2)] *

[1] *Faculty of Information Technology, Duy Tan University, Danang, Vietnam*
[2] *Institute of Research and Devolopment, Duy Tan University, Danang, Vietnam*

ARTICLE INFORMATION

Journal: *Vinh University Journal of Sciences*
ISSN: *1859-2228*
Volume: 52
Issue: 1A
Corespondance:
vonhanvan@dtu.edu.vn
Received: *03 December 2022*
Accepted: *10 January 2023*
Published: *20 March 2023*

Citation:
Viet-Hung Dang, Van Nhan Vo (2023). Generating random terrains in 3D games by an approximated version of Gaussian process.
Vinh Uni. Jour. Sci.
Vol. 52 ***(1A),*** *pp. 45-54*
doi: 10.56824/vujs.2022a047

ABSTRACT

Creating a 3D terrain or a wave surface takes a lot of time for a graphic artist in the 3D game creation process. This problem poses the following computer requirements: the generated terrain surface must be both random at all points, but also smooth and curved, and the difference between the vertices must be controlled by an input parameter. This paper will upgrade the Gaussian process (GP) from 2D to 3D in order to solve this terrain generation problem. In addition, when the generated surface becomes very large, GP requires a large amount of computation related to matrix analysis. This paper proposes an approximate solution to speed up the computation.

Keywords: Gaussian process; 3D terrain.

1. Introduction

Random terrain generation techniques have been extensively studied when the demand for graphics in movies, advertisements, and games is increasing [1]-[3]. The generating methods usually start with random values (noise) for each spatial point, then use fractal techniques to increase the details of the terrain or define a hash function that generates random values for the terrain coordinate vector [x, y]. The randomness is then controlled by the parameters of the hash function, meaning that for a fixed set of parameters, the generated terrain is the same for different trials. Random value generation techniques through hash functions with random parameters often produce rough, spiky, and unrealistic terrain. To minimize this, interpolation techniques are often used to smooth the existing terrain, but the spikes are not much reduced. Filtering techniques through convolution can also be applied to reduce spikes; however, if an area with a very high spike is encountered, the filtering technique cannot remove the spike. If using spline interpolation of levels, the amount of computation becomes very large.

GP plays an important role in the research development in the field of Machine Learning via statistics [4]-[7]. GP is also a sampling technique with a complex theory but requires programming techniques at moderate level of difficulty. The resulting GP is not a random value but a random process in which the values in the vicinity

of each other are dependent on each other in value, that is, have the same value. GP ensures smooth curvature of the generated terrain height values, regardless of whether the resolution is high or low. This paper will present the theoretical foundations of GP, how it generates a 2D random process, and how it can be improved for 3D space. Finally, the paper will present a method to reduce the computation time for GP when applied to random terrain generation.

2. Gaussian Process

GP is essentially a stationary stochastic process [8], in which the components of the process are related through a multivariable Gaussian distribution. A multivariable Gaussian distribution is a distribution where each variable x_i, being a component of the vector $\mathbf{x} = \left[x_1, x_2, ..., x_n\right]^T$, follows a univariate Gaussian distribution. Then, the vector $\mathbf{x}$ has a multivariable Gaussian distribution, i.e.,

$$\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}, \Sigma\right). \tag{2.1}$$

This distribution is described in detail as follows:

$$p\left(\mathbf{x}\middle|\boldsymbol{\mu}, \Sigma\right) = \frac{1}{\left(2\pi\right)^{n/2}\left|\Sigma\right|^{1/2}} \exp\left\{-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^T \Sigma^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right)\right\}, \tag{2.2}$$

where $\boldsymbol{\mu}$ is the mean vector of $\mathbf{x}$, and $\sum$ is the covariance matrix governing the relations among the component variables. This means

$$\begin{cases} \boldsymbol{\mu} = \left[\mu_1, \mu_2, ..., \mu_n\right]^T = E\left(\mathbf{x}\right) \\ \sum = \operatorname{cov}\left(\mathbf{x}\right) = E\left\{\left(\mathbf{x}-\boldsymbol{\mu}\right)\left(\mathbf{x}-\boldsymbol{\mu}\right)^T\right\} \end{cases}. \tag{2.3}$$

The coefficient

$$Z_{\mathbf{x}} = \frac{1}{\left(2\pi\right)^{n/2}\left|\Sigma\right|^{1/2}} \tag{2.4}$$

acts as the coefficient that normalizes the distribution function so that the sum (integral) of probabilities over the entire *n* dimensions of the distribution is equal to 1. The covariance matrix $\sum$ is a semi-positive deterministic matrix [9] and symmetric because $\left\{\left(\mathbf{x}-\boldsymbol{\mu}\right)\left(\mathbf{x}-\boldsymbol{\mu}\right)^T\right\}$ is a symmetric matrix. Every element being off the diagonal of Σ is a covariance of the two component variables

$$\sigma_{ij} = \sigma_{ji} = E\left\{\left(x_i - \mu_i\right)\left(x_j - \mu_j\right)\right\}, \tag{2.5}$$

while the elements on the diagonal are the variance of each component variable as

$$\sigma_{ii} = E\left\{\left(x_i - \mu_i\right)^2\right\}. \tag{2.6}$$

GP [3], [8] will be specifically defined by two parameters: *m(x)* and the covariance function $k\left(\mathbf{x}, \mathbf{x}'\right)$. GP is, in fact, a generalization form of the Gaussian distribution. The

difference is that a Gaussian distribution is a concept over a vector containing connected random variables, while GP is a concept over a function. Particularly, a function f is denoted by

$$f \sim \mathcal{GP}(m,k). \tag{2.7}$$

Equation (2.7) means that the distribution function is a GP with the mean of the function *m(x)* and the covariance of the function *k(x, x')*. Usually the covariance function *k(x, x')* of a GP is a family of natural exponential functions of quadratic polynomials.

$$k(x,x') = z_1 \exp\left(-z_2 (x-x')^2\right) \tag{2.8}$$

where z_1 and z_2 are positive coefficients and do not have to be a normalized coefficients to make *k(x, x')* be a probability distribution function

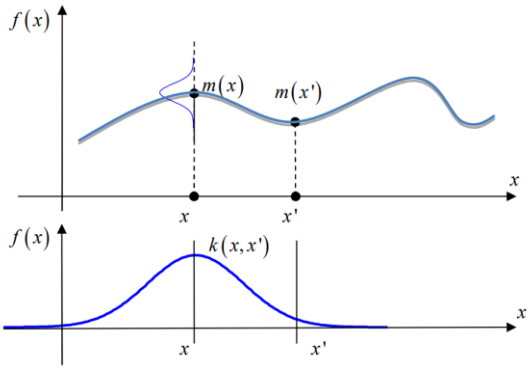

Figure 1: *The mean of a distributed sample f (x) is a stochastic process, where the value at every point x depends on those at the surrounding points x′*

Figure 1 describes more about GP that satisfies Equation (2.7) as follows: At any point *x*, the mean of $f(x)$ is $m(x)$, that is, the true value of the function follows a certain distribution whose mean is $f(x)$. The deviation from this mean depends on all other values at *x′* via covariance function constraints $k(x,x')$. In other words, a function f satisfying Equation (2.7) is a function whose value at any points *x* has an average value of $m(x)$, and whose variance depends on an infinite number of constraints related to the distance to other points *x′* according to (2.8).

3. GP random sampling

Let us consider a sequence of discrete values $X = \{x_i\}, i = 1,...,N,$ and a given covariance function $k(x,x')$. Generating a Gaussian random process $\mathcal{GP}(m,k)$ would be the same as generating a set of values $Y = \{y_i\}$ such that

$$\begin{cases} mean(y_i) = m(x_i) \\ \operatorname{cov}(y_i, y_j) = k(x_i, x_j)' \end{cases} \tag{3.1}$$

where $i, j = 1, 2, ..., N$. This means the sampling process for the random process will be converted back to the sampling process for a multivariable Gaussian distribution.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_N \end{bmatrix}, \tag{3.2}$$

such that

$$mean(\mathbf{y}) = \begin{bmatrix} mean(y_1) \\ mean(y_2) \\ ... \\ mean(y_N) \end{bmatrix}, \tag{3.3}$$

and

$$\text{cov}_{ij}(\mathbf{y}) = \left[k(x_i, x_j) \right]. \tag{3.4}$$

Since $\mathbf{y} \sim \mathcal{N}(m, C)$ can be parsed as:

$$\begin{cases} \mathbf{y} = \mathbf{y}^{(m)} + \mathbf{y}^{(C)} \\ \mathbf{y}^{(m)} \sim \mathcal{N}(m, 0), \\ \mathbf{y}^{(C)} \sim \mathcal{N}(0, C) \end{cases} \tag{3.5}$$

we can generate the sample $\mathbf{y}^{(C)} \sim \mathcal{N}(0, C)$ first; then add the average component $\mathbf{y}^{(m)}$, if any, later. Generating $\mathbf{y}^{(C)} \sim \mathcal{N}(0, C)$ is actually the process of calculating the square root matrix $C^{\frac{1}{2}}$ to multiply into a vector with a multivariable normal distribution. Then, $\mathbf{y}^{(C)} = C^{\frac{1}{2}} \mathbf{y}^{(rand)}$, in which, $\mathbf{y}^{(rand)}$ is a vector whose components are normally distributed random values. Calculating this square root matrix requires matrix diagonalization and can be approximated to reduce the computation.

4. Experiments and results

4.1. Deploying experimental program

For 2D sample generation, the sampling process will go through direct sampling as in the analysis of Section 3. In which the components of the covariance matrix follow the control of the function $k(x_i, x_j) = exp\left\{-\frac{(x_i - x_j)^2}{0.5}\right\}$. That is $z_1=1$ and $z_2=2$ for equation (2.8). Terrain elevation values are generated for 80 grid points $x_i, x_j \in \{0.0, 0.2, 0.4, ..., 20.0\}$, with the mean constraint $m(x_i) = 0, \forall x_i$. The program is written in Python language, the

analysis of the covariance matrix is done by analyzing eigenvalues and eigenvectors to diagonalize the matrix. Figure 2 shows the successful results of sampling for a randomization process. Based on these results, we extend the application of the Gaussian random process to generate samples in 3D space in the next section.

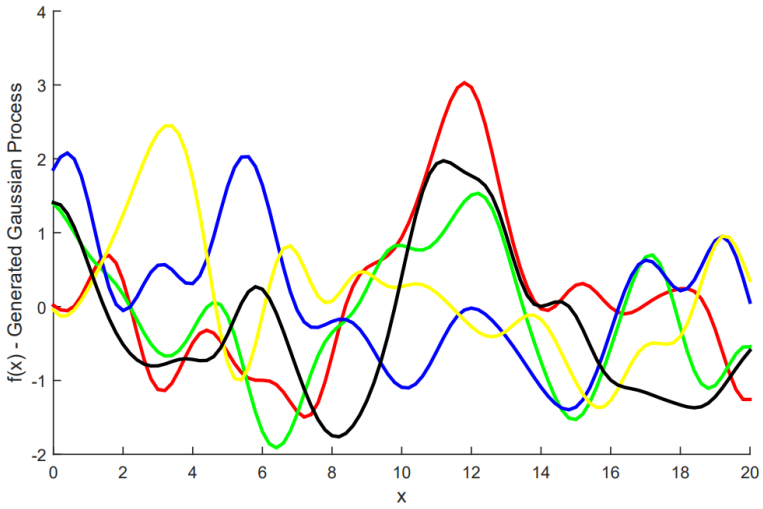

Figure 2: *Result of generating 5 random lines for 80 grid points, the adjacent points are 0.2 apart, according to the covariance function* $k(x_i, x_j) = exp\left\{-\frac{(x_i - x_j)^2}{0.5}\right\}$

4.2. GP sampling in 3D

GP sampling in 3D is not feasible if the dimension of the multivariable vector y is generalized to a multivariable matrix Y. Therefore, to do a GP sampling for a 3D space consisting of *N x N* grid points $\begin{bmatrix} x_i \\ x_j \end{bmatrix}$, we need to treat this generation of random processes as for a vector of size N^2 *x 1*. That is, the components of this vector are ordered from a matrix of grid points. In other words, we need to generate a set of values $Y = \{y_k\}$ such that

$$\begin{cases} k = (i-1)\,x\,N\ +\ j \\ u = \ (i'-1)\,x\,N\ +\ j' \\ mean\ (y_k)\ =\ m(y(x_i, x_j)) \\ cov(y_k, y_u) = k(\begin{bmatrix} x_i \\ x_j \end{bmatrix}, \begin{bmatrix} x_{i'} \\ x_{j'} \end{bmatrix}) \end{cases}, \tag{4.1}$$

where $i, j, i', j' \in \{1,2,\ldots,N\}$ and the covariance function is a function that depends on the distance between the points $\begin{bmatrix} x_i \\ x_j \end{bmatrix}$, $\begin{bmatrix} x_{i'} \\ x_{j'} \end{bmatrix}$. Specifically

$$k\left(\begin{bmatrix} x_i \\ x_j \end{bmatrix}, \begin{bmatrix} x_{i'} \\ x_{j'} \end{bmatrix}\right) = exp\left\{-\frac{1}{\sigma}\left(\begin{bmatrix} x_i \\ x_j \end{bmatrix} - \begin{bmatrix} x_{i'} \\ x_{j'} \end{bmatrix}\right)^T \left(\begin{bmatrix} x_i \\ x_j \end{bmatrix} - \begin{bmatrix} x_{i'} \\ x_{j'} \end{bmatrix}\right)\right\}. \tag{4.2}$$

After generating values for the set $Y = \{y_k\}$, these points are rearranged into grid cells and reconstructed in 3D space.

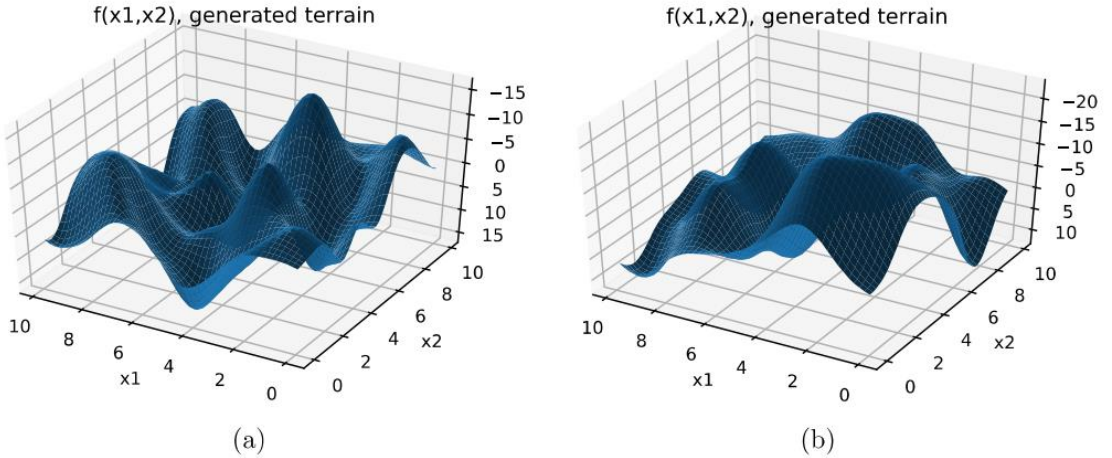

Figure 3: *Generate 3D random terrains with (a)* $\sigma = 1$ *and (b)* $\sigma = 2$

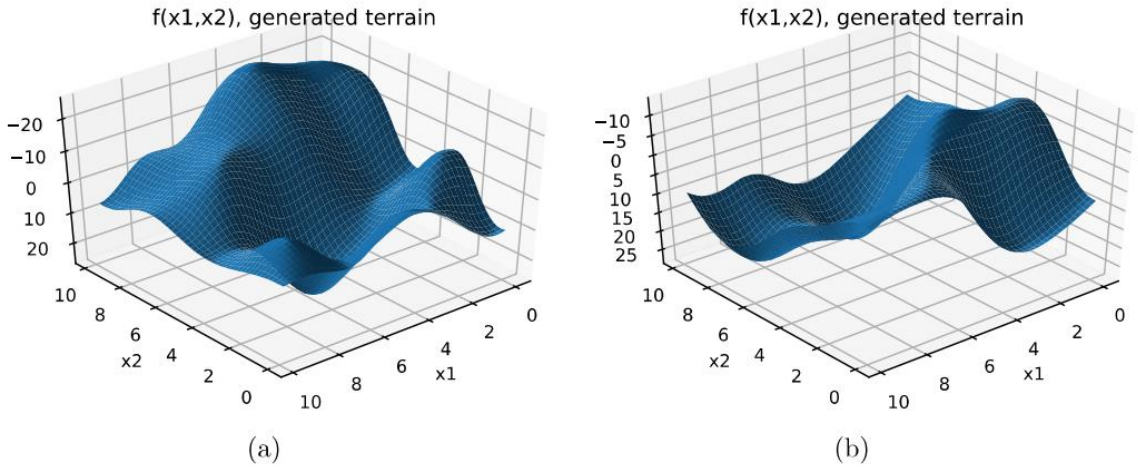

Figure 4: *Generate 3D random terrains with (a)* $\sigma = 3$ *and (b)* $\sigma = 4$

Figures 3 and 4 illustrate the results of topographic generation in 3D space using Python's Matplotlib library for different values. The results show that the terrains generate smooth curves in 3D space. Smaller constraints mean that the influence of one location on the rest is only local, which results in multi-peak and rough terrain. In contrast, the more constraints there are, resulting in a high dependence between the values generated at the points, the fewer hills, peaks, and troughs there are on the terrain.

4.3. Improving calculation speed

Consider the symmetric covariance matrix C and semi-positive definite [9]-[10]. Calculating the square root matrix will require a matrix diagonalization step, i.e.,

$$C = PDP^{-1} \tag{4.3}$$

Where D is a diagonal matrix containing the eigenvalues of the matrix C and P is an invertible matrix. Because of symmetry and semi-positive definiteness, the eigenvalues

in *D* are non-negative values and the vectors in *P* are orthogonal vectors. We can therefore normalize *P* to an orthogonal matrix, or $P = P^T$. Then,

$$C^{\frac{1}{2}} = PD^{\frac{1}{2}}P^{-1} = PD^{\frac{1}{2}}P^T \tag{4.4}$$

This also means that inverse matrix calculation can be reduced and replaced with the tranpose matrix, thereby reducing the computation significantly. Furthermore, we can further reduce the computation by decomposing the matrix multiplication in Equation (4.4). Let λ_i the eigenvalues of the matrix, with descending order $\lambda_2 \geq \lambda_2 \geq \ldots \geq \lambda_N$ and p_i are the corresponding eigenvectors. Then, λ_i are the elements of the diagonal mateix *D*, while p_i are the corresponding columns in the matrix *P*. Hence, Equation (4.4) can be rewritten as

$$C^{\frac{1}{2}} = \sum_{i=1}^{N} \sqrt{\lambda_i}\, p_i p_i^T \tag{4.5}$$

The matrix $C^{\frac{1}{2}}$ used to generate the terrain does not need to be calculated too precisely, but could be accepted with approximations, corresponding to the largest eigenvalues. The remaining eigenvalues are considered to be infinitely small and can be considered as 0. Thus, Equation (4.5) becomes as

$$C^{\frac{1}{2}} \simeq \sum_{i=1}^{K} \sqrt{\lambda_i}\, p_i p_i^T \tag{4.6}$$

In other words, the new terrain generation depends only on *K* operands corresponding to the *K* largest eigenvalues of *C*. The number of multiplications using Equation (4.6) comparing to Equation (4.3) is significantly reduced, i.e. changing the computation for determinant and matrix inversion with complexity *O (N!)* to normalization, and multiply the displacement with complexity of $O(K^2N)$. The matrix inversion using the Gauss - Jordan row transform method is faster than the inverse using the additive matrix and determinant. However, the complexity of the Gauss - Jordan transform is also $O(N^3)$. Figure 5 presents the results of GP generated from the approximation matrix $C^{\frac{1}{2}}$ corresponding to *K=20* eigenvectors with the largest value of the covariance matrix *C*, each color corresponding to two cases of direct calculation and approximation for $C^{\frac{1}{2}}$. The results show that the lines of the same color are very close to each other, the difference is not significant while the smoothness and dependency among points in the curves are still guaranteed.

Similarly, Figure 6 presents the results of GP generated from an approximation matrix of $C^{\frac{1}{2}}$ for 3D space with *N = 2500* grid points, but only for *K=40* eigenvectors with the largest values. The results for the 3D space are applied with 2 values σ= 1 and σ= 2, and the topographical efficiency are not much different from the precise matrix $C^{\frac{1}{2}}$ in Figure 3.

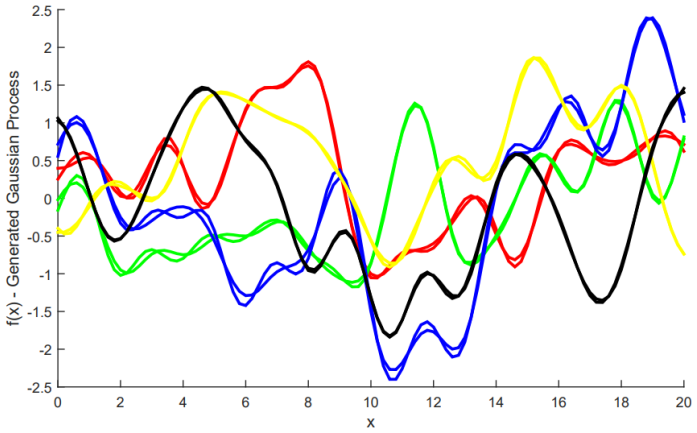

Figure 5: *GP generated from the approximation matrix* $C^{\frac{1}{2}}$ *with K = 20 largest value eigenvectors*

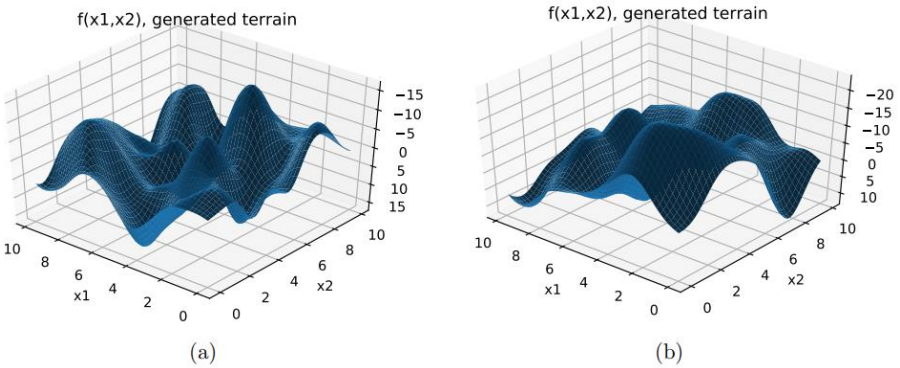

Figure 6: *GP generated from an approximation matrix* $C^{\frac{1}{2}}$ *with K = 40 and (a) σ = 1; (b) σ = 2*

For more details of computational reduction, Table 1 presents a comparison of the number of multiplications to the values *N* and *K* used in the paper. The results show that although the efficiency of terrain generation is not degraded, the number of calculation multiplications when analyzing the covariance matrix has been significantly reduced from 512080 multiplications to 32020 multiplications for the 2D model (Figures 2 and 5). For the 3D terrain generation, the number of multiplications is even reduced from 15625 million operations to 4 million (Figures 3 and 6). As a result, the effectiveness of the proposed matrix approximation is demonstrated without sacrificing the quality of the generated terrains.

Table 1: *Number of multiplications to analyze matrix C into a matrix* $C^{\frac{1}{2}}$

Number of multiplications	$C^{\frac{1}{2}} = PD^{\frac{1}{2}}P^{-1}$	$C^{\frac{1}{2}} \simeq \sum_{i=1}^{K} \sqrt{\lambda_i}\, p_i p_i^T$
K = 20, N = 80	*512080*	*32020*
K = 40, N = 2500	*15625 x 10^6*	*4 x 10^6*

5. Conclusions

The article presented the theoretical basis as well as successfully applied GP to generate smooth curved random terrains in 3D space. In which, the terrain generations are presented for 2D space and the expanded version for 3D space as well as the analysis of computation for the terrain generation. To speed up the calculation, this paper has presented an approximation of the square root matrix of the covariance matrix and, at the same time, checked the generation efficiency with this approximation by comparing the generation results and comparing the number multiplication used before and after the matrix approximation. The results show that the proposed method has significantly reduced the computation but still ensures that the quality of topographic generation is almost unchanged compared to not approximating the square root matrix.

REFERENCES

[1] G. Voulgaris, I. Mademlis, and I. Pitas, "Procedural terrain generation using generative adversarial networks," in *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 686-690.

[2] F. Gurler and E. Onba¸sioglu, "Applying perlin noise on 3d hexagonal tiled maps," in *2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2022, pp. 670-673.

[3] F. L. J. Noghani and E. F. Anderson, "Randomly generated 3d environments for serious games," in *2010 Second International Conference on Games and Virtual Worlds for Serious Applications*, 2010, pp. 3-10.

[4] T. Archer, "Procedurally generating terrain," in *3 rd Int. Conf. Comput. Support. Educ.*, 2011, pp. 1-15.

[5] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*, MIT Press, 2006.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2006.

[7] S. W. Ober, C. E. Rasmussen, and M. V. D. Wilk, "An efficicent approximation for Gaussian process regression," in *37th Conference on Uncertainty in Artificial Intelligence (UAI 2021)*, 2021, pp. 1206-1216.

[8] P. Bodesheim, A. Freytag, A. F. Rodner, E. Rodner, J. Denzler, and J. Denzler, "An efficicent approximation for Gaussian process regression," in Technical Report, Computer Vision Group, Friedrich Schiller University Jena, Germany, 2013, pp. 1-6.

[9] R. Bhatia, *Positive Definite Matrices*, Princeton University Press, United States of America, 2007.

[10] A. Cherian, P. Stanitsas, J. Wang, M. Harandi, V. Morellas, and N. Papanikolopoulos, "Learning log-determinant divergences for positive definite matrices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5088-5102, 2022.

TÓM TẮT

TẠO ĐỊA HÌNH NGẪU NHIÊN TRONG GAME 3D BẰNG MỘT PHIÊN BẢN XẤP XỈ QUI TRÌNH NGẪU NHIÊN GAUSS

Đặng Việt Hùng[1, 2], Võ Nhân Văn[1, 2]

[1] Khoa Công nghệ thông tin, Trường Đại học Duy Tân, Đà Nẵng, Việt Nam

[2] Viện nghiên cứu và phát triển, Trường Đại học Duy Tân, Đà Nẵng, Việt Nam

Ngày nhận bài 03/12/2022, ngày nhận đăng 10/01/2023

Tạo ra một địa hình 3D hay hình ảnh mặt sóng cần rất nhiều thời gian cho một đồ họa viên trong quá trình sáng tạo game 3D. Bài toán này đặt ra cho máy tính các yêu cầu: bề mặt địa hình được tạo ra phải vừa ngẫu nhiên ở mọi điểm, nhưng phải vừa cong trơn, và mức độ chênh lệc giữa các đỉnh phải được kiểm soát bằng thông số đầu vào. Bài báo này sẽ nâng cấp quá trình ngẫu nhiên Gauss (Gaussian Process) 2D thành 3D để giải quyết bài toán tạo địa hình này. Ngoài ra, khi bề mặt cần sinh là rất lớn, quá trình ngẫu nhiên Gauss cần phải thực hiện một lượng lớn phép tính liên quan đến phân tích ma trận. Bài báo này sẽ đề xuất một giải pháp xấp xỉ để tăng tốc tính toán.

Từ khoá: Quá trình Gaussian (GP); Địa hình 3D.